Patch Management Automation for Enterprise Cloud

Hai Huang, Salman Baset, Chunqiang Tang
IBM TJ Watson Research Center

Ashu Gupta, Madhu Sudhan KN, Fazal Feroze, Rajesh Garg, Sumithra Ravichandran
IBM Global Technology Services

Abstract

Applying patches to operating systems, middleware, and applications is considered a major IT pain point due to several reasons. The operating systems and software are of myriad types, there is interdependency among the updates, operating system, and applications, there is lack of standardization among different enterprise customers, and finally testing the applications and operating systems post-update is challenging. As a result, human operator is involved in different stages of the patching process, making it costly and cumbersome. Cloud can help standardize various offerings to customers, and potentially remove human operators. However, it introduces other challenges such as VM time zones and restoring VMs from snapshots which are not present in traditional enterprise environments. We discuss the challenges of achieving patch automation in a Cloud, and then describe our solution.

What is patching?

- Software patches
  - Security: Fix vulnerabilities
  - Functional: Add features, improve existing functions, change software behaviors
  - Impacted systems: Hypervisor, OS, middleware, applications
- Vendor Tools: Windows Server Update Services, Redhat Network, etc.
  - Only handles Windows/Redhat systems
  - Standalone tool with no integration with other management tools, e.g., change management
- 3rd Party Tools: IBM TEM, VMware vSphere update manager
  - TEM: Hypervisor, guests, middlewares, apps. vSphere: hypervisor & guests
  - Scheduling & deployment made easy but largely rely on human involvement, no tool integration
- Amazon EC2, Microsoft Azure, many private Clouds
  - User manages patching. VMs can be potentially vulnerable when first provisioned
- Rackspace
  - Patching is a part of managed service ($0.12 per hour, $100 per month)

Section 1: Introduction

Operating systems, middleware, and applications need to be regularly patched to guard against newly found vulnerabilities or to provide additional functionality. In the non-enterprise space, updates are typically handled by turning on the auto-update feature of the operating system or middleware, which can apply the patches as they are released by vendors. However, these mechanisms for automatically updating are problematic in an enterprise environment. First, IT administrators need to assess the impact of newly released patches before rolling them out to their IT infrastructure. Such pre-assessment is clearly not possible using the automatic update feature. Second, IT administrators need to have a consistent view of their IT infrastructure, including the vulnerability assessment which cannot be achieved by the automatic updates of vendors ISVs (independent software vendors). Lastly, IT administrators need to assess the post patch impact, including failures of existing applications running on their IT infrastructure. All these changes should be recorded in a change management system for audit and recovery from failure purposes (integration w/ other management tools).

Section 1.1: Patch Management Tools

Broadly, there are two different types of patch management tools. There include ISV-vendor specific tools and third party tools. As discussed in Introduction, each ISV vendor has its own mechanism for updating its software. However, by definition, those mechanisms are vendor specific. Examples of these vendors include Windows server update services (WSUS) [WSUS'11] and Red Hat Network (RHN) [RHN'11]. The third party tools such as Tivoli Endpoint Manager (formerly BigFix) [TEM'11] or VMware vCenter Update Manager [VMwareUpdate'11]. Although these tools make the scheduling and deployment of patches relatively easy, but they still rely on humans to schedule the patches. Moreover, these tools are not integrated with other management tools such as asset databases, change management databases, and failure recovery databases. As a result, they require additional human involvement during the end-to-end patch management process.

Among the public Cloud providers, EC2 and Azure VMs leave it to the customer to patch or update the virtual machines. This results in newly provisioned VMs being immediately vulnerable. Rackspace and Microsoft Azure (web and worker roles) provide patching of VMs as part of managed services, but there is a significant cost associated with it. For example, Rackspace offers the managed services at $100 per VM per month. It is unknown whether the patch management process in Rackspace and Azure is fully automated or partially manual.

Patch management in enterprise

- Patch management challenges in traditional enterprise
  - Lack of standardization in software/hardware/services
    - Human intervention in every step (notification, approval, scheduling, deployment, post-deployment)
  - Every customer wants a different patch management policy to suit their needs
    - Patch management solution provided to one customer can be completely different from another
      Little or no solution reuse
  - Labor intensive and cost ineffective

Section 1.2: Enterprise Requirements for Patch Management

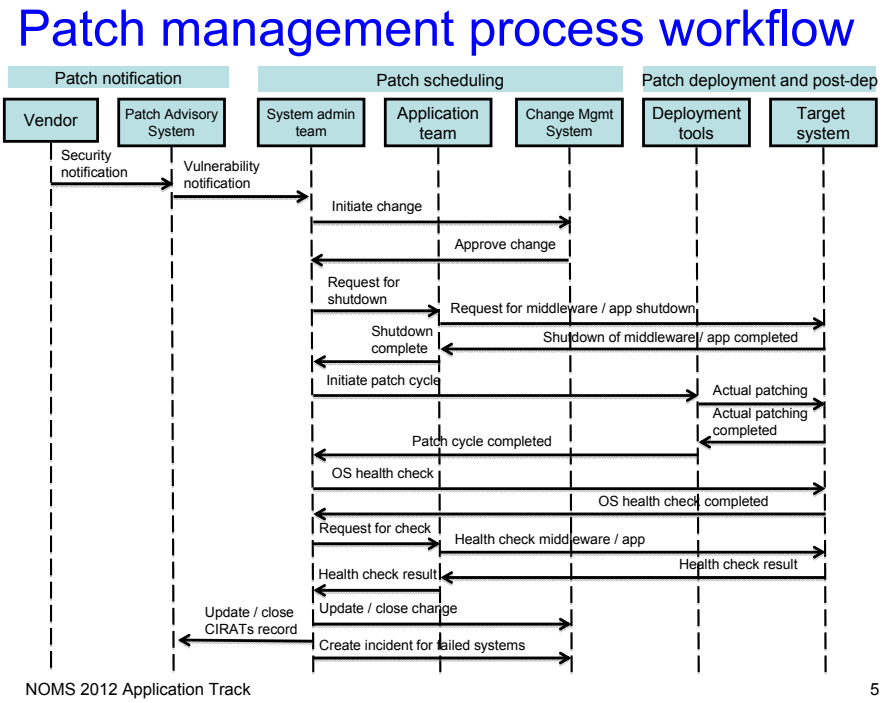
Patch management in enterprise environment is far more complicated than just using a patch deployment tool to apply patches to endpoints as making changes to a working system in any way might cause failures. In an enterprise environment, such failures can impact business in catastrophic ways if not carefully controlled. As a result, there is often a safeguarding process around patching to ensure risk is minimized. We will describe this process in more details in Section 2. Automating patch management process is challenging due to several reasons. First, there is lack of standardization in IT services. This happens because each customer has their own unique hardware and software stacks on which customized services are run. Moreover, each customer often requires a different patch policy to suit their business and infrastructure needs. Some customers may want to patch their machines immediately while other customers may want to fully understand the impact of patches before scheduling and applying them.

Patch management in Cloud

- Cloud opportunities
  - Highly standardized software/hardware stacks
    - Aggressively increase the level of services automation in many processes of IT delivery, including patch management
  - A clean slate to offer customers with more standardized solutions and services at a lower cost
    - Lower provider's operational costs get passed to customers so they are likely willing to accept standardized services such as patch management
- Cloud challenges
  - Extremely large, multi-tenancy, deeper vertical stack due to virtualization

Section 1.3: Cloud Challenges and Opportunities for Patch Management

Cloud presents unique opportunities for patch management. Cloud offers highly standardized and a clean slate solution for customers and can help lower cost for customers substantially. However, Cloud also offers unique challenges because it can be an extremely large, multi-tenant environment, with a deep vertical stack due to virtualization. First, virtual machines can be in different time zones. Since virtual machines run on hypervisors, the scheduling of patches on VMs can interfere with the maintenance of the data center physical infrastructure. Further, unlike physical machines, a customer can take snapshots of virtual machines, and restore the virtual machine from snapshots. Restoring a virtual machine from a snapshot is challenging from patching's perspective, because a compliant VM being restored from an earlier snapshot may immediately become non-compliant.

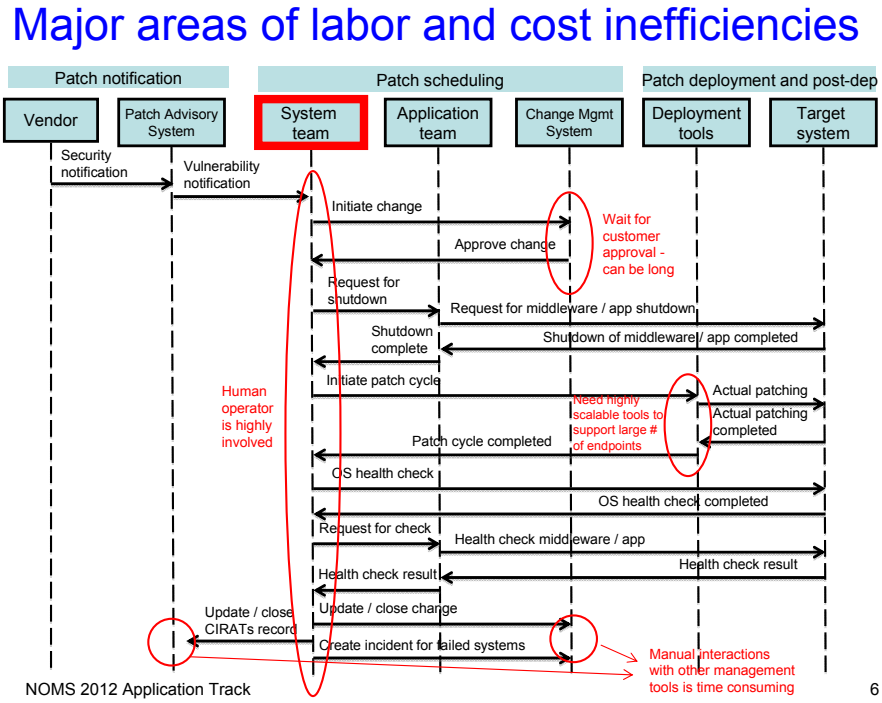


Section 2: Patch Management Process Workflow

To improve and automate patch management process, one must first understand how it works today. The above figure shows an example of a three-phrase patch management process : i) notification, ii) scheduling, and iii) deployment and post-deployment. In the notification phase, when new patches are released by vendors, their availability is then detected by either subscribing to vendors' notification systems or manually checking patch bulletins periodically. For each new patch, patch advisory system is responsible for notifying system administrative teams whose managed assets are affected so that the patch can be scheduled, applied, and tested in the least intrusive manner and timeframe. Patch advisory system is also used to keep track of the lifecycle of a patch, e.g., whether or not a patch has succeeded or failed; on which endpoints was the patch applied; who has done the work; at what time the work was done; and if a patch was not applied on time, was the customer notified, etc.

In the scheduling phase, various parties such as the system administrators, support teams, application owners, customers are notified of the new patch. A change management system is used for various teams to negotiate on what systems will be patched, at what time the patch will be applied, are there any exceptions that should be made, etc. Information exchange and negotiation between multiple teams results in unavoidable wait times. This is especially true nowadays as teams usually situated in different geographic locations with different time zones. One simple back-and-forth between two parties can take up to 24-hours or longer, and when multiple parties are involved, it is not hard to imagine that a simple routine task can take a very long time to reach a decision, dominated by wait times.

In the deployment and post deployment phase, actions agreed upon in the previous phase will be executed by the delivery teams to coordinate the efforts, e.g., when an OS patch is to be applied, application team will first need to shut down the running applications and middleware to avoid data corruption and other potential inconsistency problems. When patch is completed, various health checking tasks and regression tests are performed to verify that the applied changes did not adversely impact anything. As one can see, using patch deployment tools to apply patches onto endpoint systems is only one of many steps in the end-to-end process.



Section 2.1: Major Areas of Labor and Cost Inefficiencies

From analyzing the existing patch management workflow, we identified several major areas of labor and cost inefficiencies. The above figure shows that the system administrator is involved almost every step of the way in this entire process, from receiving patch notifications, to negotiating approval for applying patches to endpoints, to coordinating deployment actions with other support teams and customers, etc. Even the most dedicated system administrators need to take break from time to time, cannot work 24x7, and are often handling multiples concurrent tasks, thus, resulting in the human operator cannot stay completely 100% synchronized with the current progress in the workflow. When multiple of such human operators are collaborating, these little time gaps add up quickly and cause a seemingly simple task to take much longer to complete than what most people would expect. The key in automating this process is to take human operators (e.g., system administrators, support teams, customers) out of the critical path, and rather, only involve the human operators to handle complicated problems or exceptions that are hard to automate. We found improvements in four areas are critical to streamlining the workflow.

Patch management workflow automation

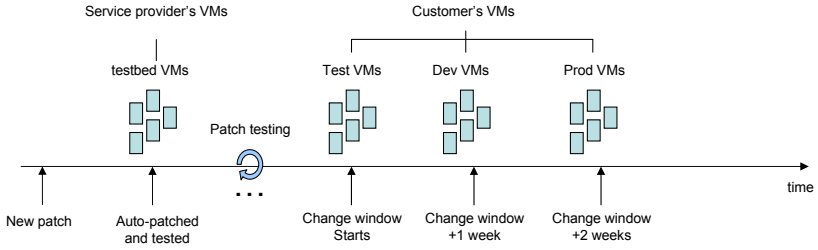
- Solution standardization
  - Customer accepts one of the standardized change windows and patch management policies at contract time
  - No more waiting for customer approval for change requests
- Tools integration
  - Enterprise system management tools have well-defined interfaces and can often be interfaced programmatically
  - Integration is critical to streamlining the end-to-end patch management workflow
- Tool automation, scalability, and multi-platform support
  - Labor and cost associated with patch management increases with number of endpoints, number of customer accounts, number of policies, etc. Need to make Δcost as close to zero as possible
- Eager Patch Testing and Lazy Health Checking
  - To minimize post deployment problems, patches should be thoroughly tested before automated
  - Health checking after post deployment can be delayed until problems actually occur

An automated solution first requires the process to be standardized. A major hurdle that has prevented patch management from being automated in existing managed environments is the lack of services standardization, e.g., a patch management solution tailored to one customer often cannot be used or easily adopted by another customer. Secondly, patch management is not simply using a patch tool to apply patches to endpoint systems, but rather, a collaboration of multiple management tools and teams, e.g., change management and patch advisory tools. Thirdly, in a large enterprise environment, patch tools need to be able to manage a large number of managed entities in a scalable way while being able to handle heterogeneity that is unavoidably in any large environments. Lastly, to avoid problems due to automatically applying patches to endpoints, thorough testing of patches beforehand is absolutely mandatory. However, post deployment health checking can be lazily done as there are a plethora of monitoring tools, e.g., at platform, middleware, and applications layers, that would help detect patch related problems as well as any adhoc health checking procedures.

Standardize patch management services

- Patch severities (examples) – tradeoffs between timeliness and downtime
  - Low/Medium severity patches to be applied every 3 months
  - High/Critical severity patches to be applied every month

- VM categories (examples) – staged patching minimizes bad patches from propagating to production systems



We implement the means to deliver standardized patching solution based on an agreed set of policies with the customer. We leave the actual policy to be defined to the customers and will honor the policy as long as it is within the our pre-defined guideline

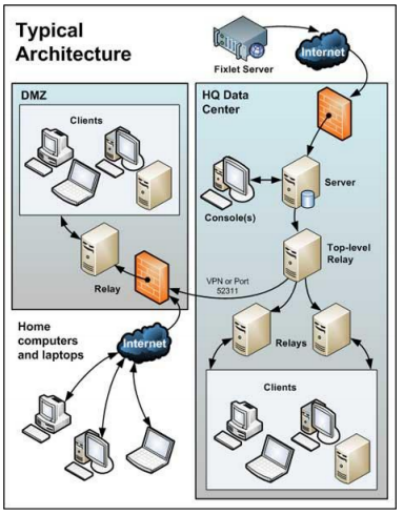
Section 2.2: Solution Standardization for Patch Management

Patch management services are often provided to customers in an one-off fashion as each customer has different business needs, compliance requirements, and uses different sets of management tools. To ensure all the needs and requirements are fulfilled and met, support teams and customers need to negotiate very carefully on which patches to apply, when to apply, dependencies between systems, and whether or not any exceptions should be made, for each new patch to be applied. We devised an alternative solution, which is consisted of a policy agreed upon between the service provider and the customer at contract time. The policy semantic is general enough that it can capture many different business needs, but still flexible enough that customers can request for exceptions as the need arises. This allows an automated solution to perform tasks based on the agreed upon policy, and when the customer's need changes, it will adjust its actions accordingly. In summary, the service provider is to implement the means to deliver standardized patch management solution based on an agreed set of policies with customers, and leaves how a policy is to be defined to the customer to best match their business needs.

Some parameters that customers can use to define patch scheduling policy are patch severity and virtual machine category. Patching causes downtime, and a customer can use patch severity to balance between the amount of downtime due to patching and the benefits of applying a patch. For example, for low and medium severity patches, they can be applied every 3 months; for high severity patches, they can be applied every month. Orthogonal to patch severity, customer can also put virtual machines into different categories, e.g., Test, Development, and Production. A patch is scheduled to VMs of different categories in a staged fashion to minimize the chance that a problematic patch is propagated to production machines.

Tool automation: TEM architecture and APIs

- Tivoli Endpoint Manager (TEM)
  - Hierarchical infrastructure: Server, relay, clients
  - Management through a console
- Core technologies
  - Fixlets – action scripts to test patch applicability and to apply patch
    - The script language is flexible and fixlets can be used beyond patch for other automation tasks
    - Hierarchical broadcast network overlay for content distribution and data collection
- Programmatic APIs
  - Implements Platform APIs and SOAP APIs to allow external entities to control its action programmatically
- Extensions needed to fully benefit Cloud
  - Fully leverage automation enabled through Cloud
  - Integrate with other management tools

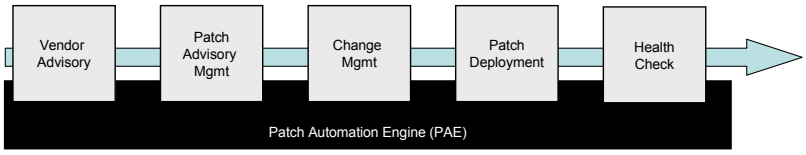


Section 2.3: Tool Automation

The patching tool we are using in our solution is IBM Tivoli Endpoint Manager (TEM) [TEM'11], formerly known as BigFix. It uses a centralized management infrastructure for managing patches, endpoints, subscriptions, and scheduling and performing actions. Patches are distributed to endpoints in a hierarchical broadcast network, where caching servers (relays) are placed at strategic locations to reduce network traffic. These relays are also used to deduplicate data sent back from endpoints to central server to minimize reverse traffic. This infrastructure has been shown to be able to scale to hundreds of thousands of endpoints in a loosely coupled network. One of TEM's core technologies is fixlet, which is a scripting language that describes various actions to apply a patch if it were performed by a human operator, e.g., checking the relevancy of a patch to an endpoint, downloading the patch from an URL, performing checksum of the downloaded patch, applying the patch, etc. Writing a fixlet is an one-time effort, but can be reused by many less experienced system administrators to apply a patch in their own environment in a safer manner.

Out-of-the-box, however, TEM is an interactive tools, where it is expected that a system administrator uses a graphic console to perform various actions. However, if we are to enable end-to-end automation, patch deployment tool needs to be driven in a programmatic way. TEM provides two such APIs: Platform and SOAP. The SOAP APIs is mainly used to read data from TEM server, but to perform any actions (e.g., schedule a fixlet to an endpoint), the Platform APIs are used. The Platform APIs are provided via a client-side DLL library on Windows platforms.

Tool integration



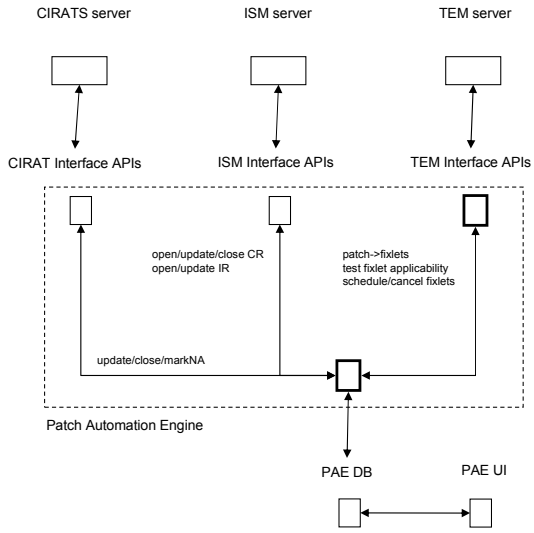
PAE orchestrates patch management flow from end-to-end

- Patch Automation Engine
  - A common substrate that allows orchestration of patch management workflow from start to end across different management tools and facilitates communications across tools

Section 2.4: Tool Integration

Various management tools are used during the lifecycle of a patch, e.g., patch advisory, change management, patch deployment, and health checking. Ideally, there should be a common protocol and communication bus that would allow tools to communicate with each other via well defined APIs. However, in practice, such APIs are rarely implemented by all tools in the tool chain. We built a common substrate layer, called Patch Automation Engine (PAE), that speaks with various tools using whichever interface implemented by individual tools to allow PAE to orchestrate the patch management workflow from start to end across different management tools. For example, for PAE to talk to TEM Server, it uses Platform APIs and SOAP APIs that TEM Server implements, and to communicate with the change management system (in our case, IBM Integrated Service Management (ISM)), it uses SOAP APIs implemented by ISM. The interfaces PAE communicates with the tools in the tool chain are well defined and modular, so in case a tool is replaced by another, only interface functions need to be replaced while the workflow process stays the same.

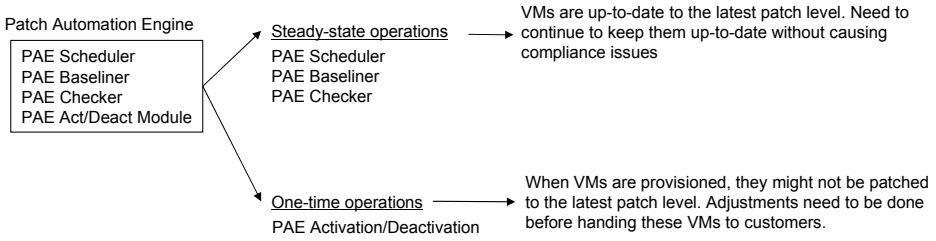
Patch Automation Engine Architecture



Section 3: Design and Implementation

Patch Automation Engine (PAE) is capable of streamlining the end-to-end patch management process largely due to its integration with various management tools, mimicking how a human operator would use these tools to carry out various tasks. Various interface functions/libraries are implemented to integrate PAE with these tools. Not all tools can be interfaced with standard protocols such as SOAP. For example, TEM uses its own proprietary protocol. The interface functions are defined such that when a tool is changed, e.g., TEM being replaced by another tool, only the interface functions need to be replaced. This plug-and-play capability would allow PAE to be re-used in other environments with all changes contained within a limited scope. PAE uses a local database instance to record persistent state information. This is used for auditing purposes, and the intermediate state information can also be used for crash recovery when failure occurs either in PAE itself or one of its dependent services. In the next section, we take a look at each of the PAE components in details.

Patch Automation Engine Components



Section 3.1: PAE components

PAE supports both one-time operations and steady-state operations. Examples of one-time operations are VM provisioning and de-provisioning. When a VM is provisioned, it needs to have the latest patches applied before being handed to the customer. Otherwise, the VM can be vulnerable from the start. The PAE Activation/Deactivation Module is responsible for detecting newly provisioned VMs and applying the latest baseline to them. For steady-state operations to keep VMs always in a compliant state, the Scheduler component detects newly released patches, opens the necessary change tickets, and once the tickets are approved, it schedules the patches onto the relevant VMs during an acceptable change window. However, as things can changed dynamically in a large managed environment, e.g., a customer can change the category of a VM from Test to Production. If there are already change record opened against this VM and patches being scheduled, these actions need to be rolled back and rescheduled accordingly without disturbing existing actions. The Baseliner deals with these dynamic aspects of PAE. When patches are applied during change windows, the Checker component detects whether or not they are applied successfully or not. If so, change records are closed and evidence is collected for audit purposes. However, if a patch has failed, it opens an incident ticket to alert system administrators so that the problem can be resolved within the allowable time frame.

Discussions

- Time synchronization and time zones
  - In a Cloud environment, even VMs located in the same physical location can be configured to use different time zones
  - Patch correctness highly depends on the correctness of time on all management and managed components
    - Time synchronization
    - Different time zones and daylight saving time
- VM suspension and snapshot
  - Suspension and resume
    - Suspended VMs cannot be patched, which may cause a resumed VM to be non-compliant
  - Snapshot and revert back
    - Revert back to an earlier snapshot may also make a VM non-compliant as it may not have the latest patches

Section 4: Discussion

In this section, we discuss some more complex scenarios related to patching that are Cloud specific or need to be more carefully handled in a Cloud environment.

Section 4.1: Multiple Time Zones

In a Cloud environment, virtual machines that are physically located in the same time zone can be configured to operate in different time zones. When a customer's VMs are spanning multiple time zones, scheduling of patches needs to be carefully done so that the correct behavior is implemented. For some patches, the correct behavior is to apply the patches at the same local time of each virtual machine, e.g., applying MS10-081 from Microsoft to all Windows machines at 11pm of their respective local time. For other patches, the correct behavior is to apply at the same absolute time to avoid mixed-mode problem where multiple versions of a software are concurrently running, resulting in data corruption. TEM supports both methods of scheduling patches, and it is up to the customers to specify the intended behavior. The default behavior is to use local time.

4.2 VM suspension and snapshot

In a virtualized environment, there are additional modes of operations available to system administrators and users, such as VM suspension and resume, snapshot and revert back. The management console that allows users of these operations need to be tightly integrated with patch management and compliance processes, otherwise, a VM could become noncompliant unexpectedly. For example, before a VM is suspended, it should have been patched to the latest patch level using the automated patch management process we have described previously. When it is resumed after an extended amount of time, it will most likely be in a noncompliant state with missing patches. Therefore, it is important that the patch management system catches it up to the latest patch level before handing the VM to user's control. Likewise, when a VM is reverted back to an earlier snapshot, baselining the VM to the latest patch level is required.

Related work

- Live Patching
  - OS [Lowell'04] [Potter'05] [Arnold'09]
  - Application [Tewksbury'01]
- Patching in distributed environment
  - Staged upgrade: [Crameri'07]
  - Mixed-mode: [Dumitras'10] [Choi'09] [Segal'89]
- Offline patching
  - VM image [Zhou'10]

Section 5: Related Works

Live patching is a powerful technique that patches running systems without causing services downtimes. This is particularly useful for enterprise applications where downtime translates directly to loss in revenue. Lowell et al. [Lowell'04] and Potter et al. [Potter'05] explored OS virtualization techniques to migrate running applications so that the underlying operating system can be patched without disturbing running workload. Arnold and Kaashoek described Ksplice [Arnold'09], where source code of a patch is converted to hot update, which can then be applied to a running kernel without restarting it. Tewksbury et al. proposes the Eternal system [Tewksbury'01] to enable distributed CORBA applications to continue provide service during upgrades, but it does not provide a general solution. Even though live patching is a powerful technology, it provides no guarantee on the correctness of the patch, thus, its benefit is only limited in practice and is not generally adopted.

Patching complex distributed applications is even more challenging than standalone applications. Crameri et al. proposed the Mirage [Crameri'07] framework to upgrade a distributed application in a stage manner to significantly reduce upgrade overhead. Several other works [Dumitras'10, Choi'09, Segal'99] further explored the problem of mixed mode where multiple versions of an application can operate concurrently during the upgrade process. Patching complex applications is beyond the scope of this paper as we focus on typical Cloud workloads.

Zhou et al. described Nuwa [Zhou'10], a patching tool that applies patches directly to VM images. A benefit to this approach is that it does not require the VM to start from the image, which would take time, before the patch can be applied. However, this also implicitly implies the technique needs to internally handle multiple types of image formats, multiple times of patch package formats, multiple operating systems, etc. In this paper, we focus more on patching running instances.

Conclusion

- Automated Patch Management Solution
  - Fully automated and integrated patch management solution
  - Schedule patches on VMs according to customer-agreed policies, while allowing user initiated exceptions
  - Support multiple categories of VMs (e.g., test, dev, production, etc.): staged patching allows patch schedule to better fit with customer's business needs and minimizing patching errors
  - Support patching of newly provisioned VMs as part of Services Activation & Deactivation
  - Detect suspended and resumed VMs and auto catch them up to the latest patch level
  - Support VM reverting back to an earlier snapshot
  - Support patching VMs located in different time zones and DST
- Future Works
  - Automated post-patch testing using runtime signatures

Section 6: Conclusion and Future Direction

In this paper, we presented our experience in building an automated patch management system for Cloud using TEM. The solution centers around three key areas: standardizing patch management solution, automation of patch tools, and integration of management tools. This essentially removes human operators out of the critical path of patch management process workflow, and only involves them to resolve exceptions and complicated problems that are hard to automate. The initial version of our solution is geared toward standalone Cloud workloads. For more complex application setups, more advanced policies are needed to capture application dependencies to better orchestrate patching of such applications. Additionally, we are working toward an automated solution to perform application/OS health checking by using and comparing runtime signatures of workloads.

[References]

[Lowell'04] D. E. Lowell, Y. Saito, and E. J. Samberg. Devirtualizable virtual machines enabling general single-node, online maintenance. In Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems. Boston, MA, USA, 2004.
[Potter'05] S. Potter and J. Nieh. Reducing downtime due to system maintenance and upgrades. In Proceedings of the 19th Conference on Large Installation System Administration Conference. San Diego, CA, 2005.
[Zhou'10] W. Zhou, P. Ning, X. Zhang, G. Ammons, R. Wang, V. Bala. Always up-to-date: scalable offline patching of VM images in a compute Cloud. In Proceedings of the 26th Annual Computer Security Applications Conference. Austin, TX, 2010.
[Tewksbury'01] L. Tewksbury, L. Moser, and M. Melliar-Smith. Live upgrades of CORBA applications using object replication. In International Conference on Software Maintenance. Florence, Italy, Nov, 2001.
[Segal'89] M. E. Segal and O. Frieder. Dynamically updating distributed software: supporting change in uncertain and mistrustful environments. In IEEE Conference on Software Maintenance, Oct, 1989.
[Choi'09] Online application upgrade using edition-based redefinition. In ACM Workshop on Hot Topics in Software Upgrades, Oct, 2009.
[Crameri'07] O. Crameri, N. Knezvic, D. Kostic, R. Bianchini, and W. Zwaenepoel. Staged deployment in Mirage: an integrated software upgrade testing and distribution system. In Symposium on Operating Systems Principles, Oct, 2007.
[Dumitras'10] T. Dumitras, P. Narasimhan, E. Tilevich. To Upgrade or Not to Upgrade: Impact of Online Upgrades across Multiple Administrative Domains. Onward, Oct, 2010.
[Arnold'09] J. Arnold, M. F. Kaashoek. Ksplice: Automatic Rebootless Kernel Updates Eurosys, 2009.
[WSUS'10] Windows server update services (URL) accessed August 2011. http://technet.microsoft.com/en-us/windowsserver/bb332157
[RHN'10] Red Hat Network (RHN), accessed August, 2011. https://www.redhat.com/red_hat_network/
[TEM'11] IBM Tivoli Endpoint Manager (URL), accessed August, 2011. https://www-01.ibm.com/software/tivoli/solutions/endpoint/?s_pkg=bfwm
[VMwareUpdate'11] VMware Update Manager (URL), accessed August 2011. http://www.vmware.com/products/update-manager/overview.html