CHEF®
Progress®

Effective OS Patch Management

Best practices for patching Linux, Windows, and MacOS systems.

10/2020

Written by: James Massardo

Table of Contents:

Objectives

What are our objectives in patching, e.g. why do we do this seemingly futile task?
What business value do we provide? How does this task help the business?

When we look at IT, there's really only a couple things it can do for a business; things that allow the business to move faster and things that reduce risk to the business. In the case of patching, we're really trying to reduce the risk to the business by minimizing the possible attack surfaces. We often hear that the 'safest' way to manage systems is to keep them current. But it is very time consuming to keep every system updated and patched. In fact, 61% of respondents in a survey sponsored by ServiceNow said that manual processes put them at a disadvantage when patching systems and 57% responded that they were breached to an unpatched vulnerability.

> "**61%** say that **manual processes** put them at a disadvantage when patching vulnerabilities."
> "**57%** of breach victims said they were breached due to an **unpatched known vulnerability**."

Ok, cool, so our goal is to reduce risk; however, making low level code changes on basically everything in a relatively short time sounds pretty risky too, right? In the ideal DevOps world, we wouldn't patch. When new updates are available, they are shipped through the pipeline like every other change. Ephemeral nodes are provisioned, integration tests run, new OS images prepped, new infrastructure provisioned, etc.

I can read your mind at this point; "We aren't all start-ups, some of us still have a lot of legacy systems to manage." And you'd be totally correct. Almost all enterprises (and most smaller businesses) have legacy systems that are critical to the business and will be around for the foreseeable future. Having said that, this doesn't mean that you can't have a well-understood, mostly automated patching process. So how do we reduce this risk without a ton of effort and expense? We automate.

Process and Tools

As you'll see below, the names of the tools within a group may be different; however, they all function using similar patterns. These similarities are what allow us to build a standardized process regardless of platform.
To start, we need a couple of core components.

- **OS Distribution Repositories** - Provides the content and content control
  (Configuration Management Tools - Controls the configuration of the client (I know... Mr. Obvious...).
  This is how we assign maintenance windows, reboot behaviors, etc.
- **Orchestration** - Tools that handle procedural tasks

We want this to be as automated as possible so everything should be set to run on a scheduled basis. The only time it requires manual intervention is when there's a patch we need to exclude. In the examples below, we'll configure the repositories to sync nightly and we'll configure the clients to check on Sunday mornings at 2AM. I can hear what you are thinking again, you're saying "We have a bunch of systems, we can't reboot all of them at once!" And you'd be right. Even if you don't have a large fleet, you still don't want to patch all the things at once.

In reality, you want at least one test system for each application, service, or role in your environment. Preferably, you want complete replicas of production (although smaller scale) for testing. Patches should be shipped and tested via the pipeline just like any other code change. Most enterprises have some process similar to Dev -> QA -> Stage -> Prod for shipping code changes so patching should follow that same pattern. Remember, the earlier we discover a problem, the easier and cheaper it is to resolve.

OS Distribution Repositories

This paper covers the three major OS platforms we see today in large organizations - Linus, Windows and MacOS. The main thing to remember is the repository tool is the source for our patches. It's also the main control point for what's available in our environment. By default, we want to automatically synchronize with the vendor periodically. We then block any patch or update that we can't use (or busted patches that break stuff).

Essentially, we want to create one or more local repositories for all of the available packages for each platform/distribution we want to support. The total number required will vary depending on your network topology, number of supported platforms/distributions, etc.

Patching can be extremely network intensive. If you have a large number of systems or multiple systems across one or more WAN links, plan accordingly and don't DDoS yourself. I can't emphasize this enough, if you don't take load (network, OS, hypervisor, etc.) into account, you will cause severe, possibly even catastrophic outages for your business.

> "If you have a large number of systems or multiple systems across one or more WAN links, plan accordingly and don't DDoS yourself. If you don't take load (network, OS, hypervisor, etc.) into account, you will cause severe, possibly even catastrophic outages for your business."

- Nowet's look at some OS specific info.

Linux

Each Linux distribution has it's own repository system. For RHEL based systems, we use Yum, and for Debian based systems, we use Apt.

RHEL has some licensing implications for running local repositories. Talk to your Red Hat rep for more info.

Customers looking for Patching multiple Linux Distros can use solutions like Kernel Care who offer the ability to to patch multiple linux distributions, they also have a Chef-Infra cookbook that deploys the Kernel Care Agent, you can learn more about it here

MacOS

Until recently, macOS Server had a Software Update Services component. Some folks are using the new caching service but it's not quite the same.

Windows

Windows Server has an Update Services role. WSUS can provide local source for the majority of Microsoft's main products. WSUS also has a replica mode for supporting multiple physical locations or very large environments.

Windows users that are running newer versions of Server and Client can also take advantage of the Branch Cache features. This allows clients on a single subnet to share content and drastically reduce the WAN utilization without needing downstream servers.

Configuration Management Tools

For the purpose of this paper I'm going to use Chef for my examples , but these principles will work with any configuration management (CM) platform. The main thing to keep in mind is the CM tool doesn't do the actual patching. This is a pretty common misconception. Yes, it's entirely possible to use a CM tool to deliver the actual patch payload and oversee the execution, but why would you want to do all that extra work? This is about making our lives better so let's use existing tools and functionality and use CM to control the operation.

Orchestration

Orchestration means different things to different people. To me, orchestration is the process of managing activities in a controlled behavior. This is different from CM in that CM is about making a system or object act a certain way whereas orchestration is about doing multiple things in a certain order or taking action based on one or more inputs.

You will need an orchestration tool if you have any of the following needs (or similar needs):

- **I need to reboot this system first, then that system for my app to work correctly.**
- **I want no more than 25% of my web servers to reboot at any given time.**
- **I want to make this DB server primary, then patch the secondary DB server.**

If any of these sound familiar, you aren't alone. These are common problems in the enterprise; however, there's no common solution. With the magnitude of various applications, systems, and platforms out there, there's no way to provide prescriptive guidance for this topic. A lot of folks use an outside system as a clearing house for nodes to log their state then check the state with a cookbook on all the nodes to make decisions.

In regard to patching, if I have a group of nodes that has a specific boot order, I tend to stagger their patch windows so the first nodes patch and reboot, then the second group, and so on. I may also patch them in one window and leave them pending reboot, then reach out to them separately and reboot them in the required order.

Technical Setup

Below are sample instructions for building the required components. These are not 100% production ready and only serve as examples on where to start. Each company has it's own flavor of doing things so it's not possible to account for all the variations.

Server Side

First thing we need is to set up our repositories. We'll set up a basic mirror for CentOS 7 and Ubuntu 16.04, then set up a Windows WSUS Server.

APT Repository

```
# metadata.rb

# attributes/default.rb

# recipes/default.rb

package 'apt-mirror'

package 'apache2'

template '/etc/apt/mirror.list' do
  source 'mirror.list.erb'
  action :create
end


directory '/var/repo_mirror' do
  owner 'www-data'
  action :create
end


execute 'Sync Mirror' do
  command 'apt-mirror'
  action :run
end
```

Yum Repository

```
# metadata.rb

depends 'yum'
```

```
# attributes/default.rb
default['yum']['repos']['centos-base'] = 'http://mirror.centos.org/centos/7/os/x86_64'
default['yum']['repos']['centos-updates'] = 'http://mirror.centos.org/centos/7/updates/x86_64'
default['yum']['combined'] = false
# recipes/default.rb
package 'createrepo'
package 'python-setuptools'
package 'httpd'

# https://github.com/ryanuber/pakrat
execute 'easy_install pakrat'

repos = ''
node['yum']['repos'].each do |name, baseurl|
  repos += "--name #{name} --baseurl #{baseurl} "
end

repos += '--combined ' if node['yum']['combined']

directory '/var/www/html/' do
  recursive true
end

#
#
# Convert to a cron resource to schedule nightly sync's
#
############################################################
execute 'background pakrat repository sync' do
  cwd '/var/www/html/'
  command "pakrat #{repos} --repoversion $(date +%Y-%m-%d)"
```

```
  live_stream true
end
#############################################################
#
#
#
#


service 'httpd' do
  action [:start, :enable]
end
```

WSUS Server

```
# metadata.rb
depends 'wsus-server'
# attributes/default.rb
default['wsus_server']['setup']['content_dir']                    = "c:/wsus_content"
default['wsus_server']['configuration']['update_languages']             = ['en']
default['wsus_server']['configuration']['properties']['TargetingMode']  = 'Client'
default['wsus_server']['subscription']['automatic_synchronization']     = true
default['wsus_server']['subscription']['synchronization_per_day']       = '1'
default['wsus_server']['subscription']['synchronization_time']          = '22:00:00'
default['wsus_server']['subscription']['synchronize_categories']        = true
default['wsus_server']['subscription']['configure_timeout'] = 3600
default['wsus_server']['freeze']['name'] = "All Approved"
default['wsus_server']['subscription']['categories'] = ['Windows Server 2012 R2', 'Windows
Server 2016']
default['wsus_server']['subscription']['classifications'] = ['Critical Updates', 'Definition
Updates', 'Security Updates', 'Service Packs', 'Update Rollups', 'Updates', 'Upgrades']
# recipes/default.rb
powershell_script 'Configure Shell Memory' do
  action :nothing
```

```
  code 'Set-Item WSMan:\localhost\Shell\MaxMemoryPerShellMB 2048'
end.run_action(:run)

include_recipe 'wsus-server'
include_recipe 'wsus-server::freeze'

powershell_script 'Set WSUS App Pool max mem' do
  guard_interpreter :powershell_script
  code <<-EOH
  import-module webadministration
  Set-WebConfiguration "/system.applicationHost/applicationPools/add[@name='WsusPool']/
recycling/periodicRestart/@privateMemory" -Value 4096000
  EOH
  not_if "import-module webadministration; if ($(Get-WebConfiguration \"/system.
applicationHost/applicationPools/add[@name='WsusPool']/recycling/periodicRestart/@
privateMemory\").value -eq 4096000){return $true}"
end
```

Client Side

Now that we have repositories, let's configure our clients to talk to them.

CentOS Client

```
# metadata.rb
# attributes/default.rb
default['yum']['local_server'] = 'my-yum-repo-server.example.com'
# recipes/default.rb
::Dir.glob('/etc/yum.repos.d/*.repo').each do |repo|
  file repo do
    action :delete
    not_if { ::File.exist?('/etc/yum.repos.d/.chef_managed') }
  end
end

node['yum']['repos'].each do |name, _|
```

```
  yum_repository name do
    baseurl "http://#{node['yum']['local_server']}/#{name}/latest/"
    gpgcheck false
  end
end

cron 'Weekly patching maintenance window' do
  minute '0'
  hour '2'
  weekday '7'
  command 'yum upgrade -y'
  action :create
end
```

Ubuntu Client

```
# metadata.rb
# attributes/default.rb
# recipes/default.rb
```

Windows Client

```
# metadata.rb
depends 'wsus-client'
# attributes/default.rb

default['wsus_client']['wsus_server']                          = 'http://wsus-
server:8530/'
default['wsus_client']['update_group']                         = 'My Server Group'
default['wsus_client']['automatic_update_behavior']            = :detect
default['wsus_client']['schedule_install_day']                 = :sunday
default['wsus_client']['schedule_install_time']                = 2
default['wsus_client']['update']['handle_reboot']              = true
# recipes/default.rb
include_recipe 'wsus-client::configure'
```

Validation

Now the real question: “Did everything get patched?” How do we answer this question? Apt and Yum have no concept of reporting and the WSUS reports can get unwieldy in a hurry. Enter InSpec. InSpec is an open source auditing framework that allows you to test for various compliance and configuration items including patches. Patching baselines exist for both Linux and Windows. InSpec can run remotely and collect data on target nodes.
You can have run InSpec via the Audit Cookbook to have each node report compliance data to Chef Automate for improved visibility and dashboards.

Helpful tips

Here are some tidbits that may help you.

- **WSUS memory limit event log error**. You will almost 100% hit this at some point.
  This is a scaling config so the more clients you have, the more memory WSUS will need.
- **Use attributes to feed required data** (maint. window, patch time, reboot behavior, etc.) to the node.
  This allows you to have one main patching cookbook that get’s applied to everything.
  You can deliver attributes different ways:
  - Environments
  - Roles
  - Role/wrapper cookbooks
- **Remember we are using our CM system to manage the configs, not do the actual patching**.

Closing

Congratulations! If you are reading this, then you made it through a ton of information.
Hopefully you found at least a couple nuggets that will help you. If you have any questions or feedback, please feel free to contact me: @jamesmassardo

Thanks to @ncerny and @trevorghess for their input and code samples.

About the Author: James Massardo

James Massardo is the Lead Customer Architect at Chef focusing on large enterprise. He has more than 20 years supporting and managing large Windows fleets. In his free time, he works with local schools to promote STEM education through competitive robotics programs.

ABOUT CHEF

Chef is the global leader in DevSecOps and the developer of Chef Enterprise Automation Stack™ automating infrastructure, compliance and application delivery for more than half of the Fortune 500. For more than 10 years, Chef has led the industry in DevOps innovation, uniting teams at organizations of all sizes and optimizing processes and outcomes to accelerate its customers' business growth. Chef Software is developed as 100 percent open source under the Apache 2.0 license.

For more visit http://www.chef.io