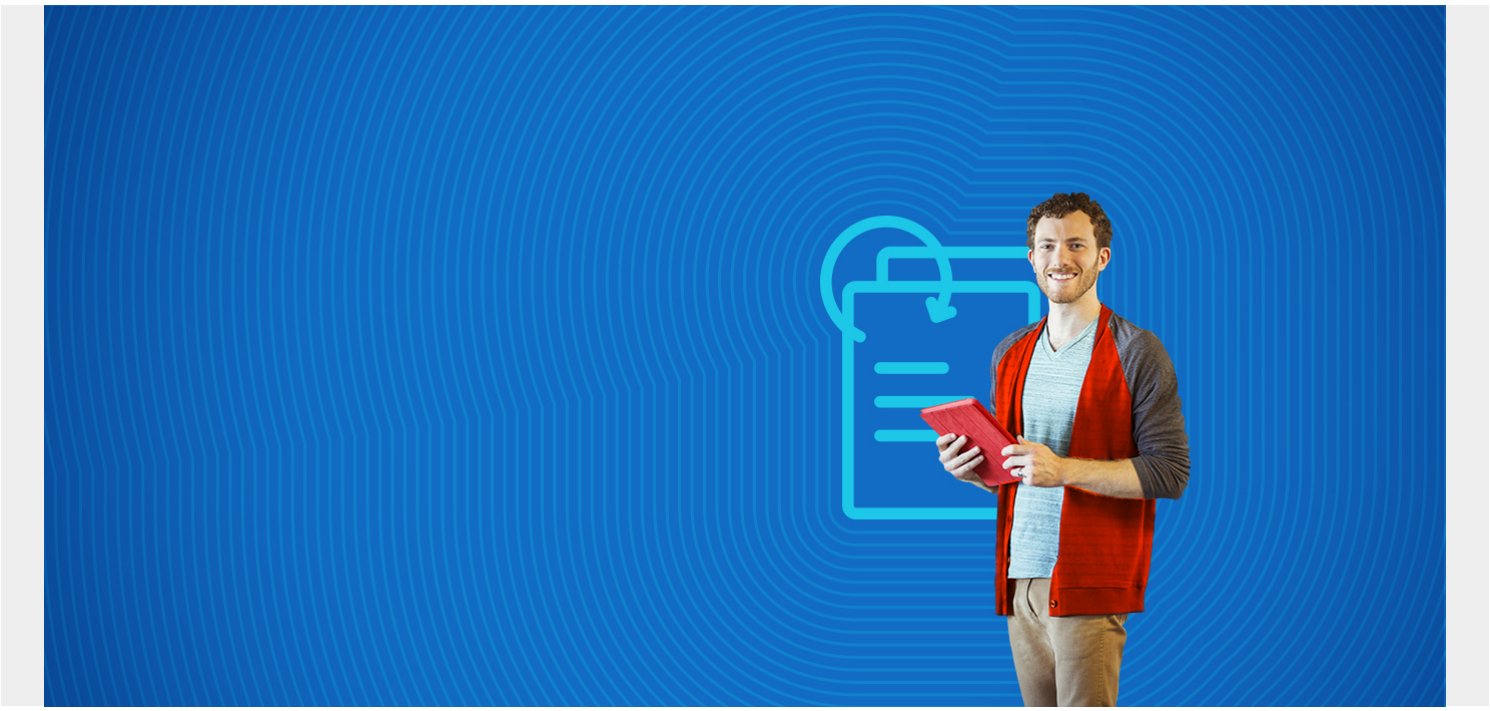
AUTOMATED PATCHING FOR IT SECURITY & COMPLIANCE



Automated patching: The easiest defense for known vulnerabilities

Patching is the easiest way to close known vulnerabilities, most common fixes, provided free or cheaply on every supported OS. The greatest challenges are often gaining approval from app owners and in executing the change approval process. Patching is hardest to start for organizations that don't regularly or consistently patch, as the initial "catch up" runs of patching tend to bring a high rate of change, and have the most impact on applications. What might have been a series of small challenges, encountered a month at a time, when rolled up in 18-24 months of patches, can have significant impacts on applications.

Most organizations start with "blanket" changes, pre-approved for a set of servers, or for an attached spreadsheet list of servers. This causes difficulty in the change calendar, as these changes are usually opaque when looking for change conflicts. As organizations mature and start using more automated and orchestrated changes, it's more common to explicitly link changes to individual target servers, network devices, etc., restoring the function of change calendars and conflict detection.

(This article is part of our Security & Compliance Guide. Use the right-hand menu to navigate.)

Fast track security with TrueSight Vulnerability Management for Third-Party Applications

See how you can accelerate vulnerability resolution, lower costs of remediation and avoid major

security incidents.

- Prioritize and remediate vulnerabilities with TrueSight Vulnerability Management for Third-Party Applications
- Reduce time spent logging changes in CM system
- Improve systems stability through granular, role-based access
- Explore the security view to see predictive SLAs and burndown with TrueSight Vulnerability Management for Third-Party Applications

It's important to note that grouping servers by application owner/customer, production environment, physical location, and maintenance window are highly useful in change planning, and in the execution of patching. While this information should live in every CMDB, it may live only in the automation tools, or in a related system of record. Wherever it does exist, it should be available for change planning, and for use by the automation system. This will let the automation teamwork with logical groupings that the business expects, and make communication straightforward.

It's also useful to note that different application groups will usually want slightly different maintenance windows. When patching is executed or maintained manually, the Operations group will naturally want to drive their customers towards a few common maintenance windows, as this simplifies the operational process. However, it's not uncommon to see many maintenance windows in enterprises with more than a thousand or so server instances. Therefore, it's very useful to automate the process end-to-end, using maintenance window as one of several server properties. These properties can be set and managed by the application or service owners to meet their operational requirements. This will often lead to greater buy-in and fewer "last minute" cancels, as the service/application owner now has more skin in the game. This helps drive up overall coverage of the patching process.

Patching is almost always policy-based, even if that policy is not explicitly defined or systematically implemented. Common policies include applying OS & application patches (especially on Windows), vs "OS-only" patches, or "everything but kernel patches", a common process on Linux. Different OS channels on Linux add a level of refinement in patching Linux. However, it's very common, whichever channels of patches are selected, to identify a level of criticality, and a time window for patching. Most common are the "vendor-critical" patches, as of a certain date, usually the set of Microsoft hotfixes released on "Patch Tuesday", the second Tuesday of the most recent month.

Once the set of patches is identified, they may be vetted by a Security team, or other gatekeepers within the organization, perhaps with particularly high-risk patches, or those that impact sensitive applications (.NET patches are a common example) set aside. Depending on the mechanism for delivering the patches, these may be delivered just as a bundle (as is common in automation environments built around a delivery mechanism, like some Altiris or WSUS environments), or they may be rolled into a policy engine, to be applied where they are needed, as is common with Shavlik and TrueSight Automation for Servers. The greatest risk with a deployment-based approach is that a server that is offline or otherwise doesn't receive patches during the month that those patches are being deployed may not see those patches again. This is usually backstopped by presenting up to 6 months of patches at a time, but still leaves a window open for "older" patches.

For those taking a policy-based approach, now things can get interesting: we can survey, per the policy, all patches missing over time. Given that there are still single digit percentages of Windows 2003 servers in many environments (in 2016), we can expect that we will need to patch many systems in our environments for at least 5 years or more. It's the policy-based approach that lets us

look for any patch meeting the policy, over the range of the lifetime of the OS. You need only meet with your local Security team to understand that there are still MS bulletins 3 years old and older being detected in production environments to appreciate this.

Now that we've surveyed, we can package patches for deployment, stage, and execute these patches.

Staging should not be overlooked, especially for global organizations, if you want to be able to reach to all corners of the globe, especially over intermittent or low-bandwidth connections, it makes sense to "queue up" patches ahead of the actual maintenance window, so that patches can be installed as quickly as possible once the maintenance window opens.

Push vs. pull: there are several methods used to get patches to servers, including both “pull” and “push”. Pull seems like a great idea: less orchestration, less effort involved, but what we find is that it provides less visibility and overall control over the process, and can require a lot more infrastructure and networking: there needs to be a staging host relatively near to every target, and there needs to be a firewall rule to allow each target to access that staging host. Using a “push” approach gives stronger control over scheduling (some customers have strong restrictions over particular windows where servers can have -any- infrastructure activity running on them), and greater visibility into the overall success and readiness of the staging process. When patches push from a centralized management service, the ports, network access, and security requirements can be significantly lighter.

As patches execute, they return exit codes, some of which indicate errors, some of which just indicate a need to reboot after executing. While there are techniques to detect and do intelligent things with these exit codes, to do this intelligently at scale can be challenging. It is often more productive to reboot those systems that need it after patching, and re-run patch analysis. A clean bill of health (no patches now needed) according to the policy is a much faster way to measure the success of the effort than interpreting deploy logs.

After deployment and re-scanning, we now have a good view into the current patch compliance of our environment, which we can then report in total, by production environment, location, OS version, etc. Good visibility into this compliance, and which parts of the organization are doing better or worse, can help drive behavior, and identify areas for improvement, and of course the overall success of the effort.

We have seen, in organizations that implement these techniques fully, scale as high as 10-12,000 targets being patched with less than 4 person-hours of effort per month. This is a fantastic improvement over legacy techniques that may have required as much as 50 to 200 person-days to achieve the same goal, which perhaps explains why some organizations aren't able to patch as often, or consistently as they would like.