Extracted from:

# Practical Security

Simple Practices for Defending Your Systems

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Pragmatic exPress

# Practical Security

Simple Practices for Defending Your Systems

Roman Zabicki

*edited by Adaobi Obi Tulton*

# Practical Security

Simple Practices for Defending Your Systems

Roman Zabicki

The Pragmatic Bookshelf

Raleigh, North Carolina

Pragmatic Bookshelf

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Susan Conant
Development Editor: Adaobi Obi Tulton
Copy Editor: Molly McBeath
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-634-1
Book version: P1.0—February 2019

*To Marnie*

*Thanks for all the geek time*

## Patching Windows

Since Windows is so prevalent, let's look at Microsoft's patching solution. Microsoft has a service called the Windows Software Update Services,[23] or WSUS for short, that helps administrators manage the patching process for all the computers in a domain. With WSUS, you can push updates for Microsoft software to all the workstations in your domain. The details of WSUS are beyond the scope of this chapter, but here are the main things you'll want to have:

- You should have automatic deployment of patches enforced at the Windows domain level.
- You'll want some level of testing of patches. Ideally this would take the form of an automated test environment, where Windows computers go through the motions of simulating commonly performed actions. More realistically, this would take the form of delaying most patches a week or two in the hopes that this will give Microsoft more time to shake out any problems in the patching. Then, patches would be deployed in waves so that even if a patch breaks something, it will only impact a portion of your fleet, instead of every Windows computer in your organization.

## Finding Published Vulnerabilities

So now we have a list of the third-party libraries, networked services, and operating systems in use on our network. Wherever possible, we also have version numbers. This list might not be complete, and might never be complete, but it's still useful. Now we need to see what vulnerabilities have been published for this software.

Searching for vulnerabilities is a manual effort. There isn't a lot of consistency in how vulnerabilities are reported, and there isn't a single centralized location for all vulnerabilities across every piece of software in the world. This means you'll need to combine searches from multiple sources to get a complete picture of the vulnerabilities you're exposed to.

You'll need to build up a list of URLs to search manually. This list will be highly specific to your organization. It will most likely contain a combination of the home pages for each piece of software you use, mailing list archives, online forums, Tavis Ormandy's Twitter stream, RSS feeds, and CVE searches.

23. https://docs.microsoft.com/en-us/windows-server/administration/windows-server-update-services/get-started/windows-server-update-services-wsus

Be sure to document your vulnerability search process. Include specific URLs as well as how to search. In some cases, like the CVE website,[24] this will involve using the search capabilities of a website. In other cases, this will include visually scanning web pages for announcements of security issues. Rotate responsibility for doing this across your team. It's good to share this work since it can be tedious. It's also good to avoid siloing this knowledge in a single team member's head in case someone leaves or takes vacation. And it's good to see how different people search for vulnerabilities. Different people will know about different third-party tools in use, so a diversity of viewpoints will help cover as much as possible. You can't patch what you don't know is in use.

You will want to perform a search for vulnerabilities in your dependencies on a regular basis. Exactly how often? As often as time allows. If it is time-consuming, try to automate it. In time, this should become a reasonably quick activity. If you're pressed for time, a compromise may be to search for third-party vulnerabilities on Microsoft's monthly "Patch Tuesday." Microsoft has established the practice of releasing patches on the second Tuesday of each month. Since your network probably contains a lot of Windows machines, syncing your vulnerability searches to coincide with Microsoft's vulnerability disclosure can be a reasonable starting cadence.

## Testing Your Patches

So we've found out-of-date software on our network. We know we want to patch it as soon as possible. But how soon is possible?

The answer is going to be a little different for every organization. You'll have to decide how comfortable you are with a given vendor's track record of providing stable patches. The answer will also depend on the criticality of the system to be patched, the severity of the vulnerability, and the availability of workarounds for the vulnerability. These variables are outside of your control. The only thing that you can do to speed up the deployment of a patch is to have a set of tests ready to give you a quick yes or no on the question of whether the patch will break things.

A set of tests that covers every piece of software in your organization will always be a work in progress. There is so much third-party software, and there are so many demands on our time other than patching. But even partial test coverage is valuable. If dedicated tests for each piece of third-party code aren't an option right now, you can still get value from integration tests that

24. https://cve.mitre.org

run against a fully running instance of your program. You may be able to test multiple libraries as well as your own code in a single integration test. Integration tests are coarser grained than unit tests. This is nice because you can cover more at once. But this is also problematic because when a test breaks you'll have more places to look to find the culprit.

If time is really tight, you may be stuck with manual tests. There's nothing wrong with manual tests. Just make sure that the tests are documented so that anyone on your team can perform the test. Just like we saw with searching for patches, there's great value in spreading the knowledge around the team, breaking down knowledge silos, and gaining multiple perspectives into this work. You may find that tests "mature" from manual to automated over time. It's entirely reasonable to start with manual testing and only automate if a software product needs to be updated often.

If your tests let problems into production, make the tests more detailed. If they break too often, make them less detailed. If you have to perform the tests frequently, automate them.

**Joe asks:**

### What Is a Breaking Change?

What makes one version of a piece of software different from the version that came before it? Sometimes it's things like improved performance or brand-new functionality. These are changes that improve the system but don't change the way users interact with it. If a user of the software wants to use the new functionality, they can. If they don't, then they don't need to change anything: the new version is a drop-in replacement for the old version. Other kinds of changes are things like renamed methods and method signature changes. These kinds of changes are breaking changes. Breaking changes require that the users change the way they use the software or it breaks.

Breaking changes are different than bugs. A bug is a problem where software does something the authors and users of the software didn't intend or expect. A breaking change is a problem where the authors of the software intend for the software to behave differently than it used to and for users to change the way they use the software in order to adapt to the change.

## If Patching Hurts, Do It More Often

Martin Fowler has written about the saying, "If it hurts, do it more often" as it pertains to activities like deployments and integration.[25] This idea fits

25. https://martinfowler.com/bliki/FrequencyReducesDifficulty.html

wonderfully into a discussion on patching. Fowler gives three main reasons why doing painful things more often makes them less painful over time:

1. It breaks work into small, manageable pieces.
2. It adds opportunities for feedback.
3. It provides practice and the potential for automation.

Let's see how breaking the patching process up into smaller pieces can help us. There's generally a long period of time between patches that fix truly critical vulnerabilities. If we wait and only apply these critical patches, the upgrade will be a bigger job and there will be a higher chance that we'll have to deal with a breaking change. However, if we break up this work over time and apply a lot of small patches all along the way, we'll be fairly current when that critical patch inevitably rolls along. This makes it less likely that we'll have to deal with a breaking change during the tight time constraints around deploying a critical patch. Also, when a less critical patch comes along, we can be more flexible and give ourselves more time; we don't have to drop everything and deploy right away. We can use this extra time to build out reliable automation and work through any bugs or breaking changes.

Feedback is important because this is how we learn what we need to improve. Unless we patch often, we won't know where to focus our efforts. Maybe we need to focus on testing, maybe we need to focus on minimizing downtime, maybe on eliminating race conditions during deployment. If we don't deploy often, we're just guessing.

Finally, practice and automation are the keys to reliable, uneventful patching. Even if we don't have an automated deployment yet, doing frequent manual deployments should motivate us to spend the time to automate deployments. We would want to spend time on automation of any frequently occurring, repetitive task anyway. Improving our security posture along the way is a bonus. Hopefully we've been using the many lower-impact patches as an opportunity to streamline our deployment process. Most of the time, improving the ability to patch reliably is more important than the patch itself.

## A Practical Application of Fear

When you approach the issue of patching production systems from the security point of view, you'll want to patch as many things as possible as often as possible. Your teammates who are in an operations role such as sysadmins or devops will want to make as few changes as possible to a working system. "If it ain't broke, don't patch it," might be their slogan. Don't let these differing viewpoints turn into conflict. The ops folks have a fear of breaking things. Fear isn't always bad; fear can keep us out of danger. So when you encounter resistance from ops about patching, don't try to out-shout them. First aim to understand and resolve the fear. Their fear might be telling you something. Maybe the fear is telling you that you need an automated test suite because you need to roll out patches frequently. Maybe the fear is telling you that you need to build out a performance testing environment, or that you have unresolved technical debt that makes deployments difficult and time-consuming, or that your vendor's patches aren't reliable and you should move to a new vendor. This fear is a valuable resource. Learn from it.