The Defense RESTs: Automation and APIs for Improving Security

Want to get better at security? Improve your operations and your development practices. The key is to centralize management, automate and test. Testing is especially key, - like Jeremiah says, "Hack Yourself First". Many vulnerabilities can be detected automatically. Let the machines do that work and find the basic XSS, CSRF and SQLi flaws, not to mention buffer overflows; save the manual effort for the more complex versions of the above attacks and for business logic flaws.

The fact is that people are really really bad at performing repeatable tasks. The more times people need to do the same task, the more likely they are to make a mistake. Fortunately for us, computers are really good at doing the same thing over and over again the same way.

Now automation isn't exactly new (okay it's not new at all), but it hasn't been used by the security industry very much to date. As a general rule, it's been mostly used for things like AV updates and for scheduling of patches or vulnerability scans. While these are very important, they essentially fall under the category of cron jobs and lack orchestration or any sort of reactive ability. The big exception here is the oft talked about but rarely deployed NAC. NAC,incidentally, is likely to get a lot more life breathed into it as the concept of BYOD becomes more popular. But that is a topic for another white paper.

While automation can make the lives of infosec professionals better, in many many cases the places where the most benefits are to be gained are actually outside of pure infosec. Most of the security tools you need aren't from security vendors, but instead look to operations and development to leverage the tools they already have. They don't even need to be commercial. You need tools like Chef & Puppet, Jenkins, logstash + elasticsearch & Splunk or even Hadoop, to name but a few.

Automation and Configuration Management

Configuration management is a huge problem for most organizations. It’s hard enough to track what assets you actually leave, let alone what software is installed and what state it is in. Unsurprisingly, one of challenges of configuration management databases is keeping them correctly up to date. More often than not, the only way the database is kept current is via manual data entry. This process doesn’t scale, is error prone, and the data quickly gets outdated. This is common enough that when it happens it’s called “configuration drift”. So why have humans involved at all? Instead, integrate your CMDB with your centralized management system. When you make a change to a system, it’s automatically recorded in the CMDB. It’s not rocket science and it’s not super sexy, but when you hear about a new bug or attack going on, it’s damned sexy to be able to query the database and actually know what your exposure is.

But it's about more then just automatically updating the database. With good configuration management tools you are actually simplifying your systems administration and improving your security. Think about how many different system images your typical company manages today. Dozens to hundreds is pretty common. If you dig in however, realistically, it's actually only 1-5 or so different operating systems and the rest is just different applications installed on top of them. Keeping all those images patched is time consuming and error prone, both from the perspective of patching but also the increased likelihood of someone accidentally using an old image.

Instead, I suggest you leverage tools like the aforementioned Chef and Puppet. Then all you need to do is to maintain a barebones image for each OS you are supporting. You then shift all that maintenance effort to keeping your recipes/manifests up to date. This has the added benefit of guaranteeing that when someone goes to push Apache or Java that they are pushing the very same version everywhere. It also acts as a gateway to prevent the wrong version from being deployed. Additionally, you now get better logging of who pushed what and when they did it. Instant auditor gratification! Essentially you've just built a CMDB instead of having to build a separate one or even "integrate" as was hinted at above.

The clients check in with their management servers regularly and make any changes as necessary. This ensures several things. Changes happen quickly and consistently and logs are generated to show that the change succeeded or failed. You now know that you've gotten effectively full patch coverage every single time. And if you pass those logs on to something like splunk or elasticsearch, you can generate automatic alerts upon failures so they can be addressed.

Chef and Puppet, as an added bonus, don't care why the current system state differs from the expected state on the central server. So if someone makes a change locally, it gets overwritten. This means that you get some basic tripwire-like functionality as well. Or in other words, you've now addressed the issue of configuration drift.

But wait, there's more! Once you are solidly using these tools, you no longer need admins logging directly into boxes to do their jobs. In fact, it's antithetical to the whole centralized management pattern. Fewer people touching individual machines makes auditors happy, and reduces the need to login to emergencies and significant troubleshooting. This means you can actually just remove access for lots of your administrators, as they no longer need it.

More Systems Automation and a first sighting of APIs

Key management is one of the banes of systems administration and security. How do you manage those all those keys, passphrases etc, especially when dealing with SSL certs or filesystem or backup encryption? Until recently there have been only two realistic choices, each of which was worse than the other. The first choice was to just not put passphrases on certain keys and the other option was to put the

passphrases or passwords into some sort of text file that the operating system could read and process. Even if it's restricted to root or a specific application user, who likes leaving unencrypted passwords lying around? No one does, but it's accepted as one of those evils necessary to get things done. Fortunately, in recent months there have been a handful of commercial products released that solve this problem not just in bare-metal but also in both private and public cloud deployments. Even better is that they are starting to support any sort of credentials, not just encryption keys.

On a similar note, system unavailability just isn't fun. Users don't care why they can't get to your application or service. It doesn't matter if it's due to a denial of service attack, that you are suddenly ragingly popular or that systems were crashing for some other reason. They just know that they can't do their job, watch their shows or buy the stuff they want to. Monitoring systems can detect these issues and can use APIs to a management console like Chef (http://wiki.opscode.com/display/chef/Server+API), or Puppet (http://docs.puppetlabs.com/guides/rest_api.html) to launch replacement or additional resources. A particular well-designed system could even bring up new resources in a different datacenter or provider. Alternatively, leverage a cloud provider such as AWS that can handle auto-scaling (http://aws.amazon.com/autoscaling/) and automatically bring up more instances as necessary.

Speaking of APIs and centralized automation, why stop at spinning up instances? If you are using a cloud provider (public or private, it doesn't matter), you can also automatically open the correct IPs/ports on the basis of security groups as the instance is provisioned. Similarly, you can also de-provision those ports when the instance dies or is intentionally terminated. See further directions below for doing similar things with more firewalls/routers.

Application Security APIs and Automation

Automation is awesome for systems management but it also allows you to do so much more. This is particularly trying when you also have access to APIs. Ones that are standards based and especially ones that are REST-based are the best. Why? Because they are super easy to implement, are language independent, and stay out of the developers' way. This means less time implementing and more time being productive. Case in point; look at the previous example of managing your systems with Chef and Puppet. You've now deployed a system that only has the "correct" software installed and you that it is properly configured to your corporate standards, whatever they are. But are they "secure"? Have you missed a patch announcement, or made an error when building the recipe/manifest? Why not automatically scan the entire system using a tool like Qualys upon installing or updating packages. This is as simple as making an API call to the correct endpoint. It would look something like this:

curl -H "X-Requested-With: DM Automation" -u $USER:$PASS "https://qualysapi.qualys.com/msp/scan.php?ip=$IP&save_report=yes"

and could be built into the actual Chef recipe. Alternatively, you can also scan the instance upon launch (or reboot). If this were hosted on Amazon's AWS it could be as simple as:

INSTANCE=`ec2-run-instances $AMI -t $TYPE -k $KEY | grep i- | cut -f 2`; until [$IP]; do sleep 15; IP=`ec2-describe-instances $INSTANCE | grep i- | cut -f 17`; done ; curl -H "X-Requested-With: DM Automation" -u $USER:$PASS "https://qualysapi.qualys.com/msp/asset_ip.php?action=add&host_ips=$IP"; curl -H "X-Requested-With: DM Automation" -u $USER:$PASS https://qualysapi.qualys.com/msp/scan.php?ip=$IP&save_report=yes

This isn't particularly fancy, but it does the trick. As an added bonus, the entire report is returned via standard out as XML, so this can be easily piped into a parser, log management system or another software package for responding to any identified issues.

Similarly, you are not limited to just vulnerability scanning your systems/applications from the infrastructure perspective. You can also leverage tools such as Qualys or Whitehat to do a web application vulnerability scan on every deployment. (Is there an OSS WebAppSec Scanner with an API? Let me know!). I am not going to fill this paper with lots of XML, but to retest existing known issues found by WhiteHat is as simple as calling

PUT https://sentinel.whitehatsec.com/api/vuln/retest/<id>

from Jenkins upon a software push. Much like the previous example, this is not terribly complicated (yay, easy to use APIs) but tremendously helpful as it saves a huge amount of time and effort automatically testing things rather then doing them manually.

Application Security But Without APIs

If your app is written in Java, then you should be automatically running FindBugs with every compilation. Jenkins will make this super easy. There are similar options (both commercial and oss) for other languages that can be automatically called as part of each and every compilation run. There's also a great paper from the folks at Coverity (http://howsoftwareisbuilt.com/2010/05/03/interview-with-andy-chou-coverity-chief-architect/) on this general topic. Write security-oriented unit, functional and integration tests. Do both positive and negative testing. Every minute saved by automation is another minute that can be used to write better code or examine hard to automate business logic problems.

And while we are on the topic, you should be checking out and contributing to Gauntlt (https://github.com/thegauntlet/gauntlt) a new tool from James Wickett that leverages Cucumber (http://pragprog.com/book/hwcuc/the-cucumber-book) for automating security checks across the entire application stack.

Future Directions

There are so many possibilities to further enhance security through automation and APIs that there just wasn't time to fully research or code for this white paper. As noted above, cloud providers have made configuring security rules a standard API addressable option. Security and networking vendors are doing the same. Of particular note are two vendors, F5 and Juniper:

F5 has released iControl (https://devcentral.f5.com/tabid/1082224/Default.aspx?returnurl=%2fwiki%2fiControl.F5Downloads.ashx), which gives you full control of F5's devices whether physical or virtual. There's so much there, it can be a bit overwhelming at first, but it is super powerful.

Juniper has released two distinctly different ways of solving this problem. First is netconf, which is a Java framework (and released publicly as an RFC). Recently, Juniper also released Space (http://www.juniper.net/us/en/products-services/junos-developer/space-sdk/) With Space, much like cloud providers have abstracted the underlying compute and storage hardware, Juniper has abstracted network gear. Now you no longer need to know the specifics of each individual piece of kit, but rather just that it's a router or a switch. This is huge.

Other resources to check out include IF-MAP (http://en.wikipedia.org/wiki/IF-MAP) and Security Automata (http://www.securityautomata.org/wiki/index.php?title=Main_Page)

Conclusion

While the concepts of automation and security have been around for a while, we are in the last year or so really beginning to make in-roads into really being functional in this regard. The combination of APIs and automation will truly allow practitioners the time to focus on the harder issues such as business logic flaws and the really difficult and yet basic issues of information security such as identity management , though with SCIM and XACML there are ways automation and APIs can help.