

SECURITY POLICY COMPLIANCE WITH PUPPET AND ANSIBLE

Sean M. Shore

Best Buy

MSP RHUG Dec 2017

THE PROBLEM

- ▶ Quarterly SOX and annual PCI audits
- ▶ Ever-expanding list of controls and covered servers
- ▶ Enormous legacy environment of artisanally-handcrafted servers



THE PROBLEM: ENFORCEMENT

- ▶ Constant drift – except for a greenfield internal cloud environment, all servers maintained individually and ad hoc
- ▶ No mechanism for enforcement, no way to add new controls
- ▶ /etc/sudoers copied from server to server, no cleanup, no review
- ▶ Impossible to provide auditors with concise list of admin privileges

THE PROBLEM: REPORTING


- ▶ No reasonable way to gather data for auditors
 - ▶ Operations staff log into servers individually and copy files
 - ▶ Invalid/incomplete results
- ▶ No way to ensure proper standards on new builds outside the cloud environment
- ▶ End result: internal auditor findings to remediate



THE PROBLEM: LIMITED PUPPET

- ▶ Existing Puppet 3.x open source environment, limited to greenfield new VMs
- ▶ Ignored large legacy environment including physicals, RHEL 5, HP-UX, etc.
- ▶ Even in greenfield environment, sudoers was not fully maintained via Puppet
 - ▶ Default `/etc/sudoers` enforced by Puppet, but all customizations manually copied and edited

STEP 1: ENFORCEMENT

- ▶ Goal: safely extend existing Puppet into brownfield
 - ▶ Maintain SOX and PCI standards on all servers, regardless of status
 - ▶ Began with build-out of new Puppet 4 capability, followed by environment-wide sudoers and access.conf rollout
 - ▶ Migrate existing Puppet 3.x clients to new environment
- 

OPTION: MONOLITHIC SUDOERS

- ▶ Single environment-wide file
- ▶ Previous experience with monolithic sudoers at other firms indicated that it was unworkable over time
 - ▶ Easy to manage, but quickly grows to 10000+ lines, no way to extract info for auditors without additional scripting

SUDOERS STUBS

- ▶ Stripped-down `/etc/sudoers` with `#include /etc/sudoers.d/`
 - ▶ Stubs for individual netgroups and service accounts, as configured by hiera
 - ▶ Increased auditability – each server has only the sudoers rules that are needed on that box
 - ▶ Centrally located in git, where internal auditors can be given read-only access to view all the stubs
- ▶ `/etc/security/access.conf` managed similarly

LEVERAGING HIERA AND PUPPET 4

- ▶ Custom facter fact to break down hostname into usable components
- ▶ Use `hierarray` to pull in stubs as configured at different layers of hiera, all the way to common stubs
- ▶ Coded to take advantage of Puppet 4 functionality re: loops



puppet


HIERA.YAML

- ▶ From most to least specific:
 - ▶ Per-node
 - ▶ Type of server (e.g., prod financial webserver, dev order mgmt app server)
 - ▶ Data center
 - ▶ OS version
 - ▶ Common
- ▶ Allows us to manage one-to-many as much as possible but allow for exceptions

SCM AND CI

- ▶ Danger of managing sudoers with Puppet: pushing bad code to entire environment
 - ▶ Even administrators will be unable to run sudo if there are syntax issues
- ▶ Solution: Automated syntax linting
 - ▶ Changes to sudoers and hieradata are linted on commit
 - ▶ On success, promoted and automatically r10ked using GitLab CI API
 - ▶ Not smart enough to monitor the wisdom of the sudo rules, but prevents catastrophe


OPERATIONALIZATION

- ▶ Puppet and GitLab CI have allowed us to safely hand over sudo administration to L1 and L2 staff
 - ▶ Lead L2 staff can review code and have rights to merge sudo and puppet_control (hieradata) into production branch
 - ▶ Commit logs contain Service Now ticket information for auditability
- 

ROLLOUT

- ▶ Moved stepwise through the environment, Puppetizing small groups of legacy boxes, and expanding list of managed resources
- ▶ Used Ansible to roll out Puppet
 - ▶ Install agent
 - ▶ Set up conf file
 - ▶ Sign certs
- ▶ NTP, resolv.conf, PAM configurations, rsyslog, etc.
- ▶ Within six months we had covered our entire Linux footprint
 - ▶ All were now meeting audit requirements
 - ▶ Misconfigurations automatically reverted

THE NEXT PROBLEM: REPORTING

- ▶ How do we prove that our boxes are meeting audit requirements?
 - ▶ Legacy method was to have Operations staff log into each server individually, or at best write a one-off script, to gather relevant files and perform checks
 - ▶ Too much effort, variable/incomplete results
- 

REPORTING

- ▶ Puppet is great at enforcement, not so great at reporting
- ▶ Has no built-in notification mechanism
- ▶ Runs every 30 minutes – do not want 48 reports per day per server
- ▶ Possible to hack, but poor fit for role

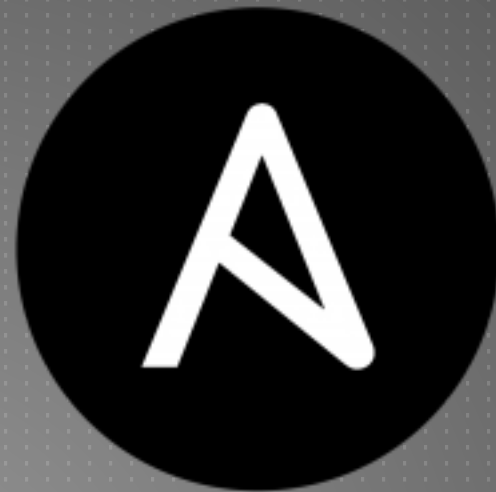


SOLUTION: ANSIBLE

- ▶ Created playbook and role, with tags for SOX and PCI, RHEL and HP-UX
- ▶ Other than copying over certain scripts and zipping up copies of files, Ansible make no changes on the systems
- ▶ Performs regex matching to ensure that configs are as expected
- ▶ Runs scripts to validate settings that would be cumbersome or impossible to run directly via Ansible
- ▶ Configured to not stop on failure, so that all systems and values are checked

SOLUTION: ANSIBLE

- ▶ Goal with each run is to have all green -- no changes needed
- ▶ Copies of all relevant files are zipped up and transferred back to the Ansible workstation
- ▶ All output, including the playbooks and roles themselves, is then uploaded to our site for auditor review



RESULTS

- ▶ Before: monthslong, error-prone effort requiring multiple Operations staff
 - ▶ Usually required remediation with associated CRs, delays, etc.
- ▶ After: with Puppet-based enforcement, no remediation needed
 - ▶ Data gathering can be performed across hundreds of servers by one individual in a couple of hours
- ▶ The big one: Audits passed, findings remediated and closed

QUESTIONS?

