INTRODUCTION

In recent years, the environments that IT professionals are responsible for managing have grown at a rapid pace. In order to keep up, traditional systems administration practices have had to evolve. Tools like Chef and Ansible are built to provide infrastructure automation, ensuring that environments are configured consistently, regardless of how many individual systems need to be managed. As configuration practices become more efficient, testing, security, and compliance can become a new bottleneck. Unwieldy and inconsistent workflows too often result in issues being identified late in the deployment cycle, if at all. To address this problem, Chef launched InSpec, a compliance automation framework that can be used by security and DevOps professionals alike to deliver software at high velocity without sacrificing its quality along the way.

The paper is intended for a technical audience using Ansible today that would like to improve their testing and compliance practices. Tools like Test Kitchen can be integrated with both InSpec and Ansible for validation of Ansible playbooks, and Chef Automate can be used with InSpec to validate the compliance of production environments, whether they are configured using Ansible, Puppet, Chef, or any other kind of configuration management automation.
SHIFT LEFT WITH COMPLIANCE AUTOMATION

In order to ensure that you’re achieving your organizational goals, it’s imperative that you first provide a target that needs to be hit. Only then can you be confident that you’ve actually succeeded in configuring your estate. Recently, the concept of ‘shifting testing left’ has grown in popularity. Organizations often don’t thoroughly test new code until it is being readied for production and discovering issues at that stage can delay releases, lead to unplanned work, and even cause costly rollbacks. The earlier code is tested, the earlier problems can be addressed, saving time and reducing stress when it comes time to promote. But testing is only part of the picture. Shifting security and compliance left can reap much the same benefits, and often in situations where the stakes can be very high. To achieve continuous compliance, your organization must be able to detect misconfigurations and security flaws consistently throughout every stage of the software development lifecycle. This paper will look at three distinct kinds of detection:

- **Code Testing** - Does my code behave the way I expect it to?
- **Security Assessment** - Is my environment vulnerable to known exploits?
- **Compliance Auditing** - Does my environment comply with defined compliance frameworks?

While there is some overlap between these concerns, they are distinct in their purpose and scope. Yet, they share one important quality: All three can be measured by assessing a running environment against a defined expectation. And of course, organizations benefit by assessing all three early in a product development cycle.

InSpec is a tool designed to tackle all three areas, and is unique in the industry as the only testing and compliance tool built from the ground up to be used by stakeholders across your organization, allowing security, operations, QA, and development to have a unified way of collaborating on requirements. Since InSpec only evaluates system state, and does not implement configuration changes, it can be run continuously as a means to automate your evaluation in every environment you manage. The following sections will highlight how you can use InSpec alongside Ansible to ensure continuous compliance across your estate.
CODE TESTING: Validate Ansible playbooks with InSpec and Test Kitchen

Test Kitchen is a testing system included in the Chef Development Kit (ChefDK) to allow users to quickly evaluate configuration management code on ephemeral test infrastructure and then validate the results. It formalizes the testing process into four steps:

Within Kitchen, these steps can be run individually, or all at once via a single command: `kitchen test`.

What’s particularly unique about Test Kitchen is that you can customize each step via a variety of plugins, making it valuable even to organizations that aren’t using Chef to automate their infrastructure. The following example will illustrate this by showing how Test Kitchen can be used to verify an Ansible playbook using InSpec.

TEST WEBSERVER AUTOMATION WITH INSPEC & ANSIBLE

A common automation task is configuring a webserver. As mentioned, a good place to start is to define an expectation that can be used to measure success, for example, confirming that your webserver is listening on the correct port (80 by default), and serving valid content. In InSpec, these requirements can be defined as “resources”, and within those resources, your expected results are defined by “matchers”.

```
describe port(80) do
  it { should be_listening }
end

describe http('http://localhost/', enable_remote_worker: true) do
  its('status') { should cmp 200 }
  its('body') { should match /Hello, World!/ }
end
```
Note that these are functional tests and they don't specify a particular implementation of your webserver. Whatever technology is in use, these resources will determine:

- Is my server listening on port 80?
- Does making a web request to localhost return a valid status?
- Does that web request return the expected content ("Hello, World!")?

This is important because it means we can validate systems regardless of how they were configured. This means that whether your systems are configured with a CM tool like Chef or Ansible, manually configured legacy or brownfield environments, or a combination, InSpec can evaluate the running state of those systems consistently. This allows you to easily move from manual operations to automation while ensuring that the automation is correct.

In Ansible, configurations are defined in YAML files, and an example playbook to configure Apache might look like this:

```yaml
- hosts: myhosts
tasks:
  - name: Update apt cache
    apt: update_cache=true

  - name: Install necessary packages
    apt:
      name: apache2
      state: latest

  - name: Configure Hello World virtual host.
    copy:
      src=helloworldconf dest=/etc/apache2/sites-available/helloworld.conf mode=0640

  - name: Create the helloworld directory
    file:
      path: /var/www/helloworld
      state: directory
      mode: 0755

  - name: Deploy the Hello World website
    copy:
      src=index.html dest=/var/www/helloworld/

  - name: Deactivate the default virtualhost
    command: a2dissite 000-default
```
This playbook will configure your site based on the expectations defined earlier in InSpec.
To evaluate, you can use Test Kitchen to quickly ensure both that the playbook runs without errors, and that the resulting state of the system it configures matches your expectations. Test Kitchen's behavior is defined in a configuration file called .kitchen.yml. The example below collects everything covered so far:

```yaml
---
driver:
  name: vagrant

provisioner:
  hosts: test-kitchen
  name: ansible_playbook
  roles_path: roles
  require_ansible_repo: true
  ansible_verbose: true
  ansible_version: latest
  require_chef_for_busser: false
  playbook: site.yml

verifier:
  name: inspec

platforms:
  - name: ubuntu-16.04

suites:
  - name: default
    verifier:
      inspec_tests:
        - path: tests/site_verify.rb
```
The key sections to note in this configuration file are:

- **driver**: How should this test instance be launched?
- **provisioner**: What tool should be used to configure the instance?
- **verifier**: What tool should be used to validate the instance once it’s been configured?

In the above example, running `kitchen test` will take the following actions:

- **Create**: Launch a local Ubuntu 16.04 VM with Vagrant
- **Converge**: Apply the playbook defined in site.yml to the VM.
- **Verify**: Apply the site_verify.rb inspec tests
- **Destroy**: If all of the above completes without error, destroy the instance when complete.

If any of your configurations fail in the “converge” step, or any of your InSpec resources return a failure, Test Kitchen will halt its execution, and leave the VM running for further inspection. Otherwise, the instance cleans up after itself, and you’re ready to apply your playbook to the instances you manage with confidence that it will behave as expected.

**SECURITY ASSESSMENT:**

**Detect and Correct vulnerabilities with InSpec & Ansible**

Another area where InSpec can have a profound impact on your organization is in security assessment. New software vulnerabilities are always being discovered, and in order to secure your estate, it’s imperative that you be able to quickly assess whether your systems are impacted and remediate accordingly.

**DETECT AND CORRECT THE POODLE VULNERABILITY**

In 2014, CVE-2014-3566 (a.k.a. POODLE) was added to the national vulnerability database and impacts SSL, the primary protocol for encrypted web traffic. In this vulnerability, SSLv3 was shown to contain a design flaw that allows attackers to obtain cleartext content of ostensibly encrypted data. Therefore, the recommendation is to disable this older SSL protocol and use only Transport Level Security (TLS) connections on modern webservers.
InSpec includes an `ssl` resource that can be used to determine whether or not your servers are configured with SSLv3 support:

```ruby
describe ssl(port: 443).protocols('ssl3') do
  it { should_not be_enabled }
end
```

Test Kitchen helped evaluate code updates on temporary infrastructure, but when it comes to vulnerability assessment, it’s important to be able to check your live systems. The ChefDK also includes the inspec command-line utility which can be used to directly scan systems you manage over SSH or WinRM. To assess one of your servers, you can run the following command:

```
inspec exec poodle_test.rb -t ssh://myuser@myhost
```

When you run the above command against one of your servers, you get back a summary that looks like this:

```
Profile: tests from poodle_test.rb (tests from poodle_test.rb)
Version: (not specified)
Target: ssh://ubuntu@myhost:22
    SSL/TLS on
        ∅ myhost:443 with protocol == "ssl3" should not be enabled
    expected SSL/TLS on myhost:443 with protocol == "ssl3" not to be enabled
```

As mentioned, the remediation for POODLE is to allow only the TLS1.2 protocol for SSL traffic in your webserver's configuration. Using Apache as an example again, a simple remediation playbook might look like this:

```yaml
tasks:
  - name: Fix SSL in Apache
    replace: dest=/etc/apache2/mods-available/ssl.conf
        regexp='^SSLProtocol.*$'
        replace='SSLProtocol -all +TLSv1.2'
    notify: restart apache2

handlers:
  - name: restart apache2
    service: name=apache2 state=restarted
```
You can then execute this playbook on one of your hosts similarly to how it was scanned with InSpec:

```
$ ansible-playbook poodle-fix.yml --user="myuser"
PLAY [myhosts] **********************************************************************************************
                             TASK [Gathering Facts] *******************************************************
                             ok: [myhost]
                             TASK [Fix SSL in Apache] ***************************************************************
                             changed: [myhost]
                             RUNNING HANDLER [restart apache2] ****************************************************
                             changed: [myhost]
                             PLAY RECAP ************************************************************************************
                              myhost : ok=3  changed=2  unreachable=0  failed=0
```

Updating Apache’s SSL configuration, and triggering a restart of the service, should be sufficient to ensure your system is not vulnerable to POODLE. The operative word, of course, is “should”. With configuration management you can put the right configuration in place, but without being able to functionally validate them, your job is only half done. What if, for example, someone manually re-edited the Apache configuration file to re-enable SSLv3, something you would not detect until some point in the future when you happen to re-run this Ansible playbook on the machine? Because your compliance requirements have been defined as code, validating your newly updated configuration is as simple as re-applying the same InSpec scan again to ensure that SSLv3 is indeed disabled on your system.

```
$ inspec exec poodle_test.rb -t ssh://myuser@myhost
Profile: tests from poodle_test.rb (tests from poodle_test.rb)
Version: (not specified)
Target:  ssh://ubuntu@myhost:22
    SSL/TLS on
          ✔ myhost:443 with protocol == "ssl3" should not be enabled
Test Summary: 1 successful, 0 failures, 0 skipped
```

Success!
COMPLIANCE AUDITING: Scanning Ansible Environments With Inspec

Scanning for compliance is functionally very similar to evaluating security. The biggest difference is that compliance frameworks formalize this process around a set of known benchmarks. That said, auditing for compliance requires some extra functionality than what’s been covered so far. In particular, evaluating compliance will require:

- **Impact Assessment:** A compliance report needs to be able to filter individual results based on their severity for prioritizing remediation.
- **Role-Specific Granularity:** Compliance Officers, InfoSec, and Operators each have a role in assessing compliance, but require varying levels of detail. High-level summaries and detailed methodologies should both be easy to reference.

InSpec controls are designed with these concerns in mind so that collecting multiple validations into a Compliance Profile can be used to generate weighted reports with multiple levels of granularity in Chef Automate.

```ruby
control 'RHEL-06-000227' do
  impact 1.0
  title 'The SSH daemon must be configured to use only the SSHv2 protocol.'
  desc 'SSH protocol version 1 suffers from design flaws that result in security vulnerabilities and should not be used.'
  tag group: 'SRG-OS-000112'
  tag vulid: 'V-38607'
  tag ruleid: 'SV-50408r1_rule'
  tag severity: 'CAT I'
  tag stigid: 'RHEL-06-000227'
  tag cci: 'CCI-000774'
  tag fixtext: 'Only SSH protocol version 2 connections should be permitted. The default setting in "/etc/ssh/sshd_config" is correct, and can be verified by ensuring that the following line appears: Protocol 2'
  tag checkcontent: 'To check which SSH protocol version is allowed, run the following command: 
  # grep Protocol /etc/ssh/sshd_config
  If configured properly, output should be Protocol 2 If it is not, this is a finding.'
  tag remediation: 'Run the ssh_hardening.yml playbook against the affected server(s)'
  ref 'http://iasecontent.disa.mil/stigs/zip/U_RedHat_6_V1R15_STIG.zip'

  describe sshd_config do
    it 'Protocol' { should eq('2') } end
  end
end
```
The above example is an InSpec control, translating a rule from the Red Hat Enterprise Linux DISA STIG benchmarks, and provides some examples of some of the extra data necessary for thorough compliance evaluation.

The “impact” of a control defines its criticality on a range from 0.0 (minor) to 1.0 (critical). The “title” and “description” provide a human-readable summary of what the control is validating. Each “tag” provides added user-defined metadata, in this case referencing where requirements are defined in the STIG. Finally, “sshd_config” is another example of an InSpec resource, like the “port”, “http”, and “ssl” resources covered earlier.

COMPLIANCE REPORTS IN CHEF AUTOMATE

Chef Automate provides a library of pre-written Compliance Profiles, as well as dashboards to give you a consolidated view of the compliance of your estate as a whole, filterable by environments, server roles, and audit severity. One challenge for Ansible users is that the default method for collecting this data is via a specialized Chef Cookbook, “audit”, which is responsible for running InSpec locally, and reporting the results to Chef Automate. Typically, this cookbook would be pulled from a Chef Server, but for organizations using Ansible, that’s not always a feasible option. Thankfully, Chef Automate now allows audits to be generated by running agentless scans directly from the Automate Server.

To start, you’ll need a Compliance Profile, which is a collection of InSpec controls organized around a specific theme. Chef Automate comes pre-loaded with a variety of profiles based on different compliance frameworks and operating systems. You can view all available profiles by clicking the “Profile Store” link within Automate’s “Compliance” dashboard, and then clicking on the “available” tab to its right.
Once you find a profile you’d like to install, click on it, then click the ‘get’ button at the top of the page:

This will install the latest version of the profile, which can now be used to start scanning your servers. From there, the ‘scanner’ link on the left-hand menu can be used to define the nodes you wish to scan, including hostnames, connection protocol (SSH or WinRM), and access credentials. Once you have at least one profile and node configured, you can create a ‘job’.

Jobs can be configured for one-time at-once execution, or with a recurring schedule, as pictured above. The two fields in the bottom of the job list all available nodes and profiles -- simply select all that you’d like to encapsulate in the job, and click “create job” in the upper right-hand corner.
Once the job has run, the reporting dashboard will now show a high-level summary of how your node(s) performed, and how many, if any, of the controls within your profiles failed.

On each node, you can also view a detailed list of each control that was run, with any failed nodes filterable based on their severity:
From this same view, you can click on any individual control for more details on its status, and even the raw InSpec source code:

Whether you’re managing one server, or thousands, Chef Automate provides a single pane of glass into compliance performance across your estate. Within Automate’s dashboards you have the ability to view results with whatever level of granularity you require in your role -- all without needing to install any additional agents on your servers!

**SUMMARY**

To deliver software at high velocity, it’s critical to have a means to detect misconfigurations and security flaws consistently and continuously across all systems you manage. InSpec and Ansible are both designed with automation and repeatability in mind, and together help ensure that you can deliver software quickly, efficiently, and above all, securely. With Chef Automate you can take this process further with a library of pre-written profiles to jump-start your compliance, and a single window into health of every environment you manage.
RESOURCES

LEARNING

Compliance Automation with InSpec
https://learn.chef.io/tracks/compliance-automation/

Scan for Compliance with Chef Automate
https://learn.chef.io/modules/try-chef-automate/scan-for-compliance#/demos-and-quickstarts

Certification Exam: Auditing with InSpec

DOCUMENTATION

InSpec Docs: https://www.inspec.io/docs/

Chef Automate Docs: https://docs.chef.io/chef_automate.html

BLOGS & WEBINARS

Augment your Audits with InSpec 2.0

Every Day Compliance with InSpec

Compliance with InSpec: Any Node. Any Time. Anywhere.

Presentation Deck: Effective Testing with Ansible and InSpec
https://www.slideshare.net/nathenharvey/effective-testing-with-ansible-and-inspec