CHEF®

Progress®

Chef PCI DSS Compliance Guide

10/2021

Executive Summary

If a company handles credit card data in any way, they are therefore subject to the Payment Card Industry Data Security Standard (PCI DSS). It is well-known that PCI audits can be difficult and time-consuming to execute. The quicker and easier it is to pass a PCI audit the better it is for an organization. Chef believes the quickest and easiest way for an organization to pass a PCD audit is by implementing a PCI Continuous Assessment approach. This guide is written for members of both technical and compliance teams working with systems in any CDE. Traditionally, these teams run on-demand assessments during a PCI DSS audit. This guide illustrates how to, at a minimum, use Chef Compliance to validate system configurations during an audit in order to map existing manual functions to automated controls.

Table of Contents

Introduction *2*

PCI DSS and CIS Controls *3*

About Chef Compliance *3*

Chef Continuous Compliance Cycle **4**

PCI Requirements *5*

Requirement 1: Install and Maintain a Firewall Configuration to Protect Cardholder Data **6**

Requirement 2: Do Not Use Vendor-Supplied Defaults for System Passwords and Other SecurityParameters **7**

Requirement 3: Protect Stored Cardholder Data **13**

Requirement 4: Encrypt transmission of cardholder data across open, public networks **14**

Requirement 5: Protect all Systems Against Malware and Regularly Update AV Software **14**

Requirement 6: Develop and Maintain Secure Systems and Applications **16**

Requirement 7: Restrict access to cardholder data by business need to know **18**

Requirement 8: Identify and Authenticate Access to System Components **20**

Requirement 10: Track and Monitor all Access to Network Resources and Cardholder Data **23**

Requirement 11: Regularly Test Security Systems and Processes **25**

Requirements 9 & 12 ..... **26**

Conclusion *26*

Legal Disclaimer *26*

ABOUT PROGRESS *27*

Introduction

Maintaining PCI DSS compliance is a struggle for many organizations, particularly those that have experienced a data breach.The 2020 Verizon Payment Security Report found that **only 27% of organizations were able to maintain full compliance** with the PCI-DSS, an 8.8% drop from the year before. Further, the report found that companies in the retail, hospitality, and finance industries struggled the most with PCI DSS requirements.

A typical approach to passing a PCI DSS audit is to issue ad-hoc remote commands to gather information, compose verification scripts to run by hand, or to manually verify a number of system settings in tandem with auditors by using approaches like screenshots. This approach is fundamentally unsustainable since it requires custom work, is specific to the context of your PCI DSS audit, and cannot be leveraged in other parts of business-critical workflows, such as checking for compliance in pre-production environments.

Adopting a continuous compliance approach allows you to quickly answer audit questions at any time, not just quarterly or yearly. With Chef Compliance, you can enter an audit cycle knowing your exact compliance posture, rather than being surprised by auditors who find weak points in your environment. You can identify compliance issues or policy breaches rapidly and react quickly to triage and remediate problems even before auditors show up, allowing you to demonstrate how compliance has evolved and improved over time.

Chef InSpec is a security hardening, testing and auditing tool that turns your compliance, security, and other policy requirements into automated tests. These tests are designed to continuously gather data about your systems using a standardized language and data format thatclearly map to the compliance controls your auditors care about. Chef Compliance has several packaged compliance rulesets that cover industry-standard use cases. However, if rules for your specific use case do not exist, Chef Compliance allows youto easily compose them. Chef Compliance is designed to leverage the same automated testing used in other parts of your organization to make discussions with your compliance auditors easier and faster to resolve.

This guide covers the 12 PCI DSS security requirements that apply to all system components included in or connected to the CDE. For each requirement, this guide illustrates how to use existing Chef Compliance controls -- or create new controls -- toverify compliance.

PCI DSS and CIS Controls

The Payment Card Industry Data Security Standard (PCI DSS) provides baseline technical and operational requirements designed to protect account data. PCI DSS contains 12 major requirements with corresponding testing procedures used for compliance assessments as part of an entity's validation process. While PCI DSS comprises a minimum set of requirements forprotecting account data, it may be enhanced by additional controls and practices to further mitigate risks.

The Center for Internet Security (CIS) publishes a set of controls that collectively form a defense-in-depth set of best practices that mitigate the most common attacks against systems and networks. The CIS Benchmarks are developed by a community of IT experts who apply their first-hand experience as cyber defenders to create a set of globally accepted security best practices. As a result, the CIS Benchmarks have matured by an international community of individuals and institutions that not only identify, track, and remediate cyber security attacks, but also map those controls to regulatory andcompliance frameworks.

This guide uses mappings of CIS controls to PCI DSS standards to inspect a Red Hat Enterprise Linux 7 system for compliance. A subscription to CIS benchmarks for other operating systems, expressed in Chef InSpec, is included with a ChefCompliance license.

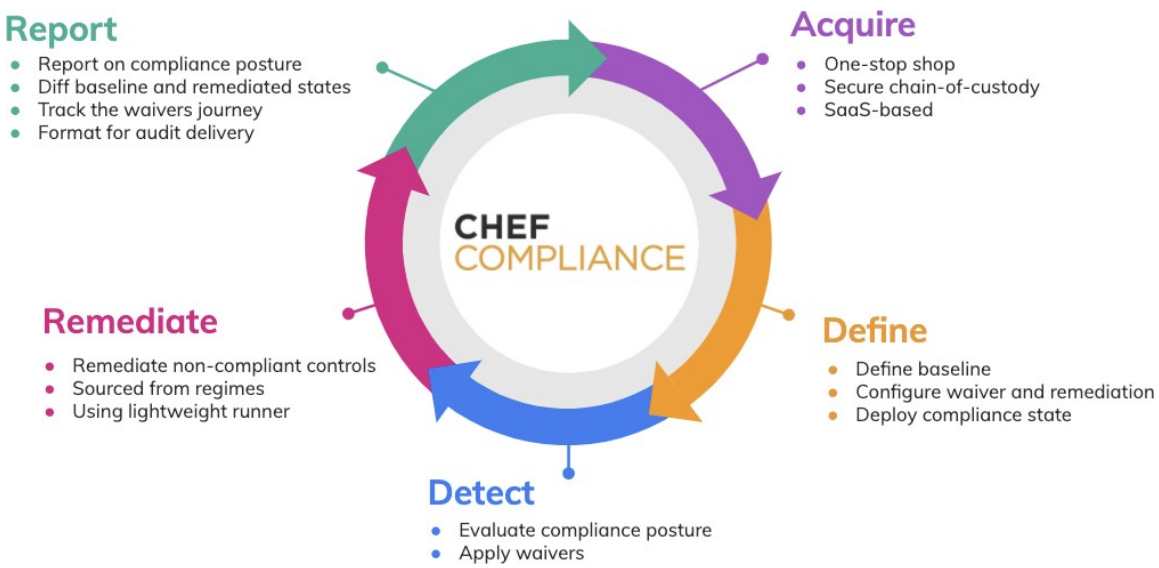
About Chef Compliance

Chef Compliance is built on Chef core technology proven in large, complex environments over the past 10+ years. It is designed to help enterprises maintain compliance and prevent security incidents across heterogeneous hybrid and multi-cloudenvironments while improving speed and efficiency. Standards-based audit and remediation content, easily-tuned baselines and comprehensive visibility and control make it easy to maintain and enforce compliance across your entire fleet.

Chef Compliance helps automate the standards by incorporating compliance processes into every stage of the developmentcycle based on the following Chef underlying core technologies:

- **Chef InSpec** allows developers and systems engineers to replace lengthy and opaque security specification documents – written in PDF or Excel – with unambiguous tests that are easily readable by all parties involved: security engineers, auditors, systems administrators, and others.
- **Chef Automate** provides a standard set of security baselines you can easily customize and extend. Examples of baselines included are CIS Compliance Benchmarks and several DISA STIGs. Chef Automate can convert existing, hard-to-use formats to human-readable InSpec in order to take advantage of NIST baselines published in those formats.
- **Chef Infra** configuration management can be used to remediate any findings and keep systems in their correct,remediated state, thereby ensuring continuous compliance.

Chef Continuous Compliance Cycle



- **Acquire** - access trusted content aligned to industry benchmarks for audit and remediation. With remediationcontent, organizations can ensure remediation actions align directly to audit results.
- **Define** - define compliance baselines and tune them to your organization's unique needs. Flexible compliance waiver capabilities allow teams to turn on/off individual controls to avoid false positives and misconfigurations.
- **Detect** - continuously monitor and evaluate your compliance posture by detecting deviations from intended stateat any point in the software delivery lifecycle.
- **Remediate** - new remediation capabilities allow you to address non-compliant nodes with individual controls that align with audit tests. Remediation is easily applied, without requiring coding skills to ensure CIS and DISA standards.
- **Report** - immutable record in maintaining comprehensive and up-to-date visibility across heterogeneous environments, easily view differences between baseline and remediated states, and track waiver status to enable fast and accurate audits any time.

Chef Compliance's collaboration capabilities allow all parties involved in pre-production development processes and production operations the ability to see, in real-time, the compliance status of any infrastructure. No longer do teams need torely on expensive, infrequently used tools accessible only by security engineers. With Chef Compliance, IT organizations can move from being reactive about compliance issues to always being compliant – and being able to prove it at any time.

Chef Compliance helps organizations reduce risk, increase speed, and improve efficiency. With security capabilities in Chef Compliance, DevOps teams can work closely with their InfoSec counterparts to ensure the software they ship is compliant and secure, turning InfoSec into an enabler rather than an inhibitor of velocity. Chef Compliance enables collaboration amongInfrastructure, Operations and Information Security teams, and is available in two options tailored to the needs of these audiences:

Chef Compliance: helps security and operations teams maintain complete visibility over the compliance posture of their estate. It comes with extensive audit content based on Center for Internet Security (CIS) benchmarks and Defense Information Systems Agency (DISA) standards that can be easily fine-tuned to meet specific organization needs. It providesup-to-date visibility across any on-premises or cloud environment.

Chef Compliance Automation: built for DevOps and Infrastructure teams to help close the loop between audit and remediation and enable continuous compliance in the enterprise. Remediation functionality and trusted, standards-basedcontent make it easy to remediate issues uncovered during audits without writing any code.

Chef Premium Content: delivers Chef-curated content for audits, remediation and desktop configuration that is based on CIS (Center for Internet Security) certified benchmarks or DISA Security Technical Implementation Guides (STIGs). Chef continuously maintains and updates the Premium Content library and, whenever an updated or new profile is identified, Chefquickly certifies the content and makes it available for content subscribers.

PCI Requirements

The requirements of the Payment Card Industry Data Security Standard encompass a variety of concerns with respect to the cardholder data environment). The CDE is comprised of people, processes and technologies that store, process, or transmit cardholder data or sensitive authentication data. The scope of a PCI DSS audit also includes interviews, process reviews, and inspection of physical assets. For the purposes of this document, we will focus on the inspection of “system components” within the CDE, including network devices, servers, hypervisors, virtual machines, and application software.

While some of the requirements focus on processes and policy not applicable on the systems level, the sections below contain an illustrate some of the CIS benchmarks included with Chef Compliance to address 10 of the 12 major audit requirements of the PCI DSS.

These controls are referenceable as a demo profile to use against a Red Hat Enterprise Linux 7 server. The controls reference underlying CIS Benchmarks, which are partially shown in the sections below that walk through how these controls operate in practice. This whitepaper and the associated demo profile are an introductory approach to how an organization could useChef Automate to address PCI DSS requirements. They are not an exhaustive guide for complete implementation.

The PCI DSS requirements covered by this whitepaper are:

1 - Install and maintain a firewall configuration to protect cardholder data

2 - Do not use vendor-supplied defaults for system passwords and other security parameters are not used3 - Protect stored cardholder data

4 - Encrypt transmission of cardholder data across open, public networks5 - Use and regularly update anti-virus software or programs

6 - Develop and maintain secure systems and applications

7 - Restrict access to cardholder data by business need-to-know8 - Assign a unique ID to each person with computer access

10 - Track and monitor all access to network resources and cardholder data11 - Regularly test security systems and processes

PCI DSS requirements 9 and 12 concern physical security and processes, respectively, and cannot be validated using automated approaches.

Requirement 1: Install and Maintain a Firewall Configuration to Protect Cardholder Data

Any devices connecting to the internet should use a firewall to limit access to open ports. Section 1.4 of the PCI DSS (Firewall software or equivalent) can be addressed with the CIS benchmark ruleset (Limitation & Control of Network Ports) to ensure firewall rules for all open ports.

1. Ensure firewall rules for all open ports

Any ports that have been opened on non-loopback addresses need firewall rules to govern traffic. Without a firewall rule configured for open ports, the default firewall policy should drop all packets to these ports.

This control uses Ruby code to create a list of all open system ports listening to non-loopback addresses. The control alsogathers a list of all currently implemented IPTables rules. The control ensures each open port listening on non-loopback addresses has an associated IPTables rule with both an inbound and outbound entry.

```
control "cisecurity.benchmarks_rule_3.6.5_Ensure_firewall_rules_for_
open_ports" do
        title "Ensure firewall rules exist for all open ports"
        desc "Any ports that have been opened on non-loopback addresses need
firewall rules to govern traffic. Rationale: Without a firewall rule
configured for open ports default firewall policy will drop all packets
to these ports."
        impact 1.0
        tag "cis-rhel7-2.1.1": "3.6.5"
        tag "level": "1"
        tag "type": ["Server", "Workstation"]
        port.where { protocol =~ /.*/ && port >= 0 && address =~
/^(?!127\.0\.0\.1|::1|::).*$/ }.entries.each do |entry|
          rule_inbound = "-A INPUT -p #{entry[:protocol]} -m
#{entry[:protocol]} --dport #{entry[:port]} -m state --state
NEW,ESTABLISHED -j"
          rule_outbound = "-A OUTPUT -p #{entry[:protocol]} -m
#{entry[:protocol]} --sport #{entry[:port]} -m state --state ESTABLISHED
-j A"
          describe iptables do
            it { should have_rule(rule_inbound) }
            it { should have_rule(rule_outbound) }
          end
        end
end
```

Requirement 2: Do Not Use Vendor-Supplied Defaults for System Passwords and Other SecurityParameters

Malicious individuals (external and internal to an entity) often use vendor default passwords and other vendor default settingsto compromise systems. These passwords and settings are well known by hacker communities and are easily determined via public information.

Section 2.1 of the PCI DSS (Always change vendor-supplied defaults) can be addressed with the CIS benchmark ruleset (Controlled use of Administrative Privileges). Five examples are shown below.

1. Ensure password creation requirements are configured

Strong passwords protect systems from being hacked through brute force methods. Settings that require strong passwords are often a good way to ensure common vendor-supplied default passwords are not used on a system. The system should enforce use of strong passwords.

```
control "cisecurity.benchmarks_rule_5.3.1_Ensure_password_creation_
requirements" do
  title "Ensure password creation requirements are configured"
desc "The pam_pwquality.so module checks the strength of passwords.
The settings shown above are one possible policy. Alter these values
to conform to your own organization's password policies. Rationale:
Strong passwords protect systems from being hacked through brute force
methods."
  impact 1.0
  tag "cis-rhel7-2.1.1": "5.3.1"
  tag "level": "1"
  tag "type": ["Server", "Workstation"]
  describe file('/etc/pam.d/system-auth') do
        its('content') { should
match(/^\s*password\s+requisite\s+pam_pwquality\.so\s+(\S+\s+)*try_
first_pass/) }
        its('content') { should
match(/^\s*password\s+requisite\s+pam_pwquality\.so\s+(\S+\
s+)*retry=[3210]/) }
  end
  describe file('/etc/pam.d/password-auth') do
        its('content') { should


match(/^\s*password\s+requisite\s+pam_pwquality\.so\s+(\S+\s+)*try_
first_pass/) }
        its('content') { should
match(/^\s*password\s+requisite\s+pam_pwquality\.so\s+(\S+\
s+)*retry=[3210]/) }
  end
  describe file('/etc/pam.d/password-auth') do
        its('content') { should
match(/^\s*password\s+requisite\s+pam_pwquality\.so\s+(\S+\s+)*try_
first_pass/) }
       its('content') { should
match(/^\s*password\s+requisite\s+pam_pwquality\.so\s+(\S+\
s+)*retry=[3210]/) }
  end
  describe parse_config_file('/etc/security/pwquality.conf') do
        its('minlen') { should match(/1[4-9]|[2-9][0-9]|[1-9][0-9][0-9]+/) }
        its('dcredit') { should match(/-[1-9][0-9]{0,}/) }
        its('ucredit') { should match(/-[1-9][0-9]{0,}/) }
       its('ocredit') { should match(/-[1-9][0-9]{0,}/) }
        its('lcredit') { should match(/-[1-9][0-9]{0,}/) }
 end
end
```

The pam_pwquality.so module checks the strength of passwords. It performs checks such as making sure a password is not adictionary word, is a certain length, contains a mix of characters (e.g. alphabet, numeric, other) and more. The control above inspects the content of the pam.d system-auth and password-auth config files to ensure:

- **try_first_pass - retrieve the password from a previous stacked PAM module. If not available, then prompt theuser for a password.**
- **retry=3 - Allow 3 tries before sending back a failure.**

It also inspects the settings in the pwquality.conf to ensure the following password strength settings:

- **minlen=14 - password must be 14 characters or more**
- **dcredit=-1 - provide at least one digit**
- **ucredit=-1 - provide at least one uppercase character**
- **ocredit=-1 - provide at least one special character**
- **lcredit=-1 - provide at least one lowercase character**

These values can be tuned to the needs of your individual organization.

2. Ensure system accounts are non-login

It's important to make sure that vendor-provided accounts are not being used by regular users and that they are preventedfrom providing an interactive shell.

```
control "cisecurity.benchmarks_rule_5.4.2_Ensure_system_accounts_are_
non-login" do
  title "Ensure system accounts are non-login"
  desc "There are a number of accounts provided with Red Hat 7 that
are used to manage applications and are not intended to provide an
interactive shell. Rationale: Prevent them from being used to provide an
interactive shell."
  impact 1.0
  tag "cis-rhel7-2.1.1": "5.4.2"
  tag "level": "1"
  tag "type": ["Server", "Workstation"]
  describe passwd.where { user =~ /^(?!root|sync|shutdown|halt).*$/ } do
    its("entries") { should_not be_empty }
  end
  describe passwd.where { user =~ /^(?!root|sync|shutdown|halt).*$/ &&
uid.to_i < 1000 && shell != "/sbin/nologin" } do
    its("entries") { should be_empty }
  end
end
```

This control inspects settings for all system accounts. By default, Red Hat Enterprise Linux 7 sets the password field for these accounts to an invalid string. Therefore, their password entries. should not be empty. It also looks at all accounts with UID below 1000 to ensure that the shell field in the password file be set to /sbin/nologin. This prevents the account from potentially being used to run any commands.

3. Ensure password fields are not empty

All accounts, including vendor-supplied accounts, must have passwords or be locked to prevent the account from being used by an unauthorized user. If an account has an empty password field that means that anybody may authenticate as that user without providing a password.

```
control "cisecurity.benchmarks_rule_6.2.1_Ensure_password_fields_are_
not_empty" do
  title "Ensure password fields are not empty"
    desc "An account with an empty password field means that anybody
may log in as that user without providing a password. Rationale: All
accounts must have passwords or be locked to prevent the account from
being used by an unauthorized user."
    impact 1.0
  tag "cis-rhel7-2.1.1": "6.2.1"
  tag "level": "1"
    tag "type": ["Server", "Workstation"]
    shadow.users(/.+/).entries.each do |entry|
      describe entry do
          its('passwords') { should_not eq [''] }
      end
  end
end
```

This control uses Ruby along with InSpec built-in functions to iterate through every user account in /etc/shadow. For each account found, it ensures its password is not empty.

4. Ensure inactive password lock is 30 days or less

Inactive accounts pose a threat to system security. This includes vendor-supplied accounts, which should be automatically disabled if they have not been accessed in 30 days or less. Further, this protects the system from inactivity that may prevent users from noticing anomalies in their accounts.

In this example, the control uses an in-line Bash script to inspect the output of the "getent shadow" command. The script handles the logic for examining the system and InSpec simply examines the exit code of the script. This approach is useful for integrating existing tools or scripts already in use throughout your organization.

```
control "cisecurity.benchmarks_rule_5.4.1.4_Ensure_inactive_pass_lock_
is_30_days_or_less" do
   title "Ensure inactive password lock is 30 days or less"
   desc "User accounts that have been inactive for over a given period
of time can be automatically disabled. Rationale: Inactive accounts pose
a threat to system security since the users are not logging in to notice
failed login attempts or other anomalies."
   impact 1.0
  describe file("/etc/default/useradd") do
     its("content") { should
match(/^\s*INACTIVE\s*=\s*(30|[1-2][0-9]|[1-9])\s*(\s+#.*)?$/) }
   end
    describe bash( "#!/usr/bin/env sh\n\n#\n# CIS-CAT Script Check
Engine\n# \n#  Name              Date         Description\n# --------------
-------------------------------------------------------------------------\n# B. Munyan
7/20/16 Ensure no users have a password inactivity period > 30\n#
\n\noutput=$(\n/usr/bin/getent shadow | awk -F : 'match($2, /^[^!*]/)
&& ($7 == \"\" || $7 > 30) { if ($7 == \"\") { print \"User \" $1 \"
password inactivity period is not defined\" } else { print \"User \" $1
\" Password Inactivity Period > 30 (\" $7 \") \" } }'\n)\n\n# we captured
output of the subshell, let's interpret it\nif [ \"$output\" == \"\" ]
; then\n exit $              XCCDF_RESULT_PASS\nelse\n     # print the reason
why we are failing\n      echo \"$output\"\n           exit $XCCDF_RESULT_FAIL\
nfi\n") do
     its("exit_status") { should eq 0 }
     end
end
```

Requirement 3: Protect Stored Cardholder Data

Section 3 of the PCI DSS examines protection methods for limiting access to critical components of cardholder data, even when it is being accessed by authorized users or by intruders that have circumvented other security controls. Because this requirement of the PCI DSS is mainly concerned with potential risk mitigation techniques of custom data, schema, and storage requirements, there are no "one size fits all" controls included in the CIS controls to account for all possible scenarios.You must build controls custom to your organizational needs. Chef makes that task easily achievable using InSpec's resource model.

1. Ensure a non-authorized user cannot reach cardholder data

In this example, we could write a control to ensure that unauthorized database users cannot access credit card numbers.

This example control would connect to a known production Oracle database (ora01. mycompany.com) and attempt to authenticate as the users "sys", "system", and "alice". In this example we presume that these users should not have access tostored credit card information. Therefore, when they attempt to select these rows there should be no data returned.

```
control "mycompany.custom_rule1.0.0_Ensure_non_privileged_users_cant_
access_cc_numbers" do
  title "Ensure that non-privileged database users do not see CC
numbers"
  desc "A legitimate and authorized database user account should not
be able to access credit card numbers if they are not specifically
authorized to do so. Rationale: Only privileged accounts should be able
to access cardholder data."
  impact 1.0
  tag "level": "1"
  tag "type": ["Server", "Workstation"]
  %w[sys system alice].each do |unprivileged|
    describe oracledb_session(user: unprivileged, password:'password',
service:'ora01.mycompany.com').query('SELECT creditcard FROM accounts').
rows do
 its('count') { should eq 0 }
    end
  end
end
```

Requirement 4: Encrypt transmission of cardholder data across open, public networks

Sensitive information must be encrypted during transmission over networks that are easily accessed by malicious individuals.Section 4.1 of the PCI DSS requires use of strong cryptography and security protocols to safeguard sensitive cardholder dataduring transmission over open, public networks, including use of secure versions. Some protocol implementations (such as SSL, SSH v1.0, and early TLS) have known vulnerabilities that an attacker can use to gain control of the affected system.

1. Ensure that the TLS 1.2 protocol is active on any SSL ports

In this example control, we check to see that TLS 1.2 is enabled on any SSL ports listening for connections.

```
control "mycompany.custom_rule2.0.0_Ensure_TLS_v1.2_is_in_use" do
  title "Ensure that TLS 1.2 is the encryption protocol in use"
  desc "TLS 1.2 is a known secure protocol that should be used for
SSL connections. Rationale: SSL2, SSL3, TLS1.0, and TLS1.1 all contain
known vulnerabilities that an attacker can use to gain control of these
systems."
  impact 1.0
  tag "level": "1"
  tag "type": ["Server", "Workstation"]
  sslports.each do |socket|
    proc_desc = "on node == #{command('hostname').stdout.strip} running
#{socket.process.inspect} (#{socket.pid})"
    describe ssl(port: socket.port).protocols('tls1.2') do
      it(proc_desc) { should be_enabled }
      it { should be_enabled }
    end
  end
end
```

This control is an example of using InSpec's built-in "ssl" resource. While this control ensures TLS 1.2 is the active protocol on any SSL ports, it does not ensure that other non-secure protocols are disabled. For a more thorough check of SSL settings,refer to the dev-sec SSL benchmark.

Requirement 5: Protect all Systems Against Malware and Regularly Update AV Software

Maintaining a vulnerability management program includes ensuring that system patches and security software is available and installed on all systems. For RHEL 7 servers, this includes ensuring that all package manager repositories are properly configured so systems may receive vendor-provided updates.

1. Ensure package manager repositories are configured

If a system's package repositories are misconfigured, important patches may not be identified or a rogue repository could

introduce compromised software. Systems should have their authorized package manager repositories configured in a way that ensures they receive the latest patches and updates.

This control uses Ruby to iterate through all package repositories defined as Red Hat system repos in the CIS benchmarks. Each of those repositories must be defined on the system and enabled. The control also ensures that repositories set up via yum to install software from additional resources are also defined and enabled on the system.

```
control "cisecurity.benchmarks_rule_1.2.1_Ensure_pkg_manager_repos_are_
configured" do
    title "Ensure package manager repositories are configured"
    desc "Systems need to have package manager repositories configured
to ensure they receive the latest patches and updates. Rationale: If a
system's package repositories are misconfigured important patches may
not be identified or a rogue repository could introduce compromised
software."
     impact 0.0
     tag "cis-rhel7-2.1.1": "1.2.1"
     tag "level": "1"
     tag "type": ["Server", "Workstation"]
     REDHAT_REPOS.each do |repository|
           describe yum.repo(repository) do
              it { should exist }
              it { should be_enabled }
           end
   end
     cmd = command('yum repolist enabled').stdout.split("\n")
     get_other_repos = cmd.slice(2..cmd.length-2) || []
     other_repos = get_other_repos.map { |repositories| repositories.
gsub(/\s.+/, '') }
     other_repos -= REDHAT_REPOS
     unless other_repos.empty?
           other_repos.each do |repository|
             describe yum.repo(repository) do
             it { should_not exist }
                it { should_not be_enabled }
           end
           end
```

2. Ensure updates, patches, and additional security software is installed

Periodically, patches are released for included software either due to security flaws or to include additional functionality. Rationale: Newer patches may contain security enhancements that would not be available through the latest full update. As a result, it is recommended that the latest software patches be used to take advantage of the latest functionality. As with anysoftware installation, organizations need to determine if a given update meets their requirements and verify the compatibilityand supportability of any additional software against the update revision that is selected.

```
control "mycompany.custom_rule3.0.0_Ensure_updates_and_patches_
installed" do
        title "Ensure that all updates, patches, and latest packages are
installed"
        desc "Periodically patches are released for included software either
due to security flaws or to include additional functionality. Rationale:
Newer patches may contain security enhancements that would not be
available through the latest full update. As a result, it is recommended
that the latest software patches be used to take advantage of the latest
functionality."
        impact 1.0
        tag "cis-rhel7-2.1.1": 1.8
        tag "level": "1"
        tag "type": ["Server", "Workstation"]
        linux.update.updates.each { |update|
          describe package(update['name']) do
            its('version') { should eq update['version'] }
          end
  }
        only_if { linux_update.updates.length > 0 }
          end
```

This control uses a custom Chef InSpec resource to defer management of updates to the local package manager based on Linux distribution. For a more thorough approach to ensuring proper Linux patching, refer to the dev-sec Linux Patchbenchmark4.

Requirement 6: Develop and Maintain Secure Systems and Applications

Attackers use system vulnerabilities to gain unauthorized privileged access to systems. Requirement 6 of the PCI DSS is a lengthy description of several approaches that collectively represent sound risk-mitigation strategies to reduce your company's exposure to different attack vectors. From a system level, it is also critical to capture and report suspicious security events and take reasonable precautions to prevent unauthorized access to begin with.

This section contains a small sample of steps you can take to accomplish those goals.

1. Ensure iptables is installed

On Red Hat Enterprise Linux systems, iptables is useful for firewall management and configuration. It allows configuration of the IPv tables in the Linux kernel and the rules stored within them. Most firewallconfiguration utilities operate as a front-end to iptables.

Checking installation is easily accomplished with the InSpec "package" resource.

```
control "xccdf_org.cisecurity.benchmarks_rule_3.6.1_Ensure_iptables_is_
Installed" do
  title "Ensure iptables is installed"
  desc "iptables allows configuration of the IPv4 tables in the linux
kernel and the rules stored within them. Rationale: iptables is required
for firewall management and configuration."
  impact 1.0
  tag "cis-rhel7-2.1.1": "3.6.1"
  tag "level": "1"
  tag "type": ["Server", "Workstation"]
  describe package('iptables') do
    it { should be_installed }
  end
end
```

2. Ensure a default deny firewall policy

When using iptables, a default “accept” policy will accept any packet that is not configured to be denied. It is easier to whitelist acceptable usage than it is to blacklist unacceptable usage. Set a default “deny all” policy on connections to ensureany unconfigured network usage is rejected.

```
control "cisecurity.benchmarks_rule_3.6.2_Ensure_default_deny_firewall_
policy" do
  title "Ensure default deny firewall policy"
 desc "A default deny all policy on connections ensures that any
unconfigured network usage will be rejected. Rationale: With a default
accept policy the firewall will accept any packet that is not configured
to be denied. It is easier to whitelist acceptable usage than to
blacklist unacceptable usage."
  impact 1.0
  tag "cis-rhel7-2.1.1": "3.6.2"
  tag "level": "1"
  tag "type": ["Server", "Workstation"]
  %w[INPUT OUTPUT FORWARD].each do |chain|
    describe.one do
      describe iptables do
        it { should have_rule("-P #{chain} DROP") }
      end
      describe iptables do
        it { should have_rule("-P #{chain} REJECT") }
      end
    end
  end
end
```

This control uses the InSpec built-in iptables resource to ensure that all INPUT, OUTPUT, and FORWARD rules are either dropped or rejected, by default, unless explicitly allowed.

The use of "describe.one" is effectively an OR test. Only one of these matches must be found in order for the control to be met.

Requirement 7: Restrict access to cardholder data by business need to know

To ensure critical data can only be accessed by authorized personnel, systems and processes must be in place to limit access based on need-to-know and according to job responsibilities. "Need to know" is when access rights are granted to allow only the least data and privileges needed to perform a job. At the very minimum, we can restrict access to administrative accounts.

1. Ensure access to the su command is restricted

Restricting the use of the "su" command, and using "sudo" in its place, provides system administrators better control of the escalation of user privileges to execute privileged commands. The sudo utility also provides a better logging and auditmechanism, as it can log each command executed via sudo. Normally, the su command can be executed by any user. By uncommenting the pam_wheel.so statement in the /etc/pam.d/su file, the su command will only allow users in the wheelgroup to execute "su".

This control inspects the contents of /etc/pam.d/su to ensure the correct content is found.

```
control "cisecurity.benchmarks_rule_5.6_Ensure_access_to_the_su_command_
is_restricted" do
  title "Ensure access to the su command is restricted"
  desc "The su command allows a user to run a command or shell as
another user. The program has been superseded by sudo , which allows for
more granular control over privileged access. Normally, the su command
can be executed by any user. By uncommenting the pam_wheel.so statement
in /etc/pam.d/su, the su command will only allow users in the wheel
group to execute su."
  impact 1.0
  tag "cis-rhel7-2.1.1": 5.6
  tag "level": "1"
  tag "type": ["Server", "Workstation"]
  describe file("/etc/pam.d/su") do
    its("content") { should
match(/^\s*auth\s+required\s+pam_wheel.so\s+(\S+\s+)*use_uid\s*(\S+\
s+)*$/) }
  end
end
```

2. Ensure SSH root login is disabled

Disallowing root logins over SSH requires system admins to authenticate using their own individual account, then escalating to root only via sudo (if you've disabled access to "su" as in the control above). This restriction limits opportunity for non-repudiation and provides a clear audit trail in the event of a security incident. The PermitRootLogin parameter specifies if the root user can login using ssh.

This control uses InSpec's built-in sshd_config resource to inspect the configuration of the OpenSSH daemon.

```
control
"cisecurity.benchmarks_rule_5.2.8_Ensure_SSH_root_login_is
_disabled" do
   title "Ensure SSH root login is disabled"
   desc "The PermitRootLogin parameter specifies if the root
user canlogin using ssh(1). The default is no. Rationale:
Disallowing root
logins over SSH requires system admins to authenticate
using their ownindividual account, then escalating to root via
sudo"
   impact 1.0
   tag "cis-rhel7-2.1.1": "5.2.8"
   tag "level": "1"
   tag "type": ["Server",
   "Workstation"]describe
   sshd_config do
     its('PermitRootLogin') {
   should eq 'no' }end
end
```

3. Ensure SSH access is limited

Restricting which users can remotely access the system via SSH will help ensure that only authorized users access the system. There are several options available to limit which users and groups can access the system via SSH. It is recommended that at least one of the following options be leveraged: AllowUsers, AllowGroups, DenyUsers, or DenyGroups. These are each space-separated lists of usernames to allow, group names to allow, usernames to deny, or group names to deny. Numeric values for users and groups are not recognized by these constructs.

```
        control "cisecurity.benchmarks_rule_5.2.15_Ensure_SSH_access_
is_limited"
        do
            title "Ensure SSH access is limited"
            desc "There are several options available to limit which users
and
        group can access the system via SSH. It is recommended that at
least
        one of the following options be leveraged: AllowUsers,
AllowGroups,
        DenyUsers, or DenyGroups. Rationale: Restricting which users
can
        remotely access the system via SSH will help ensure that only
authorized
        users access the system."
            impact 1.0
            tag "cis-rhel7-2.1.1": "5.2.15"
            tag "level": "1"
            tag "type": ["Server", "Workstation"]
            describe.one do
              describe sshd_config do
                 its('AllowUsers') { should match /[\S|\s]+/ }
              end
              describe sshd_config do
                 its('AllowGroups') { should match /[\S|\s]+/ }
              end
              describe sshd_config do
                 its('DenyUsers') { should match /[\S|\s]+/ }
              end
              describe sshd_config do
                its('DenyGroups') { should match /[\S|\s]+/ }
              end
            end
          end
```

Requirement 8: Identify and Authenticate Access to System Components

Assigning unique IDs to each person with access ensures that individuals are uniquely accountable for their actions. Withaccountability in place, any actions taken on critical data and systems can be traced to known and authorized users and processes. The effectiveness of authentication depends on several design details—particularly, how frequently password attempts can be made by an attacker, and the security methods to protect user passwords at the point of entry, during transmission, and while in storage.

1. Ensure SSH PermitEmptyPasswords is disabled

Disallowing remote shell access to accounts that have an empty password reduces the probability of unauthorized access to the system. The PermitEmptyPasswords parameter specifies if the SSH server allows login to accounts with empty password strings. This control uses InSpec's built-in sshd_config resource to inspect the configuration of the OpenSSH daemon.

```
control "cisecurity.benchmarks_rule_5.2.9_Ensure_SSH_
PermitEmptyPasswords_is_disabled" do
  title "Ensure SSH PermitEmptyPasswords is disabled"
  desc "The PermitEmptyPasswords parameter specifies if the SSH server
allows login to accounts with empty password strings. Rationale:
Disallowing remote shell access to accounts that have an empty password
reduces the probability of unauthorized access to the system"
  impact 1.0
  tag "cis-rhel7-2.1.1": "5.2.9"
  tag "level": "1"
  tag "type": ["Server", "Workstation"]
  describe sshd_config do
    its('PermitEmptyPasswords') { should eq 'no' }
  end
end
```

2. Ensure SSH LogLevel is set to INFO

SSH provides several logging levels with varying amounts of verbosity. DEBUG is specifically not recommended other than strictly for debugging SSH communications since it provides so much verbose information that it becomes difficult to identify important security information. Setting the log level to INFO records login activity of SSH users. In situations like Incident Response, it is important to determine when a particular user was active on a system. The logout record can eliminate those users who disconnected, which helps narrow the search field. This control uses InSpec's built-in sshd_config resource to inspect the configuration of the OpenSSH daemon.

```
control "cisecurity.benchmarks_rule_5.2.3_Ensure_SSH_LogLevel_is_set_to_
INFO" do
  title "Ensure SSH LogLevel is set to INFO"
  desc "The INFO parameter specifies that login and logout activity
will be logged. Rationale: SSH provides several logging levels with
varying amounts of verbosity. INFO level is the basic level that only
records login and logout activity of SSH users."
  impact 1.0
  tag "cis-rhel7-2.1.1": "5.2.3"
  tag "level": "1"
  tag "type": ["Server", "Workstation"]
  describe sshd_config do
    its('LogLevel') { should eq 'INFO' }
  end
end
```

3. Ensure SSH MaxAuthTries is set to 4 or less

Setting the MaxAuthTries parameter to a low number will minimize the risk of successful brute force attacks to the SSHserver. While the recommended setting is 4, your setting may be. different. The MaxAuthTries parameter specifies the maximum number of authentication attempts permitted per connection. When the login failure count reaches half the number, error messages will be written to the syslog file detailing the login failure. This control uses InSpec's built-in sshd_config resource to inspect the configuration of the OpenSSH daemon to look for asetting that is equal to or less than 4.

```
control "cisecurity.benchmarks_rule_5.2.5_Ensure_SSH_MaxAuthTries_set_
to_4_or_less" do
  title "Ensure SSH MaxAuthTries is set to 4 or less"
  desc "The MaxAuthTries parameter specifies the maximum number of
authentication attempts permitted per connection. When the login failure
count reaches half the number, error messages will be written to the
syslog file. Setting the MaxAuthTries parameter to a low number will
minimize the risk of successful brute force attacks to the SSH server."
  impact 1.0
  tag "cis-rhel7-2.1.1": "5.2.5"
  tag "level": "1"
  tag "type": ["Server", "Workstation"]
  describe sshd_config do
     its('MaxAuthTries') { should cmp <= 4 }
  end
end
```

Requirement 10: Track and Monitor all Access to Network Resources and Cardholder Data

Logging mechanisms and the ability to track user activities are critical in preventing, detecting, or minimizing the impact of a data compromise. The presence of logs in all environments allows thorough tracking, alerting, and analysis when something does go wrong. Determining the cause of a compromise is very difficult, if not impossible, without system activity logs.

1. Ensure rsyslog or syslog-ng is installed

The security enhancements of rsyslog and syslog-ng such as connection-oriented (i.e. TCP) transmission of logs, the option tolog to database formats, and the encryption of log data en route to a central logging server, justify installing and configuring the package. These alternative logging systems are recommended replacements to the original syslogd daemon. The use of "describe.one" ensures that either rsyslog OR syslog-ng must be installed in order for the requirements of thiscontrol to be met.

```
control "cisecurity.benchmarks_rule_4.2.3_Ensure_rsyslog_or_syslog-ng_
is_installed" do
  title "Ensure rsyslog or syslog-ng is installed"
  desc "The rsyslog and syslog-ng software are recommended replacements
to the original syslogd daemon which provide improvements over syslogd.
The security enhancements of rsyslog and syslog-ng such justify
installing and configuring the package."
  impact 1.0
  tag "cis-rhel7-2.1.1": "4.2.3"
  tag "level": "1"
  tag "type": ["Server", "Workstation"]
  describe.one do
    describe package("rsyslog") do
        it { should be_installed }
    end
    describe package("syslog-ng") do
        it { should be_installed }
    end
  end
end
```

2. Ensure permissions on all log files are configured

It is important to ensure that log files have the correct permissions to ensure that sensitive data is archived and protected. Log files stored in /var/log contain logged information from many services on the system, or on centralized logging hosts, datafrom services on other systems. This control uses InSpec's file resource to ensure that all conditions are true in order for the requirements of this control to bemet.

```
control "cisecurity.benchmarks_rule_4.2.4_Ensure_perms_on_all_logfiles_
configured" do
1010    title "Ensure permissions on all logfiles are configured"
1011    desc "Log files stored in /var/log/ contain logged information
from many services on the system, or on log hosts others as well.
Rationale: It is important to ensure that log files have the correct
permissions."
  impact 1.0
  tag "cis-rhel7-2.1.1": "4.2.4"
  tag "level": "1"
  tag "type": ["Server", "Workstation"]
  command('find /var/log -type f').stdout.split("\n").each do |log_file|
    describe file(log_file) do
        it { should_not be_writable.by('group') }
        it { should_not be_executable.by('group') }
        it { should_not be_readable.by('other') }
        it { should_not be_writable.by('other') }
        it { should_not be_executable.by('other') }
    end
  end
end
```

3. Ensure suspicious packets are logged

When enabled, this feature logs packets with un-routable source addresses to the kernel log. Enabling this feature and logging these packets allows an administrator to investigate the possibility that an attacker is sending spoofed packets to theirsystem. This control uses InSpec's built-in kernel_parameter resource to test kernel parameters on Linux platforms. These parametersare located under /proc/cmdline.

```
control "cisecurity.benchmarks_rule_3.2.4_Ensure_suspicious_packets_are_
logged" do
  title "Ensure suspicious packets are logged"
  desc "When enabled, this feature logs packets with un-routable
source addresses to the kernel log. Rationale: Enabling this feature
and logging these packets allows an administrator to investigate the
possibility that an attacker is sending spoofed packets to their
system."
  impact 1.0
  tag "cis-rhel7-2.1.1": "3.2.4"
  tag "level": "1"
  tag "type": ["Server", "Workstation"]
  describe kernel_parameter('net.ipv4.conf.all.log_martians') do
    its('value') { should eq 1 }
  end
  describe kernel_parameter('net.ipv4.conf.default.log_martians') do
    its('value') { should eq 1 }
  end
end
```

Requirement 11: Regularly Test Security Systems and Processes

The PCI DSS requires regular tests of security systems and processes. PCI DSS sections 11.2, 11.2.3, and 11.3.3 all require running internal scans against CDE components to detect, report, and remediate any possible system vulnerabilities. Yet manycompanies fail to meet requirement 11.

The 2020 Verizon Payment Security Report found that only 27.9% or organizations achieved 100% compliance. And that for the 10th year in a row, Requirement 11 posed the most difficult for companies trying to achieve full compliance. The lowest rates of compliance are getting worse year over year with only 51.9% of companies having full compliance in 2019. The report further indicates that currently companies across several verticals struggle the most with this particular requirement of the PCI DSS.

Adopting a continuous compliance approach with Chef Compliance allows you to satisfy audit requirements at any time and make audits painless. Chef Compliance allows you to detect fleetwide compliance shortcomings, prioritize, and automate necessary remediations.

Chef Compliance provides a clear view of both technology assets and real-time compliance status. Chef Compliance's easy-to-use, agentless detect mode helps you quickly assess the state of security on your systems. It's built-in metadata for impact and severity scoring allows to easily determine which areas to focus on for subsequent remediation. Severity levels per control can also be customized to increase or decrease the criticality of findings—or even turn them off if they are mitigated by other compensating controls in the environment.

PCI DSS sections 11.5 & 11.6 require use of a change-detection mechanism as well as documentation of security policy and operational procedures. Chef Compliance one to set up a detect-and-correct cycle to help respond to unauthorized changes on systems and simultaneously document the correct systems policy with use of tools like configuration management.

Setting up a continuous test and remediation cycle can help make compliance part of the development process. It can also eliminate wasteful pre-production security scans, which slow down development and create rework for engineers, by regularly integrating those scans into development processes. It allows organizations to communicate clear policy and procedural intent by replacing rules written in English (e.g., in imprecise formats like PDF or Excel) with executable code that enables collaboration with development teams.

New vulnerabilities are discovered every day. Rather than waiting for a vendor to issue opaque detection rules in a quarterly fix pack, Chef Compliance's human-readable language allows you to write and publish vulnerability detection code that same day and immediately use it to keep the company safe.

Find and remediate compliance issues as they happen, rather than in quarterly audits. Making this type of regular test and remediation cycle an integrated part of your work has the added benefit of meeting many of the requirements in this section of the PCI DSS compliance.

Requirements 9 & 12

PCI DSS requirements 9 (Restrict physical access to cardholder data) and 12 (Maintain a policy that addresses information security for all personnel) apply to physical access (e.g., authorized entrance to a facility) and to process controls that cannotbe automatically inspected using Chef Compliance.

Conclusion

This guide is not a definitive set of controls necessary to fulfill all the requirements in a PCI DSS audit, but it does illustrate what these controls do, how they apply to PCI requirements, and provides a practical guide to understanding which CIS benchmarks map to which PCI requirements.

In practice, members of both technical and compliance teams working with systems in the CDE would not have to composethese tests from scratch. A majority of the tests in this whitepaper are a part of the CIS benchmarks for Red Hat Enterprise Linux 7, available for use with a Chef Compliance license.

Our guide demonstrates some of the core functionality available when using Chef Compliance to implement compliance reporting controls. By adopting a continuous compliance approach, organizations can quickly answer audit questions at any time--not just quarterly or yearly. By customizing these controls to meet the needs of particular environment, one can enter an audit cycle knowing exact compliance posture and avoid being surprised by auditors finding unexpected weak points in theenvironment.

To see how Chef Compliance can help you achieve continuous compliance and reduce the time your teams spend on fulfillingaudit requests, visit chef.io/compliance.

A recommended next step would be to use Chef Compliance to integrate automated compliance assessments into your continuous delivery workflow. Find out more about how Chef Compliance enables continuous assessments by visiting http://www.chef.io/compliance

Legal Disclaimer

Progress Software Corporation does not provide legal advice. This document is intended for informational purposes only and does not constitute legal advice, nor shall this document or any software product or other offering referenced herein serve as a substitute for the reader's compliance with any laws (including but not limited to any act, statute, regulation, rule, directive, administrative order, executive order, etc. (collectively, "Laws")) referenced in this document. Please consult with competent legal counsel regarding any Laws referenced herein.

This document is provided "as is" without warranty of any kind. All express or implied representations, conditions and warranties, including any implied warranty of merchantability or fitness for a particular purpose, are disclaimed, except tothe extent that such disclaimers are determined to be illegal.

ABOUT PROGRESS

Progress (NASDAQ: PRGS) provides the best products to develop, deploy and manage highimpact business applications. Acquired in October 2020, Chef extends Progress offerings inDevOps and DevSecOps, with market-leading, modern infrastructure, compliance, and application automation. With Progress, you can accelerate the creation and delivery of strategic business applications, automate the process by which you configure, deploy and scale those apps, and make your critical data and content more accessible and secure— leading to competitive differentiation and business success. Over 1,700 independent software vendors, 100,000+ enterprise customers, and a three-million-strong developer community rely on Progress to power their applications. Learn about Progress
at www.progress.com or +1-800-477-6473.