DevOps Audit Defense Toolkit

March 22, 2015

DevOps Audit Defense Toolkit

1 Copyright and Authors

1.1 Copyright

For more information, please email: devopsaudittoolkit@itrevolution.net

http://www.itrevolution.com

1.2 About the Authors

James DeLuccia IV

James DeLuccia IV is a published author, practitioner, auditor, and currently a Senior Manager with Ernst & Young. He brings first-hand research and experience on third-party vendor trust and global security operations. Mr. DeLuccia leads the Americas certification and compliance services and acts as the executive implementing global security programs. He is certified as a CIA, CISA, CISM, CISSP, CPISA, CPISM, and degrees in Risk Management, Management Information Systems, and a MBA in Finance. His book, “IT Compliance and Controls: Best Practices for Implementation,” is globally available. Mr. DeLuccia's ongoing efforts focus on supporting and developing global information protection programs.

Jeff Gallimore

Jeff Gallimore has played many roles over the course of his 20+ years in information technology and consulting. He’s been a developer and architect building systems with various technologies, platforms, and methodologies. Jeff has also provided day-to-day and strategic support for high profile, complex technology programs. Throughout it all, he’s kept a passion for technology and how it can be used to support the business and achieve results. At Excella, Jeff leads Corporate Services Strategy, which includes building capabilities and thought leadership in software development, business intelligence, DevOps, program management, business analysis, and Agile. He’s proud of the solutions they provide to their clients every day – and of the work they do in the technical community.

Gene Kim

Gene Kim is a multiple award-winning CTO, researcher and author. He was founder and CTO of Tripwire for 13 years. He has written three books, including “The Visible Ops Handbook” and “The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win." Gene is a huge fan of IT operations, and how it can enable developers to maximize throughput of features from “code complete” to “in production,” without causing chaos and disruption to the IT environment. He has worked with some of the top Internet companies on improving deployment flow and increasing the rigor around IT operational processes. In 2007, ComputerWorld added Gene to the “40 Innovative IT People Under The Age Of 40” list, and was given the Outstanding Alumnus Award by the Department of Computer Sciences at Purdue University for achievement and leadership in the profession.

Byron Miller

Byron Miller is a technology enthusiast, space nut, astronomy geek, aspiring pilot, father of two great kids, and an avid cyclist who doesn’t get to do any of these things enough.

Byron studies complex systems, lean operations, and systems thinking in his daily work to continually improve and learn. He’s now working at HomeAway in Austin, TX building their cloud platform and services and serves as one of the organizers of the Austin Puppet User Group.

2 Introduction to the DevOps Audit Defense Toolkit

As IT organizations increasingly adopt DevOps patterns, there is more tension than ever between IT and audit. These new DevOps patterns challenge traditional thinking about auditing, controls, and risk mitigation. Just as “Dev” and “Ops” need to find new and better ways of working together to help their organization win, so now does IT and audit.

The goal of the DevOps Audit Defense Toolkit is to educate IT management and practitioners on the audit process so they can demonstrate to auditors they understand the business risks and are properly mitigating those risks.

We’ve studied a number of organizations using DevOps and continuous delivery practices that are also subject to various compliance requirements.

The Toolkit summarizes the techniques they use to mitigate risk, and also provides a section answering the most common questions about value creation, compliance, and DevOps. The information in this document should help organizations wanting to pursue DevOps and continuous delivery explain their approach and improve communication between IT and audit.

In the Toolkit, we describe a fictitious organization, including its business processes and control environment, and then a set of audit concerns and how management could successfully prove controls exist and are effective.

Our goal is to provide authoritative guidance about how management and auditors should conduct audits where DevOps practices are in place, in support of accurate financial reporting, regulatory compliance (e.g., SOX, HIPAA, FedRAMP, EU Model Contracts, and SEC Reg-SCI regulations), contractual obligations (e.g., PCI DSS, DOD DISA), or effective and efficient operations.

By doing this, we hope the DevOps Audit Defense Toolkit will elevate the state of the management practice. We want to address how to understand risks to business objectives and correctly scope and substantiate the effectiveness of controls -- all of which should reduce the costs and increase the effectiveness of audits and help the business win in the marketplace.

You can access the DevOps Audit Defense Toolkit community on Google+ at http://bit.ly/DevOpsAudit. In addition to being the community home for the Toolkit, people post great content all the time about DevOps and audits we’re sure you’ll find valuable.

3 Fictitious Organizational Description

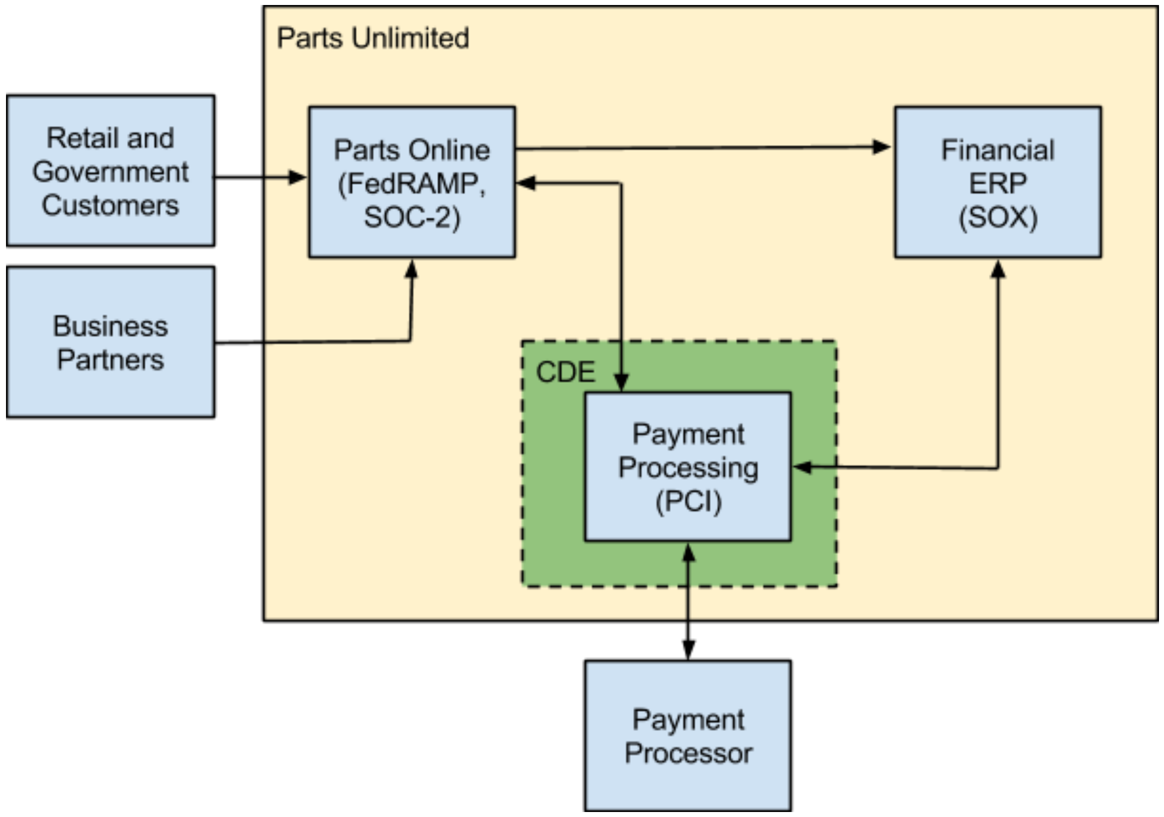
In the remainder of this document, we will present a set of scenarios between management and auditors in a fictitious company, Parts Unlimited. Parts Unlimited (PU) is a manufacturer and retailer with $4 billion in revenue annually.

This year, PU has the following relevant compliance domains:

- SOX: PU is publicly traded and must comply with SOX to attest their financial reporting is accurate. Management must also assess the effectiveness of their internal control over financial reporting.
- PCI: PU processes consumer credit cards in their retail stores and in their e-commerce application, and thus must comply with the PCI DSS.
- SOC-2: PU has numerous business partners who require a SOC-2 audit report to represent PU has adequate information security controls.
- FedRAMP: PU provides cloud-based services to the U.S. Federal Government. Therefore, PU must comply with the FedRAMP standards to provide assurance the relevant technology controls are effective at securing the online operations supporting services being provided to the Federal Government (e.g., transactions and data).

PU is currently a market leader in its industry, which is one indication they have effective operations and are successfully bringing features to market.

Many years prior, they had some high profile outages, but availability for their online services has improved dramatically. A high-level view of the interactions among customers, Parts Unlimited, and the Parts Online services is shown below.



3.1 Financial Applications

The PU financial applications include the Oracle Applications Financial Module. This module includes sub-modules, such as Accounts Receivable, Accounts Payable, Cash Management, and General Ledger.

These applications are key controls used to generate data for a part of the PU financial statements and are therefore considered by management in scope for SOX.

The financial applications have been primarily managed by internal IT, which typically used a conventional “waterfall” SDLC process and traditional change management approval processes.

However, to speed up delivery of functionality to internal and external customers, PU is increasingly allowing business analysts within IT (instead of developers) to develop and implement changes, such as report updates, vendor-provided functional patches, configuration options, data fixes, and interface table integrations.

Traditionally, business analysts would define the requirements for the work and validate the work completed by developers is correct. Increasingly, they are also implementing the work, as well, which some auditors are concerned about. Changes made by business analysts follow similar processes for review and testing to those for changes made by developers.

The team within internal IT that supports the financial applications has adopted Agile processes, including designation of a Product Owner, sprints, and end-of-sprint demos and retrospectives. Work is prioritized by the Product Owner, completed by the business analysts, and then validated and signed off by the Product Owner throughout the sprint, as well as at the end of sprint demo.

3.2 Major Operations: Parts Online

Parts Online (PO) is one of the major revenue-generating applications, which provides automotive parts ordering and fulfillment. PO is a homegrown JEE application written by 50 developers who use many DevOps work patterns. PO is hosted in PU’s private cloud environment (on-premise).

Developers are doing tens to hundreds of production deployments per day and perform their own code deployments. IT Operations is responsible for the correct functioning of the service in production. However, when deployments fail or when major incidents occur, IT Operations will bring in Development to assist with break/fix issues.

$1 billion in orders flow through PO annually. Because this represents 25% of PU’s revenue, PO is in scope for SOX. And because the Federal Government leverages this application of PO, PO is also a part of the boundary certified against FedRAMP, a program for assessing, authorizing, and monitoring the security of cloud-based products and services used by the Federal Government.

There is a centralized change management advisory board (CAB) whose approval is required for some production changes. However, most PO development code changes have been declared as "standard changes," per the published change control procedure document. This approach means most changes are pre-approved based on documented guidelines. These guidelines provide clarity on when a change should be considered “standard” based on which components of the system it involves, the type of change it is (e.g., user interface, APIs, database schema), and who made the change. To ensure quality of code changes, the development team relies on an independent peer review process (i.e., the person who reviews the code isn’t the one who wrote the code) driven through GitHub Enterprise (on-premise).

3.3 Payment Processing Operations

PO takes customer credit orders through a set of internally-developed payment processing applications that handle online order entry called ICHT (I Can Haz Tokens). ICHT handles customer-entered cardholder data, tokenizes it, communicates with the payment processor, and completes the order transaction.

The scope of the PCI DSS cardholder data environment (CDE) is “the people, processes and technology that store, process or transmit cardholder data or sensitive authentication data, including any connected system components.”

To contain the scope of PCI DSS control requirements, ICHT runs on a separate cage of servers on-premise and is logically isolated from rest of the PU production network to ensure no cardholder data is ever transmitted to the rest of the PU production environment.

The ICHT is managed by a completely separate application team of two developers, one development manager, two network engineers, two database administrators, and two system administrators. The ICHT team follows the same processes used by the teams managing the rest of PO.

4 The Audits Begin

The story begins as the external SOX auditors, PCI assessors, and FedRAMP certification auditors have finished their audit planning processes, and those teams are starting to engage with PU staff.

Management receives from the auditors their initial documentation requests, the first of which is called the Pull List. In this case, the Pull List is a blank 100-row spreadsheet describing the compliance domain being addressed, the major business processes being audited, and then asking for a management point of contact for the various areas of IT areas being audited (e.g., applications, databases, change management process owners, information security) The Pull List also includes requests for documentation around major control areas. A sample of the Pull List is provided in the table below. During the audit, the sample count and desired artifacts will be highlighted on a per system basis as it is appropriate. See the "Audit Testing and Evidence" section later in this document for a more complete set of evidence provided to the auditors.

Control Obj	Control Procedure	In Scope Area	POC	Evidence request	Specifications	Due Date
A.10.1.2 Change management	Changes to information processing facilities and systems shall be controlled.	Financial Apps, Payment Application, Parts Online	Steve	For a selection of changes, get Move/Add/Remove/Delete/Modify service requests and noted it was approved by an authorized person.		
Separation of duty and access	Development, test and operational facilities shall be separated to reduce the risks of unauthorized access or changes to the operational system.	Financial Apps, Payment Application, Parts Online	Mary	Select production accounts, and confirm that they are authorized IT Operations staff, roles, responsibilities		

As PU IT managers start being assigned to rows in the auditor Pull List, they become immediately concerned by certain documentation requests for controls that don't exist. One example is, "Provide 25 of the latest change management approvals for the PO application."

The situation gets worse when the auditors start having to answer questions such as, "What evidence should we provide you if we don't have a change approval process?" and, "We have always allowed our developers to perform production deployments."

As a result, the on-site auditors escalate this issue to their managers. The escalation leads to more auditors showing up, all wanting to schedule more meetings.

5 The Auditor Concerns and Management Responses

Each of the auditors' objections are listed below, as well as the responses given by the PU team to show they understand the business risks and have internal controls that properly mitigate the risks (i.e., either that the organization can prevent the risk from occurring or that it can quickly detect a problem and recover from it.)

5.1 Concern #1: PO Developers Are Deploying Their Own Code Without Any Formal Change Approval Processes

The SOX auditor expresses concerns that "the absence of separation of duty and change approval controls create the risk of untested and unauthorized code being introduced into production."

Understanding the Auditor Rationale

Restricting developer access to production environments is an example of a "separation of duty" control. Separation of duties (SoD) is the concept of having more than one person required to complete a task to prevent fraud and error.

Examples of SoD controls are:

- One team creates sales transactions and another team reconciles the transactions for accuracy.
- One team creates purchase orders, and a manager must approve orders above $10,000.
- The development team implements a feature in code, and the IT Operations team deploys the code into production.

In the following sections, we describe how PO management demonstrates to the auditors they understand the business risks and have an effective control environment that mitigates the risks. We also identify the evidence demonstrating the controls are effective and operating as designed.

Management Risk Analysis

In this section, we walk through the risk management thought process presented to the auditors, showing them the top business risks were identified and an effective control environment exists to prevent, detect, and correct those risks. This is achieved through developed and verified entity-level controls, and by showing the designed controls are deployed and deemed satisfactory based on this risk management process that validates the effectiveness of these controls.

For the Parts Online application, management identifies some of the top business risks as the following:

- BR1: Parts Online service goes down, and revenue can't be generated (availability)

- BR2: Parts Online user accounts and data are compromised or disclosed (confidentiality, security, privacy)
- BR3: Parts Online transactions are incorrect or compromised (integrity)

PO shows the auditors the following risk analysis they performed, not for audit compliance reasons, but to achieve their operational goals.

Business Risk	Control Strategy
R1. An internal actor abuses provided or developed privileges to commit fraud to the organization and/or its customers. (BR2, BR3)	CS1. All code is validated through defined controls prior to production deployment to prevent developers from inserting “back doors” or vulnerabilities into production.
R2. Code is deployed into production that causes an outage, service impairment, or errors in data. (BR1)	CS2. All code is validated prior to production deployment to ensure the service runs correctly in production and interruptions can be fixed quickly.
R3. An external actor gains unauthorized access to production or pre-production environments (e.g., database, OS, networking) and installs malicious code, or changes or steals data. (BR2, BR3)	CS3. Unauthorized access is prevented, detected, and corrected through the regular review of access credentials and system configuration based on the published SLA for each element of the environment (e.g., database, OS, networking, virtualization, storage, continuous integration servers).

To achieve the control strategies above, PO has designed the following control environment:

1) CS1. All code is validated through defined controls prior to production deployment to prevent developers from inserting “back doors” or vulnerabilities into production.
   a) Jenkins runs static code analysis on each code checkin to validate the code conforms to the established lexical, syntactic, and semantic ruleset.
      i) The static code analysis tool ruleset is checked into source control.
      ii) Each rule is defined as either a breaking change or non-breaking change. If a violation to a rule defined as a breaking change occurs, the build breaks.
      iii) Changes to the ruleset are approved by the CAB.
   b) All code is peer-reviewed for correctness and adherence to the organizational coding standards. To ensure the peer review happened and the reviewers are "qualified" to do the review, Development and IT Operations management has instituted the following peer review process.
      i) The published organizational coding standards closely follow, by example, Google’s published coding standards.
      ii) The code reviews include examining the appropriateness of automated tests assessed against the team’s documented automated testing standards.
      iii) Using Stash, three developers are selected at random to review every code checkin, which prevents collusion to get a change in.
      iv) Code reviews are recorded in the JIRA promotion ticket.
      v) Code is not promoted to downstream environments (e.g., UAT, staging, production) until the peer review is complete.
      vi) For any changes required to the code prior to release, a task is added to the backlog and assigned to a developer to fix.
      vii) Other recommendations not addressed prior to the release are added to the technical debt backlog.

viii) Development and IT Operations management reviews peer review statistics to see how often code is approved versus rejected to determine if any reviewers start “rubber-stamping” approvals.

ix) Developers who do peer reviews attend secure coding training, including training to help them effectively peer review so every developer knows what to look for.

c) For changes defined as “high risk” (e.g., new areas for the developer, security-sensitive modules such as authentication), changes must also be reviewed by the designated subject matter expert before production deployment.

   i) The change control procedures published in the team wiki used by developers and the CAB describe the areas of the code and environment that are “high risk”.

   ii) Only CAB members have access to make changes to these guidelines in the team wiki.

d) Automated security testing of the code and environment is performed as part of the deployment pipeline, as per CS2.e.

e) All production deployments must have a JIRA ticket number. Deployers must input the JIRA ticket number into the Jenkins build pipeline system for code to be deployed into production.

   i) Jenkins uses the JIRA plugin to pull information from JIRA to include with the build information and push information about the build into the JIRA ticket.

   ii) On weekly basis, Development and IT Operations management review a sampling of the deployments to ensure the correct information about each deployment is included in Jenkins and JIRA.

f) All production deployments are logged and published through information radiators (e.g., large screens on the wall showing the health of the deployment pipeline; weekly check-up reports; engineering reports, executive status updates on monthly reviews). Jenkins records all deployments, as well as all corresponding JIRA tickets and the results of all automated and manual tests, release notes, service incidents, peer reviews, and signoffs.

   i) Deployments require string authentication for Jenkins to validate the person deploying is authorized to do so and each deployment is traced back to the individual performing the deployment.

2) CS2. All code is appropriately validated (as defined by the following controls) throughout the code development and test process (i.e., prior to production deployment) to ensure the service runs in production as designed and service interruptions can be fixed quickly.

   a) The developers have a team norm that each developer writes the unit tests before writing the code (e.g., test-driven development). This norm is documented in the team wiki pages.

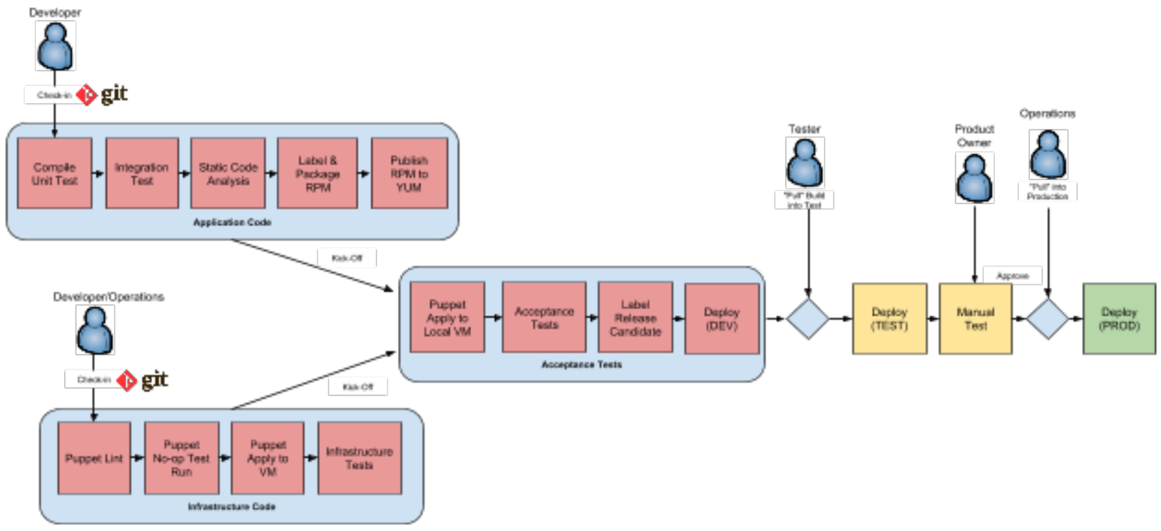
   b) All unit, integration, and acceptance tests must pass successfully before a feature is marked complete by the Product Owner. All automated tests and manual test procedures are checked into version control, alongside the feature. All test checkin requirements are listed on the team wiki page.

   c) All automated tests are performed when any new code is checked into source control. Jenkins runs all automated tests (e.g., unit, integration, acceptance) in the acceptance test environment on every code commit.

      i) When an automated test fails at any stage in the deployment pipeline, Jenkins notifies all developers on the team and the team manager.

      ii) Test results are displayed on information radiators.

d) Jenkins is configured to prevent builds with failing tests from being deployed into the acceptance test server or any other environment later in the pipeline (e.g., UAT, staging, production).



e) Automated security testing is performed as part of the main deployment pipeline.
  i) Jenkins runs static code analysis and a suite of automated security tests to identify any security vulnerabilities and errors potentially compromising the integrity and stability of the system. Any detected issues are classified as either errors or warnings.
  ii) If any errors are detected, the build fails, Jenkins notifies all developers on the team and the team manager, and no deployments are allowed into production until the error is resolved.
  iii) The automated security test suite includes tests covering the OWASP Top Ten.
  iv) Automated security testing is one of the documented enterprise security standards maintained by the CISO.

f) Production code can be deployed quickly into production, as defined by the SLA objectives documented in the team wiki. Specific SLA objectives are defined for the deployment pipeline, such as maximum time to complete automated testing, complete deployment, etc.
  i) When certain operations exceed the SLA objectives, Development and IT Operations management is notified and corrective actions are added to the technical debt backlog.

g) Development and IT Operations require that every documented feature and element of the environment (e.g., database, operating system, network) generate production telemetry (e.g., transaction counts, network usage, CPU utilization) so Development and IT Operations staff have evidence the application is functioning as designed and facilitate fast recovery (i.e., within the documented SLA).
  i) Presence of automated production telemetry is verified during the code review process.

h) When failures occur in production, Development and IT Operations determine whether an automated test could prevent future failures or if more production telemetry could be written to enable faster detection and recovery.
  i) When a failure occurs, a ServiceNow incident is created, and IT Operations is notified and begins investigating the incident within 30 minutes.

    - ii) If resolution requires changes (e.g., production, configurations, tests, code, etc.), a JIRA ticket is opened, and associated with the ServiceNow incident.
    - iii) The JIRA ticket is assigned to a developer to create new automated tests or add more production telemetry to prevent or detect the failure in the future.
    - iv) On a weekly basis, Development and IT Operations management reviews the JIRA tickets created from ServiceNow incidents to ensure required actions were performed in a timely manner.
  - i) The deployment is marked as "complete" in JIRA by the developer following the successful completion of the suite of automated smoke tests run against production servers and production telemetry confirms features are operating as designed.
  - j) When code deployments result in production failures, developers can troubleshoot using production telemetry (e.g., splunk, logs) and log onto production systems. All changes to code and environments are made through the deployment pipeline, which are then pushed to production (i.e., fixing forward), or rolled back to a previous known good state through the deployment pipeline.
    - i) Developers have read-only access to production systems without time restrictions.
    - ii) Production access logs are reviewed by IT Operations management weekly, including a report correlating developer access and code deployments.
    - iii) Every week, a blameless post-mortem is conducted where Development and IT Operations review production issues. Lessons learned are documented in the team wiki and action items are assigned in JIRA or added to the technical debt backlog.

3) CS3. Unauthorized access is prevented, detected, and corrected through the regular review of access credentials and system configuration based on the published SLA for each element of the environment (e.g., database, OS, networking, virtualization, storage, continuous integration servers).
  - a) Development, IT Operations, and InfoSec collaborate to protect the boundaries of the environment.
    - i) The network environment is architected and configured so both inbound and outbound network connections (i.e., IP address, service connections) from unauthorized sources of traffic are prevented through either physical or logical partitioning.
    - ii) The network environment is scanned hourly to validate the network devices and segments have been partitioned and configured as designed (e.g., nmap scans to ensure network traffic can't reach certain IP addresses).
    - iii) The network environment is monitored, ensuring all inbound and outbound connections come from pre-defined sources.
    - iv) External consultants conduct annual penetration testing with a report delivered to Development, IT Operations, and InfoSec management.
  - b) Development, IT Operations, and InfoSec collaborate to protect elements inside the environment.
    - i) All elements are configured according to defined security baselines to prevent unauthorized access.
      - (1) Puppet/Chef scripts have been created based on industry best practices to create known good builds, which are reviewed, updated and approved by InfoSec quarterly.

(2) All Puppet/Chef scripts have been checked into the GitHub Enterprise source code repository, and all changes are reviewed by InfoSec prior to production deployment (as a GitHub issue).

(3) When changes that could introduce security risks are detected either through manual review or automated tests, a JIRA ticket is created.

ii) All environments are created through the deployment pipeline, generating either an environment or a bootable machine image file.

(1) As part of the deployment pipeline, automated testing tools perform validations to ensure configurations and the resulting environments are secure.

(2) Exceptions created by the automated testing tools create a JIRA ticket.

(3) Only images that pass the test suite are made available for use in the production environment.

iii) Production elements are monitored to ensure configuration settings relied upon to prevent unauthorized access have not changed.

(1) To ensure running production elements remain in a secure state, all configurations are compared to known trusted builds to detect changes (using OSSEC, Tripwire, and/or docker diff).

(2) Following production deployment, the environment is monitored for configuration changes and anomalous behavior.

iv) Production elements are monitored to detect unauthorized access and anomalous behavior.

(1) When unauthorized access or anomalous behavior is detected, a ServiceNow incident is created, and IT Operations is notified and begins investigating the incident within 30 minutes.

(2) If resolution requires changes (e.g., production, configurations, tests, code), a JIRA ticket is opened, and associated with the ServiceNow incident.

c) On a weekly basis, JIRA tickets created from ServiceNow (which were created in CS3.b.iv) incidents are reviewed to ensure required actions were performed in a timely manner.

d) Development, IT Operations and InfoSec meet monthly to review and prioritize all open JIRA tickets, perform retrospectives on JIRA tickets that have been closed, and add new preventive efforts to the technical debt backlog, as needed.

Audit Testing and Evidence

To demonstrate the controls are effective and operating as designed to mitigate the risks, the auditors ask for the following evidence.

1) CS1 Evidence.

a) PU's static code analysis tool ruleset (CS1.a.i).

b) Change history for the last five changes made to the static code analysis tool ruleset (CS1.a.iii) and review against CAB membership.

c) Report of build statistics for the last six months showing the number of broken builds caused by static code analysis rule violations (CS1.a.ii).

d) PU's documented coding standards (CS1.b.i).

e) Report of the first ten code reviews in Stash and matching JIRA promotion tickets each month for the last six months (CS1.b.v).

f) JIRA report showing all tasks added to the backlog as a result of code reviews in the last six months, including the task status (e.g., Backlog, Planned, In Progress, Completed) and release tag for completed tasks (CS1.b.vi).

   g) Report from the technical debt backlog showing which items were added as a result of code reviews in the last six months, including the item status (e.g., Backlog, Planned, In Progress, Completed) and release tag for completed items (CS1.b.vii).
   h) Report of peer review statistics over the last six months (CS1.b.viii).
   i) Email "read" receipt by members of Development and IT Operations management for the monthly report of peer review statistics from Stash for the last six months (CS1.b.viii).
   j) Peer review training curriculum.
      i) Verify coding standards, peer review practices, and tools (e.g., JIRA, Stash) are covered (CS1.b.ix).
      ii) Training curriculum and practices include automated tests (CS1.b.ii).
      iii) Training curriculum and practices include production telemetry (CS2.g.i).
   k) Attendance log showing developers attended the peer review training (CS1.b.ix).
   l) Conformance to "high risk" designation based on documented peer review guidelines (CS1.c).
      i) Subject matter expert's code review comments for "high risk" changes.
      ii) PU's change control procedure "high risk" guidelines (CS1.c.i).
      iii) Change history for the last five changes made to the "high risk" guidelines in the team wiki (CS1.c.ii).
   m) Report of every fifth production deployment logged in Jenkins over the last six months (CS1.f).
      i) Verification of appropriate authorization for deployer (CS1.f).
      ii) Verification of correct JIRA ticket number (CS1.e.i).
      iii) Verification of necessary deployment documentation (e.g., test results, release notes, service incidents, peer reviews) (CS1.f).
   n) Email "read" receipt by members of Development and IT Operations management for the weekly report of deployments in Jenkins for the second week of each month for the last six months (CS1.e.ii).

2) CS2 Evidence.
   a) Team norms wiki page (CS2.a).
   b) Test check-in requirements wiki page (CS2.b).
   c) Source code, automated tests, and manual test procedures in source control for the first five features deployed to production each month for the last six months (CS2.b).
   d) Report of the first ten code checkins and matching test results in Jenkins for each environment (e.g., UAT, staging, production) each month for the last six months (CS2.c).
   e) Report of cycle time needed to fix build for the last ten build failures (CS2.c).
   f) Comparison of latest test results in Jenkins with information radiators (CS2.c.ii).
   g) Build logs for each environment (e.g., UAT, staging, production) over the last six months (CS2.d).
   h) Report of every fifth Jenkins run of the automated test suite including any errors and warnings (CS2.e.i).

i) Report of the build status for each automated security test suite run in which there was an error (CS2.e.ii).
j) Mapping of the OWASP Top Ten to the automated security test suite (CS2.e.iii).
k) CISO enterprise security standards document (CS2.e.iv).
l) Deployment pipeline SLA objectives document (CS2.f).
m) Report of Jenkins tasks that did not meet SLA objectives for the last six months and the associated corrective actions in the technical debt backlog (CS2.f.i).
n) Mapping of the production telemetry for the first five features deployed to production each month for the last six months (CS2.g).
o) Report of first five JIRA tickets created from a ServiceNow incident each month for the last six months (CS2.h.ii).
    i) Code changes associated with each of those JIRA tickets (CS2.h.iii).
p) Email "read" receipt by members of Development and IT Operations management for the weekly report of JIRA tickets created from ServiceNow incidents for the second week of each month for the last six months (CS2.h.iv).
q) Automated smoke test results for every fifth production deployment marked "complete" in JIRA over the last six months (CS2.i).
r) Puppet/Chef scripts in source control establishing developer access to production environment with change history for the last six months (CS2.j.i).
    i) Production access logs for the second week of each month compared with Puppet/Chef scripts to verify no unauthorized access occurred (CS2.j.ii).
s) Email "read" receipt by members of IT Operations management for the weekly production access logs for the second week of each month for the last six months (CS2.j.ii).
t) Wiki page for the blameless post-mortem for the second week of each month for the last six months (CS2.j.iii).
u) Report of JIRA tickets created based on blameless post-mortems for the last six months (CS2.j.iii).
v) Report of items added to the technical debt backlog based on blameless post-mortems for the last six months (CS2.j.iii).

3) CS3 Evidence.
    a) Puppet/Chef scripts checked into source control with change history for the last six months (CS3.a.i, CS3.b.i.1).
    b) Network scan results for the third and thirteenth hours every day for the last 30 days (CS3.a.ii).
    c) List of authorized sources of network traffic (CS3.a.iii).
    d) Report of all sources of network traffic over the last 30 days compared with the list of authorized sources (CS3.a.iii).
    e) Report from external consultants for the last two years (CS3.a.iv).
    f) Report of JIRA tickets linked to recommendations from external consultants (CS3.a.iv).
    g) Change history for Puppet/Chef scripts to create known good builds and corresponding InfoSec approval for last two quarters (CS3.b.i.1, CS3.b.i.2).

h) Report of JIRA tickets created from InfoSec review of Puppet/Chef scripts (CS3.b.i.3).
i) Automated test suite to validate configuration and security of environment (CS3.b.ii.1).
    i) Test results for every fifth deployment over the last six months (CS3.b.ii.1).
    ii) Report of JIRA tickets create based on exceptions from automated test suite (CS3.b.ii.2).
    iii) Change history for images available in production (CS3.b.ii.3).
j) Logs of monitoring tools for production elements (CS3.b.iii, CS3.b.iv).
    i) Report of unauthorized configuration changes over the last six months (CS3.b.iii.2).
    ii) Report of unauthorized access events over the last six months (CS3.b.iv).
k) Report of ServiceNow incidents created for unauthorized access over the last six months (CS3.b.iv.1).
l) Report of JIRA tickets created for unauthorized access over the last six months (CS3.b.iv.2).
m) Report of JIRA tickets created from ServiceNow incidents for the last six months (CS3.c).
n) Email "read" receipt by members of Development and IT Operations management for the weekly ServiceNow incident-related JIRA tickets for the second week of each month for the last six months (CS3.c).
o) Wiki page for the monthly JIRA review meeting with Development, IT Operations, and InfoSec management each month for the last six months (CS3.d).
    i) Report of items added to the technical debt backlog based on monthly JIRA review meetings for the last six months.

4) General.
    a) For all tools in use upon which controls are dependent, demonstrate the appropriate people are trained to use them and the tools are configured correctly.
        i) Training curricula.
        ii) Training attendance logs.
        iii) Configuration settings stored in source control.
        iv) Change history for configuration settings.

6 Conclusion

The DevOps community has articulated the need for further guidance on how to help IT and audit work better together. To address this need, the DevOps Audit Defense Toolkit provides information for how to bridge the gap between organizations adopting DevOps practices and their auditors who have the responsibility of assessing whether the organization is mitigating risk adequately. The Toolkit is intended to create a common perspective and shared understanding between organizations and auditors to reduce the pain and increase the effectiveness of audits, ultimately helping the business create more value and win in the marketplace.

We hope to expand on this toolkit in the future as more organizations adopt DevOps practices and we learn more about effective controls and how to prove they are, in fact, effective.

7 Acknowledgements

The authors would like to thank the following individuals for contributing their expertise and time to this Toolkit. Their invaluable review and feedback made this Toolkit clearer, more complete, more accurate, and more useful.

- Jim Bird
- Mike Kavis
- Norman Marks
- Daniel Matthis
- Gareth Rushgrove
- Simon Storm