Centro Universitário de Brasília
Instituto CEUB de Pesquisa e Desenvolvimento - ICPD

VICTOR NEVES EVANGELISTA

IMPLANTAÇÃO DE CICLO DE DESENVOLVIMENTO SEGURO PARA PCI-DSS

Brasília
2017

VICTOR NEVES EVANGELISTA

IMPLANTAÇÃO DE CICLO DE DESENVOLVIMENTO SEGURO PARA PCI-DSS

Projeto de trabalho acadêmico apresentado ao Centro Universitário de Brasília (UniCEUB/ICPD) como pré- requisito para a obtenção de Certificado de Conclusão de Curso de Pós-graduação *Latu Sensu,* do curso Redes com ênfase em segurança da informação.

Orientador: Prof. José Eduardo Malta de Sá Brandão

Brasília, 23 de fevereiro de 2017

Banca examinadora

_____

Prof. Robertson Schitcoski

_____

Prof. Tânia Cristina da Silva Cruz

Brasília
2017

RESUMO

Este trabalho tem por objetivo fornecer uma visão das atividades mínimas requeridas para construir e manter aplicações que forneçam segurança a seus usuários, frente ao fato de que a interação da sociedade com a internet vem se fazendo presente de forma cada vez mais intensa, surgindo também a preocupação com a confidencialidade, integridade e disponibilidade de serviços considerados importantes como, por exemplo, serviços ligados às atividades financeiras fornecida aos usuários. Modelos de desenvolvimento seguro foram estudados e pensados em como atender aos requisitos de segurança de aplicação do PCI Security Standards Council. Durante o trabalho foram apresentadas atividades mínimas para o cumprimento de tais requisitos.

Palavras-chave: Segurança da informação. Desenvolvimento seguro. Segurança de aplicações.

ABSTRACT

This research claims provide a vision about minimal activities required to construct and maintain applications who deliver security to their users, considering that the society interaction with internet is becoming more present and intense, arising the concern about confidentiality, integrity and disponibility of services measured importants, as those related to financial activities provided to users. Models of secure development was studied and analyzed on how to meet the requirements of secure application of PCI Security Standards Council. For the project was explained the minimum activities to the fulfillment of such requirements.

Key words: Information security. Secure development. Application security

Índice de figuras

Índice de tabelas

Sumário

Introdução

Recentemente, principalmente após a expansão da conectividade da computação móvel e com a guarda de informações na computação em nuvem, a falta de conciência pela cultura de segurança vem trazendo à luz um sério problema que afeta indivíduos, organizações, e por outrora, governos: a segurança de software.

Frequentemente, é visto nos noticiários uma reportagem envolvendo vazamento de informações, ou indisponibilidade de sistema, ou comprometimento de confidencialidade, quando nem sempre o problema reside apenas na segurança de infraestrutura, mas em uma aplicação, ou serviço, que foi desenvolvido e disponibilizado ao público.

Em 2014, “68% das vulnerabilidades web eram consideradas críticas, uma alta de 6% se comparado com 2013” (Positive Tecnologies Security, 2015). Sendo que, das vulnerabilidades encontradas, 89% estavam relacionadas à código e 11% à má configuração de infraestrutura.

Mesmo algumas vulnerabilidades tendo apresentado queda, como XSS[1] que obteve queda de 8% ficando com 70% em 2014 nos softwares analisados; e força bruta tendo queda de 29%, ficando com 40% de incidência nos softwares. Outras vulnerabilidades bastante conhecidas obtiveram alta quando comparadas com 2013, como o caso de *SQL injection*[2], que obteve alta de 5%, ficando com 48% das vulnerabilidades; e *URL redirect*[3], que obteve alta de 15%, ficando com 33% das vulnerabilidades.

---

1 “A aplicação inclui dados em um conteúdo dinâmico para enviar ao usuário *Web* sem validação apropriada com conteúdo malicioso” (CHESS ; WEST, 2007 , p. 304, tradução nossa)
2 https://www.owasp.org/index.php/SQL_Injection
3 https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet

O desafio de criar, e manter, uma estrutura de processos e verificações de segurança de software que atendam aos requisitos de segurança vem, timidamente, ganhando notoriedade. Ainda muito pouco conhecidos, já existem modelos de maturidade e modelos de ciclo ágil de desenvolvimento seguro de software, cujo objetivo é mitigar possíveis riscos que possam vir a comprometer os pilares da segurança da informação.

Uma das área onde a segurança da informação atua é em softwares de aplicações financeiras. Segundo WhiteHat Security[4], 17% das aplicações financeiras experimentaram algum tipo de brecha em 2015, compromentendo de algum forma os pilares da seguança da informação, e causando, eventualmente, prejuízos financeiros aos clientes ou/e às instituições financeiras.

Dentre modelos e padrões, o que se destina à transações eletrônicas no ramo financeiro é o Padrão de Segurança de Dados da Indústria de Cartões de Pagamento (PCI DSS). O modelo “foi desenvolvido para incentivar e aprimorar a segurança dos dados do portador do cartão e promover a ampla adoção de medidas de segurança de dados consistentes no mundo todo” (PCI Security Standards Council, 2013) .

Porém, para que os padrões de segurança de dados definidos no PCI DSS sejam atendidos é necessário planejar como os requisitos serão alcançados com sucesso, necessitando assim de modelos de desenvolvimento que visam atender tais atividades.

Alguns modelos de desenvolvimento seguro já existem, uns são focados unicamente no desenvolvimento, como Secure Software Foundation[5] (HEMEL;VRIES, 2014). Outros, como o OpenSAMM[6], são mais flexíveis, podendo permitir um enfoque ou em governanca, ou em construção ou em infraestrutura. Devido à sua grande

4 Https://info.whitehatsec.com/rs/whitehatsecurity/images/2015-Stats-Report.pdf
5 https://www.securesoftwarefoundation.org/index.php/framework-secure-software/
6 http://www.opensamm.org/

flexibilidade, para este trabalho, foi escolhido o OpenSAMM, escrito e mantido pela OWASP (Open Web Application Security Project)[7].

Problema

O problema consiste em definir quais atividades mínimas são necessárias para construir um modelo de desenvolvimento seguro que atenda a critérios de segurança de software do PCI-DSS.

Justificativa

Acerca do motivo de implantação de um ciclo de desenvolvimento seguro de software o Departamento de Segurança da Informação e Comunicações do Gabinete de Segurança Institucional da Presidência da República (2012, p.2) afirma:

> As organizações modernas cada vez mais se veem dependentes de tecnologias que possam fazer suas informações circularem rapidamente, de forma a atenderem as necessidades negociais das quais se encontram atreladas. Paralelamente ao desenvolvimento e emprego de novas tecnologias, essas mesmas organizações vêm sofrendo ataques ao seu acervo de informações, cuja frequência é cada vez maior e com uso mais aprimorado de recursos computacionais.

Objetivos

Objetivo geral

Propor através de atividades necessárias e de forma sistematizada um modelo focado em segurança de software em ambiente de empresa que lida com mercado

7 https://www.owasp.org/index.php/Main_Page

financeiro.

Objetivo específico

Apresentar as fases e suas atividades para o desenvolvimento de software baseado em considerações, requisitos, conhecimento, arquitetura e codificação segura de software para atender aos requisitos do PCI-DSS relacionados unicamente à segurança de aplicação.

Neste trabalho, serão apresentadas as atividades requeridas para atender aos requisitos de desenvolvimento seguro, tendo por base as atividades do OpenSAMM.

Organização do texto

O texto é organizado em entendimento de conceitos acerca de segurança da informação, segurança de software, ciclo de desenvolvimento seguro e sistemas financeiros. Uma explicação sobre PCI DSS é abordada pois este será o alvo da proposta; Após essa explicação, é apresentada a proposta deste trabalho, que aborda o ciclo de desenvolvimento seguro para atender ao PCI DSS. Ao final, um breve estudo de caso será apresentado, mostrando que a implantação de um ciclo que verifique a qualidade de software surtiu efeitos esperados, diminuindo a quantidade de ocorrências críticas e altamaente críticas em um software.

1 CONCEITOS

1.1 Conceitos de segurança

“Segurança da informação é a prática de assegurar que os recursos que geram, armazenam ou proliferam as informações sejam pretegidos contra quebra da confidencialidade, comprometimento da integridade e contra a indisponibilidade de acesso a tais recursos.” (DIÓGENES; MAUSER, 2013)

Entende-se por confidencialidade a “prevenção de vazamento da informação” (DIÓGENES; MAUSER, 2013) ; por integridade, “presenvação/manutenção do dado na sua forma íntegra” (DIÓGENES; MAUSER, 2013); e disponibilidade a disponibilização da informação a quem lhe é devido, obedecendo às regras da confidencialidade previamente definidas (DIÓGENES; MAUSER, 2013) .

Quando um ou mais dos conceitos citados acima são comprometidos, temos o que se pode chamar de vulnerabilidade, “vulnerabilidade é uma fraqueza que pode ser explorada no sistema” (HOLANDA; FERNANDES, 2009) para fins que causem dano a sistemas e pessoas.

1.2 Segurança de software

Um processo de software é um conjunto de atividades e resultados associados, desenvolvidos ciclicamente (como qualquer processo) em uma organização produtora de software, e que busca gerar um software com qualidade ou atributos esperados. (HOLANDA ; FERNANDES , 2009)

Para a construção de um software seguro, é necessário entender que: quanto

melhores desenvolvidos os processos de construção, menor a probabilidade de falhas críticas ocorrerem.

Esses processos vão desde a conscientização até testes automatizados durante o processo de verificação de segurança e qualidade. Se esses processos falham, vários problemas podem ser gerados, dando espaços para vulnerabilidades de XSS, por exemplo.

Segundo Maristela Holanda e Jorge Fernandes (2009, p.10), “um dos principais fatores causadores dessas vulnerabilidades [a exemplo de XSS citado] é a codificação ingênua [achar que o usuário nunca terá ações maliciosas] do software por um programador”, o que nos leva a entender que o primeiro passo para a construção de um software seguro é a educação e orientação.

Após a educação e orientação, seguem processos de desenvovlimento da arquitetura e a construção do software, que tem sua segurança planejada, desenvolvida, testada e os resultados são armazenados em uma base de conhecimento, para que sirva de histórico e insumos para análises futuras.

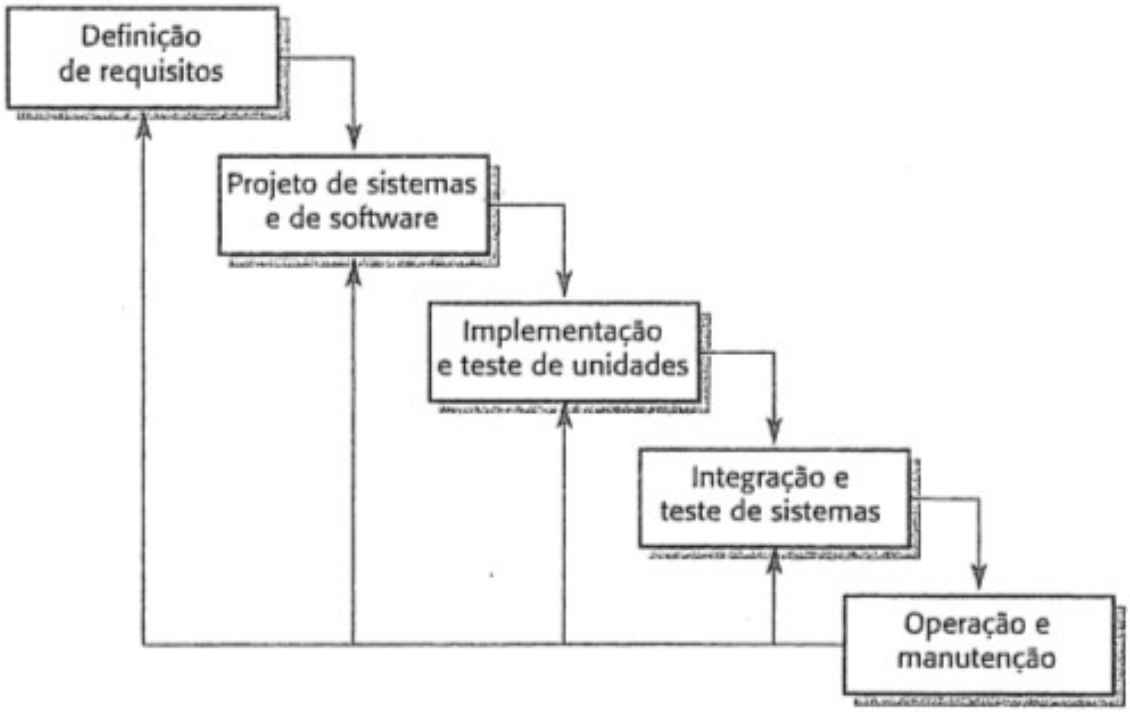
Toda essa cultura em segurança de software resulta do fato de que “informação talvez seja o melhor ativo de uma organização” (BHARGAV; KUMAR, 2011), proteger a informação, ou proteger o caminho pelo qual a informação é tratada e enviada, é a melhor forma de evitar perdas futuras que podem ser desastrosas a uma organização.

1.3 Ciclo de desenvolvimento de software

O ciclo de desenvolvimento seguro obedece aos processos e fases padrões de construção de software, como obter uma equipe capacitada, definir escopo e requisitos do sistema, iniciar processo de construção, testes , implantação e manutenção.

O primeiro modelo de desenvolvimento foi definido por Winston Royce em 1970 (apud SOMMERVILLE, 2003, p. 37), conhecido como *modelo cascata*. Esse modelo mostra os principais estágios de desenvolvimento de software, embora esse modelo não seja adotado na proposta, serve como entendimento básico sobre ciclo de desenvolvimento, conforme figura 1.

Figura 1: Desenvolvimento de software em modelo cascata (apud SOMMERVILLE, 2003, p.17)



O modelo em castata possui as seguintes etapas:

1. *Definição de requisitos:* “As funções, as restrições e os objetivos do sistema são estabelecidos por meio da consulta aos usuários do sistema”;

2. *Projeto de sistemas e de software*: “Estabelece uma arquitetura do sistema geral”;

3. *Implementação e teste de unidades:* “Durante esse estágio, o projeto de software é compreendido como um conjunto de programas ou unidades de programas”;

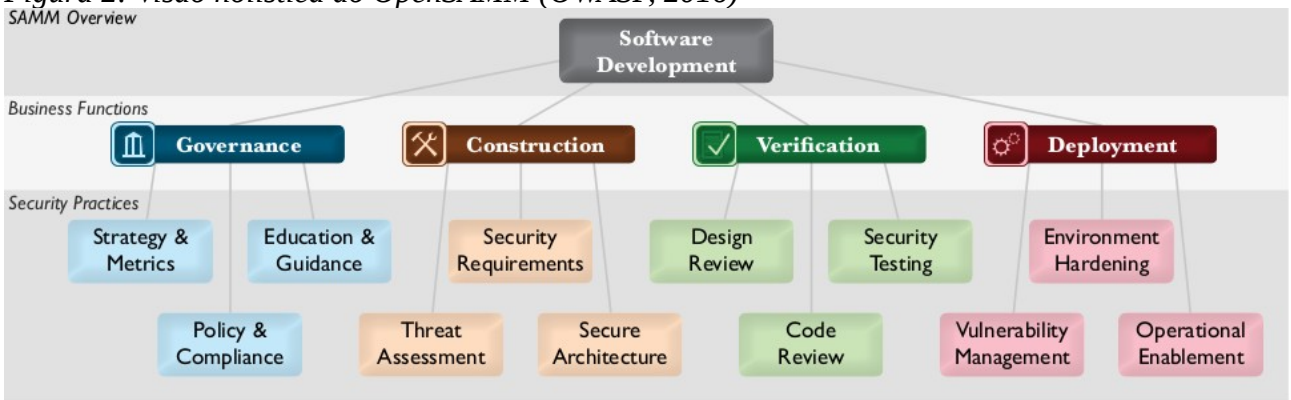
4. *Integração e testes de sistemas*: “As unidades de programas ou programas individuais são integrados e testados como um sistema completo a fim de garantir que os requisitos de software foram atendidos”;

5. *Operação e manutenção*: “O sistema é instalado e colocado em produção. A manutenção envolve corrigir erros que não foram descobertos em estágios anteriores do ciclo de vida”.

Ao final da última etapa, os resultados de saída servem como dados de entrada para a etapa de requisitos do próximo estágio de atividades, retroalimentando o fluxo, esse modelo se encaixa nas etapas onde há verificação de código e eventuais correções.

Dentro das fases do ciclo deve-se obedecer a certas atividades que tem por objetivo garantir a qualidade e segurança do software. O modelo utilizado neste trabalho foi o OpenSAMM (OWASP, 2016), devido à sua grande flexibilidade. O OpenSAMM é escrito e mantido pela OWASP, uma fundação internacional de segurança de applicações, sem fins lucrativos, que tem como objetivo “fazer segurança de aplicação se tornar visível, então as pessoas e organizações podem tomar decisões acerca dos riscos de segurança” (OWASP, 2016, p.2, tradução nossa). O modelo é construído a partir de 4 “funções de negócio”, e cada função de negócio possui 3 “práticas seguras”, como pode ser visto na figura 2.

Figura 2: Visão holística do OpenSAMM (OWASP, 2016)



Onde:

- Governança (*Governance*) “é centrado nos processos e atividades relacionados em como a organização gerencia as atividades de desenvolvimento de software” (OWASP, 2016, p.8, tradução nossa);
- Construção (*Construction*) “diz respeito aos processos e atividades relacionados em como a organização define objetivos e cria softwares dentro de projetos de desenvolvimento” (OWASP, 2016, p.8, tradução nossa);
- Verificação (*Verification*) “é focado nos processos e atividades relacionadaos em como a organização verifica e testa os artefatos produzidos durante o processo de desenvolvimento” (OWASP, 2016, p.8, tradução nossa);
- Implantação (*Deployment*) “implica em processos e atividades relacionados em como a organização gerencia *releases* dos softwares que foram criados” (OWASP, 2016, p.8, tradução nossa).

Graças à sua flexibilidade, não há necessidade de aplicar o modelo por completo, isso nos permite focar em determinadas *funções de negócios* ou *práticas seguras* necessárias para atender aos requisitos do PCI DSS. A funções de negócio que mais serão utilizadas serão construção e verificação, permeando algumas poucas e específicas práticas de segurança das outras funções de negócio.

1.4 Sistemas financeiros

Sistemas financeiros são serviços de: transferências entre contas; pagamento de débitos; pagamento de pessoa a pessoa e depósito remoto. Sistemas financeiros também oferecem serviços adicionais como renegociação de dívidas ; oferer balanços financeiros, cartões de crédito e manter comunicação com a entidade financeira máxima

de cada país.

“Os agentes financeiros disponibilizam produtos e serviços financeiros com condições específicas de acordo com a modalidade solicitada e característica do cliente” (Sebrae, 2014). São alguns tipos de serviços financeiros, de acordo com Sebrae:

- Custódia de cheques;
- Conta corrente;
- Pagamento eletrônico de salário;
- Recebimento com cartões de pagamento;
- Entre diversos outros.

Este trabalho destina-se à aplicações que são disponibilizados via serviço web para usuários finais, essas aplicações podem ou não conter um ou mais dos serviços financeiros citados acima, por exemplo , um sistema de *internet banking.*

2. O PCI-DSS

O PCI-DSS (Payment Card Industry Data Security Standards) “se aplica a todas as entidades envolvidas nos processos de pagamento do cartão, inclusive comerciantes, processadores e prestadores de serviço, bem como todas as entidades que armazenam, processam ou transmitem os dados do portador do cartão e/ou dados de autenticação confidenciais.” (PCI Security Standards Council, 2013 , p. 5)

O padrão compreende um conjunto mínimo de requisitos para proteger os dados do portador do cartão e pode ser aperfeiçoado por controles e práticas adicionais para amenizar ainda mais os riscos, bem como as normas e leis locais, regionais e do setor.

Os requisitos de segurança do PCI DSS se aplicam a todos os componentes do sistema que estejam incluídos ou conectados no ambiente dos dados do portador do cartão. Somando um total de 12 requisitos, para cada requisito há diversos *“Requisitos do PCI DSS”, “Procedimentos de teste” e “Orientação”.*

- *Requisitos do PCI DSS* - Define os requisitos do padrão de segurança dos dados; a conformidade do PCI DSS é validada de acordo com esses requisitos;
- Procedimentos de teste - Exibe os processos a serem seguidos pelo avaliador para validar se os requisitos do PCI DSS têm sido atendidos e se estão "vigentes";
- Orientação - Descreve a intenção ou o objetivo de segurança por trás de cada um dos requisitos do PCI DSS. Esta coluna contém apenas orientação e tem o objetivo de auxiliar a compreender o porquê de cada requisito. A orientação nesta coluna não substitui ou expande os Requisitos do PCI DSS e Procedimentos de teste.

É importante lembrar que “o PCI DSS não substitui as leis locais ou regionais,

Figura 3: Visão geral dos requisitos PCI DSS

Padrão de segurança de dados do PCI – Visão geral alto nível

Construir e manter a segurança de rede e sistemas	1. Instalar e manter uma configuração de firewall para proteger os dados do titular do cartão 2. Não usar padrões disponibilizados pelo fornecedor para senhas do sistema e outros parâmetros de segurança
Proteger os dados do titular do cartão	3. Proteger os dados armazenados do titular do cartão 4. Criptografar a transmissão dos dados do titular do cartão em redes abertas e públicas
Manter um programa de gerenciamento de vulnerabilidades	5. Usar e atualizar regularmente o software ou programas antivírus 6. Desenvolver e manter sistemas e aplicativos seguros
Implementar medidas rigorosas de controle de acesso	7. Restringir o acesso aos dados do titular do cartão de acordo com a necessidade de conhecimento para o negócio 8. Identificar e autenticar o acesso aos componentes do sistema 9. Restringir o acesso físico aos dados do titular do cartão
Monitorar e testar as redes regularmente	10. Acompanhar e monitorar todos os acessos com relação aos recursos da rede e aos dados do titular do cartão 11. Testar regularmente os sistemas e processos de segurança.
Manter uma política de segurança de informações	12. Manter uma política que aborde a segurança das informações para todas as equipes.

Fonte: PCI Security Standards Council, 2013 , p. 5

Ter uma melhor noção sobre do que se tratam os requisitos do PCI é imprescindível para compreender com quais e quantos são trabalhados no âmbito de desenvolvimento. A seguir, será apresentada uma visão geral de alto nível sobre os 12 requisitos do PCI DSS, conforme ilustrado na Figura 3.

2.1 Requisitos

Cada requisito trata especificamente de uma área de conhecimento pertecente à segurança da informação.

Os requisitos 1 e 2 tratam sobre “construir e manter a segurança de rede de sistemas”; os requisitos 3 e 4 tratam sobre “proteger de dados do titular do cartão”; os requisitos 5 e 6 tratam sobre “manter um programa de gerenciamento de vulnerabilidades”

e aplicações; os requisitos 7, 8 e 9 abordam sobre “implementar medidas rigorosas de controles de acesso”; 10 e 11 tratam sobre “monitorar e testar as redes regularmente”; e o requisito 12 trata sobre “manter uma política de segurança de informações”, como demonstrado na Figura 2.

A seguir, veremos detalhadamente sobre cada um deles.

2.1.1 Requisito 1

Instalar e manter uma configuração de firewall para proteger os dados do portador do cartão

“Firewalls são dispositivos do computador que controlam o tráfego do computador permitido entre a rede de uma empresa (interna) e redes não confiáveis (externa), assim como o tráfego dentro e fora de muitas áreas confidenciais na rede confiável interna de uma empresa” (PCI Security Standards Council, 2013 , p. 19), logo, firewalls e outros componentes que atuem como firewall devem obedecer a um rígido conrole de qualidade, que é abordado no requisito 1.

2.1.2 Requisito 2

Não usar padrões disponibilizados pelo fornecedor para senhas do sistema e outros parâmetros de segurança

Muitos produtos já vem com senhas padrões, ”essas senhas e configurações são bastante conhecidas pelas comunidades de hackers e facilmente determinadas por meio de informações públicas” , para evitar incidentes de segurança, o requisito 2 define orientações sobre como mitigar possíveis vulnerabilidades.

2.1.3 Requisito 3

Proteger os dados armazenados do portador do cartão

Este requisito trata sobre manter “Métodos de proteção como criptografia, truncamento, mascaramento e referenciamento são componentes essenciais da proteção de dados do portador do cartão. Se um invasor burlar outros controles de segurança e obtiver acesso aos dados criptografados, sem as chaves criptográficas adequadas, os dados estarão ilegíveis e inutilizáveis para aquele indivíduo.” (PCI Security Standards Council, 2013)

2.1.4 Requisito 4

Criptografe a transmissão dos dados do portador do cartão em redes abertas e públicas

“As informações confidenciais devem ser criptografadas durante a transmissão nas redes que são facilmente acessadas por indivíduos mal-intencionados. Redes sem fio configuradas de forma incorreta e vulnerabilidades na criptografia herdada e protocolos de autenticação continuam a ser alvos contínuos de indivíduos mal-intencionados que exploram essas vulnerabilidades para obter acesso privilegiado aos ambientes de dados do portador do cartão.” (PCI Security Standards Council, 2013)

2.1.5 Requisito 5

Proteja todos os sistemas contra softwares prejudiciais e atualize regularmente programas ou software de antivírus

Qualquer ambiente operacional está sujeito à softwares maliciosos (vírus,

worms e cavalos de Troia), portanto, um ”software de antivírus deve ser usado em todos os sistemas comumente afetados pelo malware para proteger os sistemas de ameaças atuais e potenciais de softwares mal-intencionados.” (PCI Security Standards Council, 2013)

2.1.6 Requisito 6

Desenvolver e manter sistemas e aplicativos seguros

“Indivíduos inescrupulosos usam as vulnerabilidades da segurança para obter acesso privilegiado aos sistemas” (PCI Security Standards Council, 2013), pensando nisso, é necessário desenvolver e manter um ciclo que garanta que requisitos mínimos de segurança de aplicação serão atendidos.

Para atender aos requisitos de segurança no que tange a sistemas, desconsiderando infraestrutura e políticas de governança, um enfoque é dado ao requisito 6, porém, há outras atividades de outros requisitos que estão de alguma forma ligados ao desenvolvimentos de sofware, estes, também entrarão nas atividades de desenvolvimento seguro.

2.1.7 Requisito 7

Restrinja o acesso aos dados do portador do cartão de acordo com a necessidade de conhecimento para o negócio

“Para assegurar que os dados críticos possam ser acessados somente por uma equipe autorizada, os sistemas e processos devem estar implementados para limitar o acesso com base na necessidade de divulgação e de acordo com as responsabilidades da função.” (PCI Security Standards Council, 2013)

2.1.8 Requisito 8

Identifique e autentique o acesso aos componentes do sistema

“Atribuir uma identificação exclusiva (ID) a cada pessoa com acesso assegura que cada indivíduo seja exclusivamente responsável pelas suas ações. Quando tal responsabilidade estiver em vigor, as ações desempenhadas nos dados e sistemas críticos serão realizadas e podem ser rastreadas, por usuários e processos conhecidos e autorizados.” (PCI Security Standards Council, 2013)

2.1.9 Requisito 9

Restrinja o acesso físico aos dados do portador do cartão

“Qualquer acesso físico aos dados ou sistemas que armazenam dados do portador do cartão fornecem a oportunidade para as pessoas acessarem dispositivos ou dados e removerem sistemas ou cópias impressas e deve ser restrito de forma adequada.” (PCI Security Standards Council, 2013)

2.1.10 Requisito 10

Acompanhe e monitore todos os acessos com relação aos recursos da rede e aos dados do portador do cartão

“A presença de registros em todos os ambientes permite o monitoramento, o alerta e a análise completa quando algo dá errado. Determinar a causa de um comprometimento é muito difícil, se não impossível, sem registros das atividades do sistema.” (PCI Security Standards Council, 2013)

2.1.11 Requisito 11

Testar regularmente os sistemas e processos de segurança.

“Os componentes do sistema, processos e softwares personalizados devem ser testados com frequência para assegurar que os controles de segurança continuem refletindo um ambiente em transformação.” (PCI Security Standards Council, 2013)

2.1.12 Requisito 12

Mantenha uma política que aborde a segurança da informação para todas as equipes.

“Uma política de segurança sólida determina o tom da segurança para toda a empresa e informa aos funcionários o que é esperado deles. Todos os funcionários devem estar cientes da confidencialidade dos dados e de suas responsabilidades para protegê-los.” (PCI Security Standards Council, 2013)

3 PROPOSTA DE CICLO DE DESENVOLVIMENTO SEGURO

A proposta consiste em apresentar as fases e atividades mínimas necessárias para desenvolver e manter um sistema que atenda à características de software seguro levando em consideração as boas práticas recomendadas e os requisitos de segurança do PCI DSS, no que tange a sistemas.

Entende-se por atividades mínimas como as que não podem faltar em hipótese alguma, isto pois, de acordo com a realidade de cada organização e contexto e criticidade de cada sistema, podem haver mais fases, conforme previsto nas *práticas seguras* do OpenSAMM.

O modelo OpenSAMM possui diversas práticas seguras para atender a diversas necessidas de segurança. Para atender as necessidades do desenvolvimento seguro que atenda aos requisitos de segurança de software do PCI DSS foram selecionados algumas dessas práticas seguras, que estão definidas dentro das funções de negócio do OpenSAMM. Caso uma dessas práticas não seja implementada, algumas orientações dos requisitos do PCI DSS podem não ser atendidas.

Seguindo a estrutura do OpenSAMM, dividimos o ciclo de desenvolvimento nas 4 funções de negócio e suas práticas seguras, conforme quadro a seguir.

Tabela 1: Distribuição das atividades de acordo com as fases do OpenSAMM (fonte: próprio autor)

Governance	Construction	Verification	Deployment
• Educação e orientação	• Requisitos de segurança • Segurança em arquitetura	• Verificação e integração contínua	• Implantação e gerenciamento

Após definido quais práticas estão em quais funções de negócio, a implantação

é iniciada começando por *Governança,* depois segue-se a sequência, conforme quadro acima e páticas descritas nas próximas páginas, terminando com o ciclo com a última função, *implantação.*

Os incidentes de segurança ocorridos e maturidade que são adiquiridas (através de manutenções no software) no decorrer do tempo, durante cada uma das fases, pode ser comentada na prática de *Educação e orientação*, para sempre garantir informações atualizadas a quem são ministradas.

3.1 Educação e orientação

Educação e orientação está focado em capacitar e alertar o time de projeto sobre indentificação e mitigação de possíveis riscos relacionados às atividades e requisitos do software.

A maneira mais direta e eficaz é provendo treinamento aos desenvolvedores acerca das vulnerabilidades mais comuns encontradas nos softwares, juntamente com o treinamento, *guidelines*, ou seja, guias de melhores práticas de desenvolvimento são extremamente aconcelháveis. Esses guias podem ser publicados em portais, um lugar de fácil acesso à toda a equipe para consultas das práticas e padrões de desenvolvimento que visam mitigar erros crassos.

3.1.1 Atividades

- Promover treinamento
    - Treinar os desenvolvedores sobre técnicas seguras de codificação, incluindo como evitar vulnerabilidades comuns de codificação e compreendendo como os dados confidenciais são controlados na memória. (PCI-DSS versão 3, requisito

6.5)

- Construir e manter guias de desenvolvimento
    - Publicar guias de segurança em desenvolvimento em um ponto onde todos tenham acesso. Em alguns casos, noçoes sobre segurança de infraestrutura atuarão juntamente com o desenvolvimento.
- Utilizar engenheiros de segurança de desenvolvimento
    - Criar um equipe de desenvolvedores e arquitetos com conhecimento mais aprofundado em segurança para atuar como orientação para as outras equipes de desenvolvimento.

3.2 Requisitos de segurança

Definir requisitos de segurança implica em determinar condições de segurança que o software deve possuir. Está focado em uma visão de alto nível, não possuindo necessidade de abordar detalhes técnicos de implementação.

3.2.1 Atividades

- Considerar segurança explicitamente durante o processo de requisitos de software
    - Considerar e documentar observações importantes sobre segurança, mediante a realidade e contexto ao qual o software será construído.
- Definir matriz de acesso
    - Com base nas funções de negócio, identificar os grupos de irão compor a parte de autorização, sejam eles grupos de usuários, operadores, gestores, entre outros. Este documento controla “o acesso aos componentes do sistema e aos dados do portador do cartão somente àquelas pessoas cuja função requer tal

acesso.” (PCI-DSS versão 3, requisito 7.1)

- Definir uso de transmissões criptografadas
    - De acordo com o quarto requisito do PCI-DSS, “as informações confidenciais devem ser criptografadas durante a transmissão nas redes” (PCI-DSS versão 3, requisito 3), desde os requisitos do sistema já deve ser considerado o uso de HSTS (HTTP Strict Transport Security)[8] para as comunicações.
    - Além da transmissão, a guarda de informações pessoais de clientes devem ser criptografadas. (PCI-DSS versão 3, requisito 3)

3.3 Segurança em arquitetura

“O *desing* do software tem uma larga influência na segurança, diferentes tecnologias carregam diferentes riscos para serem mitigados.” (Secure Software Foundation , 2014)

A prática de pensar em segurança desde a arquitetura do projeto necessita de profissionais com larga experiência tanto em desenvolvimento, quanto um sólido conhecimento em segurança. A ideia nesta fase é, enquanto se define a arquitetura, identificar possíveis brechas que possam se tornar vetores de ataques e proativamente pensar em possíveis soluções de mitigação. Além do conhecimento em segurança, é importante levar em consideração falhas identificadas em outros sistemas produzidos pela empresa.

3.3.1 Atividades

- Inserir considerações sobre segurança durante o processo de definição de

8 RFC 6797

arquitetura

  - Definir práticas, diretrizes, métodos e padrões de segurança a serem implementados durante a fase de construção do projeto.

- Manter lista de software de terceiros que sejam confiáveis e padronizados

  - Durante o processo de desenvolvimento, é comum a utilização de bibliotecas de terceiros, é importante manter um repositório com quais *frameworks* estão disponibilizados para a equipde de desenvolvimento. A criação desse repositório tem em vista a padronização de recursos e suas versões.

- Definir procedimentos e arquiteturas padrões de segurança

  - Conforme PCI-DSS , “Analise os processos de desenvolvimento do software por escrito para verificar se os processos foram baseados nos padrões e/ou nas melhores práticas do setor.” (PCI-DSS versão 3, requisito 6.3.a)

3.4 Verificação e integração contínua

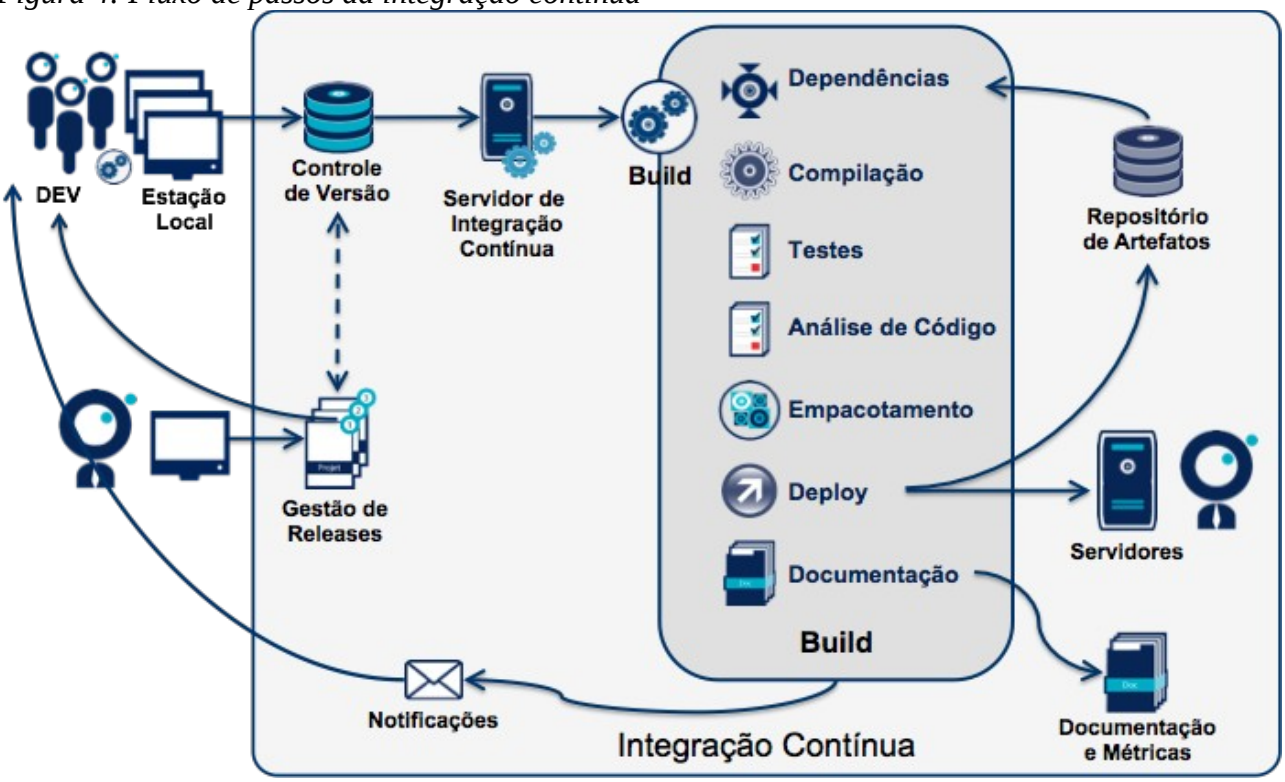
Durante o processo de desenvolvimento, processos de verificações de código e processos internos de disponibilização de versões de um produto podem ser violados ou alterados. Em face disso, muitas empresas adotam a integração contínua, que consistem em automatizar passos antes manuais no processo de desenvolvimento e disponibilização.

É de extrema importância em um processo de desenvolvimento seguro que se “revise o código personalizado antes da liberação para produção ou clientes a fim de identificar qualquer possível vulnerabilidade no código (usando processos manuais ou automatizados)” (PCI-DSS versão 3, requisito 6.3.2). Para isso existe a integração

contínua.

A integração contínua necessita de um gerenciador de integração contínua, do qual é responsável por recuperar o código fonte no repositório de códigos, submeter aos passos da integração e decidir se disponibiliza ou não uma nova *build* do produto. A seguir, temos em detalhes os passos da integração contínua.

Figura 4: Fluxo de passos da integração contínua



Fonte: http://blog.octo.com/pt-br/maturidade-da-integracao-continua/

Temos então os seguintes processos:

- O *desenvolvimento* do código, aqui, já aplicamos as atividades anteriores de *Educação e orientação, requisitos e arquitetura de segurança.* O código é , obrigatoriamente, versionado em um software de controle de versões, que pode ser gerenciado por um processo de *gestão de releases*;
- Quando a integração contínua é acionada, o gerenciador recuperar o código fonte e suas dependências e gera uma compilação;

- É inicializado a verificação com ferramentas de análise de segurança de código, pode ser SAST (Static Application Security Testing), “programa que verifica código fonte em busca de padrões de vulnerabilidade” (TECHBEACON, 2016), pode ser DAST (Dynamic Application Security Testing), “conhecida como 'teste de caixa preta', análise dinâmica de teste em busca de vários tipos de vulnerabilidades em tempo de execução da aplicação”(TECHBEACON, 2016), ou ambos;
- Um outro processo de verificação de código baseia-se nos testes de qualidade, nesse passo são analisados duplicações de código, complexidade, codificações e sintaxes padrões da linguagem. “As revisões de código garantem que o código seja desenvolvido de acordo com as diretrizes de codificação seguras.” (PCI-DSS versão 3, requisito 6.3.2);
- Baseado nos resultados dos testes de segurança e qualidade o gerenciador decide, mediante regras pré definidas, se gera ou não um novo pacote do produto. Caso tenha obtido sucesso, o novo pacote é versionado e disponibilizado para ambiente de produção, estatísticas dos testes são armazenados e a equipe notificada do processo. Caso não haja sucesso nas métricas, a compilação usada nos testes é descartada e a equipe notificada.

3.4.1 Ciclo de revisão e análise estática

O ciclo de revisão estática de código (SAST) consiste em uma etapa da integração contínua. Dentro do universo da integração contínua, durante a fase de revisão de *análise de código*, existem outras 4 etapas:

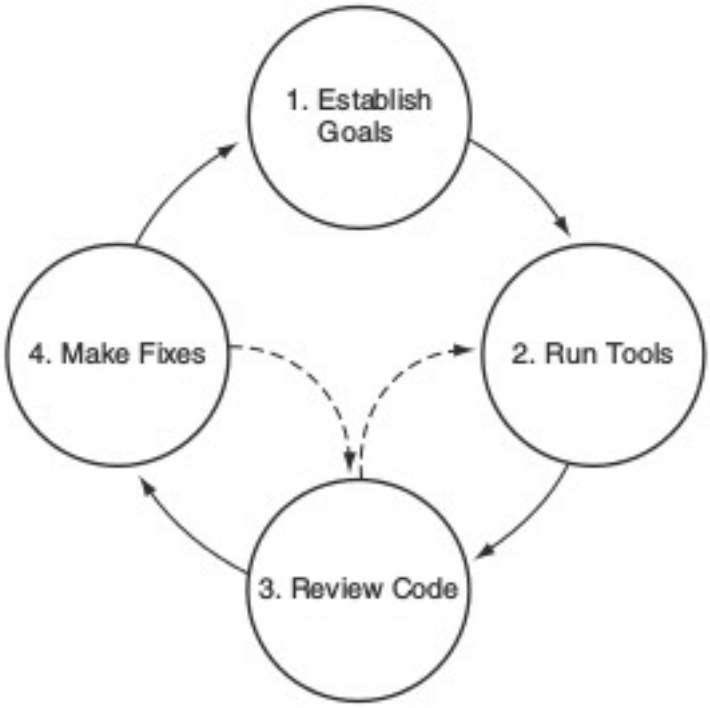
1. Definir objetivos;
2. Executar ferramentas;

3. Revisão do código, baseado nos resultados do passo anterior;

4. Aplicar correções.

A figura 5 a seguir define a ordem a seguir das fases:

Figura 5: Sequência de passos da verificação estática de código



Fonte: Secure Programming with Static analysis (Chess; West , 2007)

Definir objetivos (1. Establish Goals) significa estabelecer critérios de segurança que devam ser seguidos. Qualquer identificação abaixo do requerido invalida a versão do produto, necessitando ser reanalisado e colocado no processo de desenvolvimento corretivo. Um ponto de partida interessante para definir esses objetivos é internalizar o “OWASP Top ten”[9], uma lista das 10 vulnerabilidades mais comumente encontradas nos softwares.

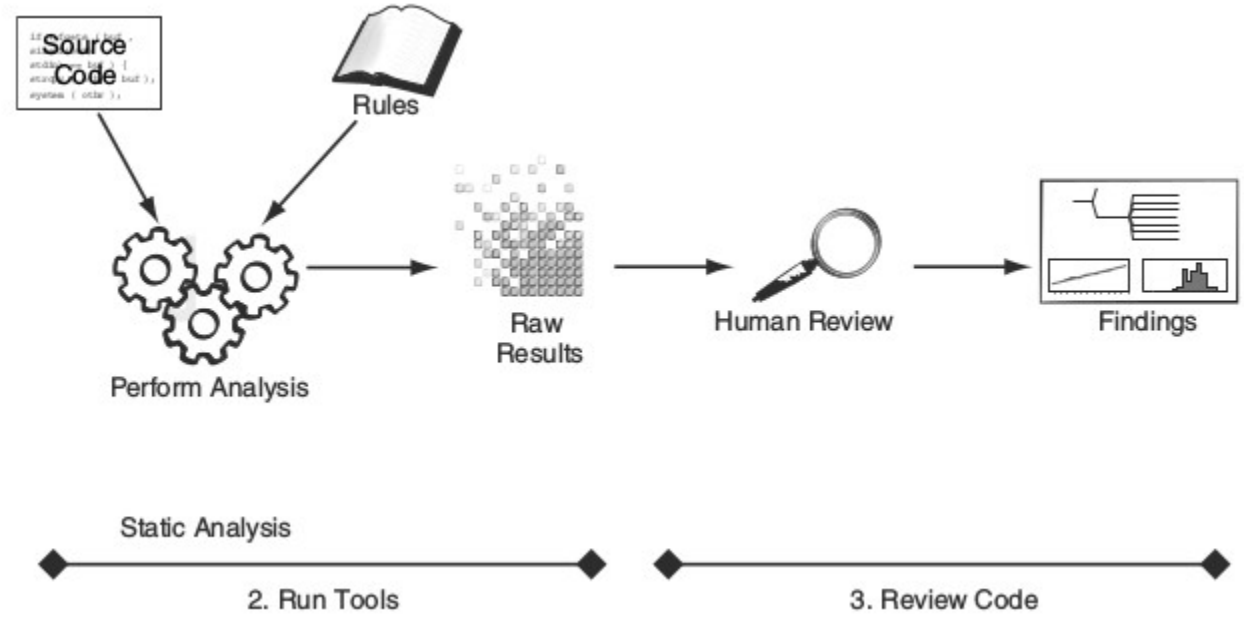
Executar ferramentas (2. Run Tools) consiste em submeter o código a ferramentas de análise de segurança e qualidade. Incialmente, é necessário saber se o código é compilável, não convém analisar o código se ele nem mesmo é compilável; as

9 https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

ferramentas irão gerar resultados baseados nas regras pré-definidas na ferramenta, cada regra pode receber uma criticidade própria que servirá como resultado final de mensuração de criticidade do código, os processos podem ser vistos na Figura 6. Os resultados da análise automática devem então ser submetidos à fase 3.

Ao executar análise automatizada alguns pontos podem ser classificados como os chamados *falsos positivos*, que são questões apontadas pela ferramenta mas que não representam necessariamente um problema, por isso a fase da r*evisão de código (3. Review Code)* é realizada por uma análise manual dos resultados da fase anterior, normalmente realizados por arquitetos e desenvolvedores. Por amostragem de código (dependendo da quantidade de linhas de código) pode-se encontrar também *falsos negativos*, que são vulnerabilidades existentes mas que passaram despercebidas pelas ferramentas. Se, durante esse processo de revisão, você identificar problemas que foram apontados pela análise estática, retorne ao passo 2 (CHESS; WEST , 2007). Na imagem a seguir, vemos em detalhes a representação dos passos 2 e 3.

Figura 6: Submetendo um código fonte a uma ferramenta e obtendo resultados legíveis

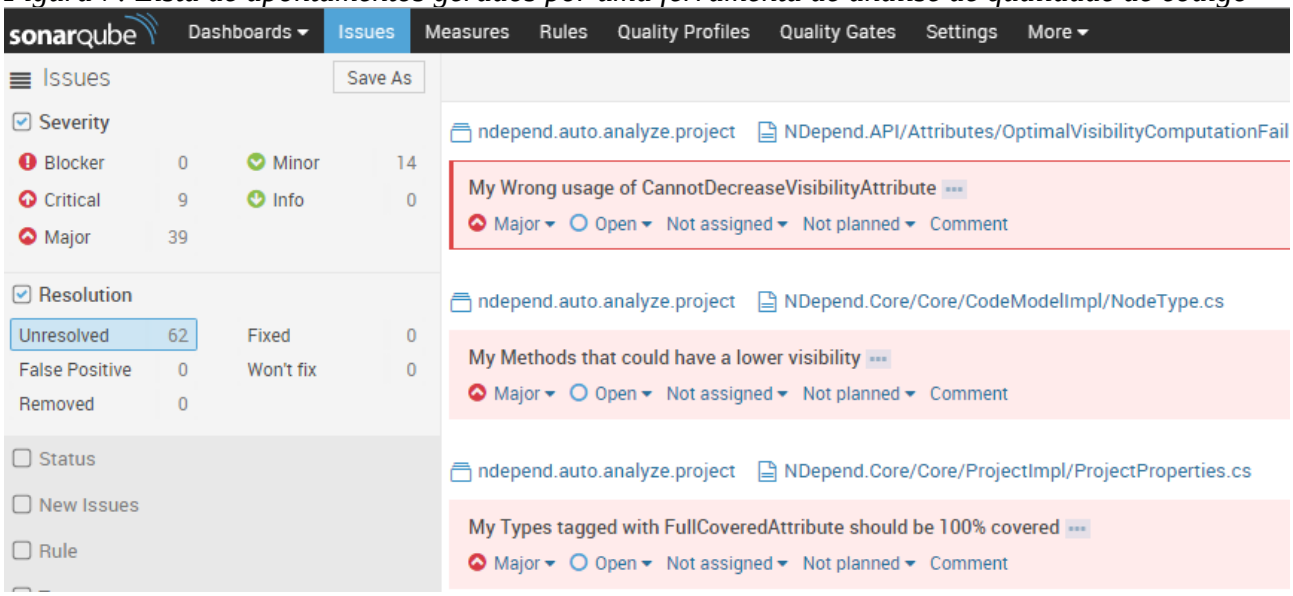


Fonte: Secure Programming with Static analysis (Chess; West , 2007)

Identificado os problemas no código, é hora de realizar o último passo, a *aplicação das correções (4. Make Fixes)*. Uma lista de defeitos é retornado à equipe de desenvolvimento, que tem por objetivo corrigir as questões apontadas. Quando os programadores corrigirem os problemas identificados na revisão, as correções devem ser verificadas (CHESS; WEST , 2007), retornando os passos 2 e 3, como demonstrado na figura 5.

Um exemplo de lista, gerados por uma ferramenta de código aberto (SonarQube[10]), com apontamentos em códigos a serem resolvidos pode ser visto na figura 7, do lado à esquerda vê-se métricas dos apontamentos considerados bloqueadores (extremamente graves), majoritários, minoritários e informativos; do lado à direita vê-se detalhes sobre o que foi apontado.

Figura 7: Lista de apontamentos gerados por uma ferramenta de análise de qualidade de código



Fonte: http://www.ndepend.com/docs/sonarqube-integration-ndepend

3.4.2 Testes dinâmicos

10 https://www.sonarqube.org/

Juntamente com análises estáticas, pode ser realizado também testes dinâmicos. O ciclo de revisão dinâmica de código (DAST) consiste em uma etapa da integração contínua. Dentro do universo da integração contínua, durante a fase de *testes.*

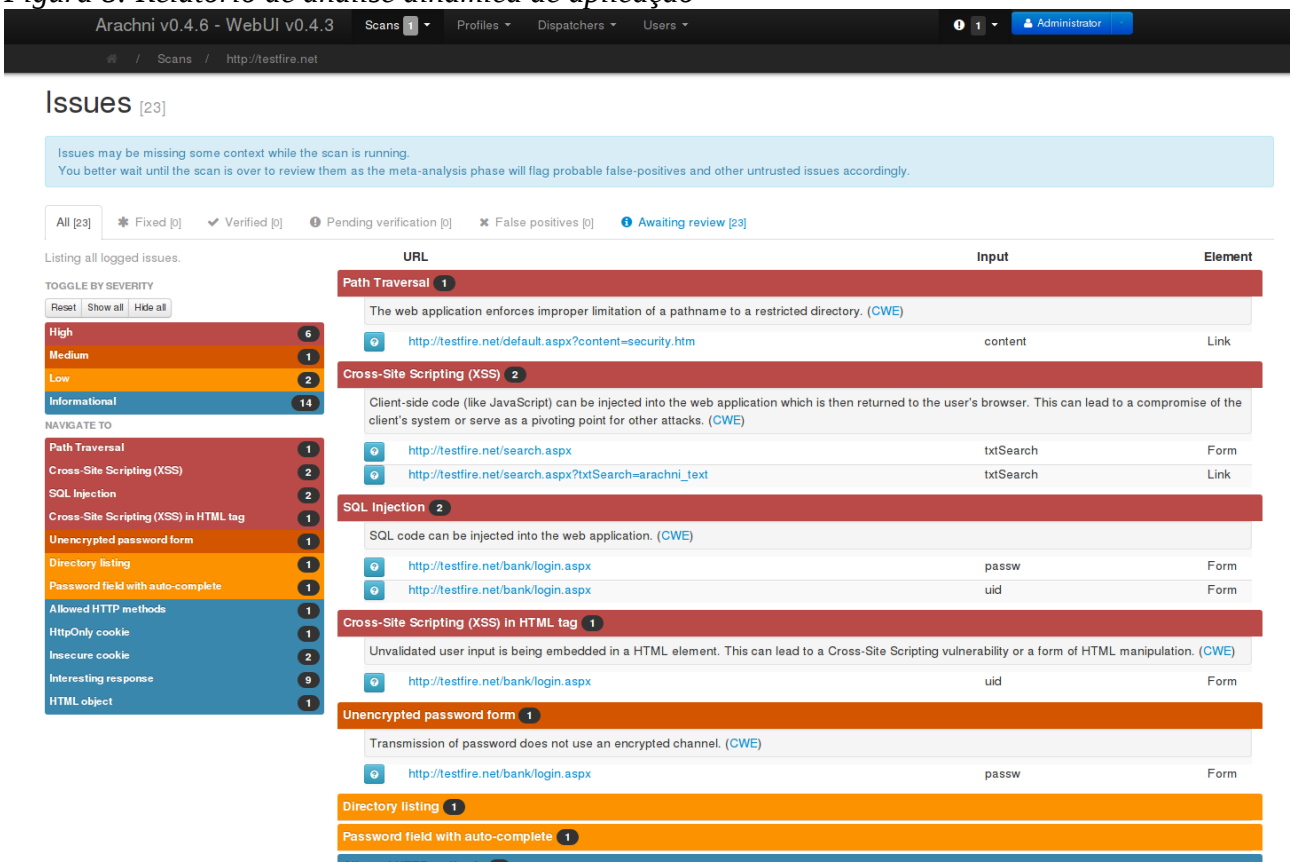
“Para aplicativos da Web voltados ao público, trate novas ameaças e vulnerabilidades continuamente e assegure que esses aplicativos estejam protegidos contra invasões conhecidos” (PCI-DSS versão 3, requisito 6.6). Normalmente, realizado por ferramentas que tem por objetivo varrer a aplicação em busca de vulnerabilidades na aplicação através das intrerfaces expostas, ou não, ao cliente.

Testes dinâmicos simulam ataques, através de uma base de conhecimentos pré definidas na ferramenta, onde o objetivo é identificar vulnerabilidades que seriam identificadas em caso de um ataque real. Estando de posse desses conhecimentos gerados pelo DAST, uma lista de problemas devem ser enviados aos arquitetos e desenvolvedores a fim de mitigar tais problemas.

Na figura 8, vemos um exemplo de resultados gerados por uma ferramenta (Web Arachni [11]) de análise dinâmica disponível gratuitamente na internet. No canto à esquerda vê-se a relação das cores com suas gravidades e uma lista contendo as vulnerabilidades encontradas; à direita vê-se , em detalhes, insumos sobre as vulnerabildades.

11 http://www.arachni-scanner.com/

Figura 8: Relatório de análise dinâmica de aplicação



Fonte : http://www.arachni-scanner.com/

Os resultados da análise estática e dinâmica servem de insumos para os desenvolvedores e arquitetos mitigarem e aprenderem sobre vulnerabilidade de código.

3.4.2.1 Fuzzing

Durante os testes dinâmicos, um teste muito comum é o chamado *fuzzing*, “testes de *fuzzing* por natureza funcionam por meio da geração de entrada de dados a ser alimentada no software. (…) Muitos *fuzzers* irão detectar quebras no programa, que podem ser indicativo de problemas de memória” (SECURE SOFTWARE FOUNDATION, 2014)

Alguns *fuzzers* bastante conhecidos são Wfuzz[12], BurpSuite[13] e ZAP[14].

3.5 Implantação e gerenciamento

Implantação conciste nos processos e atividades relacionadas em como a organização disponibiliza versões do produto que foi criado. Pode envolver produtos disponibilizados para usuários finais, ou para o ambiente interno da organização. Após a implantação, o processo de gerenciamento é iniciado, observando e catalogando eventuais problemas, tanto no produto quanto no ambiente, que venham a ocorrer para que sirva de insumos à equipe de desenvolvimento. Esses insumos devem ser analisados, estudados e, eventualmente, corrigidos, gerando uma nova versão de implantação.

3.5.1 Atividades

- Gerenciamento de vulnerabilidades
    - Gerenciamento de vulnerabilidades é a prática focada em colher potenciais vulnerabilidades identificadas no produto, submetê-las à equipe de segurança de software (desenvolvedores e arquitetos focados em segurança) e definir sua real importância. O sexto requisito do PCI-DSS recomenda “um processo para identificar as vulnerabilidades de segurança, usando fontes externas de boa reputação para informações de vulnerabilidades da segurança e classifique uma escala de risco (por exemplo, "alto", "médio" ou "baixo") para vulnerabilidades de segurança recentemente descobertas.” (PCI-DSS versão 3, requisito 6.1)

12 http://www.edge-security.com/wfuzz.php
13 https://portswigger.net/burp/
14 https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

- Criar um processo de respostas a incidentes, que envolve em reagir de modo preventivo e/ou corretivo e, em alguns casos, uma resposta formal a um cliente que possa ter sido afetado pelo incidente; (PCI-DSS versão 3, requisito 12.10)
- Coletar métricas de cada incidente.

- *Hardening* de ambiente
  - *Hardening* consiste na prática de “restringir a estação ou servidor a oferecer apenas serviços necessários para executar as tarefas para as quais foi designado” (DIÓGENES; MAUSER, 2013) ;
  - Identificar e instalar correções de vulnerabilidades do ambiente em que o produto é disponibilizado;
  - Implantar ferramentas relevantes para segurança de ambiente.

- Uso de antivírus
  - De acordo com o quinto requisito do PCI-DSS, deve-se implementar “softwares de antivírus em todos os sistemas normalmente afetados por softwares mal-intencionados” (PCI-DSS versão 3, requisito 5.1)
  - Certificar “de que os programas antivírus sejam capazes de detectar, remover e proteger contratodos os tipos conhecidos de softwares mal-intencionados.” (PCI-DSS versão 3, requisito 5.1.1)

4 ESTUDO DE CASO

No ano de 2014, uma instituição financeira iniciou seus processos de maturação de software, essa maturação envolvia processos qualidade e segurança dos sistemas, com sua devida disponibilização ao público.

Para se integrar com o processo apresentado neste trabalho, empregando atividades do OpenSAMM, foram selecionadas quais ferramentas de análise de código seriam utilizadas e qual seria a ferramenta orquestradora da integração contínua; uma rápida apresentação sobre o uso das ferramentas foram repassadas às equipes de projetos.

Com o projeto que conciste em padronizar o desenvolvimento e implantação utilizando processo de integração contínua divulgado, foi feito um trabalho de adequação da estrutura dos códigos fontes para serem utilizados de forma automatizada pelas ferramentas de análise e integração contínua. Após os projetos de software inseridos da esteira da integração, toda a requisição de geração de novas versões de um sistema passam, obrigatoriamente, por todos os passos de verificação.

4.1 Objetivo

O objetivo principal era melhorar a qualidade, confiabilidade e segurança das aplicações em plataforma Java[15] que são disponibilizadas para as agências e para o público externo e programas Cobol[16].

Participam do projeto os funcionários das áreas técnicas de desenvolvimento e funcionários terceirizados, as chamadas *fábricas de software* (empresas com vínculo com a instituição financeira por meio de licitação destinados a prover serviços de

15 https://www.java.com/pt_BR/about/
16 http://www-01.ibm.com/support/docview.wss?uid=swg27036733

desenvolvimento de software sob demanda).

O projeto conciste em padronizar o desenvolvimento e implantação utilizando processo de integração contínua, responsável por resgatar o código fonte de um versionador, submetê-los a testes de qualidade e segurança, gerar um pacote da aplicação e implantar no ambiente de desenvolvimento para que a equipe e gestores pudesse testar a aplicação.

4.2 Problemas

4.2.1 Padronização de códigos

Incialmente, os códigos não seguiam uma padronização de desenvolvimento e geração de pacotes muito bem explícitas, as equipes , ou fábricas de software, desenvolviam de acordo com o seus conhecimentos e experiência julgassem mais conveniente.

A padronização obrigou os projetos a abandonarem certas bibliotecas e adotarem outras. Isso gerou incômodo pois diversos projetos tiveram que se adequar, gerando custos de alteração e impactando em orçamentos.

4.2.2 Mudança de arquitetura

Devido à explosão do IoT (*Internet of Things*)[17], as aplicações tem convergido para a adoção da arquitetura REST (*Representational State Transfer*)[18], com adoção de bibliotecas javascript mais poderosas, adoção da nova versão HTML5[19] e CSS3[20].

17 https://aws.amazon.com/pt/iot/
18 http://www.restapitutorial.com/
19 https://www.w3.org/TR/html5/
20 https://www.w3.org/Style/CSS/

Nesta nova arquitetrura, não existe sessão no lado do servidor, não existe estado do cliente, as informações das requisições e repostas costumam trafegar no formado JSON (*JavaScript Object Notation*)[21], um tipo de formatação de dados usando linguagem javascript

A mudança para arquitetura REST gerou impacto em diversas aplicações que tiveram que se adequar.

4.2.3 Geração de pacotes

Para a geração de novos pacotes, foi adotado uma ferramenta chamada Maven[22] , desenvolvida e mantida pela Apache Software Foundation[23]. O Maven controla as versões de ferramentas dependentes na aplicação (ferramentas de terceiros do qual a aplicação faz uso), compila o código fonte e disponibiliza o pacote. O desenvolvedor fica livre do ônus de incluir as dependências e compilar o projeto, pois o Maven faz isso automatizado.

O grande problema em adotar o Maven é que, no início, ele não é muito trivial de compreender sua filosofia e seu funcionamento, diversas equipes tiveram diversos problemas em compreender o que é o Maven, como funciona e como configurar.

Um tempo considerável é gasto na curva de aprendizagem caso se decida adotar essa ferramenta como gerador de pacotes.

4.2.4 Qualidade de códigos

Após as primeiras verificações, quase todos os projetos foram reprovados na

21 http://www.json.org/json-pt.html
22 https://maven.apache.org/
23 https://www.apache.org/

verificação de qualidade. A grande maioria se tratavam de códigos fora da padronização de nomes de métodos, variáveis e classes ; e também de más práticas de programação como objetos declarados não utilizados, trechos de códigos duplicados.

4.3 Resultados

Vencidos os problemas citados acima, é possível obter códigos de sistemas que obedecem à certas regras de qualidade e segurança.

É possível visualizar aplicações com problemas de qualidade e segurança, gerar uma análise detalhada e resumida dos problemas encontrados, reportá-los às equipes e obter correção.

Com a integração contínua e suas verificações automatizadas, há o mínimo de intervenção humana desde o processo de recuperar o código fonte de um versionador e disponibiliza-lo no servidor, tornando automático os processos intermediários de verificação de código.

Com regras de qualidade e verificação dos resultados, é possível obter aplicações mais seguras, com qualidades aceitáveis e tolerância à falhas proveniente de ambientes internos (uma falha de comunição com uma base de autenticação, por exemplo) ou externos (ataques de *SQL Injection*[24], por exemplo), é possível obter aplicações onde a qualidade e segurança possam ser mensuradas e planejadas.

Nas imagens a seguir, podemos notar a evolução da qualidade de um dos projetos de software da instituição financeira referente à problemas *blocker* (**altamente** críticos[25]) e *critical* (críticos).

---

24 https://www.owasp.org/index.php/SQL_Injection

25 Aquelas consideradas acima de críticas; que podem impedir a implantação da nova versão do produto

Nas figuras 9 e 10 vê-se que em 23 de setembro de 2016 a quantidade de problemas altamente críticos eram 8; em 27 de setembro, após nova verificação de código, essas foram corrigidas, totalizando 0 ocorrências.

Figura 9: Quantidade de problemas altamente críticos em 23 de setembro de 2016

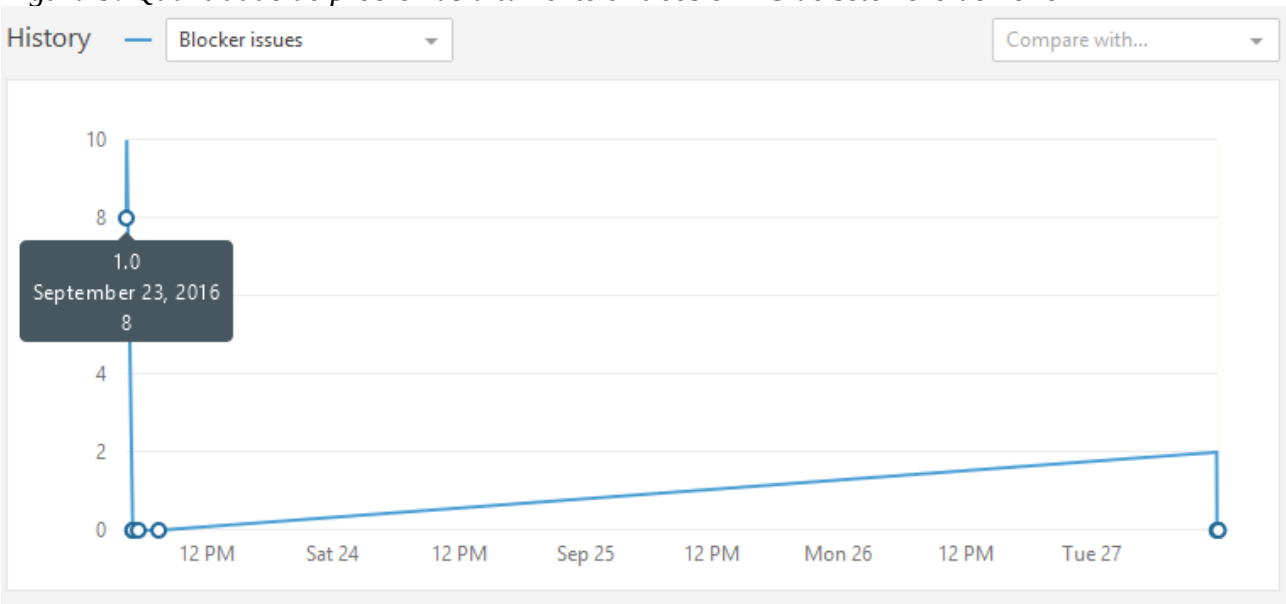
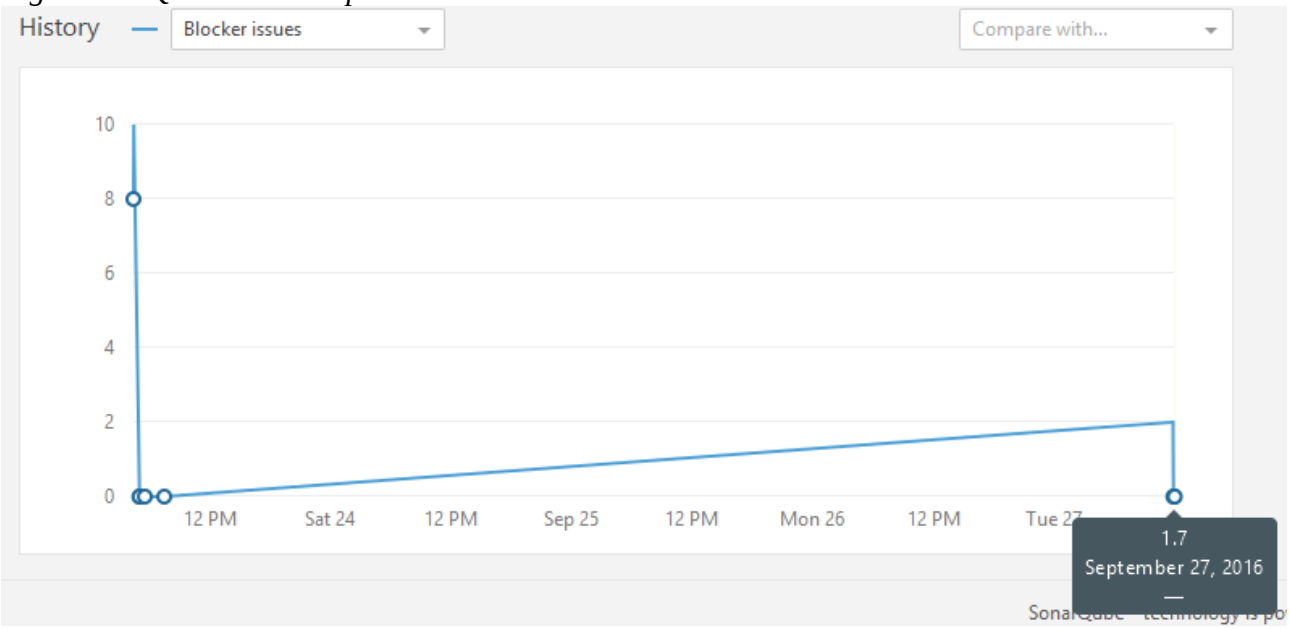


Figura 10: Quantidade de problemas altamemte críticos em 27 de setembro de 2016



Nas figuras 11 e 12 vê-se que em 23 de setembro de 2016 a quantidade de

problemas críticas eram 49; em 27 de setembro, após nova verificação de código, grande parte dessas foram corrigidas, totalizando 1 ocorrência ainda a ser corrigida.

Figura 11: Quantidade de problemas críticos em 23 de setembro de 2016

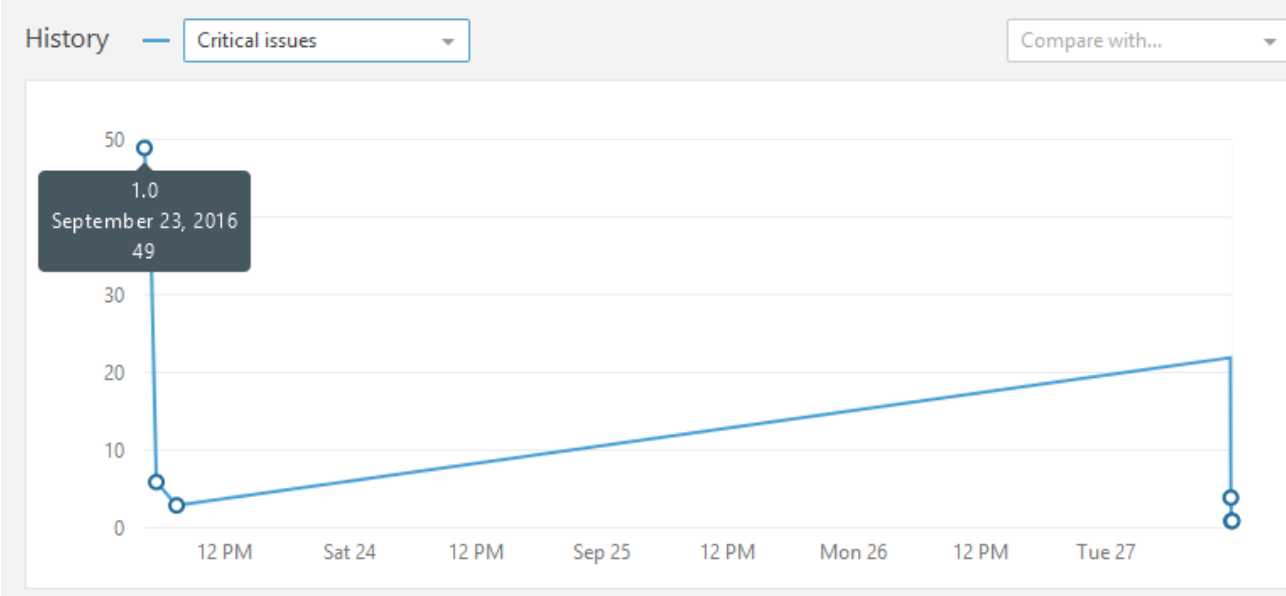
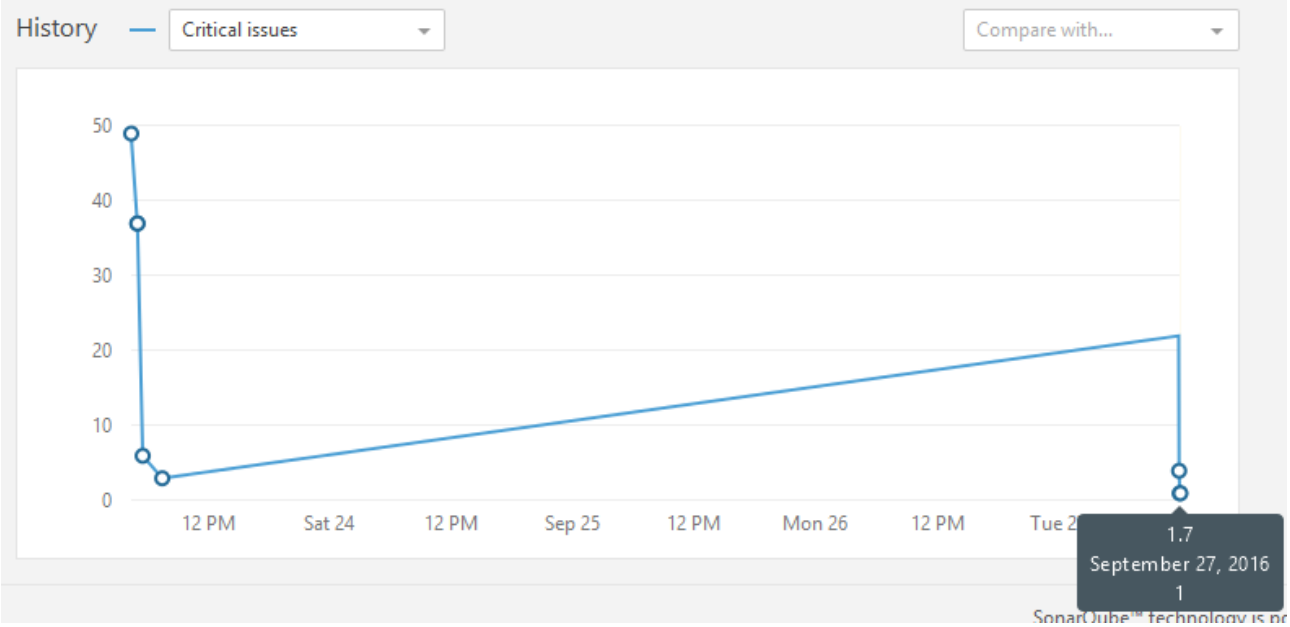


Figura 12: Quantidade de problemas críticos em 27 de setembro de 2016



CONCLUSÃO

Com a expansão, facilidades e dinamicidades oferecidas pela internet, uma grande procupação surgiu com a segurança de informações de usuários que são trafegados pela rede. Estabeler critérios mínimos para oferecer a segurança esperada tiveram que ser criados, mas saber como atingir esses critérios também é um papel fundamental.

Foi então apresentado neste trabalho as atividades mínimas para atender aos requisitos do PCI Security Standart Concil no que diz respeito à segurança de aplicação. Foi proposto um ciclo de desenvolvimento seguro, com orientação e educação dos desenvolvedores; mudanças versionadas nas aplicações; passos automatizados de geração de novas versões como também o controle de qualidade e seguança das aplicações desenvolvidas; obtendo um resultado das inspeções que poderão servir para decisões futuras a respeito do desenvolvimento.

Porém, passos como o de *orientação e educação* e *verificações* ainda podem ser melhorados. Por exemplo, como, garantir que o desenvolvedor está aplicando o que foi passado na orientação antes que a aplicação tenha que passar por nova verificação; garantir a qualidade mesmo com rotatividades de profissionais nas equipes de desenvolvedores; e o mais difícil, prever possíveis ameaças, através de análises dinâmicas e análises de possíveis vetores de ataque ainda na fase de definição da arquitetura, à uma aplicação financeira antes que ela seja aplicada por criminosos para prejudicar uma instituição ou uma pessoa específica.

REFERÊNCIAS

BHARGAV, Abhay ; KUMAR, B. V. . **Secure java for web application development** , Nova York: CRC Press, 2011

CHESS, Brian ; WEST, Jacob. **Secure Programming with Static analysis** , Boston: Pearson Education , 2007

DIÓGENES, Yuri ; MAUSER, Daniel. **Certificação Security+ da prática para o exame SY0-301** , Rio de Janeiro: Novaterra 2013

HEMEL T.C. ; VRIES J.A. . **Framework secure software** Secure Software Foundation, 2014

HOLANDA, Maristela Terto ; FERNANDES, Jorge Henrique Cabral. **Segurança no desenvolvimento de aplicações**, Brasília: Universidade de Brasília, 2009.

OWASP. **Software Assurance Maturity Model (OpenSAMM)** , 2016

PCI Security Standart Concil. **Padrão de segurança de dados da indústria de cartões de pagamento ,** 2013

POSITIVE TECHNOLOGIES. **Web application vulnerability statics (2014)** , 2015

PRESIDÊNCIA DA REPÚBLICA. **Diretrizes para desenvolvimento e obtenção de software seguro nos órgãos e entidades da administração pública federal**. Brasil , 2012.

SEBRAE. **Pequenos negócios: orientações sobre as principais alternativas de serviços financeiros**, 2014

SOMMERVILLE, Ian. **Engenharia de software.** São Paulo: Pearson Education, 2003.

TECHBEACON. **Application security Buyer's guide** , 2016

WHITEHAT SECURITY. **Website Security Statistics Report,** 2015