Relatório Técnico

Auditoria de sistema e de ambiente de votação eletrônica

SINAL
Sindicato Nacional dos Funcionários do Banco Central

MATERIAL EDITADO

Cotação: 28237/11

Cliente:	**SINAL – Sindicato Nacional dos Funcionários do Banco Central**		
Contato:	Sérgio da Luz Belsito	**E-mail:**	nacional@sinal.org.br
Endereço:	SCS Quadra 01 – Bloco G – sala 401 – Ed. Baracat Brasília – DF CEP 70.309-900		
Fone:	(61) 3322-8208	**Fax:**	

1 Resumo Executivo

1.1 Apresentação

O objetivo deste documento é apresentar os resultados das análises do sistema informatizado de votação eletrônica do SINAL realizado pelo CPqD – Fundação Centro de Pesquisa e Desenvolvimento em Telecomunicações, visando avaliar e auditar o sistema de votação eletrônica, principalmente quanto ao sigilo do voto e o respeito à expressão do voto do eleitor.

Este Relatório apresenta as atividades e análises realizadas assim como as conclusões da equipe de especialistas do CPqD a respeito do sistema de votação eletrônico em questão.

Para a elaboração deste Relatório, o CPqD iniciou as suas atividades no dia 19/08/2011, com a reunião de alinhamento e *kick-off* de projeto na sede do SINAL na cidade do Rio de Janeiro.

A metodologia de trabalho adotada pelo CPqD para a elaboração do presente Relatório de foi baseada 3 fases:

I. Avaliação do ambiente de produção e operação

II. Inspeção de código

III. Avaliação forense do ambiente

O presente Relatório descreve tecnicamente as condições da segurança da informação no ambiente e sistema de votação eletrônico do SINAL, sendo que para isto, as atividades realizadas foram determinadas e compreendidas no seguinte escopo:

- avaliação da segurança do ambiente de produção.
- inspeção de código do sistema de votação eletrônica.
- auditoria das permissões de usuário no sistema e no banco de dados.
- auditoria do processo de geração de registros de trilhas de auditoria (sistema e banco de dados).
- análise dos registros de trilhas de auditoria do ambiente (sistema operacional e banco de dados).
- checagem de integridade de código fonte e configurações do sistema e do ambiente.

3 Introdução

Em atendimento ao especificado na Proposta Técnica no 28237/11 – Versão A (Anexo I), este Relatório Técnico apresenta os resultados das análises do sistema informatizado de votação eletrônica do SINAL, a fim de avaliá-lo e auditá-lo em busca de vulnerabilidades e controles de segurança já existentes, principalmente quanto ao sigilo do voto e o respeito à expressão do voto do eleitor.

4 Segurança de software

Prover a segurança de software não é um objetivo fácil de ser alcançado, dada a complexidade dos sistemas nos dias de hoje. Facilmente, uma aplicação atinge dezenas de milhares de linhas de código, que contêm, invariavelmente, uma quantidade de erros significativa. Desses, alguns têm impacto direto em segurança, podendo acarretar desde a indisponibilidade do sistema, até o controle total do computador por um atacante. Para piorar ainda mais, há vulnerabilidades que independem de uma implementação defeituosa, pois a funcionalidade foi projetada já de maneira insegura.

Assim, a segurança deve ser considerada em todas as fases do ciclo de desenvolvimento de software, em vez de se tentar embutir segurança, somente depois que a aplicação estiver pronta. Normalmente, esta última abordagem ou não funciona, ou sai muito mais cara, pois pode haver funcionalidades que sejam completamente ortogonais à segurança. A Tabela 1 (extraída de [2]) ilustra bem esse fato, comparando o custo relativo para correção de vulnerabilidades em cada fase do ciclo de vida dos sistemas.

Fase	Custo relativo para correção
Definição	1
Projeto alto nível	2
Projeto detalhado	5
Codificação	10
Teste de unidade	15
Teste de integração	22
Teste do sistema	50
Pós-entrega	100

Tabela 1: Custo relativo para correção de erro em software.

As vulnerabilidades de software podem ser agrupadas nas seguintes categorias principais:

- Projeto – inerente ao projeto do software, decorrem de decisões equivocadas sobre algum aspecto da aplicação. Independente de quão perfeita seja a implementação do sistema, a fraqueza estará sempre presente. Exemplos incluem o emprego de mecanismos criptográficos inadequados e topologias de rede inseguras.

- Implementação – essas vulnerabilidades resultam da não aplicação de técnicas seguras de programação. Como exemplos, pode-se citar a validação inadequada de entradas e a falta de tratamento de erros de execução.
- Configuração – ocorrem quando os componentes da aplicação e ela própria são configurados de maneira a deixar fraquezas que podem ser exploradas por um usuário malicioso. Por exemplo, considere-se o uso de senhas padrão e a definição de um parâmetro que não limite os recursos alocados por um usuário.
- Operacional – resultam da falta de procedimentos bem definidos para a realização de tarefas relacionadas à segurança. São muito comuns quando mecanismos criptográficos são empregados, pois estes dependem de um gerenciamento de chaves bem implementado, para que forneçam o nível de segurança proposto.

Para se ter uma ideia mais clara do mundo real, de 2001 para 2006, o número de vulnerabilidades em sistemas, reportado pelo *Common Vulnerabilities and Exposures*, simplesmente triplicou. O campeão da lista por muitos anos consecutivos foi o "estouro de pilha", mas que perdeu o lugar, a partir de 2005, para vulnerabilidades de injeção de código como *cross-site scripting* e injeção SQL. Estes tipos de fraquezas afetam basicamente sistemas web e indicam duas coisas:

- A recente popularização de interfaces web para sistemas, sejam eles de comércio eletrônico ou para configuração de elementos de rede;
- Os problemas de segurança deste domínio não estão sendo adequadamente considerados durante o processo de desenvolvimento dos sistemas. E este fato é agravado mais ainda pelo desconhecimento de técnicas de programação segura e a pressão causada por cronogramas de entrega apertados.

A Figura 1 ilustra, para as principais vulnerabilidades, a porcentagem do total anual de entradas no *Common Vulnerabilities Exposures*. Observa-se, no período dado, a progressão das vulnerabilidades acima mencionadas.

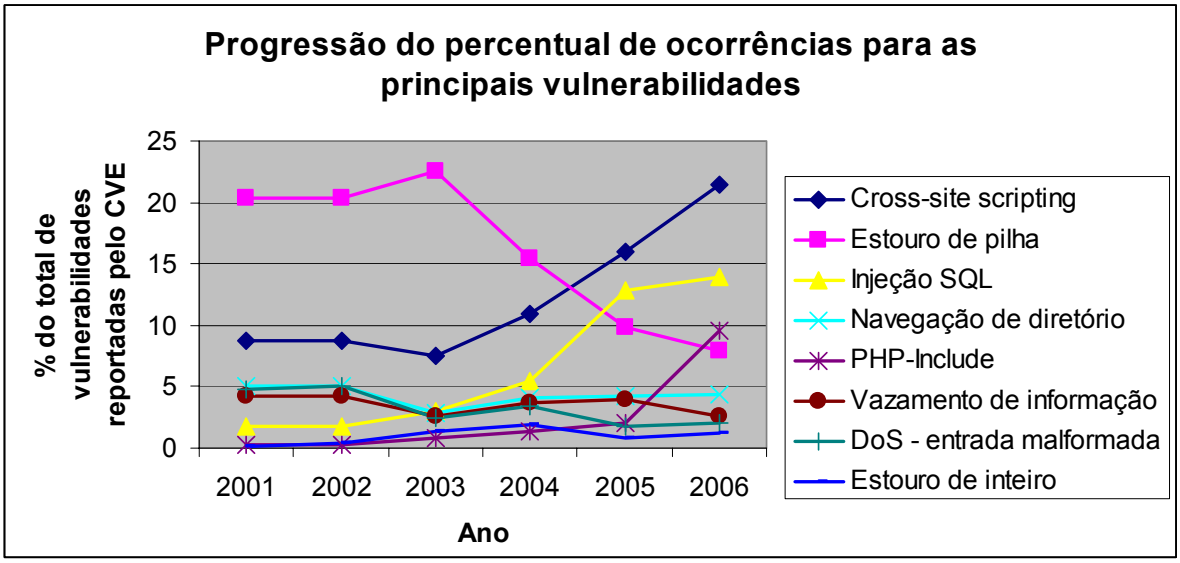

Figura 1: Progressão do percentual de ocorrências das principais vulnerabilidades.

Pode-se considerar, de modo geral, que nem um software está livre de vulnerabilidades de segurança, principalmente quando a complexidade deles for grande. A melhor estratégia a ser adotada, então, consiste em se encontrar essas vulnerabilidades o mais cedo possível e corrigi-las imediatamente, antes que sejam exploradas por usuários maliciosos. A seguir, estão listadas algumas maneiras de se realizar esta tarefa:

- Análise de código-fonte – é a melhor ferramenta possível para detecção de vulnerabilidades em software, mas pode ser proibitiva para sistemas muito grandes. Alguns tipos de defeitos de segurança, porém, somente podem ser encontrados desta maneira.
- Análise de documentos de requisitos e arquitetura – permite detectar problemas no projeto da aplicação.
- Testes de intrusão – consiste em executar ataques contra a aplicação em busca de brechas de segurança, inclusive as decorrentes de implementação insegura. Podem ser de dois tipos: (i) cega ou caixa-preta, quando nenhuma informação sobre a aplicação é fornecida; (ii) caixa-cinza, quando o analista de segurança possui acesso aos documentos de arquitetura e outras informações relevantes para guiar os ataques.
- Ferramentas automatizadas – são ótimas para auxiliar um analista de segurança na execução de processos repetitivos e detecção de alguns tipos de vulnerabilidades. Porém, não são capazes de encontrar inúmeras classes de problemas, que necessitam de um conhecimento do domínio da aplicação e da semântica dos parâmetros utilizados.
- *Checklists* – podem ser utilizados em entrevistas com os responsáveis pela aplicação para verificar se as melhores práticas de segurança para uma dada tecnologia estão sendo seguidas.

Para concluir esta seção, vale a pena ressaltar que vulnerabilidades em software, frequentemente, são as responsáveis pelo comprometimento do bem mais importante, que é a informação. Assim, esforços devem ser direcionados para mitigar os riscos decorrentes do uso de softwares inseguros nos processos de negócio das organizações.

5 Método de trabalho

O sistema de votação eletrônica foi analisado em busca de vulnerabilidades e controles de segurança já existentes por meio de testes de segurança descritos na metodologia OWASP [8] e na metodologia própria do CPqD baseada nas seguintes fases conforme apresentado na Figura 2.

Cada vulnerabilidade identificada é descrita em uma seção individual contendo:

- descrição – breve explicação da vulnerabilidade com possíveis cenários de ataque;
- recomendação – breve descrição do(s) controle(s) sugerido(s) para mitigar a vulnerabilidade.

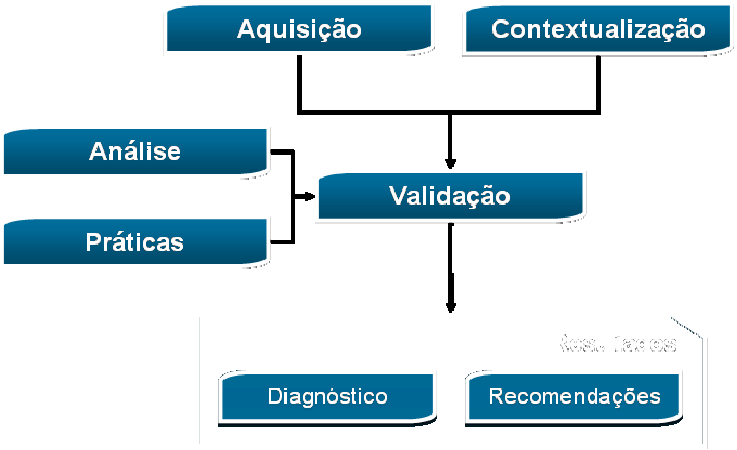

Figura 2: Metodologia de trabalho CPqD.

6 Breve descrição do sistema de votação eletrônica (VE)

O sistema de votação eletrônica (VE) do SINAL é desenvolvido e mantido internamente pelo próprio sindicato, e não possui uma base documental. O sistema está em constante alteração devido às necessidades internas da entidade.

É um sistema de votação no qual os associados podem exercer o direito de voto por meio da Internet, ou seja, o associado após estar cadastrado, pode votar em qualquer local que tenha acesso a Internet.

No escopo de sistema de votação eletrônica, baseado em [4], [5], [6] e [7], destaca-se uma série de requisitos que devem ser atendidos:

- exatidão;
    1. votos não podem ser alterados;
    2. votos inválidos não devem ser computados;
    3. o eleitor deve ter a garantia de que seu voto foi devidamente computado.
- democracia / legitimidade;
    1. somente eleitores legítimos devem poder participar;
    2. cada eleitor deve poder votar uma única vez.
- verificabilidade;
    1. o eleitor deve poder verificar se seu voto foi devidamente computado;
    2. cada eleitor deve poder verificar se todos os votos foram devidamente somados.
- transparência: todos devem poder verificar se a eleição esta sendo conduzida apropriadamente;
- justiça (*fairness*): não deve ser possível obter resultado parcial da eleição de forma a influenciar eleitores que ainda não votaram;
- privacidade do voto / anonimato: o sistema não deve conseguir associar o voto ao eleitor;
- isenção de comprovante (*receipt-freeness*) / segredo do voto: o eleitor não deve receber nenhum comprovante que o permita provar a um terceiro como votou;
- resistência à coerção: o eleitor não pode cooperar com um terceiro a fim de provar a este como votou.

8.2 Perfis de acesso

Existe um único usuário com permissões de acesso sobre o servidor *web*, e é compartilhado entre os administradores. Esse acesso ocorre por meio de servidor de *File Transfer Protocol* (FTP), onde é possível, inserir, remover e alterar qualquer arquivo do sistema, ou seja, este perfil permite acesso irrestrito ao ambiente reservado ao sindicato pela empresa provedora.

Já o acesso ao banco de dados é feito por meio de um único usuário, também compartilhado pelos administradores. As permissões deste usuário ficam condicionadas ao único *schema* do sindicato neste servidor, permitindo acesso irrestrito ao ambiente reservado ao sindicato pela empresa provedora.

O sistema também se utiliza deste único usuário para realizar suas operações sobre a base de dados.

8.3 Geração de trilhas de auditoria

De acordo com o Anexo II, para os acessos por meio do servidor de FTP para manipular arquivos no servidor *web* são gerados registros de trilha de auditoria. Porém, tais registros, ficam disponíveis por um período de, apenas, 7 dias, como especificado em contrato entre o SINAL e a LocaWeb.

De acordo com o Anexo VII, são mantidos, somente, as tentativas de acesso ao banco de dados malsucedidas, não sendo possível identificar que conta acessou a base de dados.

Ainda, de acordo com o anexo Anexo II, o registros transacionais do banco de dados não podem ser disponibilizados para o SINAL devido a restrições contratuais.

11 Recomendação para adequação do sistema de votação eletrônica

Além das recomendações de segurança descritas ao longo deste documento, é importante implementar protocolos de votação eletrônica que atendam aos requisitos exigido por este tipo de sistema. Porém, de acordo com [7], tais protocolos são extremamente difíceis de serem projetados e são muito susceptíveis a erros. Assim sendo, após análise do ambiente e das necessidades do SINAL, propõe-se, no mínimo, o atendimento dos requisitos a seguir.

11.1 Exatidão

Para assegurar a exatidão do sistema de votação é necessário garantir que seja impossível alterar o voto do usuário de forma imperceptível. Essa característica pode ser alcançada por meio de técnicas criptográficas.

Para tal, o voto deve ser cifrado utilizando-se da chave pública do sistema de votação. Para tal é recomendada a utilização de ElGamal, devido à característica que permite o re-ciframento, como descrito em [11].

Após o ciframento, o voto pode ser enviado (cifrado). Um *hash* do valor cifrado pode ser gerado e enviado por email par ao eleitor para servir de comprovante para o eleitor. Tal comprovante pode ser utilizado ao final do processo, para comparar o valor apresentado pelo servidor com o valor armazenado pelo eleitor.

Ao computar os votos, o sistema deve verificar cada um deles a fim de garantir que são válidos, ou seja, que não está direcionado a um candidato inexistente. Após a realização dessa verificação, o voto pode ser computado.

11.2 Democracia / legitimidade

É necessário garantir que apenas pessoas que possuem permissões para votar devem conseguir fazê-lo. Isso pode ser implementado por meio de controle de autenticação. O eleitor, ao enviar o voto para o servidor, deve entrar com suas credenciais de acesso ao sistema.

Além disso, após o cômputo do voto, deve-se marcar o eleitor de forma que ele não consiga votar mais do que uma vez.

11.3 Verificabilidade

Durante a votação o eleitor deve poder verificar se a cifra gerada pelo sistema corresponde, de fato, ao seu voto. O sistema, então deve exibir o valor aleatório gerado para realizar o ciframento. Por meio desse valor, o usuário pode fazer a checagem como desejar.

Porém, após a checagem uma nova cifra do voto deve ser gerada, de forma a garantir que nem mesmo o eleitor conheça o valor aleatório gerado para realizar o ciframento.

Complementar à característica de exatidão, é necessário assegurar que o processo seja verificável. Isso pode ser implementado, disponibilizando, ao final da votação, todos os votos em sua forma cifrada ao lado de ser correspondente decifrado.

Juntamente a isso, devem ser disponibilizadas provas de deciframento, ou seja, provas que permitam a verificação de que o valor cifrado, de fato, corresponde ao par decifrado.

Isso pode ser obtido por meio do protocolo de Chaum-Pedersen descrito em [12], tal protocolo permite a verificação de conhecimento zero (*Verifier Zero-Knowledge*), como descrito em [14]. Isso significa que é possível verificar o texto decifrado, gerando o texto cifrado sem o conhecimento da chave privada. Para que as verificações possam ser feitas de forma não-interativa, tal protocolo pode ser transformado para usando a heurística de Fiat-Shamir, descrita em [13].

De acordo com [9], o comprovante garante apenas a escolha feita, não há nenhuma garantia de que o voto foi, de fato, computado de tal forma. A comprovação de que os votos foram devidamente contados, deve ser feita por meio da verificação da lista de votos apresentada pelo sistema. Por meio das técnicas descritas anteriormente é possível assegurar a legitimidade dos dados apresentados pelo servidor, bastando apenas a confirmação de que as quantidades de voto condizem com o apresentado. Para tal, pode ser disponibilizado um programa para a realização dessa tarefa.

11.4 Transparência

A transparência de um processo de votação eletrônica é um dos pontos fundamentais para que se manenha a confiança em todo o processo. Assim, é necessário que o processo seja transparente o suficiente para que qualquer um tenha a possibilidade de verificar e entender como o processo está sendo conduzido e ainda dispor de mecanismos tais como auditorias internas e externas.

11.5 Justiça (*fairness*)

Devido ao impacto que a implementação desse requisito por meio de técnicas criptográficas o que poderia acarretar na dificuldade de usabilidade do sistema e ainda verificando as necessidades do SINAL, propõe-se que tal requisito seja implementado baseando-se apenas na confiança da aplicação, ou seja, na confiança no sistema de votação. Tal confiança pode ser assegurada para fins de transparência, por meio de mecanismos de auditoria do sistema durante o processo de votação.

É necessário, então, assegurar que o sistema somente apresenta os resultados da votação assim que esta tiver sido dada como encerrada.

Deve-se garantir, também que o par de chaves seja gerado para cada votação e que fique armazenado apenas na memória do programa.

A obtenção desta chave torna-se algo difícil por parte dos administradores, visto que eles têm acesso limitado ao servidor *web*.

O mesmo não seria válido do ponto de vista dos responsáveis pelo gerenciamento do servidor, que neste caso e atualmente, são funcionários da própria LocaWeb. Para esses seria factível a extração da chave da memória da aplicação, como descrito em [10], e de posse dessa chave, poderiam obter todas as informações da base de dados. Para os fins específicos do SINAL, acredita-se que esse risco seja razoavelmente aceitável.

11.6 Privacidade do voto / anonimato

Embora o eleitor tenha que se autenticar antes de efetivar seu voto, não se deve associá-lo ao voto sendo enviado. Esse requisito é assegurado por meio da confiança no sistema. Tal confiança deve ser assegurada por meio de auditoria do sistema.

Deve-se garantir que o sistema não correlacione as credenciais do eleitor com o voto sendo enviado para o servidor.

12 Outras recomendações

12.1 Classificar as informações

Classificar as informações manipuladas pelo sistema de acordo com a criticidade delas para a missão da organização. Devem ser considerados neste processo requisitos aplicáveis de segurança da informação, como confidencialidade, integridade, disponibilidade e autenticidade.

12.2 Fornecer treinamento sobre segurança para as equipes técnicas

Toda a equipe de desenvolvimento, incluindo arquitetos, analistas, programadores e testadores, deve receber treinamento englobando os diversos aspectos de segurança, relevantes para o papel desempenhado pelo profissional. Exemplos de tópicos a serem cobertos incluem conceitos gerais de software seguro e especificidades de linguagens de programação e de plataformas subjacentes utilizadas, que impactam a segurança geral da solução. Recomenda-se que uma parte da equipe receba treinamento externo, ministrado por profissional especializado que se atualize constantemente.

12.3 Criar e manter guias técnicos de desenvolvimento

Escrever documentos específicos por tecnologia, contendo um sumário de recomendações de segurança, que possam ser utilizados pela equipe técnica, ao longo do processo de desenvolvimento. Note-se que novas vulnerabilidades e métodos de ataque surgem diariamente e, assim, esses documentos precisam ser constantemente atualizados para refletir o estado da arte da tecnologia considerada.

12.4 Avaliar riscos associados aos requisitos funcionais levantados

Antes de iniciar as atividades de arquitetura e projeto, é necessário revisar os requisitos funcionais levantados, para identificar eventuais itens que possam tornar o sistema completamente inseguro. É importante que esta verificação seja realizada por um profissional com sólidos conhecimentos sobre segurança, para evitar que funcionalidades

inseguras atinjam as demais etapas de desenvolvimento. Como exemplo, imagine-se um requisito que determina que uma tela para definição de nova senha seja apresentada ao usuário, sempre que três tentativas mal-sucedidas e consecutivas de autenticação forem efetuadas. É fácil observar que o mecanismo de autenticação pode ser comprometido, bastando, para isso, que o atacante forneça senhas inválidas, três vezes seguidas, para uma conta qualquer válida.

12.5 Adotar um conjunto base de requisitos de segurança

Definir e documentar um conjunto base de requisitos de segurança, que devem ser atendidos por todas as aplicações desenvolvidas. As melhores práticas de segurança, riscos conhecidos e requisitos definidos por normas e padrões aplicáveis devem ser considerados na construção desse documento, que deve ser constantemente atualizado para refletir os avanços na área de proteção de software. Exemplos de requisitos de segurança incluem o suporte a uma política de senhas e registro de determinados eventos em trilhas de auditoria, para fins de forense computacional.

12.6 Construir uma matriz de controle de acesso

Baseando-se na finalidade da aplicação e nas necessidades de negócio, definir os possíveis perfis de acesso e as operações permitidas a cada um deles, de acordo com as tarefas desempenhadas. O princípio de mínimos privilégios deve ser sempre respeitado na construção da matriz de acesso, isto é, se uma permissão não é necessária, não deve ser concedida ao perfil. O documento resultante desta atividade serve de insumo oficial para definir os perfis da aplicação em um sistema de controle de acesso baseado em papéis.

12.7 Aplicar explicitamente princípios de segurança ao projeto

A equipe técnica deve utilizar uma lista pré-compilada de diretrizes de segurança, para auxiliar nas atividades de projeto. Exemplos dessas diretrizes incluem: defesa em profundidade, robustecer o elo mais fraco, emprego de segurança por padrão, seguir o caminho mais simples no desenho de funcionalidades de segurança, falhar de modo seguro, balancear segurança e usabilidade, executar com privilégios mínimos e não se calcar em segurança por obscuridade.

12.8 Definir, documentar e adotar arquiteturas e padrões de referência

Especificar e documentar arquiteturas e padrões seguros e validados, que sirvam de referência para a construção dos softwares. Todo novo sistema deve ser desenvolvido com base nos modelos definidos, poupando tempo com a atividade de arquitetura e garantindo um mínimo de segurança, decorrente da robustez dos padrões adotados. Devem ser considerados, por exemplo, padrões para: topologia de rede, mecanismos de autenticação, soluções criptográficas e controle de acesso.

12.9 Definir, documentar e seguir um padrão de codificação segura

Um processo de codificação segura deve considerar as boas práticas genéricas de programação e aspectos específicos da linguagem utilizada. Por exemplo, validação de dados de entrada e tratamento de erros são práticas gerais, que devem ser seguidas na codificação de um programa, independente da linguagem em que é escrito. Por outro lado,

extravasamento de buffer é um problema que afeta programas desenvolvidos em C e C++, mas não os feitos em Java.

Todas as regras de codificação segura, específicas de linguagem e genéricas, devem ser documentadas e adotadas pelos programadores. Além disso, o documento gerado deve ser constantemente atualizado, para que reflita novas descobertas relacionadas a vulnerabilidades. Para uma maior efetividade, a equipe deve ser treinada anualmente acerca do padrão estabelecido.

12.10 Efetuar análise estática de código

A análise de código tem por objetivo encontrar vulnerabilidades e *backdoors* em um programa, por meio da leitura crítica do código-fonte. O ideal é que seja realizada por uma pessoa diferente daquela que escreveu o programa, para que haja imparcialidade na execução da tarefa. Ferramentas automatizadas podem e devem ser empregadas, mas é importante ter em mente que elas não são capazes de encontrar todos os problemas de segurança. Para se ter uma idéia, a taxa média de detecção, apresentada pelas ferramentas existentes, costuma girar em torno de 30 a 40%.

12.11 Realizar testes de segurança, além de testes funcionais

Testes funcionais são realizados com base em roteiros bem definidos e não são capazes de detectar problemas de segurança. Para que vulnerabilidades sejam encontradas, o testador precisa se colocar no lugar do atacante e procurar por fluxos não documentados, que possam conduzir a um estado inseguro da aplicação. Claramente, os dois tipos de teste são bem diferentes e, por isso, a avaliação de segurança deve ser executada por profissional especializado.

No cenário ideal, um teste de invasão deve ser realizado, antes da liberação da aplicação para produção, e outro, neste ambiente, para exercitar a interação do software com as plataformas subjacentes. Sempre que grandes alterações forem realizadas, um novo teste deve ser executado, para verificar se vulnerabilidades não foram introduzidas. Quando a aplicação sofrer apenas pequenas alterações, recomenda-se que uma varredura de vulnerabilidades seja executada com ferramentas automatizadas.

12.12 Robustecer as plataformas subjacentes (quando aplicável)

Vulnerabilidades nas plataformas subjacentes podem, muitas vezes, ser utilizada para atacar aplicações e vice-versa. Assim, todos os elementos, dos quais a aplicação depende, devem ser robustecidos, de acordo com as melhores práticas de segurança conhecidas, as quais devem estar devidamente documentadas para cada plataforma. Esses padrões devem considerar a remoção de contas *default*, desativação de serviços desnecessários, configuração dos parâmetros de segurança, remoção de quaisquer softwares que não sejam utilizados, geração de trilhas de auditoria, sincronização com um servidor de horário, utilização de um servidor por funcionalidade e aplicação periódica de correções. Sempre que novas vulnerabilidades forem descobertas, os documentos de robustecimento devem ser atualizados para refletir os novos controles.

12.13 Aplicar correções de segurança regularmente (quando aplicável)

Vulnerabilidades não corrigidas em plataformas e aplicações compreendem a maneira mais fácil de comprometer a segurança de um ambiente. Desse modo, é fundamental manter um programa de gestão de correções de segurança, que identifique novas vulnerabilidades e trate de corrigi-las nos ativos afetados. Primeiramente, é necessário testar a correção em ambiente de homologação, para se ter certeza de que não afetará a operação dos componentes dependentes. Somente após isso ter sido feito é que a alteração deve ser realizada em produção, segundo um processo de gestão de mudanças.

Correções para vulnerabilidades críticas devem ser aplicadas no máximo após um mês da data de publicação pelo fornecedor. As demais correções podem ser aplicadas até três meses depois de descobertas e, no caso de não ser possível instalá-las, uma análise de risco deve ser realizada, para determinação de controles compensatórios.

12.14 Realizar gestão de mudança

Documentar e implantar procedimentos de gestão de mudança, para evitar que alterações indesejadas ou maliciosas sejam realizadas no ambiente de produção. O processo deve considerar, no mínimo, os seguintes itens: documentação da justificativa da mudança e o impacto que ela causa; aprovação formal pela gerência da mudança a ser realizada; testes a serem realizados, para verificar se a mudança não introduziu problemas funcionais ou de segurança; mecanismo de recuperação, caso o processo resulte em falha. Evidências desses passos, para todas as mudanças realizadas, devem ser mantidos indefinidamente.

12.15 Documentar e implantar procedimentos de gestão de chaves criptográficas

A correta gestão de chaves criptográficas é um ponto fundamental, para que mecanismos criptográficos satisfaçam os requisitos de segurança que propõem atender. Apesar disso, é comum que as práticas relacionadas a esse tópico não sejam consideradas em ambientes reais, o que, muitas vezes, anula a segurança até dos mais robustos criptossistemas existentes. Observe-se que chaves criptográficas devem ser protegidas ao longo de todo o ciclo de vida, que vai da criação até a destruição. As atividades que devem ser tratadas ao longo desse ciclo incluem: geração, distribuição, armazenamento, troca periódica, substituição de chaves comprometidas, custódia múltipla e destruição.

13 Requisitos mínimos de segurança

Os seguintes requisitos de segurança constituem um conjunto mínimo que deve ser atendido por todas as aplicações desenvolvidas:

13.1 Gerenciamento de contas de usuário

- Somente administradores autorizados devem ser capazes de adicionar novas contas de usuário para a aplicação.
- Identificadores de usuários devem ser únicos, isto é, um novo usuário não pode receber um ID já existente.
- Novos usuários devem receber uma senha gerada aleatoriamente, de acordo com a política de senhas fortes, que deve ser trocada logo no primeiro acesso.

- O usuário deve ser capaz de trocar a própria senha quando desejado. Para efetuar esta tarefa, a senha anterior sempre deverá ser solicitada.
- Senhas devem ser armazenadas de maneira protegida.
- O usuário deve ser informado por e-mail quando a senha de sua conta for alterada.
- A aplicação deve suportar a adoção de uma política de senhas fortes, que contemple ao menos:
    - comprimento mínimo de 8 caracteres;
    - uso de senhas contendo caracteres alfabéticos e numéricos;
    - manutenção de histórico com as quatro últimas senhas, impedindo que uma troca seja efetuada para qualquer uma delas;
    - troca periódica de senhas, pelo menos, a cada 90 dias.
- A aplicação deve desativar automaticamente as contas que estejam inativas por mais de 90 dias.
- Em caso de esquecimento de senhas, o sistema nunca deve:
    - recuperar a senha original, o que implica que as senhas não devem ser protegidas por mecanismos reversíveis, como cifras, por exemplo.
    - gerar uma nova senha e exibi-la ao usuário e, tampouco, enviá-la por correio eletrônico.

13.2 Autenticação

- A autenticação de contas administrativas deve se basear em senhas ou em protocolos criptográficos.
- As aplicações/servidores devem sempre autenticar as entidades com as quais se comunicam.
- Credenciais de acesso nunca devem ser enviadas em claro pela rede.
- A mensagem de erro exibida, quando uma tentativa de autenticação for mal sucedida, deve ser uniforme para os casos de identificador inexistente e senha incorreta, para evitar enumeração de usuários.
- A conta do usuário deve ser bloqueada, após três tentativas consecutivas e inválidas de autenticação. O bloqueio deve perdurar por 30 minutos ou até que o administrador reabilite a conta do usuário.

13.3 Gerenciamento de privilégios e autorização

- A aplicação deve suportar a utilização de perfis de acesso, preferencialmente, por meio da integração com um sistema de gestão de identidades.
- Privilégios devem sempre ser atribuídos a perfis e nunca diretamente a usuários.
- Somente devem ser atribuídos os privilégios necessários à execução das tarefas do perfil.

- Perfis devem ser definidos de modo a não violar a separação de responsabilidades.
- A aplicação deve sempre verificar, do lado do servidor, se o usuário possui os devidos privilégios, antes de executar uma operação por ele solicitada.

13.4 Auditoria

- A aplicação deve registrar em trilhas de auditoria, no mínimo, os seguintes eventos de segurança:
    - acessos individuais a dados;
    - ações administrativas;
    - tentativas inválidas de acesso lógico;
    - uso dos mecanismos de autenticação;
    - inicialização e parada do sistema de auditoria;
    - concessão/revogação de privilégios;
    - atividades não usuais para a aplicação.
- Cada registro da trilha de auditoria deve incluir, pelo menos:
    - identificador de usuário;
    - tipo de evento;
    - data e hora;
    - sucesso ou falha;
    - origem do evento;
    - objeto afetado pelo evento.
- A aplicação nunca deve gravar dados sigilosos em trilhas de auditoria.
- Todos os servidores componentes da aplicação devem estar sincronizados com um servidor NTP.
- A aplicação deve ser capaz de registrar os eventos de segurança em um servidor dedicado de trilhas de auditoria.