SEGURANÇA NO DESENVOLVIMENTO DE APLICAÇÕES WEB COM A QUALIDADE DOS DADOS

Sidney Viana
Centro Universitário Dinâmica das Cataratas – UDC/Brazil
sidney.viana@gmail.com

Richard F.Silva
Universidade de São Paulo (POLI-USP)/Brazil
silva@gmail.com

Judith Pavón Centro
Universitário Dinâmica das Cataratas – UDC/Brazil
judith.pavon@unifacs.br

Jean M.Laine
Universidade de São Paulo (POLI-USP)/Brazil
jmlaine@gmail.com

Abstract: **In recent years, data security approaches have been added typically at the end of software development life cycle. However, a more effective approach requires that the security applications be designed from earliest phases of life cycle (security built in). The aim of this paper is to show that this approach contribute to the security control implementation more robust, as well as to discuss the role of data quality area as an important ally to the security implementation of applications. As a contribution of this paper, a set of security recommendations and a high-level framework for developing safe applications are described.**

Keywords: **Security Applications, Techniques of Attack, Data Quality; Common Vulnerabilities; Web Platform.**

Resumo**: Nos últimos anos, a abordagem referente a segurança de dados tem sido considerada tipicamente no final do ciclo de vida do desenvolvimento de software. Entretanto, uma abordagem mais eficiente requer a presença da segurança a partir das fases iniciais do ciclo de desenvolvimento de software. O objetivo deste artigo é mostrar que esta ultima abordagem contribui para a implementação de um controle, para segurança de dados, mais robusto, como também discute o papel da área de Qualidade de Dados como um importante aliado para a implementação da segurança em aplicações. A contribuição deste artigo é um conjunto de recomendações de segurança e uma proposta de um framework para desenvolvimento seguro de aplicações.**

Palavras-chave**: Segurança em Aplicações; Técnicas de Ataque; Qualidade de Dados; Vulnerabilidades Comuns; Plataforma Web.**

1. INTRODUÇÃO

Frequentemente os requisitos de segurança das aplicações não recebem a atenção devida pela equipe de desenvolvimento, principalmente pelo pressuposto frágil de que uma área de segurança da organização resolverá os requisitos de segurança por meio de firewalls, sistemas de detecção de intrusões, algoritmos e protocolos de criptografia. Trata-se do mito de que as vulnerabilidades de segurança serão tratadas pela implementação, a posteriori, de uma camada provida por soluções periféricas [1] que atendam aos princípios tradicionais de segurança definidos em [2]: identificação, autenticação, autorização, confidencialidade, irretratabilidade, integridade e administração de políticas.

Embora seja verdade que tal camada periférica de segurança acoplada às aplicações promove um importante grau de proteção, é igualmente verdade que muitos ataques têm obtido êxito sobre softwares aparentemente blindados por esta camada. Trata-se de técnicas de ataque que exploraram falhas estruturais da codificação dos programas e ou fraquezas da arquitetura de dados.

Muitas vezes, o ataque é facilitado pelo fato dos programadores não se preocuparem com a qualidade dos dados. Estes dados são apresentados em retornos de eventos associados a erros que são capturados pela aplicação, mas não tratados pelo programador.

Nas tentativas de intrusão, os hackers exploram as falhas das aplicações para tentar descobrir, por meio de mensagens desprotegidas, qual o sistema operacional do site onde a aplicação está sendo executada, qual o tipo de página (ASP, JAVA, CGI) e o tipo de banco de dados (Oracle, SQL Server, MySQL) utilizado para armazenar as informações. A Figura 1 representa mensagens geradas por um site de controle acadêmico no instante em que um usuário inexistente foi informado.

Fig. 1. Exemplo de mensagens de erro que podem ser exploradas por um *hacker*

Os trechos marcados na Figura 1 revelam informações como:

• O diretório de localização dos arquivos da aplicação;

• A tecnologia utilizada, neste caso, uma implementação sobre a plataforma .NET (vide extensões .cs e .aspx);

Os nomes de arquivos internos como Login.aspx.cs, que muitas vezes estão ocultos pelo recurso de mapeamento (“alias”) utilizado na publicação dos endereços de uma aplicação web.

Alguns tipos de ataques permitem descobrir até nomes de bancos de dados e tabelas utilizadas no armazenamento dos dados, tal como será apresentado na seção 3, reservada à discussão da influência da qualidade de dados sobre a segurança das aplicações.

O objetivo deste trabalho é mostrar que a atenção à segurança deve ser levada em consideração a partir da concepção do software. Além disso, discutir o papel da área de qualidade de dados como uma importante aliada à implementação de segurança das aplicações. Como contribuição, são propostos neste trabalho um conjunto de recomendações para proteção contra técnicas de ataque e também um framework de alto nível, sob a ótica de segurança, para desenvolvimento de aplicações de segurança crítica. Como será detalhado na seção 5, este framework está baseado na integração dos papéis e responsabilidades dos desenvolvedores, arquitetos de TI e administradores de dados. Na seção 2 é apresentado um conjunto de vulnerabilidades comuns em aplicações web. Na seção 3 é descrita a relação destas vulnerabilidades com dimensões da qualidade de dados. Na seção 4

trata-se de recomendações para proteção contra estas vulnerabilidades. A seção 5 é reservada ao esboço de um framework de alto nível para o desenvolvimento de software seguro e na seção 6 são resultados dos experimentos obtidos e sugestões para trabalhos futuros.

2. Estado da Arte das vulnerabilidades em aplicações web

Neste artigo, a análise das vulnerabilidades está focada sobre as aplicações web, uma vez que problemas de segurança são bastante comuns nas soluções desenvolvidas nesta plataforma. De acordo com estudos recentes da OWASP (Open Web Application Security Project) [3], é possível destacar um conjunto de dez vulnerabilidades mais comuns em aplicações web, conforme apresentado na Tabela 1. Nesta tabela, na primeira coluna são apresentados os nomes das vulnerabilidades ou tipos de ataque e na segunda coluna a sua respectiva definição. Em especial, as vulnerabilidades enumeradas de um a oito merecem especial atenção uma vez que não podem ser evitadas por uma camada de segurança periférica que implemente os princípios tradicionais como identificação, autenticação, autorização, entre outros. Trata-se de vulnerabilidades que são alvos de ataque que atravessam esta segurança ao redor das aplicações e exploram a programação inadequada realizada pela equipe de desenvolvimento. As vulnerabilidades apresentadas são tomadas como referências para a análise com diferentes dimensões da Qualidade de Dados.

Tabela 1- Nove vulnerabilidades de segurança mais frequentes em web [3]

Vulnerabilidade	**Definição**
1. Cross Site Scripting (XSS)	As falhas XSS permitem aos atacantes executarem *scripts* no navegador da vítima, os quais podem roubar sessões de usuário, *pichar* sites *web*, introduzir *worms*, etc.
2. Falhas de Injeção	A injeção (em especial SQL Injection) ocorre quando os dados fornecidos pelo usuário são enviados a um interpretador como parte de um comando. A informação maliciosa engana o interpretador que irá executar comandos mal intencionados.
3. Vazamento de Informações e Tratamento de erros inapropriado	Exposição de informações sobre configuração das aplicações, processos internos ou qualquer violação não intencional de privacidade por meio de problemas na aplicação. Os atacantes podem usar esta fragilidade para roubar informações consideradas sensíveis.
4. Execução Maliciosa de Arquivo	Os códigos vulneráveis à inclusão remota de arquivos permitem ao atacante incluir código e dados maliciosos. Afeta PHP, XML e qualquer *framework* que aceite entrada de nomes de arquivo ou arquivos pelos usuários.
5. Referência Insegura Direta a Objeto	Exposição da referência a um objeto interno, como arquivos, diretórios, registros ou chaves da base de dados, na forma de uma URL ou parâmetro de formulário. Os atacantes manipulam estas referências para acessar outros objetos sem autorização.
6. Falha ao Restringir Acesso À URLs	Frequentemente, uma aplicação protege suas funcionalidades críticas somente pela supressão de informações como *links* ou URLs para usuários não autorizados. Os atacantes podem fazer uso desta fragilidade para acessar e realizar operações não autorizadas por meio do acesso direto às URLs.
7. Cross Site Request Forgery (CSRF)	O navegador autenticado em uma aplicação é forçado a enviar uma requisição pré-autenticada a uma aplicação *web* que, por sua vez, força o navegador da vítima a executar uma ação maliciosa em favor do atacante.

8. Falha de Autenticação e Gerência de Sessão	Credenciais de acesso e *tokens* de sessão não protegidos apropriadamente. Nestes casos, os atacantes comprometem senhas, chaves ou *tokens* de autenticação de forma a assumir a identidade de outros usuários.
9. Armazenamento criptográfico inseguro	Utilização de funções criptográficas de forma inadequada para proteção de informações e credenciais. Os atacantes se aproveitam de informações mal protegidas para realizar roubo de identidade e outros crimes.
10. Comunicações Inseguras	As aplicações frequentemente falham em criptografar tráfego de rede quando se faz necessário proteger comunicações críticas/confidenciais.

O projeto OWASP – Brasil [3] apresentou uma versão para estas vulnerabilidades em 2007. Três anos depois, em 2010, foi apresentado um conjunto de dez vulnerabilidades em função do contexto vigente. Basicamente, se mantiveram a grande maioria das vulnerabilidades apresentadas anteriormente, com modificações mínimas. Uma das diferenças significantes foi a ordem de prioridade, segundo o impacto gerado sobre os dados, em que elas se apresentaram. Neste mesmo ano, O Brasil e Portugal apresentaram uma versão em português, que foi desenvolvida como parte integrante da atividade conjunta dos capítulos luso-brasileiro, em prol da segurança dos sistemas desenvolvidos nos países de língua portuguesa. A última versão oficial americana foi lançada dia 12 de Junho de 2013, e sua correspondente versão em português pode ser acessada pelo site (https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project). Esta versão apresenta aproximadamente setenta por cento das vulnerabilidades inicialmente identificadas e tratadas neste trabalho.

O enfoque apresentado neste trabalho é apresentar uma correlação entre vulnerabilidades e qualidade de dados, razão pela qual se optou pela versão descrita neste trabalho.

2. A INFLUÊNCIA DA QUALIDADE DE DADOS SOBRE SEGURANÇA

Uma definição da qualidade de dados por meio de categorias e dimensões é apresentada em [4], [5], [6], [7] e representada na **Tabela 2**. A categoria "Intrínseca" agrupa as dimensões que dependem exclusivamente de características do dado, independentemente de contexto de sua utilização. A categoria "Contextual" define dimensões totalmente dependentes do mundo externo onde o dado é utilizado. A categoria "Representacional" consolida as dimensões referentes à formatação dos dados e facilidade de entendimento e interpretação. Sob a categoria "Acessibilidade", encontram-se dimensões fortemente atreladas à segurança da informação. A estruturação das categorias e dimensões de qualidade tal como apresentado na **Tabela 2** é denominada abordagem empírica, a qual, em particular, atende bem ao estudo aqui proposto para associação entre qualidade de dados e vulnerabilidade das aplicações. Outras abordagens existem, a exemplo da abordagem intuitiva definida por [8] mas, para o propósito deste trabalho, será utilizada exclusivamente a abordagem empírica.

Tabela 2 - Dimensões de qualidade de dados pela abordagem empírica [4], [5]

Categoria	Dimensão	Definição
Intrínseca	Credibilidade	Dados são aceitos ou considerados como verdadeiros, reais e criveis
	Precisão	Dados são corretos, confiáveis e certificados como livres de erros
	Objetividade	Dados são não enviesados e imparciais
	Reputação	Dados são confiáveis ou altamente recomendados em termos de sua fonte e conteúdo
Contextual	Valor agregado	Dados são benéficos e provêem vantagens pelo seu uso

	Relevância	Dados são aplicáveis e úteis para uma dada tarefa
	Idade	A idade do dado é apropriada para uma dada tarefa
	Completude	Dados são de profundidade e escopo suficientes para uma dada tarefa
	Quantidade de dados apropriada	A quantidade ou volume dos dados disponíveis são apropriados
Representacional	Facilidade de interpretação	Dados estão em linguagem e unidade apropriadas e com definições claras
	Facilidade de entendimento	Dados são claros sem ambigüidade e facilmente compreendidos
	Consistência representacional	Dados são apresentados no mesmo formato e são compatíveis com dados anteriores
	Representação concisa	Dados são representados de modo compacto
Acessibilidade	Acessibilidade	Dados estão disponíveis ou são fácil e rapidamente obtidos
	Segurança de acesso	Acesso aos dados pode ser restringido e mantido seguro

A influência da qualidade de dados sobre as vulnerabilidades apresentadas na seção 2 ocorre principalmente em função da exposição inapropriada dos dados da aplicação aos usuários. Usuários mal intencionados podem forçar conteúdos maliciosos em campos dos formulários da aplicação, visando inserir, alterar ou excluir dados ou simplesmente provocar erros de execução que resultem no vazamento de informações sobre a estrutura física de armazenamento. A partir destas informações, hackers recorrem à engenharia reversa para descobrir a estrutura do banco de dados e, então, tentar ataques baseados nestas informações que, por não terem valor agregado ao usuário comum, não deveriam ser expostas nas mensagens de erro. Quanto maior a quantidade de dados expostos (como nomes de tabelas, de colunas de tabelas, nomes de classes, dados sobre a rede, entre outros) maior é a probabilidade de que se consiga acesso a um conjunto de informações sobre o sistema agredido que poderão ser utilizadas na elaboração de ataques.

Frequentemente, a atenção à qualidade de dados nos projetos de desenvolvimento de software limita-se aos trabalhos do profissional administrador de dados que busca balancear as normalizações, validar integridades referenciais e limitações de valores, definições de chaves, entre outras atividades típicas de modelagem relacional. Todavia, a administração de dados deveria assumir uma responsabilidade mais ampla, incluindo a definição do nível de detalhamento dos dados a serem expostos em mensagens e logs das situações de problema. Por meio desta responsabilidade ampliada, a administração de dados contribuirá para o desenho funcional e técnico que protegem o software contra vulnerabilidades como é o caso de vazamento de informações pelo tratamento inadequado de erros.

Alguns exemplos de relações entre vulnerabilidades comuns de aplicações web e dimensões da qualidade de dados que devem ser discutidas com forte participação da equipe de administração de dados são apresentados na Figura 2.

A relação entre as falhas de injeção (especialmente as de SQL) e as dimensões de qualidade de dados “valor agregado”, “facilidade de interpretação” e “facilidade de entendimento” pode ser explorada por meio de questões a serem respondidas pela equipe de administração de dados, juntamente com a equipe de arquitetura, tais como:

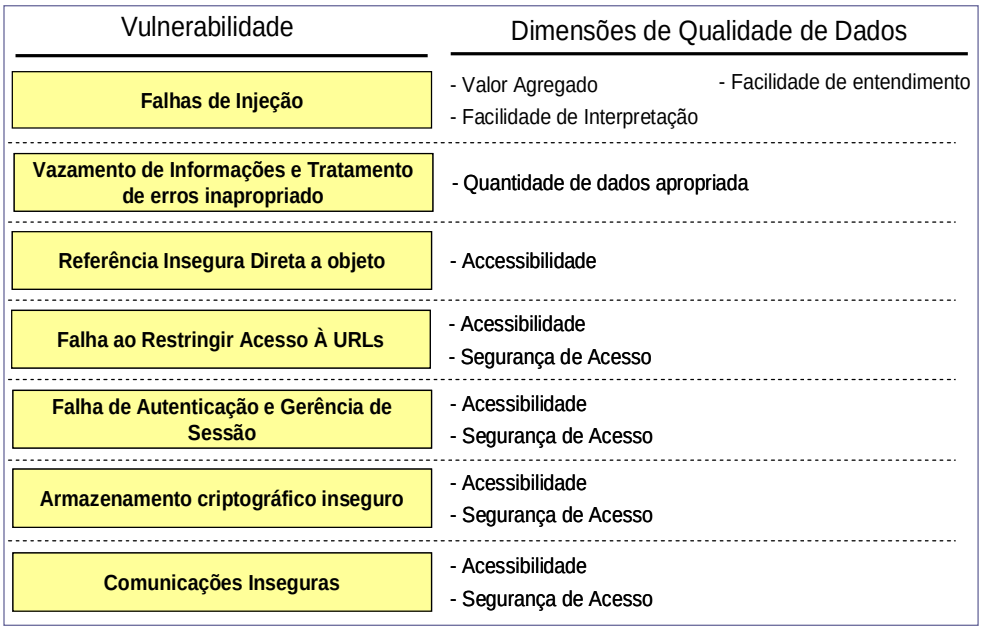

Fig. 2. Exemplos de relação entre vulnerabilidades e dimensões de qualidade de dados

• Quais os critérios de validação dos dados de entrada em aplicações web para proteção contra injeções realizadas por usuários mal intencionados?
• Qual nível de facilidade de interpretação e entendimento da estrutura de dados deve estar contido nas mensagens de erros da aplicação?
• Qual o valor agregado, para cada tipo de usuário, de se expor as informações de estrutura como nomes de tabelas, colunas e constraints (para os casos de banco de relacionais)?

A Figura 3 ilustra um exemplo de exposição indevida de informações em consequência de uma falha no sistema onde uma instrução SQL foi executada por um usuário mal intencionado. Neste exemplo, informações como nomes de tabelas ("p_version e p_namespace") e colunas ("id, namespaceId") são indevidamente expostas aos usuários finais da aplicação (vide trechos indicados em vermelho na Figura 3). A partir destas informações, um usuário mal intencionado é capaz de começar a entender a estrutura do banco de dados acessado pela aplicação e então utilizar este conhecimento na elaboração de estratégias de ataque (como tentativas de consulta, alteração ou mesmo inserção sobre registros deste banco de dados).

```
org.apache.ojb.broker.PersistenceBrokerSQLException:
* Can't prepare statement:
* sql statement was 'SELECT id FROM p_version A0 INNER JOIN
p_namespace A1 ON A0.namespaceId=A1.id WHERE (id >= ?) AND namespace =
?'
* Exception message is [Column name 'ID' is in more than one table in
the FROM list.]
* Vendor error code [30000]
* SQL state code [42X03]
        at
org.apache.ojb.broker.util.ExceptionHelper.generateException(ExceptionHelper.java:256)
```

Fig. 3. *Stack trace* com vazamento de informações

Neste contexto, o valor agregado do detalhamento das mensagens de erro depende dos diferentes tipos de usuário. Assim, para os usuários finais, com interação limitada ao acesso às telas da aplicação web por meio de browsers, as mensagens de erro devem conter apenas a informação relevante de que uma transação não foi executada com sucesso. Para este tipo de usuário, não há valor agregado em expor, por exemplo, o *stack trace* de execução com informação do encadeamento das causas do problema na execução dos programas. Também não é relevante, por exemplo, informações de código de erro da linguagem SQL com indicação de nomes internos de tabelas, colunas e constraints.

Uma solução mais adequada seria a simples apresentação de mensagem com informação de falha na transação, complementada com um código de erro mapeado por uma lógica para tratamento de exceções da aplicação. A partir deste código de erro e outros dados (como identificação do usuário, data e hora da ocorrência, entre outros) os profissionais de suporte seriam capazes de rapidamente encontrar mais detalhes da ocorrência em arquivos de log internos à aplicação, onde a exposição detalhada de *stack trace* e estrutura de dados é relevante e conveniente. Através desta distinção no nível de detalhes das mensagens de erro de acordo com o perfil do usuário, os administradores de dados e arquitetos da aplicação contribuem para proteger a aplicação contra ataques de injeção.

Analogamente, a vulnerabilidade "vazamento de informações e tratamento de erros inapropriado" é associada à dimensão "quantidade de dados apropriada", no sentido em que o excesso de dados gera informações sem um propósito definido e que podem ser utilizadas para fins de ataque à aplicação.

Quanto à vulnerabilidade "referência insegura direta a objeto", está relacionada à dimensão de qualidade de dados "acessibilidade". A equipe de desenvolvimento (especialmente arquitetos e administradores de dados) deve definir como implementar a "acessibilidade" sobre objetos internos da aplicação como arquivos e registros ou simplesmente chaves de um banco de dados. Usuários mal intencionados podem manipular referências válidas para tentar encontrar outras referências a objetos, gerando acesso a informações originalmente não autorizadas [9]. Por exemplo, em aplicações de Internet Banking, é comum usar o número da conta como chave primária, o que torna tentador utilizar esta informação diretamente na interface web. Se não houver uma verificação extra que o usuário é o proprietário da conta informada e que está autorizado a acessar informações relacionadas a esta conta, um usuário mal intencionado pode experimentar alterações do parâmetro número de conta para tentar acesso e modificações ilícitas sobre dados que não lhe pertencem.

Outro exemplo de vulnerabilidade relacionada à atenção inadequada à qualidade de dados é a "falha ao restringir acesso à URLs", sendo que as URLs devem ser entendidas como dados essenciais no desenvolvimento de aplicações web. Neste caso, as dimensões de qualidade de dados "acessibilidade" e "segurança de acesso" devem ser adequadamente consideradas na especificação da aplicação, evitando-se a falha de simplesmente omitir do usuário os links de acesso a endereços não autorizados. A falta de uma validação pelo lado do servidor permitirá que usuários mal intencionados tentem compor as URLs omitidas para que sejam então invocadas pelo browser.

Quanto às vulnerabilidades "falha de autenticação e gerência de sessão", "armazenamento criptográfico inseguro" e "comunicações inseguras", também têm relação direta com as mesmas dimensões de qualidade "acessibilidade" e "segurança de acesso" supramencionadas.

3. TÉCNICAS DE PROTEÇÃO A VULNERABILIDADES EM APLICAÇÕES WEB

Em resposta às vulnerabilidades destacadas na seção 2, há um conjunto de recomendações de proteção da OWASP [3] que vão desde orientações conceituais a práticas de programação e alguns exemplos específicos da linguagem Java.

A Tabela 3 resume estas recomendações que

devem ser objeto de atenção em todo o ciclo de engenharia de software, contribuindo para a aplicação das boas práticas de segurança.

Quanto aos custos para implantação destas técnicas nas organizações, pode-se destacar:

• Treinamento das equipes de desenvolvedores, arquitetos e administradores de dados;

• Documentação das políticas, procedimentos e também de implementações de referência para consulta pelos desenvolvedores;

• Implantação de metodologia de certificação de qualidade do desenvolvimento por meio de validação da aderência dos códigos fonte às políticas, procedimentos e práticas requeridas pelas equipes de segurança, arquitetura e administração de dados;

• Eventuais investimentos em ferramentas de suporte para varredura de códigos em busca de implementações não aderentes e ferramentas de automação de testes de invasão.

Tabela 3 - Técnicas de proteção a vulnerabilidades de aplicações web [3]

Vulnerabilidade	Técnicas de Proteção
1. Cross Site Scripting (XSS)	▪ Mecanismo padrão de validação de entrada (tamanho, tipo, sintaxe e regras de negócio) antes que seja mostrada ou armazenada; ▪ Forte codificação de saída, garantindo que qualquer dado de entrada seja codificado antes da "renderização" nas telas da aplicação; ▪ Não utilização de "listas negras" (procura e troca de poucos caracteres de risco como "<" e ">"), pois os ofensores têm frequentemente explorado com sucesso esta restrição parcial; ▪ Em J2EE, o mecanismo de saída <bean: write...> contribui para esta proteção.
2. Falhas de Injeção	▪ Mecanismo padrão de validação de entrada (tamanho, tipo, sintaxe e regras de negócio) antes que seja mostrada ou armazenada; Não utilização de abordagem de sanitização [10] de dados potencialmente hostis; ▪ Utilização de APIs de *query* fortemente "tipadas" com marcadores substituição; ▪ Atenção ao utilizar *stored procedures* uma vez que são geralmente seguras contra falhas de injeção, mas ainda assim suscetíveis por meio de comandos como exec() ou concatenação de argumentos; ▪ Em J2EE, o objeto PreparedStament fortemente "tipado" contribui para esta proteção.
3. Vazamento de Informações e Tratamento de erros inapropriado	▪ Inibição do detalhamento na exibição de erros (não exibição de informações de *debug*, pilha ou caminhos para usuários finais); ▪ Checagem e configuração prévia de erros de todas as camadas (Web Server, Aplication Server, Banco de Dados, etc) para prevenir que mensagens de erro sejam exploradas; ▪ Criação de um manipulador de erros padrão que retorne uma mensagem de erro já "sanitizada" para maioria dos usuários.
4. Execução Maliciosa de Arquivo	▪ Mapeamento indireto a objetos de referência por meio de apresentação de identificadores "hashed" ao invés de seus valores puros; ▪ Regras de *firewall* para evitar conexões indevidas a *Web sites* externos ou sistemas internos; ▪ Mecanismos de *sandbox* como virtualização para isolar aplicações umas das outras; ▪ Em J2EE, certificar que o gerenciador de segurança está habilitado e configurado adequadamente, conferindo as permissões mínimas requeridas à aplicação.
5. Referência Insegura Direta a Objeto	▪ Não exposição (sempre que possível) de referências de objetos privados como chaves primárias e nomes de arquivos; ▪ Verificação da autorização de acesso a todos os objetos referenciados.
6. Falha ao Restringir Acesso À URLs	▪ Implementação de controle de acesso efetivo que garanta que todas URLs são protegidas por um mecanismo que verifique não apenas o passo inicial de um processo (como a mera inibição de opções de menu, o que pode ser facilmente

	descoberto pelo invasor); ▪ Bloqueio do acesso a arquivos de extensões cuja execução não é requisito da aplicação.
7. Cross Site Request Forgery (CSRF)	▪ Eliminação prévia de vulnerabilidades XSS (que podem ser utilizadas como um passo anterior para acesso não autorizado a credenciais); ▪ Uso de *tokens* randômicos em todos os formulários e URL (exemplo: <form action="/transfer.do" method="post"> <input type="hidden" name="8438927730" value="43847384383">…). O *token* submetido deve então ser verificado para o usuário corrente; ▪ Re-autenticação, assinatura de transação, *tokens* por telefone e *e-mail* para confirmação; ▪ Não utilização de requisições GET para dados sensíveis.
8. Falha de Autenticação e Gerência de Sessão	▪ Uso apenas de mecanismos padrão de gerenciamento de sessão; ▪ Limitação ao uso de cookies; ▪ A função de *logout* deve destruir todas as sessões e a aplicação não deve perguntar por confirmações de *logout* quando o usuário o solicita explicitamente ou via fechamento da janela ou aba ativa; ▪ Uso apenas de mecanismos padrão de gerenciamento de sessão; ▪ Limitação ao uso de *cookies*; ▪ A função de *logout* deve destruir todas as sessões e a aplicação não deve perguntar por confirmações de *logout* quando o usuário o solicita explicitamente ou via fechamento da janela ou aba ativa; ▪ Uso de *time-out*; ▪ Uso de solução robusta para autenticação via modelo de perguntas e respostas (*cognitive password*); ▪ Checagem do *password* antigo quando há solicitação de um novo; ▪ Não utilização de credenciais falsificáveis, como endereços de IP ou máscaras de rede.
9. Armazenamento criptográfico inseguro	▪ Utilização apenas de algoritmos de criptografia aprovados publicamente; ▪ Armazenamento de chaves privadas e credenciais com extremo cuidado; ▪ Garantia que dados armazenados criptografados no disco não são fáceis de descriptografar (por exemplo, criptografia de banco de dados tem pouco valor se a conexão de banco de dados permite acessos não criptografados).
10. Comunicações Inseguras	▪ Uso de SSL para todas as conexões; ▪ Comunicações entre os elementos da infra-estrutura, como servidores *web* e sistemas de banco de dados devidamente seguras.

4. *FRAMEWORK* DE ALTO NÍVEL PARA DESENVOLVIMENTO DE APLICAÇÕES SEGURAS

Considerando a necessidade de se atentar aos aspectos de segurança desde o início do ciclo de vida de desenvolvimento, propõe-se um framework, apresentado na Figura 4, de alto nível que define atividades de segurança e interação entre as equipes nas diversas fases do desenvolvimento de software. O objetivo deste framework é contribuir para o desenvolvimento de software mais seguro por meio da correlação entre as fases do ciclo de desenvolvimento de software e as responsabilidades das equipes envolvidas.

O framework sugere uma integração forte entre as equipes de desenvolvimento, arquitetura e administração de dados desde as fases de planejamento e análise. A implementação de um software seguro exige a observância das características de segurança já nas fases iniciais, em conjunto com o entendimento do processo de negócio, ao invés de adicionar segurança apenas no final do ciclo de vida. Na fase de Planejamento e Análise, é importante definir os dados da aplicação que são fundamentais para o negócio, e os *stakeholders* que detém o conhecimento do negócio podem identificar quais são estes dados importantes e porque devem ser protegidos de ameaças, sejam elas internas ou externas à corporação. Um Sistema Gerenciador de Banco de Dados possui diversos

mecanismos que podem ser utilizados em prol da segurança dos dados, tais mecanismos são os procedimentos armazenados tais como *triggers*, *functions*, *constraints*, entre outros. Além destes mecanismos, é possível utilizar linguagens de programação como por exemplo a linguagem Java.

Os requisitos de negócio e os requisitos de software são fontes de informação que devem ser consideradas importantes para que se toma como ponto de partida para ter em conta na segurança dos dados. Na fase de Desenho (ou projeto) e Implementação, é importante considerar elementos que podem ser fornecidos pela arquitetura de software utilizada. A exemplo disto, considere-se por exemplo uma arquitetura onde um dos módulos é orientada a dispositivos móveis [11], deve-se identificar os recursos apresentados pela arquitetura que podem ser utilizados em conjunto com os requisitos de sistema definidos na fase anterior. Portanto, conhecer a arquitetura em questão é parte importante para a definição da segurança dos dados. Neste sentido, a equipe de desenvolvimento (analistas funcionais, técnicos, líderes de frente e gerentes) deve ter consciência de sua responsabilidade pelo desenvolvimento de softwares seguros, não devendo subestimar a segurança como algo a ser simplesmente acoplado por terceiros.

Por fim, para a confirmação da qualidade de implementação dos padrões de segurança em uma aplicação podem ser utilizadas ferramentas de varredura dos códigos fonte em busca de codificações não aderentes às práticas definidas pelas equipes de arquitetura e administração de dados. Adicionalmente, é de grande importância a realização de testes de caixa branca com o suporte da equipe de arquitetura e administração de dados, podendo-se também recorrer a ferramentas de automatização dos cenários de testes de ataque à aplicação.

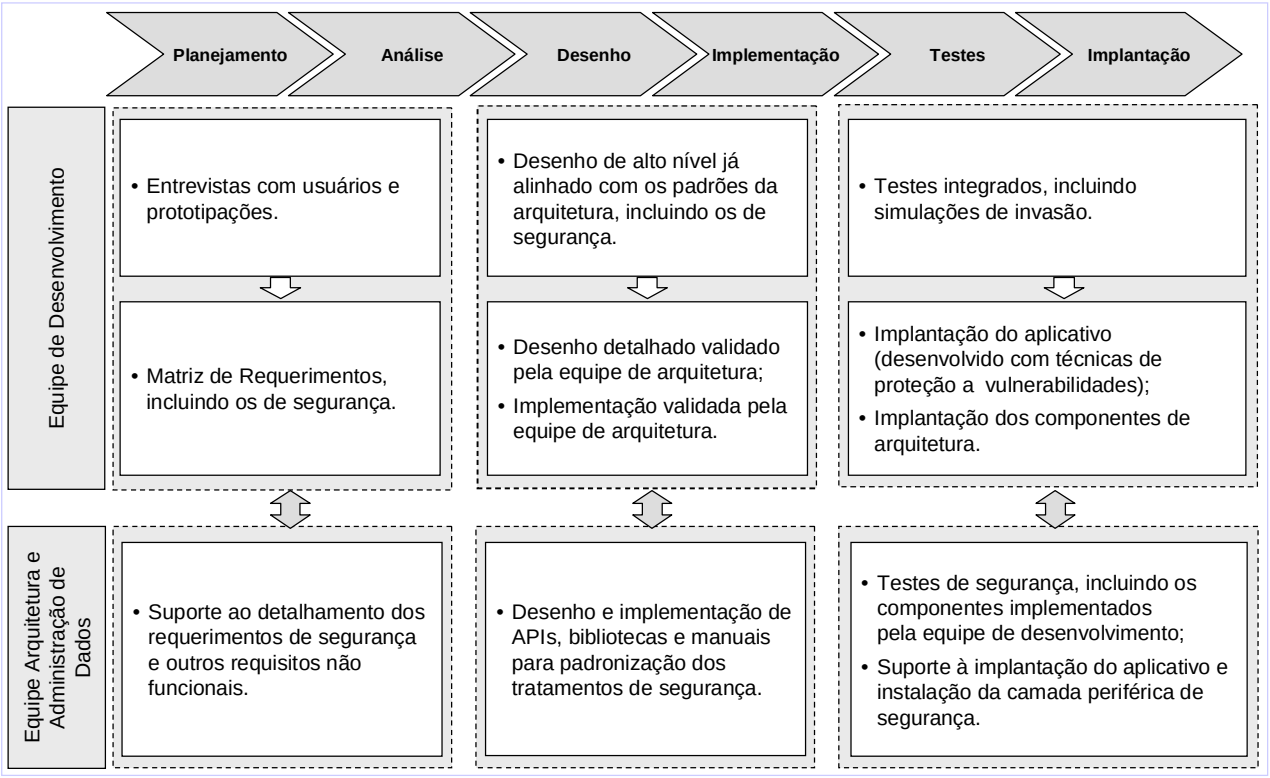

Fig. 4. *Framework* de alto nível para incorporação das atividades de segurança a partir das primeiras fases do ciclo de vida de uma aplicação (*built-in security*)

5. CONCLUSÕES

A expansão das aplicações web tem ocorrido em alta velocidade a ponto que muitas aplicações não foram implantadas com as devidas proteções às vulnerabilidades desta plataforma.

Entre as inúmeras vulnerabilidades, estudos recentes realizados pela OWASP indicam as dez mais recorrentes e mais exploradas pelos invasores. Muitas destas vulnerabilidades não são eliminadas pela simples aplicação de uma camada periférica de segurança, o que torna fundamental o abandono da cultura de que segurança é algo a ser acoplado ao final do projeto de desenvolvimento de software.

Uma abordagem para responder a estas vulnerabilidades é a chamada segurança built-in [12, 13] que se caracteriza pela atenção às boas práticas segurança desde as etapas e planejamento e análise da aplicação. No modelo de segurança built in é fundamental o reconhecimento do papel da arquitetura e administração de dados, incluindo a definição das dimensões da qualidade de dados a serem tratadas no desenvolvimento do projeto. Neste ponto, faz parte do modelo proposto estender as responsabilidades tradicionais da equipe de administração de dados.

Em conjunto com a equipe de arquitetura, os profissionais de administração de dados devem contribuir para definições de segurança. Neste sentido, a administração de dados deve contribuir na definição de como os dados são acessados, expostos e ou manipulados nas camadas da aplicação, o que inclui a definição de tratamento adequado a qualquer forma de exposição como é caso das mensagens de erro apresentadas aos diferentes tipos de usuários.

Quanto às técnicas de proteção apresentadas neste trabalho, é importante considerar que devem ser vistas como algo de natureza temporária. Ou seja, à medida que novos meios de ataque e novas tecnologias são concebidas, surge a constante necessidade de pesquisa e atualização pelas áreas de TI das organizações.

Por fim, é sugerido neste trabalho um framework que tem foco na definição conceitual de papéis e responsabilidades entre as equipes, servindo como um guia de alto nível para o desenvolvimento de aplicativos seguros. Este framework está fundamentado na necessidade de intensificar as interações entre as equipes de desenvolvimento, arquitetura e administração de dados, reforçando a atenção necessária à segurança desde o início do ciclo de vida de uma aplicação.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Lucca, G. A. D., Fasolino, A. R., Mastroianni, M. and Tramontana, P. (2004) “Identifying Cross Site Scripting Vulnerabilities in Web Applications”, In Proceedings of Sixth IEEE International Workshop on Web Site Evolution (WSE’04).

[2] Harris, S. (2008) “All in One, CISSP Exam Guide”, Ed. McGraw-Hill, 4ª edição, cap. 4-8.

[3] OWASP (Open Web Application Security Project), disponível em: https://www.owasp.org/images/4/42/OWASP_TOP_10_2007_PT-BR.pdf. Acesso em junho de 2013.

[4] Angeles, P. e MacKinnon, L. (2005) “Quality Measurement and Assessment Models including Data Provenance to grade Data sources”, In Proceedings of ATINER Conference 2005, Athens.

[5] Pipino, L., Lee, Y. W. and Wang, R. Y. (2002) “Data Quality Assessment”, In Communications of the ACM. April 2002, Vol. 45, No. 4.

[6] Tejay,G., Dhillon, G. and Chin, A. (2006) “Data Quality Dimensions for Information Systems Security: A Theoretical Exposition”, In IFIP International Federation for Information Processing, Volume 193, Security Management, Integrity, and Internal Control in Information Systems, Pages 21-39.

[7] Sidi, F., Shariat Panahy, P. H., Affendey, L. S., Jabar, M. A., Ibrahim, H. and Mustapha, A. (2012) "Data quality: A survey of data quality dimensions", In Information Retrieval & Knowledge Management (CAMP), International Conference 2012, pag.300-304, 13-15 March.

[8] Batini, C., and Scannapieco, M. (2006) “Data Quality, Concepts, Methodologies and Techniques”, Springer, cap. 2, pag. 19.

[9] Reznik, L. (2012) "Integral instrumentation data quality evaluation: The way to enhance safety, security, and environment impact" In Instrumentation and Measurement Technology Conference (I2MTC), 2012 IEEE International, pag.2138-2143, 13-16

[10] HP, Serviço de sanitização de dados HP, disponível em: http://www8.hp.com/h20195/v2/GetPDF.aspx%2F5981-9510PTE.pdf Documento atualizado em abril de 2013. Revisão 4. Acesso em outubro de 2013.

[11] Perakovic, D., Husnjak, S., Remenar, V. (2012) Research of security threats in the use of modern terminal devices. Proceedings of the 23rd International DAAAM Symposium. Vol 23. Nro 1 Vienna. Austria, EU. 2012, psg. 0545–0548.

[12] ADAXES. Built in Security Roles, disponível em: http://www.adaxes.com/help/SecurityRoles.BuiltinSR.html. Acesso em outubro de 2013

[13] APPLE. Security Overview, disponível em: https://developer.apple.com/library/mac/documentation/Security/Conceptual/Security_Overview/Security_Overview.pdf. Acesso em outubro de 2013.