

# Web 2.0 Sob a Perspectiva da Segurança

Fernando Campos, Nery Signorini Neto e Wagner F. Canto  
IPT - Instituto de Pesquisas Tecnológicas do Estado de São Paulo  
fcampos@brastel.com.br, nery.signorini@mac.com e wagner.canto@terra.com.br

## Abstract

*Web 2.0, the second phase in the Web's evolution, provides interactivity and allows the users to add input such as participatory books reviews, blogs, wikis, social networks, video, and photo sharing services. Web 2.0 has consequently become very popular with users. However, the usability increment has also decreased the level of security of these kind of application. The objective of this paper is to analyze, from the software architecture perspective, the existing tradeoffs between the quality attributes security and usability in Web 2.0 applications. To achieve that, the Architecture Tradeoff Analysis Method (ATAM) was used as a reference guide.*

## 1. Introdução

A partir do estouro da bolha da internet em 2001, uma mudança importante começou a ser percebida na arquitetura de aplicações web que passaram a ser desenvolvidas. Páginas web antes estáticas, foram substituídas por sites onde o usuário pode interferir na produção de conteúdo através da publicação de vídeos, fotos e blogs, ou participação em projetos colaborativos e redes sociais [8].

Desde então, um grande número de sites denominados “Web 2.0” têm sido desenvolvidos e lançados. Muito além de simplesmente compartilhar arquivos, a idéia por traz da Web 2.0 é fornecer ferramentas que permitam ao usuário ser um membro ativo de uma comunidade virtual através do compartilhamento de idéias e informações.

O ponto chave que guiou esta transformação foi a percepção por parte dos desenvolvedores da existência de uma característica em comum entre as empresas “ponto com” que sobreviveram à crise da internet: o fornecimento de aplicações com alto grau de interatividade e usabilidade [8].

Nesse contexto, o Ajax, sigla para *Asynchronous JavaScript* e *XML*, aparece como a técnica de

programação que viabilizou a existência dos sites interativos.

Muitas aplicações Web 2.0 têm características e funções similares às de programas tradicionais, que são instalados e executados no computador do usuário. Do ponto de vista da arquitetura de software, não há diferenças entre uma aplicação Web 2.0 e um aplicação cliente-servidor tradicional. No entanto, diferentemente de aplicações que são executadas em um ambiente controlado, os sites Web 2.0 são executados em um ambiente hostil, onde a segurança e confidencialidade de informações é um fator crítico [9].

Alvo de diversos artigos, a questão da segurança em sites Web 2.0 vem sendo bastante discutida pela comunidade desenvolvedora de software. Aparentemente, o fornecimento de funcionalidades que elevaram o patamar arquitetural das aplicações web acabaram também por aumentar a vulnerabilidade e a exposição de dados confidenciais dos usuários, afetando um outro atributo de qualidade, a segurança [3,4,5].

Sabemos que a arquitetura de software é resultado de influências sociais, técnicas e de negócio. Estas influências se dão através dos requisitos funcionais (aquilo que o sistema deve fazer) e também pelos requisitos de qualidade (muitas vezes chamados de requisitos não funcionais) solicitados pelos *stakeholders* no processo de desenvolvimento de um software. Como diferentes *stakeholders*<sup>i</sup> possuem diferentes preocupações e objetivos é de se esperar que conflitos (*tradeoffs*) surjam deste processo [1].

No caso da Web 2.0, é claro o conflito existente entre fornecer mais interatividade ao usuário e, ao mesmo tempo, uma aplicação segura.

O objetivo deste artigo é analisar, do ponto de vista da arquitetura de software, o conflito existente entre segurança e usabilidade no projeto de aplicações Web 2.0. Para isso, foi utilizada a abordagem descrita pelo

ATAM (*Architecture Tradeoff Analysis Method*) [12] como um modelo de referência.

## 2. O que é Web 2.0 ?

Originalmente o conceito de Web 2.0 foi concebido em uma sessão de *brainstorming* entre O'Reilly<sup>ii</sup> e a MediaLive International<sup>iii</sup> [8] em 2005. Após o estouro da “Bolha da Internet” em meados em 2001<sup>iv</sup> das empresas *Dot-Com*<sup>v</sup> nos Estados Unidos. Dale Dougherty, pioneiro da Web e CEO da O'Reilly Media Inc. notou que a web era mais importante do que nunca havia sido anteriormente e surgia agora com novas e excitantes aplicações como por exemplos sites dinâmicos atualizados regularmente e concluiu que o estouro da bolha da internet serviria como um ressurgimento da própria web.

As primeiras definições de Web 2.0 foram baseadas apenas em comparativos entre empresas, sites, práticas, produtos, serviços, diferenciais competitivos, comportamentos de mercado, comportamentos de usuários, de novas tecnologias e de tendências do mercado. Esses comparativos facilitaram o entendimento do que é Web 2.0, definindo uma nova era de serviços e oportunidades.

Web 1.0		Web 2.0
DoubleClick	-->	Google AdSense
Ofoto	-->	Flickr
Akamai	-->	BitTorrent
mp3.com	-->	Napster
Britannica Online	-->	Wikipedia
personal websites	-->	blogging
evite	-->	upcoming.org and EVDB
domain name speculation	-->	search engine optimization
page views	-->	cost per click
screen scraping	-->	web services
publishing	-->	participation
content management systems	-->	wikis
directories (taxonomy)	-->	tagging ("folksonomy")
stickiness	-->	syndication

Figura 1. Comparação de Web 1.0 e Web 2.0 [8]

## 3. A Web 2.0 como Plataforma

O conceito de Web 2.0 continua sendo cunhado, ainda falta muito trabalho a ser desenvolvido, principalmente quanto a conceitos e definições.

Basicamente, a Web 2.0 possui um Centro Gravitacional Central (*i.d. Gravitational Core Competency*) que contém os princípios básicos e as práticas que motivaram seu surgimento, entre elas podemos citar: *a)* a web como plataforma; *b)* você controla seus próprios dados; *c)* baseado em serviços; *d)* arquitetura colaborativa; *e)* custo acessível para escalabilidade; *f)* manipulação e transformação de dados; *g)* software está acima do nível de um único

dispositivo; *h)* aproveitamento da inteligência coletiva [8].

## 3.1 Mapa Mental da Web 2.0

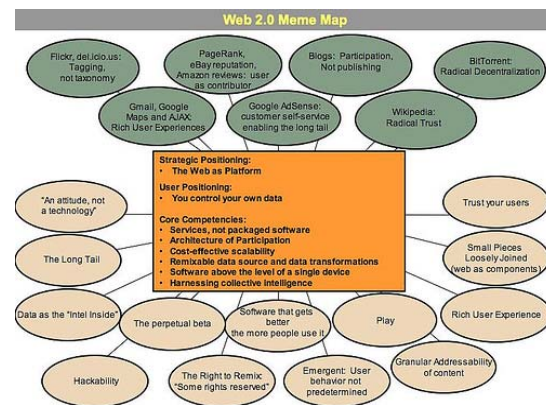


Figura 2. Web 2.0 Memevi Map [8]

## 4. Web 2.0 Design Patterns.

### 4.1 The Long tail

Permitir que o usuário se auto-atenda alinhado a poderosos algoritmos de gerenciamento de dados, possibilitam alcançar toda a web, não apenas se limitando as bordas, mas indo diretamente ao centro, com maior abrangência e profundidade.

### 4.2 Data is the next Intel Inside

As aplicações estão sendo direcionadas cada vez mais ao conteúdo. Uma diferencial na era da Web 2.0 é a construção de um repositório de dados específico, detalhado e único, como diferencial.

### 4.3 Users Add Value

A chave para a vantagem competitiva é o fato de o usuário poder incluir seus próprios dados. Não restrinja a “arquitetura colaborativa” em seu software, envolva os usuários adicionando assim valor à sua aplicação.

### 4.4 Network Effects by Default

Somente um pequeno percentual de usuários trarão problemas durante a adição de valor à sua aplicação. Configure valores padrões (*defaults*) para que a inclusão de informações pelo usuário ocorra naturalmente.

## 4.5 *Some Right Reserved*

A propriedade intelectual restringe o reuso e limita novos experimentos. O valor está na colaboração e não na restrição, garantindo que o grau de restrições e barreiras adotadas seja baixo.

## 4.6 *The Perpetual Beta*

Quando os dispositivos estão conectados à internet, as aplicações não são mais somente artefatos de software, eles tornam-se serviços contínuos. Não encapsule novas funcionalidades lançadas em versões esporádicas, ao invés disso, implemente novas funcionalidades regularmente, como sendo uma prática usual junto aos usuários. Instrumente os códigos para que você consiga monitorar a utilização das novas funcionalidades pelos usuários.

## 4.7 *Cooperate, Don't Control*

As aplicações Web 2.0 são criadas baseadas na cooperação entre serviços. Ofereça serviços na web e informações, reutilize os serviços de outros, mantenha o baixo acoplamento do sistema para facilitar a interação e interconexão entre os diversos sistemas.

## 4.8 *Software Above the level of a Single Device*

O computador pessoal não é mais o único dispositivo de acesso as aplicações da internet. Sua aplicação deve possuir a capacidade de interagir com todos os tipos de dispositivos existentes, sejam *palmtops*, *handhelds*, celulares, outros computadores e outros servidores da internet.

## 5. Lições Aprendidas na web 2.0

Lição 1: Um tipo de auto-serviço utilizado pelo próprio usuário e técnicas avançadas de gerenciamento de dados, provêm uma maior abrangência na obtenção de informações<sup>vii</sup>.

Lição 2: Os efeitos provenientes das contribuições (interações) dos usuários na rede é a chave para o controle do mercado na era Web 2.0.

Lição 3: O serviço fica automaticamente melhor à medida que mais e mais pessoas os utilizam.

Lição 4 :Usuários adicionam valor, trate o usuário como um co-desenvolvedor.

Lição 5: Incentive a colaboração entre os usuários.

## 6. Tipos mais comuns de ataques

Aqui, detalhamos os 8 principais tipos de ataque para aplicações Web 2.0 [2]. Os ataques são os mesmos que podem ocorrer na Web 1.0 e são maximizados pelo aspecto de colaboração com os usuários e entre outros serviços e aplicações da web.

Com base no protótipo criado especialmente para este artigo, estaremos simulando *websites* baseados em tecnologia Web 2.0 como ponto de partida para explorar falhas e vulnerabilidades existentes com base nos atributos de qualidade do Método ATAM [12].

O protótipo servirá como modelo para analisarmos e detalharmos apenas 2 tipos de ataques a falhas ou vulnerabilidades de segurança. Estes tipos serão descritos respectivamente nos capítulos *9.1 Cross-site scripting* e *9.2 Malicious AJAX Code Execution*.

A instância do Método ATAM [12] criada para este artigo é aplicada ao protótipo desenvolvido. Através dessa abordagem, teremos condição de explorar melhor os erros de segurança analisados e de contrapartida, um melhor entendimento dos estímulos, respostas, decisões arquiteturais e conflitos entre os atributos de qualidade.

Abaixo, estão relacionados as falhas mais comuns para aplicações Web 2.0 [8, 10].

### 6.1 *Cross-site scripting*

Quando um atacante utiliza a aplicação web para enviar códigos maliciosos para seu browser e computador. O ataque vem do próprio site que o usuário está utilizando.

### 6.2 *Malicious AJAX Code Execution*

Códigos maliciosos em AJAX que interagem com o browser do usuário, fazendo diversas chamadas externas via *XMLHttpRequest object* trazendo *cookies* para o computador do usuário. Esses *cookies* roubam informações sigilosas dos usuários enviando-as ao atacante.

### 6.3 *Cross-site Request Forgery*

É a utilização de links falsos que redirecionam o usuário para um outro site, onde o atacante poderá facilmente obter informações do usuário ou atacá-lo enviando códigos maliciosos ao seu computador.

## 6.4 XML Poisoning

Devido ao aumento da utilização de arquivos XML para trocas de mensagens na web, como documentos padrão *MS-Office* e arquivos formato *mp3*. Códigos, macros e arquivos maliciosos são embutidos em códigos formato *XML* e enviados aos usuários, que os executarão sem identificar o perigo.

## 6.5 RSS / Atom Injection

*RSS*<sup>viii</sup> é um meio muito popular de compartilhar informações em portais e aplicações web. Uma forma comum de ataque é a injeção de códigos *Java Scripts* dentro da mensagem *RSS* com o objetivo de gerar ataques no *browser* do usuário.

## 6.6 HTTP Request Splitting

O ataque é direcionado para o próprio *web proxy*, originados pela *Lan* e *Wan*. O cabeçalho da nova página *HTTP* é alterado pelo atacante para desempenhar outra função, que não a originalmente concebida possibilitando assim que a nova página execute comando que o atacante determinou.

## 6.7 WSDL<sup>ix</sup> Scanning and Enumeration

O atacante envia diferentes mensagens para todas as partes dos códigos, por exemplo, as operações da página do site que ele deseja atacar, na tentativa de encontrar algum brecha na segurança. Uma vez encontrada a brecha, ele poderá utilizá-la como base para ataques.

## 6.8 E-mail and Web Phishing

É uma atividade criminal caracterizada pela tentativa de obtenção de informações sigilosas, tais como nome de usuário, senhas pessoais, informações financeiras, números de cartão de crédito e outras informações confidenciais.

## 7. Arquitetura de Software e Atributos de Qualidade aplicados a Web 2.0

Primeiramente, vamos a algumas definições:

- 1) **Arquitetura de Software:** Não há uma definição universalmente aceita para este termo. Se pesquisarmos no *website* do Software Engineering Institute (SEI) encontraremos mais de 90

definições para o termo Arquitetura de Software. Uma das definições aceitas é:

“Organização fundamental de um sistema é composta de seus componentes, relacionamento entre eles e com o ambiente e os princípios que guiam seu projeto e sua evolução.” [2].

- 2) **Atributos de qualidade:** Produtos de software possuem atributos associados que demonstram a sua qualidade. Esses atributos não estão relacionados diretamente com o que o software faz, mas sim com o seu comportamento, estrutura, organização do programa fonte e com a documentação associada. Exemplos desses atributos (também chamados de atributos não funcionais) são: O tempo de resposta do software a uma consulta, o nível de facilidade de uso do sistema ou o entendimento do código do programa [7].

No contexto da Web 2.0, estamos interessados nos atributos de qualidade Usabilidade e Segurança, que estaremos definindo o significado a seguir:

- 1) **Usabilidade (ou Capacidade para o uso):** Conjunto de atributos que evidenciam o esforço necessário para se poder utilizar o software, bem como o julgamento individual deste uso, por um conjunto implícito ou explícito de usuários [8].
- 2) **Segurança:** Medida da habilidade de um sistema de resistir a um uso não autorizado enquanto continua provendo seus serviços para usuários legítimos. Uma tentativa de quebrar a segurança é chamada de ataque e pode ser realizada de diferentes maneiras. Podendo ser uma tentativa não autorizada de acessar/modificar dados ou serviços, ou recusar serviços para usuários legítimos [8].

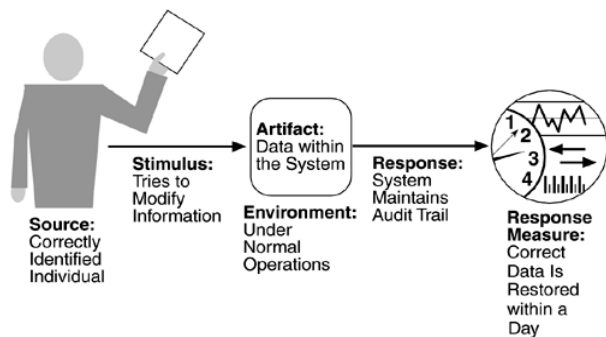


Figura 3 – Exemplo de cenário de segurança [1]

O atingimento de requisitos de qualidade deve ser considerado ao longo de todas as fases de projeto de software (projeto, desenvolvimento e implantação). Nenhum atributo de qualidade é inteiramente dependente do projeto, desenvolvimento ou implantação. Resultados satisfatórios são alcançados através de um bom entendimento do todo (arquitetura) e também os detalhes (desenvolvimento).

Há dois pontos importantes sobre esta questão que precisam ser destacados:

- A arquitetura é crítica para a realização de muitos dos atributos de qualidade de um sistema e estes atributos devem ser projetados e podem ser avaliados no nível arquitetural.
- A arquitetura, por si só, é incapaz de realizar requisitos de qualidade. Ela provê uma base importante para atingir estes atributos, porém esta base será inútil se nenhuma atenção for dada aos detalhes (desenvolvimento, implantação).

Como podemos atingir um determinado requerimento de qualidade através da arquitetura?

Os requerimentos de qualidade especificam como o software deve realizar os objetivos do negócio. O atingimento desses requerimentos dependem fundamentalmente das decisões arquiteturais. As decisões que afetam os atributos de qualidade são chamadas de táticas arquiteturais. A figura 4 representa este relacionamento.

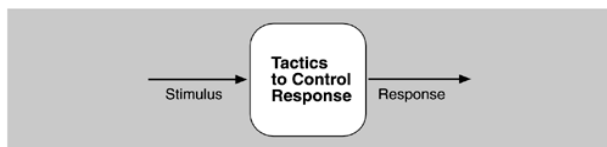


Figura 4 – Táticas arquiteturais para ampliar a segurança [1]

### 7.1 Táticas arquiteturais para ampliar a segurança

Táticas para o atributo segurança podem ser divididas em três categorias: resistir, detectar e se recuperar de ataques. Fazendo uma analogia familiar, colocar uma fechadura na porta é uma forma de resistir a um ataque, ter um sensor de movimento dentro de casa é uma maneira de detectar um ataque e contratar um seguro é uma forma de se recuperar de um ataque. A figura 5 provê um sumário das táticas de segurança.

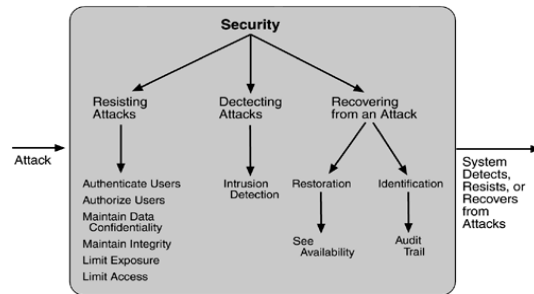


Figura 5 - Resistindo, detectando e se recuperando de um ataque [1]

## 8. ATAM (Architecture Tradeoff Analysis Method)

*Architecture Tradeoff Analysis Method* (ATAM) [12] é um processo de mitigação de riscos que normalmente é executado nas fases iniciais do ciclo de vida de um software. Tem como objetivo identificar áreas de risco potencial, mostrando onde um atributo de interesse (qualidade) é afetado pelas decisões arquiteturais do projeto. Com os riscos mapeados, é possível direcionar mais energia para análise e prototipação, visando mitigar os riscos identificados.

O ATAM revela quão bem uma arquitetura satisfaz um determinado objetivo de qualidade. Também reconhece que decisões arquiteturais tendem a afetar mais de um objetivo de qualidade, provendo uma visão de como os objetivos interagem entre si e dos conflitos gerados por estas interações, no texto original denominado *tradeoffs* [12].

Como principais benefícios do ATAM, podemos citar:

- Efetuar um levantamento preciso dos requisitos de qualidade;
- Criar uma documentação da arquitetura preliminar;
- Criar uma base documental para decisões arquiteturais;
- Promover um levantamento de riscos nas fases iniciais do ciclo de vida do software;
- Promover e melhorar a comunicação entre os *stakeholders* do projeto.

O processo ATAM consiste em reunir os *stakeholders* do projeto para analisar os direcionadores do negócio (*business drivers*) e, a partir desses direcionadores, extrair atributos de qualidade pertinentes ao sistema em questão. O próximo passo é derivar cenários desses atributos e usá-los em conjunto com as abordagens e decisões arquiteturais para

levantar os conflitos (*tradeoffs*), pontos sensíveis, riscos e não riscos. Esta análise pode ser convertida em riscos e seus impactos. Este processo é cíclico e pode ser repetido quantas vezes necessário [1,6].

### 8.1 Um roteiro para o ATAM

Segue abaixo um breve roteiro para aplicação da abordagem ATAM:

1. Apresentação do processo para os *stakeholders* do projeto e esclarecimento de dúvidas.
2. Apresentar os *Business Drives*, isto é, os principais direcionadores do negócio que vão influenciar a arquitetura do novo sistema. Nessa fase, o gerente do projeto ou o principal usuário apresenta:
  - Principais funcionalidades do sistema;
  - Contexto e objetivos do negócio;
  - Principais envolvidos (*stakeholders*);
  - Principais pontos de influências e direcionadores da arquitetura (principais atributos de qualidade que modelarão a arquitetura do sistema);
  - Quaisquer limitadores: técnicos, gerenciais, econômicos ou políticos.
3. Apresentação da arquitetura do sistema proposto em no máximo 1 hora. Esta apresentação deve possuir os seguintes tópicos:
  - Diagrama de contexto: visão geral do sistema. Pessoas e sistema que o novo sistema irá interagir;
  - Visão dos módulos ou camadas: descreve a decomposição do sistema em funcionalidades
  - Visão dos componentes: processos, *data flow*, eventos;
  - Visão da infra-estrutura: *CPUs*, armazenamento (*storages*);
  - Abordagens arquiteturais: abordagens, padrões, táticas empregadas, incluindo os atributos de qualidade que eles endereçarão.
4. Identificação das abordagens arquiteturais: propor *design patterns*, soluções padrão para o problema proposto.
5. Gerar a *Utility Tree* dos atributos de qualidade
  - 5.1. Os times de avaliação trabalham com os tomadores de decisão do projeto para identificar, priorizar e refinar os principais atributos de qualidade do sistema os quais são expressos em cenários. A *Utility Tree*

torna os requisitos mais concretos/palpáveis, forçando o arquiteto e os representantes do negócio a definir precisamente os atributos de qualidade que o novo sistema deve prover.

- 5.2. Cada atributo de qualidade/cenário deverá conter dois indicadores de priorização (alto, médio, baixo). O primeiro indicador prioriza o desenvolvimento do cenário. O segundo quantifica o nível de dificuldade para atingir este atributo de qualidade através da arquitetura do sistema.
6. Analisar abordagens arquiteturais. Nesta fase o time de avaliação analisa os itens da *Utility Tree* mais prioritários e o arquiteto explica como a arquitetura do sistema irá resolver cada ponto. Nesse ponto são identificados os riscos, não-riscos, pontos sensíveis e conflitos.
7. *Brainstorm* e priorização de cenários: Priorizar os cenários e decidir quais serão desenvolvidos em detrimento de outros.
8. Analisar abordagens arquiteturais: Reunião para alinhar os pontos discutidos no passo anterior. O arquiteto apresenta como as decisões arquiteturais irão contribuir para o atingimento/realização de cada cenário/atributo de qualidade que foi priorizado.
9. Saídas do processo
  - 9.1. Abordagens arquiteturais documentadas
  - 9.2. Cenários e priorização levantados durante o *brainstorming*.
  - 9.3. *Utility Tree*.
  - 9.4. Riscos e não-riscos documentados.
  - 9.5. Pontos sensíveis e conflitos (*tradeoffs*) encontrados.

## 9. Aplicação da abordagem ATAM

Para o desenvolvimento deste estudo, não nos baseamos em nenhuma tecnologia especificamente, seja *Java*, *DotNet* ou *Ajax*. Os argumentos e sugestões contidos nesse artigo devem ser encarados como sendo um ponto de partida, uma base de conhecimento a ser consultada pelo Arquiteto de Software durante a condução dos trabalhos de detalhamento e entendimento das decisões arquiteturais pertinentes a seu projeto.

Os exemplos utilizados neste documento como caracteres especiais, partes e linhas de códigos, referências a linguagens de programações e exemplos

de programas são genéricas e ilustrativas. O arquiteto de software deverá abstrair/derivar os cenários contidos nesse artigo, transformando-os e adaptando-os a realidade de seu projeto.

Abaixo, a *Utility Tree* dos atributos de qualidade analisados (1.1 e 1.2) e sua respectiva decomposição.

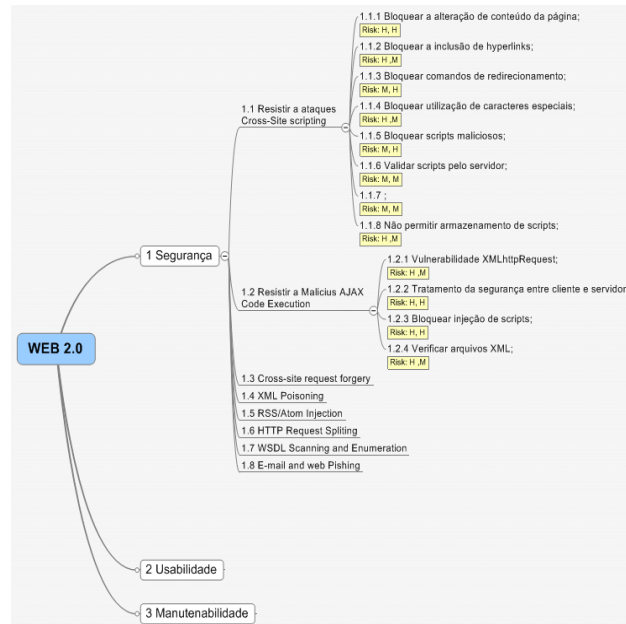


Figura 6: *Utility Tree* de um Sistema WEB 2.0

## 9.1 Cenários Eliciados: 1.1 - Resistir a ataques

### *Cross-Site Scripting (XSS).*

Atributos gerais de segurança em uma operação normal.

- Cenário #1: Bloquear a alteração de conteúdo da página.

**Stimulus:** Usuário utiliza o botão de "Busca" para inserir comandos como inclusão ou alteração de valores dentro da aplicação.

**Response:** Não permitir a execução de comandos ou scripts originados no cliente que alterem ou modifiquem qualquer conteúdo de tela ou dados originais do sistema.

**Quality Attributes:** Perda de usabilidade com diferentes navegadores; Aumento da complexidade do ambiente pelo aumento da complexidade das rotinas de verificação e de controle. Diminuição da portabilidade

da aplicação para outros ambientes. *Tradeoff* entre portabilidade, usabilidade e manutenibilidade;

- Cenário #2: Bloquear a inclusão de *hyperlinks*.

**Stimulus:** Usuário utiliza o botão de "Busca" para inserir *hyperlinks* que redirecionarão o usuário para outros *websites*, sem o consentimento ou conhecimento do usuário.

**Response:** Bloquear a inclusão de comandos ou scripts que provoquem a inserção de *hyperlinks*, que disfarçadamente possam redirecionar o usuário para o site do agressor.

Exemplo: "`search.php?q = <script language= ' javascript' > alert ('hi')</script>`".

**Quality Attributes:** Aumento da complexidade do ambiente pelo aumento de rotinas de verificação e de controle. *Tradeoff* entre portabilidade, usabilidade e manutenibilidade.

- Cenário #3: Bloquear comandos de redirecionamento.

**Stimulus:** Um *hyperlink* é inserido indevidamente pelo usuário na aplicação.

**Response:** Validar e bloquear quaisquer comandos de redirecionamento inseridos pelo usuário durante a utilização do sistema:

Exemplo: "`<script type= ""text/javascript"" > <!-- window.location = ""http://www.google.com"" //--></script>`".

**Quality Attributes:** Aumenta-se a segurança da aplicação *Web 2.0* e do usuário restringindo a capacidade de interação. *Tradeoff* entre usabilidade e segurança.

- Cenário #4: Bloquear utilização de caracteres especiais.

**Stimulus:** O usuário envia à aplicação caracteres especiais com o objetivo de descobrir alguma vulnerabilidade ou falha de segurança existente.

**Response:** Detectar, verificar e bloquear a utilização de caracteres especiais como valores permitidos durante uma interação com o usuário: Exemplo: (null) (branco) "`"" ' ! @ # $ % ^ & * ( ) _ - + = [ ] { } < > , ; : ~ ^ / ? | | "`".

**Quality Attributes:** Aumento da complexidade das rotinas de segurança e de verificação, perda de

usabilidade do usuário na utilização de caracteres especiais em senhas e acentuação de caracteres.

- Cenário #5: Bloquear scripts maliciosos.

**Stimulus:** Receber um script ou comando *SQL* do cliente.

**Response:** Não permitir que o navegador envie scripts e comandos *SQL* ao servidor.

**Quality Attributes:** Perda de segurança em função de usabilidade do usuário na aplicação. Devem ser limitadas algumas funcionalidades da aplicação, como por exemplo, a troca de arquivos entre servidor e navegador.

- Cenário #6: Validar scripts pelo servidor.

**Stimulus:** O servidor recebe um script ou comando *SQL* do cliente.

**Response:** O servidor deve detectar e bloquear scripts ou comandos *SQL*.

**Quality Attributes:** Aumento da complexidade das funções de validação e de controle, perda de portabilidade e usabilidade.

- Cenário #7: Validar scripts pelo cliente.

**Stimulus:** O cliente recebe um script ou comando *SQL* do servidor.

**Response:** O cliente deve detectar e bloquear scripts ou comandos *SQL*.

**Quality Attributes:** Aumento da complexidade das funções de validação e de controle perda de portabilidade e usabilidade;

- Cenário #8: Não permitir armazenamento de scripts.

**Stimulus:** A aplicação armazena os scripts executados.

**Response:** Validar comandos, scripts ou *SQL*'s antes de sua execução pelo servidor.

**Quality Attributes:** Perda de segurança em relação a complexidade da validação de scripts recebidos e do desempenho da aplicação.

## 9.2 Cenários Eliciados: 1.2 - Resistir a *Malicious AJAX Code Execution*.

Atributos gerais de segurança em uma operação normal.

- Cenário #1: Vulnerabilidade *XMLHttpRequest*.

**Stimulus:** Tratar vulnerabilidades para *XMLHttpRequest*.

**Response:** Implementar o protocolo de comunicação baseado em *SSL<sup>x</sup>* (*Secure Sockets Layer*) no servidor.

**Quality Attributes:** Segurança do ambiente em relação a portabilidade da aplicação.

- Cenário #2: Tratamento da segurança entre cliente e servidor.

**Stimulus:** Restringir a abrangência da vulnerabilidade entre servidor e cliente.

**Response:** Limitar o poder e execução do cliente através de verificações e tratamentos internos.

**Quality Attributes:** Utilização de protocolos de segurança e criptografia na comunicação entre cliente e servidor, como por exemplo: *SSL*, *HTTPS<sup>xi</sup>*, *SFTP<sup>xii</sup>* e demais protocolos seguros. Evitar que operações que demandem alto grau de segurança sejam feitas com Ajax sem as devidas precauções quanto a segurança. Restrição da linguagem de desenvolvimento (portabilidade/usabilidade) em relação a segurança da aplicação.

- Cenário #3: Bloquear injeção de scripts.

**Stimulus:** É detectado algum script ou comandos *SQL*'s durante a interação entre a aplicação, usuários ou outras aplicações.

**Response:** Incluir rotinas de validação, teste e tratamento dos scripts e códigos *SQL* recebidos durante a interação.

**Quality Attributes:** Relação entre segurança e usabilidade.

- Cenário #4: Verificar arquivos *XML*.

**Stimulus:** Detectado um código malicioso recebido.

**Response:** Não permitir comandos com caracteres especiais em códigos *XML*.

**Quality Attributes:** Relação entre segurança e usabilidade.

### 9.3 Sensitivity, Tradeoffs, Risks, Nonrisks Identificados.

#### **Sensitivity Points (Sn)**

S1 Usabilidade do sistema é fator crítico de sucesso.

S2 Diversos navegadores podem ser utilizados para acessar o sistema.

S3 De fácil entendimento das funcionalidades da aplicação pelos usuários.

#### **Tradeoffs (Tn)**

T1 Conflito de compatibilidade entre diferentes navegadores pela utilização de controles e validações de conteúdo adicionais.

T2 Conflito entre segurança da aplicação pela usabilidade do sistema.

T3 Conflito de portabilidade da aplicação pela segurança do mesmo.

T4 Conflito entre segurança da aplicação e manutenibilidade.

#### **Risks (Rn)**

R1 Perda de manutenibilidade da aplicação em decorrência dos controles adicionais implementados.

R2 Risco de diminuir a portabilidade do sistema para outras plataformas.

R3 Risco de aumento da complexidade do sistema devido ao aumento das rotinas de verificação e controle.

R4 Risco de perda da interatividade do sistema com os usuários devido a alguma restrição imposta pela segurança da aplicação.

R5 Risco de perda de usabilidade do usuário por alguma restrição imposta de portabilidade.

R6 Risco de perda de compatibilidade entre navegadores pela inclusão de controles e rotinas de verificação.

#### **Nonrisks (Nn)**

N1 A utilização de controles e validações adicionais durante a interação entre usuário e aplicação não afetará o desempenho ou disponibilidade.

N2 As rotinas podem ser utilizadas quantas vezes forem necessárias pelos usuários ou outras aplicações.

N3 O usuário e a aplicação podem utilizar as mesmas rotinas de validação (reuso) sem comprometimento de qualquer outro atributo de qualidade.

### 9.5 Reasoning: *Cross-Site Scripting*

- Proteção contra *XSS* está na recodificação apropriada de todos os dados de entrada, a validação habilita a detecção. A recodificação previne qualquer injeção de scripts ou comandos que serão executados no servidor ou no navegador. A prevenção de *XSS* requer constantes reavaliações arquiteturais dos atributos de qualidade como segurança e usabilidade.
- A validação da entrada deve ser baseada em mecanismos padrões, incluindo verificação de tamanhos, tipo, sintaxes, permissões e regras de negócio antes de se aceitar que o dado seja armazenado ou executado, quaisquer outras situações deverão ser rejeitadas, “aceite o conhecido como bom, rejeite o resto”.
- Todos os dados de entrada do usuário devem ser codificados, como por exemplo para formatos como: HTML, XML, antes da “renderização” da página como sugere a biblioteca *Anti-XSS* do *DotNet*. Utilize padrões de caracteres da página a ser produzida para ISO 8859-1 ou UTF-8 como por exemplo.
- Determine você qual será a codificação de saída, implemente rotinas de validação e substituição de caracteres como “<”, “>” e frases com palavras reservadas da linguagem de programação e.g: *scripts* ou *tags* i.e.: “<b>”.
- Atenção redobrada para os erros de conversão para manter a consistência da aplicação dos dados e do processo de codificação e decodificação aqui sugerido, pois este item, se não for executado com critério se caracteriza em mais uma vulnerabilidade.

### 9.6 Reasoning: *Malicious AJAX Code Execution*

- A injeção de código malicioso ocorre quando o interpretador não possui qualquer tipo de validação ou codificação e a entrada de dados recebida do cliente é um código dinâmico (*queries*).
- Evite a utilização de interpretadores o máximo possível, ou, na necessidade de utilização de interpretadores utilize *APIs* seguras com comandos (*queries*) parametrizados e bibliotecas de mapeamento de objeto relacional, as interfaces seguras diminuem a exposição e vulnerabilidades.
- Deve ser dada a mesma importância à validação das informações de entrada, como no exemplo citado anteriormente para XSS. Todos os dados devem ser validados e codificados antes de serem aceitos.
- Utilize *APIs* de queries fortemente “tipados” com “substituidores” de espaços reservados mesmo quando utilizar *Stored Procedures*.
- Imponha níveis de acessos e privilégios ao banco de dados, não deixe que a aplicação se conecte com usuários fortes (e.g. Administradores, *DBA's*). Evite mensagens detalhadas, faça uma conversão das mensagens antes de serem enviadas ao cliente, quanto menos informação o atacante receber sobre o comportamento do sistema, melhor.
- Utilize *Stored procedures*, pois são geralmente seguras quanto a injeção de *SQL's*, proteja-as de comandos como *exec()* ou pela concatenação de argumentos.
- Não utilize interfaces de *queries* dinâmicas, não utilize funções de fuga simples fracas, como por exemplo: *str\_replace()* pois serão exploradas pelo atacante. Preserve nome de tabela e comandos *SQL's*.
- Atenção para os erros de conversão (codificação e decodificação), como por exemplo: não decodifique a mesma entrada 2 vezes, pois estes erros geram novas vulnerabilidade.

## 10. Conclusão

Web 2.0 é na verdade uma evolução de técnicas e recursos de programação (*XML e Javascript*) que já existiam na web tradicional (*Web 1.0*). Mesmo as funcionalidades apresentadas como grandes novidades são na verdade uma evolução de recursos já oferecidos desde o início da *Web* [10]. Porém, a concentração desses recursos no mesmo web site potencializou as falhas de segurança, aumentando assim o risco de ataques.

Neste contexto, a decisão de se oferecer mais usabilidade ao usuário não pode ser responsabilizada pelo aumento de vulnerabilidade mencionado por diversos artigos [3,4,5]. As falhas de segurança são decorrentes de más práticas de eliciação de requisitos funcionais e não funcionais (uso do ATAM). Ausência de padrões de projeto e desconhecimento de táticas arquiteturais de resistência, detecção e recuperação de ataques [1,9].

Através do uso do ATAM, é possível mapear e equilibrar os conflitos existentes entre os atributos de segurança e usabilidade em aplicações Web 2.0. Este processo estimula a participação dos *stakeholders* nas decisões arquiteturais do projeto, promovendo o equilíbrio entre funcionalidades conflitantes.

Como tema de pesquisas futuras, sugerimos o estudo de padrões de projeto para aplicações Web 2.0 focados em táticas arquiteturais que permitissem o equilíbrio entre segurança e usabilidade.

## 11. Bibliografia

1. Bass, L., P. Clements, and R. Kazman, Software Architecture in Practice. 2nd ed. SEI Series in Software Engineering. 2003: Addison-Wesley.
2. Clements, P., et al., Documenting Software Architecture: Views and Beyond. The SEI series in software engineering. 2005: Addison-Wesley.
3. Davidson, M.A. and E. Yoran, Enterprise Security for Web 2.0. IEEE Computer Society, 2007. 40(11): p. 117-119.
4. Iliyev, D., K.H. Choi, and K.J. Kim, Dangers of Applying Web 2.0 Technologies in E-commerce Solutions, in International conference on Information Science and Security. 2008, IEEE Computer Society. p. 17-25.
5. Lawton, G., Web 2.0 Creates Security Challenges. IEEE Computer Society, 2007. 40(10): p. 13-16.

6. Kazman, R., M. Klein, and P. Clements, ATAM: Method for Architecture Evaluation, C.M.S.E. Institute, Editor. 2000.
7. NBR 13596 (ISO 9126) - Tecnologia de Informação: Avaliação de Produto de Software.
8. O'Reilly, T. What Is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software. 2005; Available from: <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>.
9. Shah, S., Web 2.0 Security: Defending Ajax, RIA and SOA. 2008: Thompson.
10. Yakovlev, I.V., Web 2.0: Is it Evolutionary or Revolutionary? IEEE Computer Society, 2007(November | December 2007): p. 43-45.
11. OWASP Top 10, The Ten Most Critical Web Application Security Vulnerabilities, 2007 Update - 2002-2007 OWASP Foundation.
12. ATAM - ATAM - Method for Architecture Evaluation, Kazman, Rick; Klein, Mark; Clements, Paul; TECHNICAL REPORT CMU/SEI-2000-TR-004, ESC-TR-2000-004, August 2000;

---

<sup>i</sup> Stakeholder é um termo sem uma tradução consensual na comunidade brasileira e por isso foi mantido no original. Basicamente significa parte interessada – com interesses – envolvida no desenvolvimento no sistema.

<sup>ii</sup> O'Reilly, Tim, CEO e fundador da O'Reilly Media Inc..

<sup>iii</sup> Empresa de eventos MediaLive International.

<sup>iv</sup> A bolha da internet, fenômeno observado entre 1995 e 2001, representa o momento mais importante da internet nos últimos dez anos.

<sup>v</sup> Empresas Ponto-Com (*Dot-Com* como i.d. NetScape e outras).

<sup>vi</sup> Substantivo. Termo em inglês (Meme), que consiste de qualquer idéia ou comportamento que pode ser transmitido de uma pessoa para outra por aprendizado ou imitação. Assumimos aqui o termo Mapa Mental em substituição ao termo original.

<sup>vii</sup> Expressão do autor para a abrangência e profundidade a ser almejada: "(...) to the edges and not just the center, to the long tail and not just the head."

<sup>viii</sup> Sigla em inglês para "*Rich Site Summary*" ou "*Really Simple Syndication*", forma simplificada de apresentar o conteúdo de um site.

<sup>ix</sup> *Web Services Definition Language*, é uma interface para *web services*.

<sup>x</sup> *Security Socket Layer SSL* é um protocolo com criptografia para aumentar a segurança e integridade da comunicação baseada em *TCP/IP*.

<sup>xi</sup> *Hypertext Transfer Protocol over Secure Socket Layer* ou *HTTPS* é uma *URL* que indica uma conexão segura.

<sup>xii</sup> *Secure File Transfer Protocol (SFTP)*, é a utilização de protocolo de transferência de arquivos baseado em criptografia.