Uso de Árvores de Ataque e Técnicas de Mutação de Código na Segurança de Aplicações Web

Célio B. Taquary Segundo[1], Luis Fernando Rust C. Carmo[1], Luci Pirmez[1]

[1] Núcleo de Computação Eletrônica – Universidade Federal do Rio de Janeiro (UFRJ)
CCMN, bloco C, Cidade Universitária – Ilha do Fundão – CEP 20.010-974 – Rio de Janeiro – RJ – Brazil

celiotaquary@gmail.com, lfrust@inmetro.gov.br, luci@nce.ufrj.br

Abstract. *The proliferation of web-based applications has increased the exposure of companies to a variety of threats. There are several stages in the life cycle of the applications that are designed to prevent or mitigate those threats. The safety tests are very useful, provided they are efficient. This work focuses on the validation of security testing of web application and proposes a methodology for validation of tools and security testing, based on attack trees, derived from known vulnerabilities disseminated by related security communities. To validate the effectiveness of tests derived from these attack trees, security vulnerabilities are inserted into applications through Mutation Code techniques.*

Resumo. *A proliferação de aplicações baseadas em web tem aumentado a exposição das empresas a uma variedade de ameaças. Há várias etapas no ciclo de vida das aplicações que são destinadas a prevenir ou mitigar essas ameaças. Os testes de segurança são muito úteis, desde que sejam eficientes. Este trabalho foca a validação dos testes de segurança das aplicações web e propõe uma metodologia para validação de ferramentas e testes de segurança, baseada em árvores de ataques, derivadas das vulnerabilidades conhecidas e divulgadas pelas comunidades de segurança afins. Para validar a eficácia dos testes derivados dessas árvores de ataque, inserem-se vulnerabilidades nas aplicações através de técnicas de Mutação de Código.*

1. Introdução

Nos últimos anos, as aplicações web se tornaram extremamente populares, tanto para grandes organizações como para indivíduos comuns. Quase tudo é comercializado, armazenado ou disponibilizado na web, através de aplicações como sites pessoais, comércio eletrônico, bancos, webmails, redes sociais, blogs, fóruns etc. Elas se tornaram tão importantes no nosso dia a dia que não surpreende que se tornem alvo de pessoas mal intencionadas. Um exemplo desse fato é o crescimento da percentagem de ataques baseados na web, que pulou de 25% em 2000 para 61% em 2006, segundo os dados da base de dados de vulnerabilidades da *Common Vulnerabilities and Exposures* (CVE, 2006).

Dessa forma, uma grande parte das aplicações web exige soluções seguras, seja pelo fato de lidarem com ativos de alto valor ou manipularem informações sigilosas ou mesmo por ser objeto de regulamentação por parte do governo ou entidades reguladoras. Embora exista uma crescente preocupação com a segurança das aplicações web, evitar as vulnerabilidades de segurança é ainda um grande desafio devido a extensa gama de fatores que contribuem para tornar essa tarefa ainda mais difícil.

As diversas tecnologias existentes para o ambiente web trouxeram muitos benefícios e facilidades para os usuários, mas também aumentaram em muito a complexidade das aplicações web (Hoglund et al., 2006). Essa diversidade possibilita uma grande quantidade de escolhas quanto a modelos, serviços, algoritmos, parâmetros e estrutura de dados que podem ser empregados no desenvolvimento, tornando as aplicações mais suscetíveis a falhas e, consequentemente, bem mais expostas a ataques.

A pressão do mercado sobre as empresas de Tecnologia de Informação faz com que as mesmas almejem sempre lançar novos produtos antes da concorrência de forma a obter vantagem competitiva. Essa corrida contra o tempo faz com que muitas empresas de TI acabem colocando no mercado produtos/aplicações não seguros, uma vez que garantir a segurança é uma tarefa demorada e os resultados nem sempre são evidentes.

Outro fator que concorre para dificuldade de prevenir essas vulnerabilidades é o fato de que é comum encontrar desenvolvedores e administradores de aplicações web sem o conhecimento necessário ou experiência na área de segurança (Fonseca, 2009).

Complementarmente, a maioria dos mecanismos existentes de defesa específicos para aplicações web atuam de modo reativo, por exemplo, restringindo o acesso não autorizado e/ou não permitindo o uso de determinadas portas. A desvantagem da abordagem reativa é que essa não atinge diretamente o cerne do problema: a existência de software de má qualidade. Torna-se, portanto, imprescindível o desenvolvimento de técnicas mais acuradas (seguras) que permitam a efetiva verificação de pontos vulneráveis dos softwares, e a sua posterior eliminação.

Com intuito de cobrir esta lacuna, várias ferramentas são lançadas no mercado com propósito de realizar testes de segurança em aplicações web. No entanto, essas ferramentas também são softwares e, portanto, podem conter os mesmos problemas. Dessa forma, a grande dificuldade está em como avaliar se essas ferramentas são eficazes para encontrar as vulnerabilidades das aplicações e garantir a segurança das mesmas.

Nesse contexto, este trabalho tem como objetivo geral apresentar uma proposta de metodologia de análise e validação de ferramentas de testes de segurança de aplicações web, as quais cobrem um determinado escopo de vulnerabilidades e usam tanto técnicas estáticas como dinâmicas na detecção dessas vulnerabilidades.

Na busca de uma melhor representação e uma modelagem estruturada desse conjunto de vulnerabilidades, das formas de ataques aplicáveis e das respectivas técnicas utilizadas para detectá-las, optou-se pelo uso de árvores de ataque. O encaminhamento intuitivo provido pelas árvores de ataque facilita o entendimento e a escolha das ferramentas apropriadas para identificar uma determinada vulnerabilidade.

A metodologia proposta compreende basicamente a geração de uma árvore de ataque para determinar o escopo de validação e, em seguida, a execução de ferramentas de testes de segurança em uma aplicação, de forma a cobrir todas as vulnerabilidades determinadas pela árvore de ataque. Caso as ferramentas utilizadas consigam identificar as vulnerabilidades as quais se propõem, temos um indicativo de que elas são eficazes para aquele contexto. Caso contrário, é preciso validar essas ferramentas antes de determinar que uma aplicação seja segura. Para validar a eficácia das ferramentas de teste de segurança que não identificaram vulnerabilidades, inserem-se intencionalmente vulnerabilidades na aplicação alvo, através de técnicas de Mutação de Código (injeção de falhas). Em seguida, aplicam-se os mesmos testes. O resultado da repetição dos testes indica quão adequado são as ferramentas de testes de segurança usadas na validação da segurança de uma aplicação web.

Essa metodologia foi validada experimentalmente através do estudo de um caso bem representativo, fornecendo resultados surpreendentes. Dentre as contribuições deste trabalho, destaca-se a elaboração de uma metodologia inovadora que gera subsídios tanto para a verificação da eficácia de ferramentas de testes de segurança de aplicativos web, bem como para a validação de segurança da própria aplicação web.

2. Trabalhos Relacionados

Dentre as abordagens relacionadas à segurança de aplicações web, podemos destacar o trabalho de Fonseca (2009) quanto ao uso de um injetor de vulnerabilidades para aplicações web. A ideia subjacente é a de que, através do ataque controlado a uma aplicação web, pode-se avaliar o comportamento de uma série de mecanismos de segurança (por exemplo, IDS, scanners de vulnerabilidades, firewall etc.), cujo objetivo é proteger uma aplicação web. Para emular com precisão as vulnerabilidades do mundo real na Internet, Fonseca (2008) utilizou um estudo de campo sobre as vulnerabilidades de segurança real e usou uma ferramenta de injeção de vulnerabilidades. No presente trabalho a injeção de vulnerabilidades é feita seletivamente por técnicas de mutação de código.

Huang et al. (2003) apresentam uma proposta que se aproxima, em termos funcionais, da proposta de injeção de vulnerabilidades para validação de mecanismos de segurança apresentada por Fonseca. Ela se baseia explicitamente na injeção de erros seguida de monitoração comportamental. A diferença básica é que sua abordagem com injeção de erros amplifica muito o escopo de validação da aplicação, pois tem que lidar com erros imprevisíveis que não geram vulnerabilidades.

2.1. Árvores de Ataque

A árvore de ataque é uma forma de identificar e documentar os possíveis ataques em um sistema de uma forma estruturada e hierarquizada. A estrutura em árvore fornece uma imagem detalhada de vários ataques que podem ser empregados para comprometer um sistema. Com o uso de árvores de ataque, cria-se uma representação reutilizável de problemas de segurança que ajudará a focar os esforços de mitigação de ameaças.

Schneier (1999) introduziu o conceito de árvores de ataque como uma forma de modelo de ameaças contra os sistemas computacionais. A ideia do conceito é compreender as ameaças para um sistema, descrevendo em uma estrutura de árvore as diferentes formas pelas quais o sistema pode ser atacado e comprometido. O nó raiz da árvore representa o objetivo de um atacante, ou o ativo/bem a ser protegido. Os nós folhas representam as diferentes formas de alcançar esse objetivo. Com base na estrutura da árvore e na informação de quem são os atacantes e das capacidades que eles possuem, seremos capazes de estabelecer as contramedidas adequadas e lidar com as verdadeiras ameaças.

Moore, Ellison e Linger (2001) descrevem um meio para documentar ataques de segurança de informações de uma forma estruturada e reutilizável, usando árvores de ataque. O trabalho aborda diversas questões sobre o uso de árvores de ataque para a documentação das vulnerabilidades.

Mauw e Oostdijk (2006) apresentam uma sugestão para formalizar os conceitos de árvore de ataque. É descrito o conceito de conjunto de ataque como um nível de abstração de uma árvore de ataque. É sugerida uma semântica formal para descrição de como as árvores de ataque podem ser manipuladas durante a sua construção e análise. Os autores argumentam que a execução dos trabalhos de semântica e formalização do método é necessária para se tornar possível a construção de ferramentas automatizadas.

Khand (2009) descreve que os tipos de nós das árvores de ataque convencionais não são suficientes para representar adequadamente os ataques para sistemas tolerantes à intrusão. São apresentados novos tipos de nós, como nós prioritários, nós que representam a quantidade de subobjetivos que devem ser alcançados para os ataques obterem sucesso, nós que possuem condições para ocorrerem, nós que forçam uma determinada sequência e nós que são usados para modelar ataques que mudam com o tempo. São apresentados três modelos de árvores de ataque para sistemas de segurança: Servidor Seguro Distribuído tolerante a intrusão, Servidor de Dados e Servidor de Aplicação com arquitetura tolerante a intrusão.

Além dos trabalhos citados anteriormente, destacamos um trabalho relacionado à validação de segurança de aplicações utilizando árvore de ataque. Em Li & He (2008), é usado um modelo de árvore de ameaças estendido, baseado em dados estatísticos históricos, para tratar dos aspectos de segurança durante o projeto de desenvolvimento da aplicação.

No nosso trabalho as árvores são usadas para representar as vulnerabilidades em aplicações web e as formas para explorá-las, diferente da proposta de Li & Hi que é dedicada à mitigação das ameaças ainda no estágio de desenvolvimento da aplicação. O modelo de árvore que propomos também se difere dos demais em relação à profundidade da árvore, que relaciona também as formas de ataque a algumas das principais vulnerabilidades de aplicações web bem como uma forma de ataque de "baixo nível", que pode ser verificada por ferramentas de testes de segurança. A árvore é ainda divida em faixas que facilitam a identificação de qual é o objetivo de ataque (o que), o local onde realizar o ataque (onde) e a forma para realizar o ataque (como).

2.2. Mutação de Código

Teste de software é o processo de investigação do software com o objetivo de encontrar seus defeitos (Myers, 2004). Testes por Mutação de Código ou Análise de Mutantes consistem num método de teste de software que realiza a injeção de falhas em um programa, através da modificação de pequenos trechos do código fonte, gerando, consequentemente, mutantes desse programa. As mudanças, chamadas de mutações, baseiam-se em operadores de mutação bem definidos que imitam erros típicos de programação (tais como o uso errado de operador ou nome de variável), ou forçam a criação de testes (tais como levar cada expressão a zero). O objetivo é ajudar o testador a: (i) desenvolver testes eficazes e (ii) localizar falhas nos dados utilizados para testar o programa, ou em seções de código que são raramente ou nunca acessadas durante a execução. O critério Análise de Mutantes foi originalmente proposto por DeMillo, Lipton e Sayward (1978).

Em Shahriar (2008) é apresentado um trabalho que faz uso da técnica de mutação para testar *bugs* que causam negação de serviço em aplicações. Foi desenvolvido um protótipo de ferramenta de teste que gera oito diferentes mutantes e os analisam em quatro programas de código fonte aberto. Giacometti et al. (2002) apresentam um conjunto operadores de mutação para validação de aplicações paralelas que utilizam o ambiente de passagem de mensagens PVM – *Parallel Virtual Machine*. A nossa proposta se difere destas anteriores por usar operadores de mutação concebidos de forma a sempre inserir uma vulnerabilidade. Na mesma linha, Vicenzi et al. (1999) propõem a redução dos custos na realização de testes por mutação através da criação de operadores essenciais de mutação. A aplicação dos testes é apoiada por uma ferramenta para seleção do conjunto de operadores essenciais.

3. Metodologia

A metodologia proposta neste trabalho para a validação de ferramentas de testes de segurança para aplicações web envolve: (i) a criação da árvore de ataque com as principais vulnerabilidades em aplicações web, (ii) a definição e escolha das ferramentas de testes que serão utilizadas para cobrir as vulnerabilidades derivados das árvores de ataque, (iii) a execução dos testes e (iv) a validação dos testes através de técnicas de mutação de código. A figura 1 apresenta as etapas do fluxo de validação proposto pela metodologia.

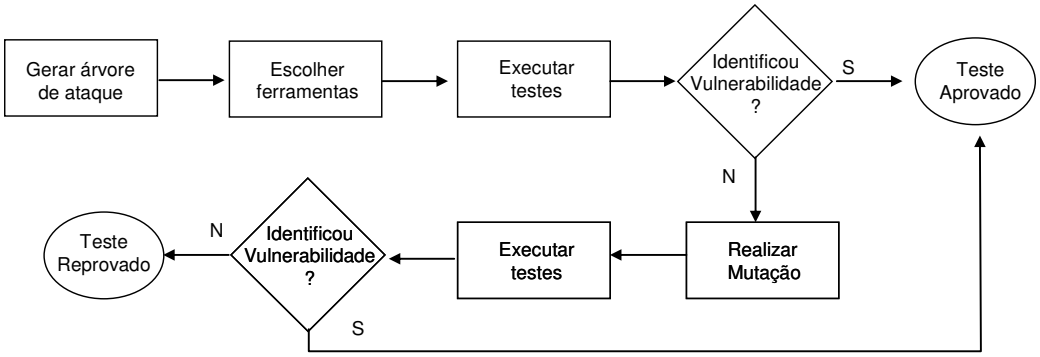

Figura 1 - Fluxo de Validação dos Testes de Segurança

Segundo essa metodologia, gera-se inicialmente uma árvore de ataque específica para um escopo de aplicações. Esse escopo é baseado: no objetivo de ataque, na linguagem empregada na programação, no banco de dados utilizado, na infraestrutura necessária para suportar a execução, no ambiente de produção etc. A árvore gerada para esse escopo é derivada de uma árvore de ataque genérica para aplicações web e está associada aos objetivos de ataque e às vulnerabilidades e possibilidades de ataques referentes ao escopo da aplicação.

Em seguida, considerando de um lado o roteiro gerado pela árvore de ataque, e por outro, as ferramentas de testes de segurança disponíveis, é efetuada a escolha das ferramentas de teste mais apropriadas. As ferramentas são escolhidas de forma a cobrir todas as vulnerabilidades descritas pela árvore.

Após a escolha das ferramentas (ou mesmo scripts de testes) que serão utilizados, os testes são executados na aplicação original. Caso sejam encontradas vulnerabilidades na aplicação testada, obtém-se um indicativo de que os testes de vulnerabilidades cobrem as vulnerabilidades para as quais se propõem. Em caso contrário, ou seja, se após a execução desses testes não forem encontradas as vulnerabilidades listadas pela árvore de ataque do escopo da aplicação, é necessário efetuar a validação dos testes executados, inserindo intencionalmente vulnerabilidades no código fonte da aplicação e executando os testes novamente.

A inserção de vulnerabilidades no código fonte é feita através da técnica de mutação de código. Após a geração do código mutante, executam-se novamente os testes selecionados no código modificado, de forma a verificar se as vulnerabilidades incluídas propositalmente pela mutação de código serão agora detectadas pelos testes de segurança.

Caso os testes identifiquem as vulnerabilidades inseridas, tem-se um indicativo de que as ferramentas de teste de segurança empregadas na cobertura das vulnerabilidades representadas pela árvore de ataque estão aptas a serem utilizados. Caso os testes não identifiquem as vulnerabilidades incluídas pela mutação de código, temos um indicativo de

que as ferramentas não cobrem adequadamente as técnicas de ataque mapeadas pela árvore de ataque do escopo da aplicação.

3.1. Geração da árvore de ataque

A elaboração da árvore de ataque para um escopo de aplicações é composta de duas etapas: (i) geração de uma árvore de ataque genérica para aplicações web, (ii) e adequação dessa árvore aos objetivos específicos da validação, através da redução dos nós e caminhos que não são necessários ou estão fora do escopo da validação. A árvore de ataque genérica auxiliará a elaboração da árvore de ataque específica para o escopo da aplicação que será utilizada na validação dos testes de segurança.

3.1.1. Semântica adotada

A semântica adotada segue a proposta de Schneier (1999). O modelo básico descreve a árvore de ataque com dois tipos diferentes de nós, nós AND e nós OR. Nos nós do tipo OR, pelo menos um subobjetivo deverá ser satisfeito para atingir o objetivo do nó raiz. Nos nós do tipo AND, o objetivo do nó só é atingido quando cada subobjetivo for satisfeito. Valores podem ser atribuídos aos nós folha, como declarações booleanas do tipo "possível/impossível" ou "exige equipamentos especiais/não exige equipamentos especiais". Para se obter o resultado de um nó, cálculos booleanos podem ser realizados sobre os valores dos seus nós subobjetivos baseado na expressão booleana do nó (AND ou OR). Os princípios básicos são mostrados no exemplo de árvore da Figura 2. O exemplo ilustra uma árvore de ataque contra um cofre trancado. Expressões OR são aplicadas na árvore através da ligação dos subobjetivos ao nó resultante. Expressões AND são representadas graficamente por um arco ligando os dois subobjetivos, acrescido da palavra "AND" no interior do arco. Realizando os cálculos booleanos, podemos concluir que as formas possíveis de "Abrir um cofre" seriam as representadas pelos caminhos em que os nós folhas estão destacados. A opção do nó "Descobrir a combinação", por exemplo, só seria possível através do nó folha "Subornar", visto que "Ameaçar" e "Chantagear" são impossíveis e apesar de "Ouvir a Conversa" ser possível, "Chegar até a combinação" é impossível, tornando o nó "Escutar atrás da porta" impossível para se alcançar a "Combinação do alvo".

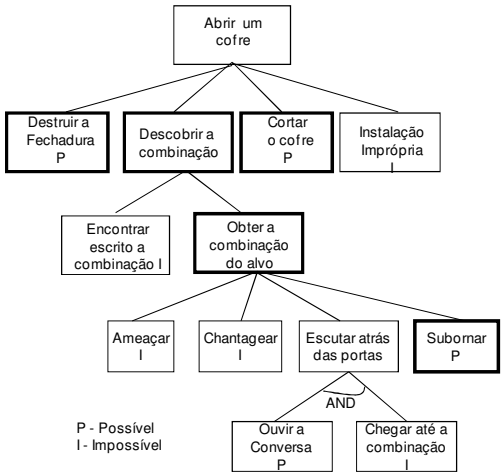

Figura 2 - Exemplo de árvore de ataque

No modelo de Schneier, valores não booleanos indicando o esforço de um ataque (custo ou tempo) também podem ser atribuídos a um nó. Entretanto, os cálculos realizados sobre esses valores são limitados ao: (i) somatório dos valores dos subnós no caso de nós AND, e (ii) valor do subnó com custo mais baixo no caso de nós OR.

3.1.2. Preenchimento

Para geração da árvore, inicialmente são identificados os ativos mais importantes da aplicação e, em seguida, é efetuada uma análise crítica sobre eles. Assim, buscam-se definir quais são os ativos essenciais e quais são as principais interações envolvidas. Um exemplo prático na seleção de ativos é analisar os processos de negócio e verificar os serviços fornecidos pelo sistema. Algumas perguntas auxiliam na definição dos ativos que serão os objetivos de ataque, por exemplo: (i) quais são as principais preocupações dos clientes que vão utilizar o serviço e o que os deixa confiantes em relação à segurança? (ii) quais são os principais ativos e informações do sistema? (iii) quais ativos são cruciais para que o sistema possa oferecer o serviço proposto?

Os ativos identificados nas respostas dessas perguntas serão os objetivos de ataques dos testes, (o que). Para cada objetivo de ataque selecionado é necessário identificar também quais as técnicas de ataques podem ser utilizadas para alcançá-los (onde) e quais as formas como eles podem ser atingidos (como). Na árvore de ataque de um ativo escolhido, a raiz da árvore, representa o objetivo de ataque (o que), identificados como mais importantes para uma aplicação. As folhas da árvore representam as vulnerabilidades que podem ser exploradas para alcançar o objetivo de ataque, bem como as possíveis formas de ataque. Elas são preenchidas com base nas principais vulnerabilidades divulgadas pelas comunidades de segurança em aplicações web. Uma fonte importante a ser considerada é o OWASP Top Ten (2007) e o OWASP Testing Guide (2007).

As árvores são desenhadas de cima para baixo, começando pelo objetivo de ataque selecionado. Os nós que representam os subobjetivos são colocados hierarquicamente sob o nó superior, e conectados com linhas para o nó anterior, com a descrição do evento, em uma cadeia de acontecimentos que conduzem a um ataque. Dessa forma, nós folhas ligados em uma cadeia até o objetivo de ataque de mais alto nível representam cadeias de eventos que conduzem a um possível ataque. Quando dois ou mais nós estão associados a um nó ascendente na árvore, dois procedimentos lógicos diferentes podem ocorrer.

O primeiro procedimento lógico ocorre quando dois ou mais nós do tipo folhas têm links que levam ao mesmo nó, um nível acima na hierarquia. Nesse caso, aplica-se entre os elementos o operador lógico OR. Por exemplo, na árvore de ataque da figura 3, “Acessar o banco de dados diretamente” ou “Acessar a aplicação como usuário privilegiado” resultam de “Acessar os dados da aplicação”.

O segundo procedimento lógico ocorre quando dois ou mais nós que não são do tipo folha têm links que levam ao mesmo nó, um nível acima na hierarquia, e um arco liga as duas linhas com uma notificação "AND". Nesse caso, aplica-se entre os elementos o operador lógico AND; por exemplo, na árvore de ataque da figura 3, tanto “Manter um ID válido” como “Modificar o ID do *cookie* do usuário” precisam ser realizados para “Fixar uma sessão”, caso tenha-se optado por seguir esse caminho de ataque.

Através de uma árvore, é possível também realizar a análise de um sistema ou subsistema em situações e ambientes específicos. Isso torna possível dividir os ataques descritos em um conjunto de possíveis caminhos, tornando um objetivo de ataque de alto nível em vários subataques, que são direcionados aos seus subobjetivos. Dessa forma, a geração da árvore de ataque genérica fornecerá uma ferramenta estrutural que poderá ser utilizada para descobrir eventuais vulnerabilidades em uma aplicação específica.

A possibilidade de projetar e analisar as árvores de ataque em diferentes níveis de profundidade e em áreas selecionadas de interesse e criticidade permite a escolha dos testes

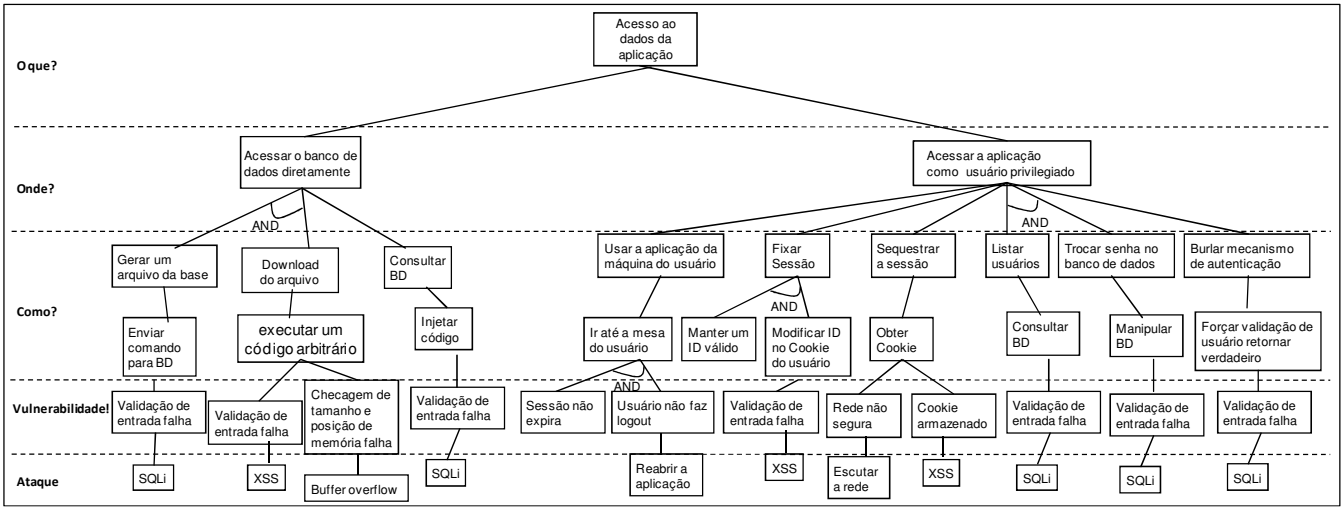

Figura 3 - Árvore de ataque genérica para o ativo "Dados da aplicação"

que serão executados e a definição de quais caminhos serão seguidos para alcançar o objetivo de ataque. Cada aplicação específica que utilizar a árvore genérica como base, deverá considerar o escopo da aplicação na seleção das alternativas de caminhos da árvore para alcançar o objetivo de ataque.

3.2. Escolha das ferramentas de teste

A escolha das ferramentas que serão utilizadas nos testes é feita a partir da análise das vulnerabilidades listadas na árvore de ataque específica do escopo da validação. Após listar as vulnerabilidades citadas e as possíveis formas de ataque, selecionam-se as ferramentas e os possíveis scripts maliciosos que se propõe a detectar essas vulnerabilidades.

Algumas ferramentas possuem parâmetros e itens de configuração que precisam ser ajustados de acordo com o contexto. Elas devem ser configuradas conforme as orientações dos fabricantes e devem ser as mesmas tanto nos testes executados no código original como no código mutante. Após a escolha e configuração das ferramentas são executados os testes no código original. Se os testes identificarem no código original as vulnerabilidades pesquisadas, temos um indicativo de que o teste é valido para identificá-las dentro escopo definido. Caso não seja identificado as vulnerabilidades pesquisadas é preciso validar a eficácia dos testes. Essa validação é feita através da mutação de código.

3.3. Geração da Mutação

Utilizando-se da técnica de testes por análise de mutantes, ou mutação de código, pode-se validar os testes de segurança que foram executados no código original e que não conseguiram identificar as possíveis vulnerabilidades pesquisadas.

A validação inicia-se com o mapeamento das vulnerabilidades definidas na árvore de ataque e em seguida com uma pesquisa em busca de operadores de mutação que irão realizar a inserção dessas vulnerabilidades no código. Para cada vulnerabilidade ou um conjunto delas, teremos um ou mais operadores de mutação.

As regras dos operadores de mutação indicam as mudanças que devem ser realizadas no código original para inserir uma determinada falha. No entanto, os geradores automáticos de mutação de código conhecidos possuem operadores de mutação bastante simples, como por exemplo, trocar ‘&&’ por ‘||’ ou ‘==’ por ‘>=’. Esses tipos de operadores dificilmente vão incluir uma vulnerabilidade de segurança em uma aplicação.

Para que os operadores de mutação gerados realmente incluam vulnerabilidades de segurança no código, é preciso pesquisar pelas vulnerabilidades conhecidas e divulgadas pelas comunidades de segurança afins, bem como a forma como elas são corrigidas. Dessa forma é possível gerar operadores de mutação capazes de localizar e modificar trechos no código original que possuem as características de códigos escritos de forma correta, de modo a torná-los vulneráveis. Isso torna esses operadores fortemente dependentes do contexto da aplicação. Espera-se, em longo prazo, que da mesma forma que existem hoje comunidades e sites que publicam as principais vulnerabilidades em aplicações web, também possa ser criado comunidades e sites que divulguem operadores de mutação provocadores das principais vulnerabilidades.

Um dos grandes problemas dos testes por mutação é saber quando um mutante é equivalente (não altera o funcionamento de um código). Segundo Offutt (1994), a detecção de mutantes equivalentes é um dos maiores obstáculos para a prática de utilização de testes por mutação. O esforço necessário para verificar se os mutantes são equivalentes pode ser muito elevado, mesmo para pequenos programas. Este trabalho assume o uso de operadores de mutação que gerem apenas mutantes não equivalentes, ou seja, que produzem vulnerabilidades reais, eliminando a necessidade de análise dos mutantes para identificar os equivalentes.

Após a geração dos códigos mutantes, cada um deles é testado com o mesmo conjunto de ferramentas de testes a que o código original foi submetido.

Se algum dos testes produzir um resultado diferente do resultado no código original, temos que este teste foi capaz de distinguir o mutante do código original, ou seja, foi capaz de identificar a vulnerabilidade inserida. Neste caso, a ferramenta pode ser validada como eficaz para o escopo definido.

Se nenhum dos testes pôde matar o mutante, ou seja, se não conseguiram identificar a vulnerabilidade inserida, o mutante é dito vivo, indicando que o teste executado pela ferramenta (ou pelo script) não é adequado para identificar a vulnerabilidade mapeada naquele escopo.

4. Validação Experimental

Para validação experimental da metodologia proposta, escolhemos uma aplicação web como estudo de caso. A aplicação *Engajar* faz parte do projeto *Turma Cidadã,* patrocinado pelo CEFET-RJ. Ela foi construída no framework de desenvolvimento para aplicações web *Ruby on Rails*. Para guardar os dados por ela manipulados, é utilizado um sistema de banco de dados MySql. Descreveremos algumas das vulnerabilidades referentes ao escopo da aplicação *Engajar* e que serão determinantes para a geração da árvore de ataque específica desta aplicação.

4.1. Vulnerabilidades existentes no escopo da aplicação

Uma das ameaças mais comuns e devastadoras para aplicações web é a injeção, categoria de ataque que introduz códigos maliciosos ou parâmetros em uma aplicação web, de forma a executar esses códigos dentro do contexto de segurança da aplicação (OWASP Ruby Guide, 2009). Os exemplos mais proeminentes de injeção são o *Cross Site Scripting* (XSS) e o *SQL injection*. Conforme descrito na figura 3, podemos observar na árvore de ataque genérica que a vulnerabilidade de injeção de comandos SQL pode ser utilizada para alcançar vários subobjetivos de ataque.

Apesar de o Ruby on Rails possuir métodos inteligentes, as aplicações construídas nesse framework também podem ter problemas com *SQL Injection,* caso os desenvolvedores não tomem certos cuidados (OWASP Ruby Guide, 2009).

Os ataques de injeção de SQL visam alterar as consultas realizadas ao banco de dados, manipulando os parâmetros de uma aplicação web. Um objetivo muito conhecido de ataques de *SQL Injection* é realizar uma manipulação ou leitura arbitrária de dados.

Outro objetivo é burlar os mecanismos de autorização. Normalmente, uma aplicação web possui um controle de acesso. O usuário digita suas credenciais de *login* e a aplicação tenta localizar o correspondente registro na tabela usuários. A aplicação concede o acesso quando ela encontra o registro. No entanto, um invasor pode burlar esta verificação com a injeção de comandos SQL (OWASP Ruby Guide, 2009).

Se um usuário mal intencionado entrar com `'OR'1' ='1` no campo *nome* e `'OR'2'>'1` no campo *senha*, a consulta que a aplicação enviará para o banco ficaria assim:

```
SELECT * FROM users WHERE login = '' OR '1'='1' AND password = '' OR '2' > '1' LIMIT 1
```

Esta consulta localiza o primeiro registro no banco de dados, concedendo o acesso a este usuário.

4.2. Geração da árvore de ataque

Para geração da árvore de ataque específica o ativo crítico selecionado foi: “Dados Cadastrais dos Interessados”. Esses dados são armazenados no banco de dados da aplicação e foram considerados críticos porque são informações pessoais dos usuários do sistema, e estes não autorizaram a divulgação dos mesmos. Esse ativo será o objeto de ataque da árvore. A partir do ativo crítico selecionado, definimos um objetivo de ataque para esse ativo e iniciamos o trabalho de listar as formas de como seria possível alcançá-lo. O objetivo é acessar a base de cadastro dos interessados. Baseado na árvore de ataque genérica para aplicações web, nas vulnerabilidades conhecidas do framework *Ruby on Rails* e nas características referentes ao banco de dados, levantamos as possíveis vulnerabilidades e as suas formas de serem exploradas, listando os respectivos ataques. Para cada forma diferente foi criado um nó folha no subnível abaixo. A árvore de ataque do escopo de validação das ferramentas de testes deste estudo de caso foi resumida à árvore da figura 4.

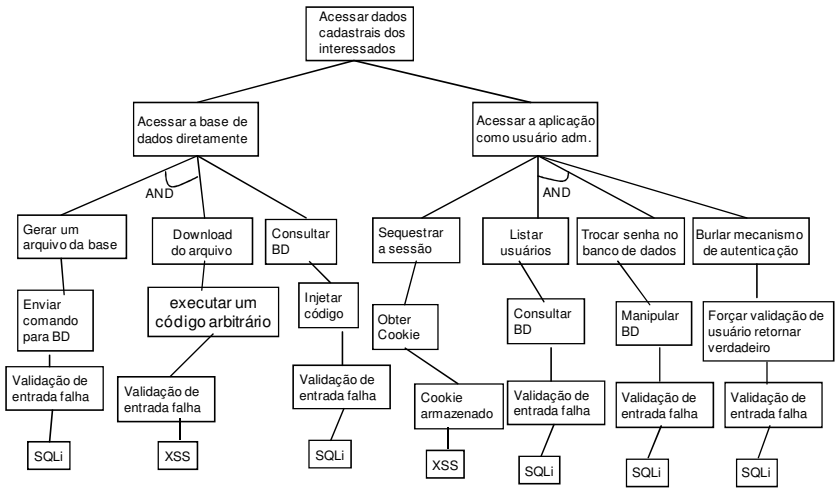

Figura 4 - Árvore de ataque do ativo "Dados cadastrais dos interessados”

4.3. Escolha das ferramentas

Considerando, de um lado, o roteiro gerado pela árvore de ataque, e de outro, as vulnerabilidades cobertas e declaradas pelas ferramentas de testes de segurança disponíveis, foi efetuada a escolha das ferramentas de testes que serão utilizadas.

Para cobrirmos os ramos da árvore de ataque, escolhemos três ferramentas para testes de segurança. Elas realizam testes que cobrem as vulnerabilidades de injeção de dados, por exemplo, ataques de SQL *Injection* e XSS. Duas das ferramentas escolhidas são de uso livre. A ferramenta Pblind se propõe a testar vulnerabilidades de SQL *Injection*. Ela faz parte de um "Kit" de ferramentas para teste de invasão de uso livre, chamado BackTrack 4. A segunda ferramenta, chamada w3af, também de uso livre, é integrante do "Kit" de ferramentas da fundação OWASP (Owasp Live CD, 2009), e se propõe a testar as principais vulnerabilidades em aplicações web, inclusive as de SQL *Injection* e XSS. A terceira ferramenta, chamada NStalker, é uma ferramenta comercial que se propõe a testar tanto vulnerabilidades de SQL *Injection* e XSS como outras vulnerabilidades em aplicações web e na infra-estrutura que as suportam. As ferramentas foram escolhidas de forma a cobrir todas as vulnerabilidades descritas pela árvore. Elas foram configuradas para procurar por qualquer tipo de vulnerabilidade, no entanto o escopo deste estudo de caso é a validação dos testes para as vulnerabilidades de injeção de dados, o que não impede o uso da metodologia na validação dos demais testes.

4.4. Testes iniciais

Inicialmente, foi executado cada teste na aplicação original. As ferramentas foram configuradas conforme orientação dos fabricantes. A ferramenta de uso comercial, apesar de não ter identificado vulnerabilidades de falhas de injeção, informou haver outras vulnerabilidades fora do escopo dessa validação. As ferramentas de uso livre executaram a varredura e também não identificaram vulnerabilidades relacionadas a falhas de injeção.

4.5. Geração do código mutante

Seguindo os passos da metodologia, vamos inserir intencionalmente as vulnerabilidades que estão sendo objeto dos testes, ou seja, aquelas indicadas pela árvore de ataque. Para isso foi identificado as vulnerabilidades mapeadas e elaborado uma lista de operadores de mutação para alterar o código original, com intuito de incluir essas vulnerabilidades. Foi pesquisado pelas vulnerabilidades relativas ao escopo da aplicação e a forma como elas são corrigidas. Assim geramos operadores que procuram no código fonte por características de código escrito de forma correta e os modificam, tornando-o vulnerável. Dessa forma, foi possível gerar operadores que realmente incluíssem vulnerabilidades na aplicação e não apenas erros comuns. Foram criados três operadores de mutação que se propõem a inserir de forma automática vulnerabilidades relacionadas à falha de injeção, removendo as funções de validação ou proteção dos dados de entrada. A vulnerabilidade de "validação de entrada falha" permite ataques de injeção de comando no banco de dados, *SQL injection*, e ataques de execução de *script* na aplicação, *Cross Site Scripting* (XSS). Esses ataques possibilitam percorrer os possíveis caminhos de ataque representados pela árvore de ataque específica gerada para a validação experimental. A tabela 1 apresenta os operadores de mutação que serão utilizados.

Tabela 1 - Operadores de mutação

m	Cod.Operador	Descrição do Operador
1	Mod_rec_param	Moficar a forma como algumas variáveis recebem dados fornecido por usuários
2	Rem_AttrProtected	Comentar as linhas que possuem funções para ativar proteção de variáveis importantes
3	Inibe_Validates	Desabilita as funções de validação de valores de variáveis

Os operadores de mutação apresentados atuam, respectivamente, da seguinte forma:

(1) Procuram no código fonte original por variáveis que tem seus valores atribuídos através do caractere "?" em vez de ser atribuído diretamente da string digitada pelo usuário, e modificam o código para que elas tenham seus valores atribuídos diretamente;

(2) Procuram no código fonte original as variáveis que possuem algum tipo de proteção pela função "*Attr_Protected*", como por exemplo, *update_attributes* ou *new(attributes)*. O operador comenta as linhas que ativam essa proteção;

(3) Procuram no código fonte original as funções que fazem a validação de entrada dos dados, inseridos pelos usuários na aplicação, e as desabilita, por exemplo, inserindo uma função que determina se a validação será executada ou não, e forçando o seu resultado negativo;

Para este conjunto de operadores de mutação foi gerado um mutante *m*. Iremos aplicar no mutante os mesmos testes executados inicialmente e analisar o resultado da aplicação dos mesmos.

4.6. Teste no código mutante

Após a realização das mutações de código, objetivando inserir as vulnerabilidades que estão sendo procuradas, repetimos os testes realizados antes das mutações. Foram executados os mesmos testes e as configurações das ferramentas foram mantidas para podermos comparar os resultados obtidos.

4.7. Resultados

Os resultados da massa de testes, no código original e posteriormente no código mutante, nos permitem afirmar que os testes executados por duas das ferramentas escolhidas, quando aplicados no código original, foram eficazes para identificar as vulnerabilidades dos ramos da árvore de ataque selecionados para atingir o objetivo de ataque descrito na raiz da árvore ("Acessar Dados Cadastrais dos Interessados"). No entanto, essa validação de ferramentas e testes está condicionada ao escopo das tecnologias empregadas na aplicação testada, ou seja, um determinado teste ou ferramenta será validado para o escopo delimitado pela árvore de ataque utilizada. Nesse caso, o escopo da árvore foi determinado com base no framework utilizado no desenvolvimento da aplicação, banco de dados usado no armazenamento dos dados manipulados e no servidor de aplicação. Ao analisarmos cada teste ou ferramenta individualmente, podemos perceber que nem todos foram capazes de matar todos os mutantes, conforme podemos observar na tabela 2.

Considerando todos os testes manuais e as ferramentas utilizadas como um conjunto de teste único, a validação para este conjunto de teste, na identificação das vulnerabilidades de "validação de entrada falha", que podem ser explorada pelos ataques do tipo SQL *Injection* e *Cross Site Scripting,* foi garantida. A validação foi garantida uma vez que ao aplicar o conjunto de teste no código original não foi detectada nenhuma vulnerabilidade e ao aplicar o mesmo conjunto de teste nos códigos que sofreram mutação para inserção de vulnerabilidades, o conjunto de teste demonstrou que a aplicação está vulnerável a esses tipos de ataque.

Tabela 2 - Resultado dos testes com cada ferramenta

	Ferramentas de Testes de Segurança				
	Pblind (SqlI)	w3af (SqlI)	w3af (XSS)	Nstalker (SqlI)	Nstalker (XSS)
Mutante 1	S	N	N	N	S
Mutante 2	S	N	N	N	S
Mutante 3	S	N	N	N	S
Qtde Mutantes Mortos por ferramenta/testes	3	0	0	0	3

Nos casos dos testes das ferramentas que tiveram a quantidade de mutantes mortos igual a zero, ou seja, não foram capaz identificar nenhuma vulnerabilidade inserida, esses testes precisam ser redefinidos e/ou as ferramentas re-configuradas ou corrigidas, de forma a poderem identificar as vulnerabilidades para as quais eles se propõem.

5. Conclusão

Este trabalho buscou a criação de uma metodologia para a avaliação dos testes de segurança em aplicações web. O emprego de árvore de ataques permite um verdadeiro mapeamento estruturado do universo de possíveis vulnerabilidades, de forma a construir uma análise sistemática dos elementos de maior criticidade. Na realização do estudo de caso ficou evidente a importância de definir os cenários de avaliação do teste, de forma a efetivamente cobrir os pontos mais vulneráveis. O uso de técnicas de mutação de código é sem dúvida um potente aliado para realizar, de forma automatizada, a inserção de vulnerabilidades em uma aplicação. É patente que para cada vulnerabilidade e para cada ambiente de desenvolvimento será necessário construir um conjunto de operadores de mutação específico. Porém, uma vez criado um conjunto de operadores para um dado ambiente, é relativamente fácil instanciá-lo para outro ambiente.

Os resultados obtidos surpreenderam: de um conjunto de três ferramentas analisadas, uma não cobriu totalmente e outra, parcialmente, os pontos definidos como críticos, o que reforça a importância de uma contribuição no sentido de averiguar a real eficácia das ferramentas de avaliação de segurança de aplicativos. A repetição sistemática (reprodutibilidade) dessa abordagem para outros pontos críticos permite a posterior construção de um perfil de abrangência de cada ferramenta de teste.

A apresentação dos resultados, a partir da proposta de avaliação experimental, contribuiu para verificar a viabilidade do uso da metodologia proposta. Os resultados gerados foram considerados úteis para determinar o sucesso da metodologia proposta.

Os trabalhos futuros incluem a investigação sobre a viabilidade de automação das diversas etapas abordadas na metodologia; a extensão do escopo de validação dos testes, ampliando a árvore de ataque genérica para outros ativos críticos de aplicações web; a elaboração de novos operadores de mutação e testes para cobrir toda a nova árvore de ataque; e o desenvolvimento de uma ferramenta para realizar as mutações de código, com intuito de inserir, de forma automática, as vulnerabilidades de segurança no código.

6. Referências

CVE, Common Vulnerabilities and Exposures. Disponível em: http://www.cve.mitre.org, 2006. Acesso em 23/08/2009.

Demillo, R.A.; Lipton, R.J.; Sayward, F.G. Hints on Test Data Selection: Help for the Practicing Programmer. IEEE Computer, abril, 1978.

Fonseca, J.; VIEIRA, M.; Mapping Software Faults with Web Security Vulnerabilities. IEEE/IFIP International Conference on Dependable Systems and Networks, June 2008.

Fonseca, J. et al. Vulnerability & Attack Injection for Web Applications, 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2009.

Giacometti, C. et al.; Teste de Mutação para a Validação de Aplicações Concorrentes usando PVM, REIC, Eletrônica de Iniciação científica, v.II, n.III, 2002.

Guidetti, S.A. Aplicação de Análise de Mutantes à Geração de Dados de Teste para Detecção de Vulnerabilidade do Tipo Buffer Overflow, Universidade Estadual de Campinas, Campinas, Brasil, 2005.

Hoglund, G. et al; Exploiting software: how to break code, Pearson, 2006.

Huang,Y.W.; Huang, S.K.; Lin,T.P.; Tsai, C.H. Web application security assessment by fault injection and behavior monitoring; 12th international conference on World Wide Web, Budapest, Hungary pp. 148 – 159, 2003.

Khand, P.A. System level Security modeling using Attack trees. IEEE; 2nd International Conference on Computer, Control and Communication, 2009.

LI, X.; He, K. A Unified Threat Model for Assessing Threat in Web Applications, International Conference on Information Security and Assurance, pp. 142-145, 2008.

Mauw, S.; Oostdijk, M. Foundations of Attack Trees. LNCS, 3935, 186–198, 2006.

Moore, A. P.; Ellison, R. J.; Linger, R. C. Attack Modeling for Information Security and Survivability. Technical report, Carnegie Mellon University, 2001.

Myers, G. J., The Art of Software Testing, 2º ed., New Jersey: John Wiley & Sons, 2004.

Offutt, A.J. A practical system for mutation testing: help for the common programmer, Test Conference, 1994. Proceedings, International, Washington, DC, USA.

OWASP (Open Web Application Security Project) Live CD Project, 2009. Disponível em: <http://www.owasp.org/index.php/Category:OWASP_Live_CD_Project>. Acesso em: 15/dez/2009.

OWASP (Open Web Application Security Project) Ruby on Rails Security Guide V2, 2009. Disponível_em:<http://www.owasp.org/index.php/Category:OWASP_Ruby_on_Rails_Security_Guide_V2> Acesso em: 15/dez/2009.

OWASP (Open Web Application Security Project) Testing Guide v2, 2007. Disponível em: <http://www.owasp.org/index.php/Category:OWASP_Testing_Project> Acesso em: 15/dez/2009.

OWASP (Open Web Application Security Project) Top Ten, 2007. Disponível em: <http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project> Acesso em: 15/dez/2009.

Schneier, B. Attack Trees, 1999. Disponível em: <http://www.schneier.com/paper-attacktrees-ddj-ft.html>, Dr. Dobb's Journal. Acesso em 13/mai/2009.

Shahriar, H.; Zulkernine, M. Mutation-based Testing of Format String Bugs, 11th IEEE High Assurance Systems Engineering Symposium, 2008.

Taquary C.; Uso de Árvores de Ataque e Técnicas de Mutação de Código na Segurança de Aplicações Web, NCE/UFRJ, 2010.

Vicenzi, A.M.R et al; Operadores Essenciais de Interface: Um Estudo de Caso, USP/UEM, 1999. Disponível em < http://www.inf.ufsc.br/~sbes99/anais/SBES-Completo/34.pdf>, Acesso em 12/ago/2009.