# 4
# Avaliação Quantitativa de Segurança de Sistemas

Neste capítulo são descritos diferentes métodos para a avaliação quantitativa de segurança de sistemas existentes na literatura. A ordem em que os métodos são apresentados considera o ano em que foi publicado. Com isso, podemos ter uma idéia da evolução dos métodos de avaliação quantitativa de segurança.

## 4.1 Esforço despendido por um atacante

Brocklehurst e co-autores (Brocklehurst et al., 1994) desenvolveram uma teoria com a intenção de trabalhar métricas de segurança operacional de forma similar àquelas que hoje são empregadas para medir a confiabilidade de software e de hardware.

A confiabilidade do software é a probabilidade de um sistema operar sem apresentar defeitos, sob certas condições, em um determinado intervalo de tempo. Expressamos a confiabilidade em uma escala de 0 a 1. Um sistema altamente confiável terá uma medida de confiabilidade próxima de 1, e um sistema que não é confiável terá uma medida próxima de zero. A confiabilidade é medida durante o tempo de execução e não durante o tempo real (ou seja, o tempo medido pelo relógio), de modo a refletir mais precisamente o uso do sistema. Uma função de densidade de probabilidade *f* do tempo *t*, escrita como *f(t)*, que descreve a nossa compreensão sobre a freqüência esperada com que o software falhará, pode ser usada quando modelamos um defeito de software. Nem toda função de densidade é uniforme, e um dos problemas que torna difícil entender e medir a confiabilidade é obter o comportamento da falha em uma função apropriada de densidade de probabilidade. Isto é, podemos definir uma função *f(t)* e utilizá-la para calcular a probabilidade de um componente de software falhar, em determinado intervalo de tempo [$t_1$, $t_2$]. Como essa probabilidade é a área abaixo

das curvas entre os limites do intervalo, a probabilidade de um defeito entre $t_1$ e $t_2$ é

$$\int_{t_2}^{t_1} f(t)\, \mathrm{dt}$$

Especificamente, a função de distribuição *F(t)* é o valor dessa integral no intervalo de 0 até *t*. *F(t)* é a probabilidade de que o software falhe antes do tempo *t*, e definimos a função de confiabilidade *C(t)* como sendo $1 - F_i(t)$, que é a probabilidade de que o software funcione apropriadamente até o tempo *t*.

De forma análoga à medição da confiabilidade do software, agora considere a segurança de um sistema em um determinado instante de tempo. Segundo Brocklehurst e co-autores (Brocklehurst et al., 1994), se aceitarmos que esforço em segurança pode ser representado da mesma forma como é feita em confiabilidade (em inglês: *reliability*), então, é razoável considerar a distribuição do "esforço para vencer uma brecha" como análoga à distribuição de tempo para falha de confiabilidade. Assumindo que podemos obter uma variável apropriada *E*, variável randômica, para representar este esforço, a segurança operacional poderia, em princípio, agora ser expressa em termos de probabilidade: por exemplo, significar o esforço para a próxima brecha de segurança, probabilidade de resistir com sucesso a um ataque envolvendo um determinado gasto de esforço, etc. Considerando, então, a distribuição de probabilidade do esforço requerido (variável randômica), *E*, para a próxima violação do sistema. A função segurança, por analogia com a função de confiabilidade, é dada por:

$$C = C(e) = \text{Probabilidade}\ (E > e)$$

A função distribuição e a função distribuição de probabilidade (se existe) de esforço para a próxima violação são respectivamente:

$$F(e) = 1 - C(e),\ f(e) = F'(e)$$

A taxa de violações de segurança (número esperado de violações em um determinado período de tempo) pode ser então definida como $v(e) = f(e)/C(e)$

**Vantagens do método**

1. É possível capturar informações sobre o esforço que determinados atacantes fazem para violar um sistema.

2. Talvez seja possível definir padrões de ataques.

3. Pode ser feito um estudo sobre o esforço despendido com alguns tipos de vulnerabilidades conhecidas.

**Desvantagens do método**

1. A técnica é aplicada depois que o software é desenvolvido.

2. Atacantes têm habilidades e experiência diferentes. Não está claro ser possível determinar uma medida de esforço que possa ser associada ao sistema.

3. Muitas vezes um ataque à segurança não é reconhecido. O processo de medição pode falhar para vulnerabilidades desconhecidas.



## 4.2 O Índice de Vulnerabilidade do Sistema

Alves-Foss e Barbosa (1995) apresentam uma abordagem para identificar e quantificar vulnerabilidades de sistemas computacionais ao introduzir um método de avaliação denominado Índice de Vulnerabilidade do Sistema, IVS. O IVS é formulado para avaliar fatores que podem afetar enormemente o estado de segurança do sistema. Ações de super usuários[2], bem como indícios de um potencial estado de insegurança, servem como base para a escolha destes fatores relevantes para a segurança. Estes fatores são divididos em três categorias:

[2] Super usuário - A conta super usuário, normalmente chamada *root*, é usada para configurar e facilitar a administração do sistema, e não deve ser usada para tarefas cotidianas como envio e recebimento de e-mail, exploração geral do sistema, ou para programação.

- Características do sistema - configurações do software e hardware que influenciam a segurança do sistema, mas não são facilmente modificadas devido ao custo, as complexidades ou aos fatores externos. Como exemplos destes fatores podemos ter:
    - Vulnerabilidades de segurança física
    - Senhas antigas
    - Atualização do sistema operacional em relação a *bugs* de segurança
- Atos potencialmente negligentes - toda ação que degrada a segurança quando omitida, ou impropriamente empregada. Alguns exemplos destes fatores são:
    - Número de indivíduos com privilégios de super usuário
    - Usuários com senhas fracas
    - Contas inativas
- Atos potencialmente malevolentes - caracterizam ações que podem enfraquecer ou violar a segurança, inclusive toda aquela que tenta enganar ou vencer os mecanismos de segurança. Como por exemplo:
    - Objetos com o mesmo nome de comandos do sistema ou programas.
    - Super usuários estranhos.

O Índice de Vulnerabilidade do Sistema é uma medida entre 0 e 1 que indica a vulnerabilidade de um sistema em relação a condições adversas que podem levar a uma exploração de um sistema em um determinado momento. Para entender o significado do IVS, devemos começar definindo que ele (o índice) não significa. Enquanto o IVS é uma medida entre 0 e 1 (quanto mais próximo de 1, maior é o nível de vulnerabilidade), não deve ser interpretado como um valor de probabilidade estatística. Portanto, um IVS de 0.84 não deve ser entendido como a probabilidade de um sistema ser invadido por pessoal não autorizado. Este valor serve, simplesmente, para alertar que condições para a exploração existem, pois se

invasores explorarem técnicas comuns, tornar-se-ão conscientes do estado de vulnerabilidade do sistema.

Segundo os autores, os fatores que podem afetar o estado de segurança do sistema são avaliados e combinados, através do uso de regras especiais, para fornecer uma medida da vulnerabilidade do sistema. Para cada regra IVS, definida a partir dos fatores que afetam a segurança, é atribuído um valor que responde a seguinte questão: “Até que ponto a presença deste fator, torna o sistema vulnerável para a exploração?”. Cada regra, então, pode ser definida da seguinte forma: Se antecedente Então Conseqüente, por exemplo, Se “o sistema operacional não for atualizado quanto a *bugs* de segurança já informados” Então o nível de vulnerabilidade do sistema é 0,6. Para chegar ao valor final do indicador de vulnerabilidade do sistema, devem ser combinados os índices de segurança de todas as regras IVS aplicáveis ao fator considerado.

Demonstrar a vulnerabilidade de um software com valores numéricos entre 0 e 1 é inútil sem uma interpretação apropriada. Portanto, os autores introduzem uma classificação de valores de IVS em quatro grupos, cada um deles representando diferentes espectros de vulnerabilidades:

- Valores entre 0 e abaixo de 0,15 indicam uma baixa vulnerabilidade.
- Valores entre 0,15 e menores do que 0,3 indicam um nível moderado de vulnerabilidades.
- Um valor entre 0,3 e menor do que 0,6 é considerado suficiente para ser explorado por classes de ataques mais comuns.
- Um Valor igual ou acima de 0,6 é considerado extremamente vulnerável.

**Vantagens do método**

1. A vantagem deste método se situa na abstração do problema que o torna útil para avaliar vulnerabilidades em sistemas operacionais e implementações de software e hardware distintas.

2. Um sistema especialista pode ser implementado a partir das regras

**Desvantagens do método**

1. A técnica é aplicada depois que o software foi desenvolvido.
2. Avaliadores diferentes podem chegar a resultados diferentes.
3. O método é muito dependente da experiência do avaliador.
4. É difícil avaliar o quanto um elemento influência a segurança de outro elemento.

## 4.3 MTTI – Minimum-Time-To-Intrusion

Voas e co-autores (Voas et al., 1996) propõem uma métrica de tolerância a falhas denominada *minimum-time-to-intrusion*, MTTI, para avaliar quantitativamente segurança e a capacidade de resistência de sistemas de informação. Esta métrica é baseada no período de tempo esperado para uma invasão simulada poder acontecer.

Uma métrica quantitativa é computada para determinar se as ameaças simuladas minam a segurança do sistema definida pelos usuários, conforme a especificação do programa. A abordagem do método, chamada de Análise de Vulnerabilidade Adaptativa (AVA), exercita o software, simulando a entrada de ataques maliciosos e não-maliciosos que se incluem em várias classes de ameaças conhecidas. Esta abordagem permite conhecer antecipadamente se sistemas estão seguros em relação a um conjunto pré-definido de ameaças $T = \{t1, t2,..., tn\}$, constituído por ameaças recorrentes mais comuns de serem encontradas. O conjunto de ameaças $T$ é aberto para permitir a inclusão de novas ameaças, podendo variar conforme a necessidade do avaliador. A possibilidade de existência de vários conjuntos de ameaças diferentes torna este método adaptativo, conforme o entendimento dos autores.

Este método não fornece uma métrica absoluta, tal como MTTF (Mean

Time to Failure), mas ao invés disso o método fornece uma métrica relativa, que permite ao usuário comparar a segurança de diferentes versões do mesmo sistema ou comparar com funcionalidades similares de outros sistemas.

A Análise de Vulnerabilidade Adaptativa (AVA) é um algoritmo de análise de software adaptado a partir da extensão de uma técnica de análise de propagação denominada em inglês *Extended Propagation Analysis* (EPA), utilizada na avaliação de segurança (em inglês:safety) de softwares críticos. A simulação de ameaças no AVA é efetuada através de uma combinação de injeção de faltas e geração de casos de testes. A detecção de invasão é efetuada usando uma linguagem de especificação de invasões baseada em predicados (PRED).

Baseado na modelagem de vulnerabilidades, entradas do programa e invasões, o algoritmo AVA determina a proporção de invasões que ocorrem como um resultado de injeção um programa com ameaças simuladas. O número resultante deste algoritmo pode ser usado para calcular à métrica de segurança.

Seja P o programa, x representa um valor de entrada deste programa, Q representa a distribuição provável da curva normal, Q' denota o inverso da distribuição provável da curva normal usada e *l* corresponde a uma localização de programa em P.

**Algoritmo AVA:**

1. Para cada localização apropriada ***l*** em P,

   execute passos 2-7.

2. Inicialize a variável **conta** com zero.

3. Randomicamente selecione uma entrada **x** ou seqüência de ***Q*** ou ***!Q***, e

   Se P parar sobre **x** após um período de tempo fixado ***t***, encontre o estado dos dados correspondentes criados por **x** imediatamente depois da execução de ***l***.

   Chame este estado dos dados de Z.

4. Altere o valor experimentado da variável ***a*** encontrada em Z criando $Z^i$, e Execute o código seguinte em $Z^i$. A maneira pela qual ***a*** é alterado será representativo da classe de ameaça de T que é desejada.

5. Se a saída de ***P*** satisfaz **PRED**, incremente ***conta***.

6. Repita os passos 3-5 **n** vezes, onde n é o número de entradas de casos de testes.

7. Divida ***conta*** por **n** produzindo $\psi alPQ$, uma avaliação de vulnerabilidade para cada linha ***l***. Isto significa que (1 – $\psi alPQ$) é a avaliação de segurança que foi observada, dados P, Q e T.

Os primeiros dois passos do algoritmo são básicos. AVA é uma metodologia baseada em código fonte, significando que instrumentação é colocada entre linhas de comando específicas (que chamamos "localização" no código). Ou um sistema automatizado que implementa o algoritmo (se ele é inteligente suficiente), ou o usuário deve informar ao sistema que localizações são relevantes para a injeção de faltas. Deste modo, o primeiro passo é localizar aonde a injeção vai ocorrer. O passo seguinte, um contador é inicializado com zero, porque desejamos observar quantas intrusões de segurança ocorrem devido às ameaças simuladas que o protótipo empreendeu para uma localização particular "l".

Diferente da maioria das métricas de software atualmente em uso, nossa medição de avaliação de software AVA não considera a estrutura do software, mas particularmente o seu comportamento. Então o algoritmo seleciona casos de testes sob quais os programa rodará, visto que necessitaremos executar o software com o objetivo de quantificá-lo estatisticamente o seu comportamento. Esta amostra de entradas é executada no passo 3. As entradas podem surgir de diferentes esquemas de testes que são mais prováveis para disparar uma intrusão bem sucedida: eventos raros (com respeito para o perfil operacional), seqüências de entradas conhecidas que não são usuais ou prováveis de serem tratadas, entradas totalmente randômicas, ou mesmo que o perfil operacional do sistema. O quarto passo executa o estado de "corrupção" atual do programa ou mutação

sintática do código (o passo da injeção de faltas). Uma vez que a falta é injetada no passo 4 é executado durante a fase de análise, o programa tem sido "derrubado" em alguma maneira. O passo 5 determina se o problema forçado no passo 4 ocasiona o programa a produzir um evento de saída que satisfaz nossa definição do que constitui uma violação de segurança. Nesse caso o contador é incrementado de 1. Para prover uma estimativa estatística de vulnerabilidades, passos 3, 4 e 5 são repetidos (passo 6). Esta estimativa é calculada no passo 7.

A métrica *minimum time to intrusion* (MinTTI) é baseada na informação produzida no algoritmo acima. Duas considerações devem ser feitas:

1. A métrica pode ser ajustada para qualquer unidade de tempo desejada.

2. O maior valor avaliado, a maior segurança, será prevista para o sistema.

A Figura 10 fornece uma visão geral do protótipo AVA. Os três principais componentes do sistema (injeção de falhas, geração de caso de teste, especificação da invasão) fornecem um esquema para determinar o valor das métricas de segurança.

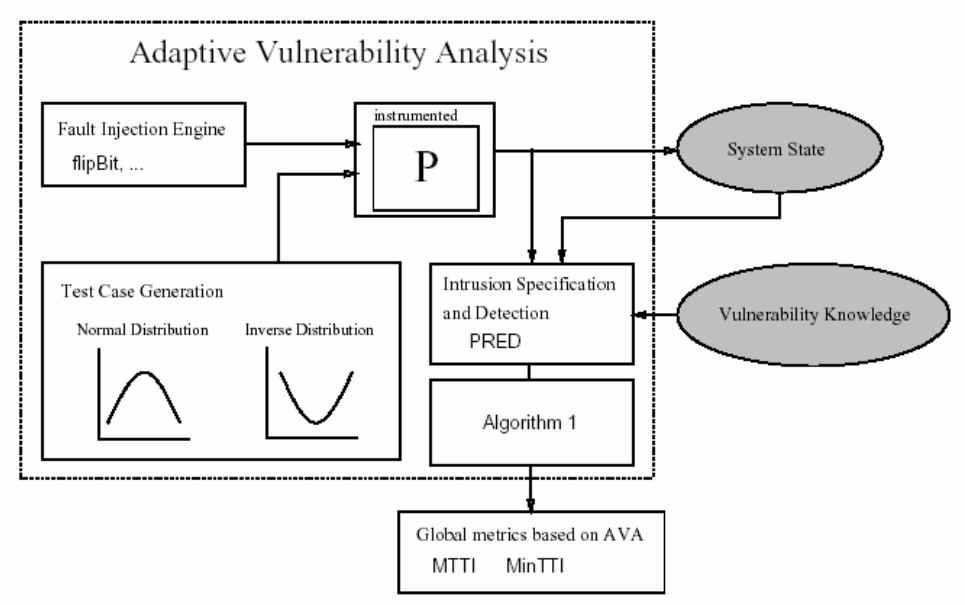


Figura 10: Visão Geral do protótipo AVA (extraída de .Voas et al., 1996)

A métrica MTTI, definida como o intervalo de tempo médio antes que uma invasão ocorra, é baseada nos casos de entrada em Q (e seu inverso), nas classes de injeção de falhas que são usadas e nas classes de invasões definidas nos predicados (PRED). O seu calculo é demonstrado na figura 11 a seguir:

$$\mathbf{MTTI} = \left[\frac{\sum_{l=1}^{M}[\hat{\psi}_{alPQ} \cdot \hat{\varepsilon}_{lPQ}] \cdot \left(\frac{\text{program executions}}{\text{unit of time}}\right)}{M}\right]^{-1}$$

Figura 11: Métrica MTTI. M é o número de localizações aonde o AVA foi aplicado (extraída de .Voas et al., 1996)

O tempo mínimo para a invasão (MinTTI) é o período de tempo mais curto previsto antes que qualquer invasão definida em PRED ocorra. O seu cálculo é demostrado na figura 12 a seguir:

$$\mathbf{MinTTI} = \left[\max_{l}[\hat{\psi}_{alPQ} \cdot \hat{\varepsilon}_{lPQ}] \cdot \left(\frac{\text{number of program executions}}{\text{unit of time}}\right)\right]^{-1}$$

Figura 12: Métrica MinTTI (extraída de .Voas et al., 1996)

**Vantagens do método**

1. A técnica pode ser modificada para simular novas classes de ameaças sem atrapalhar o processamento das classes antigas.

2. A técnica pode ser customizada pelo usuário para classes de invasões específicas sem a necessidade de qualquer mudança na forma de avaliação.

3. Determina de forma dinâmica o comportamento do sistema em relação à segurança.

4. Produz informações importantes para a melhoria do processo de desenvolvimento.

5. Os mesmos parâmetros de análise produzem os mesmos resultados após um certo tempo.

6. Pode ser usada em conjunto com testes de penetração e outras técnicas utilizadas para a melhoria da qualidade do software.

**Desvantagens do método:**

1. Está baseada em modelos de injeção de falhas e dessa forma qualquer resultado será dado em relação às classes de ameaças informadas. Novas classes de ameaças com um MinTTI menor do que qualquer ameaça anterior resultará em um nível de segurança superestimado.

2. A técnica é aplicada depois que o software foi desenvolvido.

3. A técnica pode ser usada contra o próprio sistema caso um dos desenvolvedores tenham acesso às informações internas durante a execução do algoritmo AVA.



## 4.4 Avaliação Quantitativa para Monitorar a Segurança Operacional

Ortalo et al. (1999) modelam o sistema como um grafo de privilégios[3], que exibe suas vulnerabilidades e estima o esforço despendido por um atacante para conseguir um ataque bem-sucedido, a partir da exploração de tais vulnerabilidades. O cálculo da métrica quantitativa de segurança é realizado a considerando a dificuldade que um possível atacante teria em explorar estas vulnerabilidades e vencer os objetivos de segurança do sistema.

[3] Um privilégio pode ser entendido como uma concessão cedida a um ou mais usuários, possibilitando a utilização de funcionalidades no software, segundo o seu nível de permissão.

### A Modelagem de Segurança do Sistema

Em um grafo de privilégios, um nó X representa um conjunto de privilégios concedidos a um usuário ou a um grupo de usuários. Uma aresta em um grafo de privilégios representa uma vulnerabilidade. Por exemplo, sejam X e Y dois nós em um grafo de privilégios, uma aresta de um nó X para um nó Y representa uma vulnerabilidade que permite a um usuário que possui os privilégios de X obter os privilégios do nó Y (Figura 13).

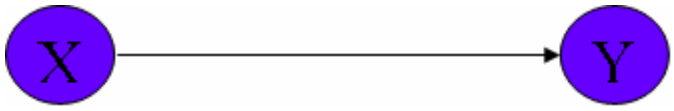


Figura 13: X obtém os privilégios do nó Y

Três classes de arestas (vulnerabilidades) podem ser encontradas em um grafo de privilégios:

1. Uma vulnerabilidade representada por uma aresta pode ser uma falha de segurança, tal como uma senha facilmente descoberta ou uma proteção ruim para diretórios e arquivos, que permita a implantação de um Cavalo de Tróia.

2. Uma vulnerabilidade não é necessariamente uma falha de segurança. A vulnerabilidade pode ocorrer a partir do uso durante um ataque de qualquer recurso ou funcionalidade designada para melhorar a segurança. A segunda classe de arestas representa os recursos ou funcionalidades utilizadas para melhorar a segurança do sistema que podem ser utilizados contra a segurança do próprio sistema.

3. A terceira classe de arestas é a representada por subconjuntos de privilégios decorrentes do esquema de proteção.

Um caminho no grafo representa um conjunto de vulnerabilidades que podem ser utilizadas por um atacante para alcançar um alvo. O peso de cada aresta representa o esforço para explorar uma vulnerabilidade representada pela aresta.

A Figura 14 a seguir exemplifica um grafo de privilégios com arestas rotuladas por classes de vulnerabilidades.

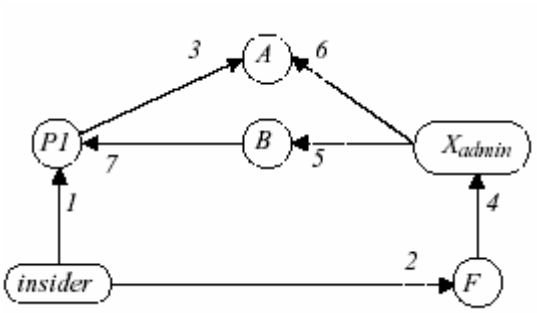


Figura 14: Grafo de Privilégios (extraída de Ortalo et al., 1999)

Cada aresta no grafo é representada com os seguintes pesos:

1. X pode descobrir a senha de Y.
2. X pode instalar um cavalo de Tróia que Y pode ativar em algum momento.
3. X pode explorar uma falha no programa de correio de Y.
4. Y é um subconjunto de X.
5. Y usa um programa que X pode modificar.
6. X pode modificar um programa de propriedade de Y
7. X está trabalhando com os privilégios de Y

Alguns conjuntos de privilégios são altamente sensíveis, como, por exemplo, os privilégios de super usuários. Estes nós são chamados de “alvos” visto que estes são provavelmente alvos de ataques. Os nós com os privilégios de possíveis atacantes são chamados de nós atacantes. Se um caminho existe entre um nó atacante e um nó alvo, então, por transitividade, uma brecha de segurança pode ocorrer, permitindo que um possível atacante explore as vulnerabilidades do sistema para obter privilégios alvo.

Mesmo que exista um caminho entre um nó atacante e um nó alvo, dificilmente a segurança do sistema será derrotada se um atacante precisar de

muita inteligência, competência ou tempo para percorrer todas as arestas que compõem o caminho. Esta dificuldade dos atacantes em alcançar os alvos pode ser uma boa métrica de segurança do sistema. Para poder medir este esforço por parte do atacante, basta atribuir para cada aresta no grafo um peso que quantifique o esforço requerido para explorar uma determinada vulnerabilidade. Esta definição de esforço é composta por várias características do processo de ataque tal como uma ferramenta de ataque pré-existente, o tempo necessário para realizar um ataque, o poder computacional disponível para o atacante, etc. Por exemplo, o esforço necessário para obter uma senha pode ser medido pelo poder computacional e pelo tempo necessário para quebrar uma senha. Para um ataque de um cavalo de Tróia, o esforço pode ser medido como a competência necessária para desenvolver um cavalo de Tróia, o tempo necessário para implantá-lo em um programa que pode ser usado para atacar um usuário, e o tempo necessário para um usuário ativá-lo. O peso do esforço atribuído para um aresta é deste modo um parâmetro composto, que pode ser representado como uma taxa de sucesso para o correspondente ataque elementar.

**Estudando o comportamento do Atacante**

Para avaliar métricas quantitativas que caracterizem a segurança operacional baseada no grafo de privilégios é importante identificar os cenários de ataques que podem ser empreendidos por um atacante em potencial para alcançar o alvo.

Normalmente, em um ataque, um atacante começa com algum privilégio mínimo e parte em busca da obtenção de algum privilégio maior no sistema. Sendo assim, em um grafo de privilégios, o caminho do nó de um atacante até o nó alvo descreve o processo de ataque. Em um grafo podem existir mais de um caminho do nó do atacante para o nó alvo. A figura 15 a seguir representa o caminho de um atacante até o alvo:

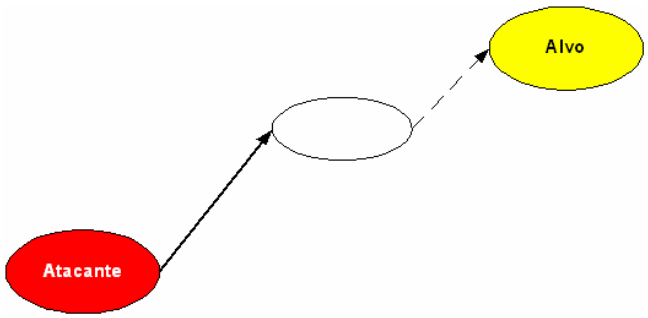


Figura 15: O Atacante e o caminho até o alvo.

O progresso do atacante pode ser caracterizado por um grafo de transição de estados onde cada estado identifica o conjunto de privilégios que ele já ganhou e a transição entre os estados ocorre quando o atacante tem sucesso em um ataque que o permite adquirir novos privilégios. Este grafo, denominado *Attack state graphs*, consiste numa representação das diferentes maneiras que um invasor qualquer pode utilizar para alcançar certo objetivo, tal como o acesso a informações sobre senhas de usuários de um sistema. Os nós do grafo representam os privilégios que podem ser alcançados pelo atacante.

Para estudar o comportamento dos atacantes, diferentes modelos podem ser definidos dependendo das suposições consideradas sobre o seu comportamento. No primeiro modelo o atacante escolhe o caminho mais curto que o conduz até o alvo (denotado como SP), isto é, o caminho que tem o mais baixo valor de esforço totalizado. Para ser possível o uso deste caminho por parte do atacante, ou o atacante é um super usuário, ou o sistema atacado não possui nenhuma proteção. Para melhor aplicar a abordagem de árvore de privilégios, os autores supõem que o atacante não sabe qual é o caminho mais curto.

Duas outras suposições para específicos modelos de processos de ataque (comportamento do atacante) são:

- Memória total (TM): todas as possibilidades de ataque serão consideradas em qualquer estágio de ataque

- Sem memória (ML): em cada novo nó visitado, somente ataques possíveis a partir daquele nó serão considerados.

A figura 16 a seguir representa a derivação de *Attack state graphs a partir de um grafo de privilégios, considerando os modelos de processos de ataque memória total, TM, e sem memória, ML.*

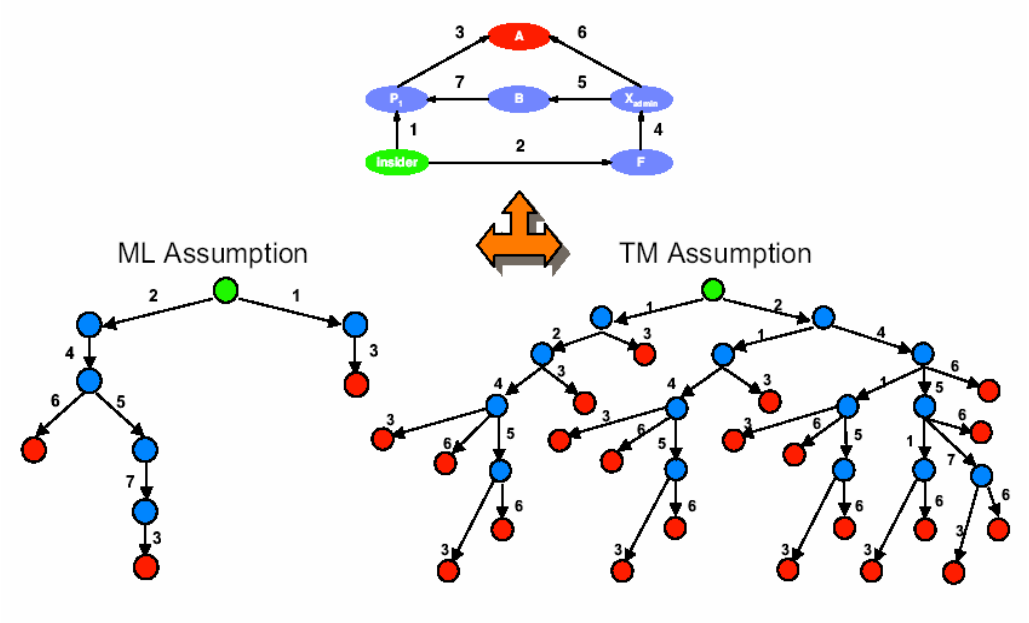


Figura 16: Derivação das *Attack state graphs* a partir do Grafo de Privilégios. Figura baseada no artigo de Ortalo e co-autores (Ortalo et al., 1999)..

**Modelo Matemático**

Para sermos capazes de comparar a evolução da medida de segurança correspondente às suposições TM, ML e SP, necessitamos especificar um modelo matemático para avaliar o esforço médio despendido por um atacante para que alcance um determinado alvo. Os autores Ortalo et al. (1999) propõem a utilização do modelo de Markov, pois, segundo eles, este modelo satisfaz algumas propriedades intuitivas relacionadas à evolução da segurança (Dacier et al., 1996). O modelo de Markov está baseado na suposição de que a probabilidade de conseguir um ataque elementar bem sucedido à segurança, antes que uma quantidade de esforço "*e*" seja despendida, pode ser descrita por uma distribuição exponencial dada por $P(e) = 1 - exp(-\lambda e)$, onde $\lambda$ é uma taxa atribuída ao ataque. Considerações práticas derivadas do uso desta distribuição são as seguintes:

- Um atacante em potencial será bem sucedido, eventualmente, em

pesquisar o alvo, se um caminho o levar ao alvo existente, considerando que ele gasta um esforço razoável para isto.

- O esforço médio para quês atacante tenha sucesso em um determinado ataque é dado por 1/ $\lambda$

O último ponto é particularmente valioso por que o conhecimento da taxa do ataque é suficiente para caracterizar a distribuição completa. O primeiro ponto ainda merece alguns esclarecimentos. De fato, segundo os autores, o objetivo deste trabalho foi o de avaliar a resistência do sistema em relação a ataques bem sucedidos a um alvo específico, considerando cenários de ataques que, eventualmente, levam ao alvo e não os cenários que podem ser abortados durante o processo de ataque.

Baseado na suposição de Markov, cada transição no processo de transição de estados é taxado com a taxa de sucesso da vulnerabilidade correspondente. Várias medidas probabilísticas podem ser derivadas a partir do modelo, entre estas, o esforço médio para um atacante potencial alcançar o alvo especificado, denotado como METF (*Mean Effort To Security Failure*, em analogia com o *Mean Time to Failure*). Esta métrica permite fácil interpretação dos resultados: quanto mais alto o METF, melhor será a segurança. Além disso, simples expressões analíticas podem ser facilmente obtidas e analisadas para checar a plausibilidade dos resultados do modelo.

O METF é dado pela soma dos esforços médios gastos nos estados que levam até ao alvo que são sobrecarregados pela probabilidade de visitar estes estados. O esforço médio gasto no estado *j*, denotado como *Ej*, é dado pelo inverso da soma das taxas de transição de estados *j*'s :

$$E_j = 1 / \sum_{i \in out(j)} \lambda_{ji}$$

*Out(j)* é o conjunto de estados alcançáveis em uma transição simples do estado *j* e $\lambda_{ji}$ é a taxa de transição do estado *j* para o estado *i*.

Denotamos por $METF_k$ o esforço médio quando o estado *k* é o estado inicial

e $P_{ki}$ a probabilidade condicional de transição do estado *k* para o estado *i*, então:

$$METF_k = E_k + \sum_{i \in out(k)} P_{ki} * METF_i$$

$$P_{ki} = \lambda_{ki} * E_k$$

De acordo com este modelo, o valor mais alto da probabilidade condicional de saída corresponde às transições com as taxas mais altas de esforço bem sucedido.

**Vantagens do método**

1. Similar ao método de Brocklehurst et al.(1994), "Esforço despendido por um atacante".

**Desvantagens do método**

1. A técnica é aplicada depois que o software é desenvolvido.
2. Atacantes têm habilidades e experiência diferentes.
3. Muitas vezes um ataque à segurança não é reconhecido. O processo de medição pode falhar para vulnerabilidades desconhecidas.



## 4.5 Método Wang e Wulf para Medir Segurança

Chenxi Wang e William A. Wulf (1997) apresentaram um *framework* para medir de forma sistemática o nível de segurança de um software. O nível de segurança é apresentado como uma n tupla de números reais, cada tupla representando uma propriedade de segurança. Por exemplo, se a segurança do sistema for definida como uma combinação de confidencialidade, integridade e disponibilidade, uma possível métrica de segurança é a tupla *<f1, f2, f3>*, *f1* representando confidencialidade, *f2* representando integridade e *f3* representando disponibilidade. Os valores nesta tupla indicam a força da segurança, medida a partir de sua confidencialidade, integridade e disponibilidade do sistema.

No caso em que uma medida simples é desejada, um modelo para derivar a medida final a partir de medidas de diferentes aspectos. Por exemplo, a tupla *<f1 (confidencialidade), f2 (integridade), f3(disponibilidade)>*, *g(f1, f2, f3)>* onde *g(f1, f2, f3) = 0.65 f1 + 0.25 f2 + 0.1 f3* define uma medida em que o valor final depende 65% da confidencialidade, 25% da integridade e 10% da disponibilidade do sistema.

Esta metodologia utiliza a técnica de decomposição para desenvolver modelos que relacionam atributos de segurança de nível mais alto com os de nível mais baixo, que, segundo os autores, são componentes mais mensuráveis. O processo de avaliação sugerido consiste de cinco passos:

- Decomposição
- Determinar os relacionamentos funcionais, isto é, as interações dos componentes do sistema.
- Determinar pesos e prioridades
- Definir medições básicas, e
- Analisar a sensibilidade do componente

Estes cinco passos são analisados a seguir.

**Decomposição**

Decompor um sistema em partes menores pode ser um processo difícil. Ainda mais, se ele resultar em elementos independentes que podem ser avaliados sem qualquer necessidade de promover o particionamento. Estes componentes básicos têm grande influência na segurança do software como um todo.

Os autores declaram que uma decomposição funcional de um software será possível se a pergunta: *"O que precisa acontecer para que o objetivo atual não falhe?"* for respondida. Para exemplificar a técnica de decomposição (figura 17), os autores fazem uma analogia com a segurança de uma casa., considerando que para a garantia da integridade e privacidade da segurança da casa, às portas e

janelas são os fatores principais que necessitam ser decompostos em seus componentes funcionais.

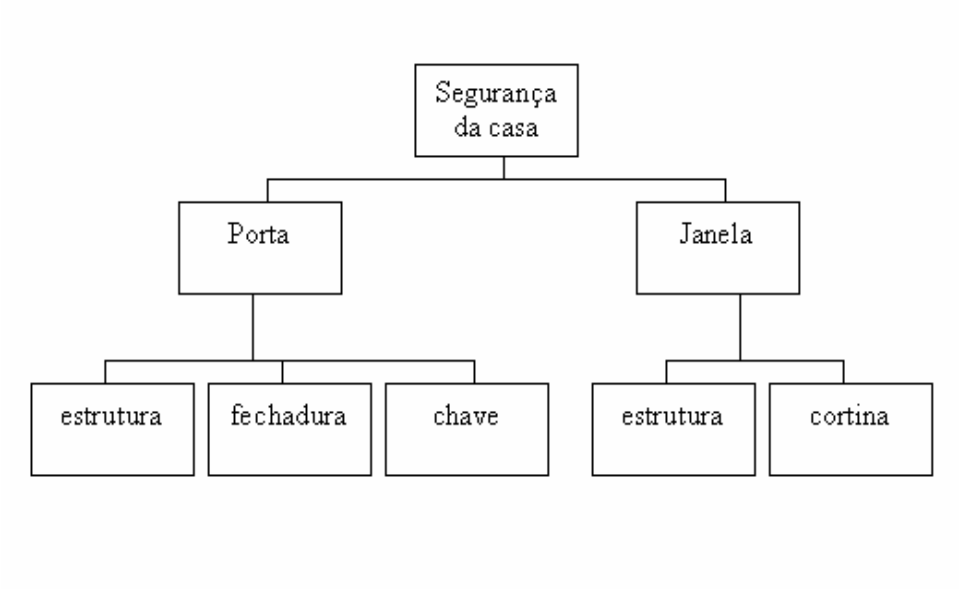


Figura 17: Exemplo de decomposição. Figura extraída do artigo “A Framework for Security Measurement” de Wang e Wulf (1997)

Cada componente de mais baixo nível na representação é avaliado ao atribuirmos um valor de 0 a 1 em relação a uma propriedade de segurança.

O processo de decomposição pode ser executado nos seguintes passos:

1. Identifique um conjunto de objetivos de segurança para o sistema sujeito à análise.

2. Identifique os componentes que contribuem para o sucesso dos objetivos. Estes são as funções que têm acontecer para que os objetivos não falhem.

3. Examine os nós subordinados, verificando a necessidade de se decompor ainda um pouco mais. Caso seja necessário, repita o processo com os nós subordinados.

4. Termine o processo de decomposição quando nenhuma folha necessitar mais ser decomposta.

**Determinando os Relacionamentos Funcionais**

Uma vez empregado o processo de decomposição, a interação entre os muitos componentes que contribuem para a segurança também deve ser considerada. A análise das relações entre estes componentes não é uma tarefa simples e nos leva a uma nova complexidade de significativos problemas representados pela análise de suas relações.

A seguinte lista descreve um conjunto de relacionamentos lógicos entre os componentes do sistema e as regras de composição associadas com eles.

**Link mais fraco (*WL*)** – *WL* significa que o funcionamento do pai é, em última análise, restringido pela fraqueza dos seus filhos. A "fraqueza", neste contexto, se refere a uma medida da força da segurança. Em particular, uma falha em um nó filho em uma relação *WL* causará uma falha da função objetivo. Matematicamente, *WL* pode ser descrita como:

$$S(pai) = min(S(filho_1), S(filho_2),..., S(filho_n)),$$

Onde *S* representa os valores da avaliação de nós e n é o número de nós filhos.

Por exemplo, considerando o exemplo da casa, a segurança da casa depende de dois fatores: a porta e a janela. É facilmente percebido que o comprometimento de um dos dois fatores pode resultar na invasão da casa e as conseqüências são igualmente prejudiciais. Portanto, uma relação *WL* existe entre a porta e a janela.

Supondo que os escores de avaliação para a porta e a janela são respectivamente 0.75 e 0.83, o escore da casa é calculado da seguinte forma:

$$S(casa) = min\ (0.75, 0.83) = 0.75$$

**Link mais Fraco Sobrecarregado (*WWL*)**- O *WWL* é uma generalização do *WL*. Enquanto o *WL* não diferencia entre trivial e fatores importantes, o *WWL* leva em conta que diferentes nós filhos podem ter vários graus de impacto no nó

pai.

Por exemplo, considere o funcionamento de uma porta no exemplo da casa, foi decidido que três fatores são importantes: a estrutura da porta, a fechadura e a chave. Portanto, a segurança fornecida pela porta pode ser mais fortemente influenciada por um subconjunto de outros fatores. Por exemplo, dependendo de que espécie de vizinho da casa a casa possua, pode ser que vale à pena dar mais atenção à resistência da porta e a segurança da fechadura do que dar atenção à chave. Portanto, a relação *WWL*, que parece ser mais apropriada para esta situação, está representada na figura 18 a seguir:

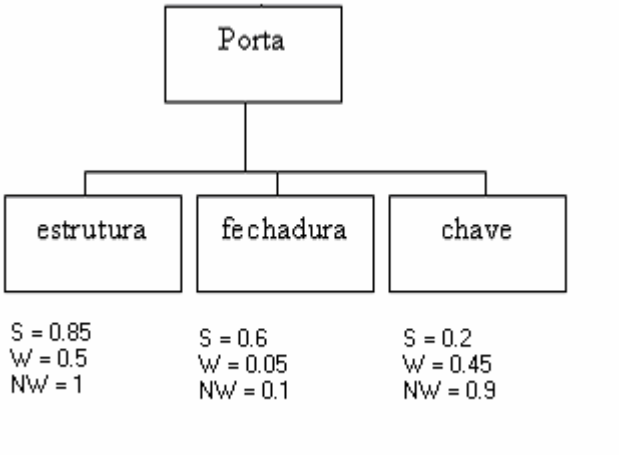


Figura 18: Exemplo da Porta (extraída do artigo “A Framework for Security Measurement” de Wang e Wulf (1997))

Matematicamente, a saída de uma relação *WWL* é calculada seguindo os seguintes passos:

1. Suposições: cada nó filho tem um escore *S* de avaliação e um peso *W*, onde *S* é um número entre 0 e 1 e a soma dos pesos entre todos os irmãos é igual a 1. Na figura 18, os pesos dos três nós folhas são 0.5, 0.05 e 0.45 e seus escores de avaliação são 0.85, 0.6 e 0.90.

2. Normalize os pesos a partir do peso de valor mais alto. Os pesos normalizados (*NWs*) para os três nós folhas na figura 18, são 1, 0.1 e 0.9.

3. Selecione o filho que possui a menor relação *S/NW*, *min* ($S_1/NW_1$, $S_2/NW_2$,...,

$S_n/NW_n)$, sendo *n* é o número de nós filhos. Na figura 18, o filho com menor valor é a chave, onde *S/NW = 0.222*.

4. Compute a soma dos pesos dos nós filhos $SomaPesos = \sum_{i=1}^{n} (S_i * W_i)$, onde *n* é o número de nós filhos. No exemplo da figura 18, a soma dos pesos é: *(0.85 * 0.5 + 0.6 * 0.05 + 0.2 * 0.45) = 0.545*

5. O escore *WWL* do nó pai é calculado como a raiz quadrada do produto da soma dos pesos com o escore do filho mais fraco, isto é, de menor relação *S/NW* é igual a:

   $\sqrt{0.545 * 0.2} = 0.33$

**Irmãos Priorizados (*PS*):** Esta relação existe entre irmãos, cada um contribuindo para um aspecto independente da função pai. Por exemplo, o elemento janela, no exemplo anterior da casa, é decomposto em estrutura e cortina. É facilmente percebido que a cortina e a estrutura fornecem funcionalidades independentes que, coletivamente, contribuem para a segurança do elemento janela. Todavia, uma falha funcional de um elemento não necessariamente causaria uma falha funcional na janela. Uma relação *PS* também reconhece a relação de importância dos nós filhos para o pai. Formalmente, uma relação *PS* pode ser descrita como:

$S(pai) = \sum_{i=1}^{n} (S_i * W_i)$, onde *S* é o escore de avaliação e *W* representa o peso em percentual, e *n* representa o número total de nós filhos.

Supondo que o valor de *S (estrutura)* é 0.98 e o valor de *S(cortina)* é 0.63, e seus respectivos pesos são 0.85 e 0.15, o escore da janela pode ser calculado como: *S(janela)= 0.98 * 0.85 + 0.63 * 0.15 = 0.928*. A figura 19 a seguir ilustra o cálculo de S, segundo a relação irmãos priorizados:

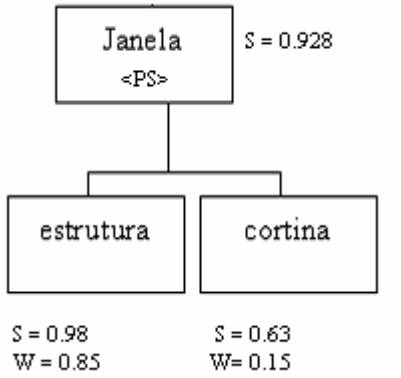


Figura 19: Funcionamento da janela em relação aos seus fatores (extraída do artigo “A Framework for Security Measurement” de Wang e Wulf (1997))

**Determinando Pesos e Prioridades**

Quando decompomos, os pesos indicam os graus com que os nós filhos influenciam seus pais. Esta avaliação de pesos é crítica porque os pesos são utilizados para computar a influência de vários elementos no sistema como um todo.

Em um modelo baseado na técnica de decomposição de elementos, o nó filho pode influenciar seus pais de maneiras diferentes. Portanto, de acordo com os autores do método, considerarmos esta influência é de extrema importância antes de introduzirmos os pesos que indicam os valores de tais efeitos.

Os pesos dos nós filhos podem ser definidos a partir de um questionário ou experiência do próprio avaliador, o que torna o método aqui apresentado bastante subjetivo.

**Análise da Sensitividade do Componente**

O último elemento da metodologia consiste em uma análise de sensitividade de um componente. Segundo os autores, uma análise de sensitividade é realizada para avaliar o impacto de variações nos componentes individuais. Ela permite identificar a contribuição de um componente ou conjunto de componentes para a segurança do sistema como um todo.

Se o valor da medida para a propriedade de segurança variar drasticamente com a mudança de um certo conjunto de componentes, uma segunda visão mais próxima deve ser executada sobre este conjunto particular de componentes. Segundo os autores, recomendações para melhorar a força da segurança dos componentes podem ser feitas baseadas neste estudo.

Vamos retornar ao exemplo da casa (Figura 17) para brevemente descrever o processo de uma análise de sensitividade de um componente. Usando as regras de decomposição definidas por cada relacionamento funcional, é possível medir os atributos de segurança de nível mais alto a partir dos componentes básicos.

A figura 20 a seguir ilustra o exemplo anterior do sistema da casa com os respectivos pesos dos nós folhas e o tipo de relacionamento funcional empregado. A lista seguinte de equações é derivada a partir do relacionamento funcional usado na árvore de decomposição.

1. $S(casa) = min\ (S(porta, S(janela)))$.

2. $S(janela) = 0.98 * S(estrutura) + 0.63 * S(cortina)$

3. $S(porta) = \sqrt{(0.5 * S(estrutura) + 0.15 * S(chave) + 0.05 * S(fechadura))} * \sqrt{Min\ peso(\ S(estrutura), S(chave), S(fechadura))}$

4. $S(casa)=S(porta)= \sqrt{S(fechadura)*(0.5*S(estrutura)+0.45*S(chave)+0.05*S(fechadura)}$

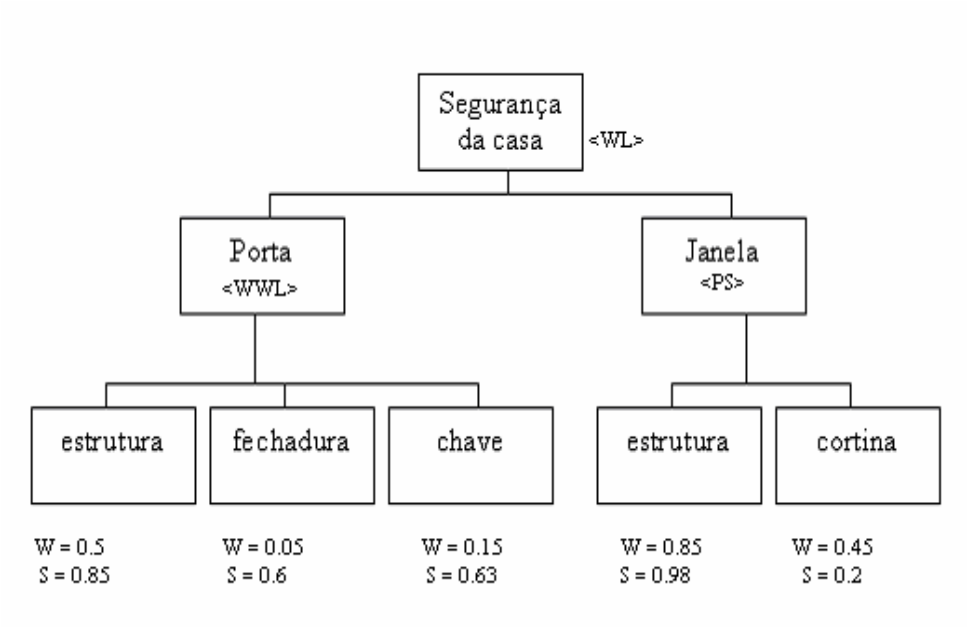


Figura 20: Árvore de Decomposição para uma Casa (Extraída do artigo "A Framework for Security Measurement" de Wang e Wulf (1997))

Podemos agora calcular os indexes de sensitividade do componente. Assumindo que os escores dos outros componentes são mantidos constantes, o index de sensitividade (*SI*) de um determinado componente é definido como a razão entre o escore geral e o escore do componente. Por exemplo, o SI da estrutura da porta é calculado como segue:

$$S.I._{estrutura\_porta} = \delta S(casa)/\ \delta S(estrutura)$$

$$= \frac{0.5 * S(chave)}{2\sqrt{S(chave) * (0.5*S(estrutura) + 0.45*S(chave) + 0.05 * S(fechadura))}}$$

$$= \frac{0.5 * 0.2}{2\sqrt{0.2\ (0.5*S(estrutura) + 0.45*0.2 + 0.05 * 0.6)}}$$

$$= \frac{0.5}{\sqrt{10\ S(estrutura) + 2.4}}$$

Substituindo o valor de *S (estrutura),* nós temos que $S.I._{estrutura\_porta}$ = 0.1514. Para cada unidade de mudança na performance da estrura da porta, a força da segurança da casa como um todo mudará 0.1514 unidades.

A tabela 2 abaixo mostra os indexes de sensitividade para todos os

componentes folhas do exemplo da casa:

| Componente | Equação S.I. | Valor do S.I. |
|---|---|---|
| Estrutura da porta | $\frac{0.5}{\sqrt{10\ S(estrutura) + 2.4}}$ | 0.1514 |
| Fechadura | $\frac{0.5}{\sqrt{S(fechadura) + 26.5}}$ | 0.096 |
| Chave | $\frac{2.275 + 4.5S(chave)}{\sqrt{45.5 + 45\ S(chave)^2}}$ | 0.4617 |
| Estrutura da Janela | 0 | 0 |
| Cortina | 0 | 0 |

Tabela 2: Indexes de Sensitividade (S.I.s) dos componentes básicos da casa (extraída do artigo “A Framework for Security Measurement” de Wang e Wulf (1997)).

Uma análise da sensitividade a partir da tabela 2 acima mostra que ao melhorar a força da segurança da chave temos um maior aumento da segurança da casa como um todo. Um componente em que o S.I. é maior do que 0.1 é considerado um componente não trivial. Segundo os autores, a estrutura da janela e a cortina possuem valores iguais a zero por causa da seguinte condição: S(janela) > S(porta).

A análise de sensitividade pode também ser utilizada para validarmos o esforço modelado. O modelo produzido após a decomposição pode estar errado se o S.I. de algum componente tiver um valor muito mais alto do que dos outros. O processo de decomposição, associado com uma análise de sensitividade adequada, pode fornecer um guia útil para isolarmos as áreas de vulnerabilidade do sistema, identificarmos a fonte do problema e , por último, levar a descobertas de falhas de

segurança ou vulnerabilidades.

**Vantagens do método**

1. Método simples e de fácil entendimento

**Desvantagens do método**

1. Descrever atributos fundamentais com diferentes unidades e escalas leva a uma inconveniente situação de não poder comparar resultados de avaliação de segurança.

2. Decompor um sistema em partes menores pode ser um processo difícil.

3. Estabelecer o quanto um elemento afeta a segurança de outro é uma tarefa muito difícil.

4. Atribuir pesos sem subjetividade é muito difícil.

## 4.6 Redução da Superfície de Ataque

Os autores Manadhata e Wing (2004) apresentaram um trabalho focado em definir uma métrica para medir de forma sistemática uma superfície de ataque de um sistema. Intuitivamente, a superfície de ataque de um sistema constitui-se do subconjunto de recursos que um atacante pode utilizar para atacar um sistema. Quanto mais funcionalidades estiverem disponíveis a um usuário ou quanto mais recursos forem acessíveis por essas funcionalidades, mais exposta será a superfície de ataque. E, quanto mais exposta estiver, mais susceptível o sistema estará a ataques com chances de sucesso, portanto mais inseguro será. A métrica visa reduzir a superfície de ataque com o objetivo de diminuir a probabilidade de um possível ataque, o que tornaria o sistema mais seguro.

Segundo os autores, os ataques registrados nos últimos anos mostraram que certos recursos dos sistemas têm mais tendência de serem utilizados em um ataque do que outros. Por exemplo, serviços executados com privilégio de *root* estão mais sujeitos de virarem alvos de ataques do que serviços que operam sem grandes restrições de privilégios. Arquivos com informações sobre o controle e configuração de um sistema são mais prováveis de serem atacados do que arquivos com privilégios mais restritos. Uma medida da superfície de ataque indica o nível de dano que um suposto atacante pode, potencialmente, causar ao sistema e o esforço necessário para o atacante causar tal dano.

Sendo assim, a métrica proposta considera que nem todos os recursos de um sistema devem ser tratados de forma igual. Para medir a superfície de ataque precisamos identificar estes recursos e sua contribuição para esta superfície. A identificação dos recursos de um sistema, possíveis alvos de ataques, é feita através de um conjunto de propriedades destes recursos categorizadas em classes de ataque. Tais propriedades referem-se à oportunidade de ataque – ou "atacabilidade" (em inglês: *attackability*) – de um tipo de recurso, isto é, alguns tipos de recursos são mais vulneráveis a ataques que outros. . Normalmente, um atacante conecta-se a um sistema utilizando os seus canais (portas abertas), invocando os métodos deste sistema (API's) ou enviando e recebendo dados a

partir do sistema.

A contribuição de cada recurso depende do potencial dano do recurso, isto é, do nível de dano que um atacante pode causar ao sistema ao usar o recurso em um ataque, e do esforço necessário para que um possível atacante adquira os direitos de acesso necessários para ser capaz de utilizar estes recursos em um ataque. Quanto maior o dano, mais alta será a contribuição. Para estimar a contribuição de cada recurso para a superfície de ataque, os autores calculam esta contribuição em função dos atributos destes recursos. Por exemplo, para estimar a contribuição dos métodos, basta verificá-lo em relação aos seus direitos de acesso e concessão de privilégios. De forma similar, para estimar a contribuição dos canais, devemos considerá-la em termos dos protocolos e direitos de acesso do canal e para estimar a contribuição dos itens de dados, devemos considerá-la em termos dos direitos de acesso e tipo de itens. A superfície de ataque é medida, então, a partir destas três dimensões: método, canal e dados.

Com um conjunto de classes de ataque e duas versões diferentes de um mesmo sistema, é possível medir qual é o mais relativamente seguro submetendo-os a uma comparação em relação às classes de ataque propostas. As comparações podem ser feitas de formas diferentes. Um exemplo seria, para cada versão, deve-se contar o número de instâncias de cada classe de ataque (número de serviços rodando como *root* e a quantidade de *sockets* abertos) e comparar os números entre cada uma das respectivas versões. Podem-se refinar as contagens determinando pesos maiores para certas classes em relação a outras. Os pesos representariam a probabilidade de ataque.

**O Método para Medir a Superfície de Ataque**

A medição da superfície de ataque consiste dos seguintes passos:

1. Dado um sistema, *S*, e seu ambiente, $E_s$, identificamos um conjunto *M* de pontos de entrada e saída, um conjunto *C* de canais e um conjunto *I* de itens de dados não confiáveis de *S*.

2. Estime o esforço e o potencial dano, d(m) para cada método m Є M, d(c) para cada canal c Є C e para cada item de dados i Є I.

3. A medida da superfície de ataque de S é a tripla:

$$<\sum_{m \in M} d(m), \sum_{c \in C} d(c) , \sum_{i \in I} d(i)>$$

O método utilizado nesta métrica é bem eficaz também quando comparamos sistemas similares. Porém, quando comparamos dois sistemas completamente diferentes o resultado não é muito bom, pois sistemas diferentes podem ter conjuntos de classes de ataque distintos.

A redução da superfície de ataque está baseada na redução do uso do código a zero. Tal redução seria alcançada, por exemplo, pelo emprego da regra 80/20, a qual pode ser definida como: se oitenta por cento dos usuários não usam determinada funcionalidade de um software, desligue a funcionalidade, mas permita que possa ser ligada quando necessário.

A combinação de qualidade do código desenvolvido com a técnica de redução da superfície de ataque pode levar a produção de um software mais seguro, o que, segundo o autor, é inatingível apenas com um código perfeito. Um processo simples para ajudar a reduzir a superfície de ataque e, conseqüentemente, melhorar a segurança do sistema seria:

1. Reduzir a quantidade de código em execução – aplicando a regra 80/20 a todas as áreas funcionais, se oitenta por cento dos usuários não a usarem, deve-se considerar a possibilidade de desligá-la.

2. Reduzir o acesso a pontos de entrada para usuários não confiáveis – restringir o acesso a quem não deveria, em essência, usar determinado recurso e fortalecer os princípios de autenticação.

3. Reduzir o privilégio para limitar o potencial de dano – reduzir os privilégios sob os quais o código deve ser executado.

4. Análise das entradas de dados – analisar o diagrama de fluxo de dados (DFD) ou diagrama de interação da UML (*Unified Modeling Language*) e identificar os pontos de entrada do sistema. A análise permitirá identificar se há a necessidade de aumentar o nível de

autorização nesses pontos.

5. Reduza a superfície de ataque de forma preventiva – descrever na fase de projeto como será a superfície de ataque, preferencialmente registrando em um documento. Alguns itens a serem considerados são: protocolos de rede, pontos (*endpoints*) que devem suportar autenticação ou autorização (atenção redobrada nos *endpoints* anônimos), desligar recursos por default, componentes reutilizáveis usados, identidades de processos de todos os códigos executados e contas instaladas de usuários. Dessa forma, os desenvolvedores conhecerão desde o início como será a superfície de ataque.

6. Medir a superfície de ataque – determine a superfície de ataque mínima no início e meça-a ao longo do desenvolvimento.

7. Grande superfície de ataque resulta em grande trabalho de segurança – se uma grande superfície de ataque for inevitável, o código deveria ser de boa qualidade, conservador e defensivo.

**Vantagens do método**

1. Pode ser empregada na fase de desenvolvimento

**Desvantagens do método**

1. Sistemas grandes tendem a ter grandes superfícies de ataque. Definir esta superfície não parece ser algo muito simples.

2. Necessita de um razoável gerenciamento de mudanças para controlar a evolução da superfície de ataque.

3. Sistemas com classes diferentes de ataque não podem ser comparados.

## 4.7 Proposta de um Framework para medir a segurança de um software

Como um primeiro passo em direção a definição de um framework estruturado de propriedades, os autores (Scandariato et al, 2006) começaram elicitando-os a partir da lista de práticas e princípios de segurança. Uma lista parcial destes princípios está disponível em em (Graff, 2003) e Stoneburner et al (2004). Os princípios propostos são mapeados para as fases do processo correspondente em que eles se aplicam.

Segundo ou autores, práticas e princípios são um importante ponto de partida para elicitar propriedades que podem ser medidas em diferentes níveis de abstração.

**Propriedades Consideradas na Metodologia**

**1. Faça pequeno e de forma simples (Keep it small and simple)**

Os mecanismos simples tendem a ter poucas falhas exploráveis e exigem menos manutenção. Além disso, como questões de gerência de configuração são simplificadas, atualizar ou trocar um mecanismo simples se torna um processo menos intensivo. Propriedades que podem ser usadas para estimar o esforço deste princípio são as seguintes:

- Tamanho
- Complexidade
- Tamanho da superfície de ataque

**Métricas**

Tamanho e complexidade podem ser medidos com métricas de software no nível do design e no código. Possíveis recursos para medir o tamanho da superfície de ataque em diferentes níveis de abstração já foram discutidos anteriormente.

**2. Separação de Interesses (Separation of concerns)**

Para obedecer este princípio enquanto desenvolver sistemas, características podem ser otimizadas de forma independente um da outra, para que falhas de uma não causem falhas nas outras também. Em geral, a separação de interesses torna mais fácil entender, design, e gerenciar sistemas independentes complexos. Para este princípio, a seguinte propriedade pode ser investigada.

- Grau de separação de interesses de segurança

**Métricas**

O grau de separação de interesses de segurança pode ser medido por meio de métricas que foram definidas nas pesquisas de desenvolvimento orientado a aspectos.

**3. Implemente Segurança em Camadas**

Projetistas de segurança devem considerar a abordagem de camadas para endereçar uma ameaça específica oureduzir uma vulnerabilidade. Por exemplo, o use de um roteador (filtrar pacotes) junto com um sistema para detectar invasão aumenta o esforço de um atacante ao explorar com sucesso o sistema. Uma propriedade simples foi definida para medir este princípio.

- Linhas de defesa

**Métricas**

Possíveis meios para avaliar a existência de um projeto (design) de segurança em camadas são:

- O número de controles de validação de dados por fluxo de informação
- O número de controles de autorização/autenticação por cenário de uso.

**4. Identifique e minimize criticidades**

Em relação ao termo “módulos críticos” os autores consideram todas as

entidades (dados ou métodos) que são vulneráveis a ataques. Um módulo pode ser avaliado como crítico a partir de vários critérios definidos, tais como: está ele seguro, está ele localizado em um ambiente seguro, o módulo é um importante ativo do proprietário do software, e o módulo está fundamentado no projeto e, por conseguinte, pode ser um alvo de um ataque “Denial of Service”? Contudo, também o número de módulos completamente confiáveis, isto é, componentes intencionalmente não submetidos a um exame detalhado de segurança, devem ser mantidos tão longe quanto possível das interfaces do sistema. Conseqüentemente, a propriedade relevante a ser medida é:

- Número de módulos críticos

**Métricas**

A identificação de métodos pode empregar diagramas UML como entrada. Por exemplo, diagramas de deployment descrvem as informações relacionadas à localização de uma aplicação, e tal informação pode ser utilizada para apontar relações de confiança, ambientes de deployment não confiáveis, e possíveis gargalos (Dos). Para fazer a identificação de módulos que são importantes (asset-wise), técnicas de análise de riscos devem ser usadas. Mais adiante métricas de interesse são o “número de entidades confiáveis”, que têm que ser minimizadas, e a “aferição do acomplamento dos componenetes”, que podem ser usados para identificar módulos fundamentais. ( por conseguinte sensível ao Dos)

5. **Alguns Indivíduos devem ser responsabilizados por seus atos** (em inglês: *accountability*)

Sistemas de software devem manter um registro das atividades para certificar se os recursos de sistema estão funcionando adequadamente e que eles não estão sendo utilizados de forma não autorizada. A trilha de auditoria pode ser usada tanto para monitorar recursos, e também como evidência no caso de violações à política de segurança. Neste caso, é interessante avaliar a seguinte propriedade:

- Grau de responsabilidade final (*accountability*)

**Métricas**

Uma abordagem ingênua para avaliar a propriedade acima é representada ao contar o número de operações não-auditadas (em relação a arquitetura) ou métodos (em relação ao design) com respeito ao número total de operações/métodos. Estas métricas podem ser checadas cuidadosamente com aquelas da seção anterior, por causa da identificação dos módulos críticos que devem ser auditados mais cuidadosamente, por causa de sua natureza sensível.

**Vantagens do método**

1. Pode ser utilizado na fase de desenvolvimento do software.

2. Pode ser usado com outros princípios.

**Desvantagens do método**

1. Não foi comprovada pelo autor.



## 4.8 Análise dos Métodos Apresentados

Apesar de todas estas iniciativas para avaliação quantitativa de segurança, nenhum dos métodos aqui apresentados consegue determinar com precisão que um sistema é seguro. Os trabalhos de Brocklehurst et al.(1994), Alves e Foss (1995) , Voas et al. (1996) , Ortalo et al. (1999) focam nas vulnerabilidades de um sistema como uma métrica de segurança. A abordagem de usar vulnerabilidades como uma métrica são falhas porque não conseguem tratar futuros ataques ao sistema, por isso, não são totalmente indicadas para a avaliação da segurança de um software antes que seja colocado em produção. A métrica da redução da superfície de ataque parece ser interessante, mas necessita de estudos mais aprofundados para que seja plenamente utilizada.

Infelizmente, métricas úteis relacionadas com o desenvolvimento de produtos codificados de acordo com os requisitos de segurança do software ainda estão em sua infância, e não existe até hoje nenhum consenso de quais métricas constituem melhores práticas. Uma revisão da literatura revela a escassez de

trabalhos publicados, validados e aceito por todos, sobre métricas de segurança relacionadas com o ciclo de vida de desenvolvimento. Apesar de tudo, existem algumas métricas e práticas usadas no desenvolvimento de software que podem ser proveitosas se estendidas para endereçar requisitos de segurança.