Uma Ontologia para Mitigar XML Injection

Thiago M. Rosa, Altair O. Santin, Andreia Malucelli

Programa de Pós-Graduação em Informática (PPGIa) – Pontifícia Universidade Católica do Paraná (PUCPR) – Curitiba – PR – Brasil

{tmattosr,santin,malu}@ppgia.puc.br

Abstract. *The underlying technologies used by web services bring well-known vulnerabilities from other domains to this new environment. Anomaly-based intrusion detection approaches produce high false positive rates, while signature-based intrusion detection approaches do not detect attack variations. This paper presents a novel hybrid attack detection engine that brings together the main advantages of these classical detection approaches. An ontology is applied as a strategy-based knowledge-base to assist mitigating XML injection attacks, while maintaining low false positive detection rates.*

Resumo. *As tecnologias utilizadas em web services trazem vulnerabilidades conhecidas em outros domínios para este novo ambiente. As abordagens de detecção de intrusão baseadas em anomalia geralmente produzem alta taxa de falsos positivos, enquanto que abordagens baseadas em assinatura não detectam variações de ataque. Este artigo apresenta um mecanismo híbrido de detecção de ataques que agrega as principais vantagens destas abordagens clássicas. Aplica-se uma ontologia como a base de conhecimento de ataques baseada em estratégia (sequencia encadeada de ações) para mitigar ataques de XML injection, mantendo baixas as taxas de falsos positivos.*

1. Introdução

Web services vem sendo usados crescentemente em sistemas distribuídos na internet, já que provêm um meio padrão de interoperabilidade entre diferentes aplicações, que podem estar executando em uma variedade de plataformas e *frameworks* [Booth et al. 2004]. Entretanto, as tecnologias utilizadas por *web services* (e.g. SOAP − *Simple Object Access Protocol*, HTTP − *Hypertext Transfer Protocol* e XML − *Extensible Markup Language*) favoreceram a exploração de vulnerabilidades conhecidas neste novo ambiente.

De acordo com o relatório anual de segurança do *Open Web Application Security Project* [OWASP 2009], ataques de *injection* estariam entre as vulnerabilidades mais exploradas em 2010. Esta estatística se confirma no *ranking* das 25 falhas de software mais perigosas [CWE e SANS 2010], pois as duas primeiras posições da lista são relacionadas a *injections*.

Este artigo aborda especialmente ataques de XML *injection* − XML *Cross-Site Scripting* e XPath/XQuery *injection*. XML *injections* são ataques que produzem alguma mudança nos componentes internos de um documento XML tentando comprometer aplicações que executam *web services*. Isto pode ser alcançado inserindo conteúdo malicioso em uma mensagem XML, por exemplo, através da inserção de caracteres XML inválidos [CWE 2011].

Uma maneira de proteger *web services* de ataques de *injection* é através de

sistemas de detecção baseados em assinaturas [Siddavatam e Gadge 2008]. Uma assinatura é um *payload* que identifica um ataque através de um conteúdo malicioso. Sistemas de detecção baseados em assinaturas normalmente produzem baixa taxa de erros de classificação dos ataques – conhecidos como falsos positivos. Entretanto, uma limitação importante da detecção de ataques baseada em assinaturas é que esta abordagem não detecta ataques desconhecidos (novos), mesmo que estes representem pequenas variações de um *payload* conhecido.

Outra forma de proteger *web services* contra ataques de *injection* é através de sistemas de detecção baseados em anomalias [Yee, Shin e Rao 2007], que aplicam uma técnica normalmente baseada em algum tipo de comportamento (implementado como um perfil). Por exemplo, os ataques podem ser modelados/classificados em duas classes distintas, uma para comportamentos normais − contendo todas as ações esperadas para tal perfil − e outra representando ataques, i.e., envolvendo ações que não são consideradas normais. Técnicas de detecção baseadas em anomalias conseguem detectar novos ataques, mas na maioria das vezes produzem alta taxa de falsos positivos (erros de avaliação) na detecção.

A técnica de detecção baseada em assinaturas é a mais utilizada, porém, permite ataques do tipo *zero-day*, que ocorrem quando uma vulnerabilidade (falha de software) se torna publicamente conhecida antes que sua correção esteja disponível para ser inserida na base de assinaturas do sistema de detecção [Zero Day Initiative 2011].

Independentemente de como uma vulnerabilidade se torna publicamente conhecida, a efetividade de um ataque *zero-day* pode variar de horas até meses [Zero Day Initiative 2011].

A abordagem de detecção clássica constrói sua base de assinaturas catalogando os ataques independentemente um do outro. Neste caso, não é levado em consideração que a estratégia (*engine*) de um ataque é similar para a maioria das categorias e que geralmente só há variações no *payload*, gerado em função de alterações na estratégia de cada ataque. Porém, os resultados desta mudança são suficientes para confundir a abordagem de detecção por anomalias, produzindo alertas falhos (falsos positivos), ou driblar a *engine* de detecção da abordagem por assinaturas.

O objetivo deste trabalho é modelar os ataques através de uma estratégia representada por classes e seus relacionamentos em uma ontologia. Acredita-se que conhecendo a estratégia de um ataque, que define o relacionamento semântico entre elementos do mesmo, pode-se facilmente detectar variações nos *payloads* e, como consequência, adicioná-los automaticamente à base de conhecimento da ontologia.

A contribuição desta proposta consiste em prover uma abordagem de sistema de detecção para XML *injection* baseado em estratégia (XID), para também mitigar ataques *zero-day* resultantes de variações dos ataques contidos na ontologia. Já que ataques novos (desconhecidos) são derivados de estratégias conhecidas, deverá ser observada uma baixa taxa de falsos positivos na detecção. Para este propósito apresenta-se o sistema de detecção baseado em estratégia como uma abordagem híbrida − suportando detecção baseada em anomalias, derivada da detecção baseada em assinaturas. Aplica-se uma ontologia para construir a base de ataques de XML *injection* contra *web services*.

O restante do artigo está organizado da seguinte forma: a seção 2 aborda brevemente ontologia; a seção 3 explica como foi construída a ontologia baseada em estratégia; a seção 4 apresenta a proposta (XID) e alguns cenários de avaliação; na seção 5 são descritos alguns trabalhos relacionados e a seção 6 conclui o trabalho.

2. Ontologia

Uma ontologia descreve um vocabulário comum usando conceitos (objetos) e propriedades (relacionamentos) que são importantes para um determinado domínio. Esta descrição é alcançada através de um conjunto de definições que associam os conceitos à linguagem humana, descrevendo seus significados e um conjunto de axiomas (formais) que restringem a interpretação e o uso destes conceitos [Konstantinou, Spanos e Mitrou 2008]. Por exemplo, no domínio "alunos de uma universidade", conceitos poderiam ser aluno, professor, curso, disciplina, etc. Os relacionamentos poderiam ser "é aluno de", "é professor de", "é disciplina de", etc.

Outro recurso de uma ontologia é permitir interoperabilidade entre sistemas através de seu vocabulário comum, permitindo que inferências sejam feitas de acordo com os axiomas pré-definidos [Dou, McDermott e Qi 2004]. Axiomas são definições, expressas através de lógica de primeira ordem, que envolvem classes, instâncias, relacionamentos e operadores lógicos. Axiomas são utilizados para representar uma verdade reconhecida para um determinado domínio de conhecimento. Os mesmos são avaliados por máquinas de inferência – software que faz deduções a partir do conhecimento expresso em uma ontologia, com o intuito de derivar desta ontologia um novo conhecimento. Tomando como exemplo o domínio de conhecimento "alunos de uma universidade" citado acima, um axioma para tal domínio poderia ser "todo aluno do curso de matemática é aluno da disciplina álgebra", apresentado aqui em lógica de primeira ordem:

"AlunoDisciplinaAlgebra ≡ ∃éAlunoDe.CursoMatematica"

De acordo com Gruber [1993], os critérios preliminares que devem ser levados em consideração antes de se construir uma ontologia são clareza, coerência, extensibilidade e um mínimo compromisso ontológico.

Observando esses critérios, algumas escolhas devem ser feitas quando se vai construir uma ontologia. Por exemplo, definições de classes e axiomas devem restringir o número de interpretações possíveis para satisfazer o critério da clareza. Porém, minimizar o compromisso ontológico significa admitir vários possíveis modelos para um domínio de conhecimento. Portanto, decisões de modelagem devem ser tomadas de acordo com o objetivo da ontologia.

A principal vantagem de se utilizar ontologia para modelar dados é o fato desta prover uma semântica explícita e formal. Além disto, a ontologia pode ser compartilhada (utilizada em vários sistemas escritos em linguagens diferentes) e reutilizada − adaptada para contemplar outros objetivos. Através do uso de inferências, ontologias também podem prover informações sobre um determinado domínio que não estejam explicitamente definidas na base de conhecimento [Konstantinou, Spanos e Mitrou 2008]. Usando o exemplo do domínio de conhecimento "alunos de uma universidade", se existe um aluno do curso de matemática na base de conhecimento da ontologia, a informação de que esse aluno está inscrito na disciplina álgebra não precisaria ser manualmente adicionada na ontologia, pois esta informação seria automaticamente inferida a partir de um axioma.

3. Ontologia Baseada em Estratégia

Para satisfazer os critérios propostos por Gruber [Gruber 1993] na construção da ontologia, utilizou-se inicialmente a taxonomia de ataques apresentada pelo *website* da CAPEC [CAPEC 2011]. A CAPEC descreve mecanismos utilizados para explorar falhas de software de acordo com a perspectiva do atacante. Esta descrição é feita em

alto nível, por isto a ontologia foi refinada baseando-se também em algumas ferramentas de teste de segurança (seção 4.1).

A ontologia proposta é composta de classes e propriedades (Fig. 1), instâncias (Fig. 2) e axiomas. Para facilitar o entendimento, neste artigo utiliza-se somente uma classe de ataques a *web services* (*XMLInjection*) e três subclasses (*XPathInjection*, *XQueryInjection* e *XSSInjection*) para exemplificar a estratégia dos ataques. As duas superclasses da ontologia são *AttackAction* e *WebServicesAttack*. *AttackAction* possui subclasses contendo instâncias de ações suspeitas (*payloads*) capturadas na rede.

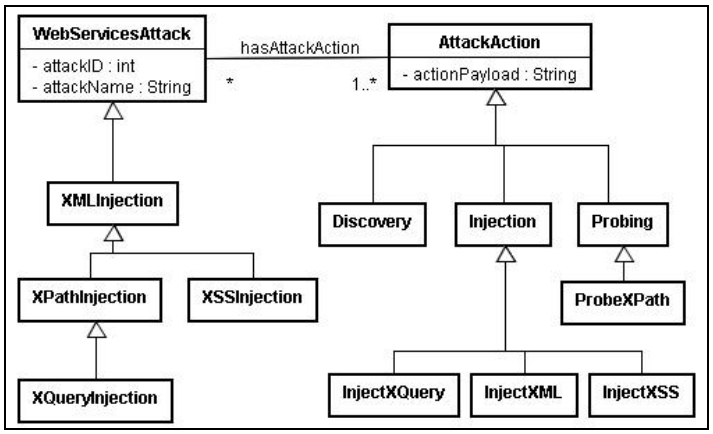

Figura 1. Diagrama de classes da ontologia

A classe *WebServicesAttack* possui subclasses representando categorias de ataques a *web services*. Cada uma destas subclasses possui instâncias representando assinaturas de ataques conhecidos. A assinatura de uma instância de ataque é representada na ontologia por relacionamentos que a mesma tem com ações – implementados através da propriedade *hasAttackAction*. Uma instância de ataque pode ter uma ou mais ações e uma ação pode ser parte de várias instâncias de ataque.

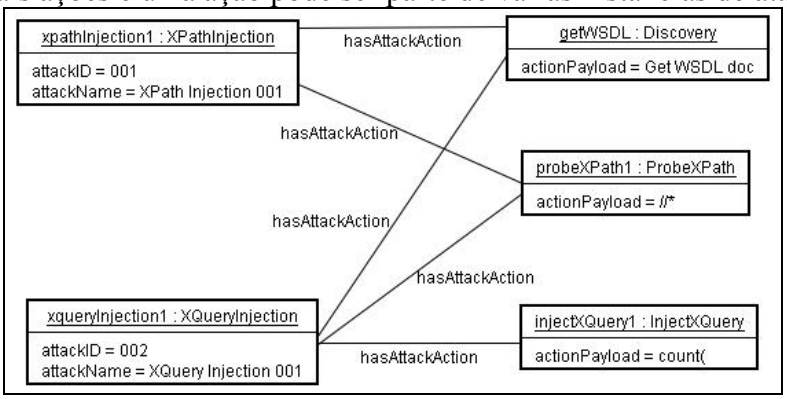

Figura 2. Exemplos de instâncias da ontologia

Na ontologia, os axiomas definidos para uma classe (categoria de ataque) restringem o número e o tipo de ações que as instâncias daquela classe terão. Além disso, neste caso os axiomas também podem ajudar a máquina de inferência a deduzir se um tipo de ataque ocorreu quando o padrão identificado (instância) ainda não está na base de conhecimento da ontologia, permitindo que este novo conhecimento seja adicionado à ontologia a partir desta dedução. Os axiomas foram modelados para cada

“XQueryInjection ≡ ∃hasAttackAction.Discovery ⊓ ∃hasAttackAction.ProbeXPath ⊓ ∃hasAttackAction.InjectXQuery” (1)

classe de ataque baseando-se nas estratégias de ataque propostas pela CAPEC, e foram validados/ajustados baseando-se em ferramentas de teste de segurança para web services (seção 4.1). Por exemplo, na ontologia criou-se o axioma (1) para a classe *XQueryInjection*, representado aqui através de lógica de primeira ordem.

Este axioma instrui a máquina de inferência a deduzir que qualquer ataque possuidor de uma ação do tipo *Discovery*, uma ação do tipo *ProbeXPath*, e uma ação do tipo *InjectXQuery* terá que obrigatoriamente ser uma instância da classe *XQueryInjection*. Na lógica de detecção do XID isto significa que um ataque do tipo *XQueryInjection* ocorreu.

Um exemplo prático do uso deste axioma específico é representado pelos pacotes (2), (3) e (4), detectados pelo XID antes da instância de ataque *xqueryInjection1* ser adicionada na ontologia.

"*GET* /WSDigger_WS/WSDigger_WS.asmx*?wsdl* HTTP/1.0\r\n" (2)

O pacote (2) representa um usuário acessando o documento WSDL de um *web service*, fazendo com que o XID crie um relacionamento (através da propriedade *hasAttackAction*) com a ação específica *getWSDL* (Fig. 2) − uma das instâncias da classe *Discovery* na ontologia.

"<query>//*</query>" (3)

O pacote (3) contém caracteres ('//*') que não deveriam estar presentes em nenhum campo de um pacote enviado por um usuário em uma operação *XPath*. Com isto, o XID cria outro relacionamento, desta vez com a ação *probeXPath1* (Fig. 2) − instância da classe *ProbeXPath* que representa o *payload* '//*'.

"<query>*count(/child::node())*</query>" (4)

Finalmente, o pacote (4) contém o *payload 'count('*, que representa uma ação ilegal de um usuário, neste caso tentando obter o número de ocorrências de algum elemento da estrutura da base de dados XML. Isto fez o XID criar um relacionamento com a ação *injectXQuery1* (Fig. 2), instância da classe *InjectXQuery* na ontologia.

O XID então inferiu na ontologia para verificar se essas ações poderiam representar um ataque. Observe que mesmo a base de conhecimento da ontologia não contendo esta instância de ataque específico, a máquina de inferência considerou o conjunto de eventos capturados como uma instância da classe *XQueryInjection* – de acordo com o axioma definido. Isto permitiu à ontologia aprender um novo ataque, pois em seguida o mesmo foi automaticamente adicionado pelo XID à base de conhecimento na forma de uma instância da classe *XQueryInjection*. Ou seja, na próxima vez que este ataque ocorrer será detectado sem que a máquina de inferência precise ser acionada.

As instâncias e relacionamentos na ontologia podem ser comparados com os padrões de ataques conhecidos em uma abordagem de detecção baseada em assinaturas. Porém, o padrão de ataque na ontologia representa a estratégia do ataque (sequência bem definida de ações), enquanto que na abordagem de detecção baseada em assinaturas o padrão é apenas um *payload*. Adicionalmente, as classes e axiomas da ontologia permitem que a máquina de inferência deduza que um ataque ocorreu mesmo que esse ainda não esteja na base de conhecimento, dando à abordagem proposta similaridade com a abordagem de detecção baseada em anomalias. Esta similaridade do XID com as outras duas abordagens de detecção de ataques o torna uma abordagem híbrida que explora as melhores características das outras duas.

4. Detecção de XML Injection Baseada em Ontologia

A ontologia proposta foi modelada utilizando a ferramenta Protégé [Stanford 2011] e a

linguagem *Web Ontology Language* [McGuinness e Harmelen 2009].

Utilizando o diagrama de classes da Fig. 1 modelou-se a estrutura das classes na ontologia (lado esquerdo da Fig. 3), criou-se instâncias de ataques (e.g. *xqueryInjection1*) e suas propriedades e relacionamentos (lado direito da Fig. 3).

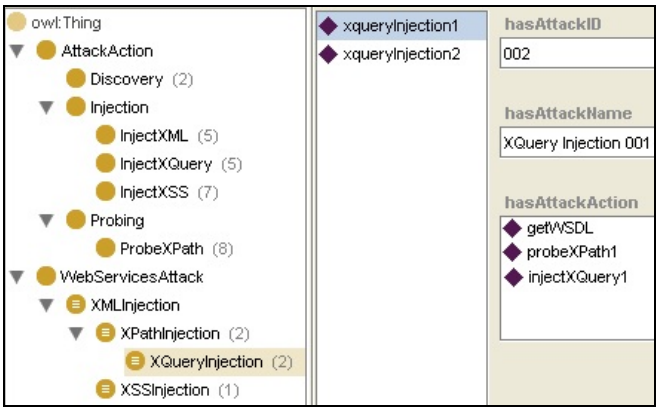

Figura 3. Visão parcial da ontologia proposta catalogada no Protégé

A máquina de inferência Pellet [Clarck&Parsia 2011] foi invocada através da interface DIG [Bechhofer 2006] no Protégé para inferir na ontologia. Na fase de modelagem da ontologia a máquina de inferência pode sugerir mudanças estruturais e identificar inconsistências, baseando-se nos axiomas criados para as classes.

Primeiramente, as classes *XQueryInjection* e *XPathInjection* estavam no mesmo nível (classes irmãs) abaixo da classe *XMLInjection*, como sugerido pela taxonomia da CAPEC. Entretanto após a execução do Pellet, esse sugeriu que *XQueryInjection* deveria ser uma subclasse de *XPathInjection*. Depois de analisar tal inferência, concluiu-se que esta sugestão fazia sentido, já que a *XQueryInjection* possui todas as restrições da *XPathInjection*. Também é possível encontrar fundamentação para isto no site da W3C [Boag et al. 2011], que sugere que a linguagem XQuery seja uma extensão da XPath. A inferência, neste caso, ajudou a aperfeiçoar a organização das classes de ataque na ontologia e, por conseguinte, a tornar os resultados da detecção mais efetivos.

4.1. Protótipo

Para avaliar a ontologia proposta foi desenvolvido um protótipo de sistema de detecção de intrusão (Fig. 4), utilizando a tecnologia Java [Oracle 2011].

Para capturar pacotes na rede foi utilizada a Jpcap [SourceForge 2011], uma biblioteca de captura de pacotes de rede para aplicações Java. A Jpcap pode filtrar pacotes IP e TCP, que são então transferidos para o módulo de detecção, cuja função é analisar conteúdos relativos a *web services* – conteúdo codificado em XML que serve para detecção no XID.

A base de conhecimento da ontologia foi manipulada pelo protótipo em tempo de execução utilizando o *framework* Jena [SourceForge 2011], que já possui uma interface nativa do Pellet. As instâncias de ataque foram consultadas utilizando SPARQL [Prud'hommeaux e Seaborne 2008], uma linguagem para consulta em arquivos RDF e OWL (arquivo da ontologia).

As estratégias dos ataques (extraídas inicialmente da CAPEC) foram refinadas e validadas utilizando o *framework* Metasploit [Metasploit 2011] e algumas das ferramentas de teste de segurança sugeridas pelo *Open Web Application Security Project* [OWASP 2011] para gerar ataques de XPath/XQuery *injection*. Também foram utilizados scripts contidos no site ha.ckers [Hansen 2008] para gerar ataques de XML

Cross-Site Scripting. O *sniffer* Wireshark [Combs 2011] foi utilizado para capturar pacotes na rede para análise funcional dos módulos do protótipo.

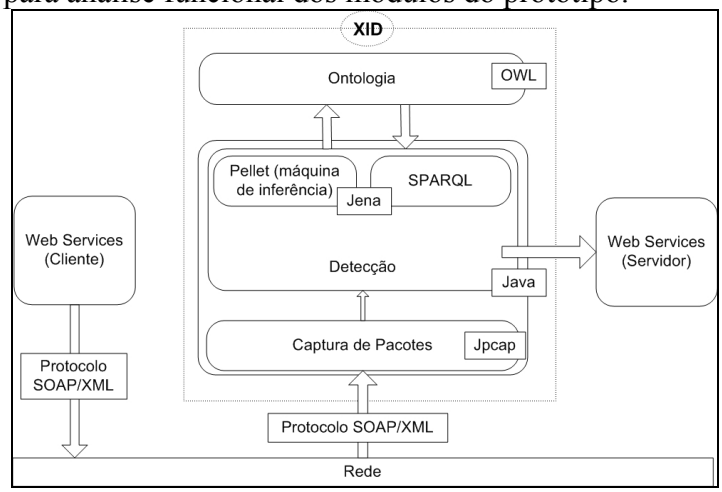

Figura 4. Visão geral da arquitetura do XID

Uma visão geral do funcionamento do módulo de detecção e inferência do XID é apresentada na Fig. 5. É possível observar que quando uma ação é detectada (evento i) em um pacote da rede pela primeira vez o protótipo cria uma instância (evento ii) de ataque (por enquanto sem persisti-la na ontologia), criando também um relacionamento da instância com esta ação (evento iii). Ou seja, a estratégia de um possível ataque começa a ser perseguida.

A seguir o protótipo verifica se existe alguma instância na base de conhecimento da ontologia que seja idêntica à criada anteriormente (evento iv) – o que indicaria que o ataque é conhecido (evento v). Isto é feito através de uma busca na ontologia por uma instância de ataque que esteja relacionada exatamente com as mesmas ações encontradas na detecção até este evento.

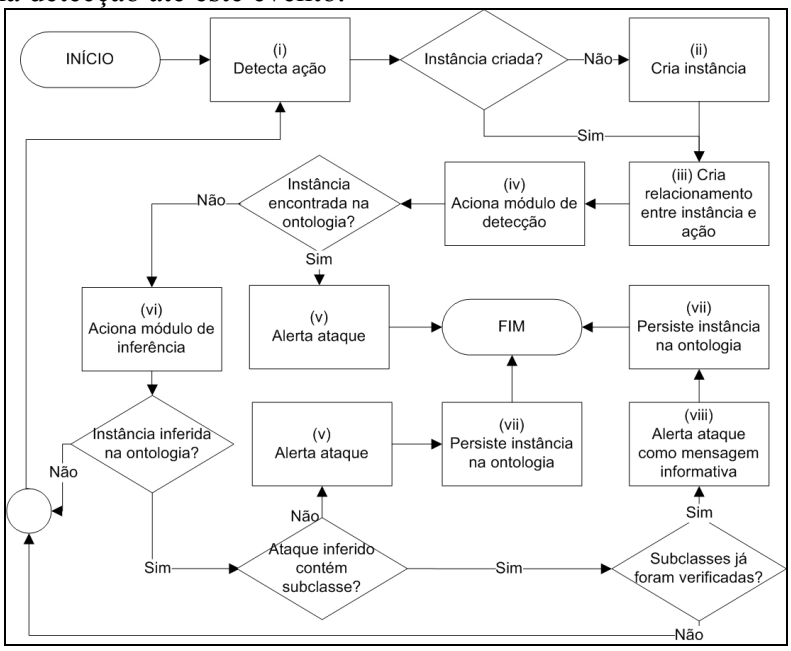

Figura 5. Fluxograma de detecção do XID

Quando não é encontrada uma instância idêntica a criada no evento ii, o protótipo tenta inferir um novo ataque a partir das informações contidas na base de

conhecimento da ontologia (evento vi), verificando se a instância pode ser considerada uma variação de ataque.

É através desta inferência, levando em conta classes e axiomas na ontologia, que a abordagem tenta aprender novos ataques e adicioná-los à base de conhecimento. Não chegando a uma inferência conclusiva, o protótipo continua analisando os próximos pacotes até encontrar uma nova ação. Esta nova ação é adicionada à instância (contendo agora duas ações) e novamente é verificado se um ataque ocorreu.

Este ciclo de eventos ocorre até que uma instância seja apontada como um ataque de acordo com o conjunto de ações detectadas, quando então a sequência do algoritmo de detecção é reiniciada.

Um ataque pode ser alertado pelo protótipo (evento v) se for encontrada uma instância idêntica na ontologia (através do SPARQL) ou se for inferida uma instância como um novo ataque (através do Pellet).

Quando um ataque é inferido, antes de alertar o ataque o protótipo verifica se a classe da instância inferida contém subclasses, o que significa que há possibilidade do ataque ser mais específico. Caso encontre subclasses na ontologia, o protótipo aguarda a próxima ação a ser detectada e faz uma nova inferência. Se esta nova inferência não alertar nenhuma subclasse mais específica, o protótipo alerta o ataque inferido inicialmente como uma mensagem informativa (evento viii), o que significa que o ataque pode não estar completo ou que se trata de uma nova categoria (subclasse) de ataques. Em caso contrário alerta o ataque – porque a subclasse mais específica foi alcançada.

Além de emitir o alerta o protótipo adiciona a nova instância à base de conhecimento da ontologia (evento vii), abaixo da classe correspondente. Assim, o protótipo e a ontologia (XID) podem ser considerados um sistema de detecção com uma abordagem híbrida. O XID permite detecção baseada em assinaturas através das instâncias, mas também permite a detecção de variações de ataques através de inferência nas classes e axiomas.

4.2. Avaliação quantitativa

Para avaliar a eficiência do XID foram desenvolvidos três cenários. No primeiro foi aplicado SPARQL, no segundo Pellet e no terceiro um arquivo texto – base de assinaturas Snort [Sourcefire 2011].

O objetivo dos experimentos foi comparar a escalabilidade e o desempenho da base de conhecimento da ontologia com os da base de dados baseada em assinaturas, pois havia a dúvida se devido à necessidade de inferências em alguns casos da detecção de ataques tal abordagem seria viável.

Para avaliação foi utilizada uma base composta de até 128 ataques catalogados no Protégé. Esta base iniciou com quatro classes de ataques pré-cadastradas (*XMLInjection, XPathInjection, XQueryInjection* e *XSSInjection*) e quatro instâncias de ataques (*xpathInjection1, xpathInjection2, xqueryInjection1 e xssInjection1*). Adicionando incrementos de 2^x (x = 2, 3, 4, ...) instâncias de ataques por vez a base foi sendo aumentada até totalizar os 128 ataques. Para compor a base, diversas instâncias de ataques e ações foram simuladas com o intuito de imitar as variações de ataques que vão sendo incorporadas à ontologia em uma aplicação real da proposta.

O primeiro experimento testou a ontologia consultando-a com apoio do SPARQL. Esta abordagem analisou os pacotes da rede procurando por ações maliciosas, que já estavam pré-cadastradas na base de conhecimento da ontologia,

relacionando as mesmas com instâncias de ataques que foram previamente introduzidas utilizando o Protégé.

O segundo experimento utilizou o Pellet para avaliar a ontologia em tempo de execução. Neste experimento não havia instâncias de ataques na ontologia quando a mesma foi consultada pelo SPARQL, portanto a máquina de inferência tentou derivar novos ataques baseando-se em axiomas pré-definidos para cada classe de ataques. Em outras palavras, já que o módulo SPARQL falhou, o módulo de inferência foi invocado para determinar se os conjuntos de ações sendo capturadas poderiam ser considerados novos ataques.

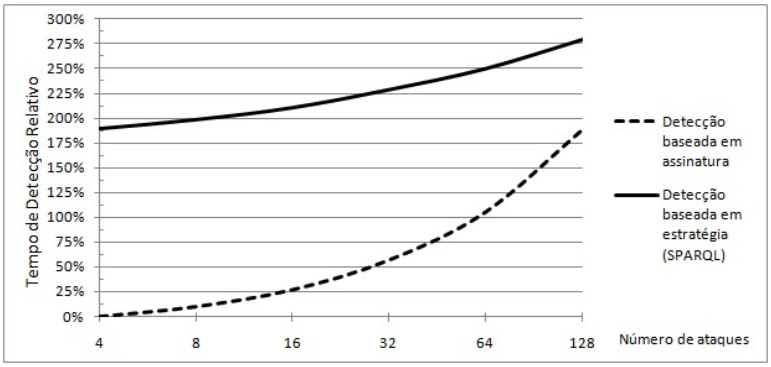

Figura 6. Tempo de detecção relativo (Assinaturas x SPARQL)

Sempre que uma nova sequência de ações é inferida como sendo um ataque (nova assinatura), uma nova instância para este ataque é automaticamente adicionada à base de conhecimento da ontologia. Assim, não se desperdiça tempo invocando o módulo de inferência caso este ataque específico seja capturado no futuro, pois o SPARQL irá detectá-lo primeiro, eliminando a possibilidade de ataques *zero-day*.

O terceiro experimento não utilizou a ontologia como base de conhecimento; foi utilizado o arquivo de texto contendo regras do Snort como base de dados de assinaturas – sem nenhuma técnica de otimização de consultas.

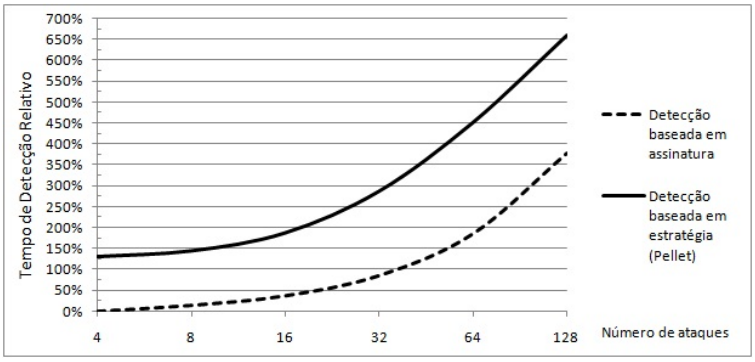

Figura 7. Tempo de detecção relativo (Assinaturas x Pellet)

O terceiro experimento foi executado duas vezes. Na primeira vez o arquivo de assinaturas foi consultado para procurar *payloads* (aleatoriamente inseridos do início ao final do arquivo), com o objetivo de comparação com o desempenho do SPARQL (Fig. 6). Na segunda vez, o arquivo de assinaturas foi consultado buscando-se por *payloads* (em número, variando entre 4 e128) que não se encontravam no mesmo – o objetivo foi comparar seu desempenho com o Pellet (Fig. 7).

O gráfico da Fig. 6 compara o experimento de detecção baseada em assinaturas com o experimento utilizando o SPARQL em um cenário onde os ataques estão pré-cadastrados no arquivo de texto e na base de conhecimento da ontologia.

O gráfico da Fig. 7 compara o experimento de detecção baseada em assinaturas com o experimento do Pellet, em um cenário onde as assinaturas sendo procuradas não estão presentes no arquivo texto e os conjuntos de ações sendo capturadas não correspondem a nenhuma instância de ataque na base de conhecimento da ontologia.

As Fig. 6 e Fig. 7 mostram gráficos comparando o tempo de detecção relativo, tomando como referência o tempo de busca por quatro ataques através de detecção baseada em assinaturas. A motivação para tal escolha é que, observando-se o ponto de partida da curva do Pellet na Fig. 7, nota-se que o mesmo gastou um tempo extra (se comparado a abordagem baseada em assinaturas), necessário para derivar novas instâncias através dos axiomas das classes de ataques da ontologia. Foi observado que o tempo gasto para avaliar as classes de ataques sem sucesso utilizando Pellet e o tempo gasto para inferência e adição de uma nova instância abaixo de uma das classes é similar.

O gráfico da Fig. 7 mostrou o pior caso para detecções do XID, pois as consultas resultam em perda de tempo de processamento pelo fato dos ataques não estarem na base, logo toda a base é consultada sem sucesso. Entretanto, mesmo o Pellet consumindo três vezes mais tempo do que a detecção baseada em assinaturas, sua abordagem ainda é vantajosa (em relação à detecção baseada em assinaturas) pelo fato de que esse módulo é executado uma única vez para cada nova variação de ataque. Uma vez que o Pellet infere uma nova instância de ataque, este módulo não será mais executado no futuro se o mesmo ataque for capturado novamente, pois o SPARQL irá detectar o ataque primeiro na sua próxima ocorrência. Já na detecção baseada em assinaturas este processamento sempre significará perda de tempo.

Observando a Fig. 6 constata-se uma tendência do desempenho do SPARQL ultrapassar a detecção baseada em assinaturas quando a base chegar a 512 ataques. Em aplicações reais as bases de ataques são muito amplas, logo a abordagem proposta teria a vantagem quantitativa de maior escalabilidade no tempo de detecção.

Baseando-se nos resultados relatados acima é possível concluir que a proposta deste trabalho, que mistura o primeiro e o segundo cenário, é vantajosa em relação ao terceiro cenário (abordagem clássica), obtendo a melhor relação custo benefício de detecção – baseada em assinaturas (SPARQL) *vs* baseada em conhecimento (Pellet).

4.3. Avaliação qualitativa

Em ontologias as definições de conceitos devem ser feitas através de axiomas lógicos [Gruber 1993]. Além disso, Gruber menciona que estas definições devem ser completas, ou seja, especificadas explicitando-se as condições necessárias e suficientes que as instâncias devem atender para serem classificadas por tais conceitos. Isto porque se uma instância atende às condições necessárias e suficientes (definidas através de axiomas) de uma classe, obrigatoriamente será inferida como instância daquela classe.

Considerando as afirmações de Gruber, em um mundo perfeito não haveria a possibilidade de alertas falsos serem gerados pelo XID, já que instâncias detectadas ou inferidas necessariamente atendem aos axiomas definidos para suas classes de ataques. Porém, Gruber também ressalta que se o resultado de uma inferência gerar um conhecimento que não corresponde à definição formal do domínio sendo representado, a ontologia pode estar incoerente. Ou seja, mesmo que os axiomas garantam que nada

diferente do que foi definido será detectado ou inferido pela proposta, sempre há a possibilidade de uma entrada incorreta na definição da ontologia por erro humano – assim como em qualquer outro sistema automatizado, se a entrada está incorreta, o resultado será impreciso. No caso da proposta a possibilidade de erro de especificação do ataque na ontologia é minimizada devido ao uso da taxonomia da CAPEC.

Se o ataque está completamente descrito (contendo as condições necessárias e suficientes que refletem a estratégia do ataque no mundo real) a probabilidade de falso positivo é nula. Porém, se o conjunto de atributos (relacionamentos) e restrições não estiver precisamente descrito há possibilidade de ocorrência de falsos positivos.

A partir destas considerações, dois cenários foram criados para testar a taxa de falsos positivos da abordagem proposta na detecção através do Pellet (módulo de inferência do XID que se utiliza dos axiomas para deduzir ataques), visando garantir que a ontologia tenha sido projetada de forma coerente.

Para o teste dos cenários foram criadas 128 instâncias (reais e simuladas) para avaliar os axiomas das quatro classes de ataques da proposta (*XMLInjection, XPathInjection, XQueryInjection* e *XSSInjection*). No primeiro cenário a lógica de detecção alertou ataques a cada inferência conclusiva, ou seja, assim que as restrições (axiomas) de qualquer classe eram atendidas o ataque era alertado. Já no segundo cenário (abordagem do XID), o protótipo verificou se os ataques continham subclasses (indícios de que um ataque mais específico estaria ocorrendo) antes de alertá-los.

A avaliação dos cenários foi dividida em duas fases. Na primeira fase, 64 dos 128 ataques criados foram utilizados, com o intuito de se fazer uma avaliação (treinamento) inicial da ontologia e ajustar as classes e axiomas caso fosse necessário. Portanto, 50% da amostragem de instâncias já estavam na ontologia ao término da primeira fase. Pequenos ajustes foram feitos na lógica de detecção do protótipo após os resultados deste treinamento, nenhuma alteração foi feita na ontologia. Na segunda fase, as 64 instâncias de ataque restantes foram simuladas para testar a porcentagem de acerto da proposta nas inferências feitas em tempo de execução, para ambos os cenários.

O resultado da avaliação para o primeiro cenário foi que 7/64 instâncias de ataque não foram detectadas corretamente. Estas sete instâncias continham três ações cada uma, uma ação da classe *Discovery*, uma da classe *ProbeXPath* e uma da classe *ProbeXQuery*. Estas sequências de três ações deveriam alertar um ataque do tipo *XQueryInjection* de acordo com o axioma definido para esta classe. Porém, o protótipo alertou para cada uma das sete instâncias como ataques de *XPathInjection* e de *XMLInjection*, respectivamente (totalizando 14 ataques alertados). Isto ocorreu porque a primeira parte dos ataques gerados (ação de *Discovery* e ação de *ProbeXPath*) satisfaz o axioma da classe de ataque *XPathInjection*, e a parte restante (ação de *ProbeXQuery*) satisfaz sozinha o axioma da classe genérica *XMLInjection*.

Assim, no primeiro cenário o resultado da dedução foi impreciso em alguns casos porque não foi considerado integramente o conjunto de ações que atendem as restrições da classe mais específica. Isto é, a dedução considerou apenas um subconjunto de ações que satisfaziam as restrições dos axiomas de classes mais genéricas – primeiras a serem testadas na lógica de detecção.

No segundo cenário, o protótipo foi programado para apenas alertar um ataque mais genérico após verificar as classes mais específicas do ataque sendo inferido. Desta forma, todas as 64 instâncias de ataque foram inferidas com sucesso e adicionadas às classes corretas na ontologia. Como neste cenário o protótipo aguardou a detecção da próxima ação antes de alertar um ataque – quando havia a possibilidade de um ataque

mais específico estar ocorrendo – não houve imprecisões na detecção.

Imprecisões de inferência não são consideradas como falsos positivos para o XID na abordagem do segundo cenário (ataque genérico alertado mesmo depois de verificadas as subclasses), porque falsos positivos são resultantes de classificação errônea de ações normais consideradas como ataques. Na abordagem proposta estas imprecisões são deduzidas em classes mais genéricas e, portanto, são alertadas como mensagens informativas e não como ataques. Isto é, quando o nível de especificidade do ataque sendo deduzido não é suficiente para atingir um grau de precisão confiável (depois de verificadas suas subclasses na ontologia), o mesmo é alertado como informativo ao invés de ataque.

Neste caso, quando a dedução não é conclusiva pode haver indícios de que o ataque detectado pode não estar completo (alguma ação das estratégias das subclasses foi perdida na detecção, por exemplo), ou uma nova categoria de ataque pode ter sido detectada. Este tipo de informação pode então ser investigado por um administrador/especialista para verificar se uma nova subclasse teria que ser criada ou se a instância se encaixa em alguma subclasse existente.

4.4. Considerações

Apesar de aplicar ontologia, a performance da detecção utilizando SPARQL é similar à abordagem baseada em assinaturas, levando em conta que instâncias de ataques estão pré-cadastradas na base de conhecimento. Além disso, o Pellet trabalha inferindo na ontologia para derivar novos ataques quando o SPARQL não encontra combinações exatas dos ataques. A inferência neste caso mantém a taxa de falsos positivos na detecção similar à de abordagens baseadas em assinaturas, pois novos ataques só podem ser derivados de classes e axiomas pré-cadastrados na ontologia.

A falha encontrada no primeiro cenário da avaliação qualitativa, que gerou imprecisão na detecção, foi devida a adoção de uma estratégia de detecção que buscava apenas identificar condições necessárias e suficientes, nem sempre levando em conta os axiomas das subclasses mais específicas. No segundo cenário esta falha não ocorreu, já que os axiomas das subclasses eram sempre verificados. Porém, quando classes de ataque mais específicas não eram encontradas, as instâncias deduzidas eram alertadas como mensagens informativas ao invés de ataques.

A inferência na ontologia pode ser utilizada tanto em tempo de execução (quando necessário) para aprender novos ataques, quanto na fase de modelagem para sugerir mudanças estruturais e encontrar inconsistências, otimizando a hierarquia de classes. Além disso, dependendo do tamanho da ontologia, redundâncias na definição da mesma que levariam horas para serem encontradas por um humano podem ser encontradas por uma máquina de inferência em alguns segundos.

5. Trabalhos Relacionados

A grande maioria das propostas encontradas na literatura técnica utiliza abordagens de detecção clássicas [Siddavatam e Gadge 2008][Yee, Shin e Rao 2007][Bravenboer, Dolstra e Visser 2010] e as que utilizam outras abordagens não trabalham com ataques a *web services* [Undercoffer et al. 2004].

Siddavatam e Gadge [Siddavatam e Gadge 2008] propuseram submeter requisições SOAP a uma série de algoritmos de teste para determinar se as mesmas poderiam representar algum tipo de ataque. Todas as requisições que não passam no teste são separadas para que ações posteriores possam ser tomadas. Os autores apresentam testes e resultados para sua proposta, porém não detalham suficientemente

os algoritmos e os mecanismos de detecção dos ataques.

Yee e seus colegas [Yee, Shin e Rao 2007] aplicaram um *framework* adaptável para tentar compensar as diferenças entre a detecção baseada em anomalias e assinaturas, através da integração de agentes, técnicas de mineração de dados e técnicas de lógica difusa. Para os autores, o uso destas técnicas permite tomar decisões em ambientes incertos e imprecisos. Porém, nenhum resultado concreto foi apresentado.

A abordagem de Bravenboer e seus colegas [Bravenboer, Dolstra e Visser 2010] sugere a incorporação de sintaxe, de acordo com as linguagens utilizadas no *guest* e *host* (e.g. *XPath* com Java), para gerar automaticamente o código que irá prevenir vulnerabilidades para ataques de *injection* (e.g. adicionando funções para filtrar caracteres inválidos). Os autores argumentam que a proposta é genérica, podendo ser aplicada com facilidade a qualquer combinação de linguagens. Porém, uma limitação apontada é o fato de nem todas as linguagens possuírem uma sintaxe livre de contexto.

A proposta de Undercoffer e seus colegas [Undercoffer et al. 2004] aplica ontologia para categorizar classes de ataques baseando-se principalmente no componente alvo dos mesmos, também considera os meios e as conseqüências do ataque e a localização do atacante. Esta ontologia é compartilhada por diversos sistemas de detecção de intrusão – o intuito é disseminar a todos um ataque descoberto por um deles. Porém, não é mencionado o uso de axiomas ou inferência, além disto, os autores não avaliam a proposta com testes.

Vorobiev e Han [Vorobiev e Han 2006] propuseram a abordagem que está mais próxima deste trabalho, os autores aplicaram uma ontologia especificamente para abordar o domínio de ataques a *web services*. Entretanto, a implementação da ontologia não foi encontrada e a proposta não utiliza inferência para detectar novos ataques. A ontologia é principalmente um 'dicionário' comum para diferentes ambientes.

6. Conclusão

Este artigo apresentou uma abordagem baseada em ontologia (XID) para proteger *web services* de ataques de XML *injection* e para mitigar o problema dos ataques que são variações de ataque conhecidos. Os ataques de XML *injection* que já estavam na base de conhecimento foram detectados com sucesso utilizando SPARQL para consultar a ontologia. Adicionalmente, as variações de *payload* foram detectadas utilizando inferência com nenhuma ocorrência de falsos positivos, já que novos *payloads* (instâncias) foram detectados baseando-se somente em classes e axiomas de ataques pré-existentes. Os novos *payloads* foram automaticamente adicionados à base de conhecimento da ontologia como instâncias – abaixo das classes relacionadas, eliminando os ataques que são variantes de ataques conhecidos.

O XID agrega as principais vantagens das abordagens clássicas de detecção. Permite a detecção de ataques conhecidos, como na abordagem baseada em assinaturas, e permite a detecção de novos ataques, como na abordagem baseada em anomalias. Esta segunda abordagem de detecção é feita pelo XID através de inferência na ontologia.

Como relação ao desempenho a proposta é comparável à detecção baseada em assinaturas quando os ataques são conhecidos. Quando os ataques não são conhecidos a proposta perde em desempenho quando comparada à abordagem por assinaturas, porém, neste caso podem ser detectadas variações de *payloads* com taxa nula de falsos positivos, mitigando ataques *zero-day,* por exemplo. Esta inferência de ataques que não estão pré-cadastrados na base não é possível na abordagem clássica de detecção baseada em assinaturas e imprecisa na baseada em anomalias.

Referências

Bechhofer, S. (2006) "DIG 2.0: The DIG Description Logic Interface", http://dig.cs.manchester.ac.uk.

Boag, S., Chamberlin, D., Fernández, M. F., Florescu, D., Robie, J. e Siméon, J. (2011) "XQuery 1.0: An XML Query Language (Second Edition)", http://www.w3.org/TR/xquery.

Booth, D., Haas, H., Mccabe, F., Newcomer, E., Champion, M., Ferris, C. e Orchard, D. (2004) "Web Services Architecture", http://www.w3.org/TR/ws-arch.

Bravenboer, M., Dolstra, E. e Visser, E. (2010). Preventing injection attacks with syntax embeddings. In *Science of Computer Programming archive*, pages 473-495.

CAPEC (2011) "Common Attack Pattern Enumeration and Classification", http://capec.mitre.org/data/graphs/1000.html.

Clarck&Parsia (2011) "Pellet: OWL 2 Reasoner for Java", http://clarkparsia.com/pellet.

Combs, G. (2011) "Wireshark – Go Deep", http://www.wireshark.org.

CWE e SANS (2010) "2010 CWE/SANS Top 25 Most Dangerous Software Errors", http://cwe.mitre.org/top25/index.html.

CWE (2011) "Common Weakness Enumeration", http://cwe.mitre.org/data/definitions/91.html.

Siddavatam, I. e Gadge, J. (2008). Comprehensive Test Mechanism to Detect Attack on Web Services. In *16th IEEE International Conference on Networks*, pages 1-6.

Dou, D., McDermott, D. e Qi, P. (2004). Ontology Translation on the Semantic Web. *In Journal on Data Semantics (JoDS) II*, pages 35-57.

Gruber, T. R. (1993). Toward Principles for the Design of Ontologies Used for Knowledge Sharing. In *International Journal Human-Computer Studies 43*, pages 907-928.

Hansen, R. (2008) "XSS (Cross Site Scripting) Cheat Sheet", http://ha.ckers.org/xss.html.

Konstantinou, N., Spanos, D. e Mitrou, N. (2008). Ontology and Database Mapping: A Survey of Current Implementations and Future Directions. In *Journal of Web Engineering*, pg. 1-24.

McGuinness, D., e Harmelen, F. (2009) "OWL 2 Web Ontology Language", http://www.w3.org/TR/owl-features.

Metasploit (2011) "Metasploit - Penetration Testing Resources", http://www.metasploit.com.

Oracle (2011) "For Java Developers", http://www.oracle.com/technetwork/java/index.html.

OWASP (2009) "The Open Web Application Security Project", http://www.owasp.org/images/3/3f/2009AnnualReport.pdf.

OWASP (2011) "The Open Web Application Security Project", http://www.owasp.org.

Prud'hommeaux, E., e Seaborne, A. (2008) "SPARQL Query Language for RDF", http://www.w3.org/TR/rdf-sparql-query.

Sourcefire (2011) "Sourcefire VRT Certified Rules - The Official Snort Ruleset", http://www.snort.org/snort-rules.

SourceForge (2011) "Jena – A Semantic Web Framework for Java", http://jena.sourceforge.net.

SourceForge (2011) "Network Packet Capture Facility for Java", http://sourceforge.net/projects/jpcap.

Stanford (2011) "The Protégé Ontology Editor and Knowledge Acquisition System", http://protege.stanford.edu.

Undercoffer, J., Pinkston, J., Joshi, A. e Finin, T. (2004). A Target-Centric ontology for intrusion detection**.** In *Proceedings of the IJCAI W. on Ontologies and Dist. Sys.*, pg. 47-58.

Vorobiev, A. e Han, J. (2006). Security Attack Ontology for Web Services. In *Proceedings of the Second International Conference on Semantics, Knowledge, and Grid*, paper 42 (6pp).

Yee, C. G., Shin, W. H. e Rao, G. S. V. R. K. (2007). An Adaptive Intrusion Detection and Prevention (ID/IP) Framework for Web Services. In *Proceedings of IEEE International Conference on Convergence Information Technology*, pages 528-534.

Zero Day Initiative (2011) "Zero Day Initiative", http://www.zerodayinitiative.com/advisories/upcoming/.