Mitigating XML Injection ∅-Day Attacks through Strategy-Based Detection Systems

Thiago Mattos Rosa | Exxon Mobil Information Technology
Altair Olivo Santin and Andreia Malucelli | Pontifical Catholic University of Parana

Ontologies can help build a strategy-based knowledge attack database. A novel hybrid attack detection engine brings together the main advantages of knowledge- and signature-based classical approaches.

Web services are increasingly used as distributed systems on the Internet; they provide a standard means of interoperation among different software applications running on a variety of platforms and frameworks.[1] However, the underlying technologies used by Web services, such as SOAP, HTTP, and XML, have fostered the deployment of well-known vulnerabilities in this new environment.

In a 2009 report, the Open Web Application Security Project (OWASP) predicted that injections would be among the most exploited vulnerabilities the following year.[2] This forecast was confirmed by 2010's top 10 attacks related to injections according to OWASP (www.owasp.org/index.php/Top_10) and the 25 most dangerous software errors according to the CWE/SANS report (cwe.mitre.org/top25/index.html).

This article specifically addresses XML injection attacks—those that produce some change in the XML's internal components that aims to compromise the Web service application. This can be achieved by, for instance, injecting malicious content into an XML message, such as invalid XML characters (cwe.mitre.org/data/definitions/91.html).

One way to protect Web services from injection attacks is through signature-based detection systems.[3] A signature is a payload that identifies an attack through some specific malicious context. Signature-based detection systems usually lead to low software-detection mistake rates—namely, false-positive rates.[4] However, one important limitation of signature-based attack detection is that it doesn't detect new unknown attacks, even if they have only small variations from a known payload.

Another way to protect Web services from injection attacks is through knowledge-based detection systems, which apply protection based on some kind of previously known and cataloged behavior.[5] Usually, two distinct classes are modeled: one for normal behavior, containing all expected actions that define such a profile, and another for attacks containing actions that aren't considered normal. Knowledge-based detection techniques can detect new attacks, but they usually produce a high false-positive rate.

Signature-based detection techniques are widely used, but they allow for ∅-day attacks, which occur when a vulnerability (software flaw) becomes publicly known before its fix is available.[6] No matter how a vulnerability is disclosed to the world, a ∅-day attack's effectiveness can vary from hours to months (www.zerodayinitiative.com/advisories/upcoming).

The classical detection system approach relies on building a signature-based database, cataloging attacks independently from each other. It doesn't take into account whether an attack's strategy is similar for a set of existing attacks.

Our goal in this work is to model attack elements as strategies—representing them by classes and relationships—in an ontology. Our belief is that by knowing the attack strategy, which defines a semantic relationship among attack elements, attack variations can be easily detected and automatically added to the ontology's database.

In this article, we describe an XML injection strategy-based detection system, XID, to mitigate the time gap for 0-day attacks resulting from an ontology's attack variations. Because many new and unknown attacks are derived from known strategies—considered signatures—low false-positive detection rates should occur. We present XID as a hybrid approach that supports knowledge-based detection derived from a signature-based approach. We then apply an ontology to design the knowledge database for XML injection attacks against Web services. (See the "Related Work in Attack Detection" sidebar for more information on other intrusion detection methods.)

Ontology

An ontology describes a common vocabulary with concepts and attributes that are important for a given domain. This description is achieved through a set of definitions that associate the concepts with human-readable text, describing their meaning and a set of formal axioms (premises) that restrict the interpretation and usage of these concepts.[7]

Another feature of ontologies is that they allow interoperability among systems through a *common vocabulary* and for inferences to be made into logical axioms.[8] Axioms are first-order logic statements that involve classes, instances, relationships, and logical operators and are used to express well-known truths for a given knowledge domain.

According to Tom Gruber, the preliminary criteria one should take into account before designing ontologies are clarity, coherence, extendibility, and minimal ontological commitment.[9] An ontology designer might need to make trade-offs. For example, definitions should restrict the possible interpretations of terms to satisfy the clarity criterion. However, minimizing ontological commitment means admitting several possible models. Therefore, a knowledge domain expert must make design choices according to the ontology's goal.

Using an ontology to model data brings the advantages of providing explicit and formal semantic relationships for such data, allowing for shareability (that is, being portable to several systems written in different programming languages) and reusability. Moreover, through inference, an ontology can also provide information about a certain domain that the knowledge database hasn't explicitly stated.[7]

Strategy-Based Ontology

To deal with Gruber's proposed criteria,[9] we used the taxonomy and attack features outlined on the Common Attack Pattern Enumeration and Classification (CAPEC) website to model the ontology's class structure and axioms (capec.mitre.org). CAPEC describes mechanisms to exploit software flaws from the attacker's perspective.

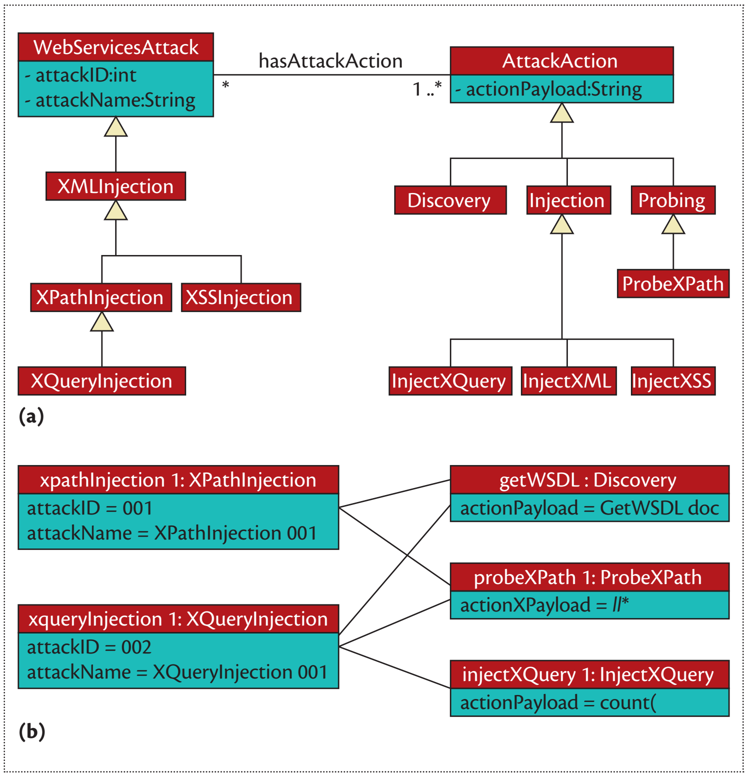

Figure 1. An overview of the XML injection strategy-based detection system (XID) ontology: (a) a class diagram and (b) an object diagram.

Classes

Our proposed ontology (see Figure 1) is composed of classes and properties (Figure 1a), instances (Figure 1b), and axioms. For the sake of simplicity, we used only one Web services attack class (`XMLInjection`) and three subclasses (`XPathInjection`, `XQueryInjection`, and `XSSInjection`) to exemplify the injection strategy. The two superclasses in the ontology are `AttackAction` and `WebServicesAttack`. `AttackAction` has subclasses containing instances that represent actions—for example, malicious content such as a payload that can be captured in the network.

The `WebServicesAttack` class has subclasses representing categories of Web services attacks. Each one of these subclasses has instances representing previously known attack templates of a specific strategy. In the ontology, the specific strategy for an attack instance is represented by the relationships it has with actions through the property `hasAttackAction`. One attack instance can have one or more `AttackAction`

Related Work in Attack Detection

Most proposals for Web services attack detection found in the technical literature use classical approaches that don't work properly. However, when an alternative approach for injection detection is used, it does not apply suitably to Web services attacks. The following approaches address some aspects related to our work.

Irfan Siddavatam and Jayant Gadge proposed running captured SOAP requests through a series of tests to determine if they could be classified as Web services attacks. Requests that fail the tests are filtered to be examined later.[1] The researchers presented tests and results, but didn't detail the test detection mechanisms in their proposal.

Chan Yee and colleagues proposed an adaptable framework, applying agents, data mining, and fuzzy logic techniques to compensate for differences between anomaly and signature-based detection for Web services.[2] According to the authors, these techniques allow for decision making in uncertain and inaccurate environments, but no concrete results were provided.

Martin Bravenboer and his colleagues suggested using syntax embedding, according to the guest and host languages (such as XPath and Java).[3] The aim is to automatically generate code that will prevent vulnerabilities to injection attacks—for example, adding character-escaping functions. The approach is generic and therefore can be applied to any language combination, but a limitation is the fact that not all languages have a context-free syntax.

Meisam Najjar and Mohammad Abdollahi Azgomi developed a hybrid Web services intrusion detection service to detect malicious behaviors in SOAP request/response messages.[4] The authors used misuse detection to verify an attack against a known set of signatures. Afterward, they use an anomaly-detection technique, based on normal profiles obtained through training, to alert unknown attacks. The authors mention that the proposal has an acceptable detection rate after a prototype implementation, but no actual test results were shown in the paper.

Zhichun Li and his colleagues proposed an approach to mitigate the Ø-day attack problem through an automated signature-generation system, focusing on polymorphic worms.[5] The system generated signatures based on invariant worm contents and on small content variations. Tests showed that the approach is viable both in performance and accuracy for the proposal's scope, but the article didn't describe Web services attacks.

Jeffrey Undercoffer and his colleagues applied an ontology to model attack classes mainly on the basis of the attack target, while also considering the attack's means and consequences.[6] The proposed ontology was shared by several attack detection systems with the objective of disseminating new attacks acknowledged by any of them. This approach, however, didn't use axioms or inference, and no tests were presented in their work.

Zakaria Maamar and colleagues presented an ontology-based approach to specify and secure contexts of Web services.[7] Each context in the ontology had specific encryption/decryption mechanisms to authenticate messages that were sent and received. The authors didn't model attacks in their ontology; they mainly focused on representing the context of Web services composition and security needs.

Artem Vorobiev and Jun Han proposed the closest approach to the work we present in this article; they applied an ontology to specifically address the Web services attacks domain.[8] However, the ontology implementation wasn't included in the research, and the proposal didn't use inference to detect unknown attacks; it mostly used ontologies to represent a common vocabulary.

References

1. I. Siddavatam and J. Gadge, "Comprehensive Test Mechanism to Detect Attack on Web Services," *Proc. 16th IEEE Int'l Conf. Networks* (ICON 08), IEEE, 2008, pp. 1–6.
2. C.G. Yee, W.H. Shin, and G.S.V.R.K. Rao, "An Adaptive Intrusion Detection and Prevention (ID/IP) Framework for Web Services," *Proc. Int'l Conf. Convergence Information Technology*, IEEE, 2007, pp. 528–534.
3. M. Bravenboer, E. Dolstra, and E. Visser, "Preventing Injection Attacks with Syntax Embeddings," *Science of Computer Programming*, vol. 75, no. 7, 2010, pp. 473–495.
4. M.S.A. Najjar and M.A. Azgomi, "A Distributed Multi-approach Intrusion Detection System for Web Services," *Proc. 3rd Int'l Conf. Security of Information and Networks* (SIN 10), ACM, 2010, pp. 238–244.
5. Z. Li et al., "Hamsa: Fast Signature Generation for Zero-Day Polymorphic Worms with Provable Attack Resilience," *Proc. 2006 IEEE Symp. Security and Privacy* (SP 06), IEEE CS, 2006, pp. 32–47.
6. J. Undercoffer et al., "A Target-Centric Ontology for Intrusion Detection," *Proc. IJCAI-03 Workshop Ontologies and Distributed Systems*, Morgan Kaufmann, 2004, pp. 47–58.
7. Z. Maamar, N.C. Narendra, and S. Sattanathan, "Towards an Ontology-Based Approach for Specifying and Securing Web Services," *Information and Software Technology*, Elsevier, 2005, pp. 441–455.
8. A. Vorobiev and J. Han, "Security Attack Ontology for Web Services," *Proc. 2nd Int'l Conf. Semantics, Knowledge, and Grid* (SKG 06), IEEE CS, 2006, p. 42.

instances, and one `AttackAction` instance can be part of several attack instances.

We produced the proposed ontology class structure (in Figure 2) by pulling the ontology class structure from Figure 1a. Additionally, we created attack instances (middle, Figure 2) and their properties and relationships (right, Figure 2) using the Protégé tool (protege.stanford.edu).

Axioms

A class's axiom will restrict the number and type of `AttackActions` its attack instances have in the ontology. Moreover, axioms can also help the *reasoner* (ontology inference engine) infer if a type of attack has happened. When an inferred pattern (instance) isn't in the ontology's knowledge database yet, this new attack can be added to it. For instance, the following `XQueryInjection` class is presented in first-order logic:

```
XQueryInjection ≡ ∃ hasAttackAction.
Discovery ⊓ ∃ hasAttackAction.
ProbeXPath ⊓ ∃ hasAttackAction.
InjectXQuery
```

This code tells the reasoner that any attack instance having at least one `AttackAction` of the type `Discovery`, one `AttackAction` of the type `ProbeXPath`, and one `AttackAction` of the type `InjectXQuery` will be an instance of the class `XQueryInjection`. For XID, this means that an `XQueryInjection` attack was detected.

An example of practical usage for this specific axiom is represented by packets i, ii, and iii, which XID detected using a parser to extract the attack pattern content from each packet. The packet sequence represents a new instance for the reasoner because the detection occurs before the attack instance `xqueryInjection1` was in the ontology.

Packet i represents a user accessing the Web Services Description Language (WSDL) document. XID created a relationship (through property `hasAttackAction`) with the specific action `getWSDL` (Figure 1b), one of the instances of the `Discovery` class in the ontology, identified in the payload as `GET` and `?wsdl`:

```
GET /WSDigger_WS/WSDigger_WS.asmx?wsdl
HTTP/1.0\r\n                                  (i)
```

Packet ii contains characters that shouldn't show up in any normal user input for an XPath operation (`//*`), causing XID to create another relationship, now with the action `probeXPath1` (Figure 1b), an instance of class `ProbeXPath` that represents the payload `//*`:

```
<query>//*</query>                           (ii)
```

Finally, packet iii contains the payload `count(`, which represents an illegal user action trying to obtain the number of occurrences of some XML structure element. This caused XID to create a third relationship, with the action `injectXQuery1` (Figure 1b) as an instance of class `InjectXQuery` in the ontology:

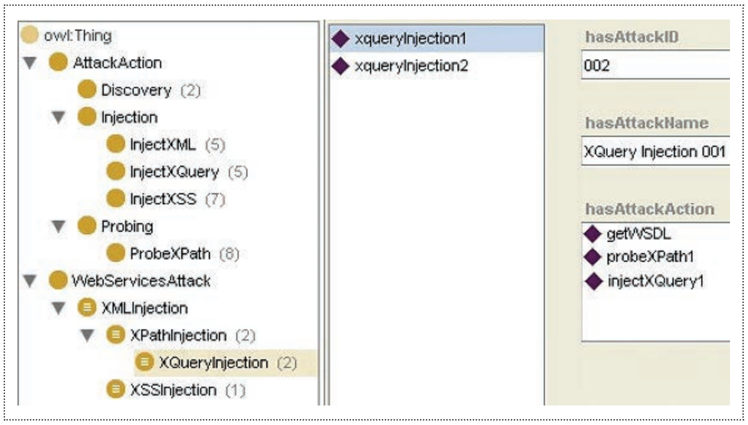

Figure 2. XID's class structure, with attack instances, properties, and relationships, designed with the Protégé tool.

```
<query>count(/child::node())</query>
                                        (iii)
```

Inference

XID, through the reasoner, infers over the ontology to check if these `AttackAction` instances represent an attack. Even though this specific attack instance wasn't in the ontology's knowledge database, the reasoner inferred the set of captured events as an instance of the class `XQueryInjection`—complying with the defined axiom. Thus, this procedure allowed XID to learn a new attack instance and automatically add it to the ontology's knowledge database.

The instances and relationships (specific strategy) in the ontology can be considered as known attack patterns in a signature-based detection approach. However, from the mechanism's perspective, the strategy is represented by a well-known set of actions (malicious/suspicious activity or content) that are semantically linked, whereas for a signature-based approach, an attack pattern is represented only by a payload. In addition, the classes and axioms (attack category strategy) let the reasoner infer whether an attack occurred, even if it's not yet in the ontology's knowledge database.

The addition of new attacks in the classical knowledge-based approach will generate new detection models (similar to the ontology's database) because the attacks as a whole will define the attack class.

In XID, the attack variation (new attack) will simply be added to the ontology's database, reusing or defining new classes and relationships. After that, nothing else is necessary, because in XID, each attack is modeled independently; each attack has its own profile. So, as the strategy defines an attack, the inferred (derived) new attack instance certainly will be an attack. This is a reason that XID maintains a low false-positive rate, whereas in the classical approach, complex

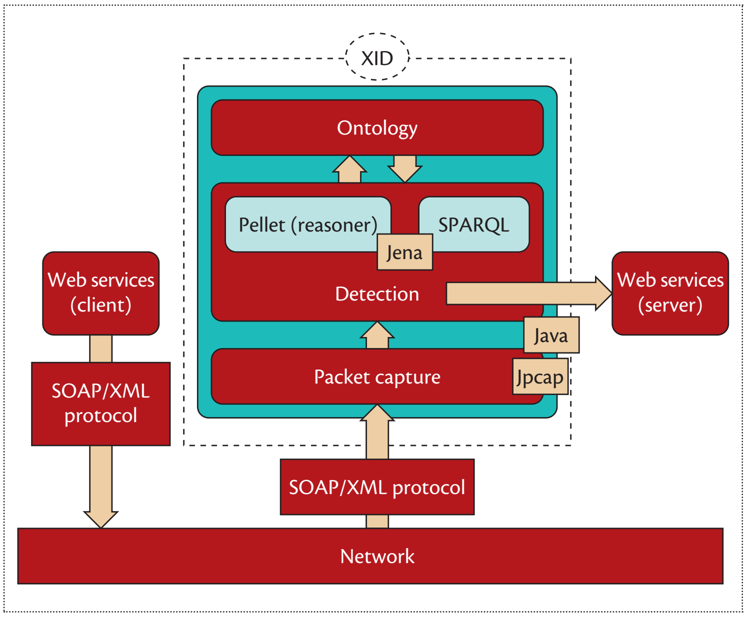

Figure 3. An overview of XID's architecture.

alert-correlation techniques are applied to obtain better false-positive rates.[10]

XML Injection Ontology-Based Detection

We designed our ontology using the Protégé tool and the Web Ontology Language (OWL; www.w3.org/TR/owl-features).

We used the Pellet reasoner (clarkparsia.com/pellet) through the DL Implementation Group (DIG) interface (dig.cs.manchester.ac.uk) in Protégé to infer in the ontology. When the ontology is under development, the reasoner can suggest structural changes due to inconsistencies on the basis of the axioms defined for each attack class.

In the ontology design, `XQueryInjection` and `XPathInjection` were sibling classes under `XMLInjection`, as suggested by the CAPEC taxonomy. However, after we ran Pellet, it suggested that `XQueryInjection` should be a subclass of `XPathInjection`. After our analysis, we agreed; the first class has all the restrictions that the second one has. We also found support for that assumption in the World Wide Web Consortium (www.w3.org/TR/xquery), which states that XQuery language is an extension of XPath. The inference, in this case, helped us enhance the attack classes' organization in the designed ontology. Figure 3 shows the detection system architecture for XID that we developed to evaluate the proposed ontology.

We used jpcap (jpcap.sourceforge.net) to capture packets from the network. Jpcap filters IP and TCP packets, transferring them to the detection engine. So, the detection engine analyzes only Web services–related content—XML content—for the purpose of XID.

The prototype handled the ontology knowledge database using the Jena framework (jena.sourceforge.net), which already supported the Pellet's reasoner. The ontology's knowledge database was queried for instances by SPARQL and RDF (Resource Description Framework; www.w3.org/TR/rdf-sparql-query), a SQL-like language used to query RDF and OWL files (the ontology's knowledge database). We applied a raw text file obtained from the well-known database of rules for the Snort intrusion detection system (www.snort.org/snort-rules) as a signature-based database for performance comparison purposes.

We studied attack strategies using the Metasploit framework (www.metasploit.com) and some of the security testing tools that the OWASP project suggested to generate XPath and XQuery injection attacks. We also used scripts from the ha.ckers website (ha.ckers.org/xss.html) to generate XML cross-site scripting attacks. We used the Wireshark sniffing tool (www.wireshark.org) to capture packets on the network for analysis of prototype functionalities.

XID Procedure

Figure 4 presents XID's procedure for inference and detection. When a strategy action is first detected in a network packet (Event 1), the prototype creates an attack instance (Event 2) and links it with that `AttackAction` (Event 3).

The prototype then queries the ontology through SPARQL, looking for an instance linked to exactly the same set of `AttackActions` (Event 4), indicating that the attack is known (Event 5). If no identical instance is found, the prototype tries to infer (using Pellet) a new attack—on the basis of classes and axioms from the ontology (Event 6)—indicating that the instance could be an attack variation. If no attack is inferred, the prototype continues analyzing the subsequent packets until a new `AttackAction` is found. This new `AttackAction` is then added to the attack instance (which is now linked to two actions), and the prototype evaluates the attack possibility again.

An attack can be alerted by the prototype (Event 5) if an identical instance is found or if a new instance is inferred based on the ontology. When a new instance is inferred, before alerting the attack, the prototype verifies if the instance's class has subclasses, in which case a more specific attack could be happening. If subclasses are found, the prototype waits for the next detected `AttackAction` for the same attack target and performs a new inference in the ontology.

If the new inference doesn't detect any of the subclasses, the original attack is alerted as an informative message (Event 8), meaning that the attack might not be complete yet or that a new subclass might need to be created for it. If the new inference does point to a subclass, the more specific attack is alerted. In both cases, the prototype adds the attack variation under the corresponding class in the ontology's knowledge database (Event 7).

Experiments

To evaluate XID's efficiency, we developed four experiment scenarios with the goal of evaluating the scalability and performance of the ontology's knowledge-based approach against signature-based databases, ensuring that the use of inference for new attacks wouldn't jeopardize the proposal's viability.

We applied an ontology database composed of 128 known attacks (added through Protégé) for the evaluation. The database initially contained four attack classes (`XMLInjection`, `XPathInjection`, `XQueryInjection`, and `XSSInjection`) and four attack instances (`xpathInjection1`, `xqueryInjection1`, `xqueryInjection2`, and `xssInjection1`). To compose a 128-attack database, we simulated several attack and action instances to mimic attack variations.

The first experiment queried the ontology through SPARQL. XID analyzed the packets, looking for sets of suspicious `AttackActions` that were already in the ontology's knowledge database, represented as attack instances, previously added through Protégé.

The second experiment used Pellet as the reasoner to infer within the ontology at runtime. Because the attack instances were missing when SPARQL queried the ontology database, Pellet tried to derive new attacks based on preset axioms for each attack class in the ontology.

The third experiment used the text file containing Snort rules as a signature database with no query optimization technique for searching within it. We queried the signature file for payloads that were in the text file (randomly inserted from the beginning to end) to compare with SPARQL's performance.

The fourth experiment also used Snort rules as a signature file, but in this case, we queried it for payloads that weren't in the text file; the goal was to compare it with Pellet's performance.

Results

Figure 5 shows the relative detection times of the four search attacks, using signature-based detection methods as bases for comparison.

The first comparison evaluates the first and third experiments (Figure 5a), where we observed that SPARQL started with a 190 percent slower detection time than the signature approach (with four attacks

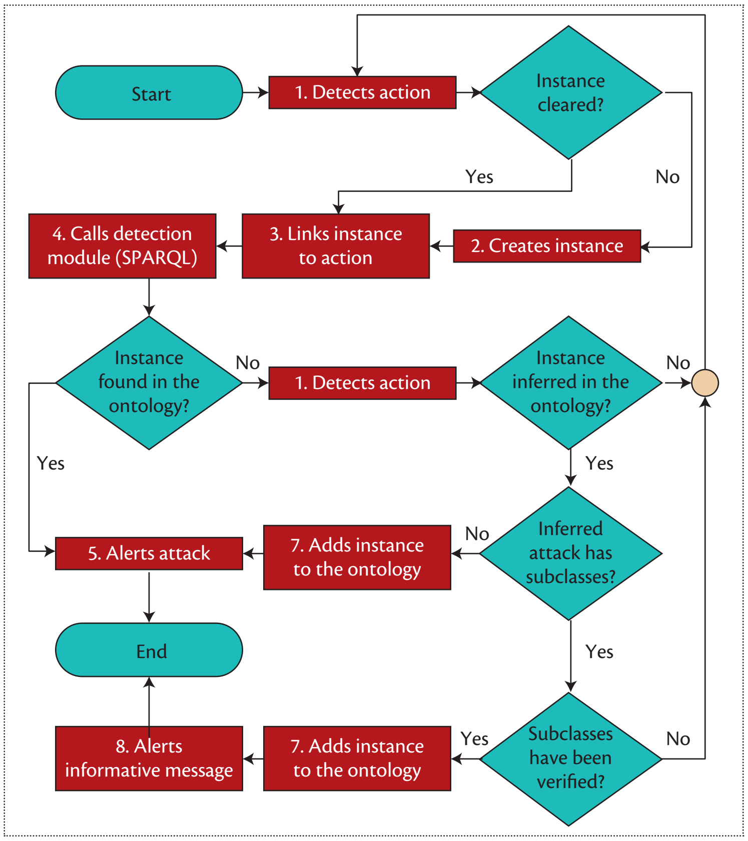
Start
1. Detects action
Instance cleared?
Yes
No
4. Calls detection module (SPARQL)
3. Links instance to action
2. Creates instance
Instance found in the ontology?
No
1. Detects action
Instance inferred in the ontology?
No
Yes
Yes
5. Alerts attack
7. Adds instance to the ontology
No
Inferred attack has subclasses?
End
Yes
8. Alerts informative message
7. Adds instance to the ontology
Yes
Subclasses have been verified?
No

Figure 4. XID's detection and inference procedure overview.

being searched). As the databases grew, SPARQL proved to be more scalable than the signature-based detection approach. When the databases reached 128 attacks, the signature-based detection was only 90 percent faster than SPARQL. Owing to the scalability tendency, we expect SPARQL to outrun the signature-based approach when the database reaches 512 attacks.

The second comparison (Figure 5b) evaluates the second and fourth experiments. Pellet started 129 percent slower than the signature-based approach (with four attacks being searched). Pellet was less scalable than the signature-based detection approach—when the databases reached 128 attacks, the fourth experiment was 281 percent faster than Pellet.

The second comparison reflects the worst-case scenario because the full databases were searched. Nevertheless, using Pellet is advantageous because for each unknown attack, the inference should be used only once, adding the attack instance to the ontology. Thus, the next time these specific sets of actions are captured, SPARQL will alert the newly added attack first. For signature-based detection, such processing will always mean wasted time because attack variations can't be detected.

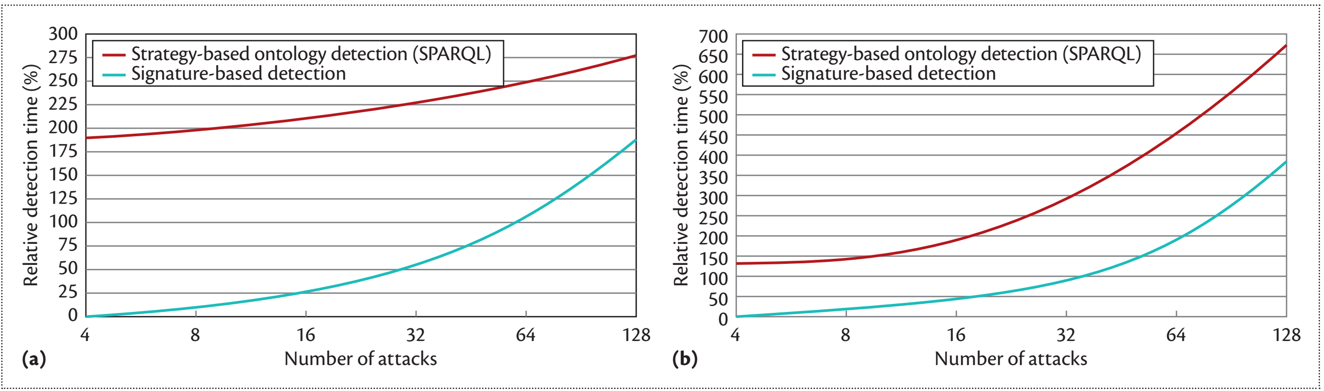

Figure 5. Relative detection time performance comparison for the (a) first and third experiments and (b) second and fourth experiments.

Observations

Based on these results, we concluded that XID, which mixes the first and second experiments, is advantageous against the classical approach of the third and fourth ones for obtaining the best detection trade-off—with SPARQL to detect known attacks and Pellet to infer new attacks.

Gruber mentions that in ontologies, classes (or concepts, as Gruber called them) should always be defined using axioms, and their definitions should be complete.[9] To express complete definitions, the axioms are set as necessary and sufficient conditions for each concept. This will allow a reasoner to infer attacks as instances of the respective classes. So, in an ideal case, XID would never generate false-positive alerts because detected or inferred instances must comply with the defined axioms for the attack classes.

Gruber highlights that there is, however, always the possibility of an incorrect input in the ontology's definition. In such a case, if the input model is incorrect, the results will be inaccurate, which is true for any other automated system. For XID, though, such an attack ontology specification error is initially minimized owing to the use of CAPEC's taxonomy.

Taking into account the false-positive detections that affect the XID alert's accuracy, we reevaluated the fourth experiment (the second experiment developed to test the Pellet approach, which used inference from the axioms to detect attack variations).

We observed that the prototype was alerting attacks immediately after reaching any axiom restrictions (necessary and sufficient conditions of any given class). This showed us that some attack instances were being incorrectly detected. These instances had three actions each, one `AttackAction` of class `Discovery`, one of class `ProbeXPath`, and one of class `InjectXQuery`. This set of three `AttackActions` should alert an `XQueryInjection` attack according to the defined axiom for this attack class. However, for each of the instances, the prototype alerted `XPathInjection` and `XMLInjection` attacks (totaling 14 inaccurately alerted attacks).

The inconsistences happened because the first part of these simulated instances (`AttackActions` of `Discovery` and `ProbeXPath`) satisfied the axiom restrictions of class `XPathInjection`, and the remaining part (`InjectXQuery`) satisfied the axiom restrictions of the generic class `XMLInjection`.

Attack alerts were inaccurate in some cases because the XID engine wasn't considering the complete set of actions that would satisfy the axiom restrictions of the most specific attack class. In other words, the inference was considering only a subset of `AttackActions` that satisfied the axiom restrictions of the generic classes—the first ones tested in the detection procedure.

Follow-Up Experiment

To fix the inaccurate XID alerts, we developed a fifth experiment to test the prototype, aiming to eliminate the attack-alert inconsistency.

We split the fifth experiment into two phases. The first phase used 64 of the 128 attack instances, with the objective of evaluating the ontology and tuning XID's inference engine when needed. Therefore, half of the instance sampling was already in the ontology by the end of the first phase. After the first phase of this experiment, we programmed the prototype to alert only a generic attack after its more specific subclasses had been verified, resulting in XID's definitive detection procedure (Figure 4). No design changes were needed in the ontology for this adjustment because it was an implementation issue.

The second phase used the remaining 64 attack instances to test the accuracy rate at runtime. All 64 attack instances were successfully inferred and added to the correct classes in the ontology. At that time, the prototype verified the more specific attack subclasses before alerting a generic attack; there were no errors (false positives) in the detection.

We didn't consider possible detection inaccuracies (a generic attack being alerted even after the subclasses had been verified) to be false positives for XID, because false positives in classical approaches result from incorrect evaluation of normal actions that are considered attacks or vice versa.

In the prototype approach, detection inaccuracies are identified and alerted as informative messages (Figure 4, Event 8). This might mean that the attack isn't completely defined or that a new subclass might need to be created in the ontology. To the best of our knowledge, this feature is exclusive to the strategy-based detection system.

A Web services administrator can consider the informative messages for investigation, aiming to tune the detection system, ensuring that the proposal approach doesn't produce false positives during detection. SPARQL also queries the ontology, but the difference is that it doesn't use inference. The detection through SPARQL is similar to the signature-based approach, provided that the attack instances being queried are present in the ontology's database. Moreover, Pellet uses inference to derive new attacks in the ontology when SPARQL doesn't find exact matches, giving the XID approach a similarity to a classical knowledge-based detection approach. The inference in this case maintains a false-positive detection rate similar to signature-based approaches because new attacks are only derived based on preset classes and axioms.

Another advantage is that 0-day attacks are eliminated for the inferred instances because a new attack instance is automatically added to the ontology's knowledge database at the moment it's first detected.

In future work, we intend to extend the ontology to contemplate other attacks that burden Web services, such as denial of service. As the number of attack classes and axioms grows, so does the inference power of our hybrid approach. ■

Acknowledgments

The National Council for Scientific and Technological Development (CNPq) grant 310319/2009-9 and State of Paraná Research Foundation (Araucária Foundation) grant 7374 partially supported this research.

References

1. D. Booth et al., "Web Services Architecture," working group note, W3C, Feb. 2004; www.w3.org/TR/ws-arch.
2. *OWASP Annual Report*, Open Web Application Security Project, 2009; www.owasp.org/images/3/3f/2009AnnualReport.pdf.
3. I. Siddavatam and J. Gadge, "Comprehensive Test Mechanism to Detect Attack on Web Services," *Proc. 16th IEEE Int'l Conf. on Networks* (ICON 08), IEEE, 2008, pp. 1–6.
4. N. Antunes and M. Vieira, "Benchmarking Vulnerability Detection Tools for Web Services," *Proc. IEEE Int'l Conf. Web Services* (ICWS), IEEE CS, 2010; doi;10.1109/ICWS.2010.76.
5. C.G. Yee, W.H. Shin, and G.S.V.R.K. Rao, "An Adaptive Intrusion Detection and Prevention (ID/IP) Framework for Web Services," *Proc. Int'l Conf. Convergence Information Technology*, IEEE, 2007, pp. 528–534.
6. E. Levy, "Approaching Zero," *IEEE Security & Privacy*, vol. 2, no. 4, 2004, pp. 65–66.
7. N. Konstantinou, D. Spanos, and N. Mitrou, "Ontology and Database Mapping: A Survey of Current Implementations and Future Directions," *J. Web Eng.*, vol. 7, no. 1, 2008, pp. 1–24.
8. D. Dou, D. McDermott, and P. Qi, "Ontology Translation on the Semantic Web," *J. Data Semantics*, vol. 2, 2004, pp. 35–57.
9. T.R. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," *Int'l J. Human-Computer Studies*, vol. 43, nos. 5–6, 1993, pp. 907–928.
10. B. Morin et al., "A Logic-Based Model to Support Alert correlation in Intrusion Detection," *Information Fusion*, vol. 10, no. 4, 2009, pp. 285–299.

Thiago Mattos Rosa is a system analyst at Exxon Mobil Information Technology. His research interests include intrusion detection, distributed systems, Web, and ontologies. Rosa received an MSc in computer science from Pontifical Catholic University of Parana. Contact him at tmattosr@gmail.com.

Altair Olivo Santin is a professor in the graduate program of computer science at Pontifical Catholic University of Parana. His research interests include usage and access control models, mechanisms for distributed systems, Web security, cloud computing security, intrusion detection systems, and digital forensics. Santin received a PhD in electrical engineering from Federal University of Santa Catarina. He's a member of IEEE, ACM, and the Brazilian Computer Society. Contact him at santin@ppgia.pucpr.br.

Andreia Malucelli is an associate professor in the graduate program of computer science at Pontifical Catholic University of Parana. Her research interests include software engineering, ontologies, multiagent systems, and information systems in healthcare. Malucelli received a PhD in electrical and computer engineering from the University of Porto. She's a member of the Brazilian Computer Society. Contact her at malu@ppgia.pucpr.br.