XML Security

Notes-set10 - XPath Injection

XML DBs

- With the increasing complexity of the applications, the integration of data from heterogeneous resources makes the data semi-structured or unstructured in which the data types and data values does not have a predefined structure.
- This need was one of the reasons that led to the development of eXtensible Markup Language (XML), which is designed to help store, structure, and exchange data on the internet.
- XML documents contain both the data and the informative relationship structuring of that data, so they are self describing.
- XPath is a query language used to select data from XML data sources. Usage of XML documents instead of relational databases makes web vulnerable to XPath Injection Attakcs (XPIAs).

XML-Injection

XML Specific Attacks:

- Attacks which are based on XML.
- As XPath is a query language for XML, XPath injection is not treated separately as another injection technique but is considered synonymous to XML Injection when the attack is carried through queries to XML database.

XML-Injection

- The use of XML documents instead of RDB makes web applications vulnerable to XML based injection attacks
- Allow one to query all items of the DB , but has no access control to protect it.
- XML Injections manipulate or compromise the logic of an XML application or Service
- Allow **malicious code** to be injected into web application by exploiting
    - 1. The improper input validation mechanisms of web application
    - 2. The application logic of web application

XML-Injection

- XPath injection takes place in web site, where invalidated user input is used for framing XPath queries.
- Dangers involved in Xpath injection are even more severe than SQL injection as XPath allows one to query all items of the database, unlike SQL DBMS in which user might restrict to some tables using access control.
- XML does not offer role-based security like other database applications .
- XPath statements do not throw errors when the search elements are missing from the XML file in the same way that SQL queries do when the search table or columns are missing from the database.

XML-Injection

- Because of the forgiving nature of XPath, it is much easier for an attacker to use malicious code in order to perform an XPath injection attack than an SQL injection attack.

- The lack of access control mechanism and difficulty in identifying errors necessitates identification of XPIAs.

XML-Injection

- Using XPath querying, a malicious user may extract a complete XML document, expose sensitive information, and compromise the integrity of the entire database.

- One can argue that the XML injections can be detected the same way SQL injections are detected, but the increased and complex use of the XML based database in the modern scenario demands for a new approach to detect the vulnerabilities in web applications

XML-Injection[2]

- **The attacker trys to inject XML**
  - The application relies on XML (stores information in an XML DB for instance)
  - The information provided by the attacker is evaluated together with the existing one.
- **We will see a practical example**
  - A XML style communication will be defined
  - Method for inserting XML metacharacters
  - Then the attacker has information about the XML structure
  - Possibility to inject XML data and tags.

[2]Source: OWASP Testing Guide

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<orders>
    <customer id="1">
        <name>Dimitris Mitropoulos</name>
        <email>dimitro@aueb.gr</email>
        <creditcard>12345678</creditcard>
        <order>
        <item>
            <quantity>1</quantity>
            <price>1.000</price>
             </item>
         <item>
             <quantity>2</quantity>
             <price>9.00</price>
             <name>CD-R</name>
         </item>
         </order>
    </customer>
...
</orders>
```

Example

The web application allows its users to search for items in their order history based on the price. The query that the application performs could look like this:

```
string query =
"/orders/customer[@id='" + cId + "']" +
"/order/item[price >= '" + pFilter + "']";
```

Without any proper input validation a malicious user will be able to select the entire XML document by entering the following value:

```
'] | /* | /anything[bar='
```

With a simple request, the attacker can steal personal data for every customer that has ever used the application. XPath statements do not throw errors when the search elements are missing from the XML file in the same way that SQL queries do when the search table or columns are missing from the database. Because of the forgiving nature of XPath, it is much more easier for an attacker to use malicious code in order to perform an XPpath injection attack than an SQL injection attack.

Example

- **Let us suppose we have the following xmlDB file (information is stored in an XML)**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>!c3</password>
    <userid>0</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
  <user>
    <username>Stefan0</username>
    <password>w1s3c</password>
    <userid>500</userid>
    <mail>Stefan0@whysec.hmm</mail>
  </user>
</users>
```

Insertion of a new user

- **Is done with a form (with the GET method)**
  - Three fields: username, password and email
- **Suppose the clients sends the following values**
  - username=Emmanuel
  - password=B3n0is7
  - email= emmanuel@bfh.ch
- **It produces the following GET request**

http://www.benoist.ch/addUser.php?username=Emmanuel&password=B3n0is7&email=emmanuel@bfh.ch

Insertion of a new user (Cont.)

- **The program will create a new XML `user`-node**

```
<user>
   <username>Emmanuel</username>
   <password>B3n0is7</password>
   <userid>500</userid>
   <mail>emmanuel@bfh.ch</mail>
</user>
```

- **The new entry in entered inside the XML DataBase**

Types of XML Injections

1. **Tautology Injection Attack:**

- The general goal of a tautology-based attack is to inject code in one or more conditional statements so that they always evaluate to true. The consequences of this attack depend on how the results of the query are used within the application. The most common usages are to bypass authentication pages and extract data.

- Example of such attack strings are: i) ’or 1 = 1 or ‘a’ = ‘a (Used for Login Bypass) ii) ’]|*|user[@roll = admin (To select all the nodes)

Types of XML Injections

2. Comment Injection Attack:

- By injecting < !--/-- > in username parameter, the application builds a node which is not a valid XML sequence.

- In most cases, when comment symbol is entered with user input, newly built node gets added to the XML database and the remaining part of the XML document gets commented.

Types of XML Injections

3. Piggybacked Injection Attack:

- In piggybacked injection attack, the XPath queries are appended along with another query using semicolon symbol (;).
- The appended XPath queries can be commands or scripts that can attack the web application.
- Similar to SQL Piggybacked Injection attack, XML piggybacked injection also affects the security of web application.

Types of XML Injections

4. Meta character Injection Attack:

Meta characters in XML are as follows:

1) Single and Double Quote (',"): If the input slot is not sanitized, and if an attribute value in the XML tag is injected along with this character, it could throw an exception during XML parsing.
2) Angular Parenthesis (> and <): If an open '<' or closed '>' angular parenthesis is injected in a user input parameter, then the resulting XML sequence won't be valid sequence, because, the application will build a new node for injected angular parenthesis.
3) Ampersand (&): Adding '&' can be used to exploit XML injection, if not encoded with &amp.

XML Metacharacters:

- **XML metacharacters like single quote, ampersand, double quote etc. when injected can cause the XML to become invalid and thus, errors are displayed showing vulnerabilities in the application.**

- **These errors show tag data and the structure of the document which can later be used to perform other attacks**

Vulnerability Testing

- **First step for XML Injection vulnerability**
  - Try to insert XML metacharacters
- **Metacharacters are:**
  - ’ (single quote)
  - " (double quote)
  - > and < (angular partentheses)
  - <!-- --> XML comment tags

Single Quote ’

- **This character could throw an exception during XML parsing**
- **Suppose we have the following attribute**

  <node attrib=’$inputValue’/>

- **So if:** `inputValue = foo’` **we obtain the following XML**

  <node attrib=’foo’’/>

  Which is a malformed XML expression: Exception at parsing the DB

Double Quote ″

- **Has the same meaning as single quotes**
  - Can be used instead of ’ if " is used in the document
- **So if we create the following XML**

<node attrib=”$inputValue”/>

and we set `inputValue = foo"` we obtain the following XML

<node attrib=”foo””/>

Which is also malformed

Angular parentheses 〈 and 〉

- **We create an unbalanced tag**
- Suppose we use the value `username = foo<` in the user XML-DataBase
- This creates a new user:

```
<user>
   <username>foo< </username>
   <password>B3n0is7</password>
   <userid>500</userid>
   <mail>test@test.de</mail>
</user>
```

- This document is not valid anymore.

Comments tags <!-- -->

- **This sequence of fharacters is interpreted as the beginning and end of a comment.**
- One can inject this sequence in the username parameter:
  `username= foo<!--`
- The application would create such a node:

```
<user>
   <username>foo<!- - </username>
   <password>Un6R34kb!e</password>
   <userid>500</userid>
   <mail>s4tan@hell.com</mail>
</user>
```

- Which is not valid

Ampersand &

- **Ampersand is used to represent XML entities**
  - Like &symbol;
  - Example < for representing the character <
- **Can be used to test injection**
  - One can give username=&foo
  - The created node contains:

    <username>&foo</username>

  - Which is a malformed expression, &foo should be ended with a ;
  - but &foo; would also be undefined.

Types of XML Injections

5. CDATA Injection:

- For escaping the blocks of text, CDATA sections could be used, which would otherwise be recognized as markup.

- If the characters/text enclosed with CDATA section, then the XML parser will not parse those characters/text.

- An attacker could inject the end character sequence (’]]>’) of CDATA string for invalidating the XML sequence.

CDATA section delimiters

- <![CDATA[**and**]] **are start and end delimiters of CDATA**
- **Inside a node a cdata section may be:**

```
<node>
    <![CDATA[<foo>]]
</node>
```

- <foo> **won't be parsed as markup is a character data.**
- **If a node is build in the following way**

```
<username><![CDATA[<$userName]]></username>
```

- **Tester will try to inject]] to invalidate the page.**
  - if username=]]¿
  - Then the node contains <username><![CDATA[]]>]]></username> which is not a valid XML fragment.

Result of the Test

- **Once having tested all the possiblities,**
  - Insert metacharacters of any type
- **Result**
  - The site is vulnerable to XML injection
  - The structure of the XML format has been discovered.

Possible Attacks using XML injection

- **XSS Cross Site Scripting**
- **External Entity**
- **Tag Injection**

CDATA Injection – XSS attacks

Blocks of data containing characters such as ‘¡’, ‘¿’ and so on which would otherwise be recognized as markup tags are escaped using CDATA Sections. The XML parsers do not parse these sections as XML.

Attackers can try to inject CDATA strings in order to invalidate the XML document. An example of the CDATA Injection is given below [?] [?] [?].

<![CDATA[<]]>script<![CDATA[>]]>alert(‘xss’)
<![CDATA[<]]>/script<![CDATA[>]]>

Use CDATA for XSS

- **Suppose we have a node containing some text that will be displayed back to the user**

```
<html>
$HTMLCode
</html>
```

- **Then an attacker can provide the following input**

```
$HTMLCode = <![CDATA[<]]>script<![CDATA[>]]>alert"-,
→('xss')<![CDATA[<]]>/script<![CDATA[>]]>
```

- **And we obtain the following node**

```
<html>
<![CDATA[<]]>script<![CDATA[>]]>alert('xss')
<![CDATA[<]]>/script<![CDATA[>]]>
</html>
```

Use CDATA for XSS (Cont.)

- **Durring the process, CDATA delimiters are eliminated, so the following HTML code is generated**

  <script>alert('XSS')</script>

Types of XML Injections

6. Alternate Encoding Injection Attack:

- In most injection attack detection mechanisms, attack strings are identified by checking against all possible Meta characters that are used for attack.
- But in alternate encoding attack, the attacker can bypass the detection mechanism by using character encodings for the attack string.
- The character encoding methods like hexadecimal encoding, ASCII encoding and Unicode character encoding are used by attacker to perform alternate encoding injection attack

Types of XML Injections

7. Tag Injection:

- XML tags can be used for injection.
- In input fields, new node can be injected by the attacker that can overwrite the existing nodes, if input validation is not done properly.
- For example, if there is a tag called *price* in the XML database and in the input field instead of writing a valid input string, Tag injection can be used to add new nodes to XML database or to modify existing values in XML database

Tag Injection

- The XML document can be injected with the tags if the attacker obtains the tag names in the document. In these attacks, the users inject valid XML tags into the data being sent to the server.

- The parser would parse these injected these tags as valid tags leading to an attack.

- An example could be injection of the following input for an email id field.

s4tan@hell.com</mail><userid>0</userid>

< mail>s4tan@hell.com

Tag Injection

- **The tester has gained information about the XML structure**
- **It is possible to inject data and tags**
- **Example: priviledge escalation attack in the previous example**
- **Suppose we have the following inputs**

Username: tony
Password: Un6R34kb!e
E-mail: s4tan@hell.com</mail><userid>0</userid><"-,
→mail>s4tan@hell.com

Tag Injection (Cont.)

- **The database becomes**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
        <user>
                <username>Stefan0</username>
                <password>w1s3c</password>
                <userid>500</userid>
                <mail>Stefan0@whysec.hmm</mail>
        </user>
        <user>
                <username>tony</username>
                <password>Un6R34kb!e</password>
                <userid>501</userid>
                <mail>s4tan@hell.com</mail>
                <userid>0</userid>
                <mail>s4tan@hell.com</mail>
        </user>
</users>
```

Tag Injection (Cont.)

- **Result**
  - User Tony gets the userid 0 (super-user)
- **Problem**
  - Userid tag appears twice for Tony
  - If XML documents is associated with a shema or a DTD, it will be rejected
  - UserID tag has cardinality 1.
- **Comment out the superfluous userid**

Username: tony
Password: Un6R34kb!e</password><!--
E-mail: --><userid>0</userid><mail>s4tan@hell.com

Tag Injection (Cont.)

- **The final XML is**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
 <user>
    <username>Stefan0</username>
    <password>w1s3c</password>
    <userid>500</userid>
    <mail>Stefan0@whysec.hmm</mail>
  </user>
  <user>
    <username>tony</username>
    <password>Un6R34kb!e</password><!-- </password>
    <userid>501</userid>
    <mail>--><userid>0</userid><mail>s4tan@hell.com</mail>
  </user>
</users>
```

Types of XML Injections

8. External Entity Injection:

- If the definition of an entity is a URI, the entity is called an external entity.
- External entities force the XML parser to access the resource specified by the URI.
- The intriguing thing is that most of the XML parsers are vulnerable to external entity injection by default. XML External entity (XXE) injection attack can be used to perform denial of service attacks and to gain access to unauthorized files in system

External Entity

- **The set of valid entities can be extended by defining new entities.**
  - If the definition of an entity is a URI, the entity is called an external entity.
  - External entities force the XML parser to access the resource specified by the URI (Unless configured to do otherwise).
- **Such an application is exposed to XML eXternal Entity (XXE) attacks.**
  - For performing a denial of service of the local system
  - gain unauthorized access to files on the local machine
  - scan remote machines
  - perform denial of service of remote systems.

Test for XXE vulnerability

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///dev/random" >]><foo>&
  →xxe;</foo>
```

- **This test could crash the web server (on a UNIX system),**
  - if the XML parser attempts to substitute the entity with the contents of the /dev/random file

Other XXE tests

- **Access the content of** /etc/passwd **file**

Types of XML Injections

9. Blind XPath Injection:

- Blind XPath Injection attack enables an attacker to extract a complete XML document used for XPath querying - without prior knowledge of the XPath query .
- The attack is “complete” since all possible data is exposed. The attack makes use of two techniques - XPath crawling, and Booleanization of XPath queries.
- Using this attack, it is possible to get hold of the XML “database” used in the XPath query. This can be most powerful against sites that use XPath queries (and XML “databases”) for authentication, searching, and other uses.

XML Signature Wrapping

- When a SOAP message is sent some parts of the message is signed in order to validate it. But, in such attacks the content of the signed part of the message is changed without invalidating the signature.

- *Content and Parameter Tampering* Since WSDL document describes how to use parameters of the application, the attackers can enter special characters or unexpected content to access illegal information

Schema Poisoning

- The attackers can learn about the schema files for XML documents through vulnerabilities like directory listing and so on.
- Then, the attacker accesses the XML schema from the server using some exploit, changes the schema and replaces it.
- Then he can perform other kinds of attacks such as buffer overflow attacks. The attacker may try to change the bounds of the xml tags and so on.

Conclusion

XML Injection

- Attacker can force the server to load entities from outside
- He can change the content of an XML database, and gain illegal priviledges in the application.
- Solution: Filter/Sanitize input, allow no metacharcters in your normal inputs, or escape them.

References

- **OWASP Top 10 - 2007**
  http://www.owasp.org/index.php/Top_10_2007
- **A Guide for Building Secure Web Applications and Web Services**
  http://www.lulu.com/content/1401012
- OWASP Testing for XML Injection
  http://www.owasp.org/index.php/Testing_for_XML_Injection_%28OWASP-DV-008%29
- Wikipedia.org Code injection.