NCSC Vulnerability Record

NCSC Reference: 497142
Vulnerability ID: 496067
February 2018
© Crown Copyright 2018

This information is exempt under the Freedom of Information Act 2000 (FOIA) and may be exempt under other UK information legislation. Refer any FOIA queries to ncsinfoleg@ncsc.gov.uk
About this document
This document details a vulnerability identified by the National Cyber Security Centre (NCSC). This vulnerability allows an attacker to gain Remote Code Execution (RCE).

Bug Bounty Payment
If this vulnerability is eligible for a Bug Bounty payment, we ask that the money be donated directly to NSPCC (Registered Charity Number: 216401, https://www.nspcc.org.uk). Please contact the NCSC mailbox to inform us of the donation amount and the donation date.

NCSC Contact Information
The vulnerability disclosure mailbox is 'security@ncsc.gov.uk'. Please contact us for our PGP key.

Crediting NCSC
NCSC would appreciate appropriate credit as 'The UK’s National Cyber Security Centre (NCSC)' in any advisories which you may publish about this issue.

Verification, Resolution and Release
Please inform NCSC via the 'security@ncsc.gov.uk' mailbox, quoting the NCSC Reference above, should you:
  • confirm that this is a security issue
  • allocate the issue a CVE identifier
  • determine a date to release a patch
  • determine a date to publish advisories

NCSC Disclosure Policy
NCSC has adopted the ISO 29147 approach to vulnerability disclosure and, as such, follows a coordinated disclosure approach with affected parties. We have never publicly disclosed a vulnerability prior to a fix being made available.

NCSC recognises that vendors need a reasonable amount of time to mitigate a vulnerability, for example, to understand the impact to customers, to triage against other vulnerabilities, to implement a fix in coordination with others, and to make that fix available to its customers. As this will vary based on the exact situation NCSC does not define a set time frame in which a fix must be made available, and we are happy to discuss the circumstances of any particular disclosure.

If NCSC believes a vendor is not making appropriate progress with vulnerability resolution, we may, after discussion with the vendor, choose to share the details appropriately (for example, with service providers and our customers) to ensure that we provide appropriate mitigation of the threat to the UK and to UK interests.

Disclaimer
Any NCSC findings and recommendations made have not been provided with the intention of avoiding all risks, and following the recommendations will not remove all such risk. Ownership of information risks remains with the relevant system owner at all times.
Summary

A Use After Free (UAF) vulnerability has been discovered in Xerces which affects the latest version. This vulnerability allows an attacker to gain Remote Code Execution (RCE).

This vulnerability has a Severity Score of 6.1 and a Medium Severity Rating (based on the Common Vulnerability Scoring System v2). The Severity Score and Severity Rating are calculated from the Exploitability and Impact Metrics in Table 1. Table 2 presents a summary of these vulnerability metrics.

Exploitability Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Vector</td>
<td>Local</td>
</tr>
<tr>
<td>Access Complexity</td>
<td>Low</td>
</tr>
<tr>
<td>Authentication</td>
<td>No Authentication</td>
</tr>
</tbody>
</table>

Impact Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality Impact</td>
<td>Partial</td>
</tr>
<tr>
<td>Integrity Impact</td>
<td>Partial</td>
</tr>
<tr>
<td>Availability Impact</td>
<td>Complete</td>
</tr>
</tbody>
</table>

Table 1: Exploitability and Impact Metrics

<table>
<thead>
<tr>
<th>Measure</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVSS Score</td>
<td>6.1</td>
</tr>
<tr>
<td>Severity Rating</td>
<td>Medium</td>
</tr>
<tr>
<td>CWE (Common Weakness Enumeration)</td>
<td>20: Improper Input Validation</td>
</tr>
</tbody>
</table>

Table 2: Summary of Vulnerability Metrics

Details

When parsing an XML document (via the SAX API) with an external DTD, Xerces loses track of which part of the code is responsible for releasing the object representing the DTD declaration:

```cpp
(void/xercesc/internal/IGXMLScanner.cpp):

1234 void IGXMLScanner::scanDocTypeDecl()
1235 {
...  
1480     if (fLoadExternalDTD || fValidate)
1481     {
...  
1532     DTDEntityDecl* declDTD = new (fMemoryManager) DTDEntityDecl(gDTDStr, false,fMemoryManager);
1533     declDTD->setSystemId(sysId);
1534     declDTD->setIsExternal(true);
[1] 1535     Janitor janDecl(declDTD);
1536     
1537     // Mark this one as a throw at end
1538     reader->setThrowAtEnd(true);
1539
```

[1] Reference to line 1532 in the code snippet.
1540 // And push it onto the stack, with its pseudo name
[3] 1541 fReaderMgr.pushReader(reader, declDTD);
1542
1543 // Tell it its not in an include section
1544 dtdScanner.scanExtSubsetDecl(false, true);
1545 }
1546 }
1547 

1. Allocate a new DTDEntityDecl to represent and keep track of the DTD declaration. As this is a SAX parser, we keep a stack of tokens to keep track of where we are (managed by fReaderMgr).
2. Use a Janitor templated class to wrap the allocated DTDEntityDecl. Janitor instances are used to automatically free the objects they wrap during destruction. Its use here appears to be a mistake.
3. Push the DTDEntityDecl, declDTD onto the state stack. When the stack-allocated Janitor at [2] goes out of scope, it will automatically free its data (i.e. declDTD):

(src/xercesc/util/Janitor.c):

41 template Janitor::~Janitor()
42 {
43   reset();
44 }
...
87 template void Janitor::reset(T* p)
88 {
89   if (fData)
90     delete fData;
91
92   fData = p;
93 }

(A default value for p is defined to be 0 in src/xercesc/util/Janitor.hpp.)
Now we're in a position where the fReaderMgr has a freed element at the top of its stack. This value is stored in the fCurEntity member when we reach the bottom of pushReader:

(src/xercesc/internal/ReaderMgr.cpp):

868 bool ReaderMgr::pushReader( XMLReader* const reader
869     , XMLEntityDecl* const entity)
870 {
871   //
872   // First, if an entity was passed, we have to confirm that this entity
873   // is not already on the entity stack. If so, then this is a recursive
// entity expansion, so we issue an error and refuse to put the reader
// on the stack.

// If there is no entity passed, then it's not an entity being pushed, so
// nothing to do. If there is no entity stack yet, then of course it
// cannot already be there.

//
// if (entity && fEntityStack)
// {
const XMLSize_t count = fEntityStack->size();
const XMLCh* const theName = entity->getName();
for (XMLSize_t index = 0; index < count; index++)
{
    const XMLEntityDecl* curDecl = fEntityStack->elementAt(index);
    if (curDecl)
    {
        if (XMLString::equals(theName, curDecl->getName()))
        {
            // Oops, already there so delete reader and return
            delete reader;
            return false;
        }
    }
}
// Fault in the reader stack. Give it an initial capacity of 16, and
// tell it it does own its elements.

// if (!fReaderStack)
// fReaderStack = new (fMemoryManager) RefStackOf(16, true, fMemoryManager);
// And the entity stack, which does not own its elements
// if (!fEntityStack)
// fEntityStack = new (fMemoryManager) RefStackOf(16, false, fMemoryManager);
// Push the current reader and entity onto their respective stacks.
// Note that the current entity can be null if the current reader
// is not for an entity.

// if (fCurReader)
// {
    fReaderStack->push(fCurReader);
    fEntityStack->push(fCurEntity);
//}
921 //
922 // Make the passed reader and entity the current top of stack. The
923 // passed entity can (and often is) null.
924 //
925 //
926 fCurReader = reader;
927 fCurEntity = entity;
928
929 return true;
930 }

ASAN Manifestation

Running a simple PoC through samples/StdInParse triggers ASAN. This is triggered because
the ReaderMgr::getLastExtEntityInfo method (called by the error-reporting path of
IGXMLScanner::scanDocument) reads and calls a method on the fCurEntity field through
ReaderMgr::getLastExtEntity:

998 const XMLReader*
999 ReaderMgr::getLastExtEntity(const XMLEntityDecl*& itsEntity) const
1000 {
1001 //
1002 // Scan down the reader stack until we find a reader for an entity that
1003 // is external. First check that there is anything in the stack at all,
1004 // in which case the current reader is the main file and that’s the one
1005 // that we want.
1006 //
1007 const XMLReader* theReader = fCurReader;
1008
1009 //
1010 // If there is a current entity and it is not an external entity, then
1011 // search the stack; else, keep the reader that we’ve got since its
1012 // either an external entity reader or the main file reader.
1013 //
1014 const XMLEntityDecl* curEntity = fCurEntity;
1015 if (curEntity && !curEntity->isExternal())
1016 {

Exploitation

If able to groom the heap between the Janitor releasing the DTDEntityDecl and the
getLastExtEntity method calling the isExternal method, it may be possible to gain control
of the instruction pointer and therefore achieve code execution.