Use After Free

Severity

High

Confidence

Not Tested/Unsure

Overview

A potential use after free vulnerability exists in the firefox desktop browser. The element pointer “head” on line 117 in the firefox/editor/libeditor/SetDocTitleTxn.cpp can be removed by malicious JavaScript running on an attackers HTML page. Once removed firefox will attempt to use what the pointer is pointing too (malicious or unknown memory stale memory).

Proof of Concept

The following code snippets demonstrates this vulnerability:

A raw pointer (head) is created on line 117 pointing to the head of the document. On line 152 the “AppendChild” method is called which eventually fires a DOM Event.

```
...SNIP...
dom::Element* head = document->GetHeadElement();
  NS_ENSURE_STATE(head);

  bool     newTitleNode = false;
  uint32_t newTitleIndex = 0;

  if (!titleNode)
  {
    // Didn't find one above: Create a new one
    nsCOMPtr<nsIDOMElement>titleElement;
         res = domDoc->CreateElement(NS_LITERAL_STRING("title"),
getter_AddRefs(titleElement));
    NS_ENSURE_SUCCESS(res, res);
    NS_ENSURE_TRUE(titleElement, NS_ERROR_FAILURE);

    titleNode = do_QueryInterface(titleElement);
    newTitleNode = true;

    // Get index so we append new title node after all existing HEAD children.
    newTitleIndex = head->GetChildCount();
  }

  // Append a text node under the TITLE
  //  only if the title text isn't empty
```

```
  if (titleNode && !aTitle.IsEmpty())
  {
    nsCOMPtr<nsIDOMText> textNode;
    res = domDoc->CreateTextNode(aTitle, getter_AddRefs(textNode));
    NS_ENSURE_SUCCESS(res, res);
    nsCOMPtr<nsIDOMNode> newNode = do_QueryInterface(textNode);
    NS_ENSURE_TRUE(newNode, NS_ERROR_FAILURE);

    if (newTitleNode)
    {
      // Not undoable: We will insert newTitleNode below
      nsCOMPtr<nsIDOMNode> resultNode;
      res = titleNode->AppendChild(newNode, getter_AddRefs(resultNode));
    }
```

Figure 1 – firefox/editor/libeditor/SetDocTitleTxn.cpp

```
  nsINode* AppendChild(nsINode& aNode, mozilla::ErrorResult& aError)
  {
    return InsertBefore(aNode, nullptr, aError);
  }
```

Figure 2 – firefox-49/dom/base/nsINode.h

```
  nsINode* InsertBefore(nsINode& aNode, nsINode* aChild,
                        mozilla::ErrorResult& aError)
  {
    return ReplaceOrInsertBefore(false, &aNode, aChild, aError);
  }
```

Figure 3 – firefox-49/dom/base/nsINode.h

```
nsIDOMNode *aRefChild, nsIDOMNode **aReturn)
{
  nsCOMPtr<nsINode> newChild = do_QueryInterface(aNewChild);
  if (!newChild) {
    return NS_ERROR_NULL_POINTER;
  }

  if (aReplace && !aRefChild) {
    return NS_ERROR_NULL_POINTER;
  }
```

```
  nsCOMPtr<nsINode> refChild = do_QueryInterface(aRefChild);
  if (aRefChild && !refChild) {
    return NS_NOINTERFACE;
  }

  ErrorResult rv;
  nsINode* result = ReplaceOrInsertBefore(aReplace, newChild, refChild, rv);
  if (result) {
    NS_ADDREF(*aReturn = result->AsDOMNode());
  }
  return rv.StealNSResult();
}
```

Figure 4 - /firefoxdom/base/nsINode.cpp

After calling replaceOrInsertBefore several methods fire dom events. An attacker could use javascript to create the proper event listener. Once the event fires, the javascript would remove the raw pointer “head” in figure one. At this point “head” is pointing to raw memory.

```
nsINode*
nsINode::ReplaceOrInsertBefore(bool aReplace, nsINode* aNewChild,
                               nsINode* aRefChild, ErrorResult& aError)
{
  // XXXbz I wish I could assert that nsContentUtils::IsSafeToRunScript() so we
  // could rely on scriptblockers going out of scope to actually run XBL
  // teardown, but various crud adds nodes under scriptblockers (e.g. native
  // anonymous content).  The only good news is those insertions can't trigger
  // the bad XBL cases.
  MOZ_ASSERT_IF(aReplace, aRefChild);

  EnsurePreInsertionValidity1(*aNewChild, aRefChild, aError);
  if (aError.Failed()) {
    return nullptr;
  }

  uint16_t nodeType = aNewChild->NodeType();

  // Before we do anything else, fire all DOMNodeRemoved mutation events
  // We do this up front as to avoid having to deal with script running
  // at random places further down.
  // Scope firing mutation events so that we don't carry any state that
  // might be stale
```

```
{
  // This check happens again further down (though then using IndexOf).
  // We're only checking this here to avoid firing mutation events when
  // none should be fired.
  // It's ok that we do the check twice in the case when firing mutation
  // events as we need to recheck after running script anyway.
  if (aRefChild && aRefChild->GetParentNode() != this) {
    aError.Throw(NS_ERROR_DOM_NOT_FOUND_ERR);
    return nullptr;
  }

  // If we're replacing, fire for node-to-be-replaced.
  // If aRefChild == aNewChild then we'll fire for it in check below
  if (aReplace && aRefChild != aNewChild) {
    nsContentUtils::MaybeFireNodeRemoved(aRefChild, this, OwnerDoc());
  }

  // If the new node already has a parent, fire for removing from old
  // parent
  nsINode* oldParent = aNewChild->GetParentNode();
  if (oldParent) {
    nsContentUtils::MaybeFireNodeRemoved(aNewChild, oldParent,
                                         aNewChild->OwnerDoc());
  }

  // If we're inserting a fragment, fire for all the children of the
  // fragment
  if (nodeType == nsIDOMNode::DOCUMENT_FRAGMENT_NODE) {
    static_cast<FragmentOrElement*>(aNewChild)->FireNodeRemovedForChildren();
  }
  // Verify that our aRefChild is still sensible
  if (aRefChild && aRefChild->GetParentNode() != this) {
    aError.Throw(NS_ERROR_DOM_NOT_FOUND_ERR);
    return nullptr;
  }
}
```

Figure 5 - firefox-49/dom/base/nsINode.cpp

“head” is now pointing to stale memory but is “used” again.

```
...SNIP...
    if (newTitleNode)
```

```
    {
      // Not undoable: We will insert newTitleNode below
      nsCOMPtr<nsIDOMNode> resultNode;
      res = titleNode->AppendChild(newNode, getter_AddRefs(resultNode));
    }
    else
    {
      // This is an undoable transaction
      res = editor->InsertNode(newNode, titleNode, 0);
    }
    NS_ENSURE_SUCCESS(res, res);
  }

  if (newTitleNode)
  {
    // Undoable transaction to insert title+text together
    res = editor->InsertNode(titleNode, head->AsDOMNode(), newTitleIndex);
  }
  return res;
}
```

Figure 6 - firefox/editor/libeditor/SetDocTitleTxn.cpp

The entire process with comments:

```
dom::Element* head = document->GetHeadElement(); //Raw pointer Declared
  NS_ENSURE_STATE(head);

  bool     newTitleNode = false;
  uint32_t newTitleIndex = 0;

  if (!titleNode)
  {
    // Didn't find one above: Create a new one
    nsCOMPtr<nsIDOMElement>titleElement;
    res = domDoc->CreateElement(NS_LITERAL_STRING("title"),
getter_AddRefs(titleElement));
    NS_ENSURE_SUCCESS(res, res);
    NS_ENSURE_TRUE(titleElement, NS_ERROR_FAILURE);

    titleNode = do_QueryInterface(titleElement);
    newTitleNode = true;
```

```
    // Get index so we append new title node after all existing HEAD children.
    newTitleIndex = head->GetChildCount();
  }

  // Append a text node under the TITLE
  //  only if the title text isn't empty
  if (titleNode && !aTitle.IsEmpty())
  {
    nsCOMPtr<nsIDOMText> textNode;
    res = domDoc->CreateTextNode(aTitle, getter_AddRefs(textNode));
    NS_ENSURE_SUCCESS(res, res);
    nsCOMPtr<nsIDOMNode> newNode = do_QueryInterface(textNode);
    NS_ENSURE_TRUE(newNode, NS_ERROR_FAILURE);

    if (newTitleNode)
    {
      // Not undoable: We will insert newTitleNode below
      nsCOMPtr<nsIDOMNode> resultNode;
      res = titleNode->AppendChild(newNode, getter_AddRefs(resultNode));
       // AppendChild method called which will eventually fire a dom event. Attacker
will use javascript on a malicious HTML page to wait for this event and remove the
element "Head" is pointing too.
    }
    else
    {
      // This is an undoable transaction
      res = editor->InsertNode(newNode, titleNode, 0);
    }
    NS_ENSURE_SUCCESS(res, res);
  }

  if (newTitleNode)
  {
    // Undoable transaction to insert title+text together
    res = editor->InsertNode(titleNode, head->AsDOMNode(), newTitleIndex);
    // Head Points to stale memory but is accessed.
  }
  return res;
}
```

Asset(s) Affected

- https://firefox-49/editor/libeditor/SetDocTitleTxn.cpp
    - Pointer to element “Head” Ln 117

Severity Description

An attacker can gain unauthorized access to sensitive data or pages without being required to authenticate to the application.

The following factors were used in determining the severity classification:

- Mitigating factors - None.
- Ease of exploitation - **Medium**. I haven’t tried to exploit the vulnerability as it’s in an area that I don’t know much about (The “editor” api’s) however this is known exploitable bug pattern.
- Technical impact - An attacker may gain remote code execution or disclose sensitive memory.
- Attack surface - Unauthenticated external attackers can launch these attacks via a crafted HTML page.
- Likelihood of attack - **High**. It is likely that an attacker would identify and exploit this vulnerability. Last years pwn to own used the same bug pattern. Also a class at blackhat identified it as well. See references.

Recommendation

The application should use a “smart” pointer instead of a raw pointer to ensure proper reference counting is kept for the life time of the object.

References

https://www.owasp.org/index.php/Using_freed_memory

https://github.com/struct/mms (WebKit Use after free Bug Pattern)

https://www.mozilla.org/en-US/security/advisories/mfsa2016-24/ (same bug pattern)