Is MathML dangerous?

Christopher Späth [1]

Abstract: HTML5 forms the basis for modern web development and merges different standards. One of these standards is MathML. It is used to express and display mathematical statements. However, with more standards being natively integrated into HTML5 the processing model gets inherently more complex.
In this paper, we evaluate the security risks of MathML. We created a semi-automatic test suite and studied the JavaScript code execution and the XML processing in MathML. We added also the Content-Type handling of major browsers to the picture. We discovered a novel way to manipulate the browser's status line without JavaScript and found two novel ways to execute JavaScript code, which allowed us to bypass several sanitizers. The fact, that JavaScript code embedded in MathML can access session cookies worsens matters even more.

Keywords: MathML; Web Security; XSS

1 Introduction

HTML has largely contributed to the success of the World Wide Web. HTML can be enriched with custom styles and scripts. With the current release of HTML5 even more technologies are meant to be processed by web browsers natively. SVGs [Da11] are one example where the integration into the web browser resulted in novel security threats. These have received considerable attention in the meantime from the literature [Za12], the academic field [He11b] and the community [He11a, DW17].
In this paper, we discuss MathML [Au14] which is also natively integrated into HTML5, however, has not received any attention from the research community yet. Developers and users are waged with uncertainty about the question which risks must be dealt with due to the support of MathML, both in popular hosting sites such as Wikimedia [wi15] and in sanitizers [Ah16]. We contribute a systematic and thorough investigation of the MathML specification and implementation in popular browsers and answer the following research questions: **RQ1:** Which elements and attributes of MathML can be considered dangerous? **RQ2:** How does the handling of Content-Types by major browsers affect the security of MathML? **RQ3:** How do sanitizers and websites deal with MathML?

Officially, Firefox and Safari implement MathML [Mo17b]. However, our evaluation demonstrates that both Internet Explorer and Chrome have already implemented the

[1] Ruhr-University Bochum, christopher.spaeth@rub.de

doi:10.18420/sicherheit2018_09

processing (parsing) of MathML. According to a publicly available report there is also a prototype implementation for Chrome available [Wa17]. Currently, Google Blogger [2], WolframAlpha[3], fmath.info[4], mathjax[5] and several other Math-related websites (e.g. mathmlcentral.com[6]) support MathML.
We investigated which of the elements and attributes can be misused for script execution (and thereby Cross-Site Scripting (XSS) attacks). We identified two novel XSS variants based on MathML.
Furthermore, we uncovered a flaw in the MathML specification leading to a Status Line Manipulation attack. This means, every browser which implements the specification is susceptible to this attack. This can be used to trick the user into executing unwanted actions (e.g., redirecting to an attacker's site). Currently, for such an attack to work, JavaScript code execution is necessary (e.g. by using an event handler which overwrites the default triggered action on the fly). Our approach improves on existing attacks in respect to that no script execution is necessary.
We investigated the handling of Content-Types by major browsers. This is important as MathML is both embeddable in HTML and XML. So the question arises, how is a document handled which has a MathML file extension (*.mml*), a XML MIME-Type (*application/xml*) and *HTML* markup as content? By using a semi-automatic test suite we provide a detailed insight into the handling of documents by browsers and investigate the XSLT processing. Additionally, by extending the scope to *Namespaces*, we show how MathML can be used to execute scripts and thereby access to authentication tokens, such as cookies, is possible.
To measure the impact of our vectors, we applied them to Web Application Firewalls and sanitizers. Our investigation yields two results: Firstly, currently none of the evaluated sanitizers implement specific actions to handle MathML but rather rely on general detection mechanisms. Secondly, when benign MathML is allowed in these sanitizers, 50 % can be bypassed using at least one of our vectors. No sanitizer detects the Status Line Manipulation attack. The evaluation of a sample of websites, which currently use MathML, supports our claim that the relation between MathML and security implications is not obvious to developers yet. We found one reflected and one DOM-based XSS vulnerability.
In summary, we deliver the following contributions.
• We developed a semi-automatic test tool to investigate the security of MathML regarding Script Execution, XML (DTD) and XSLT processing.
• We found two novel XSS vectors and a scriptless Status Line Manipulation attack.
• We extend our analysis to Content-Types and demonstrate that XSLT can be processed within MathML and cookie access is possible.
• Our evaluation shows that our identified attacks can be used to bypass state-of-the-art sanitizers.

[2] https://www.blogger.com
[3] http://www.wolframalpha.com/
[4] http://fmath.info/
[5] https://www.mathjax.org/
[6] http://www.mathmlcentral.com/

2 Technical Background

2.1 MathML

MathML is a markup language to express mathematical statements. MathML can be embedded within XML (XHTML) or HTML. Therefore, it is processed by an XML or HTML parser which transforms the input ”tag soup” to a structured output. Also HTML and XHTML - as well as other languages - can be embedded into MathML. Currently, MathML is officially supported by Firefox and Safari [ca17].

The Presentation Markup is used to construct mathematical expressions which can be rendered and displayed on screen. Token elements represent the smallest unit of an expression and are, for example, combined in layout elements, such as a fraction. Consider for example List. 1 which shows how a fraction would be constructed in MathML.

```
<math>
<mfrac>
<mn>2</mn>
<mn>3</mn>
</mfrac>
</math>
```

List. 1: Example of Presentation Markup

```
<math>
<apply>
<csymbol>times</csymbol>
<ci>a</ci>
<ci>b</ci>
</apply>
```

List. 2: Example of Content Markup

The element *mfrac* starts a fraction and expects exactly two arguments - a numerator and a denominator. The rendering of this markup would be similar to $\frac{2}{3}$. Of course there are a lot more expressions available, such as root ($\sqrt{2}$), sub- (i_1) and superscript (2^3).

The Content Markup is used to describe mathematical semantics unambiguously. Consider for example the multiplication of two operands. These could be represented as *a x b*, *a*b* or simply *ab*. A human reader familiar with common mathematical notations can probably infer the meaning from the context. However, a machine can not do so. Therefore, Content Markup offers a way to represent the semantics of a mathematical expression. The example in List. 2 demonstrates how to express a product in MathML.

The element *apply* expects as its first parameter an operator which is the multiplication-operator in this case. Depending on the chosen operator one or two additional parameters are expected - in this case at least two coefficients are needed.

MathML attributes can be of 18 different types, such as string, number, color or URI [Au14]. These can be applied to either Presentation, Content Markup or both, depending on the attribute. They influence, for example, the display of MathML (such as spacing, font), reference additional semantic resources for the elements or link ressources.

2.2 Cross-Site Scripting

Web-Applications are written to interact with the user. This interaction on the client-side is mainly achieved by the use of JavaScript, which can be used to read and write values of elements and properties. The JavaScript is scoped to an origin, which is a tuple of protocol, domain and port. Cross-Site Scripting (XSS) is a code injection attack, where an attacker can make the Web-Application execute his code. Thereby, this can lead to the unwanted modification of the website or theft of authentication tokens (e.g. cookies).

3 Methodology

We executed the tests for Firefox (55.0.3), Chrome (61.0), Internet Explorer (11) on Windows 7. The tests for Safari (10.1.1) were executed on a MacBook Pro Retina with OS X (10.12.6). On the server-side an Ubuntu 16.04 with Apache 2.4.18 was used in the default configuration.

3.1 Threat Model

We refer to the Web Attacker model [Ak10] and assume that the attacker can interact with any web application on the Internet (e.g. upload files, create posts) or host its own website. He cannot intercept or inject traffic into the victim's network connection. Web Applications may be protected by Web Application Firewalls or Sanitizers. The victim will freely interact with these web applications. An attack is considered successful if an attacker can bypass the security measures and execute scripts in the originating domain.

3.2 Script Execution

According to Section 6.4.3 [Au14], MathML can be parsed by either an HTML or an XML parser. Within certain MathML elements, such as *mtext*, *mo*, *mn*, *mi*, *ms*, *annotation* or *annotation-xml*, HTML elements are allowed and processed. We verified these specification guidelines in all browsers and particularly checked for script execution, XSLT and XML processing.

We investigated all 41 elements of the Presentation and eleven basic elements from the Content Markup. Additionally, we included the elements *semantics*, *annotation* and *annotation-xml*, which belong to both the Presentation and Content markup. We supplied JavaScript code (called %vector) as content of a MathML element as shown in List. 3.

The selection process for attributes was way more complex because of the exhaustive number of attributes and attribute data types available. Some attributes are available for all elements (both Presentation and Content Markup), some are only available for either one of those and some may only be available for selected elements. Therefore, we considered it appropriate to first analyze the attribute data types. Regarding script execution we primarily focused on the ones of type URI. This decision is based on the fact that this type is analogous to the URI type of HTML elements (e.g. <a href=...). We verified that *href* (and *xlink:href* respectively) can be used for script execution. Other attributes of type URI (math: (altimg | cdgroup | macros); mglyph: src; annotation: (definitionURL | src)) are currently not implemened by any of the browsers. To test URI attributes we supplied the vector *javascript:alert(1)* into the attribute value, as shown in List. 4. Then we clicked on the link. If an alert window opens we consider the attribute to have scripting capabilities. Plese refer to Table 1 for the list of elements and attributes which were found to be susceptible to script execution. The complete listing can be found in the extended version[7].

The element *maction* can bind an action to an expression. It has an attribute *actiontype* with the values: toggle, statusline, tooltip and input. We checked all of these values and will elaborate on the results in section 7. The elements *semantics*, *annotation* and *annotation-xml* facilitate the insertion of supplementary information for a mathematical expression. Each of these elements can declare the *encoding* of its content by using an eponymous attribute. Since MathML can be embedded in HTML and XML contexts, we focused our investigation on related Content-Types. We assigned the attribute *encoding* the values ”text/html”, ”application/xml” and ”application/xhtml+xml”. To cross-reference the implications with the chosen Content-Type of the containing document, we embedded each attribute value combination into an .html, .xml and .xhtml file to observe differences. This is shown in List. 7 (c.f. Appendix).

```
<math>
<mi>2 %vector </mi>
<math>
```

List. 3: Test Methodology for Elements

```
<math>
<mi href="%vector">2</mi>
<math>
```

List. 4: Test Methodology for Attributes

3.3 Content-Types

Although several browser vendors provide online documentation about how the browser treats certain MIME types [Mo08, Mi, We] these resources do not reflect important details. For our evaluation we take the file extension, HTTP Content-Type Header and selected elements and namespaces into account. In more detail, we check the file extensions *.html*, *.xml*, *.xhtml* and *.mml*. We evaluate if the file extension or the Content-Type header (*application/xml*, *application/xml+xhtml*, *text/html*) has precedence. To make sure the results are not influenced by the content of the file, we create one version with a random element (e.g. <greeting>) and one with an HTML element (<b>). Additionally, we observe the

[7] https://goo.gl/vqyY2i

impact of the XHTML namespace on XML-based Content-Types. Our results are described in section 4.

Limitations. We leave the investigation of mobile browsers with MathML support, such as UC Browser for Android, iOS Safari, Blackberry and Opera Mini as future work. We focused on Content-Types of technologies closely related to MathML and did not consider the remaining majority of available Content-Types, since we believe that this is a research paper on its own.

4 Content-Type Handling

As elaborated in section 2 MathML can either be processed by an HTML or XML parser. In order to understand MathML's processing, it is important to first understand the general processing heuristics of HTML and XML in web browsers. Furthermore, we then apply our results to the processing of MathML. We consider the file extension (.html, .xml, .mml), the Content-Type (text/html, application/xml, application/xml+xhtml, text/mathml) and the MathML namespace (http://www.w3.org/1998/Math/MathML)

Our results show that the HTTP Content-Type header always takes precedence over the file extension. Generally speaking, if the Content-Type is set to *text/html* the browser processes the contents in an HTML context. The standard file extension for MathML *.mml* is associated with Content-Type *text/mathml*. It is quite surprising that none of the browsers render this Content-Type but rather offer to download the document with the Download Manager. We will now elaborate on the different browser behaviors when the Content-Type is either unknown or *application/xml(+xhtml)*.
When the Content-Type is unknown (no HTTP Header, file extension, known elements) Firefox displays the content as tree-view, Chrome/Safari output the content of the file as plaintext within *pre* tags, Internet Explorer interprets the content as HTML (not placed in pre tags).
By default, all browsers display a document with Content-Type *application/xml* as a tree-view (XML-context). Also, it is common knowledge that the Content-Type *application/xml+xhtml* is associated with XHTML [We] and therefore facilitates script execution.
When a not well-formed XML document is delivered, Firefox will raise an error and abort the processing, Chrome/Safari will raise an error but output the content of the document until the first error occurs, Internet Explorer outputs the content of the element as text.
We found out that all browsers upgrade an XML document to an HTML context, if an XHTML namespace is added. This facilitates the execution of JavaScript code. In Chrome, Safari and Internet Explorer this is even more problematic as this allows access to properties such as document.cookie. This way an attacker could steal authentication tokens from the victim.

Attack Scenario: Script Execution in XML. For illustration purposes, consider a web-application which accepts *.xml* files for upload. If a user accesses this file, it is delivered with Content-Type *application/xml* and displayed as tree view. Assuming an attacker includes an XHTML namespace, this can lead to XSS and cookie theft.

5 XSLT Processing

We investigated if an XML and/or HTML parser process XSLT. We found that XSLT processing is only possible, when a document is processed by an XML Parser. In our study, this is fulfilled when the Content-Type is set to *application/xml* or *application/xhtml+xml*. XSLT execution is not possible in documents which are delivered as Content-Type *text/html*. Our studies show that a downloaded *.mml* file which is subsequently opened in Firefox, also has XSLT processing capabilities.
Furthermore it is interesting to consider how an XSLT interacts with the Same-Origin Policy. Our tests show that all major browsers allow the reference of an same-origin XSLT stylesheet. None of the browsers, however, allows the inclusion of an XSLT from a foreign origin. Firefox is the only browser to process an inline XSLT styelesheet. In the past [He11b] this could be used to create an SVG Chamaeleon. A similar attack is possible with MathML, as shown in List. 8.

6 Script Execution in MathML

We checked which elements of MathML support scripting capabilities and should therefore be considered potentially dangerous. The complete results are listed in the extended version.

All Browsers. Our investigation shows that the elements *mn*, *mi*, *mo*, *ms* and *mtext* (not in IE) have scripting capabilities. It should be noted that this does not apply if the script element is a child of the parent *math* element. This insight applies to all browsers - even those which do not officially implement MathML. We can conclude from this fact, that also browsers, such as Chrome and Internet Explorer, already implement the processing of MathML as part of HTML5. It should also be taken into account that the parsing context switches to HTML if an HTML element is found outside the previously mentioned MathML elements. Hence, all further MathML elements are no more in the MathML scope. Therefore, this could also be used to trigger script execution.

Our investigation shows that the *href* and *xlink:href* [He11b] attribute is susceptible for script execution - for example by using the well-known *javascript:* pseudo-protocol. Additionally, we adapted a vector from [Ma17] using an *xml:base* and *href* attribute for the use with MathML. The vectors are provided in List. 5.

```
<math href="javascript:alert(1)"> <mi>2</mi></math>
<math><mi xml:base="javascript:alert(1)//" href="#">2</mi></math>
```

List. 5: An Vector Based on xml:base which can be used to test for XSS

Script Execution in *annotation*, *annotation-xml* and *semantics* depends on the value of the *encoding* attribute and on the Content-Type of the document. Our evaluation shows that 1. script execution is largely not possible within an HTML document. There are two exceptions: When the attribute *encoding* has either the value *text/html* or *application/xhtml+xml*. 2. if the Content-Type of the host document is *application/xml* or *application/xhtml+xml* scripts are executed in all of the elements irregardless of the chosen value for the attribute *encoding*. This is quite surprising, as one would expect that JavaScript code is executed within an HTML document.

No combination of a document's Content-Type and the value of the *encoding* attribute can be used to trigger XML Entity Attacks or the processing of a XSLT, which is supplied as the child element of the elements *annotation*, *annotation-xml* and *semantics*.

7 Status Line Manipulation Attack

The *maction* element has the attribute *actiontype*. This attribute can be assigned the values: *toggle*, *tooltip*, *input* and *statusline*. The value *toogle* displays different subexpressions. *tooltip* displays a tooltip when hovering over the expression and *input* facilitiates modification of the expression. The value *statusline* modifies the browser's status line with a stored text. Before discussing any details of the attack, one should consider the implications of being able to modify the status line. The status line is used to show the target of a hyperlink (i.e. value of the attribute *href*). This provides the user with additional information, which action the browser is going to take after clicking the link. Therefore, the correct display of the value is clearly security relevant. Malicious websites may add a JavaScript event handler (onClick), which executes a different action despite displaying the correct destination of the link. This facilitates redirecting users to an arbitrary destination or executing unwanted actions. Our attack differs from existing work in that no script execution is necessary but can be achieved solely with MathML. A proof of concept code for Firefox is provided in List. 6.

```
<html> <body>
<math href="http://attacker.com/target.html">
<maction actiontype="statusline">
<mfrac><mn>1</mn><mn>2</mn></mfrac>
<mtext>http://www.w3.org/TR/MathML3/chapter3.html#presm.mfrac</mtext>
</maction>
</math></body></html>
```

List. 6: A Scriptless Status Line Manipulation Attack with the Element maction

While the browser will display the value of the element *mtext*, implying a reasonable resource as the target of the link, when clicking the link the user is redirected to attacker.com. Execution of arbitrary JavaScript is also possible. by using the *javascript:* pseudo protocol. This issue has been reported to Mozilla [Ch17].

8 Evaluation of Sanitizer and Web-Application Firewalls

To show the feasibilty of our attacks, we tested our vectors against a selected set of Web Application Firewalls and PHP-based sanitizers. In detail we considered Modsecurity CRS, RaptorWAF, HTMLPurifier and HTMLSafe. First of all, we checked if the inclusion of JavaScript inside of MathML (e.g. `<math><mi><script>...</script></mi></math>`) can be used to bypass any of the aforementioned sanitizers. Additionally, we tested the vectors of List. 5 and List. 6. We excluded the testing of XSLT because sanitizing is usually applied in an HTML context and the resulting page will be of Content-Type *text/html*, essentially making the XSLT instructions void. Our tests show that including a *script* element inside a MathML element does not yield any advantage in bypassing Sanitizers compared to supplying the vectors in plain.

HTMLSafe [Go10] is a sanitizer available for PHP. It implements a combination of black- and whitelists. According to our source code analysis and tests, HTMLSafe blacklists the *javascript* protocol and does not whitelist the *xml:base* attribute by default. Therefore both XSS vectors are blocked. However, the Status Line Manipulation attack passes.
HTMLPurifier [ht17] is one of the recommended ways for sanitizing HTML markup with PHP and is a whitelist based sanitizer. Albeit, MathML is currently not implemented in the whitelist. Therefore, by default, even benign MathML is blocked and of course our vectors. To investigate the possible implications of allowing arbitrary MathML elements, we created a prototype which whitelists the elements *math*, *mn*, *mfrac*, *mi* and *maction*. We did not do any further modifications to the source code. Additionally, we whitelisted the attribute *href* for the element *math*. Our tests show that if the attribute *href* is whitelisted as data type CDATA, HTMLPurifier does not sanitize the value and a bypass is possible. In order to do sanitization correctly, the attribute *href* has to be of type URI.
RaptorWAF [Co17] is a Web-Application Firewall. It can be easily bypassed with both vectors by supplying a HTML encoded version of the vectors. Additionally, certain keywords, such as alert and script, must not be used or send with different spelling (i.e. SCRipt). The Status Line Manipulation vector also passes.
Modsecurity with the Core Rule set [Mo17a] is a freely available Web-Application Firewall. It has to be modified[8] in order to allow benign MathML. Both XSS vectors are blocked. However, the Status Line Manipulation vector passes.

We conducted a small sample of tests with our vectors on websites which currently use MathML. We found that both the live demo on mathjax.org and MathJax Sandbox [9] are

[8] Disable rules: 950901, 981173, 900048
[9] http://jbergknoff.github.io/mathjax-sandbox/

susceptible to script execution by using vector 1 from List. 5. While the former is not exploitable, the latter has a DOM-XSS vulnerability. MathMLCentral [10] processes an uploaded file without sanitation. It is vulnerable to reflected XSS, which can exploited with all vectors. Google Blogger[11] supports the usage of MathML markup. While both XSS vectors would only lead to ”Self-XSS” a malicious blogger could use the Status Line Manipulation attack to redirect users to unwanted locations.

9 Related Work

To our knowledge there is no academic publication available dealing with the security of MathML. Heiderich et al. [He11b] have investigated the dangers of SVGs which motivated this work and some attacks could be directly applied. Heiderich has also reported on scriptless attacks by abusing Cascading Style Sheets [He12] Barth et al. [BCS09] have reported on the dangers of MIME Sniffing in browsers. However, they have neither investigated the precedence of file extensions, Content-Types and Namespaces. Various posts on the Internet appeared in the past, discussing MathML-based XSS vectors [Pa17, ja17, Sp17, cu]. DOMPurify [HSS17] - a DOM-based Sanitizer - implements the sanitization of HTML, SVG and MathML. Although these resources contribute to the public awareness of MathML and its potential dangers, they do not provide a systematic and thorough investigation of MathML.

10 Conclusion

Regarding our question if *MathML is dangerous* we can conclude with the following facts to consider: MathML as a standard embedded in HTML5 is implemented by all major browsers. Therefore any security issue found will affect a large user base. Due to the embedding of MathML in a HTML context all elements can be used for script execution. Furthermore XSLT execution might constitute a security issue. The scriptless manipulation of the status line should also be taken into account. MathML is a novel threat, which should be taken seriously. We encourage the community to extend our work and investigate the support of MathML in other software, such as accessibility software and editors.

11 Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments. This research was supported by the German Ministry of research and Education (BMBF) as part of the SyncEnc research project.

[10] http://www.mathmlcentral.com/Tools/FromMathMLFile.jsp
[11] https://www.blogger.com/

References

[Ah16] Ahmady, Abdul: , Implementation of MathML DTD3. http://htmlpurifier.org/phorum/read.php?5,8091, 2016.

[Ak10] Akhawe, Devdatta; Barth, Adam; Lam, Peifung E; Mitchell, John; Song, Dawn: Towards a formal foundation of web security. In: Computer Security Foundations Symposium (CSF), 2010 23rd IEEE. IEEE, pp. 290–304, 2010.

[Au14] Ausbrooks, Ron: , Mathematical Markup Language (MathML) Version 3.0 2nd Edition. https://www.w3.org/TR/2014/REC-MathML3-20140410/, 2014.

[BCS09] Barth, Adam; Caballero, Juan; Song, Dawn: Secure content sniffing for web browsers, or how to stop papers from reviewing themselves. In: Security and Privacy, 2009 30th IEEE Symposium on. IEEE, pp. 360–371, 2009.

[ca17] caniuse: , MathML. http://caniuse.com/mathml, 2017.

[Ch17] Christopher Späth: , MathML maction statusline - status bar text doesn't accurately reflect the target of the link. https://bugzilla.mozilla.org/show_bug.cgi?id=1392258, 2017.

[Co17] CoolerVoid: , Raptor - WAF - Web application firewall using DFA. https://github.com/CoolerVoid/raptor_waf, 2017.

[cu] cure53: , HTML5 Security Cheatsheet. https://html5sec.org/.

[Da11] Dahlström, Erik: , Scalable Vector Graphics (SVG) 1.1 (Second Edition). https://www.w3.org/TR/2011/REC-SVG11-20110816/, 2011.

[DW17] DW: , What does a HTML filter need to do, to protect against SVG attacks?, 2017. https://security.stackexchange.com/questions/26264/what-does-a-html-filter-need-to-do-to-protect-against-svg-attacks/30390.

[Go10] Gocobachi, Miguel: , Package Information: HTML_Safe. https://pear.php.net/package/HTML_Safe, 2010.

[He11a] Heiderich, Mario: , The image that called me, 2011. https://www.owasp.org/images/0/03/Mario_Heiderich_OWASP_Sweden_The_image_that_called_me.pdf.

[He11b] Heiderich, Mario; Frosch, Tilman; Jensen, Meiko; Holz, Thorsten: Crouching tiger-hidden payload: security risks of scalable vectors graphics. In: Proceedings of the 18th ACM conference on Computer and communications security. ACM, pp. 239–250, 2011.

[He12] Heiderich, Mario; Niemietz, Marcus; Schuster, Felix; Holz, Thorsten; Schwenk, Jörg: Scriptless attacks: stealing the pie without touching the sill. In: Proceedings of the 2012 ACM conference on Computer and communications security. ACM, pp. 760–771, 2012.

[HSS17] Heiderich, Mario; Späth, Christopher; Schwenk, Jörg: DOMPurify: Client-Side Protection Against XSS and Markup Injection. In: European Symposium on Research in Computer Security. Springer, pp. 116–134, 2017.

[ht17] htmlpurifier: , htmlpurifier. http://htmlpurifier.org, 2017.

[ja17] jackmasa: , Math. https://twitter.com/jackmasa/status/930096423168655361, 2017.

[Ma17] Masato Kinugawa: , SVG xml base. `https://twitter.com/kinugawamasato/status/898950198826721280`, 2017.

[Mi] Microsoft: , MIME-Handling Changes in Internet Explorer. Accessed: 31.01.2017.

[Mo08] Mozilla: , How Mozilla determines MIME Types. `https://developer.mozilla.org/en-US/docs/Mozilla/How_Mozilla_determines_MIME_Types`, 2008. [Online; accessed 31-January-2018].

[Mo17a] Modsecurity: , ModSecurity. `http://modsecurity.org/`, 2017.

[Mo17b] Mozilla: , MathML. `https://developer.mozilla.org/en-US/docs/Web/MathML`, 2017.

[Pa17] Payloads, XSS: , Filter evasion. `https://twitter.com/XssPayloads/status/935051449670750209`, 2017.

[Sp17] Späth, Christopher: , MathML xml base. `https://secanalysis.wordpress.com/2017/08/28/mathml-xmlbase/`, 2017.

[Wa17] Wang, Frederic: , Review of Igalia's Web Platform activities, 2017. `http://frederic-wang.fr/review-of-igalia-s-web-platform-activities-H1-2017.html`.

[We] Webkit: , Understanding HTML, XML and XHTML. Accessed: 31.01.2017.

[wi15] wikimedia: , Consider opening up POST /media/math/format to external users. `https://phabricator.wikimedia.org/T116147`, 2015.

[Za12] Zalewski, Michal: The tangled Web: A guide to securing modern web applications. No Starch Press, 2012.

A Results

A.1 Semantics, Annotation and Annotation-xml

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
<semantics encoding="text/html">
<iframe xmlns='http://www.w3.org/1999/xhtml' src='javascript:alert(1);'></iframe>
</semantics>
<semantics encoding="application/xml">
<iframe xmlns='http://www.w3.org/1999/xhtml' src='javascript:alert(1);'></iframe>
</semantics>
<semantics encoding="application/xhtml+xml">
<iframe xmlns='http://www.w3.org/1999/xhtml' src='javascript:alert(1);'></iframe>
</semantics>
</math>
```

List. 7: Script Execution Test for element semantics; tests for elements annotation and annotation-xml are constructed analogous

A.2 MathML Chamaeleon

```
<?xml-stylesheet type="text/xml" href="#style1"?>
<math>
<mfrac><mi>2</mi><mi>3</mi></mfrac>
<xsl:stylesheet id="style1" version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
      Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:template match="/">
<html xmlns="http://www.w3.org/1999/xhtml">
<b>testing</b>
</html>
</xsl:template>
<xsl:template match="xsl:stylesheet"></xsl:template>
</xsl:stylesheet>
</math>
```

List. 8: A MathML Chamaeleon

A.3 Dangerous Elements and Attributes

For all tables: A ”0” means "no script execution, while ”1” means ”script execution”.

Tab. 1: Only elements are listed in which at least one element in a browser is susceptible to script execution. Attributes are tested with attribute href;

Element	**Firefox**	**IE 11**	**Chrome**	**Safari**
mi	1	1	1	1
mn	1	1	1	1
mo	1	1	1	1
mtext	1	0	1	1
ms	1	1	1	1
Attributes	1	1	1	1
cn	1	0	0	1
ci	1	0	0	1
csymbol	1	0	0	1
cs	1	0	0	1
apply	1	0	0	1
bind	1	0	0	1
bvar	1	0	0	1
share	0	0	0	0
semantics	1	0	0	1
cerror	1	0	0	1
cbytes	1	0	0	1

Attributes	**FF**	**IE 11**	**CH**	**SA**
mglyph	1	0	0	0
mi	1	0	0	1
mn	1	0	0	1
mo	1	0	0	1
mtext	1	0	0	1
ms	1	0	0	1
mrow	1	0	0	1
mfrac	1	0	0	1
msqrt	1	0	0	1
mroot	1	0	0	1
mstyle	1	0	0	1
merror	1	0	0	1
mpadded	1	0	0	1
mfenced	1	0	0	1
msub	1	0	0	1
msup	1	0	0	1
msubsup	1	0	0	1
munder	1	0	0	1
mover	1	0	0	1
munderover	1	0	0	1
mmultiscripts	1	0	0	1
mprescripts*	0	0	0	1
none*	0	0	0	1
mtable	1	0	0	1
mtr	1	0	0	1
mlabeledtr	1	0	0	1
mtd	1	0	0	1
maligngroup	0	0	0	1
malignmark	0	0	0	1
mstack	1	0	0	1
mlongdiv	1	0	0	1
msgroup	1	0	0	1
msrow	1	0	0	1
mscarries	1	0	0	1
mscarry	1	0	0	1
msline	1	0	0	1
math	1	0	0	1
maction	1	0	0	0
semantics	1	0	0	1
annotation	1	0	0	0
annotation-xml	1	0	0	0