UNDERSTANDING THE ATTACK SURFACE AND ATTACK RESILIENCE OF PROJECT SPARTAN'S (EDGE) NEW EDGEHTML RENDERING ENGINE

Mark Vincent Yason
IBM X-Force Advanced Research
yasonm[at]ph[dot]ibm[dot]com
@MarkYason

ABSTRACT

EdgeHTML is the new rendering engine that will power the next generation web browser (codenamed Spartan) to be introduced in Windows 10. Because EdgeHTML will be widely deployed - from Windows 10 mobile devices to PCs, it is important that we have understanding of its attack surface and its stance against exploitation.

In this presentation, I'll discuss EdgeHTML's attack surface and the different methods for enumerating it. I'll also describe a method of comparing EdgeHTML and MSHTML to identify and understand what had changed from the forking process, and more importantly, identify new features and added internal functionalities that can contribute to its attack surface. Finally, I'll discuss the exploit mitigations in place, how they help against certain classes of vulnerabilities, and discuss known bypass techniques that are still applicable.

CONTENTS

1. INTRODUCTION

EdgeHTML [1, 2] is the new rendering engine that will be introduced in Microsoft's new Edge browser (previously codenamed Project Spartan) to support modern web standards and at the same time to remove legacy code. This new rendering engine is a fork of the Trident (MSHTML) rendering engine which is currently being used in Internet Explorer.

It was estimated [3] that around 220k+ lines of code were removed from the forking process while around 300k+ lines of code were added for interoperability fixes and new features. From a security research standpoint, it is both interesting and important to understand what are the side-effects of these changes in terms of how the attack surface of the rendering engine have changed – what are the new attack vectors? are some of the attack vectors previously used by attackers now removed? Another important matter to understand is the resiliency of this new rendering engine against attacks – are there changes in the exploit mitigations in place? how difficult is it for attackers to exploit this new rendering engine compared to its predecessor in a default configuration? Answering these essential questions is the purpose of this paper.

This paper is divided into three sections. The first section (Overview) gives an overview of the attack surface and exploit mitigations which are discussed in-depth in the later sections, it contains the "EdgeHTML Attack Surface Map and Exploit Mitigations" diagram which can serve as a visual reference for the rest of the paper. The first section also discusses the diffing method I used in the early part of my research for diffing EdgeHTML against MSHTML. The second section (Attack Surface) is where I discuss in-depth the different attack vectors that comprises EdgeHTML's attack surface and point out important attack vector changes from MSHTML to EdgeHTML. Finally, the last section (Exploit Mitigations) is where I discuss the different exploit mitigations in place that overall contributes to making the exploitation of EdgeHTML vulnerabilities difficult or costly.

All information in this paper is based on Microsoft Edge running on 64-bit Windows 10 build 10240 (edgehtml.dll version 11.0.10240.16384).

2. OVERVIEW

Before diving deep into the details of EdgeHTML's attack surface and applied exploit mitigations, this section will first give a brief overview. This section will also discuss the method I used for diffing EdgeHTML against MSHTML to help identify major changes in the code.

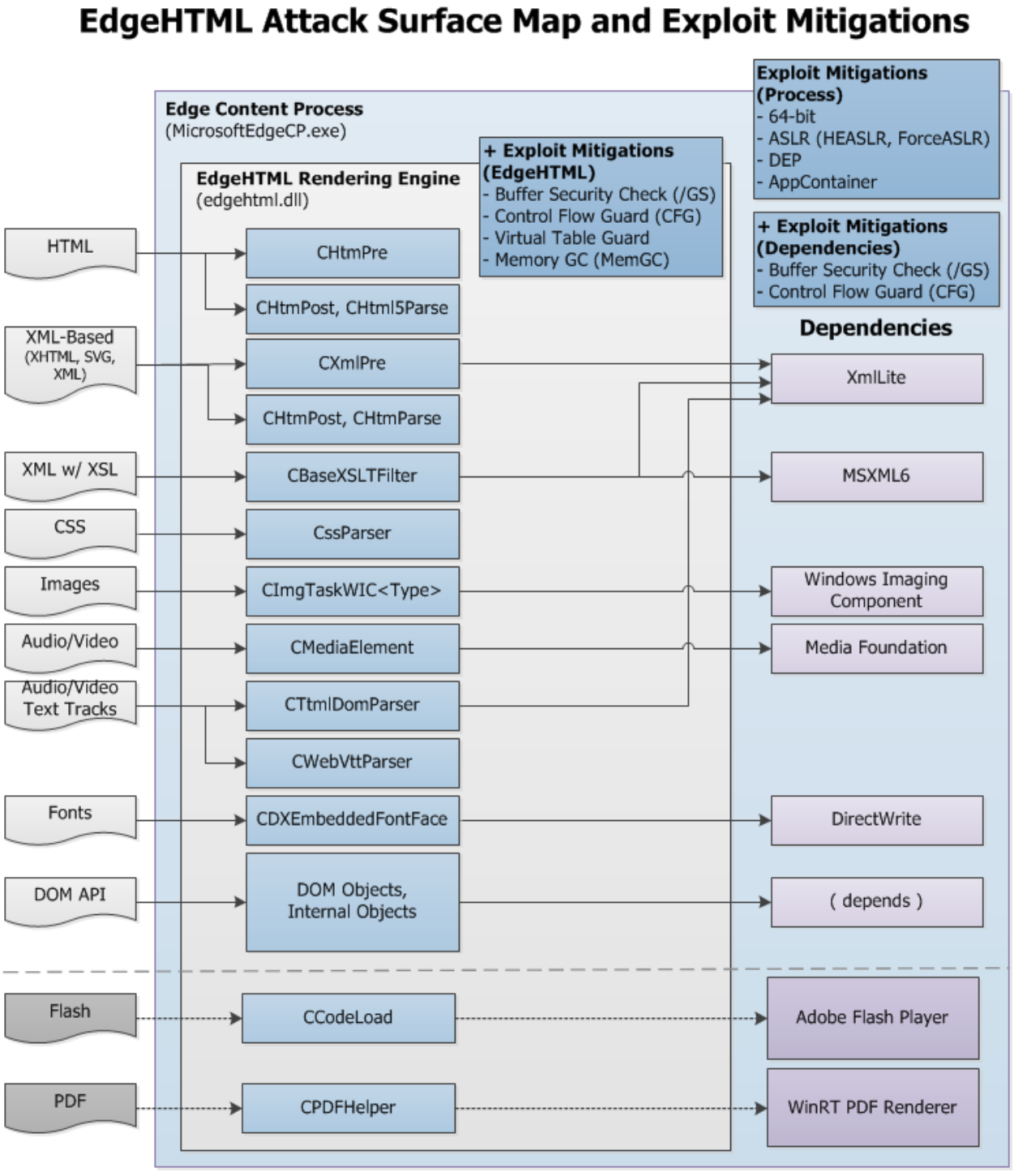
2.1. ATTACK SURFACE MAP AND EXPLOIT MITIGATIONS

The EdgeHTML rendering engine DLL (%System32%\edgehtml.dll) is responsible for parsing and rendering potentially malicious web content. It is hosted in the Edge browser content process (MicrosoftEdgeCP.exe) which is a 64-bit AppContainer-sandboxed process.

The attack surface map in the next page shows the different input types that EdgeHTML accepts and the EdgeHTML classes that process them. Note these EdgeHTML classes are the initial entry point for the parsing/processing; some of those listed may require the use of additional EdgeHTML classes. The library that a particular EdgeHTML class uses to process the input is also listed.

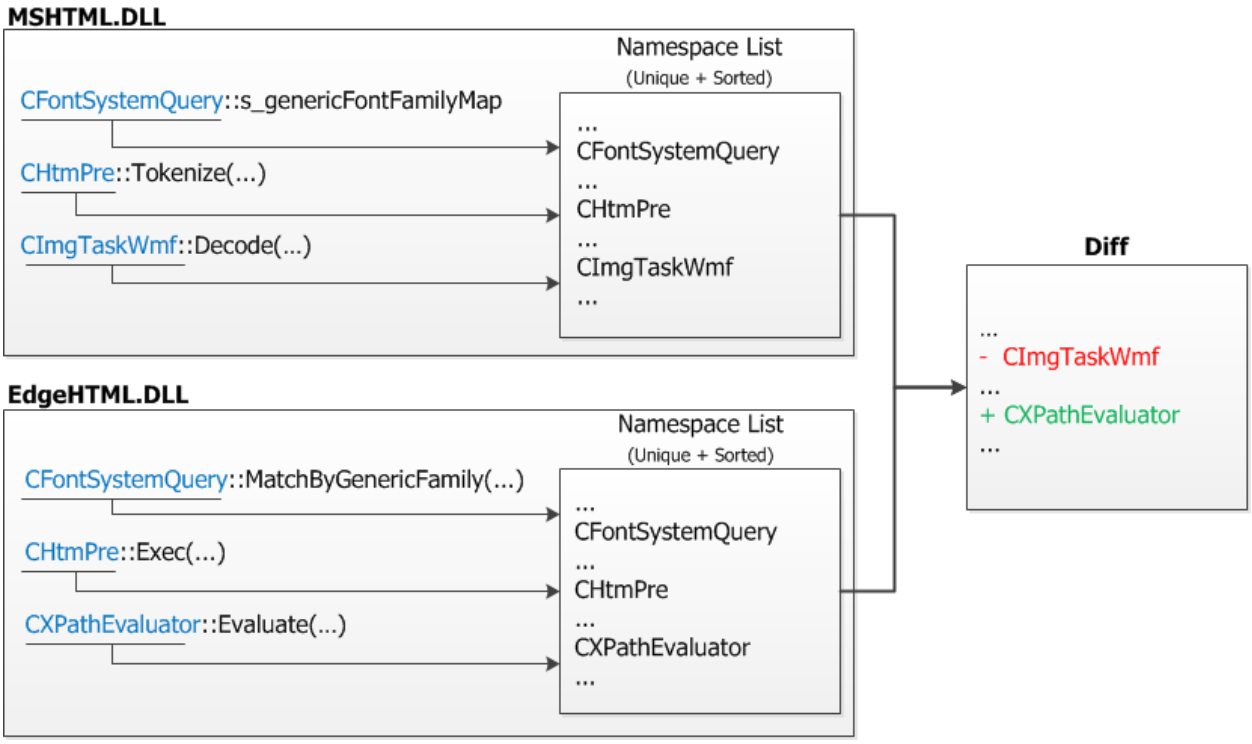
For processing HTML, EdgeHTML use its internal parser for parsing the markup. While for XML-based markups (XHTML, SVG, XML), it additionally uses XmlLite for the XML parsing. If the markup is an XML that references a XSL stylesheet, MSXML6 first performs the XML transformation and the output is then fed into the markup processing. CSS parsing is performed using an internal parser. For decoding images, EdgeHTML mainly relies on Windows Imaging Component (WIC). For decoding audio/video content, EdgeHTML mainly relies on Media Foundation (MF). Timed text tracks files which are used for audio/video captioning are parsed by EdgeHTML using internal parsers with the help of XmlLite. Font rendering is handled by DirectWrite. The DOM objects which are created when markup tags are parsed or are dynamically created are exposed via the DOM API, some of these DOM objects may in turn rely on libraries for their functionality. Also, EdgeHTML by default, can instantiate additional built-in/pre-installed renderers - the WinRT PDF renderer to handle PDF content, and Adobe Flash Player to handle Flash content.

Several exploit mitigations are applied to the 64-bit Edge content process where the EdgeHTML rendering engine is hosted. Namely, ASLR (with high entropy and force ASLR enabled), DEP, and the AppContainer processes isolation mechanism. EdgeHTML and its dependencies are compiled with stack buffer security check (/GS) and the recently introduced Control Flow Guard (CFG). Finally, additional exploit mitigations such as Virtual Table Guard (VTGuard) and the new Memory GC (MemGC) are specifically applied to EdgeHTML.


EdgeHTML Attack Surface Map and Exploit Mitigations
Exploit Mitigations (Process)
- 64-bit
- ASLR (HEASLR, ForceASLR)
- DEP
- AppContainer
Edge Content Process
(MicrosoftEdgeCP.exe)
+ Exploit Mitigations (EdgeHTML)
- Buffer Security Check (/GS)
- Control Flow Guard (CFG)
- Virtual Table Guard
- Memory GC (MemGC)
EdgeHTML Rendering Engine
(edgehtml.dll)
+ Exploit Mitigations (Dependencies)
- Buffer Security Check (/GS)
- Control Flow Guard (CFG)
Dependencies
HTML
CHtmPre
CHtmPost, CHtml5Parse
XML-Based (XHTML, SVG, XML)
CXmlPre
CHtmPost, CHtmParse
XmlLite
XML w/ XSL
CBaseXSLTFilter
MSXML6
CSS
CssParser
Images
CImgTaskWIC<Type>
Windows Imaging Component
Audio/Video
CMediaElement
Media Foundation
Audio/Video Text Tracks
CTtmlDomParser
CWebVttParser
Fonts
CDXEmbeddedFontFace
DirectWrite
DOM API
DOM Objects, Internal Objects
(depends)
Flash
CCodeLoad
Adobe Flash Player
PDF
CPDFHelper
WinRT PDF Renderer

2.2. INITIAL RECON: DIFFING EDGEHTML AGAINST MSHTML

When I started my research on EdgeHTML, I wanted to find out what are the major code changes from the forking process of MSHTML to EdgeHTML. Since classes and namespaces suggest group of related code which may in turn correspond to a program feature, one method I tried was, using IDAPython, enumerate function/variable names and then extract the namespace part of the function/variable names. Duplicates are removed and the resulting namespace list is sorted. Then finally, the namespace list of MSHTML and EdgeHTML are diffed using the diff(1) utility:



One example diff output which suggests a change in the support for WMF and EMF images (discussed in 3.3) are the following missing classes in EdgeHTML:

```
-CImgTaskEmf
-CImgTaskWmf
```

Another example diff output are the new namespaces under the CFastDOM namespace which suggest the new DOM object types that are exposed via the DOM API (discussed in 3.5):

```
+CFastDOM::{…more…}
+CFastDOM::CXPathEvaluator
+CFastDOM::CXPathExpression
+CFastDOM::CXPathNSResolver
+CFastDOM::CXPathResult
+CFastDOM::CXSLTProcessor
```

Finally, another interesting diff output suggests that some code had been ported from the Blink rendering engine [4, 5] (before most of the namespace were renamed from "WebCore" to "blink"):

```
+blink::WebThread
+WebCore::AnalyserNode
+WebCore::AudioArray<float>
+WebCore::AudioBasicInspectorNode
+WebCore::Audio{…more…}
```

A further look at EdgeHTML's code confirmed that these ported code were for EdgeHTML's Web Audio [6] support.

In addition to diffing namespaces, this rudimentary diffing method can be applied to function names, class method names (to identify new/removed class functionalities), strings (which can give telltale signs of what the new functionalities are – i.e.: log strings), imports (to identify new libraries and the specific APIs used by the binary) and exports.

Note that a caveat of this method is that the renaming of a namespace will result into an added namespace plus a deleted namespace; therefore, the results need to be further verified. Additionally, this method requires that the symbols for the binaries are available. Binary diffing [7] is another option for identifying differences between two binaries down to the changes at the function level.

3. ATTACK SURFACE

In this section, I'll enumerate the different types of untrusted input processed by EdgeHTML and the code in EdgeHTML that processes them. Note that the listed EdgeHTML classes are the initial entry point for the parsing/processing; some of those listed may use additional EdgeHTML classes for the processing. The purpose of identifying these EdgeHTML classes is that they can be used for setting breakpoints in the code when understanding how EdgeHTML handles a particular input type. For example, when understanding how the XML-based markup pre-parsing works, breakpoints in the CXmlPre methods can be set via:

```
(WinDbg)> bm edgehtml!CXmlPre::*
```

If EdgeHTML relies on a library for processing a particular input type, the library and the specific interface used by EdgeHTML is also listed.

3.1. MARKUP/STYLE PARSING

One of the primary tasks performed by a rendering engine is processing markups and styles. For HTML and CSS, EdgeHTML relies on its internal classes for the parsing, while for parsing XML-based markups, EdgeHTML additionally uses XmlLite [8] and MSXML6 [9]:

Markup/Style	EdgeHTML Class	Library (and Interface) Used
HTML	CHtmPre (Pre-parsing) CHtmPost, CHtml5Parse (Post-parsing)	
XML-Based (XHTML, SVG, XML)	CXmlPre (Pre-parsing) CHtmPost, CHtmParse (Post-parsing)	XmlLite (IXmlReader)
CSS	CssParser	
XML w/ XSL	CBaseXSLTFilter	XmlLite (IXmlReader) MSXML6 (IXMLDOMDocument)
VML	(Removed in EdgeHTML: No Binary Behaviors)	

EdgeHTML processes markups in two phases, the pre-parsing phase which starts at CHtmPre::Exec() or CXmlPre::Exec() involves the initial parsing of the markup, writing the parsed tags to a tag stream, and starting download of referenced resources (if applicable - such as images and CSS files). The post-parsing phase which starts at CHtmPost::Exec() fetches the tags from the tags stream, performs additional parsing of the tags if necessary, and eventually results to the creation of the DOM objects.

XmlLite's IXmlReader interface which is instantiated by EdgeHTML via a call to xmllite!CreateXmlReader() is used by CXmlPre as a parser when pre-parsing XML-based markups, it is also used by CBaseXSLTFilter as an XML parser when detecting if an XML file references an XSL stylesheet. Additionally, MSXML6's IXMLDOMDocument interface is used by CBaseXSLTFilter for transforming an XML file that references an XSL stylesheet.

An important change in EdgeHTML is that support for binary behaviors [10], including the built-in VML binary behavior was removed. VML-based (VGX.DLL) vulnerabilities [11] are deemed important because, though obsolete, VML is still available by default even on IE11/MSHTML.

The markup/style processor of rendering engines, especially the CSS and HTML parsers are expected to be updated overtime as there will be new web standards that will require new HTML tags, HTML attributes, CSS properties,

etc. to be parsed and handled. An example of well-known zero-day vulnerability that relates to markup/style processing is the MSHTML CSS recursive import vulnerability [12].

3.2. IMAGE DECODING

Image rendering is another basic task performed by a rendering engine. Arbitrary image files can be passed to the rendering engine via direct link to the image file or via various HTML tags such as <img> and <embed> tags.

Support for image formats can be enumerated by looking at the g_rgMimeInfoImg array which contains MIMEINFO items that specifies the image mime type and the function that will instantiate the EdgeHTML image handler class.

Below are the image formats supported by EdgeHTML. In the library field, you'll notice that Windows Imaging Component (WIC) [13] is used for all image processing (SVG is processed via markup processing described in 3.1):

Image Format	EdgeHTML Class	Library (and Interface) Used
PNG	CImgTaskWICPng	WIC (IWICImagingFactory)
JPG	CImgTaskWICJpgTemplate	WIC (IWICImagingFactory)
GIF	CImgTaskWICGif	WIC (IWICImagingFactory)
DDS	CImgTaskWICDds	WIC (IWICImagingFactory)
TIFF	CImgTaskWICTiff	WIC (IWICImagingFactory)
BMP	CImgTaskWICBmp	WIC (IWICImagingFactory)
HDP	CImgTaskWICHdp	WIC (IWICImagingFactory)
ICO	CImgTaskWICIco	WIC (IWICImagingFactory)
WMF	(Removed in EdgeHTML: Processing Removed)	
EMF	(Removed in EdgeHTML: Processing Removed)	

For processing images, EdgeHTML first instantiates WIC's IWICImagingFactory interface via CWicGlobals::GetWicImagingFactory(). IWICImagingFactory::CreateDecoder() is then called to instantiate an IWICBitmapDecoder interface for a particular image format.

An interesting change in EdgeHTML is that support for processing of WMF and EMF images was removed; this means that previous dependence to the GDI library for parsing remotely-supplied WMF and EMF files was removed. The code in GDI that parses WMF and EMF files is historically known for remotely-exploitable vulnerabilities [14, 15, 16].

3.3. AUDIO/VIDEO DECODING

Through a direct link or via the HTML <audio> and <video> tags, the rendering engine can be passed arbitrary audio/video content. Mime types for supported audio/video container formats are reflected by the g_rgMimeInfoAudio and g_rgMimeInfoVideo array. As listed on the table below, the library that EdgeHTML uses to process audio/video content is Media Foundation (MF) [17]:

Media Container Format	EdgeHTML Class	Library (and Interface) Used
MP4	CMediaElement	MF (IMFMediaEngine)
MP3	CMediaElement	MF (IMFMediaEngine)
WAV	CMediaElement	MF (IMFMediaEngine)

Via MFCreateMediaEngine(), EdgeHTML instantiates MF's IMFMediaEngine interface which is used for setting up the media source and controlling playback.

In addition to audio/video file processing, EdgeHTML also processes timed text tracks [18] which can be specified via the HTML <track> tag. EdgeHTML supports two text track file formats:

Text Track Format	EdgeHTML Class	Library (and Interface) Used
TTML	CTtmlDomParser	XmlLite (IXmlReader)
WebVTT	CWebVttParser	

TTML is XML-based and the EdgeHTML class that processes it use XmlLite's IXmlReader for the XML parsing.

3.4. FONT RENDERING

Arbitrary fonts can be passed to the rendering engine via the @font-face CSS rule [19]. Vulnerabilities in font parsing [20], especially if reachable via a browser rendering engine, can be a cause of remote compromise – an example is CVE-2011-3402 [21, 22], a font parsing vulnerability in GDI (in Win32k.sys) that was initially used in a zero-day attack and later been used by browser exploit kits.

EdgeHTML uses DirectWrite [23] for rendering fonts and supports the following font formats:

Font Format	EdgeHTML Class	Library (and Interface) Used
TTF	CDXEmbeddedFontFace	DirectWrite (IDWriteFactory1, IDWriteFactory2)
OTF	CDXEmbeddedFontFace	DirectWrite (IDWriteFactory1, IDWriteFactory2)
WOFF	CDXEmbeddedFontFace	DirectWrite (IDWriteFactory1, IDWriteFactory2) (after extraction of TTF/OTF via CDXEmbeddedFontFace::UnpackFontFromWOFFData())
EOT	(Removed from EdgeHTML: Processing Removed)	

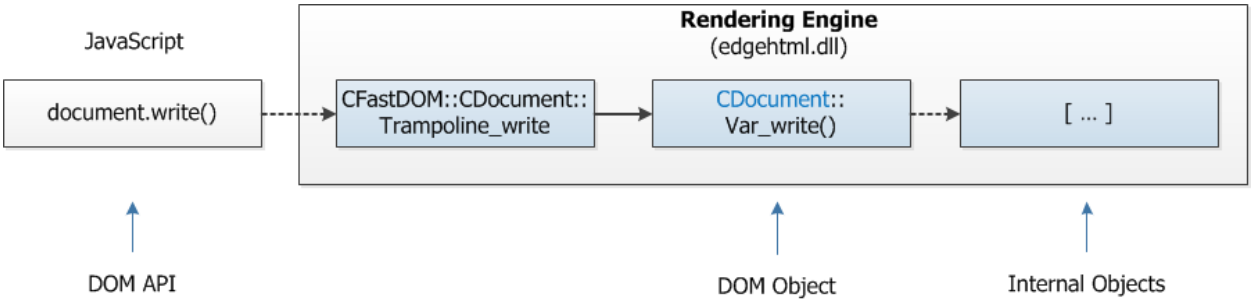
Instantiation of the required DirectWrite interfaces is performed in CDXResourceDomain::EnsureDXFactories(). EdgeHTML detects the actual format of the font via CDXEmbeddedFontFace::Initialize() and starts performing the custom private font registration in CDXPrivateFont::Initialize() via a call to IDWriteFactory:: CreateCustomFontFileReference().

Unlike GDI, DirectWrite (DWrite.dll) parses the font file in the user-mode process where it is hosted. Also, a notable change in EdgeHTML is that EOT font support was removed, this means that dependence to T2EMBED [24] and GDI for parsing EOT fonts was removed, therefore reducing the number of libraries used in parsing remotely-supplied fonts.

3.5. DOM API

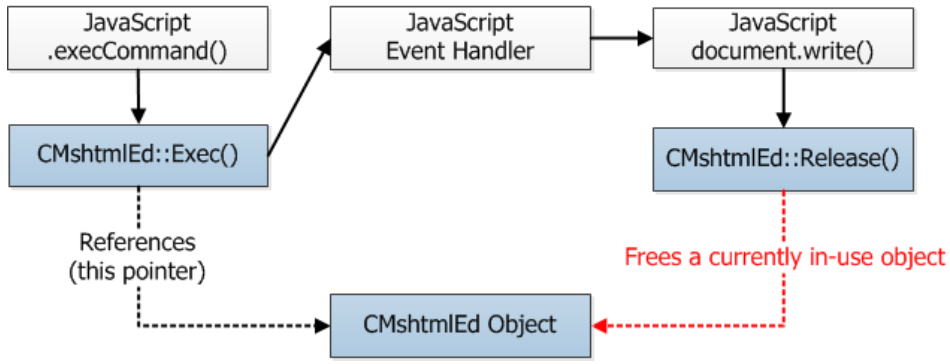
The Document Object Model (DOM) API [25] is one of the largest attack surfaces in a rendering engine. When parsing an HTML document, the rendering engine instantiates DOM objects that represents the parsed HTML tags. The rendering engine will also create core DOM objects such as the DOM 'document' object. And finally, JavaScript code can also dynamically instantiate new DOM objects. The DOM API provides a way to manipulate these DOM objects.

When DOM object properties are set/retrieved or when DOM object methods are invoked via a script, corresponding code in the rendering engine is executed:



Because code in the rendering engine that are executed via the DOM API can change the state of the DOM tree, DOM objects and other internal rendering engine objects, unexpected input, unexpected state change or an incorrect internal state when a DOM API is called can result into memory corruption vulnerabilities such as use-after-frees [26] (example below), heap overflows [27], invalid pointer access [28], etc.

CVE-2012-4969 (IE CMshtmlEd UAF)



Using the diffing method described in section 2.2, we can identify the new DOM object types in EdgeHTML by looking at the changes under CFastDOM namespace:

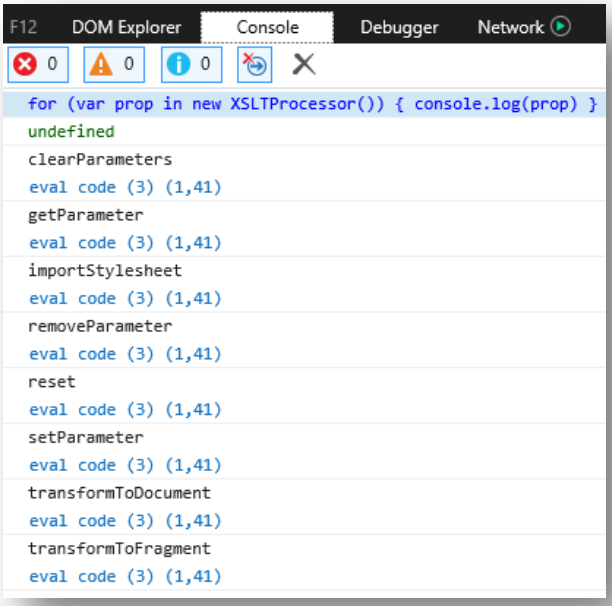
```
+CFastDOM::CAnalyserNode
+CFastDOM::CAriaRequestEvent
+CFastDOM::CAudioBuffer
+CFastDOM::CAudioBufferSourceNode
+CFastDOM::CAudioContext
+CFastDOM::CAudioDestinationNode
+CFastDOM::CAudioListener
+CFastDOM::CAudioNode
+CFastDOM::CAudioParam
+CFastDOM::CAudioProcessingEvent
+CFastDOM::CBiquadFilterNode
+CFastDOM::CClipboardEvent
+CFastDOM::CCommandEvent
+CFastDOM::CConvolverNode
+CFastDOM::CCryptoKey
+CFastDOM::CCryptoKeyPair
+CFastDOM::CCSS
```

```
+CFastDOM::CCSSConditionRule
+CFastDOM::CCSSGroupingRule
+CFastDOM::CDataCue
+CFastDOM::CDataTransferItem
+CFastDOM::CDataTransferItemList
+CFastDOM::CDeferredPermissionRequest
+CFastDOM::CDelayNode
+CFastDOM::CDynamicsCompressorNode
+CFastDOM::CEventTarget
+CFastDOM::CGainNode
+CFastDOM::CGamepad
+CFastDOM::CGamepadButton
+CFastDOM::CGamepadEvent
+CFastDOM::CHashChangeEvent
+CFastDOM::CIsolatedGlobalScope
+CFastDOM::CMediaDeviceInfo
+CFastDOM::CMediaDevices
+CFastDOM::CMediaStream
+CFastDOM::CMediaStreamError
+CFastDOM::CMediaStreamErrorEvent
+CFastDOM::CMediaStreamTrack
+CFastDOM::CMediaStreamTrackEvent
+CFastDOM::CMSAppAsyncOperation
+CFastDOM::CMSHeaderFooter
+CFastDOM::CMSPrintManagerTemplatePrinter
+CFastDOM::CMSTemplatePrinter
+CFastDOM::CMSWebViewSettings
+CFastDOM::CNavigationEventWithReferrer
+CFastDOM::COfflineAudioCompletionEvent
+CFastDOM::COfflineAudioContext
+CFastDOM::COscillatorNode
+CFastDOM::COverflowEvent
+CFastDOM::CPannerNode
+CFastDOM::CPermissionRequest
+CFastDOM::CPermissionRequestedEvent
+CFastDOM::CRTCDtlsTransport
+CFastDOM::CRTCDtlsTransportStateChangedEvent
+CFastDOM::CRTCDtmfSender
+CFastDOM::CRTCDTMFToneChangeEvent
+CFastDOM::CRTCIceCandidatePairChangedEvent
+CFastDOM::CRTCIceGatherer
+CFastDOM::CRTCIceGathererEvent
+CFastDOM::CRTCIceTransport
+CFastDOM::CRTCIceTransportStateChangedEvent
+CFastDOM::CRTCRtpListener
+CFastDOM::CRTCRtpReceiver
+CFastDOM::CRTCRtpSender
+CFastDOM::CRTCRtpUnhandledEvent
+CFastDOM::CRTCSrtpSdesTransport
+CFastDOM::CRTCSsrcConflictEvent
+CFastDOM::CScriptProcessorNode
+CFastDOM::CServiceUIFrameContext
+CFastDOM::CStereoPannerNode
+CFastDOM::CSVGForeignObjectElement
+CFastDOM::CVideoTrack
```

```
+CFastDOM::CVideoTrackList
+CFastDOM::CWaveShaperNode
+CFastDOM::CXMLHttpRequestUpload
+CFastDOM::CXPathEvaluator
+CFastDOM::CXPathExpression
+CFastDOM::CXPathNSResolver
+CFastDOM::CXPathResult
+CFastDOM::CXSLTProcessor
```

These new DOM object types represent new code or new code paths in EdgeHTML that are exposed and reachable via the DOM API.

In terms of enumerating DOM object properties and methods, JavaScript's for…in statement can be used. The example below uses the new XSLTProcessor DOM object type:

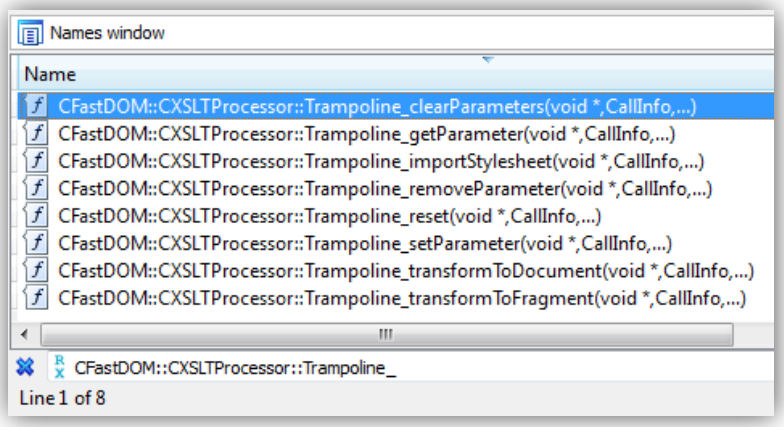


By diffing the results of the object property enumeration, one can find out the property changes in already-existing DOM object types. Below is a property diff output snippet for the DOM "document" object:

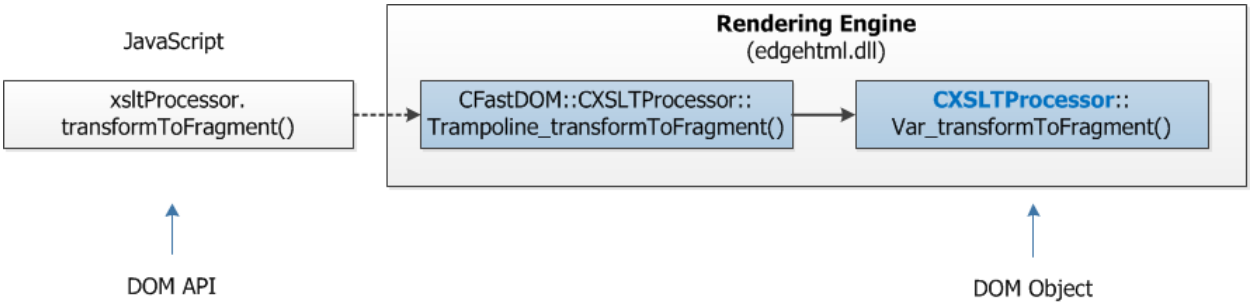
```
 […]
+document.evaluate
 document.execCommand
 document.execCommandShowHelp
+document.exitFullscreen
 document.fgColor
-document.fileCreatedDate
 […]
```

Same with new DOM object types, new DOM object properties/methods represent new code or code paths in EdgeHTML that are exposed and reachable via the DOM API.

Lastly, another way to identify a DOM object's properties and methods is by performing a query in IDA's Names window:



By following one of the listed functions in the Names window, one can find out the actual EdgeHTML class that represents the DOM object type:



As new features are added to the Edge browser [29], new DOM objects types will likely be introduced or new properties/methods will be added to already-existing DOM objects in order to expose the new features to developers. These new DOM object types, properties, and methods will in turn represent new attacker-reachable code which increases the rendering engine's attack surface.

3.6. FLASH AND PDF RENDERERS

Finally, though technically not part of the rendering engine and are performing another set of complex parsing and rendering themselves; the built-in PDF renderer [30] in Windows (since Windows 8.1) and the pre-installed Adobe Flash player (since Windows 8) can be considered as just one of the many dependencies that EdgeHTML uses to render their respective file formats since they are pre-installed [3] and can be instantiated by the EdgeHTML rendering engine by default:

Content Type	EdgeHTML Loader/Helper Class	Built-in/Pre-installed Renderer
PDF	CPDFHelper	Built-in WinRT PDF Renderer in Windows (%System32%\Windows.Data.Pdf.dll)
Flash	CCodeLoad	Pre-installed Adobe Flash Player (%System32%\Macromed\Flash\Flash.ocx)

The PDF renderer is instantiated in CPDFHelper::LoadPdfDoc(), while the Flash renderer instantiation starts at CCodeLoad::BindToObject().

From an attacker's standpoint, being able to use additional complex renderers have certain advantages such as: (1) these complex renderers have another set of attack surface and vulnerabilities that attackers can leverage, and (2) some of their functionalities can be repurposed to bypass exploit mitigations – an example is when Flash JIT-generated code was leveraged to bypass Control Flow Guard [31], and another example is when a zero-day exploit used the well-known Flash Vector object corruption technique [32] to bypass ASLR by leveraging an IE rendering engine vulnerability to corrupt a Flash Vector object in memory [33]. Though the CFG bypass via the Flash JIT and the ASLR bypass via the Flash Vector object have been mitigated (see 4.3 and [34]), they showed how software functionalities can be leveraged for exploitation.

3.7. ANALYSIS AND SUMMARY: ATTACK SURFACE

In the area of image and font rendering, EdgeHTML's attack surface had been reduced because of the removal of support for EMF images, WMF images, and EOT fonts. The code in the libraries (GDI and T2EMBED) that processed these file formats had a history of remotely exploitable vulnerabilities. Removal of support for VML (via binary behaviors) also further contributes to the attack surface reduction in EdgeHTML.

However, as with any other modern browser, new features are added and these features are exposed via new DOM object types/properties/methods and updated markup/style specifications. In the case of EdgeHTML, new attack vectors were found in the DOM API in the form of new DOM objects types and the addition of new properties and methods in already-existing DOM object types.

Also, the following libraries are identified as being used by EdgeHTML:

- XmlLite for XML parsing
- MSXML6 for XML transformation
- Windows Imaging Component (WIC) for image decoding
- Media Foundation (MF) for audio/video decoding
- DirectWrite for font rendering
- Built-in WinRT PDF Renderer for PDF rendering
- Pre-installed Adobe Flash Player for Flash rendering

By identifying how these libraries are used by EdgeHTML, we can further recognize their importance since we now have an additional understanding on how they are used by the rendering engine and how attackers might be able to remotely reach code in these libraries via malicious inputs.

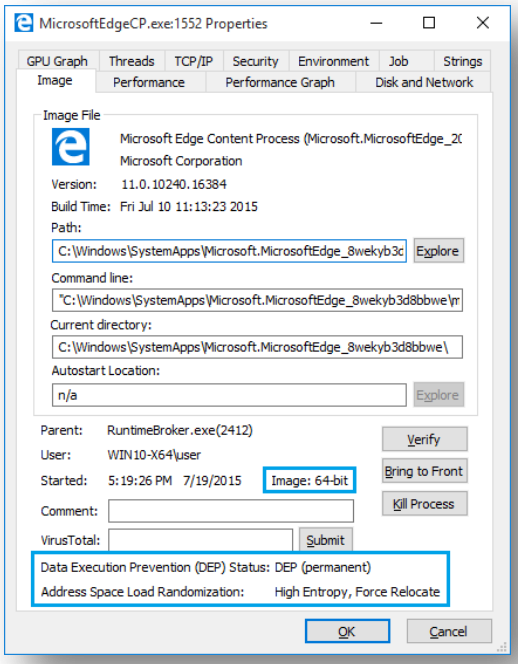
4. EXPLOIT MITIGATIONS

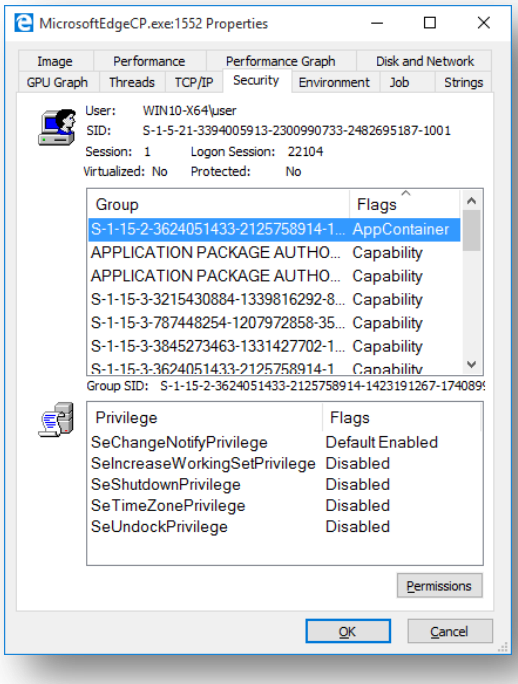
Now that we have an understanding of EdgeHTML's attack surface, let's now take a look at the exploit mitigations that an attacker will need to bypass in order to successfully exploit a vulnerability in EdgeHTML or any of its dependencies. Known/published mitigation bypass or weakness discovered/researched by various security researchers are also discussed and referenced.

Mitigations applied to the Edge content process where EdgeHTML is hosted are briefly discussed, while a more detailed discussion on the specific exploit mitigations to EdgeHTML and its dependencies will be made. Mitigations applied to the Windows heap are thoroughly discussed in various papers/presentations [35, 36, 37, 38, 39, 40] and will not be discussed here.

4.1. 64-BIT, ASLR, DEP AND APPCONTAINER

In Windows 10 x64, the Edge content process (MicrosoftEdgeCP.exe) that hosts the EdgeHTML rendering engine (%System32%\edgehtml.dll) is by default running 64-bit, with ASLR (HEASLR, ForceASLR) and DEP enabled, and is sandboxed using AppContainer:





The content process mitigations in Edge are the same as the content process mitigations in Immersive IE on Windows 8 and different from the content process mitigations in IE11 on Windows 10, Windows 8 (Desktop IE) and Windows 7.

The table below shows the default mitigations applied to the content process of Edge and IE on different Windows versions:

	Win10/Edge	Win10/IE11	Win8/Immersive IE	Win8/IE11	Win7/IE11
64-bit	**Yes**	**No**	**Yes**	**No**	**No**
ASLR	**Yes** (HEASLR, ForceASLR)	**Yes** (ForceASLR)	**Yes** (HEASLR, ForceASLR)	**Yes** (ForceASLR)	**Yes** (ForceASLR)
DEP	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Process Isolation	**AppContainer**	**Low Integrity**	**AppContainer**	**Low Integrity**	**Low Integrity**

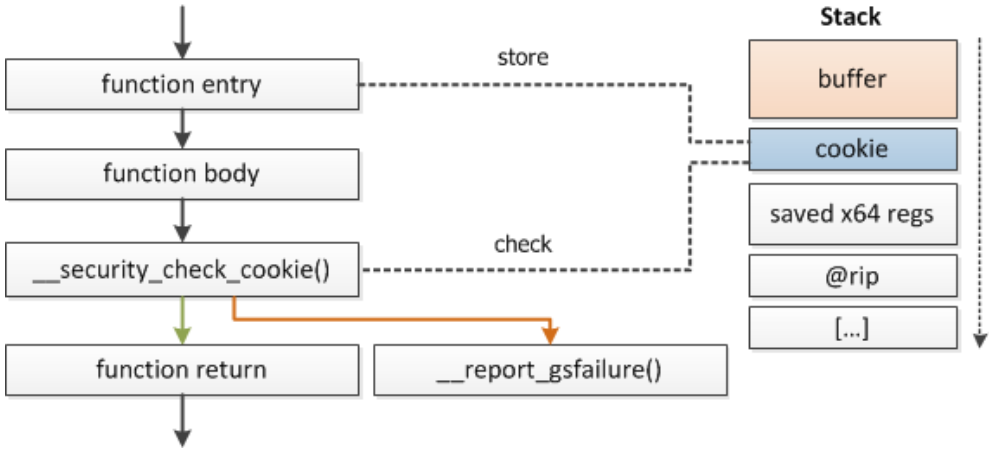
64-bit mitigates traditional heap spraying which involves spraying the heap with attacker-controlled values and the attacker will be fairly successful in landing the controlled data at a particular address. However, depending on the vulnerability, a "relative heap spray" [28, 41] may be possible in cases where a vulnerability involves a valid heap pointer being added an attacker-controlled or erroneous value in a pointer computation.

The content process also has High Entropy ASLR (HEASLR) and Force ASLR (ForceASLR) [42, 43] enabled. HEASLR gives additional entropy to where memory regions can be relocated, while ForceASLR prevents loading of DLLs that do not support ASLR from being loaded at a static address. The bypass for ASLR (and consequently, DEP) in a process in which ForceASLR is enabled is to take advantage of pointers in predictable memory regions or using a vulnerability to disclose memory contents. Since Microsoft is actively resolving the former [44, 42], more and more attacks will rely on using a vulnerability to disclose memory contents [45, 32].

AppContainer is a process isolation mechanism first introduced in Windows 8 and was used in IE's Enhanced Protected Mode [46] sandbox. Among other things, AppContainer limits a process's read/write access and capabilities. There are several ways to bypass the AppContainer sandbox (and other sandboxes), these includes exploiting kernel vulnerabilities [47, 48], exploiting vulnerabilities in the broker or higher-privileged processes [46, 49, 50], and leveraging writable resources [49].

4.2. STACK BUFFER SECURITY CHECK (/GS)

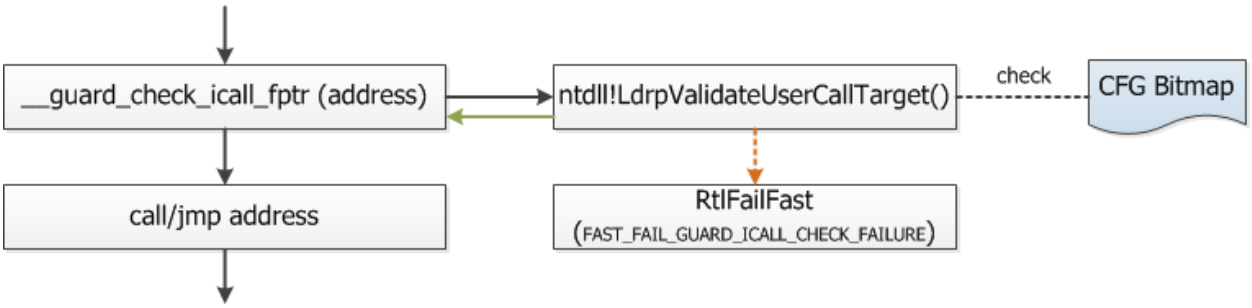
To detect stack-based buffer overflows that attempt to control the program's execution flow, EdgeHTML and its dependencies were compiled with the Buffer Security Check (/GS) [51] compiler option. This mitigation involves storing a security cookie in the stack just after local buffers and then checking the security cookie before the function returns to make sure that the return address and saved x64 registers were not overwritten via a buffer overflow. This mitigation also performs variable reordering so that a copy of the parameters and local variables are located before the buffers to prevent them from being corrupted in the event of a buffer overflow:



This mitigation had been thoroughly discussed in various papers [52, 53] and is continually being updated [42] for improved coverage. One limitation of this mitigation is that it does not cover cases where an attacker has control of where in the stack buffer to write data to (such as a controllable stack buffer index/pointer) [54, 20], allowing the attacker to write beyond the stack cookie.

4.3. CONTROL FLOW GUARD (CFG)

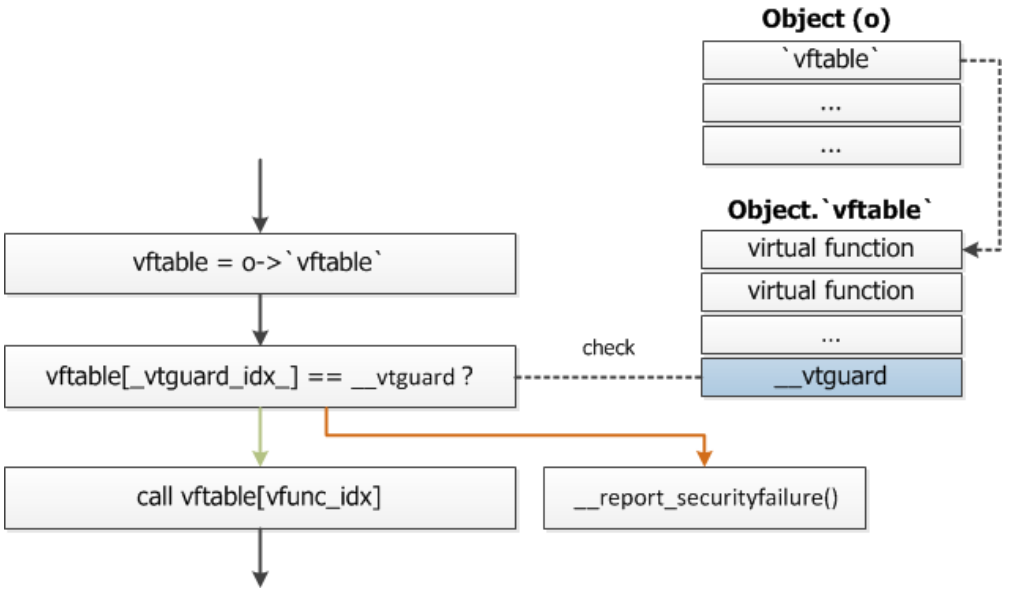
A recently introduced exploit mitigation that is applied to EdgeHTML and its dependencies is Control Flow Guard (CFG) [55, 56]. When CFG is enabled, the compiler will add checks in the code to ensure that the destination of indirect calls is valid. This exploit mitigation attempts to detect and prevent abnormal control flow which can occur if an exploit is trying to redirect execution to ROP gadgets in select executable code addresses.



Internals of this exploit mitigation are well-researched and published in a various papers/presentations [57, 58]. One published bypass technique [31] against CFG is by taking advantage of Flash JIT-generated code, which due to its dynamic nature, any indirect calls made inside it will not be covered by CFG. This bypass technique, however, is now mitigated in Flash by additionally JIT-generating code that calls ntdll!LdrpValidateUserCallTarget() whenever call instructions are generated. Other published ideas to bypass CFG include jumping to valid API addresses (i.e. LoadLibrary) [41], overwriting stack data (such as a return address) [57, 41], etc.

4.4. VIRTUAL TABLE GUARD (VTGUARD)

An exploit mitigation applied to EdgeHTML but not in its dependencies is Virtual Table Guard (VTGuard) [42]. VTGuard was first introduced in IE10 and its purpose is to detect an invalid virtual function table which can occur if an exploit is trying to control execution flow via a controlled C++ object in memory. It adds an ASLR-randomized value (__vtguard) in the virtual function table which is then checked before performing a virtual function call:



A shortcoming of this mitigation is that it is applied only to select EdgeHTML classes and it can be trivially bypassed if the address of __vtguard is leaked via memory content disclosure.

4.5. MEMORY GC (MEMGC)

Memory GC (MemGC) [59] was first introduced in EdgeHTML and in MSHTML on Windows 10. It is the successor to the Memory Protector [60, 61, 62] exploit mitigation in the aforementioned rendering engines on Windows 10.

Similar to Memory Protector, the purpose of MemGC is to mitigate exploitation of use-after-free [26] vulnerabilities by preventing the freeing of memory chunks if references to them are found. However, unlike Memory Protector which only checks the registers and the stack for memory chunk references, MemGC additionally scans the contents of MemGC-managed chunks for references. This additional check means that MemGC further decreases the number of usable use-after-free bugs that an attacker can exploit.

CONFIGURATION

MemGC is enabled by default. One way to configure it in both Edge and IE is via the "OverrideMemoryProtectionSetting" configuration which can be set via the following registry entry:

```
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Internet Explorer\Main
OverrideMemoryProtectionSetting = %DwordValue%
```

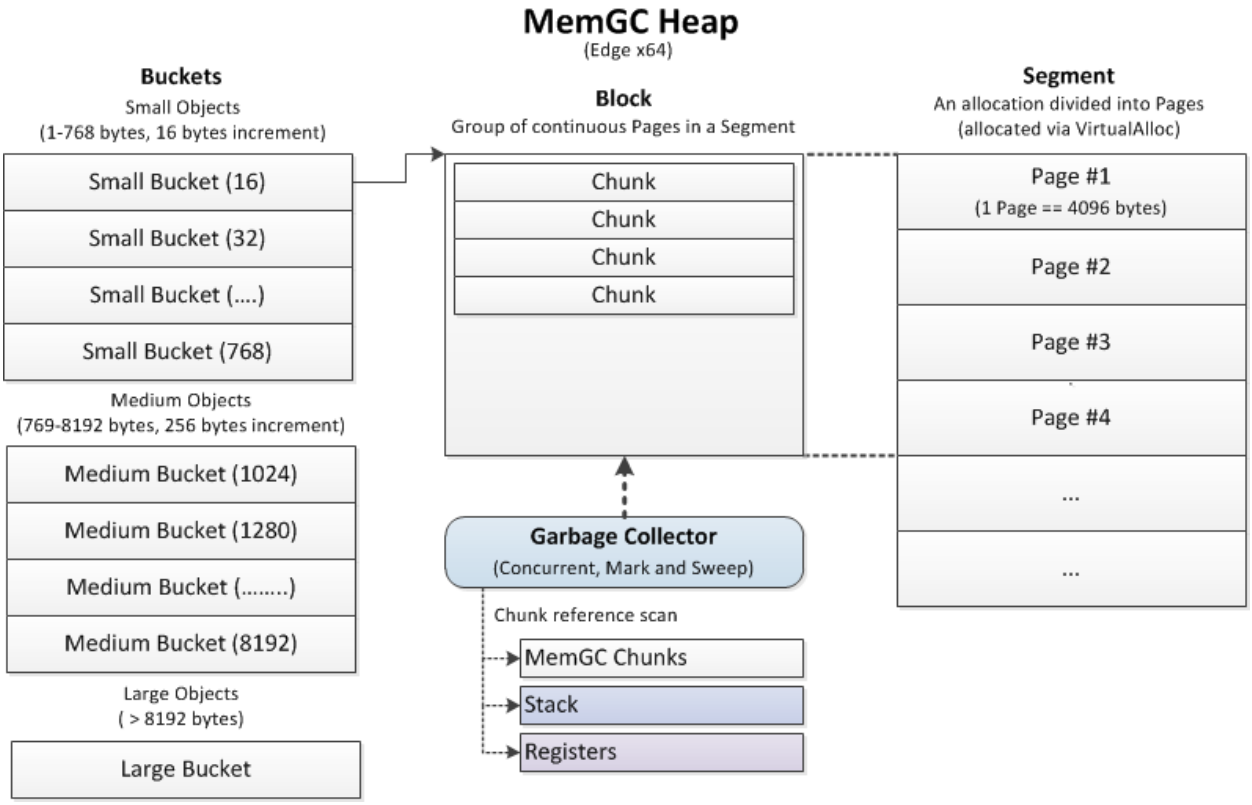
Where %DwordValue% can be any of the following:

Value	Meaning
3	MemGC is enabled (default)
2	Memory Protector is enabled (Force mark-and-reclaim)
1	Memory Protector is enabled
0	MemGC and Memory Protector are disabled

MEMGC HEAP

MemGC uses a separate managed heap (MemGC heap) for object allocation and a concurrent garbage collector that performs a mark-and-sweep operation to identify and reclaim unreferenced chunks in the MemGC heap. MemGC depends on the Chakra JavaScript engine memory management routines for most of its functionality.

The allocation scheme used by MemGC involves committing chunks of memory called Segments using VirtualAlloc() and then dividing these Segments into 4096-byte Pages. A group of these Pages are then treated as a Block which is in turn used in the allocation of similarly-sized objects:



EdgeHTML/MSHTML DOM objects and a large number of internal rendering engine objects are managed by MemGC. And since MemGC already uses a separate managed heap, the Isolated Heap is unused if MemGC is enabled.

ALLOCATION

In EdgeHTML's implementation of MemGC, when a MemGC-managed object needs to be allocated, edgehtml!MemoryProtection::HeapAlloc<1>() or edgehtml!MemoryProtection::HeapAllocClear<1>() is called which in turn calls chakra!MemProtectHeapRootAlloc(). Chakra!MemProtectHeapRootAlloc() will then allocate a chunk from a Block in the appropriate bucket, and then flag the chunk as a "root". "Root" in garbage collection parlance means that the object/chunk is directly referenced by the program and therefore should not be garbage collected. Root objects/chunks are also used by the garbage collector when scanning for chunk references.

PROTECTED FREEING

When an object is to be freed, edgehtml!MemoryProtection::HeapFree() is called which in turn calls chakra!MemProtectHeapUnrootAndZero(). Chakra!MemProtectHeapUnrootAndZero() attempts to locate the Block where the object's chunk is located, zero-out the chunk, then clear the "root" flag of the chunk. By clearing the "root" flag of the chunk, the chunk will become a candidate for garbage collection and will be reclaimed if no references to the chunk are found by the garbage collector (more on this below).

GARBAGE COLLECTION

Once the total size of the unrooted chunks reaches a dynamically computed threshold value, garbage collection will be triggered via chakra!MemProtectHeap::Collect(). The garbage collection (a complicated process in which only the core functions are described here) will perform a mark-and-sweep operation to reclaim unreferenced unrooted chunks. Some parts of the mark-and-sweep operation are executed in a separate thread (chakra!Memory::Recycler::ThreadProc) that is notified via chakra!Memory::Recycler::StartConcurrent().

In the marking phase, the mark bit for all chunks are first cleared and then all root chunks are marked (via chakra!Memory::Recycler::BackgroundResetMarks()). Then, the root chunks (via chakra!Memory::Recycler:: ScanImplicitRoots()), the registers and the stack (via chakra!MemProtectHeap::FindRoots()) are scanned for chunk pointers, referenced chunks found in the scan are marked. Chunks that are not marked after the marking phase will eventually be made available for reallocation.

As of this writing, MemGC and Memory Protector still have no known bypass for covered cases, but as with other exploit mitigations, new bypass techniques may be developed in the future. However, exploits were demonstrated for cases not covered Memory Protector [61], a technique on bypassing ASLR on 32-bit IE by leveraging Memory Protector was also published [61], and finally, a timing attack by leveraging Memory Protector to approximate the address range of bottom-up allocations (which includes heap allocations) on 64-bit IE was also published [63].

4.6. ANALYSIS AND SUMMARY: EXPLOIT MITIGATIONS

The default applied exploit mitigations to the Edge content process where EdgeHTML is hosted are more comprehensive compared to the default exploit mitigations applied in IE11's content process running on Windows 10, Windows 8 (desktop IE) and Windows 7 – it runs 64-bit by default which allowed ASLR to be additionally operated in HEASLR mode. This makes traditional heap spraying techniques infeasible or very unreliable and attackers would need to develop more precise techniques in order to write controlled data in a reliable location.

Another major difference is that the more restrictive AppContainer is used for sandboxing the content process where EdgeHTML is hosted. This greatly limits the access and capabilities of a successful rendering engine exploit and, unless the remote vulnerability also exists in a privileged process or system component [20], would require exploitation of a separate vulnerability in order to escape the AppContainer sandbox.

Stack Buffer Security Check (/GS) decreases the number potentially exploitable stack-based vulnerabilities and the recently introduced Control Flow Guard required the development of new exploitation techniques. Finally, MemGC, like its predecessor, will further decrease the number of exploitable use-after-free vulnerabilities in EdgeHTML.

Overall, with all these exploit mitigations in place, an attacker would need to invest more in finding exploitable vulnerabilities in EdgeHTML, and more investment in developing reliable exploits for them. Nonetheless, attackers will continually find novel ways to bypass these mitigations and we could expect that these exploit mitigations will evolve over time.

5. CONCLUSION

It is inevitable that the attack surface of EdgeHTML (and other browser rendering engines) will continue to expand as new web standards are implemented. Most of the increase in its attack surface will come from parsing and translating new markup/style specifications, and most notably, in the new functionalities that will be exposed to developers (and attackers alike) via the DOM API.

However, the increase in the attack surface in EdgeHTML is balanced by the comprehensive exploit mitigations applied to the content process where it is hosted, to the libraries it uses, and to EdgeHTML itself. These applied exploit mitigations will make a number of EdgeHTML vulnerabilities unexploitable or very costly/difficult to develop an exploit for.

Also, new or additional research in the following areas related to EdgeHTML will be both important and interesting as they are libraries/features that are remotely-reachable and widely-used:

- Internals research, code audit and fuzzing of the following Windows components which are used by EdgeHTML: XmlLite, MSXML6, Windows Imaging Component (WIC), Media Foundation (MF), DirectWrite and the WinRT PDF Renderer. Some of them may already have published research (such as DirectWrite [20]) and we need more of them so we have an understanding of the security posture of these critical components.
- Internals (algorithm details, data structures map, etc.), heap grooming, heap metadata attacks (if possible), and bypass research for MemGC. Initial research of MemGC was discussed here; further analysis of its internals, how the MemGC heap can be groomed or attacked, and how MemGC can be bypassed will be beneficial to the understanding of its weaknesses, and eventually, improvements to it.

Finally, I hope that this paper helped you understand the security aspect of the EdgeHTML rendering engine.

6. BIBLIOGRAPHY

[1] C. Morris, "A break from the past: the birth of Microsoft's new web rendering engine," [Online]. Available: http://blogs.msdn.com/b/ie/archive/2015/02/26/a-break-from-the-past-the-birth-of-microsoft-s-new-web-rendering-engine.aspx.

[2] J. Rossi, "Inside Microsoft's New Rendering Engine For The "Project Spartan"," [Online]. Available: http://www.smashingmagazine.com/blog/2015/01/26/inside-microsofts-new-rendering-engine-project-spartan/.

[3] C. Morris and J. Rossi, "A break from the past, part 2: Saying goodbye to ActiveX, VBScript, attachEvent," [Online]. Available: https://blogs.windows.com/msedgedev/2015/05/06/a-break-from-the-past-part-2-saying-goodbye-to-activex-vbscript-attachevent/.

[4] "Chromium Git Repository (chromium/chromium/blink/master/./Source/modules/webaudio)," [Online]. Available: https://chromium.googlesource.com/chromium/blink/+/master/Source/modules/webaudio/.

[5] "Chromium Git Repository (chromium/chromium/blink/master/./Source/platform/audio)," [Online]. Available: https://chromium.googlesource.com/chromium/blink/+/master/Source/platform/audio/.

[6] Microsoft, "MSDN - Web Audio," [Online]. Available: https://msdn.microsoft.com/en-us/library/dn985708.aspx.

[7] J. Oh, "Fight against 1-day exploits: Diffing Binaries vs Anti-diffing Binaries," [Online]. Available: http://www.blackhat.com/presentations/bh-usa-09/OH/BHUSA09-Oh-DiffingBinaries-PAPER.pdf.

[8] Microsoft, "MSDN - XmlLite," [Online]. Available: https://msdn.microsoft.com/en-us/library/ms752872.aspx.

[9] Microsoft, "MSDN - MSXML," [Online]. Available: https://msdn.microsoft.com/en-us/library/ms763742.aspx.

[10] Microsoft, "MSDN - Security Considerations: Element Behaviors," [Online]. Available: https://msdn.microsoft.com/en-us/library/aa753685.aspx.

[11] IBM Corporation, "Microsoft Internet Explorer Vector Markup Language Exploit Alert," [Online]. Available: http://www.iss.net/threats/237.html.

[12] MITRE, "CVE-2010-3971," [Online]. Available: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3971.

[13] Microsoft, "MSDN - Windows Imaging Component," [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ee719902.aspx.

[14] MITRE, "CVE-2005-4560," [Online]. Available: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4560.

[15] MITRE, "CVE-2008-2249," [Online]. Available: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2249.

[16] MITRE, "CVE-2008-1083," [Online]. Available: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1083.

[17] Microsoft, "MSDN - Microsoft Media Foundation," [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ms694197.aspx.

[18] Microsoft, "MSDN - Timed text tracks," [Online]. Available: https://msdn.microsoft.com/en-us/library/bg123962.aspx.

[19] Microsoft, "MSDN - @font-face rule," [Online]. Available: https://msdn.microsoft.com/en-us/library/ms530757.aspx.

[20] M. Jurczyk, "One font vulnerability to rule them all," [Online]. Available: http://j00ru.vexillium.org/dump/recon2015.pdf.

[21] MITRE, "CVE-2011-3402," [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3402.

[22] J. Wolf, "CVE-2011-3402 - Windows Kernel TrueType Font Engine Vulnerability," [Online]. Available: https://cansecwest.com/slides/2013/Analysis%20of%20a%20Windows%20Kernel%20Vuln.pdf.

[23] Microsoft, "MSDN - DirectWrite," [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/dd368038.aspx.

[24] B. Cavenah, "MS09-029: Vulnerabilities in the EOT parsing engine," [Online]. Available: http://blogs.technet.com/b/srd/archive/2009/07/14/ms09-029-vulnerabilities-in-the-eot-parsing-engine.aspx.

[25] Microsoft, "MSDN - Basic DOM Reference," [Online]. Available: https://msdn.microsoft.com/en-us/library/hh771916.aspx.

[26] M. V. Yason, "Use-after-frees: That pointer may be pointing to something bad," [Online]. Available: http://securityintelligence.com/use-after-frees-that-pointer-may-be-pointing-to-something-bad.

[27] A. Pelletier, "Advanced Exploitation of Internet Explorer Heap Overflow (Pwn2Own 2012 Exploit)," [Online]. Available: http://www.vupen.com/blog/20120710.Advanced_Exploitation_of_Internet_Explorer_HeapOv_CVE-2012-1876.php.

[28] I. Fratric, "Exploiting Internet Explorer 11 64-bit on Windows 8.1 Preview," [Online]. Available: http://ifsec.blogspot.com/2013/11/exploiting-internet-explorer-11-64-bit.html.

[29] Microsoft, "Internet Explorer Web Platform Status and Roadmap - status.modern.IE," [Online]. Available:

https://status.modern.ie/.

[30] PDF API Team, "Introduction to PDF APIs in Windows 8.1," [Online]. Available: http://blogs.technet.com/b/pdf_api_blog/archive/2013/11/11/introduction-to-pdf-apis-in-windows-8-1.aspx.

[31] F. Falcón, "Exploiting CVE-2015-0311, Part II: Bypassing Control Flow Guard on Windows 8.1 Update 3," [Online]. Available: https://blog.coresecurity.com/2015/03/25/exploiting-cve-2015-0311-part-ii-bypassing-control-flow-guard-on-windows-8-1-update-3/.

[32] H. Li , "Smashing the Heap with Vector: Advanced Exploitation Technique in Recent Flash Zero-day Attack," [Online]. Available: https://sites.google.com/site/zerodayresearch/smashing_the_heap_with_vector_Li.pdf.

[33] D. Kindlund, X. Chen, M. Scott, D. Caselden and N. Moran, "Operation SnowMan: DeputyDog Actor Compromises US Veterans of Foreign Wars Website," [Online]. Available: https://www.fireeye.com/blog/threat-research/2014/02/operation-snowman-deputydog-actor-compromises-us-veterans-of-foreign-wars-website.html.

[34] M. Brand and C. Evans, "Significant Flash exploit mitigations are live in v18.0.0.209," [Online]. Available: http://googleprojectzero.blogspot.com/2015/07/significant-flash-exploit-mitigations_16.html.

[35] N. Waisman, "Understanding and bypassing Windows Heap Protection," [Online]. Available: http://www.immunityinc.com/downloads/Heap_Singapore_Jun_2007.pdf.

[36] B. Moore, "Heaps About Heaps," [Online]. Available: https://www.insomniasec.com/downloads/publications/Heaps_About_Heaps.ppt.

[37] B. Hawkes, "Attacking the Vista Heap," [Online]. Available: http://www.blackhat.com/presentations/bh-usa-08/Hawkes/BH_US_08_Hawkes_Attacking_Vista_Heap.pdf.

[38] J. McDonald and C. Valasek, "Practical Windows XP/2003 Heap Exploitation," [Online]. Available: https://www.blackhat.com/presentations/bh-usa-09/MCDONALD/BHUSA09-McDonald-WindowsHeap-PAPER.pdf.

[39] C. Valasek, "Understanding the Low Fragmentation Heap," [Online]. Available: http://illmatics.com/Understanding_the_LFH.pdf.

[40] C. Valasek and T. Mandt, "Windows 8 Heap Internals," [Online]. Available: http://illmatics.com/Windows%208%20Heap%20Internals.pdf.

[41] Y. Chen, "The Birth of a Complete IE11 Exploit Under the New Exploit Mitigations," [Online]. Available: https://syscan.org/index.php/download/get/aef11ba81927bf9aa02530bab85e303a/SyScan15%20Yuki%20Chen%20-%20The%20Birth%20of%20a%20Complete%20IE11%20Exploit%20Under%20the%20New%20Exploit%20Mitigations.pdf.

[42] K. Johnson and M. Miller, "Exploit Mitigation Improvements in Windows 8," [Online]. Available:

http://media.blackhat.com/bh-us-12/Briefings/M_Miller/BH_US_12_Miller_Exploit_Mitigation_Slides.pdf.

[43] M. Miller, "Software defense: mitigating common exploitation techniques," [Online]. Available: http://blogs.technet.com/b/srd/archive/2013/12/11/software-defense-mitigating-common-exploitation-techniques.aspx.

[44] M. Miller and W. Peteroy, "Mitigating the LdrHotPatchRoutine DEP/ASLR bypass with MS13-063," [Online]. Available: http://blogs.technet.com/b/srd/archive/2013/08/12/mitigating-the-ldrhotpatchroutine-dep-aslr-bypass-with-ms13-063.aspx.

[45] F. Serna, "The info leak era on software exploitation," [Online]. Available: https://media.blackhat.com/bh-us-12/Briefings/Serna/BH_US_12_Serna_Leak_Era_Slides.pdf.

[46] M. V. Yason, "Diving Into IE10's Enhanced Protected Mode Sandbox," [Online]. Available: https://www.blackhat.com/docs/asia-14/materials/Yason/WP-Asia-14-Yason-Diving-Into-IE10s-Enhanced-Protected-Mode-Sandbox.pdf.

[47] T. Ormandy and J. Tinnes, "There's a party at ring0 and you're invited," [Online]. Available: https://www.cr0.org/paper/to-jt-party-at-ring0.pdf.

[48] Nils and J. Butler, "MWR Labs Pwn2Own 2013 Write-up - Kernel Exploit," [Online]. Available: https://labs.mwrinfosecurity.com/blog/2013/09/06/mwr-labs-pwn2own-2013-write-up---kernel-exploit/.

[49] J. Forshaw, "Digging for Sandbox Escapes - Finding sandbox breakouts in Internet Explorer," [Online]. Available: https://www.blackhat.com/docs/us-14/materials/us-14-Forshaw-Digging-For_IE11-Sandbox-Escapes.pdf.

[50] P. Sabanal and M. V. Yason, "Digging Deep Into The Flash Sandboxes," [Online]. Available: https://media.blackhat.com/bh-us-12/Briefings/Sabanal/BH_US_12_Sabanal_Digging_Deep_WP.pdf.

[51] Microsoft, "MSDN - /GS (Buffer Security Check)," [Online]. Available: https://msdn.microsoft.com/en-us/library/8dbf701c.aspx.

[52] A. Sotirov and M. Dowd, "Bypassing Browser Memory Protections - Setting back browser security by 10 years," [Online]. Available: https://www.blackhat.com/presentations/bh-usa-08/Sotirov_Dowd/bh08-sotirov-dowd.pdf.

[53] M. Miller, T. Burrell and M. Howard, "Mitigating Software Vulnerabilities," [Online]. Available: http://blogs.technet.com/b/srd/archive/2011/07/12/mitigating-software-vulnerabilities.aspx.

[54] C. Evans, "What is a "good" memory corruption vulnerability?," [Online]. Available: http://googleprojectzero.blogspot.com/2015/06/what-is-good-memory-corruption.html.

[55] J. Hogg, "Visual Studio 2015 Preview: Work-in-Progress Security Feature," [Online]. Available: http://blogs.msdn.com/b/vcblog/archive/2014/12/08/visual-studio-2015-preview-work-in-progress-security-feature.aspx.

[56] Microsoft, "MSDN - /guard (Enable Control Flow Guard)," [Online]. Available: https://msdn.microsoft.com/en-us/library/dn919635.aspx.

[57] MJ0011, "Windows 10 Control Flow Guard Internals," [Online]. Available: http://powerofcommunity.net/poc2014/mj0011.pdf.

[58] J. Tang, "Exploring Control Flow Guard in Windows 10," [Online]. Available: http://sjc1-te-ftp.trendmicro.com/assets/wp/exploring-control-flow-guard-in-windows10.pdf.

[59] C. Cowan, "Microsoft Edge: Building a safer browser," [Online]. Available: http://blogs.windows.com/msedgedev/2015/05/11/microsoft-edge-building-a-safer-browser/.

[60] M. V. Yason, "Understanding IE's New Exploit Mitigations: The Memory Protector and the Isolated Heap," [Online]. Available: http://securityintelligence.com/understanding-ies-new-exploit-mitigations-the-memory-protector-and-the-isolated-heap.

[61] A.-A. Hariri, S. Zuckerbraun and B. Gorenc, "Abusing Silent Mitigations: Understanding weaknesses within Internet Explorer's Isolated Heap and MemoryProtection," [Online]. Available: http://h30499.www3.hp.com/hpeb/attachments/hpeb/off-by-on-software-security-blog/599/1/WP-Hariri-Zuckerbraun-Gorenc-Abusing_Silent_Mitigations.pdf.

[62] @promised_lu, "New Exploit Mitigation In Internet Explorer," [Online]. Available: http://hitcon.org/2014/downloads/P2_01_Keen%20Team%20-%20New%20Exploit%20Mitigation%20In%20Internet%20Explorer.pdf.

[63] I. Fratric, "Dude, where's my heap?," [Online]. Available: http://googleprojectzero.blogspot.com/2015/06/dude-wheres-my-heap.html.