Sri Lanka Institute of Information Technology

Chromium Based Microsoft Edge

Case Study: Vulnerability History of Microsoft Edge

Secure Software Engineering – IE4042

Assignment

Submitted by Thomas R.L - IT19095936

Contents

Chapter 1: Domain and Historical Analysis

Product Overview

Description

Microsoft Edge is a cross-platform web browser that Microsoft created and developed. It was first included with Windows 10 and Xbox One in 2015, and it was later made available for other platforms, including Android and iOS in 2017, macOS and older Windows versions (Windows 7 and later) in 2019, and Linux in 2020. In Windows 11, the Microsoft-based Edge browser replaced Internet Explorer (IE) as the default web browser (for compatibility with Google Chrome). Edge was originally built with Microsoft's proprietary browser engine EdgeHTML and their Chakra JavaScript engine, and is now known as Microsoft Edge Legacy. Microsoft announced plans to rebuild the browser as Microsoft-based with Blink and V8 engines in December 2018.

Microsoft Edge has replaced Internet Explorer 11 and Internet Explorer Mobile as the default web browser on Windows 10, Windows 10 Mobile, Windows 11, Xbox One, and Xbox Series X and Series S consoles. It is not included in Windows 10 Enterprise Long-Term Servicing Channel (LTSC) builds because its development and release are dependent on the Windows as a service model. Microsoft initially stated that Edge would support the legacy MSHTML (Trident) browser engine for backwards compatibility, but later stated that Edge would use a new engine due to "strong feedback," while Internet Explorer would continue to provide the legacy engine. The EdgeHTML-based versions' developer toolset included an option to emulate the rendering behavior ("document mode") of Internet Explorer versions 5 to 11.

The Hub, a sidebar that provides functionality similar to Internet Explorer's Downloads manager and Favorites Center, allows you to view your favorites, reading list, browsing history, and downloads. Edge includes a built-in PDF reader, eliminating the need to install a separate program, and it supports WebAssembly. Edge also included an integrated Adobe Flash Player until January 2021. (with an internal whitelist allowing Flash applets on Facebook websites to load automatically, bypassing all other security controls requiring user activation). Edge does not support legacy technologies like ActiveX and Browser Helper Objects, instead relying on an extension system. For compatibility, Internet Explorer 11 is still available alongside Edge on Windows 10; it is identical to the Windows 8.1 version and does not use the Edge engine, as previously announced. Microsoft Edge became the sole browser available in Windows 11. It does, however, include an "Internet Explorer mode" to address compatibility issues.

Edge works with Microsoft's online platforms to provide voice control, search functionality, and dynamic information related to address bar searches. Users can annotate web pages and save them to and share them with OneDrive, as well as save HTML and MHTML pages to their computers. It also works with the "Reading List" feature and has a "Reading Mode" that removes unnecessary formatting from pages to improve legibility. Edge also has a new vertical tabs feature that allows users to move tabs on the left side of the screen. In March 2016, build 14291 added preliminary support for browser extensions, initially supporting three extensions. Microsoft stated that the reason for the delay in allowing extensions and the low number was due to security concerns.

Overview of findings

Projects like Microsoft Edge must continue to guard against exploits and patch vulnerabilities as soon as they are found as web browser functionality increases. Despite the project contribution process's efforts to cut down on potential bugs, new vulnerabilities can still be created. The Microsoft Edge project has a number of vulnerabilities that have been marked as fixed, and this paper will discuss a few of them. The analysis of the flaws listed below will go over their underlying causes, how they were fixed, and suggested engineering procedures that could have avoided them in the first place.

Product Assets

Numerous assets owned by Microsft Edge may be vulnerable to security breaches. There is a significant risk if an attacker were to compromise the system because the project serves as the foundation for many browsers. The user data, which includes user profiles, cookies, preferences, and browser histories, is one of Microsoft's most valuable resources. Since most users automatically trust their browsers, it's crucial to keep these assets secure to maintain the user's trust. Most people believe that emails and malicious websites are the only sources of danger. The project may suffer if this asset is not safeguarded. Microsoft's ability to add extensions is another one of its key advantages. Although the Microsoft team does not maintain the extensions, they are in charge of how they might impact the system since they are third-party software. This makes it possible for unofficial software to have an impact on the browser. Since many of them access crucial browser functionality, extensions may pose a risk to the system if not handled properly. Allowing extensions these privileges can lead to cross-site scripting, denial-of-service attacks, and even the download of malicious software onto a user's computer. As shown in the first vulnerability analysis below, there was a significant security risk when relying on a strong extension.

To allow web applications to provide richer functionality, the Microsoft Edge Project must protect its users from web-based threats. However, Microsoft Edge can be abused through phishing, cross-site scripting, and other web-based threats, just like any other browser on any other platform. A browser connects the user's environment to all web services. This information could be very helpful to a hacker. Microsoft Edge must keep the browser and its processes safe in a sandbox to prevent access to the system that is using it. If an attacker managed to get out of this sandbox, the OS would be responsible for reducing security risks. Additionally, if a user's browser directory is kept on a server, the server could be hacked, giving anyone access to the cache and history. For a personal computer, this is less of an issue, but if the device is linked to an enterprise system, the cached data and history may contain sensitive data.

In their threat model, Microsoft Edge takes into account two different types of adversaries: opportunistic adversaries and dedicated adversaries. An attacker who uses attacks to lure users to websites that compromise their computers or to apps that attempt to obtain unauthorized privileges is known as an opportunistic adversary because they do not specifically target any one user or enterprise for their attacks. On the other hand, a committed adversary may choose to attack a user or an organization and is ready to steal devices in order to recover data or login credentials. They will act in any way that a cunning foe might.

Example Attacks

Three different attack paths are available through a web browser: A web browser user, the web browser program itself, or a web application running inside the web browser are all potential targets of an attack. In light of this, we can investigate potential flaws that could be used to undermine the security of a web browser application, like Microsoft Edge.

An effective web browser works to defend the user against social engineering attacks like phishing. This is handled by Microsoft Edge by displaying alerts for known malicious websites. Without enough anti-phishing protection, the browser might enable a man-in-the-middle attack, which would serve an unaware victim a different web page than the one they thought they were accessing. The attacker can obtain the victim's personal information from there if they provide any sensitive information. This jeopardizes both user privacy and the accuracy of website data (the page a user accesses is the one they intended to).

A web browser can also be attacked by making use of its features against it. Third-party programs, also referred to as extensions, can be installed and used in Microsoft Edge. This will enable developers to enhance the functionality that the web browser offers. The browser must guard against the possibility of exploiting vulnerabilities in these extensions because they require a certain level of trust in order to run code.

Chrome comes pre-installed with a Flash extension. If a flaw is later discovered in that version of Flash, an attacker could take advantage of it and use the Chrome web browser to execute code on a victim's computer. Microsoft Edge defends against this, among other things, by operating in a "sandboxed process," which forces an attacker to find a second exploit to leave the sandbox in order to run code on the system. Such an attack would compromise the system's integrity.

A web application is another typical target for attack via a web browser. Numerous flaws, including cross-site request forgery and cross-site scripting, can be used by an attacker to compromise the system's confidentiality. A malicious actor might try to get the web browser to run Javascript code in the context of another web app, which could expose private information in that web app to a third party. An attacker could intercept user input by enclosing a real application in an invisible HTML frame. Clickjacking is an attack that could jeopardize the system's confidentiality and integrity. Microsoft Edge attempts to combat these problems by warning users or blocking specific cross-site scripting attacks.

Vulnerability History

The team discovered four intriguing vulnerabilities to investigate given their understanding of Microsoft edge's known assets and vulnerabilities. The severity of these varies from high to low. Each vulnerability's team notes are listed below, along with a thorough explanation of its particular history in the Microsoft edge network.

CVE-2022-3075: Insufficient data validation in Mojo

Google says it anticipates the update to roll out to all users in the upcoming days or weeks after an anonymous tipster reported the issue on August 30. The Verge claims that the flaw stems from "Insufficient data validation" in Mojo, a set of runtime libraries used by Chromium, the codebase upon which Google Chrome and Microsoft Edge are based. Google is aware of rumors that an exploit is being used in real-world attacks, but no specifics have been disclosed. To close the hole, Google and Microsoft both released updates. The problem is a case of incomplete data validation in Mojo, a set of runtime libraries that offers a platform-independent method for inter-process communication (IPC). Google is limiting access to bug information until the vast majority of users have received a fix. We'd love to be able to determine whether this bug results in a concerning security outcome like EoP, short for elevation of privilege, or if it can be abused for a more disastrous outcome like full-blown RCE, short for remote code execution, given that the bug relates to the improper handling of input data.

In case you're wondering, Mojo is the name of a Google code library for inter-process communication, or IPC. A process, broadly speaking, can be made up of multiple threads, which are essentially "sub-processes" inside the main process. This allows a single program to quietly carry out two tasks at once, such as printing a document while you scroll through it or performing a background spelling check. Because all threads inside a process have access to the same memory chunk, splitting a single-process application into threads is more practical (by which we mean "is much quicker and easier, but way less secure") than splitting it into separate processes.

Because they can just access the same common pool of data, threads can interact and share information much more easily. Examples include checking the current configuration settings, exchanging memory addresses, sharing file handles, reusing cached images directly from RAM, and much more. However, because all of the program's components share a single large memory space, a bug in one area of the program, such as the thread that is busy rendering and displaying your first open browser tab, could interfere with or negatively impact other areas of the program, such as the threads handling the other tabs you have open. In order to prevent one runaway tab from trivially stealing data, such as cookies and access tokens, from other tabs related to completely different websites, modern browsers typically divide themselves into numerous separate processes, such that each tab is handled in an independent process.

Microsoft CVE-2020-0816: Microsoft Edge Memory Corruption Vulnerability

When Microsoft Edge improperly accesses memory objects, a remote code execution vulnerability exists. The flaw could corrupt memory, allowing an attacker to execute arbitrary code in the context of the current user. A successful exploit of the vulnerability could grant the attacker the same user rights as the current user. An attacker could take control of an affected system if the current user is logged in with administrative user rights. After that, an attacker could install programs, view, change, or delete data, or create new accounts with full user rights. In order to exploit the vulnerability in Microsoft Edge, an attacker could host a specially crafted website and persuade a user to view it. By including specially crafted content that could exploit the vulnerability, the attacker could also take advantage of infected websites and websites that accept or host user-provided content or advertisements. However, an attacker would never be able to compel users to view content they had access to. Instead, an attacker would need to persuade users to act, typically by offering an alluring incentive in an email or Instant

Messenger message or by persuading them to open an email attachment. The security update changes how Microsoft Edge manages objects in memory to fix the flaw.

CVE-2017-0002: Microsoft Edge Elevation of Privilege Vulnerability

When using about: blank to enforce cross-domain policies, Microsoft Edge has an elevation of privilege vulnerability that could let an attacker access data from one domain and inject it into another. In the affected versions of Microsoft Edge, an attacker who was successful in exploiting this vulnerability could increase their privileges.

An attacker could host a website that is used to try to exploit the vulnerability in a web-based attack scenario. Additionally, maliciously constructed content could be present on compromised websites and websites that accept or host user-provided content. However, an attacker would never be able to compel users to view content they had access to. An attacker would need to persuade users to act instead. Users could be tricked by an attacker, for instance, into clicking a link that leads to the attacker's website.

The update addresses the vulnerability by assigning a unique origin to top-level windows that navigate to Data URLs.

A security update has been released which resolves the above vulnerability in Microsoft Edge.

CVE-2022-2294: Heap buffer overflow in WebRTC

A heap buffer overflow was discovered to be regularly exploited in the open-source real-time communications technology WebRTC. There hasn't been much information on how to protect against the issue, other than an update. Microsoft refused to go further, stating only that since Edge "ingests" Chromium, the vulnerabilities had been fixed.

The other update in the maintenance release, CVE-2022-2295, addresses a type misunderstanding in V8, Google's JavaScript engine. It should be downloaded automatically. The problem seems to be plaguing the browser project of the search giant. Type confusion concerns existed with CVE-2022-1096 (fixed in March), which might result in out-of-bounds memory access.

Confusion may also surround the distribution of vulnerabilities across the major IT companies. Although CVEs given by industry partners (such as Chromium) are supported by Microsoft's Security Update Guide, it appears that the numbers have been flipped around and CVE-2022-2294 is reported as the type confusion and 2295 is listed as the WebRTC whoopsie. The Register contacted the business to get further information. What chance do we have that they will be able to repair the browser if they can't even get their numbers straight?

Initially, Microsoft's Edge browser ran on a custom-built, exclusive browser engine. 2019 saw the company admit defeat and switch to Chromium. In January 2020, a release to general availability took place. Since then, Microsoft has been pressuring users to download the app as it tries to win back lost users. Microsoft Edge reached double digits in market share at the time of writing, surpassing competitors like Firefox but still trailing Chrome by a significant margin.

In addition to a patch for the stable version, Microsoft also released a patch for the extended stable incarnation, bringing the version number to 102.0.1245.56. While the adoption of Chromium has reduced Edge's compatibility issues, vulnerabilities in the Chromium rendering engine will affect both Edge and Chrome (and other browsers based on the code).

CVE-2021-34506: Microsoft Edge Security Feature Bypass Vulnerability

The flaw, identified as CVE-2021-34506, results from universal cross-site scripting, or UXSS, which is activated when a webpage is automatically translated using the built-in feature of the Microsoft Edge browser via Microsoft translator. (See: Microsoft Customers Were Targeted by Group Behind SolarWinds Attack.) Vulnerabilities in the browser or its plug-ins are exploited in a UXSS attack. According to researchers, an attacker can trick victims into visiting a specially crafted website in order to get around security measures that have been put in place. In contrast to typical XSS attacks, UXSS is a type of attack that takes advantage of client-side flaws in browsers or browser extensions to create an XSS condition and run malicious codeThe behavior of the browser is affected and its security features may be disregarded or disabled when such vulnerabilities are discovered and used, according to a blog post by Cyber Xplore researchers. CVE-2021-34506 was found and reported by Ignacio Laurence, Vansh Devgan, and Shivam Kumar Singh, according to Microsoft. On June 3, he and Singh allegedly began their analysis. According to Devgan, they used the website translator Microsoft Edge Browser and discovered that it was loaded with XSS payloads. We noticed a strange number of pop-ups on Microsoft Edge, so we switched to Chrome and repeated the procedure. This time, however, there were none, according to Devgan.

According to Devgan, the translation feature contained vulnerable code that did not properly sanitize input. This flaw allowed an attacker to insert malicious JavaScript code anywhere on the webpage, which is then executed when a victim tries to translate the page. As a result, we both began investigating the system and discovered that the Microsoft Edge (Internal Translator That Comes Pre-Installed) has vulnerable code that actually takes any HTML tags containing a ">img tag without sanitizing the input or converting the payload into text while translating. As a result, that internal translator was actually taking ">img src=x onerror=alert(1)> payload and executing it as javascript because there were no proper validation.

The researchers asked a friend who uses Edge to act as the victim after creating a Facebook profile with a name in a different language and an XSS payload to show the vulnerability. According to Devgan, the auto translation led to an XSS pop-up hack as soon as the victim checked their profile.

The three researchers also made an attempt on YouTube and Google, according to Devgan, and both attempts were successful. "We have posted a review of HackENews on Google using a different language and an XSS payload. Anyone who clicked on the review link was compromised (the auto translation caused an XSS pop-up), and we entered a comment on YouTube with an XSS payload in a different language. Anyone using Edge to view that video was compromised (auto translation caused an XSS pop-up); "Devgan said.

On June 3, the researchers notified Microsoft of the incident for the first time. On June 7, Microsoft sent them an email requesting more information, which was complied with. The researchers were then awarded a $20,000 bounty, and on June 24 Microsoft released a patch update and assigned a CVE.

Chapter 2: Design Analysis

Architecture Overview

Chromium, which has a reputation for being a resource-intensive web browser, is the foundation upon which Edge was created. In order to ensure that the Chrome web browser functions properly, Google has made a number of upgrades and adjustments to both the Chrome web browser and the Chromium Base that supports it. On Windows 10 computers in particular, Chrome is still regarded as one of the largest memory eaters.

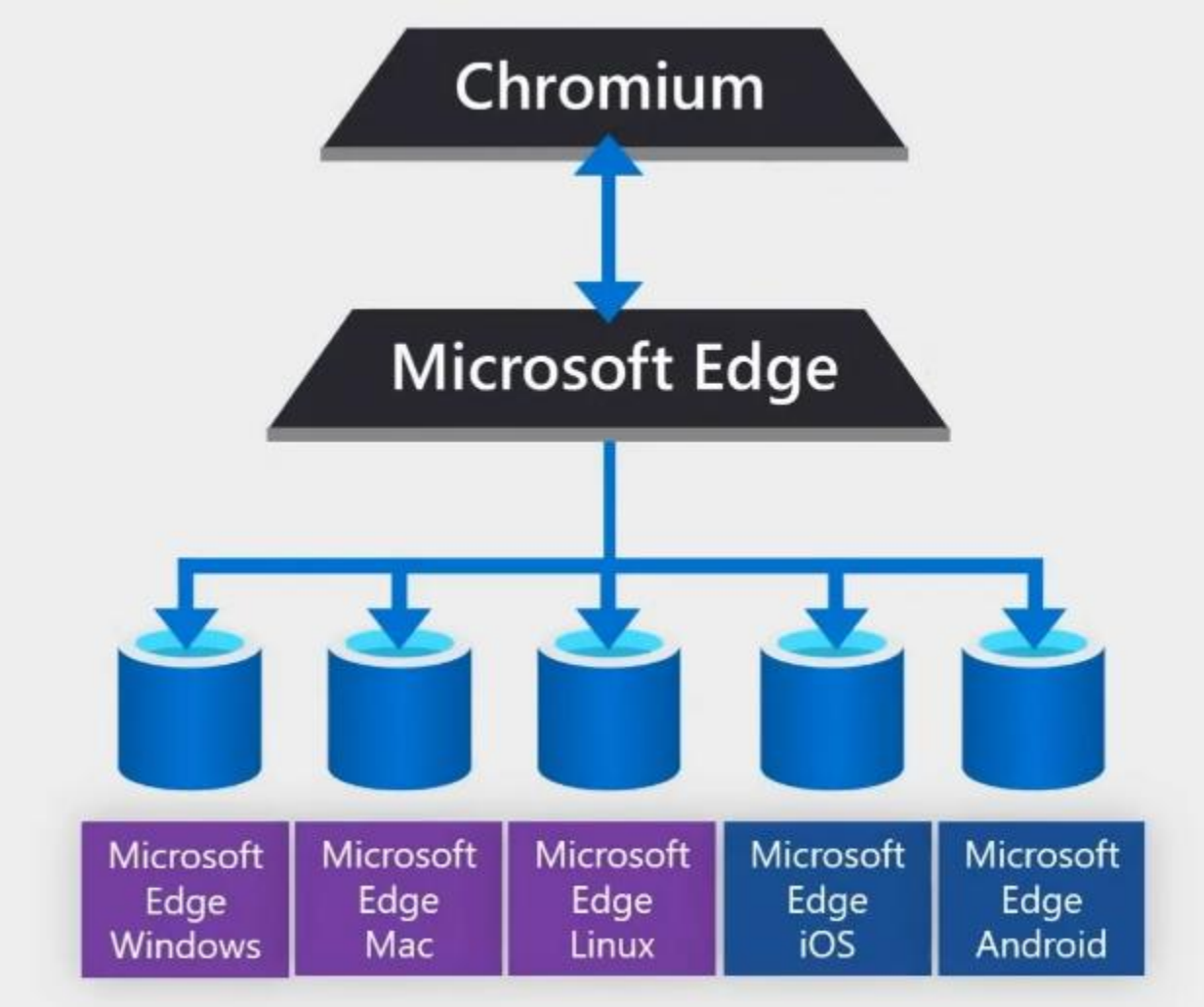

Figure 1 Microsoft Edge Software Stack Architecture Overview

The Google Chromium base is the same as that of the Microsoft Edge web browser. Microsoft, however, obviously wants to avoid having its browser branded as a resource-hungry browser and moving in the same way as Google Chrome. Microsoft has outlined the multi-process architecture of Microsoft Edge in a fairly lengthy blog post. The business made an effort to demonstrate how Microsoft Edge makes the

best use of system resources and listed some of the main advantages of using a multi-process architecture.

In essence, the Microsoft Edge browser is divided into various processes, yet each process cooperates with the others to provide users with a personalized surfing experience. The browser process, renderer process, GPU process, utility process, crashpad handler process, plug-in process, and extension process are the main processes that power the new Microsoft Edge web browser.

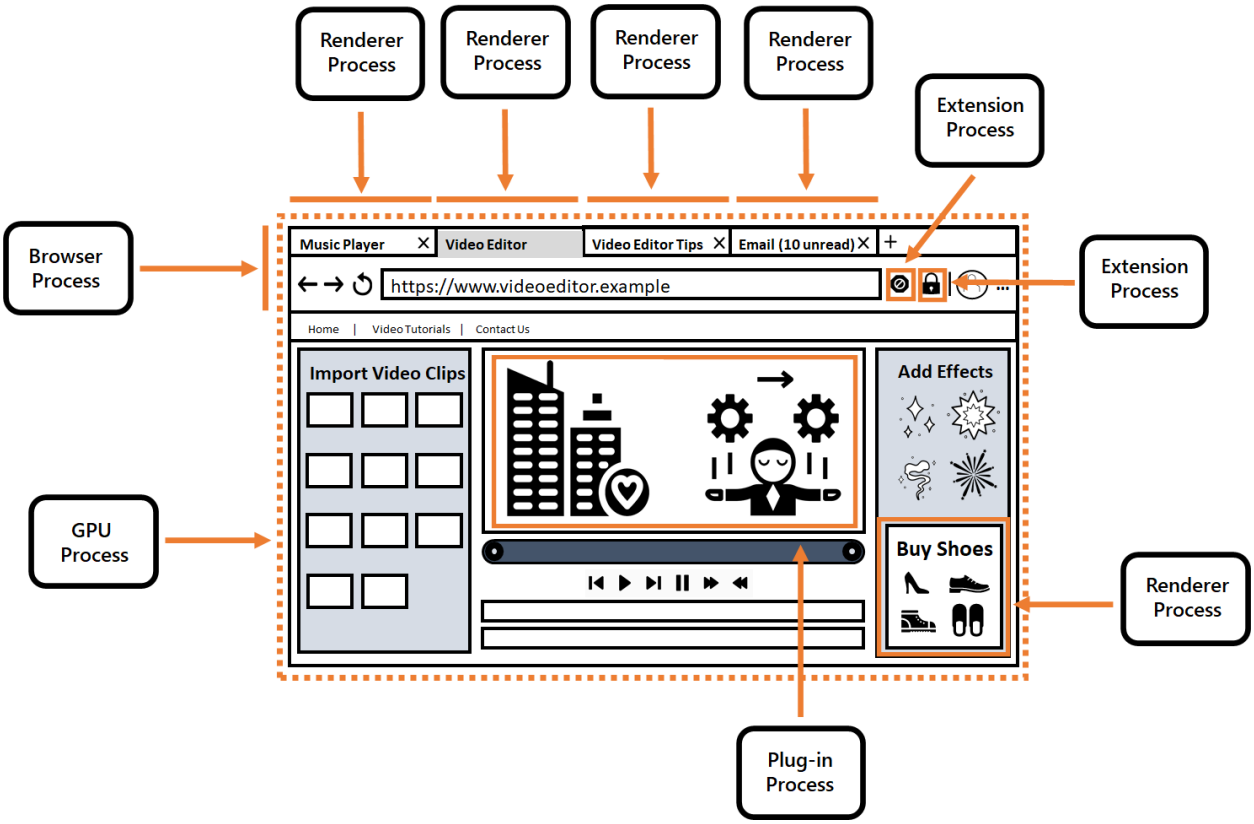

Figure 2 Microsoft Edge Multi-process Architecture

If you're wondering how this architecture benefits the browser in terms of security, the answer is that it makes it more difficult for malware to infect other processes after exploiting a security flaw in one of the renderer processes.

Because it interacts with the website, the renderer process is the one that is most likely to be attacked. Malware would be unable to gain control of the machine if it were to take control of this process since it has very low privileges and very limited access to the operating system. The renderer process and the browser process only have limited and secure contact. It is challenging for malware to use this to attack the browser process.

Process isolation further enhances the security of a browser by preventing one process from accessing the memory of another. Consider a scenario where you are buying a shirt online and come across an advertisement. Your credit card information is required by the website you're on to complete the

transaction; the ad does not require access to this data. Ads are integrated into their own processes so that, even if they are compromised, they won't have easy access to your personal data.

Targeted assaults, in which attackers design customized attacks against a particular business in an effort to seize control of corporate networks and data, have become an increasing threat to many enterprises throughout the world. Microsoft unveils Windows Defender Application Guard for Windows 10 Enterprise, a new layer of defense-in-depth security for organizations with the highest security standards. Application Guard uses Microsoft's Hyper-V virtualization technology to deliver unmatched defense against targeted threats.

In recent years, there has been a substantial shift in the threat environment. Today, over 90% of attacks start with a hyperlink in order to steal credentials, spread malware, or take advantage of weaknesses. This harms tens of thousands, if not millions, of users whose accounts and private information may be taken in addition to the firm that was targeted. Attackers that are highly driven and persistent will frequently begin with a social engineering technique by sending a personalized email to known corporate workers.

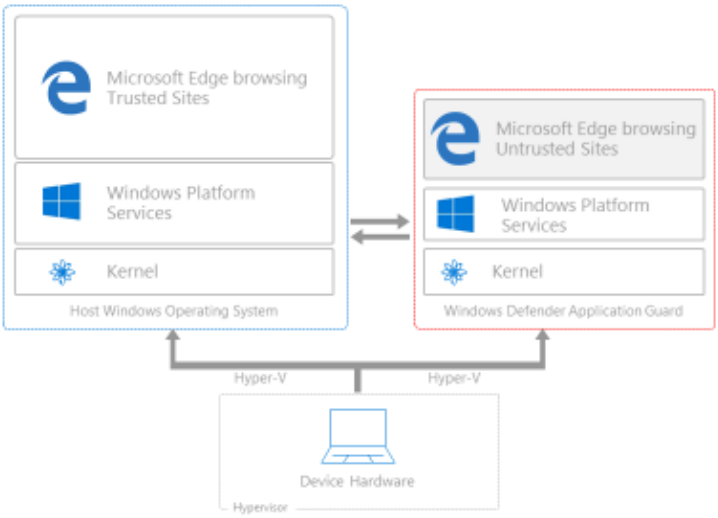

Figure 3 Isolation of new instances of Edge by Application Guard

To effect this disruption, Application Guard makes use of virtualization technology developed in the Microsoft Cloud. Microsoft Edge continues to function as it does right now when a user navigates to a reputable website, such as an internal accounting system web application. It has access to local storage, can use corporate credentials to authenticate users to internal websites, standard cookies function, users can store files to the local machine, and Windows, in general, just works. The Host version of Windows is this mode, which is indicated in the chart below by the blue outline.

However, Application Guard intervenes to isolate the possible threat when an employee browses to a website that the network administrator does not recognize or trust. Application Guard builds a fresh

instance of Windows at the hardware layer, complete with a distinct copy of the kernel and the bare minimum of Windows Platform Services needed to run Microsoft Edge, as seen in the configuration highlighted in red above. This distinct copy of Windows is prohibited from accessing the user's typical operating environment by the underlying hardware.

Even untrusted websites are frequently totally safe to access and free of malware; users simply assume that they will function as intended. These sites can operate essentially the same way they would if they were using the host version of Windows thanks to this isolated environment. In this instance, Application Guard does offer the fundamental functions that users would anticipate working, even when accessing untrusted websites, such as the ability to copy and paste text with the Windows clipboard and print it to their office printer. This enables the user to continue working while Application Guard is protecting the host. Using Microsoft management tools and policies, the enterprise administrator has control over this feature and can decide what they are comfortable with based on their own risk assessment.

Threat Model

The security parameters for the most significant assets of Chromium are described in the following threat models. These include managing extension rights through sandbox boundaries, creating different channels through an Inter-Process Communication (IPC) process, and separating processes for each of Chromium's tabs.

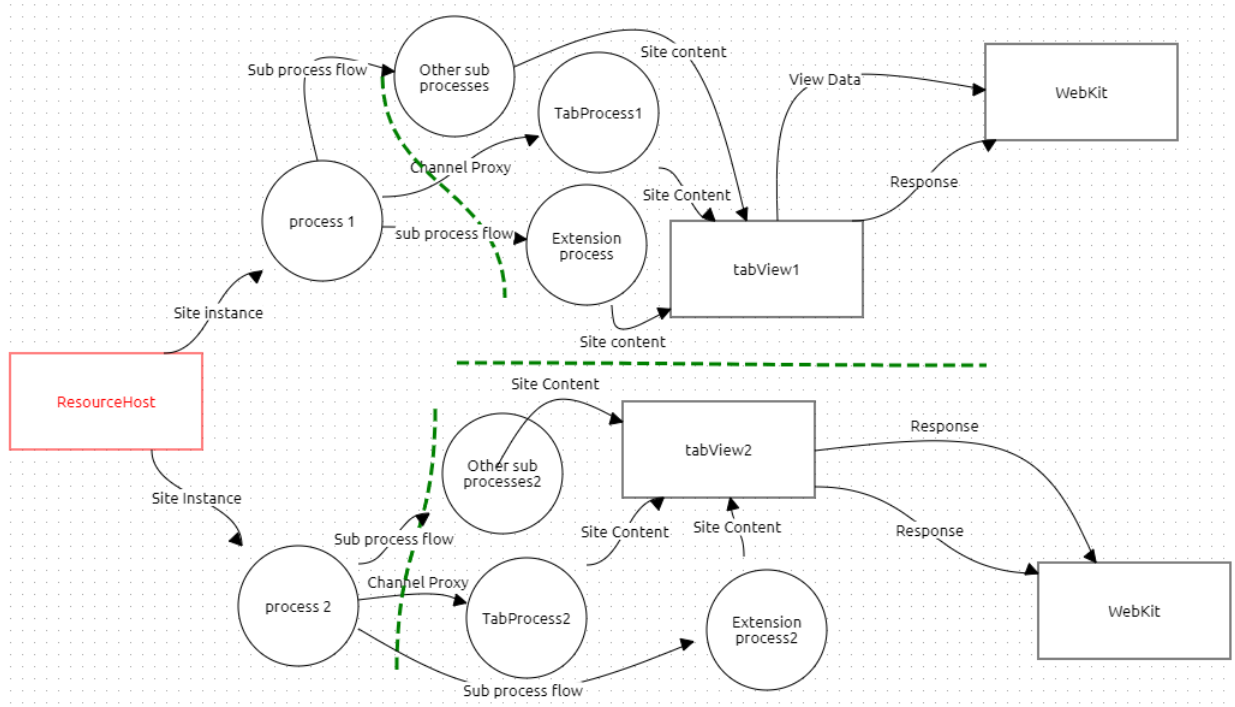

Figure 4 Edge's Multiprocess architecture Threat model

Every process has a unique purpose, and a wide range of factors influence the overall number of processes. So why utilize this multi-process architecture in Microsoft Edge?

Microsoft Edge leverages this design, like other contemporary browsers, to improve security, dependability, and resource accountability into how the browser is using resources. The Chromium project serves as the foundation for Microsoft Edge, and other Chromium browsers all use the same implementation.

Although it can be difficult to explain in simple words how depending on multi-process architecture can promote efficiency, the technique unquestionably increases security and reliability. This is due to the fact that the browser as a whole never functions as a single platform. Instead, the browser is essentially divided into numerous processes, making it incredibly challenging to compromise or simultaneously attack all of the processes. Additionally, if one process isn't functioning properly, the browser won't crash until the issue is fixed.

Assets to Threat Model Tracing

Enterprise security architects must resolve the conflict between productivity and security. Locking down a browser to just allow a select group of reliable websites to load is comparatively simple. Although this strategy will strengthen security overall, it is arguably less effective. Lessening the restrictions raises the risk profile while increasing output. Striking a balance is challenging!

In this constantly shifting risk environment, it is even more difficult to stay on top of new emerging risks. Because their major function is to enable users to access, download, and read untrusted content from untrusted sources, browsers continue to be the main attack surface on client devices. New browser exploits are continually being developed by malicious actors using social engineering. Strategies for security incident prevention, detection, and reaction cannot ensure complete security.

The Assume Breach Methodology, which assumes that an attack will succeed at least once despite efforts to stop it, is an important security method to take into account. This mentality necessitates creating barriers to contain the harm, which ensures that in this circumstance, the corporate network and other resources are kept safe. Application Guard deployment for Microsoft Edge is a perfect fit for this tactic.

Microsoft collaborated with a number of commercial and government customers on a hardware-based isolation strategy to solve these issues and enhance the security provided by purely software-based sandboxes. Customers may browse more securely and worry-free thanks to Microsoft Edge's Application Guard defense against sophisticated assaults that can infiltrate your network and devices via the Internet.

But what if the suspicious website is actually a part of a sinister attack plan? Let's review the aforementioned assault. An innocent employee of the organization receives a carefully designed email from an attacker tempting them to click on a link on a site under the attacker's control. The naïve person clicks on the link to an unreliable website without recognizing anything fishy about the email. Application Guard collaborates with Microsoft Edge to open that site in a temporary and isolated copy of Windows in order to preventatively protect the user and enterprise resources. Even if the attacker's code in this instance is successful in trying to exploit the browser, the attacker will still discover that their code is executing in a clean environment with no access to any user credentials, any useful data, or any other endpoints on the corporate network. The onslaught has been entirely stopped. This temporary container and any malware inside it are removed as soon as the user is finished, whether or

not they are even aware that an attack has occurred. Even a hacked browser instance cannot be used to launch additional attacks on the company's network because there is no way for the attacker to stay active on that local machine. For subsequent browser sessions, a brand-new container is established after deletion.

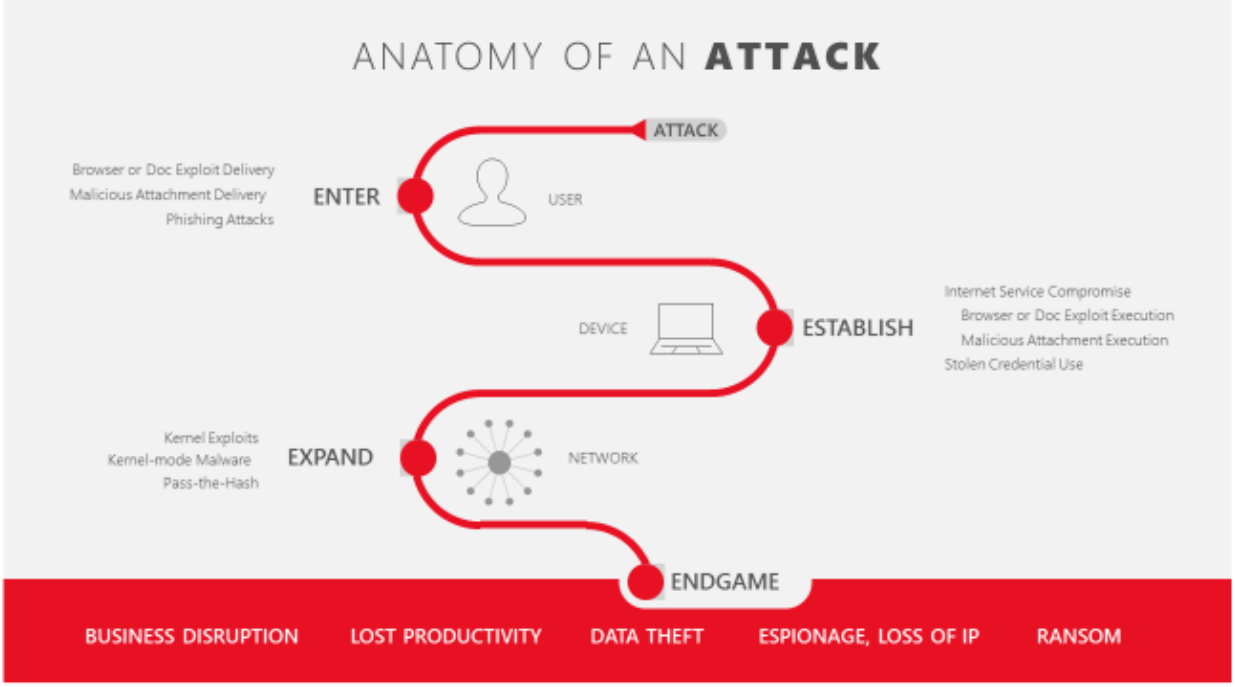

Figure 5 Attack Surface against Large Enterprises

In recent years, there has been a substantial shift in the threat environment. Over 90% of assaults today start with a hyperlink in order to steal credentials, install malware, or exploit security flaws.

This harms tens of thousands, if not millions, of users whose accounts and private information may be taken in addition to the firm that was targeted. Attackers that are highly driven and persistent will frequently begin with a social engineering technique by sending a personalized email to known corporate workers. The employee may be asked to click a link in this email, which frequently appears to be from a credible corporate official, to view an allegedly critical document. Unfortunately, that link leads to a rogue website that has been specifically designed to infect users' computers with malware using a previously unknown vulnerability. Once established on that one computer, the attackers can then take control of it, steal credentials, and use them to probe the rest of the network for other weak points. They can then repeat this process on additional computers until they accomplish their goal, which could be data theft, intellectual property theft, or business disruption.

In addition to the multi-process architecture, threat modeling also needs to interface with the browser kernel, which keeps track of specific permission information for each sandbox, describing what it may (and may not) access. The assets of Chromium are logically arranged within its multi-process, sandboxed design. Security was considered throughout the engineering process. Some crucial elements of this architecture are highlighted in the threat model above. 16 Chromium makes it very challenging for an

exploit to escape its sandbox and get access to or influence other processes through a combination of defense in depth techniques and strict adherence to the concept of least privilege.

Chapter 3: Code Inspection Assessment

Inspection Selection

We selected files with promising content, a history of vulnerabilities, and potential connections to our product assets for the code inspection. The Web Content that is displayed in the web browser and any extensions that are added to that application window are two extremely essential product assets, and the Chakra library was chosen since it connects both. It was decided to use the Chakra JIT Compiler file since it contains private data that could possibly become vulnerable to attack.

After reviewing our selection criteria, we probably would have made a different decision. A list of security issues and promising code inspections were features of the Chakra library and Chakra JIT compiler. It's vital to keep in mind that lengthier files don't always produce any results when selecting files for code inspections. Finding files with a history of vulnerabilities or with any open, persistent concerns is more crucial. Additionally, files that include sensitive data or other product assets must to be taken into account for code inspections.

Code Inspection Results

Chakra

Microsoft created Chakra, a JavaScript engine, in 2008. It has the most comprehensive coverage of ESC2015 features, is dependable, scalable, and performant, and features a special multi-tiered pipeline that supports an interpreter, a multi-tiered background JIT compiler, and a mark and sweep garbage collector that can perform concurrent and partial collections.

The Microsoft Edge browser, Xbox, Windows Phones, and traditional PCs are all currently powered by Chakra. It also powers Azure's Document DB, Cortana, and NodeJS, making NodeJS available on the Windows 10 IoT core.

Severity	**Description**	**Issues/Bugs**	**Fix?**
High	CVE-2019-0539	The NewScObjectNoCtor and InitProto opcodes are viewed as having no side effects, but they fact can have them because to the type handler's SetIsPrototype method, which can result in a change in type. In the JITed code,	The security update addresses the vulnerability by modifying how the Chakra scripting engine handles objects in memory.

		this could result in type confusion.	
High	CVE-2018-8617	If you add a numeric property to an object in Chakra that already has some inlined properties, the object will begin to transition to a new type, using the space previously occupied by some of the inlined properties as pointers to the property slots and the object array that stores the numeric properties. To avoid type confusion, it must kill relevant type symbols when optimizing an InlineArrayPush instruction that may start a transition. But since it doesn't, it might cause type confusion.	The security update addresses the vulnerability by modifying how the Chakra scripting engine handles objects in memory.

Chakra JIT

For greater efficiency, the Chakra just-in-time (JIT) Java compiler creates machine codes for the JavaScript functions that are called repeatedly. A special solution was required to enable ACG in Microsoft Edge since contemporary web browsers depend on Just-In-Time (JIT) compilation of JavaScript to achieve faster performance and the code compilation in JIT is incompatible with ACG: The JIT engine was converted into a separate JIT Process from the Edge Content Process. The JIT Process receives JavaScript bytecode from the Content Process and converts it into machine code before mapping the machine code back to the Content Process.

Severity	**Description**	**Issues/Bugs**	**Fix?**
High	CVE-2018-8298	An Intl.NumberFormat object is initialized using the InitializeNumberFormat function in Intl.js, and an Intl.DateTimeFormat	The security update addresses the vulnerability by modifying how the scripting engine

		object is initialized using InitializeDateTimeFormat. Each initializer comes in two different iterations. The other is for the ICU, and one is for WinGlob. The issue is that the ICU versions don't verify that an object has been initialized before using it. This enables initializing the same object more than once, which may cause type confusion.	handles objects in memory.
High	CVE-2018-8288	Any access to an Intl object's property will start the initialization process, which will launch Intl.js, if the object hasn't already been initialized. It runs Intl.js without regard for the ImplicitCallFlags flag, which is an issue. In the Proof of Concept, it redefines Map.prototype.get to prevent Intl.js from running.	The security update addresses the vulnerability by modifying how the scripting engine handles objects in memory.

The JavaScript bytecode is one element of the Microsoft JIT design that is completely in the hands of the attacker. Even when an attacker already has a read/write memory primitive, bytecode opcodes occasionally implement functionality that could be useful to the attacker. For instance, the main weakness of Microsoft CFG at the moment is that returns aren't protected, therefore all that is required for a successful bypass is overwriting a return address. An attacker often has to be aware of where stack is in order to be able to overwrite a return address on the stack. By including opcodes that may be used to read from and write to the stack, such as, for instance, the ArgOut A bytecode, Chakra bytecode eliminates this necessity.

However, it is not particularly difficult to locate the stack's address in Chakra. In fact, if an attacker possesses a read/write primitive, they can retrieve the stack address by following a chain of pointers from any JavaScript variable. Because this is utilized, for instance, to provide stack walking in JavaScript, it is frequently possible to find stack references on the heap in JavaScript engines (i.e. Function.caller).

The remainder of the section will look at other types of bypasses that would still be present even if the return protection is implemented, as Microsoft has apparently given up on fixing those kinds of bypasses individually and we assume that the long-term plan is to implement return flow protection using Intel CET.

When there is a boundary, all data that crosses it needs to be considered unreliable. It is challenging to make this switch because, prior to the development of JIT server, all of these components were included in a single module, and any data passing between them was presumed to be trustworthy. It is now necessary to evaluate all those areas of the code.

Our results show that this boundary is not currently being properly enforced. We discovered numerous problems including out-of-bound writes and integer overflows(CVE-2017-8637 and CVE-2017-8659) when examining the portions of the first JIT phase (IRBuilderPhase) that are directly exposed to the untrusted data (but so are, to a lesser extent, other phases).

Project-Specific Checklist

Overwriting return address

The JavaScript bytecode is one element of the Microsoft JIT design that is completely in the hands of the attacker. Even when an attacker already has a read/write memory primitive, bytecode opcodes occasionally implement functionality that could be useful to the attacker. For instance, the main weakness of Microsoft CFG at the moment is that returns aren't protected, so all that is required for a successful bypass is overwriting a return address. An attacker typically needs to be aware of where stack is in order to be able to overwrite a return address on the stack. By including opcodes that can be used to read from and write to the stack, such as, for instance, the ArgOut A bytecode, Chakra bytecode eliminates this requirement.

Memory corruption in the JIT Process

A sophisticated piece of software is Chakra. Microsoft has effectively established a security boundary between the JIT components and all other components they communicate with by separating the components responsible for creating the JIT code into a separate process.

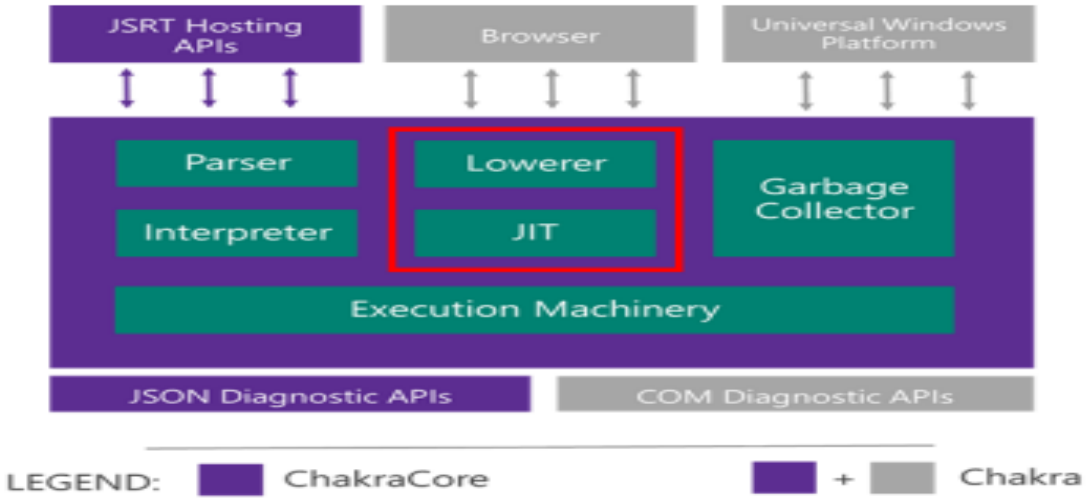

Figure 6 Chakra Architecture with JIT server components

When a boundary is present, all data that crosses it must be regarded as unreliable. This is a challenging change to make because, prior to the development of JIT server, all of these components were included in a single module, and any data passing between them was presumed to be trustworthy. Now, every one of those spots in the code needs to be reevaluated.

The software program Chakra is complicated. Microsoft has effectively established a security boundary between the JIT components and all other components they communicate with by separating the components that are in charge of producing the JIT code into a separate process. This boundary is shown in Image 5 as the red square.

An attacker may be able to take control of the JIT server by exploiting vulnerabilities like this one. This would essentially mean that all Content Processes would be able to access native code and avoid ACG. Instead of fully infiltrating the JIT server, an attacker could also try to write arbitrary data to the JIT Process' memory regions that are mapped as executable in the Content Process.

Exploiting process permissions

The JIT server is required by design to be able to map executable memory in the Content Process, which in Windows entails having a handle to the Content Process with the necessary permissions. This is how it functions, and during the ConnectProcess call, the handle is passed to the JIT Process (used to be in InitializeThreadContext).

The Content Process used to call DuplicateHandle() to create a handle to itself that the JIT Process could access in order to send the handle to the JIT Process. Because DuplicateHandle also needs a handle to the target process with the PROCESS DUP HANDLE access right, doing this in version 17 is problematic. Therefore, Content Process had access to JIT Process through the PROCESS DUP HANDLE handle. Sadly, this access right also has a rather unpleasant side effect that makes duplicate handles possible. As a result, the JIT Process would be totally compromised by the Content Process.

The problem was resolved by moving the Content Process handle to the JIT Process using an undocumented system handle IDL attribute. The Content Process no longer needs to call

DuplicateHandle() or have a handle to the JIT Process because handle passing is now the responsibility of the Windows RPC mechanism.

Vulnerability Trends Over Time

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2015	27	19	19	19	19		2			4	3	1			
2016	133	64	73	69	70		5			8	22	2			
2017	39		21	21	18		1			5	6				
2018	21		14		14					2	1				
2020	4		1	1	1										
2021	26			2						2	1				
Total	250	83	128	112	122		8			21	33	3			
% Of All		33.2	51.2	44.8	48.8	0.0	3.2	0.0	0.0	8.4	13.2	1.2	0.0	0.0	

Figure 7 Microsoft Edge Vulnerability Categories

Microsoft Edge - 'OpenProcess()' ACG Bypass

According to the research, researchers are been able to create an exploit that exemplifies the ACG bypasses in order to show that they are not only theoretically possible but also practically applicable. Because it was the most recent problem to be identified, the memory management logical problem from the previous section was chosen for this exercise.

```
00 KERNELBASE!SetProcessMitigationPolicy
01 MicrosoftEdgeCP!SetProcessDynamicCodePolicy+0xc0
02 MicrosoftEdgeCP!StartContentProcess_Exe+0x164
03 MicrosoftEdgeCP!main+0xfe
04 MicrosoftEdgeCP!_main+0xa6
05 MicrosoftEdgeCP!WinMainCRTStartup+0x1b3
06 KERNEL32!BaseThreadInitThunk+0x14
07 ntdll!RtlUserThreadStart+0x21
```

The exploit presupposes that a hacker already obtained a memory read/write primitive in the Content Process via a different vulnerability. The attacker's intention is to use this ability to create and execute any arbitrary payload. 19 Two variations of the exploit were made by us: The first one uses a return address overwrite technique to circumvent CFG, which is currently possible, under the assumption that it can be done. The second iteration of the exploit bypasses ACG without using a separate CFG bypass because it assumes the CFG is impenetrable. This is to show that even if Microsoft implemented return flow protection and fixed all other known bypasses, the problem would still be exploitable.

Both exploits must first identify the address of a JIT section. In order to create new Thread and Script contexts for it, they first create a Web Worker. This is done to prevent the Web Worker thread from interfering with the JIT server calls made by the main JavaScript thread. The web worker then causes a number of functions to be compiled and records their addresses. It is easy to predict what the next address will be because, at this point, the addresses are typically going to be in increasing order.

UnmapViewOfFile and VirtualAlloc are the next two Windows API functions that must be called by the exploits, as stated in the vulnerability description. The two exploit versions diverge here.

The initial exploit can make use of the return address overwrite to build a ROP chain that calls these two functions while setting the right argument. The ROP device that was used in this is particularly intriguing. Typically, an attacker would use the ROP chain below to call any function with predetermined arguments.

Domain Specific Concerns

- XSS (client and server side): Being a web browser, Chromium is in danger of these attacks. These sorts of attacks can often be used to bypass checks and maybe even authentication.
- DOS

Coding Mistakes

- Overflows: Data shows that this security flaw is often missed in code reviews, but it needs to be checked since most of the system is written in C or C++. This mistake can cause availability problems in the browser.
- Memory Corruption
- Command injection: This vulnerability has been in the OWASP top ten most critical web security risks and is also often missed in Chromium code reviews.

Design Concerns

- Gain privileges
- Gain information
- Bypass permissions

Past Vulnerabilities

Many of the design, domain, and coding errors listed in the above checklist are what lead to the problems in Figure 7. It might be wise to concentrate on the security blunders that lead to the most vulnerabilities if the checklist needed to be prioritized. The handling of exceptions, race conditions, buffer and integer overflows, and other issues are thus important for developers to be aware of. The two primary causes of vulnerabilities could be eliminated by concentrating on these problems: Memory corruption and Code Execution

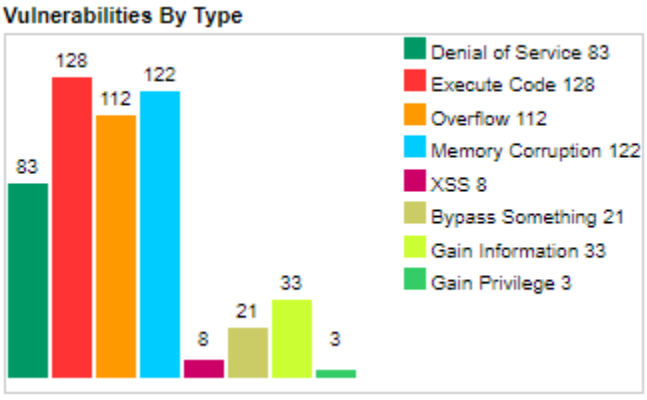

Figure 8 Edge Vulnerabilities Graph

Availability Concerns

Figure 8 illustrates how Code Execution attacks are the main reason for system vulnerabilities. Many of the availability problems should be reduced by dealing with the problems mentioned in Past

Vulnerabilities. Remember that Microsoft Edge does not view minor (local) availability issues as vulnerabilities, so it might not be as crucial for the Microsoft team to address some of these worries.

Inspection Summary

ACG accomplishes its goal of preventing the allocation and modification of executable memory, implementation issues aside. However, due to the interdependence of CFG, ACG, and CIG as well as the limitations of CFG in Microsoft Windows, ACG cannot be relied upon to prevent sophisticated attackers from getting past a browser's sandbox and launching additional attacks. Our examination of the two libraries turned up a few possible issues with the Microsoft Edge project. Therefore, before CFG can truly be an exploitation barrier, Microsoft must commit to fixing all of its known flaws. Whether or not this is possible is still up for debate. The existence of ACG, however, can be seen as a step in the right direction because it can be viewed as a requirement for having useful CFG. Microsoft does a good job of identifying problems, but it is clear that they don't always get around to making the necessary changes.

The JIT server implementation, where numerous issues were found, is the most vulnerable part of the ACG aside from the issues with CFG. The bigger problem is that the security boundary between the Content Process and the JIT Process isn't adequately enforced, even though the implementation is new and the first of its kind, so some issues are expected. We believe that the JIT server needs more development and review work in order to harden it against both memory corruption and logical problems.

Overall, we think that Microsoft Edge's design, as well as its requirements for contributions to the open source community, have a strong process for handling security and vulnerabilities. Edge could do better in their communication after figuring out the causes of vulnerabilities and what fixes are required. The team frequently applies quick fixes that temporarily solve the issue at hand but aren't regarded as the "best fix." When this occurs, programmers frequently include comments in the code describing how a long-lasting solution would look. Unfortunately, many of these suggestions are rarely revisited later and many of them are still unfulfilled.