Strengthening the Microsoft Edge Sandbox

By Microsoft Edge Blog

Published: 2017-03-23 · Archived: 2026-04-05 18:25:51 UTC

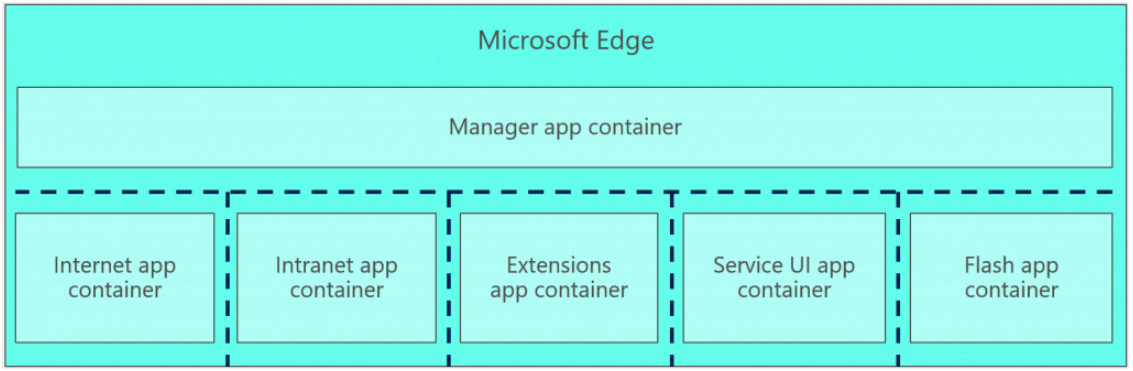
In a recent post, we outlined the layered strategy that the Microsoft Edge security team employs to protect you from vulnerabilities that could be used to compromise your device or personal data. In particular, we showed how Microsoft Edge is leveraging technologies like Code Integrity Guard (CIG) and Arbitrary Code Guard (ACG) to break some of the techniques that hackers rely on when exploiting vulnerabilities to obtain Remote Code Execution (RCE). This is where the attacker seeks to escape from web code (JS and HTML) in the browser to run native CPU code of the attacker's choosing. This lets the attacker violate all of the browser's rules for the web, such as same-origin policy, and so it is important to web users that we try as hard as possible to block RCE attacks.

However, despite our best efforts, sometimes attackers get RCE anyway. In this post, we'll explore some of the significant improvements we've made in the Windows 10 Creators Update to strengthen our next line of defense: the Microsoft Edge sandbox.

The history of the Microsoft Edge sandbox

Because RCE can happen, and in the past, often did, browser vendors have sought to sandbox the browser to defend the rest of the user's PC from attack. In 2007, IE7 introduced Protected Mode, the first web browser sandbox. Windows 8 added app container to the OS, primarily to support the new Windows Store app model and support the confidence promise that apps are safe to use. IE10 and IE11 leveraged app container to create EPM (Enhanced Protected Mode) a stronger browser sandbox. However, the EPM sandbox restrictions were incompatible with older ActiveX controls that predated app container, and so EPM was provided as a security enhancing option.

Microsoft Edge does not support ActiveX, so it is able to run entirely inside app container sandboxes at all times. Since the beginning, Microsoft Edge has used several app containers. At first there was a parent app container for the Manager, which created a small number of additional app containers to host content from the internet separate from intranet content. The Windows 10 Anniversary Update moved Flash into its own, separate AC. Today the Microsoft Edge app container model looks like this:



The Manager provides browser features such as the URL bar, the back button, tabs, and your favorites list. The other app containers are:

- **Internet AC:** hosts content from Internet sites.
- **Intranet AC:** hosts content from Intranet sites. For enterprise users, that is enterprise web sites. For consumers, that is "web sites" that are control interfaces for devices on your home network, such as your Wi-Fi router, or IoT devices. Separating this RAC from the Internet e.g. protects your home Wi-Fi router from Internet attackers.
- **Extensions AC:** hosts the new extensions for Microsoft Edge.
- **Flash AC:** hosts the Adobe Flash player, to isolate it from the main content processes.
- **Service UI AC:** hosts special web pages, such as about:flags, and the default home page.

The Internet AC is where the action is. Its job is to host web pages from anywhere, including the JS code provided by that web page, images, and multimedia. Hosting web pages is extremely complex, due to the richness of the modern web; this is the platform of the Internet, and developers need to be able to create any application and run it in this environment. Because it is complex, and hosts web pages from anywhere, this is where web security attacks begin. A malicious web site presents content intended to exploit bugs in the content hosting system, to take over control of the content process.

If an attacker gains control of an Internet AC process, they need to find some way to achieve their goals. If their goals involve compromising the user's device or personal data stored on the device, then they'll need to contend with escaping from the sandbox first.

Reducing the attack surface of the Microsoft Edge sandbox

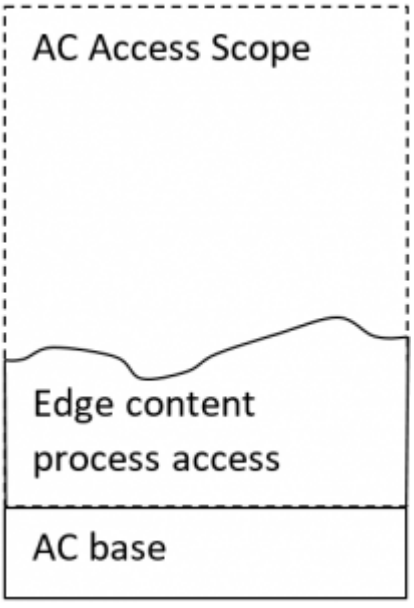
One of the most effective ways to eliminate vulnerabilities in complex applications is to minimize the amount of code that an attacker can try to find vulnerabilities in. This is often referred to as *attack surface reduction* and it is a key tactic in our overall strategy security. To this end, Microsoft Edge in the Creators Update of Windows 10 has significantly reduced the attack surface of the sandbox by configuring the app container to further reduce its privilege.

To understand how we tuned the Microsoft Edge AC, it is first necessary to know how app container itself works. An app container process is *deny-by-default* for any secured object unless the object security descriptor has an allow access control entry (ACE) that would permit the app container to have access. There are three kinds of SIDs that can be used in such ACEs:

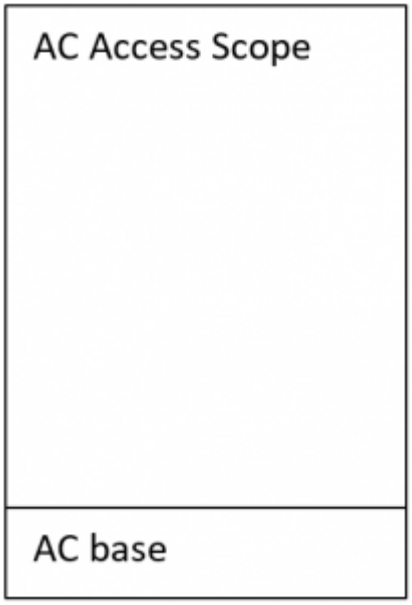
- Capabilities: if a Capability SID based allow ACE is in the security descriptor, and the requesting AC has that Capability SID in its token, then it gets access. E.g. Webcam-related devices are accessible to processes that have the WEBCAM Capability SID such as Microsoft Edge.
- AppID SIDs: if a specific AppID SID based allow ACE is in the security descriptor, and the requesting AC is that specific AppID, then it gets access. The main use for this is per-app storage, which uses the AppID of that app in the security descriptor, ensuring that its storage is private to that app.
- "ALL APPLICATION PACKAGES" ("AC" SID): this is the AC wild card, allowing *all* app containers to access the resource. The "AC" SID exists so that Windows can provide an app platform that is useful to developers, allowing rich UWPs in the Windows Store. For instance, most of the WinRT API surface is open to the "AC" SID, so that all UWP apps can access the WinRT APIs.

The Microsoft Edge AC has been made different in only one way: the "AC" SID is not sufficient to let a requesting process access a resource. To get access, the AC must either have a matching Capability SID, or be precisely one of the named AppIDs in the security descriptor.

Thus the AC loses access to the entire WinRT API set, and every other resource that app containers normally can access. That's great for security, with the *slight* problem that it would break everything about the browser, turning it into a highly secure brick.



Microsoft Edge "Tuned" AC



“Normal” app container

So to make Microsoft Edge work again, we used specific Capabilities to light up specific areas of functionality. For instance, we created a Capability that enables COM to work. We similarly added other new capabilities, granting access to specific resources, until Microsoft Edge worked again. Then we added some telemetry to detect access control failures for the content process and shared it to users in the Windows Insider Program (WIP) to ensure that we had granted access to all resources needed for all scenarios.

What we just did here was to create a *tuned* sandbox for the Microsoft Edge content process, with a much tighter fit to the functional needs of the software than a normal app container provides. That is a lot of work, which is why this is *not* how the UWP platform works. But because web browsers are among some of the most threatened software there is, it is worth it in this case.

We repeated this work of hand-tuning a sandbox for the Flash AC, because it also is subject to a lot of attacks. Placing Flash in a tighter sandbox complements our work in this release to make Flash be click-to-run.

Making it more difficult to exploit sandbox escape vulnerabilities

The main threats in a sandbox are the broker interfaces that provide access to resources according to a defined policy. So why have brokers at all? Brokers exist to grant access according to policy, e.g. the File broker allows a website to say “upload a file by browsing your files” and the *user* gets to pick the file to be uploaded, without giving the web site access to *all* of the user’s files.

Brokers are a threat to sandboxes because brokers are *code* (and therefore can have bugs) and because the brokers run outside the sandbox. If an attacker can hack a broker and run code *in the broker*, then the attacker can escape, similar to an inmate mugging a prison guard and then putting on the guard’s uniform to escape.

The tightened Microsoft Edge content process sandbox cuts off access to many brokers (reduced attack surface) but leaves the needed brokers accessible (residual threat). An attacker might try to exploit bugs in the accessible brokers to elevate privileges and execute arbitrary native code in the broker, and thus escape. To mitigate this

threat, we have applied many of the same exploit mitigation technologies to the brokers that provide service to Microsoft Edge.

Microsoft Edge also has some surprising brokers, because the exploit mitigation work forced several components to move out-of-process (OOP), including the Chakra JIT code generator and the Flash runtime process. These components had to move out of process because they necessarily do code generation (JIT) and that is not compatible with the ACG mitigation applied to the content process. Thus, these components function as the "JS code generator" broker and the "Run Flash bytecode" broker.

They can be viewed as brokers precisely because they have permission to do something that the content process is not permitted to do for itself. They are also a threat, because an attacker that can gain control of OOP JIT or the Flash sandbox could e.g. create executable code and use that for further exploitation, bypassing ACG. Because they are a threat, we have also have put these processes into tuned, less privileged app containers, again with a custom-crafted container profile built from capabilities.

Impact to attackers

Making the Microsoft Edge content process run in this tighter sandbox reduces the sandbox attack surface, but by how much? As it turns out, the reduction is quite significant:

- 100% reduction access to MUTEXes: allow a process to lock up a resource, causing hangs.
- 90% reduction in access to WinRT and DCOM APIs: this is the **large** win here, dramatically reducing Microsoft Edge's attack surface against the WinRT API set.
- 70% reduction access to events and symlinks: symlinks are especially interesting, because they are often used in creative bait & switch attacks to escape sandboxes.
- 40% reduction in access to devices: Windows supports many device drivers, and their quality is somewhat beyond Microsoft's control. The tuned sandbox cuts off access to any device that Microsoft Edge does not explicitly need, preventing attackers from using vulnerabilities in device drivers to escape, or from abusing the devices.

While attack surface reduction does not guarantee that an attacker cannot escape the sandbox, it does dramatically reduce the opportunities for attack, much like reducing the number of windows and doors in a fortress. In addition, the enabling of additional exploit mitigations for the brokers that Microsoft Edge is legitimately able to access also increases the difficulty of exploiting vulnerabilities that remain.

Conclusion

Security is a process, not a destination, so we will continue to invest in both RCE and sandbox mitigations for Microsoft Edge. These exploit mitigations combined with the strengthened sandboxing should make Microsoft Edge significantly more work for attackers to exploit, and thus discourage attackers from trying in the first place.

— Crispin Cowan, Senior Program Manager, Microsoft Edge
