# Browser isolation as an enterprise security control

Received (in revised form): 1st August, 2022

**Henry Harrison**
Co-founder and Chief Scientist, Garrison, UK

Henry Harrison is co-founder and chief scientist at Garrison, developers of a hardware-based browser isolation platform which is supplied both on-premises and as a cloud-based service to mainstream enterprise, and is also supplied as a cross-domain solution to government customers in the Five Eyes and allied nations. Prior to founding Garrison, Henry was Technical Director for Cyber Security at BAE Systems following BAE's acquisition of Detica plc. Henry has a physics degree from Oxford University and an MSc in electronic engineering from Surrey University and holds multiple patents for security technologies.

Garrison, 117 Waterloo Road, London, SE1 8UL, UK
Tel: +44 (0)7736 675484; E-mail: henry.harrison@garrison.com

**Abstract** Browser isolation is a category of security control that allows users of sensitive endpoint devices to access potentially risky web content without putting their devices at risk of compromise by malware. A key use case is to provide web access from the privileged access workstations that should be used by those with elevated system privileges such as systems administrators. If endpoints for such users are compromised, then the attacker may gain the 'keys to the kingdom', making the risk of direct access to unknown and untrusted websites too high. Browser isolation, however, may also be used as a control to protect endpoints for broader classes of users to prevent attacks such as phishing e-mails containing malicious uniform resource locators (URLs). In order to form a useful control, browser isolation must deliver a significant 'step up' in security compared to the extensive web security already typically deployed within the enterprise, both in third-party security products such as proxies and endpoint agents, and within existing browser software such as Google Chrome. The Browser Isolation security model depends critically on the data transfer format between an untrusted component responsible for processing risky web content and a trusted component responsible for transmitting information to the user's endpoint. The gold standard in this area is a technique known as 'pixel pushing', whereby risky web content is transformed into raw pixels. Beyond today's implementations, browser isolation may likely play a broader role in future, in keeping with the role that equivalent technologies already play within the military and intelligence sectors, as referenced by a recent White House memorandum.

KEYWORDS: browser isolation, web security, privileged access workstations, pixel pushing, phishing, ransomware, cross-domain solutions

## INTRODUCTION

In the late 1990s, Microsoft's ActiveX technology[1] invited arbitrary websites to send code to be run natively on Microsoft Windows® endpoints. The operating system's (OS) security model was intended to ensure that malicious ActiveX objects could not gain access to sensitive data and software: in practice, of course, inadequacies and vulnerabilities in the OS meant that the scope for malicious attack was high (as predicted by observers at the time[2]).

ActiveX did not long prosper as an Internet technology.

The sandboxing techniques used by modern web browsers are far superior to the ActiveX security model.[3] And yet, as probably the single most complicated piece of software installed on modern corporate endpoint devices, it is no surprise that browsers also exhibit vulnerabilities of varying criticality. See for example CVE-2020-6572[4] for a vulnerability that permitted an attacker to escape a Chrome sandbox and which was reported exploited in the wild.

Within the enterprise, the web browser is the user interface of choice for almost every business activity, providing access and control over critical data and services. But as well as providing access to trusted systems and cloud services, the browser continues its original role as the window onto the World Wide Web. In most enterprises today, the user can invite arbitrary websites to send complex content for parsing and execution on their endpoint. In most cases, neither the user nor the enterprise has any real knowledge about the website owner or about their security practices: they may have malicious intent, or the site may be used or compromised by other parties who have malicious intent.

This is hardly news. A thriving web security industry[5] exists to try and counter the risk that this browser usage presents. Identification and blocking of malicious uniform resource locators (URLs), on-the-fly identification of malicious content and OS-level endpoint protection tools all aim to prevent, detect or mitigate potential website-launched attacks. And for the most part, over the past decades, they have succeeded.

For the most part — but of course, not wholly. Sophisticated web-based attacks remain a significant threat from those whose intent may be to install ransomware, to conduct espionage or even to carry out disruption and destruction. For some organisations, the residual risk even in the presence of proxies, threat intelligence and endpoint protection agents remains too high — at least for those users whose endpoints are used to access the most critical systems and data.

The archetypal example is the systems administrator with elevated privileges. If their endpoint were to be compromised, the attacker gets the keys to the kingdom: immediate access to all data and all systems. Best practice has long advised that endpoints with privileged access should not have broad-based access to the World Wide Web. Yet anyone who has observed the work of systems administrators will have noticed that broad-based web access is central to their work: Google is probably the single most important tool for today's sysadmin who must hunt down the nuggets of information that will help them identify the fix or workaround that they need. So how can best practice be achieved?

The historic answer has been the use of jump boxes,[6] requiring systems administrators to remotely access a locked-down, non-web-connected virtual desktop in order to carry out privileged tasks. This can provide a bump in the road; but with techniques such as man-in-the-browser, the bare fact is that — with a little effort — anything the legitimate endpoint user can do, the attacker can too. That includes the use of a jump box.

What are the alternatives? The simplest is, of course, to require such users to use two physical devices: one to perform privileged tasks, and the other to access potentially risky web-based content.[7] Apart from the inevitable user push-back this entails, there are two real practical issues with this approach. First, in many cases the result of a Google search will be to unearth a particular set of potentially complex commands that need to be tried. For practicality, it is important that the user can copy and paste these commands from the website onto the command line. Secondly, in other cases, rather than Googling, the user needs to follow a link provided from a trusted source: they need the ability to click through, rather

than having to retype a potentially long and complicated URL.

The promise of browser isolation is to provide a better solution — a way to access potentially risky websites from a highly sensitive endpoint device that enables common workflows such as click-through and copy-and-paste, while providing an equivalent level of protection to the use of a physically separate device.

One might reasonably ask: is this not exactly what all web security vendors have been promising for over 20 years? What makes browser isolation different from the myriad security tools that have been developed and promoted over that period? These are questions which are not only reasonable but essential: security professionals have learned time after time that glossy marketing promises rarely translate to robust risk mitigation. We must unpick the promise and understand what lies behind it.

At the heart of the browser isolation promise is a web security model that does not rely on detection. Rather than seek to detect malicious content, the isolation model assumes that content may be malicious unless there is good reason to believe otherwise. With a detection model (the default for historic web security tools), the response to detection of malicious content is simply to block it. This is not a useful solution if the vast majority of content is to be assumed potentially malicious, and it is precisely the role of browser Isolation to provide users with safe access to this potentially malicious content.

This is of course the same approach that browsers themselves have incorporated since those far-off times of ActiveX. Website content is sandboxed in an effort to ensure that, even if it is bad, the attacker cannot gain access to other tabs within the browser, or to the OS. But browsers are just software applications like any other, and it is precisely the presence of vulnerabilities in this sort of sandboxing technique that gives rise to the promise of browser isolation. That promise must then not only be to provide safe access to potentially malicious content, but to do this with a substantially higher level of security than is provided by the browser software itself.

The first part of the browser isolation solution, then, is to parse and execute the potentially malicious content on a different physical machine. This is a well-understood concept: indeed, it is essentially time-honoured remote desktop under a new guise. If we leave it at that, however, we ignore the fact that remote desktop technologies were not developed primarily as security controls — their introduction and development have been driven predominantly by productivity requirements.

If, instead, we look at browser isolation (or remote desktop) technologies from first principles, we see that their primary job is transformation: they receive, from some remote system, data that causes them to create some sequence of screen images and then send, to the user's physical endpoint, data that causes that endpoint to create some sequence of screen images (see Figure 1).

One seemingly absurd approach is simply to send the same data to the endpoint that they receive from the remote system: yet in some cases, precisely this approach is used. For example, for performance reasons, most remote desktop systems include a 'media acceleration' option whereby video data is not rendered and then re-encoded, but rather 'passed through' in its native form (see for example 'Multimedia Redirection for Azure Virtual Desktop'[8]). A 'pass through' approach is not useful for a browser isolation platform; clearly, some element of transformation must be applied to the data.

The fundamental requirement is this: the platform must ensure that the data delivered to the physical endpoint is safe even if the data that is received from the remote system is not. Furthermore, we must assume that the software system that processes the data from the remote system — for example, the data received from a remote website — has been

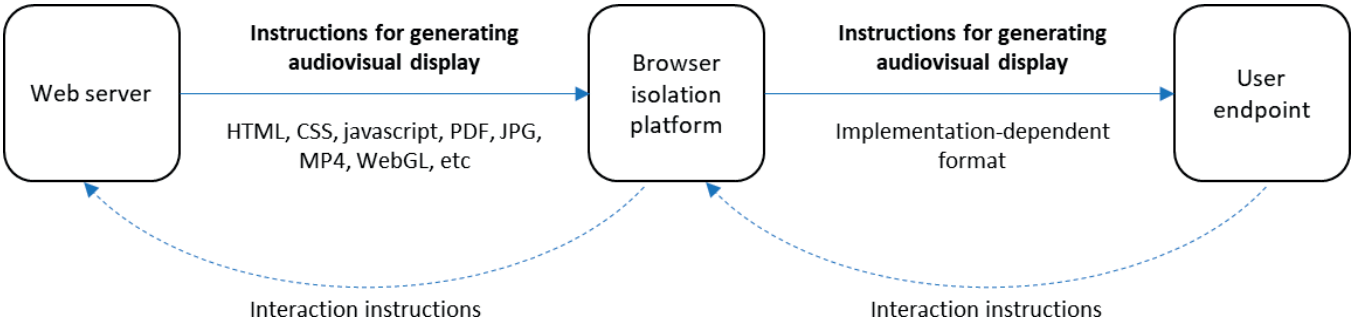


**Figure 1:** First principles of browser isolation technologies

compromised by malware. If we believed we had techniques that could prevent this, we would of course simply deploy those techniques to our users' endpoints and avoid the need for browser isolation altogether. The browser isolation platform must therefore consist of at least two separate systems: system A, which processes remote data (and must at all times be assumed compromised by malware); and a separate system B, which must remain trusted at all times, and which will send a safe data stream to the user's endpoint. Inevitably, of course, there exist browser isolation implementations that do not contain a separate system A and system B, and where the data stream sent to the user's endpoint is generated by system A: but since system A must be considered compromised, this can lead to the generation of a malicious data stream for delivery to the user's endpoint. We will ignore such flawed architectures.

The focus must therefore be on the format of the data that is transferred from the first (assumed compromised) system A to the second (unimpeachably trusted) system (see Figure 2). This data format must provide three things. First, it must not be possible to use this data format as a vector for compromising system B. Secondly, it must allow the second system to generate a data stream which itself cannot possibly be used to compromise the user's endpoint. And thirdly, it must faithfully represent the visual output of system A (for example, a web page).

The gold standard for this transfer format is raw pixels — an approach commonly known as 'pixel pushing'. With pixel pushing, system A delivers to system B merely a stream of pixels representing its visual output. Raw pixels present a unique data format for visual data because there is no such thing as invalid pixel data. A 1080p raw 24-bit RGB bitmap (for example) is a buffer in memory containing 3x1920x1080 bytes: any data written into that memory buffer represents a valid image and can be displayed

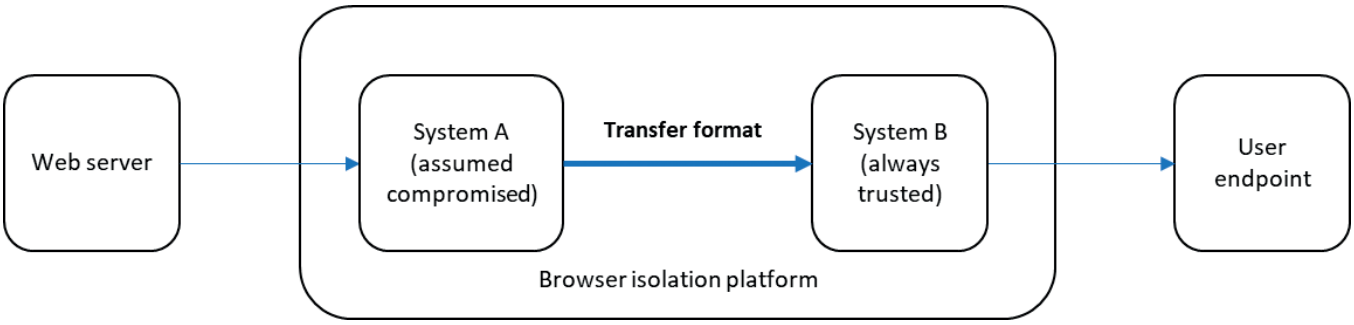


**Figure 2:** Transfer of data from compromised system to trusted system

onto a screen with a simple memory copy (or at worst, a trivial algorithm for colour space conversion). This is quite unlike other visual formats — jpeg, pdf, html — which require sophisticated parsing that may potentially contain vulnerabilities that could be exploited by a carefully crafted data stream. We should note briefly that exactly the same argument can be used for sound representation as raw pulse-code modulation [PCM] audio:[9] again, absolutely any data in the sound buffer constitutes a valid (if potentially unpleasant) sound.

The challenge with pixel pushing is the sheer volume of data generated. At 30 frames per second (for example) a 1080p red, green, blue (RGB) screen image will generate 3x1920x1080x30 bytes — or a data rate of 1.5Gbit/s. It is clearly not feasible to deliver that 1.5Gbit/s to the user's endpoint without either very substantial data compression or excessive network utilisation. The good news is that suitable compression algorithms are well-known: this is a video compression problem, and technologies like H264 are well-established. The bad news is that cost-effective implementation of these techniques requires dedicated hardware, just as the compression of ubiquitous mobile phone videos requires dedicated H264 hardware within the mobile phone chipset.

For those seeking to deliver browser isolation platforms on commodity clouds using software only, the cost of video compression is excessive. That means an alternative to pixel pushing needs to be found — a variety of techniques collectively known as 'transcoding'.[10] Under a transcoding approach, some other data format is sent by system A to system B, this data format being inevitably more complex than raw pixels in order to reduce the bitrate and hence the need for data compression. Many such 'transcoding' data formats are proprietary and unpublished, and hence it is hard to judge the extent to which they might be used as a vector to compromise system B, or alternatively to 'smuggle' an attack through system B in order to directly compromise the user's endpoint. A well-known 'transcoding' format — originating from remote desktop technologies — is remote desktop protocol (RDP). Certainly this is a format which has historically given rise to multiple vulnerabilities, with associated exploits — for example, a 2022 Remote Code Execution Vulnerability.[11]

Use of a robust browser isolation technology can allow users to interact with even the riskiest websites from even the most sensitive endpoints. The click-through workflow is easily solved: just as the remote system (System A) must be instructed about mouse movements and key presses in order to enable scrolling and interaction, so too the remote system can be instructed what URL to browse. The copy-and-paste workflow requires more work, because this is not simply the transfer of pixels but the transfer of other data to be inserted into a clipboard. Suitable content security techniques need to be deployed, integrated with the browsing user experience. It may be sensible to restrict the types of data handled to simple formats that can be rapidly sanitised or verified, since more complex data types may require content security techniques which take longer (for example, a sandbox) and which will not integrate well with a copy-and-paste workflow. Further workflow integrations (for example, with potentially longer-duration content security for file transfer) can be added.

Of course, while they are in all cases probably the riskiest users, it is not just systems administrators who are at risk of web-based attacks. Many organisations are particularly concerned about risks arising from phishing attacks against a wide range of users, with e-mails containing links to malicious websites that might seek to install ransomware or other forms of malware. And it is not only systems administrators who have access to critical systems and data: a wide range of roles in most organisations involve working with at least one form of

sensitive information. Browser isolation can naturally be deployed as a control for this broader user base, either using the same model as described for systems administrators (definition of an allow list of trusted sites, with all others accessed using browser isolation) or by defining other business rules for discriminating between 'more trusted' links that can be accessed natively and 'less trusted' links that must be accessed using browser isolation. The largest UK retail bank, Lloyds Banking Group, provides a case study for such a deployment.[12]

In either case, the deployment requires some line to be drawn between those parts of the Web which are more trusted and those which are less trusted. One might be drawn to ask: why not use browser isolation to access all links? For most organisations, such an approach would make little sense, because the browser provides the interface to highly trusted cloud services that provide critical business services and are used to hold and access sensitive confidential data. The function of browser isolation is to provide an extremely strong barrier between those things that are accessed using browser isolation and those that are not: the barriers between different things accessed using browser isolation (for example, between tabs) will typically be weaker, relying on the same sorts of isolation technology (for example, sandboxes and containers) which are already present in the native desktop browser. So those highly trusted cloud services — with their sensitive confidential data — should reside on one side of that strong barrier, and arbitrary unknown websites on the other, in order to prevent attackers using the latter as a basis for getting access to the former.

A more complex question, however, is to ask whether one can really categorise all of the digital world into only two categories: more trusted, and less trusted. Surely there is a vast spectrum of digital content and services meeting a wide variety of trust criteria? How can this reality fit with a simplistic binary division? In principle, it might be attractive to work with an infinite spectrum of trust, but this runs up against both technological and human boundaries.

First, in many cases it is not desirable to maintain very strong isolation between different websites: to give one example, while the use of third-party cookies is not always popular, it is a critical component of delivering the contemporary web experience. Strong isolation breaks that mechanism.

Secondly, the concept of trust is inherently human: users need to understand what trust environment they are working in. Is this a trusted site where I can safely enter sensitive customer data? Is this a site on whose information I can reasonably base business decisions? Experience suggests that humans are not good at dealing with infinitely subtle gradations of trust: it is necessary to work with broad brushstrokes and a limited number of categories of trust.

That experience has been gained principally in areas that have long depended on strong isolation — in particular, military and intelligence organisations. Where today's commercial and civilian deployments of browser isolation might revolve around 'more trusted' and 'less trusted', the world of national security has long worked with a broader set of trust boundaries. A member of US military personnel might, for example, work routinely with US Secret, Five Eyes Secret, NATO Secret, Unclassified and 'high threat' environments, each representing a different trust boundary. (Similarly, their UK counterpart would work with UK Secret, Five Eyes Secret, NATO Secret, etc.) Of course, within NATO Secret there are fine gradations between systems, most likely with trust levels that vary over time with the geopolitical situation and levels of third-party capability. But from both a technological and human perspective, defining broad-based categories provides a framework for both sensible system implementation and practical human comprehension and decision making.

Browser isolation technology is routinely deployed within these governmental

environments, not only to provide access to the World Wide Web but to provide access, for example, from UK Secret to NATO Secret. In this context, however, different terminology is typically used: rather than browser isolation, governmental users typically talk about 'cross-domain solutions'[13,14] or 'browse down'.[15] That terminology is often poorly adapted to comprehension in the wider world, as with the section on cross-domain solutions in the US White House's January 2022 'Memorandum on Improving the Cybersecurity of National Security, Department of Defense, and Intelligence Community Systems'.[16] Nonetheless, one important area of what that world calls cross-domain solutions is precisely the sort of browser isolation technology that has been described in this paper.

Perhaps over time commercial and civilian organisations will evolve a similar approach to trust categories, with browser isolation technology deployed in a more subtle form than today's 'more trusted' versus 'less trusted' approach. But while we can accept that this may be a crude instrument, it already represents a significant change for enterprises which have historically taken an approach of 'default trusted unless found to be malicious'.

## References


1. Wikipedia, 'ActiveX', available at https://en.wikipedia.org/wiki/ActiveX (accessed 1st August, 2022).
2. Garfinkel, S. (November 1996), 'Will ActiveX Threaten National Security?', Wired, available at https://www.wired.com/1996/11/will-activex-threaten-national-security/ (accessed 1st August, 2022).
3. Sylvain, N. (October 2008), 'A new approach to browser security: The Google Chrome Sandbox', Chromium, available at https://blog.chromium.org/2008/10/new-approach-to-browser-security-google.html (accessed 1st August, 2022).
4. Hawkes, B. (April 2020), 'CVE-2020-6572: Chrome MediaCodecAudioDecoder Sandbox Escape', 0-Days In The Wild, available at https://googleprojectzero.github.io/0days-in-the-wild//0day-RCAs/2020/CVE-2020-6572.html (accessed 1st August, 2022).
5. Future Market Insights, 'Corporate Web Security Market Overview (2022–2032)', available at https://www.futuremarketinsights.com/reports/corporate-web-security-market (accessed 1st August, 2022).
6. Wikipedia, 'Jump server', available at https://en.wikipedia.org/wiki/Jump_server (accessed 1st August, 2022).
7. Microsoft (January 2022), 'Securing devices as part of the privileged access story', available at https://docs.microsoft.com/en-us/security/compass/privileged-access-devices (accessed 1st August, 2022).
8. Microsoft, 'Multimedia redirection for Azure Virtual Desktop (preview)', available at https://docs.microsoft.com/en-us/azure/virtual-desktop/multimedia-redirection (accessed 1st August, 2022).
9. Wikipedia, 'Pulse-code modulation', available at https://en.wikipedia.org/wiki/Pulse-code_modulation (accessed 1st August, 2022).
10. Clyde, R. (November 2022), 'Non-Porous Web Isolation', LinkedIn, available at https://www.linkedin.com/pulse/non-porous-web-isolation-rob-clyde/ (accessed 1st August, 2022).
11. CVE, 'CVE-2022-23285', available at https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-23285 (accessed 1st August, 2022).
12. Garrison, 'Lloyds Banking Group', available at https://www.garrison.com/lloyds-banking-group (accessed 1st August, 2022).
13. National Cyber Security Centre (January 2022), 'Security principles for cross domain solutions', available at https://www.ncsc.gov.uk/collection/cross-domain-solutions (accessed 1st August, 2022).
14. Australian Cyber Security Centre (October 2021), 'Fundamentals of Cross Domain Solutions', available at https://www.cyber.gov.au/acsc/view-all-content/publications/fundamentals-cross-domain-solutions (accessed 1st August, 2022).
15. National Cyber Security Centre (May 2019), 'Security architecture anti-patterns', available at https://www.ncsc.gov.uk/whitepaper/security-architecture-anti-patterns#section_3 (accessed 1st August, 2022).
16. The White House (January 2022), 'Memorandum on Improving the Cybersecurity of National Security, Department of Defense, and Intelligence Community Systems', available at https://www.whitehouse.gov/briefing-room/presidential-actions/2022/01/19/memorandum-on-improving-the-cybersecurity-of-national-security-department-of-defense-and-intelligence-community-systems/ (accessed 1st August, 2022).