Menlo Security

WHITE PAPER

Menlo Security Isolation Platform

Adaptive Clientless Rendering

Browser isolation promises to safeguard users from future zero-day threats by running active content away from the endpoint.

Introduction

Web browsers are among the most important applications in our business lives, yet they are also the most vulnerable to attack. The simple act of loading a malicious web page suffices to compromise the user's endpoint, leading to malware installation, data theft, and penetration of corporate networks. Unfortunately, an ever-increasing set of browser features ensures that attackers will continue to have an unlimited supply of vulnerabilities to exploit.

A critical ingredient in today's browser exploits is active content. In the modern web, active content comes in two predominant forms: Flash and JavaScript. Regardless of form, active content executes in the context of the user's browser and enables significant attacker control and visibility into the browser's workings and vulnerabilities. For instance, active content enables the attacker to discern memory locations (address space disclosure), influence data layout (heap spray), and dictate code generation (JIT spray)—all of which are key techniques in crafting a successful exploit.

Modern endpoints have built-in defenses against simple browser exploits, but active content execution enables determined adversaries to bypass these defenses with sophisticated, multi-stage attacks. In particular, two pervasive defenses—Data Execution Prevention (DEP/NX) and Address Space Layout Randomization (ASLR)—thwart simple code injection and Return-Oriented Programming (ROP) exploits, respectively. However, with the aid of active content, an exploit can bypass both DEP and ASLR, typically by triggering a secondary vulnerability—one that, for instance, reveals the memory location of native code. The exploit can then use that code to craft ROP code sequences that execute the attacker's bidding.

Browser Isolation Is the Future

Browser isolation is an emerging technology that offers a solution to the security challenge posed by executing active content on the endpoint. It centers around the notion of an isolated browser—a web browser that loads and runs pages, including all embedded active content, inside a contained environment with the goal of isolating any potential browser infection away from the endpoint. In most incarnations, the isolated browser and the endpoint are separated by a secure channel using a minimal, highly restrictive protocol designed to carry rendering updates to the endpoint and user input to the isolated browser, and nothing else. This “air-gapped” browsing mode enables browser isolation to defend against today’s sophisticated zero-day exploits. Specifically, by running untrusted active content on the isolated browser and preventing it from probing and exfiltrating data from the endpoint, browser isolation precludes exploitation of secondary vulnerabilities essential to bypassing standard endpoint defenses.

The Browser Isolation Challenge: Making It Practical

Despite its compelling security benefits, browser isolation must meet IT and end-user needs if it is to become a widely adopted security technology. To that end, we have identified two key requirements for a practical browser isolation solution.

The first requirement, **clientless deployment**, reduces IT burden by altogether avoiding the need for endpoint software installs, and with it, the risk of destabilizing the endpoint. The clientless nature also enables easy enterprisewide deployment via in-network proxy configuration, as well as fully centralized management of browsing policies and security updates across all devices—including personal devices—within the enterprise network.

Equally important, a **native user experience**, in which users do not perceive a difference from a native experience, is critical for ensuring end-user productivity and buy-in. In particular, users should not have to alter the way they browse the web or be distracted by changes to their browser’s behavior. Moreover, rendering speed and quality should be identical to native for a broad range of media types (text and video), and day-to-day operations such as printing and copy-paste should just work.

25%

Through 2022:
25% of organizations will adopt browser isolation.
—Gartner 2018 SWG Magic Quadrant

Meeting both requirements means solving the challenging problem of transparently remoting the isolated browser’s rendered output to the existing endpoint browser without requiring additional endpoint modifications (no agents or special plug-ins). Unfortunately, the traditional and most prevalent remoting technique—pixel mirroring—falls short, mainly because it treats the isolated web page as a bag of pixels to be mirrored with a client that has little understanding of what those pixels represent. The result is a one-size-fits-all approach that not only precludes adapting the remoting technique to the kind of content being displayed (text vs. video), but also slows down page load time and responsiveness by eliminating opportunities to harness the browser’s hardware-accelerated rendering features, and hinders everyday workflow operations such as printing and copy-paste.

Recognizing the challenges posed by pixel-mirroring technology, several browser isolation solutions have traded off the clientless form factor for specialized endpoint browsers, plug-ins, and virtualization. While these trade-offs are acceptable in some environments, the resulting IT burden caused by trouble tickets and endpoint disruption has proved to be a significant barrier to broader deployment.

Adaptive Clientless Rendering

Menlo Security's patented Adaptive Clientless Rendering™ (ACR) is the core technology used in the Menlo Security Isolation Platform (MSIP). In a clear departure from traditional pixel-remoting technology, ACR combines a web-based delivery vehicle with a greater understanding of the isolated page to simultaneously enable clientless deployment and a native user experience.

Clientless Architecture

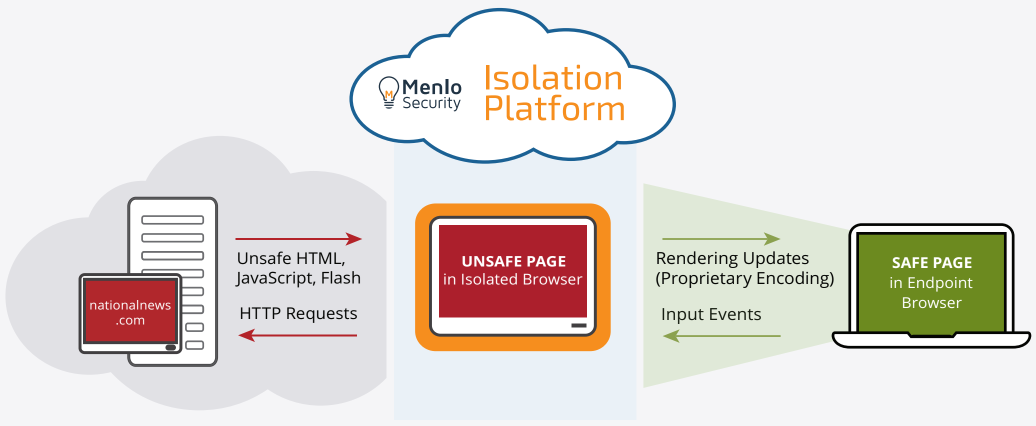
Menlo Security
Isolation Platform
nationalnews .com
Unsafe HTML, JavaScript, Flash
HTTP Requests
UNSAFE PAGE in Isolated Browser
Rendering Updates (Proprietary Encoding)
Input Events
SAFE PAGE in Endpoint Browser

Figure 1: The ACR clientless architecture. The existing endpoint browser loads a safe, transcoded version of the original page (Safe Page) that interprets rendering updates coming from the isolated browser and relays input events back to it—all over a secure HTTPS channel.

Depicted in Figure 1, the ACR architecture involves two major components: the Safe Page and the isolated browser.

The Safe Page is a safe, transcoded version of the target web page that is loaded by the existing endpoint browser in lieu of the original page. Served via a secure web proxy, the Safe Page establishes an SSL-encrypted communication channel to a freshly allocated isolated browser upon loading, and then applies rendering updates coming from the isolated browser and relays user inputs back to it. The Safe Page capitalizes on rapidly converging web standards and advances in browser engines to work accurately and efficiently regardless of which major browser or device is used on the endpoint.

Running on the MSIP, the isolated browser loads web pages on the endpoint's behalf. It sends rendering updates to the Safe Page in response to dynamic page changes and injects user inputs coming from the Safe Page. Based on an up-to-date version of the Chromium browser engine, the isolated browser inherits its security, stability, and feature set. However, since no browser engine is immune to infection, the MSIP operates under the assumption that its isolated browsers will eventually be infected as well. Thus, as a key step in securing the isolation platform as well as end users, the MSIP employs frequent disposal of isolated browsers along with multi-level container isolation to avoid both persistent and lateral infection.

Transparent Remoting Technology

The key to ACR's transparent user experience is a patented remoting technique we term Document Object Model (DOM) Mirroring. The DOM is the dynamic, browser-internal representation of the user-visible rendered result. Intuitively, the goal of DOM Mirroring is to mirror only the benign portions of the isolated browser's DOM tree on the endpoint browser.

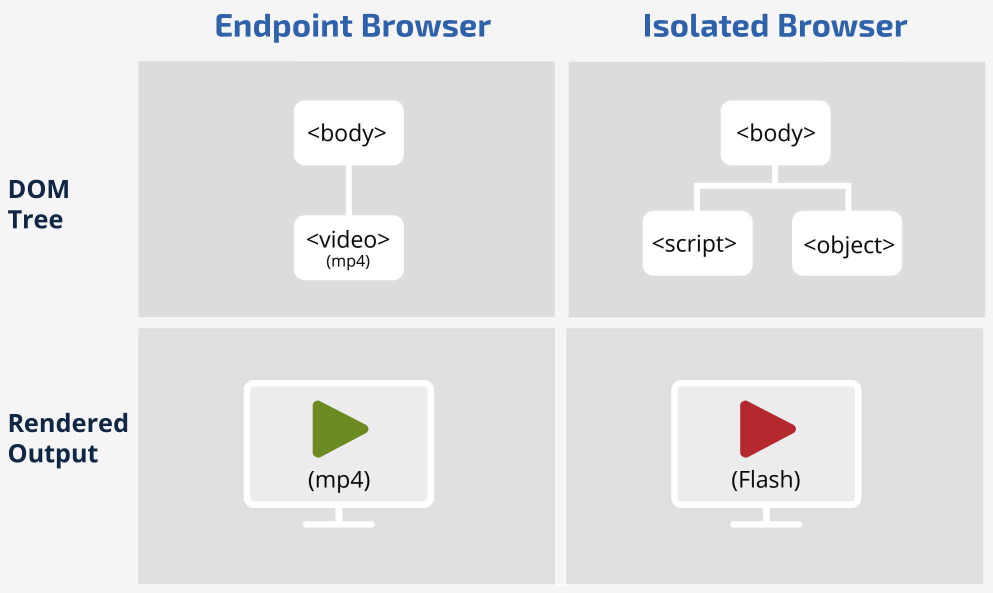

Figure 2: Isolated browser architecture. DOM Mirroring in operation on a sample web page.

To reflect DOM changes on the client, each isolated browser tab actively monitors the currently loaded page's DOM tree for changes, packages these changes into DOM commands sans active content, and sends them to the Safe Page for application. Once received, the Safe Page applies these updates to its local DOM using the standard DOM API available in all browsers.

Adaptive Transcoding

DOM Mirroring confers distinct advantages over pixel-mirroring approaches, primarily owing to the selective exposure of DOM elements to the client. A key benefit is that it enables the selection of remoting strategy at DOM element granularity, whereby nonactive safe elements are left as is and active unsafe elements are either dropped altogether or are replaced with a safe, transcoded variant that is best suited for the element's media type.

For instance, ACR always drops <script> elements but transcodes CSS to a layout-preserving form, free of active content. Of special importance is Flash content, which often takes the form of video but can include interactive elements as well (e.g., video player UI). In order to preserve the native user experience, ACR dynamically detects the level of user interaction and adapts its Flash transcoding method accordingly: Low-interactive, high-frame -rate video content is transcoded into a high-definition video that is streamed to the client while highly interactive Flash elements are remoted in real time for optimal responsiveness.

DOM Mirroring's semantically aware rendering also enables the client browser to apply natively available fonts and UI widgets to the final rendered result for a truly native look and feel, regardless of endpoint browser or platform.

Rendering and Workflow Offloading

Central to providing a truly transparent user experience, DOM Mirroring enables CSS reflow, render-tree computation, and graphics compositing to be performed on the endpoint browser as it would be natively, resulting in visible benefits such as fast page loads, smooth scrolling and animations, and crisp, high-quality HTML5 video playback. DOM Mirroring's semantically aware rendering also enables the client browser to apply natively available fonts and UI widgets to the final rendered result for a truly native look and feel, regardless of endpoint browser or platform. A pixel-based approach, by contrast, is limited to the fonts and UI widgets on the isolated browser.

Finally, DOM Mirroring avoids disruption to workflow operations such as copy-paste, find-replace, and printing. Copy-paste, for example, is difficult to emulate in a truly native fashion using pixel mirroring because of browser-enforced security restrictions on asynchronous clipboard manipulation. Printing, too, is encumbered by the endpoint browser's view of the page as a block of pixels, as opposed to a document that can be reflowed to accommodate any output device. In contrast, DOM Mirroring provides the endpoint browser's existing workflow mechanisms with all the information it needs to do its job, so emulation is not needed.

Security

DOM Mirroring enables a native user experience by exposing a web page's semantic structure (the DOM) to the endpoint browser. However, to ensure that this additional exposure does not sacrifice security for user experience, ACR employs two security mechanisms that, in conjunction, offer a strong defense against even the most determined adversaries. These mechanisms are motivated by the following guiding security principle: Active content execution is the key to successful evasion of endpoint defenses; without it, an exploit has little hope of bypassing existing defenses, regardless of what else an infected isolated browser sends down to the client.

Active content execution is the key to successful evasion of endpoint defenses; without it, an exploit has little hope of bypassing existing defenses, regardless of what else an infected isolated browser sends down to the client.

The first mechanism, **active content blocking and transcoding**, leverages the fact that active content entry points in the DOM are well defined to filter all incoming DOM elements, attributes, and CSS against a whitelist at both the Safe Page and the isolated browser. For instance, <script> elements and onclick attributes are dropped, while <object> elements are replaced with a safe remoting widget that displays the transcoded, real-time output of the plug-in window as rendered on the isolated browser. As an added layer of protection, the Safe Page also employs Content Security Policy at its strictest setting (no inline script, no plug-in) to ensure that the endpoint browser blocks all active content executions.

A second security mechanism, **protocol checking and enforcement**, ensures that the Safe Page is not fooled into executing active content by malformed updates coming from an infected isolated browser, and that it does not inadvertently leak exploit-aiding information to an infected isolated browser. In particular, all DOM updates coming from the isolated browser are expected to be in a canonical format. For example, DIV elements must have the "div" tag. The Safe Page does not accept any other string variant, even those including strange character codes that may be interpreted unexpectedly by the endpoint browser. Secondarily, the Safe Page verifies that outgoing messages adhere to a simple user-input protocol: e.g., "click button 1", "keypress code 45", or "scroll to 45". This leaves an infected isolated browser without a channel to probe the endpoint for vulnerabilities or to exfiltrate information useful for bypassing standard endpoint defenses.

Applicability to Document Isolation

Like web browsers, document applications such as Microsoft Office and PDF viewers are also susceptible to malicious content downloaded off the web or sent via email attachments. Here, too, active content embedded within a malicious document plays an enabling role in exploiting the host application's vulnerabilities. It is no surprise that the core techniques behind ACR apply equally well to the problem of isolating documents. In particular, upon downloading a malicious document through the browser, MSIP uses ACR to transcode the document into a layout-preserving HTML5 page and then loads the transcoded content into the isolated browser. DOM Mirroring mechanisms then ensure safe, clientless, and transparent mirroring of the document to the endpoint.

Conclusion

The browser feature set continues to expand with new JavaScript-accessible HTML5 APIs slowly supplanting Flash. Going forward, we can anticipate novel exploits against this newly exposed attack surface, with active content firmly remaining the predominant vector for exploitation. Browser isolation promises to shield users from these future threats by running active content away from the endpoint. However, to reach its true potential, browser isolation faces the challenge of providing both clientless deployment and a fully transparent user experience. Adaptive Clientless Rendering—the novel remoting technology at the core of the Menlo Security Isolation Platform—meets this challenge by selectively mirroring DOM elements to the existing endpoint browser.

With mechanisms to ensure that active content never executes on the endpoint browser, ACR defends against zero-day threats while providing a clientless and native browsing experience.

To learn more, visit menlosecurity.com/resources or get in touch via ask@menlosecurity.com to see how you can use isolation to stop email and web-based attacks.

Contact us
menlosecurity.com
(650) 614-1705
ask@menlosecurity.com