December 31, 2023

Whitepaper:

Unity and Content Security Policy

1 Document Purpose

This document describes the history and meaning behind certain CSP warnings that may occur on scanning a Unity application. It describes what these warnings mean, why they occur, and how they are currently mitigated to provide a safe and security Unity-based application.

2 Executive Summary

Static scans of Unity applications that surface warnings or issues around a CSP header's usage of "unsafe-eval" and "unsafe-inline", indicating the potential for cross-site or other injection attacks, can generally be construed as false-positives.

For some of Unity's UI screens, we utilize an underlying toolkit called ExtJS. This toolkit uses methodologies that, for the most recent CSP standard, require adding "unsafe-inline" and "unsafe-eval" to the CSP header.

Intellective offers both short-term mitigation of the issue and long-term plans for full remediation. Intellective has added input sanitization to message handling in the system, such that methods are screened--based on current industry security best practices--to identify and stop attempts at cross-site or other malicious scripting behavior. In general, for basic usage of out of the box Unity, this mitigates the actual risk of such attacks.

Security for any system is based on that system's specific use cases and context. Please contact Intellective Support if you have questions about your specific implementation of Unity.

3 Background

3.1 Content Security Policy (CSP)

Content Security Policy (CSP) is "a computer security standard introduced to prevent cross-site scripting (XSS), clickjacking and other code injection attacks resulting from execution of malicious content in the trusted web page context."

Introduced in 2014 as a candidate standard, and changed over time to its current version, CSP is an evolving security standard that has generally been adopted by the wider community to improve the security of web applications.

It contains a series of directives, on a variety of behaviors and topics, that organizations can apply to constrain the usage of a given application and reduce available attack surfaces for a malicious actor to affect a web application. Users may optionally adopt or not adopt various directives in the header, depending on desired application behavior, limitations on underlying technologies, and desired security posture.

Application developers can make a CSP header more restrictive or less restrictive to meet specific application and security objectives. This can be done in various ways, including settings in the header. In some cases, these settings are prefixed "unsafe-" to indicate to administrators that a specific security restriction or feature is being "turned off". Using them does not eliminate the other security features in the CSP header; it generally only changes behavior within a specific scope like inline code behavior or usage of "eval" functions.

Many organizations, including current Unity customers, utilize CSPs with various directives to provide the above benefits for their Unity-based applications.

3.2 Static code scanning

Static code scanners are often used by organizations to identify possible areas of concern in applications. Intellective uses such tools against our own codebase to identify and remediate or mitigate any concerns that occur in the evolving software landscape.

Generally, they indicate areas of potential concern or risk, ranging from trivial "informational only" items to items of greater risk, requiring more urgent remediation.

While they continue to increase in sophistication over time, they generally lack some degree of context and nuance, and often recommendations must be viewed by security professionals through a given lens to determine a given course of action.

Static code scanning is generally considered to be an ongoing maintenance activity, with new code releases from third party vendors continuously occurring, and an ever-evolving world of possible threats and remediations happening over time. As such, Intellective continuously applies fixes and remediations to the Unity codebase as threats are identified, depending on the severity, urgency, and likelihood of threats resulting in risk or harm to customers. Please contact Intellective support for more information about release policies with respect to security fixes.

4 "Unsafe" directives in the CSP header

Current static code scanners may raise an issue with a Unity-based application, reporting that the application's CSP header contains an "unsafe-eval" or "unsafe-inline" directive. This may depend on the given type of code scanner, its settings, and the current state of a Unity-based application's CSP header.

Generally, Unity can accommodate more restrictive CSP headers, allowing customers to make very restrictive configurations. However, an underlying technology that some of Unity's user interfaces employ--Sencha ExtJS--requires the usage of specific "unsafe"-labelled directives.

5 Why are they used?

First released in 2007, ExtJS is a full JavaScript-based UI platform for building complex and performant user interfaces. It provides a flexible set of UI building blocks that allow for both rapid application development and a high degree of customization. Unity makes use of these key features to provide extremely performant and customizable applications to customers.

ExtJS was built prior to the existence of CSP as a concept. At a core level, ExtJS uses several techniques that allow for very fast generation of HTML content, generally even faster than some of the most recent or "modern" JavaScript frameworks.

Those techniques involve dynamic generation of content at runtime, in a way that requires the usage of one or more specific "unsafe"-labelled directives in the CSP header. The methodology used is not inherently unsafe, but it falls into a category of behavior that a highly restrictive CSP strategy may exclude.

The interaction between CSP and Sencha's ExtJS platform is well understood and has been evaluated by Sencha. Per their statement below, it can be treated as a false positive if basic input sanitization best practices are followed.

See Sencha's statement on this topic here: https://docs-devel.sencha.com/extjs/7.6.0/guides/other_resources/use_eval_func.html

Based on this guidance, Intellective employs input sanitization to mitigate the risk. See section 7 "How does Intellective mitigate this risk?".

6 Can Intellective modify the third-party ExtJS code to allow removal of the directives?

No, the behavior in ExtJS that requires unsafe directives is a core part of the proprietary ExtJS platform and cannot be modified by other parties.

There are no current Sencha-supported paths for "sidestepping" or otherwise eliminating the need for the directives while still using ExtJS code.

7 How does Intellective mitigate this risk?

Customers whose systems are internal-only, and not exposed to the external network, may determine that this risk is significantly mitigated due to no opportunity for external exploitation being available. However, Intellective has provided an additional layer of mitigation described below.

At the Unity layer, where Unity uses ExtJS components, we have applied a standard, robust screening methodology to sanitize inputs that could be affected by an inline or eval based attack.

An example of the industry practices followed in creating this mitigation can be found here: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html)

In practice, this involves:

- Create a screening filter in the core application code through which all requests are passed. All requests, including those involving eval or inline behavior that require the unsafe directives, are passed through the screening filter prior to being processed by the system. This would include client-side elements like basic field inputs or manipulation of the DOM.
- Within the filter, evaluate/sanitize the contents of the request.
- If malformed data, improper tags, injection of malicious or malformed code, suspicious usage, or other excluded behavior is detected, the filter throws an error, causing the request to halt and preventing the application from proceeding with the entire request. Some examples of what specific areas of concern are being checked in the filter: https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html
- If the request is properly formed/sanitized with no unallowed elements, process the request as normal.

In this way, the application seeks to mitigate the risk presented by the usage of the unsafe directives, by filtering and proactively preventing any execution of injected/manipulated inputs.

Many Unity customers currently operate with this mitigation strategy in place, and Intellective continues to evaluate and improve Unity's overall security posture over time. No single approach is foolproof, and security is an ever-evolving topic, however security is of paramount concern to Intellective and we seek to provide the strongest possible options for customers to create secure and performant applications.

Please note: these mitigations were introduced generally in Unity version 7.8.x. Some customers may have also received custom development work, either created by themselves or by Intellective professional services, to do the same kind of mitigation. If you're unsure about your current status, please contact Intellective Support.

8 Will Sencha fully remove the need for Unsafe directives in the future?

Sencha is fully aware of the security finding by some static scan tools and is aware of the impact to those customers who require more restrictive CSP headers. They have not publicly indicated specific plans or timelines for a long-term change to the platform to remove the need for the unsafe directives.

9 What are long-term options to fully remediate this risk?

Intellective has deprecated the usage of ExtJS components in the Unity UI as of Unity 8.81. Additionally, we plan to extend Unity's React UI capabilities in the future to provide a full UI experience that does not require the usage of ExtJS. This will both fully remediate the issue described in this whitepaper, as well as provide additional modernization and features for customers utilizing the Unity UI. Additional React components are planned for general release in 2024, providing a path for customers to update their Unity-based systems to take advantage of the new components and fully remediate the issue.

Please contact Intellective Support with any questions or concerns, or to obtain help from Intellective Professional Services in evaluating your long-term UI strategy.