

Understand Cross-Origin Resource Sharing (CORS) for Finesse

Contents

[Introduction](#)

[Prerequisites](#)

[Requirements](#)

[Components Used](#)

[Background Information](#)

[What is CORS](#)

[Lifecycle of a CORS](#)

[CORS in Action with Cisco Finesse](#)

[Illustrative Example: Analyzing CORS Behavior with the Live Data Gadget](#)

[TAC Tool for CORS Connection Testing](#)

Introduction

This document describes Cross-Origin Resource Sharing completely so that, during troubleshooting, the underlying processes are thoroughly understood.

Prerequisites

Requirements

Cisco recommends that you have knowledge of these topics:

- Cisco Unified Contact Center Enterprise (UCCE) Release 12.6.X
- Cisco Packaged Contact Center Enterprise (PCCE) Release 12.6.X
- Cisco Finesse Release 12.6.X
- Cisco Unified Intelligence Center (CUIC) Release 12.6.X

Components Used

The information in this document is based on these software and hardware versions:

- UCCE Release 12.6.2
- Finesse Release 12.6.2
- CUIC Release 12.6.2

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, ensure that you understand the potential impact of any command.

Background Information

What is CORS

Cross-Origin Resource Sharing (CORS) is a way for servers to control which websites (domains, protocols, and ports) are allowed to access their resources. While browsers normally block requests from different origins (the same-origin policy), CORS gives servers the power to selectively relax this restriction. Essentially, servers use special HTTP headers to tell the browser which origins are permitted, what types of requests are allowed (like GET, POST, and so on), and which custom headers can be included. This lets servers decide who can access their APIs and how, ranging from completely open to strictly limited access. CORS works by having the browser and server communicate through these HTTP headers to manage cross-origin requests.

CORS uses HTTP headers to enable controlled cross-origin requests. The browser and server communicate via these headers, with the server specifying allowed origins, methods, and headers. If the server's response headers are missing or invalid, the browser blocks the response, enforcing the same-origin policy. For certain requests, the browser first sends a preflight request to the server to ensure it accepts the actual cross-origin request.

Browsers use preflight requests to check if a server allows a cross-origin request before sending the real request. These preflight requests include details like the HTTP method and custom headers. CORS-enabled servers can then respond, either permitting or denying the actual request. If a server is not configured for CORS, it does not respond correctly to the preflight, and the browser blocks the actual request, thus protecting the server from unwanted cross-origin access.

Cross-Origin Resource Sharing (CORS) is crucial for web security and functionality. It allows controlled access to resources from different origins (domains, protocols, ports), which is necessary because browsers enforce a Same-Origin Policy that normally blocks such access.


Lifecycle of a CORS

A CORS request consists of two sides: the client making the request, and the server receiving the request. On the client side, the developer writes JavaScript code to send the request to the server. The server responds to the request by setting special CORS-specific headers to indicate that the cross-origin request is allowed. Without both the client's and the server's participation, the CORS request fails.

The key players in a CORS request are the client, the browser, and the server. Client wants some piece of data from the server, such as a JSON API response or the contents of a web page. The browser acts as the trusted intermediary to verify that the client can access the data from the server.

Client:

The client is a snippet of JavaScript code running on a website, and it is responsible for initiating the CORS request

 **Note:** Finesse is a web application. It is installed on a server, and agents access it simply by using their web browsers, eliminating the need for client-side installations or maintenance of plugins or other software. As demonstrated in the CORS in Action with Cisco Finesse example, this architecture supports features like live data reports. In this context, the Cisco Finesse live data gadget's JavaScript code acts as the client, while Cisco CUIC serves as the server within the CORS lifecycle. Essentially, the browser-based Finesse client interacts with the CUIC server to retrieve live data.

Client versus user:

Sometimes the words client and user are used interchangeably, but they are different in the context of

CORS. A user is a person visiting a website or Finesse user (agent or supervisor) accessing Finesse in this context, while a client is the actual code served by that website. Multiple users can visit the same website and be served the same JavaScript client code.

Browser:

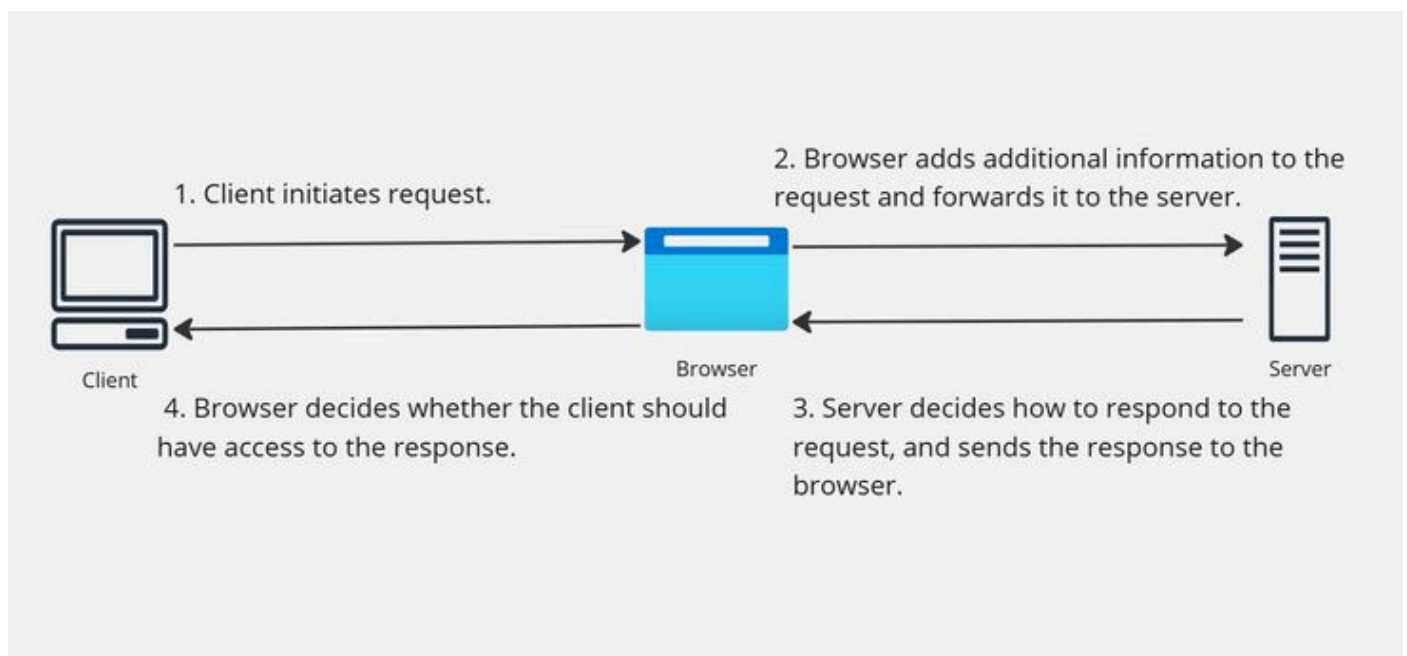
The browser, also known as the user agent, hosts the client-side code. It plays a crucial role in CORS by adding extra information to outgoing requests, enabling the server to identify the client. Furthermore, the browser interprets the server's response, determining whether to deliver the data to the client or return an error. These browser-side actions are essential for maintaining the security provided by the same-origin policy. Without the browser's enforcement of CORS rules, clients could make unauthorized requests, compromising this vital security mechanism.

Server:

The server is the destination of the CORS request, and it is CUIC for Live data gadget example with Cisco Finesse. The server stores the data that the client wants, and it has the final say as to whether the CORS request is allowed or not.

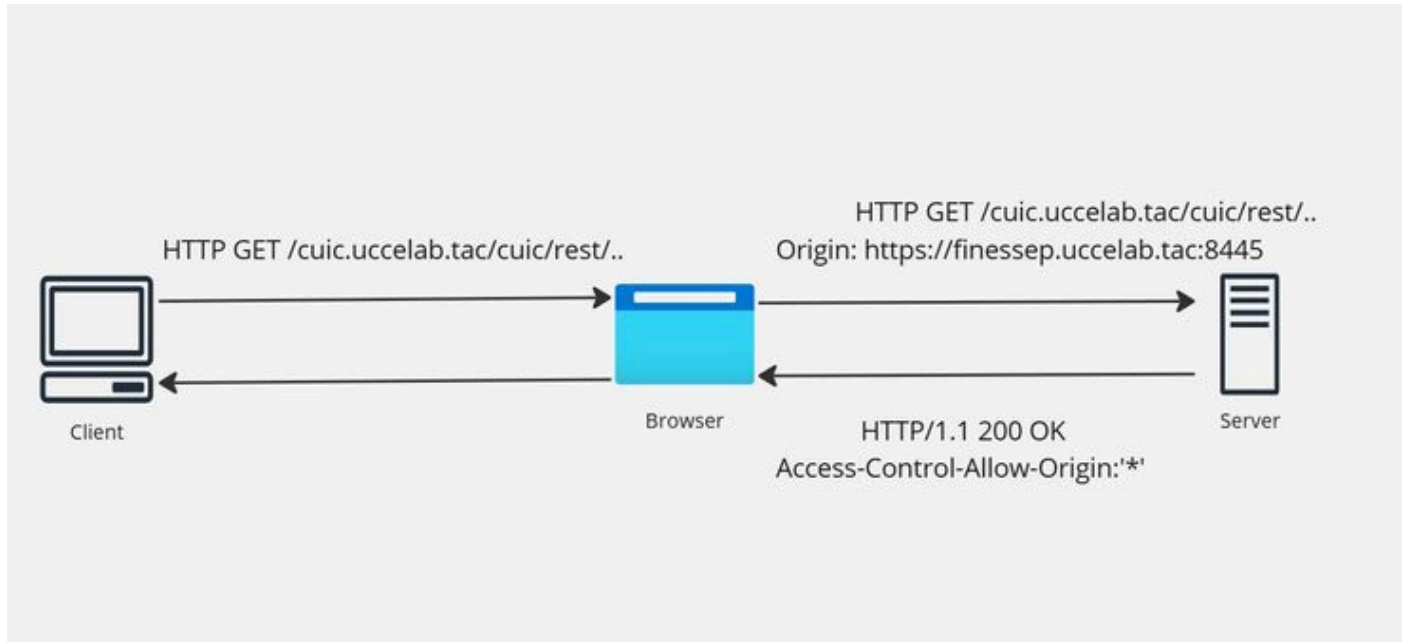
Now that you know who is involved in a CORS request, let us take a look at how they all work together. The subsequent images illustrate the high-level CORS lifecycle:

1. The client initiates the request.
2. The browser adds additional information to the request and forwards it to the server.
3. The server decides how to respond to the request, and sends the response to the browser.
4. The browser decides whether the client must have access to the response, and either passes the response to the client or returns an error.

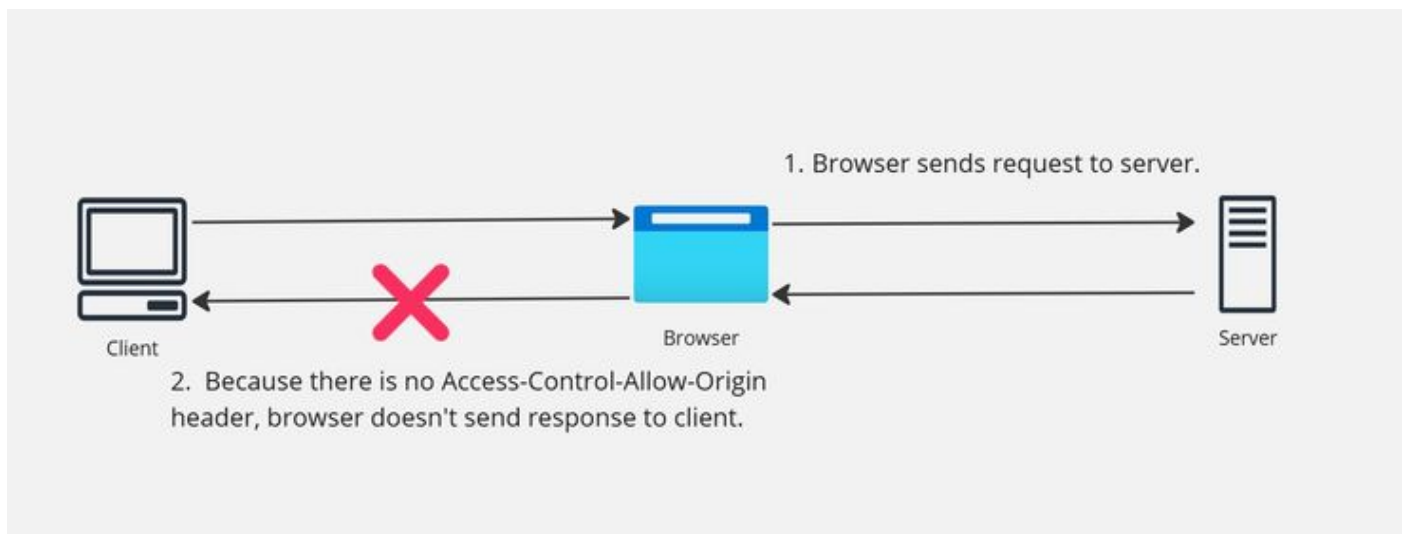


Before sending a cross-origin request, the browser automatically adds an origin header to the HTTP request. This header, which the client cannot modify, is a crucial part of CORS and serves to identify the client's origin (that is, the domain, protocol, and port from which the client resource was loaded). This security measure prevents clients from impersonating other origins. The origin header is fundamental to CORS, as it is how the client tells the server where it comes from.

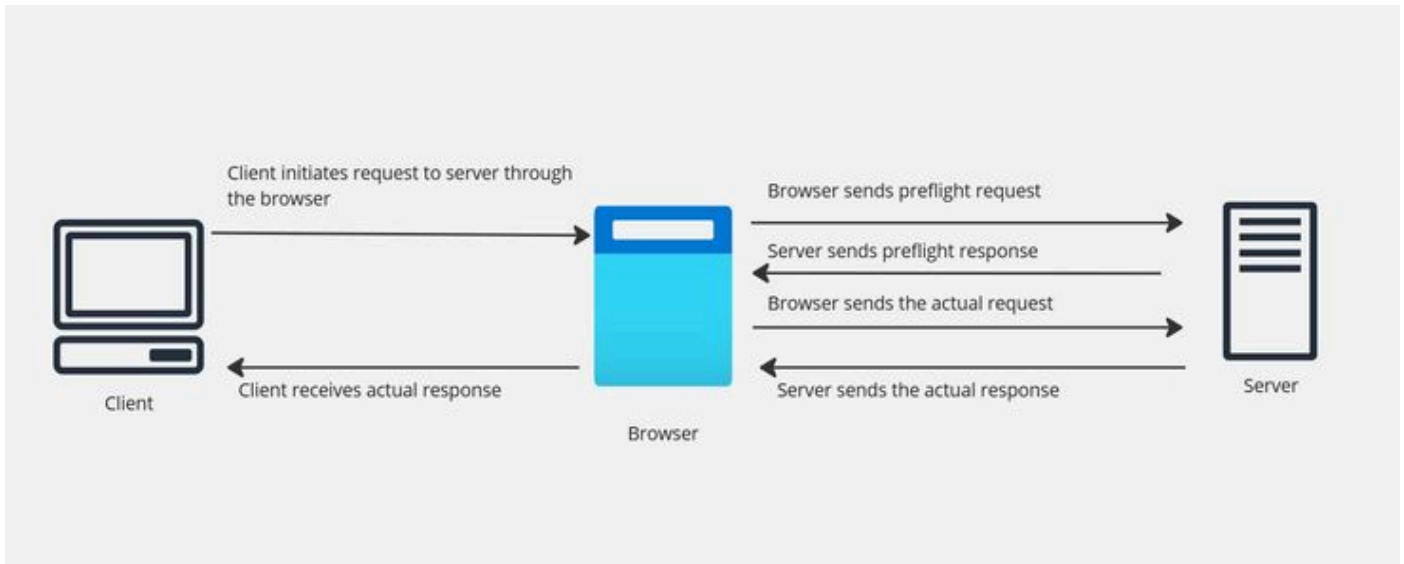
In a Cross-Origin Resource Sharing (CORS) interaction, the client's origin is identified by the Origin header in the initial request. The server then uses the Access-Control-Allow-Origin header in its response to indicate whether the client is permitted to access the requested resource. This response header is crucial; if absent, the CORS request fails. The Access-Control-Allow-Origin header can contain either a wildcard (*), allowing access from any origin, or a specific origin, granting access only to that particular client. While the image shows Access-Control-Allow-Origin: *, implying CUIC allows all origins, CUIC typically sends this header with a specific origin in real-world scenarios.



When a browser rejects a CORS request, it means the client receives no information about the server's response. The client only knows that an error occurred, but lacks details about the specific issue. This can make debugging CORS errors challenging, as it is difficult to distinguish a CORS failure from other types of errors. Even though the initial request is sent to the server, if the server's response lacks a valid Access-Control-Allow-Origin header, the browser blocks the response and triggers an error on the client side, preventing the client from ever seeing the server's detailed response.



This image explains the entire CORS process, with a particular focus on the preflight stage, which is essential for handling specific types of cross-origin requests.

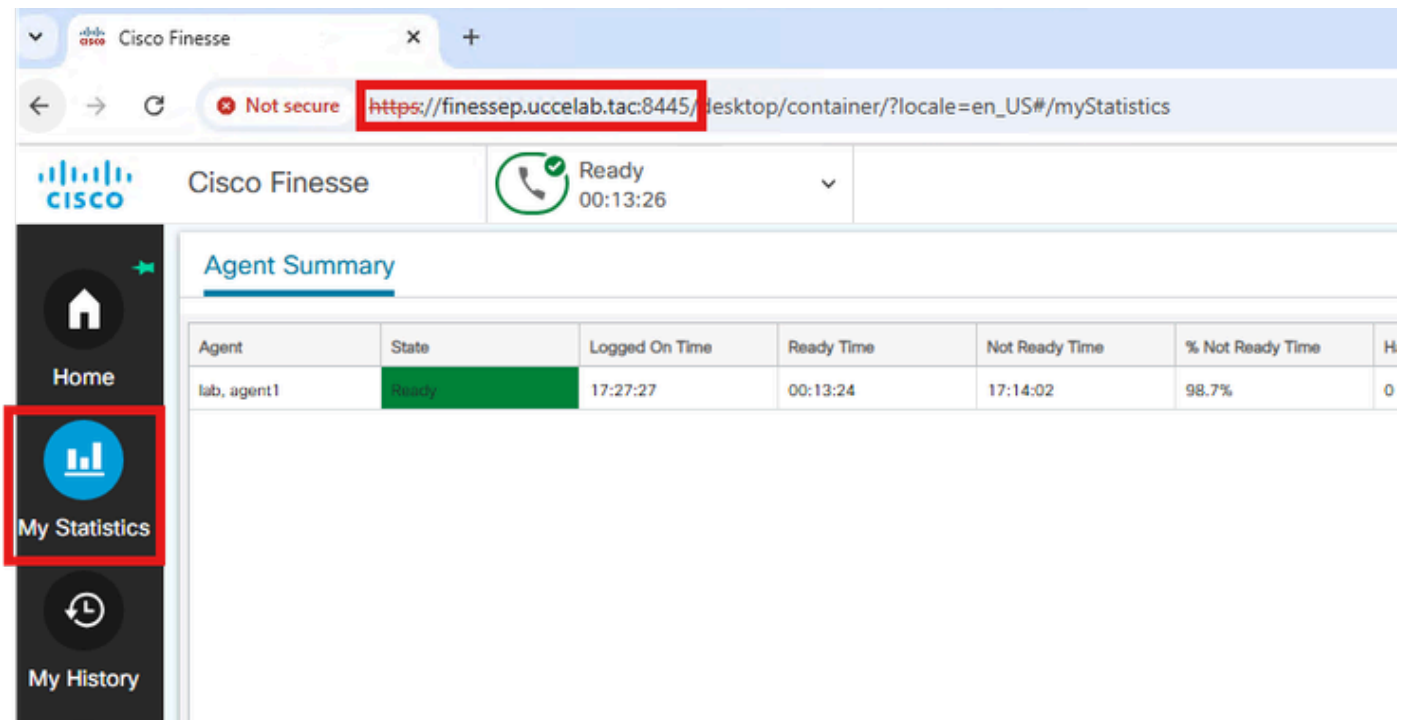


CORS in Action with Cisco Finesse

Illustrative Example: Analyzing CORS Behavior with the Live Data Gadget

This section describes the typical use of Cross-Origin Resource Sharing (CORS) with Cisco Finesse in contact centers. Agents and supervisors commonly use Cisco Finesse to access real-time data reports (as illustrated in the example image).

When an agent or supervisor clicks a report gadget, their action initiates a data retrieval request. This request is sent from the Finesse application's JavaScript code (acting as the client) to the CUIC/Live Data server using a GET method. As demonstrated in the SAML tracer image, the browser first sends a preflight request to the server, the CORS lifecycle described earlier.



An HTTP OPTIONS request (the preflight request) is sent to the CUIC/Live Data server. This request specifies the origin as the fully qualified domain name (FQDN) of the Finesse server, including port 8445. This is the same address and port that agents use to access the Cisco Finesse application.

SAML-tracer

X Clear II Pause Autoscroll Filter resources Colorize Export Import

GET https://cuicpub.ucclab.tac/security?1738431200084

GET https://cuicpub.ucclab.tac/livedata/security?1738431200084

GET https://cuicsub.ucclab.tac/livedata/security?1738431204035

GET https://cuicsub.ucclab.tac/security?1738431204035

GET https://cuicpub.ucclab.tac/security?1738431212114

GET https://cuicpub.ucclab.tac/livedata/security?1738431212114

GET https://cuicsub.ucclab.tac/security?1738431212115

GET https://cuicsub.ucclab.tac/livedata/security?1738431212115

OPTIONS https://cuicpub.ucclab.tac/livedata/api/snapshotRequest/agentConfig?userId=agent1&ids=5001

GET https://cuicpub.ucclab.tac/livedata/api/snapshotRequest/agentConfig?userId=agent1&ids=5001

HTTP

OPTIONS https://cuicpub.ucclab.tac/livedata/api/snapshotRequest/agentConfig?userId=agent1&ids=5001 HTTP/1.1

Accept: */*

Access-Control-Request-Method: GET

Access-Control-Request-Headers: authorization, content-type, domain, ldauthheader, locale, peripheralid

Origin: https://finessep.ucclab.tac:8445

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36

Sec-Fetch-Mode: cors

Sec-Fetch-Site: same-site

Sec-Fetch-Dest: empty

Referer: https://finessep.ucclab.tac:8445/

Accept-Encoding: gzip, deflate, br, zstd

Accept-Language: en-US,en;q=0.9

HTTP/1.1 200

server: nginx

date: Sat, 01 Feb 2025 17:33:34 GMT

content-type: application/octet-stream

content-length: 0

access-control-allow-origin: https://finessep.ucclab.tac:8445

access-control-max-age: 600

access-control-allow-credentials: true

access-control-allow-methods: GET,POST,OPTIONS,PUT,DELETE

access-control-allow-headers: Content-Type,X-Requested-With,accept,Origin,Authorization,Access-Control-Request-Method,Access-Control-Request-Headers,Domain,locale,peripheralid,ldauthheader

access-control-expose-headers: Access-Control-Allow-Origin,Access-Control-Allow-Credentials,Access-Control-Allow-Methods,Access-Control-Allow-Headers,Access-Control-Max-Age

The command-line interface (CLI) commands on the CUIC/Live Data server control which origins are permitted to access its live data resources. If the Finesse server's origin (its FQDN and port) is configured in these settings, then agents able to view the live data gadget details within Finesse.

```
admin:utils live-data cors allowed_origin list

cors_allowed_origin
=====
1. https://finessep.ucclab.tac
2. https://finessep.ucclab.tac:8445
3. https://finesses.ucclab.tac
4. https://finesses.ucclab.tac:8445
```

```
admin:utils cuic cors allowed_origin list
cors_allowedorigins
=====
1. https://finessep.uccelab.tac
2. https://finesses.uccelab.tac
3. https://finesses.uccelab.tac:8445
4. https://finessep.uccelab.tac:8445
admin:
```

TAC Tool for CORS Connection Testing

CORS misconfigurations on the server side can sometimes cause problems with third-party or live data gadgets in Cisco Finesse. This article provides a link to a CORS Quick Check gadget, a troubleshooting tool is designed to help diagnose Cross-Origin Resource Sharing issues affecting Finesse gadgets, including live data displays and other third-party integrations.

Technically, this gadget works by sending preflight requests from the Cisco Finesse client to a specified target resource. This quick check functionality helps to quickly identify and resolve CORS-related problems, speeding up the troubleshooting process.

To deploy the Contact Center CORS Quick Check 12.6-v1.0 gadget within the Finesse desktop:



1. Download [gadget's files](#) from Contact Center CORS Quick Check 12.6-v1.0 folder.
2. Copy the contents of the Contact Center CORS Quick Check 12.6-v1.0 folder into the 3rdpartygadget directory within your Finesse installation.
3. Add the gadget to the desired user role (Agent, Supervisor, and so on) in the Finesse desktop layout. The provided example XML demonstrates the correct configuration for adding this gadget.

```
<gadget>/3rdpartygadget/files/TestCORSgadget.xml</gadget>
```

See the Third Party Gadgets chapter in the [Finesse Developer Guide](#) and the Manage Third-Party Gadgets chapter in the [Finesse Administration Guide](#) for more information about uploading third-party gadgets and adding them to the desktop.

Once the gadget files are uploaded and the Cisco Finesse Tomcat service is restarted, the gadget is available and displays the graphical user interface (GUI).

← → ↻ Not secure https://finessep.uccelab.tac:8445/desktop/container/?locale=en_US#/GadgetTest

 Cisco Finesse  Ready 00:47:55

Contact Center CORS Quick Check

Insert FQDN/IP address of the targeted resource:

Choose which Cisco product you are testing:

Cisco CUIC

--Please choose an option--

Cisco Finesse

Cisco CUIC



Other

[Test CORS Connection](#)

Home
My Statistics
My History

You can select CUIC from the top drop down list at the top. Enter the fully qualified domain name (FQDN) of the CUIC server in the provided field. A successful test is going to be as shown here.

← → ↻ Not secure https://finessep.uccelab.tac:8445/desktop/container/?locale=en_US#/GadgetTest

 Cisco Finesse  Ready 00:51:44

Contact Center CORS Quick Check

Insert FQDN/IP address of the targeted resource:

Choose which Cisco product you are testing:

Cisco CUIC

[Test CORS Connection](#)

CORS Preflight Test Succeeded ✓

Home
My Statistics
My History

A successful test means the CUIC server is correctly configured for Cross-Origin Resource Sharing (CORS) with the Finesse server. The browser's SAML tracer logs show that an HTTP OPTIONS request (the CORS preflight) was sent to the CUIC server. This request included the Finesse server's address in the Origin header. The CUIC server responded with a 200 OK HTTP message, and importantly, the Access-Control-Allow-Origin header in the response also contained the Finesse server's address. This confirms that the CUIC server is configured to allow requests from the Finesse server's origin, verifying that CORS is set up correctly.

<#root>

OPTIONS https://cuicpub.uccelab.tac/cuic/ HTTP/1.1

sec-ch-ua-platform: "Windows"

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome..

sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A Brand";v="24"

sec-ch-ua-mobile: ?0

Accept: */*

Origin: https://finessep.uccelab.tac:8445

Sec-Fetch-Site: same-site

Sec-Fetch-Mode: cors

Sec-Fetch-Dest: empty

Referer: https://finessep.uccelab.tac:8445/

Accept-Encoding: gzip, deflate, br, zstd

Accept-Language: en-US,en;q=0.9

<#root>

HTTP/1.1 200

server: nginx

date: Sat, 08 Feb 2025 01:27:47 GMT

content-length: 0

strict-transport-security: max-age=31536000; includeSubDomains

set-cookie: JSESSIONID=bE73993C4A7C1Fc1b33A7AaF897B8428; Path=/cuic; Secure; HttpOnly; SameSite=Strict

pragma: No-cache

cache-control: no-cache

expires: Thu, 01 Jan 1970 00:00:00 GMT

x-frame-options: SAMEORIGIN

x-xss-protection: 1; mode=block

x-content-type-options: nosniff

content-security-policy: default-src 'self' ; script-src 'self' data: 'unsafe-inline' 'unsafe-eval' ; s

vary: origin,access-control-request-method,Access-Control-Request-Headers

access-control-allow-origin: https://finessep.uccelab.tac:8445

access-control-allow-credentials: true

access-control-expose-headers: access-control-allow-origin,access-control-allow-credentials,access-cont

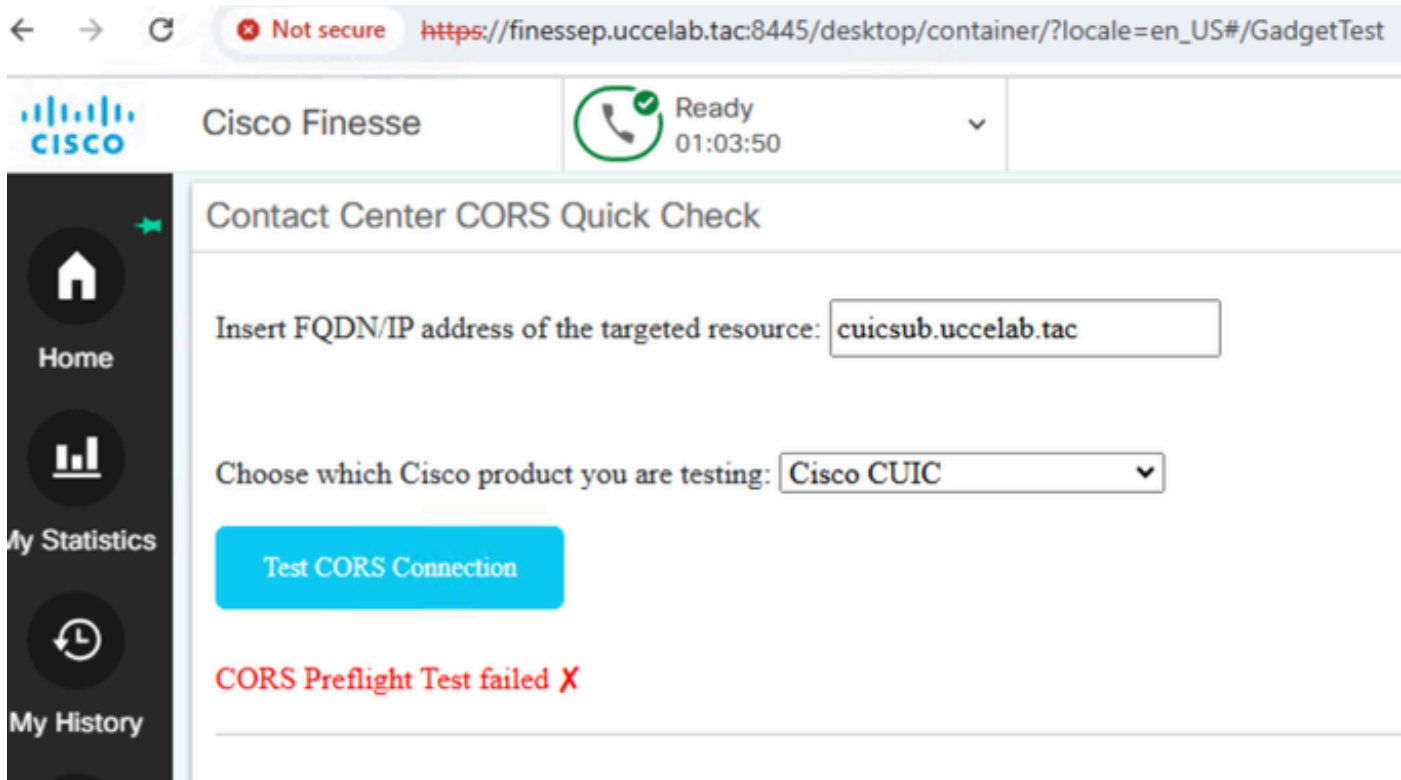
access-control-max-age: 600

access-control-allow-methods: DELETE,POST,GET,OPTIONS,PUT

access-control-allow-headers: referer,peripheralid,origin,access-control-request-method,locale,accept,a

allow: GET,POST,OPTIONS,PUT,DELETE

In this scenario, the tool demonstrates a non-working configuration. Unlike the previous example, the Finesse server is not configured as a subscriber on the CUIC server. Instead, it is only configured on the CUIC publisher. As a result, the CORS preflight request fails, and the CUIC server responds with an HTTP 403 (Forbidden) error.



<#root>

OPTIONS https://cuicsub.ucelab.tac/cuic/ HTTP/1.1

Accept: */*

Access-Control-Request-Method: OPTIONS

Origin: https://finessep.ucelab.tac:8445

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome..

Sec-Fetch-Mode: cors

Sec-Fetch-Site: same-site

Sec-Fetch-Dest: empty

Referer: https://finessep.ucelab.tac:8445/

Accept-Encoding: gzip, deflate, br, zstd

Accept-Language: en-US,en;q=0.9

<#root>

HTTP/1.1 403

server: nginx

date: Sat, 08 Feb 2025 01:54:52 GMT

content-type: text/html; charset=utf-8

content-length: 2143

strict-transport-security: max-age=31536000; includeSubDomains

set-cookie: JSESSIONID=1C7606841B83d7847486c3d18D31cEfD; Path=/cuic; Secure; HttpOnly; SameSite=Strict

pragma: No-cache

cache-control: no-cache

expires: Thu, 01 Jan 1970 00:00:00 GMT

x-frame-options: SAMEORIGIN

x-xss-protection: 1; mode=block

x-content-type-options: nosniff

As you can see from the output of the CUIC subscriber command-line interface (CLI), Cisco Finesse is not listed. This indicates that Finesse is not currently configured as a subscriber on this CUIC server.

```
<#root>
```

```
admin:utils cuic cors allowed_origin list
```

```
cors_allowedorigins
```

```
=====
```

1. https://finessep.ucclab.tac
2. https://finesses.ucclab.tac
3. https://finesses.ucclab.tac:8445