

Secure Contexts

W3C Candidate Recommendation Draft, 18 September 2021

**This version:**

<https://www.w3.org/TR/2021/CRD-secure-contexts-20210918/>

Latest published version:

<https://www.w3.org/TR/secure-contexts/>

Editor's Draft:

<https://w3c.github.io/webappsec-secure-contexts/>

Previous Versions:

<https://www.w3.org/TR/2021/CRD-secure-contexts-20210916/>

Version History:

<https://github.com/w3c/webappsec-secure-contexts/commits/main/index.bs>

Feedback:

public-webappsec@w3.org with subject line “[secure-contexts] ... *message topic* ...” ([archives](#))

Implementation Report:

<https://wpt.fyi/results/secure-contexts>

Issue Tracking:

[GitHub](#)

Editor:

[Mike West](#) (Google Inc.)

Former Editor:

Yan Zhu (Brave)

Participate:

[File an issue](#) ([open issues](#))

Copyright © 2021 [W3C](#)® ([MIT](#), [ERCIM](#), [Keio](#), [Beihang](#)). W3C [liability](#), [trademark](#) and [permissive document license](#) rules apply.

Abstract

This specification defines "secure contexts", thereby allowing user agent implementers and specification authors to enable certain features only when certain minimum standards of authentication and confidentiality are met.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the [W3C technical reports index at https://www.w3.org/TR/](https://www.w3.org/TR/).

This document was published by the [Web Application Security Working Group](#) as a Candidate Recommendation Draft. This document is intended to become a W3C Recommendation.

The ([archived](#)) public mailing list public-webappsec@w3.org (see [instructions](#)) is preferred for discussion of this specification. When sending e-mail, please put the text “secure-contexts” in the subject, preferably like this: “[secure-contexts] ...*summary of comment*...”

Publication as a Candidate Recommendation does not imply endorsement by the W3C Membership. A Candidate Recommendation Draft integrates changes from the previous Candidate Recommendation that the Working Group intends to include in a subsequent Candidate Recommendation Snapshot. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

The entrance criteria for this document to enter the Proposed Recommendation stage is to have a minimum of two independent and interoperable user agents that implement all the features of this specification, which will be determined by passing the user agent tests defined in the test suite developed by the Working Group. The Working Group will prepare an implementation report to track progress.

This document was produced by a group operating under the [W3C Patent Policy](#). W3C maintains a [public list of any patent disclosures](#) made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains [Essential Claim\(s\)](#) must disclose the information in accordance with [section 6 of the W3C Patent Policy](#).

This document is governed by the [15 September 2020 W3C Process Document](#).

Table of Contents

1	Introduction
1.1	Top-level Documents
1.2	Framed Documents
1.3	Web Workers
1.4	Shared Workers
1.5	Service Workers
2	Framework
2.1	Integration with WebIDL
2.2	Integrations with HTML
2.2.1	Shared Workers
2.2.2	Feature Detection
2.2.3	Secure and non-secure contexts
3	Algorithms
3.1	Is <i>origin</i> potentially trustworthy?
3.2	Is <i>url</i> potentially trustworthy?

- 4 Threat models and risks**
 - 4.1 Threat Models
 - 4.1.1 Passive Network Attacker
 - 4.1.2 Active Network Attacker
 - 4.2 Ancestral Risk
 - 4.3 Risks associated with non-secure contexts

5 Security Considerations

- 5.1 Incomplete Isolation
- 5.2 localhost

6 Privacy Considerations

7 Implementation Considerations

- 7.1 Packaged Applications
- 7.2 Development Environments
- 7.3 Restricting New Features
- 7.4 Restricting Legacy Features
 - 7.4.1 Example: Geolocation

8 Acknowledgements

Conformance

- Document conventions
- Conformant Algorithms

Index

- Terms defined by this specification
- Terms defined by reference

References

- Normative References
- Informative References

§ 1. Introduction

This section is not normative.

As the web platform is extended to enable more useful and powerful applications, it becomes increasingly important to ensure that the features which enable those applications are enabled only in contexts which meet a minimum security level. As an extension of the TAG's recommendations in [\[SECURING-WEB\]](#), this document describes threat models for feature abuse on the web (see [§ 4.1 Threat Models](#)) and outlines normative

requirements which should be incorporated into documents specifying new features (see [§ 7 Implementation Considerations](#)).

The most obvious of the requirements discussed here is that application code with access to sensitive or private data be delivered confidentially over authenticated channels that guarantee data integrity. Delivering code securely cannot ensure that an application will always meet a user’s security and privacy requirements, but it is a necessary precondition.


Less obviously, application code delivered over an authenticated and confidential channel isn’t enough in and of itself to limit the use of powerful features by non-secure contexts. As [§ 4.2 Ancestral Risk](#) explains, cooperative frames can be abused to bypass otherwise solid restrictions on a feature. The algorithms defined below ensure that these bypasses are difficult and user-visible.

The following examples summarize the normative text which follows:

§ 1.1. Top-level Documents

EXAMPLE 1


`http://example.com/` opened in a [top-level browsing context](#) is not a [secure context](#), as it was not delivered over an authenticated and encrypted channel.



`http://example.com/`

EXAMPLE 2

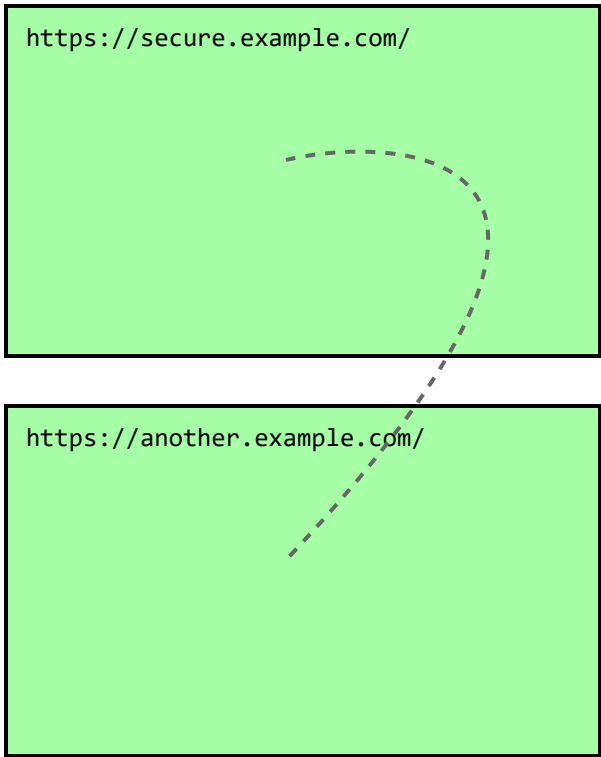
`https://example.com/` opened in a [top-level browsing context](#) is a [secure context](#), as it was delivered over an authenticated and encrypted channel.



`https://example.com/`

EXAMPLE 3

If a secure context opens `https://example.com/` in a new window, that new window will be a secure context, as it is secure on its own merits:

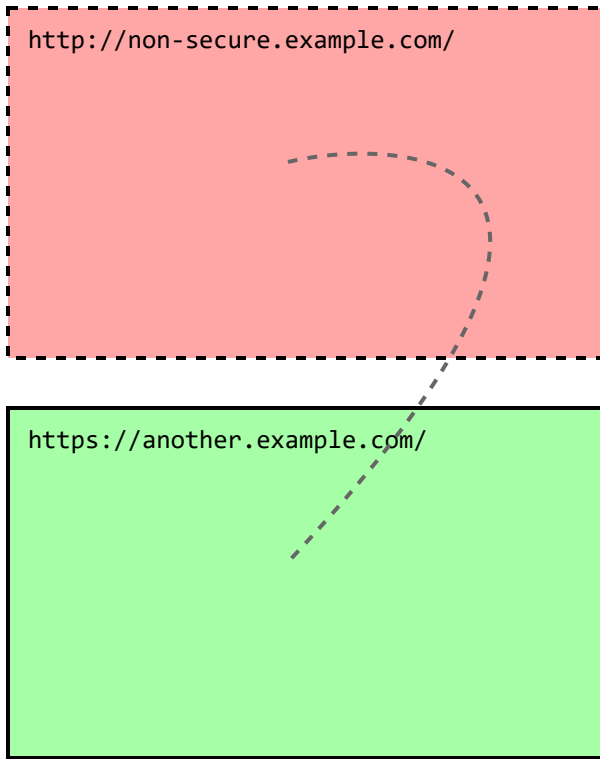


`https://secure.example.com/`

`https://another.example.com/`

EXAMPLE 4

Likewise, if a non-secure context opens `https://example.com/` in a new window, that new window will be a secure context, even though its opener was non-secure:

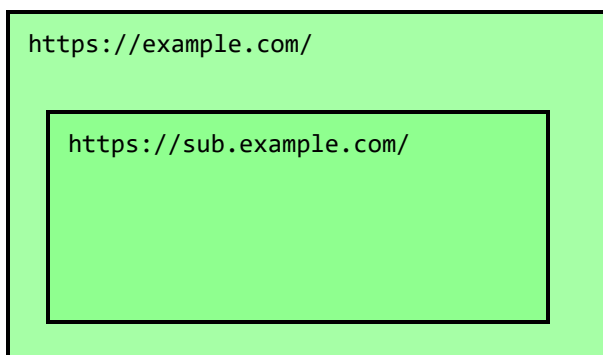


§ 1.2. Framed Documents

Framed documents can be [secure contexts](#) if they are delivered from [potentially trustworthy origins](#), *and* if they're embedded in a [secure context](#). That is:

EXAMPLE 5

If `https://example.com/` opened in a [top-level browsing context](#) opens `https://sub.example.com/` in a frame, then both are [secure contexts](#), as both were delivered over authenticated and encrypted channels.



EXAMPLE 6

If `https://example.com/` was somehow able to frame `http://non-secure.example.com/` (perhaps the user has overridden mixed content checking?), the top-level frame would remain secure, but the framed content is not a secure context.



EXAMPLE 7

If, on the other hand, `https://example.com/` is framed inside of `http://non-secure.example.com/`, then it is *not* a secure context, as its ancestor is not delivered over an authenticated and encrypted channel.

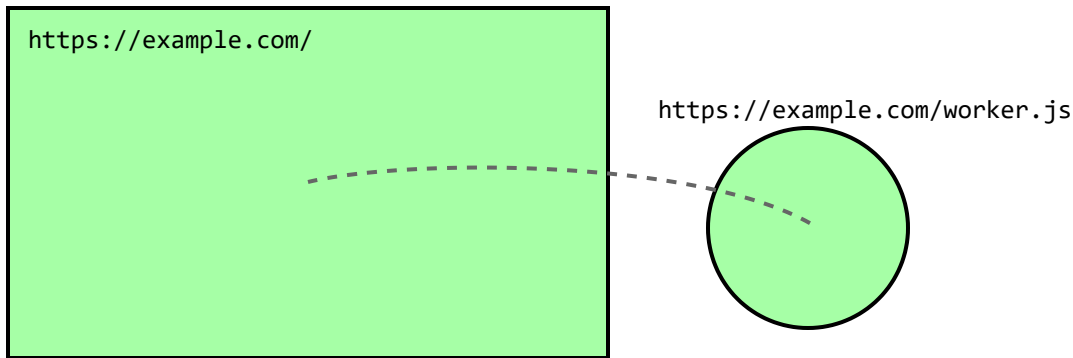


§ 1.3. Web Workers

Dedicated Workers are similar in nature to framed documents. They're [secure contexts](#) when they're delivered from [potentially trustworthy origins](#), only if their owner is itself a [secure context](#):

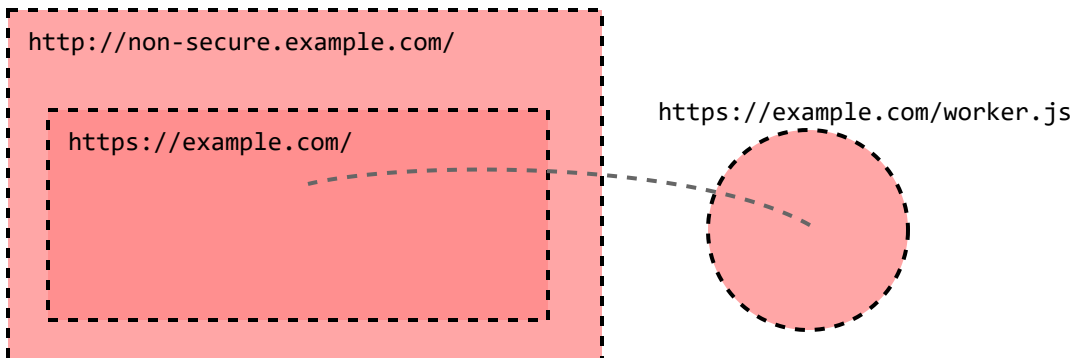
EXAMPLE 8

If `https://example.com/` in a [top-level browsing context](#) runs `https://example.com/worker.js`, then both the document and the worker are [secure contexts](#).



EXAMPLE 9

If `http://non-secure.example.com/` in a [top-level browsing context](#) frames `https://example.com/`, which runs `https://example.com/worker.js`, then neither the framed document nor the worker are [secure contexts](#).

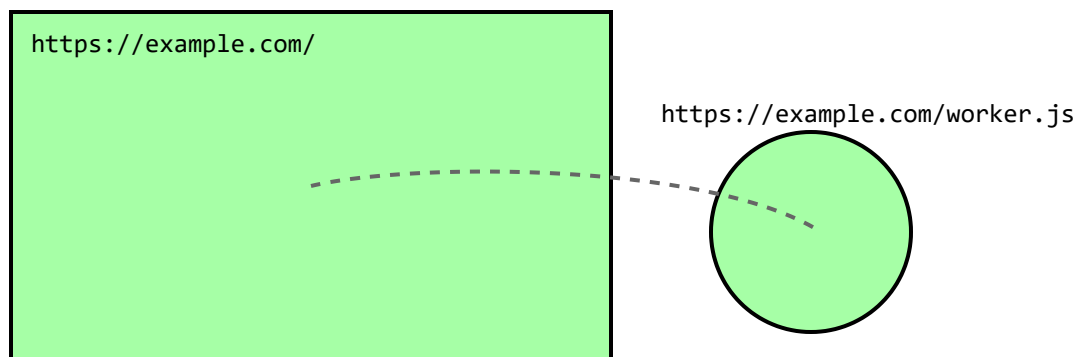


§ 1.4. Shared Workers

Multiple contexts may attach to a Shared Worker. If a [secure context](#) creates a Shared Worker, then it is a [secure context](#), and may only be attached to by other [secure contexts](#). If a non-secure context creates a Shared Worker, then it is *not* a [secure context](#), and may only be attached to by other non-secure contexts.

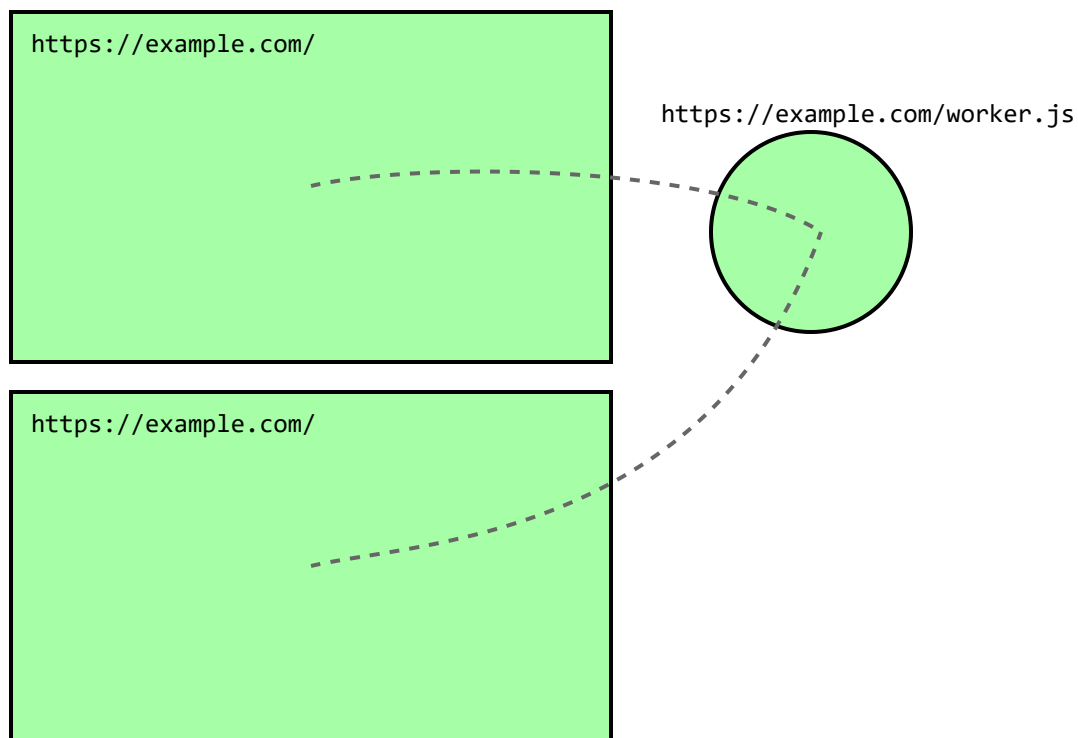
EXAMPLE 10

If `https://example.com/` in a [top-level browsing context](#) runs `https://example.com/worker.js` as a Shared Worker, then both the document and the worker are considered secure contexts.



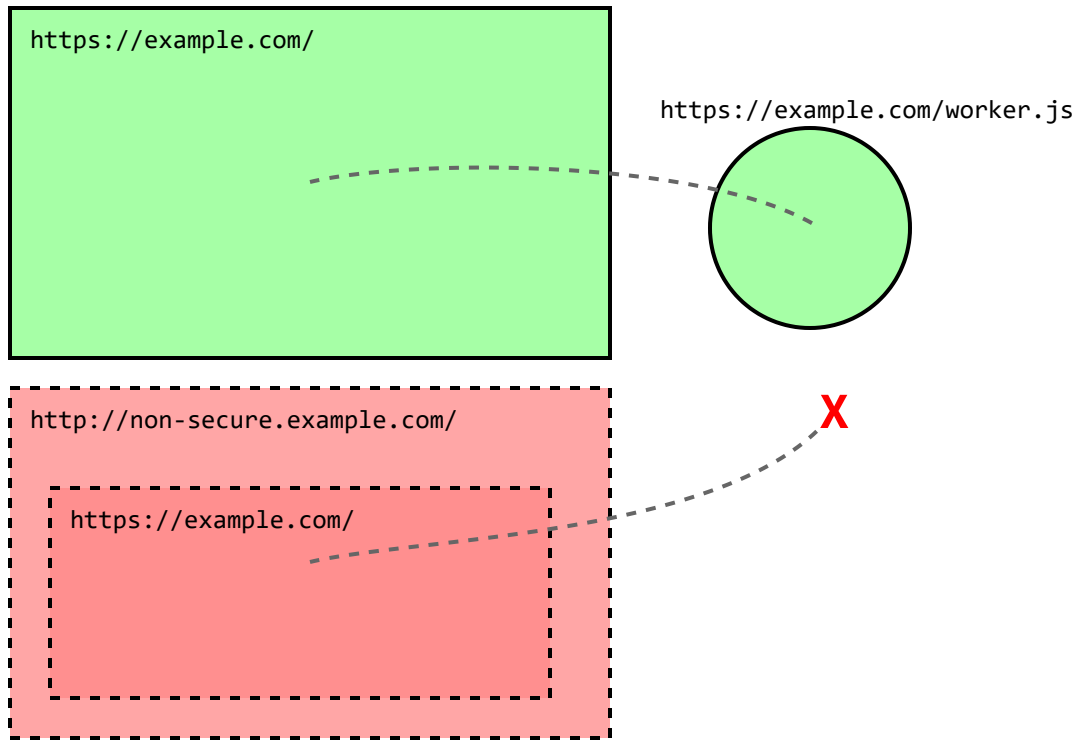
EXAMPLE 11

`https://example.com/` in a different [top-level browsing context](#) (e.g. in a new window) is a secure context, so it may access the secure shared worker:



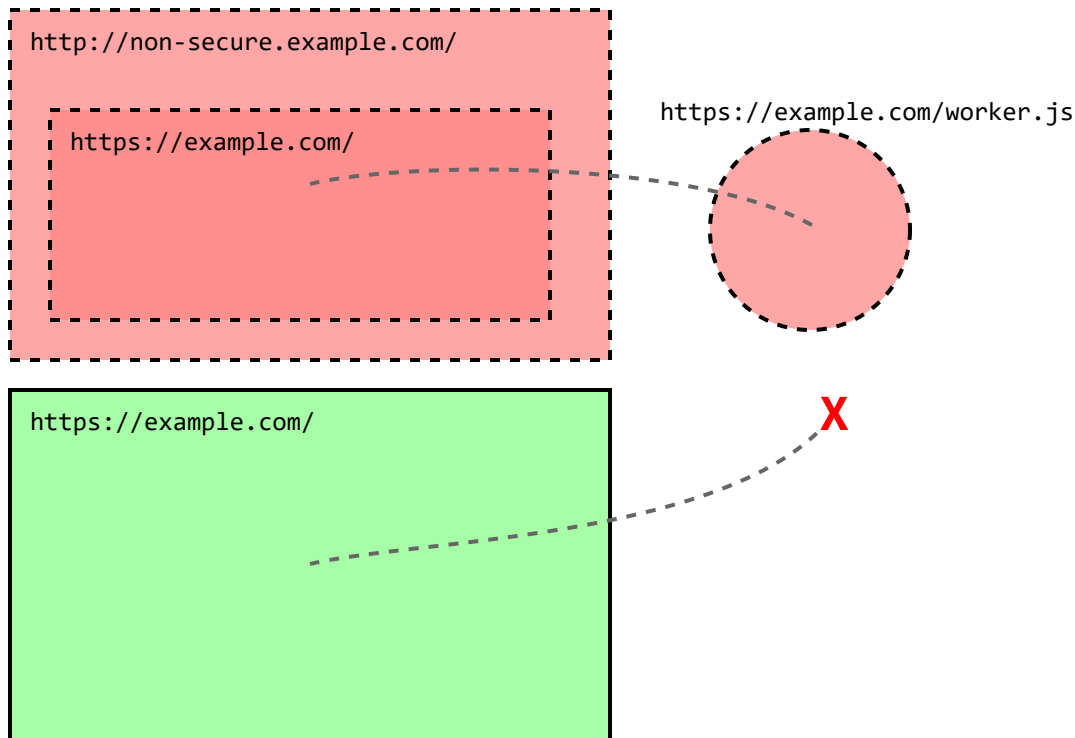
EXAMPLE 12

`https://example.com/` nested in `http://non-secure.example.com/` may not connect to the secure worker, as it is not a secure context.



EXAMPLE 13

Likewise, if `https://example.com/` nested in `http://non-secure.example.com/` runs `https://example.com/worker.js` as a Shared Worker, then both the document and the worker are considered non-secure.

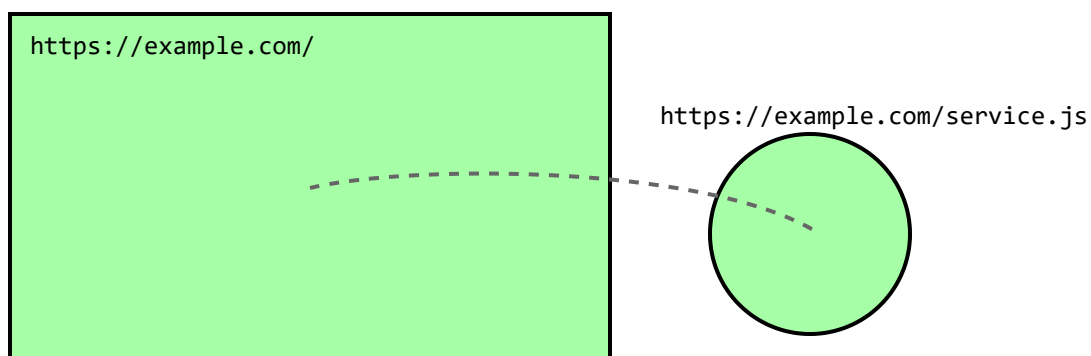


§ 1.5. Service Workers

Service Workers are always [secure contexts](#). Only [secure contexts](#) may register them, and they may only have clients which are [secure contexts](#).

EXAMPLE 14

If `https://example.com/` in a [top-level browsing context](#) registers `https://example.com/service.js`, then both the document and the Service Worker are considered secure contexts.



§ 2. Framework

This section is non-normative.

§ 2.1. Integration with WebIDL

A new `[SecureContext]` attribute is available for operators, which ensures that they will only be [exposed](#) into secure contexts. The following example should help:

EXAMPLE 15

```
interface ExampleFeature {
  // This call will succeed in all contexts.
  Promise<double> calculateNotSoSecretResult();

  // This operation will not be exposed to a non-secure context.
  [SecureContext] Promise<double> calculateSecretResult();

  // The same applies here: the operation will not be exposed to a non-secure context.
  [SecureContext] boolean getSecretBoolean();
};

[SecureContext]
interface SecureFeature {
  // This interface will not be exposed to non-secure contexts.
  Promise<any> doAmazingThing();
};
```

Specification authors are encouraged to use this attribute when defining new features.

§ 2.2. Integrations with HTML

§ 2.2.1. Shared Workers

The [SharedWorker](#) constructor will throw a "[SecurityError](#)" [DOMException](#) exception if a [secure context](#) attempts to attach to a Worker which is not a [secure context](#), and if a non-secure context attempts to attach to a Worker which is a [secure context](#).

§ 2.2.2. Feature Detection

An application can determine whether it's executing in a [secure context](#) by checking the [isSecureContext](#) boolean defined on [WindowOrWorkerGlobalScope](#).

§ 2.2.3. Secure and non-secure contexts

The HTML Standard defines whether an [environment](#) is a [secure context](#) or a [non-secure context](#). This is the primary mechanism used by other specifications.

Given a [global object](#), specifications can check whether its [relevant settings object](#) (which is an [environment](#)) is a [secure context](#).

§ 3. Algorithms

§ 3.1. Is *origin* potentially trustworthy?

A ***potentially trustworthy origin*** is one which a user agent can generally trust as delivering data securely.

This algorithm considers certain hosts, scheme, and origins as potentially trustworthy, even though they might not be authenticated and encrypted in the traditional sense. In particular, the user agent SHOULD treat file URLs as potentially trustworthy. In principle the user agent could treat local files as untrustworthy, but, *given the information that is available to the user agent at runtime*, the resources appear to have been transported securely from disk to the user agent. Additionally, treating such resources as potentially trustworthy is convenient for developers building an application before deploying it to the public.

This developer-friendliness is not without risk, however. User agents which prioritize security over such niceties MAY choose to more strictly assign trust in a way which excludes file.

On the other hand, the user agent MAY choose to extend this trust to other, vendor-specific URL schemes like `app:` or `chrome-extension:` which it can determine *a priori* to be trusted (see [§ 7.1 Packaged Applications](#) for detail).

Given an [origin](#) (*origin*), the following algorithm returns "Potentially Trustworthy" or "Not Trustworthy" as appropriate.

1. If *origin* is an [opaque origin](#), return "Not Trustworthy".
2. Assert: *origin* is a [tuple origin](#).
3. If *origin*'s [scheme](#) is either "https" or "wss", return "Potentially Trustworthy".

Note: This is meant to be analog to the [a priori authenticated URL](#) concept in [\[MIX\]](#).

4. If *origin*'s [host](#) matches one of the CIDR notations 127.0.0.0/8 or ::1/128 [\[RFC4632\]](#), return "Potentially Trustworthy".
5. If the user agent conforms to the name resolution rules in [\[let-localhost-be-localhost\]](#) and one of the following is true:
 - *origin*'s [host](#) is "localhost" or "localhost."
 - *origin*'s [host](#) ends with ".localhost" or ".localhost."

then return "Potentially Trustworthy".

Note: See [§ 5.2 localhost](#) for details on the requirements here.

6. If *origin*'s [scheme](#) is "file", return "Potentially Trustworthy".
7. If *origin*'s [scheme](#) component is one which the user agent considers to be authenticated, return "Potentially Trustworthy".

Note: See [§ 7.1 Packaged Applications](#) for detail here.

8. If *origin* has been configured as a trustworthy origin, return "Potentially Trustworthy".

Note: See [§ 7.2 Development Environments](#) for detail here.

9. Return "Not Trustworthy".

Note: Neither *origin*'s [domain](#) nor [port](#) has any effect on whether or not it is considered to be a [secure context](#).

§ 3.2. Is *url* potentially trustworthy?

A **potentially trustworthy URL** is one which either inherits context from its creator (`about:blank`, `about:srcdoc`, `data`) or one whose [origin](#) is a [potentially trustworthy origin](#). Given a [URL record](#) (*url*), the following algorithm returns "Potentially Trustworthy" or "Not Trustworthy" as appropriate:

1. If *url* is "about:blank" or "about:srcdoc", return "Potentially Trustworthy".
2. If *url*'s [scheme](#) is "data", return "Potentially Trustworthy".
3. Return the result of executing [§ 3.1 Is origin potentially trustworthy?](#) on *url*'s [origin](#).

Note: The origin of `blob:` URLs is the origin of the context in which they were created. Therefore, blobs created in a trustworthy origin will themselves be potentially trustworthy.

§ 4. Threat models and risks

This section is non-normative.

§ 4.1. Threat Models

Granting permissions to unauthenticated origins is, in the presence of a network attacker, equivalent to granting the permissions to any origin. The state of the Internet is such that we must indeed assume that a network attacker is present. Generally, network attackers fall into 2 classes: passive and active.

§ 4.1.1. Passive Network Attacker

A "Passive Network Attacker" is a party who is able to observe traffic flows but who lacks the ability or chooses not to modify traffic at the layers which this specification is concerned with.

Surveillance of networks in this manner "subverts the intent of communicating parties without the agreement of these parties" and one "cannot defend against the most nefarious actors while allowing monitoring by other actors no matter how benevolent some might consider them to be." [RFC7258] Therefore, the algorithms defined in this document require mechanisms that provide for the privacy of data at the application layer, not simply integrity.

§ 4.1.2. Active Network Attacker

An "Active Network Attacker" has all the capabilities of a "Passive Network Attacker" and is additionally able to modify, block or replay any data transiting the network. These capabilities are available to potential adversaries at many levels of capability, from compromised devices offering or simply participating in public wireless networks, to Internet Service Providers indirectly introducing security and privacy vulnerabilities while manipulating traffic for financial gain ([VERIZON] and [COMCAST] are recent examples), to parties with direct intent to compromise security or privacy who are able to target individual users, organizations or even entire populations.

§ 4.2. Ancestral Risk

The [secure context](#) algorithm walks through all the ancestors of a particular context in order to determine whether or not the context itself is secure. Why wouldn't we consider a securely-delivered document in an [<iframe>](#) to be secure, in and of itself?

The short answer is that this model would enable abuse. Chrome's implementation of [WEBCRYPTOAPI] was an early experiment in locking APIs to secure contexts, and it did not walk through a context's ancestors. The assumption was that locking the API to a resource which was itself delivered securely would be enough to ensure secure usage. The result, however, was that entities like Netflix built [<iframe>](#)- and `postMessage()`-based shims that exposed the API to non-secure contexts. The restriction was little more than a speed-bump, slowing down non-secure access to the API, but completely ineffective in preventing such access.

While the algorithms in this document do not perfectly isolate non-secure contexts from [secure contexts](#) (as discussed in [§ 5.1 Incomplete Isolation](#)), the ancestor checks provide a fairly robust protection for the guarantees of authentication, confidentiality, and integrity that such contexts ought to provide.

§ 4.3. Risks associated with non-secure contexts

Certain web platform features that have a distinct impact on a user's security or privacy should be available for use only in [secure contexts](#) in order to defend against the threats above. Features available in non-secure contexts risk exposing these capabilities to network attackers:

1. The ability to read and modify sensitive data (personally-identifying information, credentials, payment instruments, and so on). [CREDENTIAL-MANAGEMENT-1] is an example of an API that handles sensitive data.

2. The ability to read and modify input from sensors on a user's device (camera, microphone, and GPS being particularly noteworthy, but certainly including less obviously dangerous sensors like the accelerometer). [\[GEOLOCATION-API\]](#) and [\[MEDIACAPTURE-STREAMS\]](#) are historical examples of features that use sensor input.
3. The ability to access information about other devices to which a user has access. [\[DISCOVERY-API\]](#) and [\[WEB-BLUETOOTH\]](#) are good examples.
4. The ability to track users using temporary or persistent identifiers, including identifiers which reset themselves after some period of time (e.g. `window.sessionStorage`), identifiers the user can manually reset (e.g. [\[ENCRYPTED-MEDIA\]](#), Cookies [\[RFC6265\]](#), and [\[IndexedDB\]](#)), as well as identifying hardware features the user can't easily reset.
5. The ability to introduce some state for an origin which persists across browsing sessions. [\[SERVICE-WORKERS\]](#) is a great example.
6. The ability to manipulate a user agent's native UI in some way which removes, obscures, or manipulates details relevant to a user's understanding of their context. [\[FULLSCREEN\]](#) is a good example.
7. The ability to introduce some functionality for which user permission will be required.

This list is non-exhaustive, but should give you a feel for the types of risks we should consider when writing or implementing specifications.

Note: While restricting a feature itself to [secure contexts](#) is critical, we ought not forget that facilities that carry such information (such as new network access mechanisms, or other generic functions with access to network data) are equally sensitive.

§ 5. Security Considerations

§ 5.1. Incomplete Isolation

The [secure context](#) definition in this document does not completely isolate a "secure" view on an origin from a "non-secure" view on the same origin. Exfiltration will still be possible via increasingly esoteric mechanisms such as the contents of `localStorage/sessionStorage`, storage events, `BroadcastChannel`, and others.

§ 5.2. localhost

Section 6.3 of [\[RFC6761\]](#) lays out the resolution of `localhost.` and names falling within `.localhost.` as special, and suggests that local resolvers SHOULD/MAY treat them specially. For better or worse, resolvers often ignore these suggestions, and will send `localhost` to the network for resolution in a number of circumstances.

Given that uncertainty, user agents MAY treat `localhost` names as having [potentially trustworthy origins](#) if and only if they also adhere to the `localhost` name resolution rules spelled out in [\[let-localhost-be-localhost\]](#) (which boil down to ensuring that `localhost` never resolves to a non-loopback address).

§ 6. Privacy Considerations

The [secure context](#) definition in this document does not in itself have any privacy impact. It does, however, enable other features which do have interesting privacy implications to lock themselves into contexts which ensures that specific guarantees can be made regarding integrity, authenticity, and confidentiality.

From a privacy perspective, specification authors are encouraged to consider requiring secure contexts for the features they define.

§ 7. Implementation Considerations

§ 7.1. Packaged Applications

A user agent that support packaged applications MAY consider as "secure" specific URL schemes whose contents are authenticated by the user agent. For example, FirefoxOS application resources are referred to by a URL whose [scheme](#) component is `app:`. Likewise, Chrome's extensions and apps live on `chrome-extension:` schemes. These could reasonably be considered trusted origins.

§ 7.2. Development Environments

In order to support developers who run staging servers on non-loopback hosts, the user agent MAY allow users to configure specific sets of origins as trustworthy, even though [§ 3.1 Is origin potentially trustworthy?](#) would normally return "Not Trustworthy".

§ 7.3. Restricting New Features

This section is non-normative.

When writing a specification for new features, we recommend that authors and editors guard sensitive APIs with checks against [secure contexts](#). For example, something like the following might be a good approach:

EXAMPLE 16

1. If the [current settings object](#) is not a [secure context](#), then:
 1. *[insert something appropriate here: perhaps a Promise could be rejected with a `SecurityError`, an error callback could be called, a permission request denied, etc.].*

Authors could alternatively ensure that sensitive APIs are only exposed to [secure contexts](#) by guarding them with the [\[SecureContext\]](#) attribute.

```
[SecureContext]
interface SensitiveFeature {
    Promise<double> getTheSecretDouble();
};

// Or:

interface AnotherSensitiveFeature {
    [SecureContext] void doThatPowerfulThing();
};
```

§ 7.4. Restricting Legacy Features

This section is non-normative.

The list above clearly includes some existing functionality that is currently available to the web over non-secure channels. We recommend that such legacy functionality be modified to begin requiring a [secure context](#) as quickly as is reasonably possible [\[W3C-PROCESS\]](#).

1. If such a feature is not widely implemented, we recommend that the specification be immediately [modified](#) to include a restriction to [secure contexts](#).
2. If such a feature is widely implemented, but not yet in wide use, we recommend that it be quickly restricted to [secure contexts](#) by adding a check as described in [§ 7.3 Restricting New Features](#) to existing implementations, and [modifying the specification](#) accordingly.
3. If such a feature is in wide use, we recommend that the existing functionality be deprecated; the specification should be [modified](#) to note that it does not conform to the restrictions outlined in this document, and a plan should be developed to both offer a conformant version of the feature and to migrate existing users into that new version.

§ 7.4.1. Example: Geolocation

The [\[GEOLOCATION-API\]](#) is a good concrete example of such a feature; it is widely implemented and used on a large number of non-secure sites. A reasonable path forward might look like this:

1. [Modify](#) the specification to include checks against [secure context](#) before executing the algorithms for [getCurrentPosition\(\)](#) and [watchPosition\(\)](#).

If the [current settings object](#) is not a [secure context](#), then the algorithm should be aborted, and the errorCallback invoked with a code of PERMISSION_DENIED.

2. The user agent should announce clear intentions to disable the API for non-secure contexts on a specific date, and warn developers accordingly (via console messages, for example).
3. Leading up to the flag day, the user agent should announce a deprecation schedule to ensure both that site authors recognize the need to modify their code before it simply stops working altogether, and to protect users in the meantime. Such a plan might include any or all of:
 1. Disallowing persistent permission grants to non-secure origins
 2. Coarsening the accuracy of the API for non-secure origins (perhaps consistently returning city-level data rather than high-accuracy data)
 3. UI modifications to inform users and site authors of the risk

§ 8. Acknowledgements

This document is largely based on the Chrome Security team’s work on [\[POWERFUL-NEW-FEATURES\]](#). Chris Palmer, Ryan Sleevi, and David Dorwin have been particularly engaged. Anne van Kesteren, Jonathan Watt, Boris Zbarsky, and Henri Sivonen have also provided very helpful feedback.

§ Conformance

§ Document conventions

Conformance requirements are expressed with a combination of descriptive assertions and RFC 2119 terminology. The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in the normative parts of this document are to be interpreted as described in RFC 2119. However, for readability, these words do not appear in all uppercase letters in this specification.

All of the text of this specification is normative except sections explicitly marked as non-normative, examples, and notes. [\[RFC2119\]](#)

Examples in this specification are introduced with the words “for example” or are set apart from the normative text with `class="example"`, like this:

EXAMPLE 18

This is an example of an informative example.

Informative notes begin with the word “Note” and are set apart from the normative text with `class="note"`, like this:

Note, this is an informative note.

§ Conformant Algorithms

Requirements phrased in the imperative as part of algorithms (such as "strip any leading space characters" or "return false and abort these steps") are to be interpreted with the meaning of the key word ("must", "should", "may", etc) used in introducing the algorithm.

Conformance requirements phrased as algorithms or specific steps can be implemented in any manner, so long as the end result is equivalent. In particular, the algorithms defined in this specification are intended to be easy to understand and are not intended to be performant. Implementers are encouraged to optimize.

§ Index

§ Terms defined by this specification

[potentially trustworthy origin](#), in § 3.1

[potentially trustworthy URL](#), in § 3.2

§ Terms defined by reference

[GEOLOCATION-API] defines the following terms:

getCurrentPosition()

watchPosition()

[HTML] defines the following terms:

SharedWorker

WindowOrWorkerGlobalScope

current settings object

domain

environment

global object

host

iframe

isSecureContext

non-secure context

opaque origin

origin

port

relevant settings object

scheme

secure context

top-level browsing context

tuple origin

[MIX] defines the following terms:

a priori authenticated url

[URL] defines the following terms:

origin

scheme

url record

[W3C-PROCESS] defines the following terms:

modify a specification

[WebIDL] defines the following terms:

DOMException

SecureContext

SecurityError

exposed

§ References

§ Normative References

[HTML]

Anne van Kesteren; et al. *HTML Standard*. Living Standard. URL: <https://html.spec.whatwg.org/multipage/>

[LET-LOCALHOST-BE-LOCALHOST]

Mike West. *Let 'localhost' be localhost*. URL: <https://tools.ietf.org/html/draft-west-let-localhost-be-localhost>

[RFC2119]

S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. Best Current Practice. URL: <https://datatracker.ietf.org/doc/html/rfc2119>

[RFC4632]

V. Fuller; T. Li. *Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan*. August 2006. Best Current Practice. URL: <https://www.rfc-editor.org/rfc/rfc4632>

[URL]

Anne van Kesteren. *URL Standard*. Living Standard. URL: <https://url.spec.whatwg.org/>

[W3C-PROCESS]

Charles McCathie Nevile. *World Wide Web Consortium Process Document*. URL: <https://www.w3.org/20145/Process-20150901/>

[WebIDL]

Boris Zbarsky. *Web IDL*. 15 December 2016. ED. URL: <https://heycam.github.io/webidl/>

§ Informative References

[COMCAST]

David Kravets. *Comcast Wi-Fi serving self-promotional ads via JavaScript injection*. URL: <https://arstechnica.com/tech-policy/2014/09/why-comcasts-javascript-ad-injections-threaten-security-net-neutrality/>

[CREDENTIAL-MANAGEMENT-1]

Mike West. *Credential Management Level 1*. 17 January 2019. WD. URL: <https://www.w3.org/TR/credential-management-1/>

[DISCOVERY-API]

Rich Tibbett. *Network Service Discovery*. 12 January 2017. NOTE. URL: <https://www.w3.org/TR/discovery-api/>

[ENCRYPTED-MEDIA]

David Dorwin; et al. *Encrypted Media Extensions*. 18 September 2017. REC. URL: <https://www.w3.org/TR/encrypted-media/>

[FULLSCREEN]

Philip Jägenstedt. *Fullscreen API Standard*. Living Standard. URL: <https://fullscreen.spec.whatwg.org/>

[GEOLOCATION-API]

Andrei Popescu. *Geolocation API Specification 2nd Edition*. 8 November 2016. REC. URL: <https://www.w3.org/TR/geolocation-API/>

[IndexedDB]

Nikunj Mehta; et al. *Indexed Database API*. 8 January 2015. REC. URL: <https://www.w3.org/TR/IndexedDB/>

[MEDIACAPTURE-STREAMS]

Cullen Jennings; et al. *Media Capture and Streams*. 19 August 2021. CR. URL: <https://www.w3.org/TR/mediacapture-streams/>

[MIX]

Emily Stark; Mike West; Carlos IbarraLopez. *Mixed Content Level 2*. 18 September 2021. CR. URL: <https://www.w3.org/TR/mixed-content/>

[POWERFUL-NEW-FEATURES]

Chrome Security Team. *Prefer Secure Origins For Powerful New Features*. URL: <https://www.chromium.org/Home/chromium-security/prefer-secure-origins-for-powerful-new-features>

[RFC6265]

A. Barth. *HTTP State Management Mechanism*. April 2011. Proposed Standard. URL: <https://httpwg.org/specs/rfc6265.html>

[RFC6761]

S. Cheshire; M. Krochmal. *Special-Use Domain Names*. February 2013. Proposed Standard. URL: <https://www.rfc-editor.org/rfc/rfc6761>

[RFC7258]

S. Farrell; H. Tschofenig. *Pervasive Monitoring Is an Attack*. May 2014. Best Current Practice. URL: <https://www.rfc-editor.org/rfc/rfc7258>

[SECURING-WEB]

Mark Nottingham. *Securing the Web*. TAG Finding. URL: <https://www.w3.org/2001/tag/doc/web-https>

[SERVICE-WORKERS]

Alex Russell; et al. *Service Workers 1*. 19 November 2019. CR. URL: <https://www.w3.org/TR/service-workers-1/>

[VERIZON]

Mark Bergen; Alex Kantrowitz. *Verizon looks to target its mobile subscribers with ads*. URL: <http://adage.com/article/digital/verizon-target-mobile-subscribers-ads/293356/>

[WEB-BLUETOOTH]

Jeffrey Yasskin. *Web Bluetooth*. Draft Community Group Report. URL: <https://webbluetoothcg.github.io/web-bluetooth/>

[WEBCRYPTOAPI]

Mark Watson. *Web Cryptography API*. 26 January 2017. REC. URL: <https://www.w3.org/TR/WebCryptoAPI/>