July 14, 2026

Web Application Integrity, Consistency and Transparency

Anna Weine

Mozilla

Exercise 0: An application

- Imagine you're a web developer who wants to build his application

Exercise 0: An application

- Imagine you're a web developer who wants to build his application

- Running the app:
    - cd exercises/exercise-1-breaking-the-app
    - node server.js
    - open http://localhost:3000/

Do you see a cat?

CedarCrypt

Purveyors of fine felines since 2026.

Meet our mascot. Look how cute!

Admin console

Exercise 0: Discussion: What could go wrong?

- Imagine you're an attacker. What would you like to do?

Exercise 0: Discussion: What could go wrong?

- Imagine you're an attacker. What would you like to do?

- What are the possible attacks?

Exercise 0: Discussion: What's simple to set up?

- Network-level attacker
- Compromised origin server (stolen credentials)
- Compromised CDN/third-party host
- Local malware
- CA compromise

Exercise 1: breaking an app

- What do you see in the browser?

Now imagine that you are an attacker

Go to an administration panel and try to log-in

Our goal is to replace a cat image with an evil image

Exercise 1: breaking an app

Now imagine that you are an attacker

Go to an administration panel and try to log-in

Our goal is to replace a cat image with an evil image

If you know how hashcat works or/and don’t care about hash breaking, go check the source code for the admin password

A gentle introduction to hashcat

- Hashcat = GPU-accelerated password cracking tool
- Given a hash, it guesses inputs and re-hashes them until it finds a match
- Two main modes: dictionary attack (-a 0, tries a wordlist) and brute-force (-a 3, tries all combinations)
- -m sets the hash type (e.g. 1400 = SHA256),
- Used for security audits & pentesting

Hashcat - Disclaimer

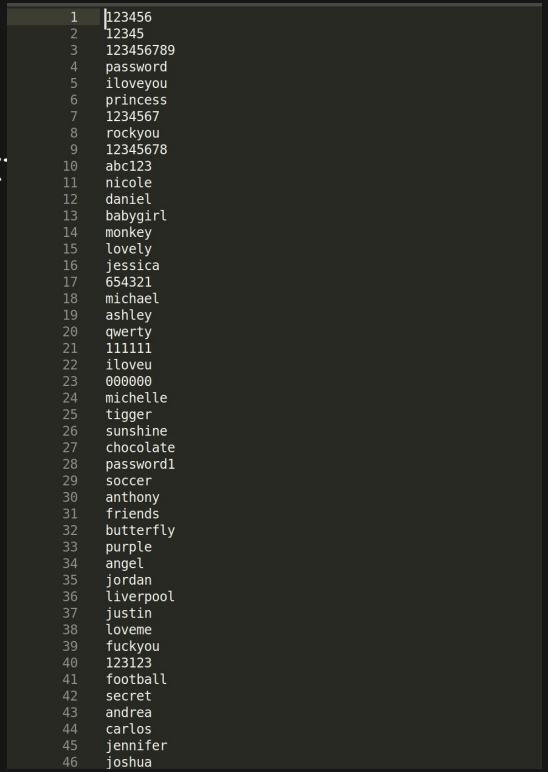
Unauthorized access to computer systems, or attempting to crack credentials you don't own or have permission to test, is illegal in most jurisdictions (e.g. under the Computer Fraud and Abuse Act in the US, the Computer Misuse Act in the UK, or equivalent laws elsewhere) and can carry serious criminal penalties.

If you're unsure whether you have authorization — don't run it.

Try it

```
echo -n "123" | sha256sum | awk '{print $1}' > hash.tx
```

```
hashcat -m 1400 -a 0 hash.txt wordlist
```



```
123456
12345
123456789
password
iloveyou
princess
1234567
rockyou
12345678
abc123
nicole
daniel
babygirl
monkey
lovely
jessica
654321
michael
ashley
qwerty
111111
iloveu
000000
michelle
tigger
sunshine
chocolate
password1
soccer
anthony
friends
butterfly
purple
angel
jordan
liverpool
justin
loveme
fuckyou
123123
football
secret
andrea
carlos
jennifer
joshua
```

Try it - break into the app using hashcat

```
echo -n "123" | sha256sum | awk '{print $1}' > hash.txt

hashcat -m 1400 -a 0 hash.txt wordlist


hashcat ... {your hash here} wordlist
```

Try it - break into the app using hashcat

From our admin “console” change the image to the “evil” one :)

Then update the page and check in the browser

What we want to get?

CedarCrypt

Purveyors of fine felines since 2026.

⚠️ This is NOT the cat you were promised.

We all agree that no one should do it, right?

```
 * ------------------------------------------------------------- */
const ADMIN_USER = 'admin';
// FIXME(dev): temporary creds for local testing -> admin / sunshine
//             rotate this and move it to a secret store before shipping!!
const ADMIN_PASSWORD_SHA256 =
  'a941a4c4fd0c01cddef61b8be963bf4c1e2b0811c037ce3f1835fddf6ef6c223';
```

Side quest: why SHA-256 is bad for passwords?

Side quest: why SHA-256 is bad for passwords?

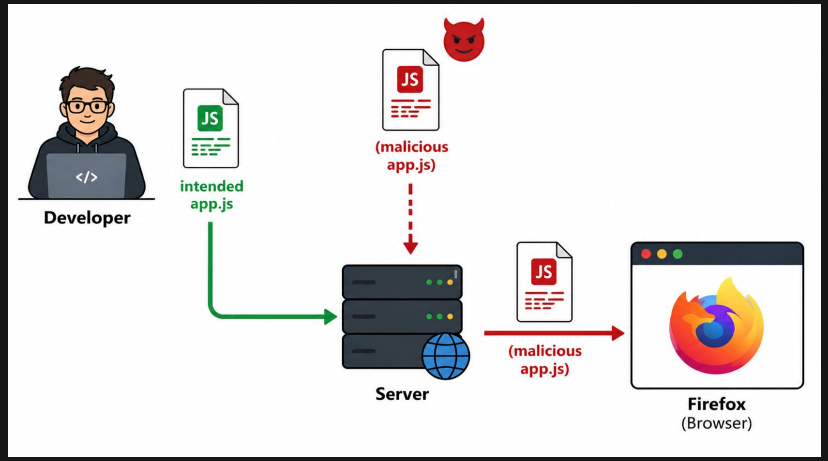
Lesson to learn:

- The same cryptographic primitive can be an excellent tool in one context and the wrong tool in another. The security property you need determines the construction you should use.

Exercise 1: Discussion

- Could the potential user observe that the application got “hacked”?
- Could the browser distinguish a genuine page from a hacked one?
- How can we protect the user in such case? Propose your ideas!

The Browser Trusted the Wrong JavaScript



What existing mechanisms do?

Mechanism	What it protects	Limitations
HTTPS	Connection to the server	

What existing mechanisms do

Mechanism	What it protects	Limitations
HTTPS	Connection to the server	Does not protect against a malicious or compromised server

What existing mechanisms do

Mechanism	What it protects	Limitations
HTTPS	Connection to the server	Does not protect against a malicious or compromised server
CSP		

CSP aka Content Security Policy

Content Security Policy (CSP) is a feature that helps to prevent or minimize the risk of certain types of security threats. It consists of a series of instructions from a website to a browser, which instruct the browser to place restrictions on the things that the code comprising the site is allowed to do.

The primary use case for CSP is to control which resources, in particular JavaScript resources, a document is allowed to load. This is mainly used as a defense against cross-site scripting (XSS) attacks, in which an attacker is able to inject malicious code into the victim's site.

CSP - Example

Content-Security-Policy: default-src 'self'; img-src 'self' example.com

The default-src directive restricts what URLs resources can be fetched from the document that set the Content-Security-Policy header.

CSP - Example

Content-Security-Policy: default-src 'self'; img-src 'self' example.com

The default-src directive restricts what URLs resources can be fetched from the document that set the Content-Security-Policy header.

We have set the default-src directive to `self` which means the **same** origin, or **same** domain and scheme.

CSP - Example

Content-Security-Policy: default-src 'self'; img-src 'self' example.com

The default-src directive restricts what URLs resources can be fetched from the document that set the Content-Security-Policy header.

We have set the default-src directive to `self` which means the **same** origin, or **same** domain and scheme.

In this case we are allowing images to be loaded from 'self' and the domain cdn.example.com.

What existing mechanisms do

Mechanism	What it protects	Limitations
HTTPS	Connection to the server	Does not protect against a malicious or compromised server
CSP	Where scripts may come from	

What existing mechanisms do

Mechanism	What it protects	Limitations
HTTPS	Connection to the server	Does not protect against a malicious or compromised server
CSP	Where scripts may come from	Doesn't verify the contents of same-origin scripts

What existing mechanisms do

Mechanism	What it protects	Limitations
HTTPS	Connection to the server	Does not protect against a malicious or compromised server
CSP	Where scripts may come from	Doesn't verify the contents of same-origin scripts
SRI		

Subresource Integrity aka SRI

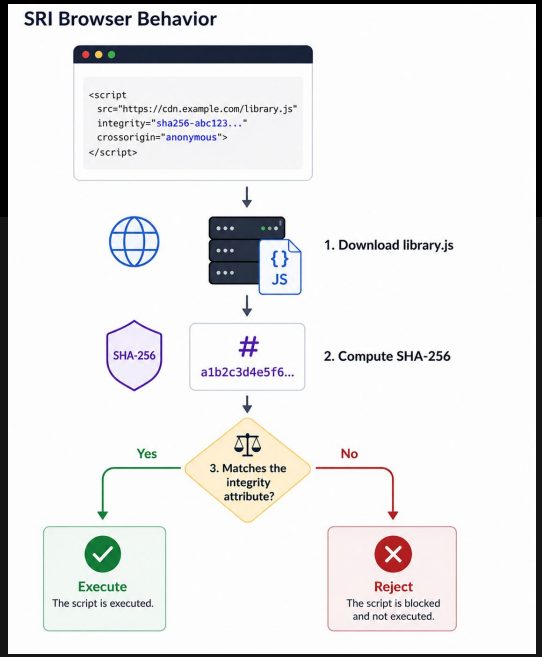
SRI lets a developer tell the browser:

> "This particular file should
> have this exact hash."

```
<script
  src="https://cdn.example.com/library.js"
  integrity="sha256-abc123..."
  crossorigin="anonymous">
</script>
```

Subresource Integrity aka SRI

```
<script
  src="https://cdn.example.com/library.js"
  integrity="sha256-abc123..."
  crossorigin="anonymous">
</script>
```



What existing mechanisms do

Mechanism	What it protects	Limitations
HTTPS	Connection to the server	Does not protect against a malicious or compromised server
CSP	Where scripts may come from	Doesn't verify the contents of same-origin scripts
SRI	Individual External Resources	

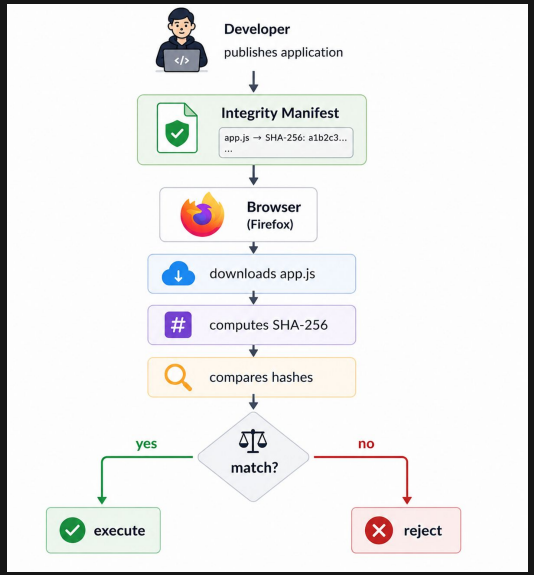
WAICT: Web Application Integrity, Consistency and ~~Unicorns~~

WAICT: Web Application Integrity, Consistency and ~~Unicorns~~ Transparency

WAICT enables a browser to verify that the resources it receives belong to the version of the application published by the developer.

WAICT: Web Application Integrity, Consistency and Transparency

WAICT enables a browser to verify that the resources it receives belong to the version of the application published by the developer.



Exercise 2: Implementing Integrity

Part A - Inspect the application

Exercise 2: Implementing Integrity

Our application contains:

- index.html
- app.js
- style.css
- cat.jpg

What do we want to protect?

Exercise 2: Implementing Integrity

Part B: Generate hashes

Using your favourite tool, compute SHA-256 hash of the application files

Exercise 2: Implementing Integrity

Part B: Generate hashes

Or use any of these:

```
openssl dgst -sha256 -binary app.js | openssl base64
```

```
sha256sum app.js | cut -d' ' -f1 | xxd -r -p | base64
```

```
node -e
"console.log(require('crypto').createHash('sha256').update(require('fs').readFileSync('app.js')).
digest('base64'))"
```

Exercise 2: Implementing Integrity

Correct hashes:

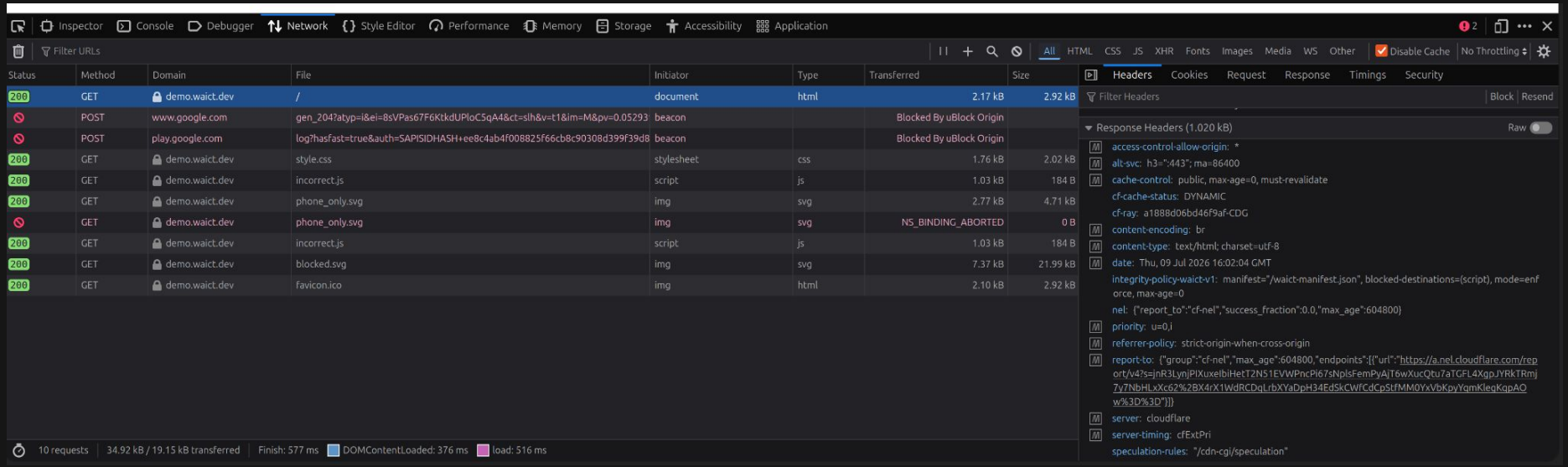
- Index.html - RBSSCxiScRS2iwhN1nZfXj6cjP1LKB+ekhDWYh0pHUk=
- app.js - TmqLBM6xXW5qxKVU0WoN919oikqzLXv+e3RKF6uK45s=
- style.css - 8n9/U26gAHWUmRavlixzWmiHScYL6T8/4w04klIA7xM=
- cat.png - D2u5FviCajXhQXP/E0RDRRMyyEiot2zYp8T4G+9VunM=

Exercise 2: Implementing Integrity

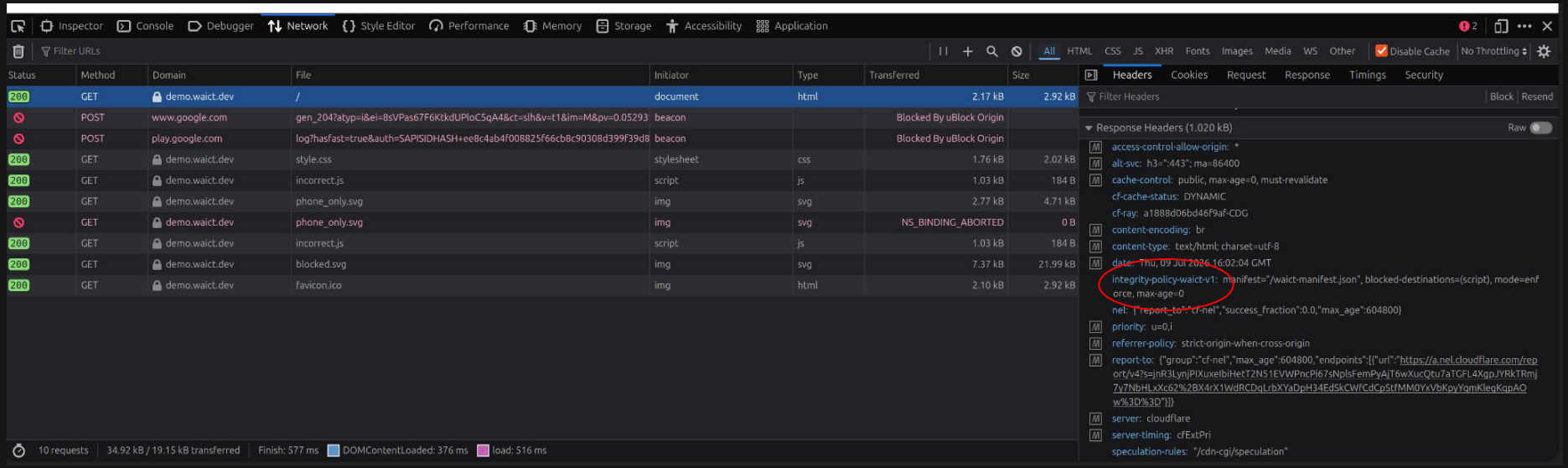
- Our task now will be to implement WAICT support for the application:
  - Implementation of WAICT header
  - Implementation of WAICT manifest
  - Testing

Exercise 2: Integrity - HTTP Response

Go to demo.waict.dev



Exercise 2: Integrity - HTTP Response



WAICT is just another security policy header.

Exercise 2: WAICT Header

```
Integrity-Policy-WAICT-v1:
        max-age=0,
        mode=enforce, // or report
        blocked-destinations=(script),
        manifest="./waict-manifest.json"
```

Exercise 2: Add a new header to your application

Your task is:

- Add the new header to the nodejs app
  - Take a look at server.js file
- Restart the server
- Verify that the header is present (Use FF devtools or curl -I
- And don't worry about the manifest file for now)

```
Integrity-Policy-WAICT-v1:
  max-age=0,
  mode=enforce, // or report
  blocked-destinations=(script),

manifest="./waict-manifest.json"
```

```
curl -I http://localhost:3000
```

Exercise 2: Implementing integrity - Adding the manifest


```
{
  "hashes": {
    "/resources/correct.webp": "04voAZyqjRJGy0QUzKL+2Qnz4gq/s4sqQVquTHW8Ob4=",
    "/resources/incorrect.webp": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=",
    "/resources/correct.js": "Lfy9VlsQurFkeKqdqlvYCv8KSiuFhh0Aq9T4rVPzNpo=",
    "/resources/incorrect.js": "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB=",
    "/resources/fox.webp": "UVaQ+gl+jjdqoesrDu+Qbf2TGIdWWoG2QpWuAjt7Cik=",
    "/resources/phone_only.svg": "gcT4todkqXUzI49+yt6jJpjoCmCO5Dvwv1sNd9tZ614=",
    "/resources/blocked.svg": "gt4aBLooDNzuDyQZSJqua1nwIFk+bV1MzI0f7Jmb+XU="
  }
}
```

Exercise 2: Implementing integrity - Adding the manifest

Your task is

- To generate a manifest file
- Add it to your application
- Rebuild the application
- Open it in the browser

Exercise 2: Adding the manifest

Your task is

- To generate a manifest file
- Add it to your application
- Rebuild the application
- Open it in the browser

- Then change the manifest to contain a wrong hash
- Rebuild

Exercise 2: Adding the manifest

Your task is

- To generate a manifest file
- Add it to your application
- Rebuild the application
- Open it in the browser

- Then change the manifest to contain a wrong hash
- Rebuild

Look at the exercise-2 solutions if you're stuck

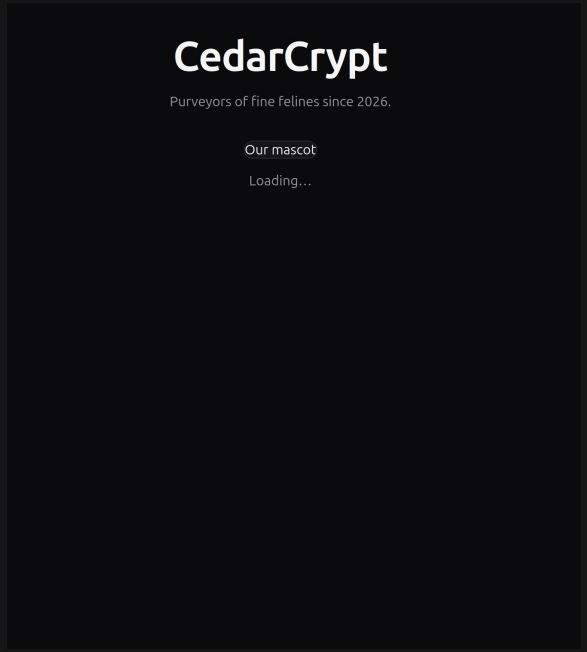
This is how your page should look like if all worked

CedarCrypt

Purveyors of fine felines since 2026.

Meet our mascot. Look how cute!

And this - if not

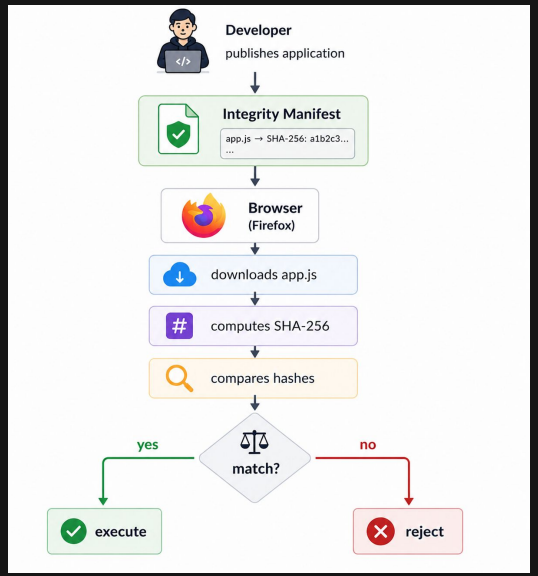


Victory!

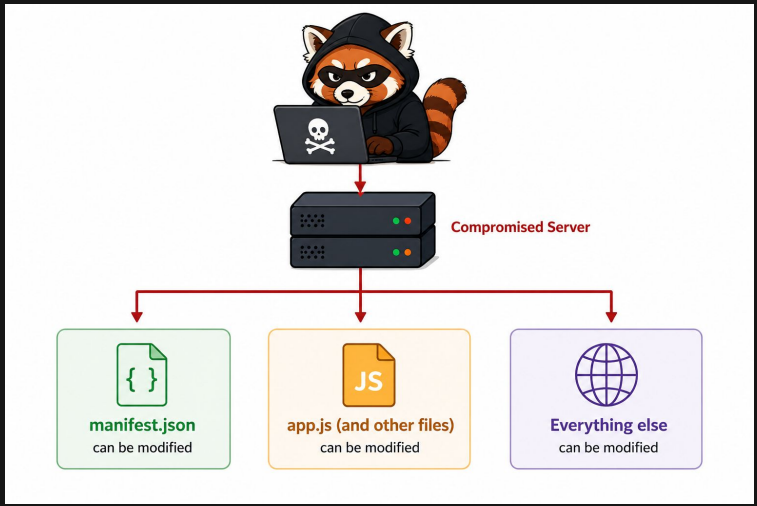
Coffee/water/bathroom break

Victory?

Discussion: The browser now knows where the manifest is and what hashes to expect. Is the application protected?



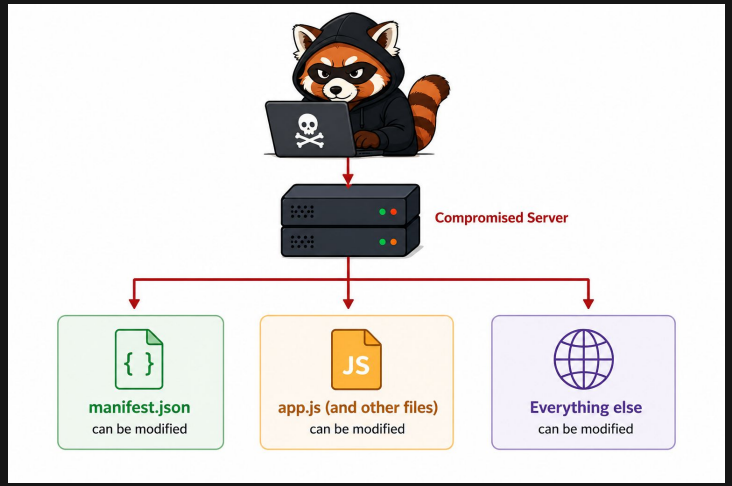
Compromising the application - a stronger attacker



Compromising the application - a stronger attacker

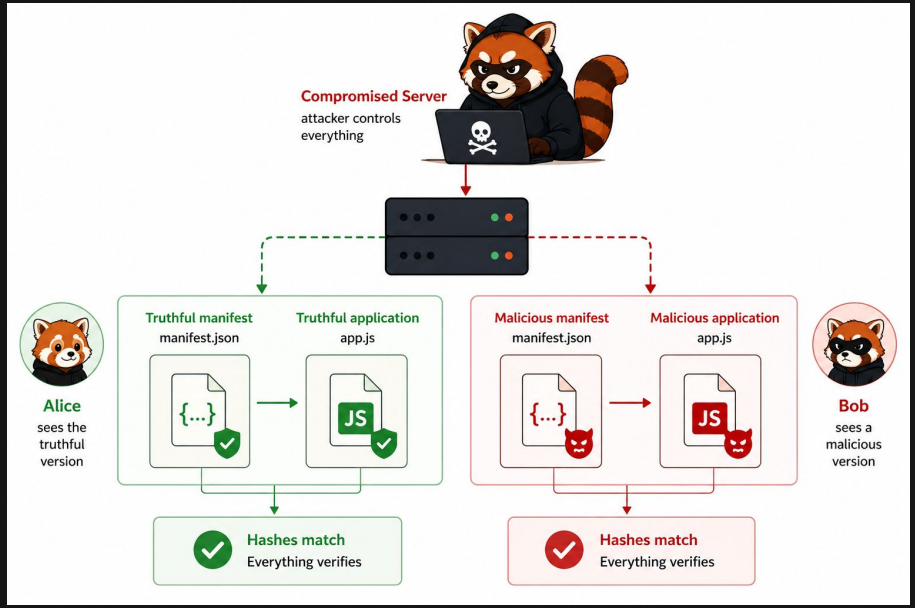
Can the browser detect that the application it received is not the one the developer intended?

What else can we do?



Integrity is not enough (otherwise the approach would be called WAI)

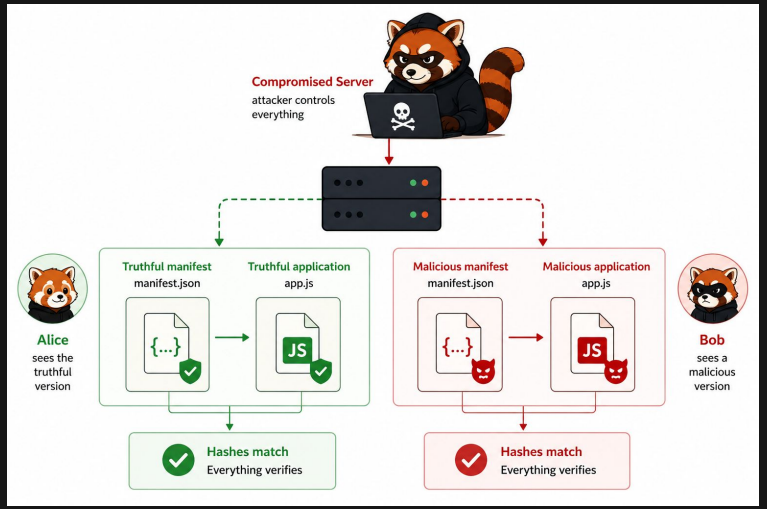
How Can the Browser Know This Is the "Right" Manifest?



Consistency

Consistency means that all users receive the same published application version.

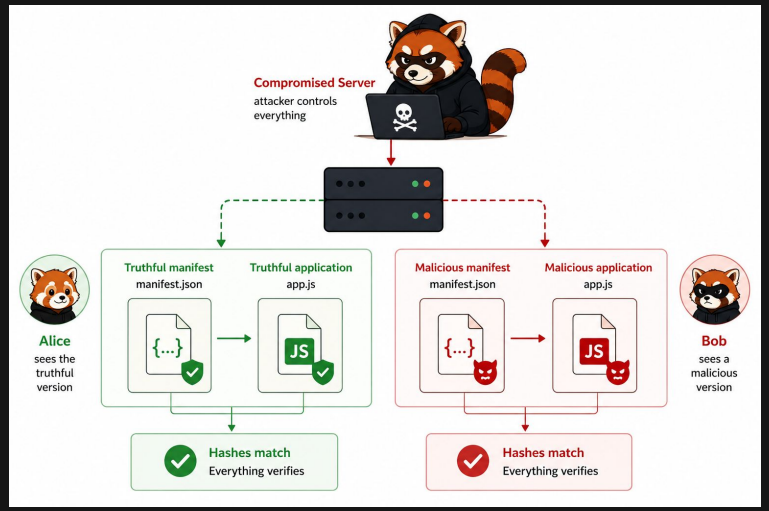
But why to care?



Consistency

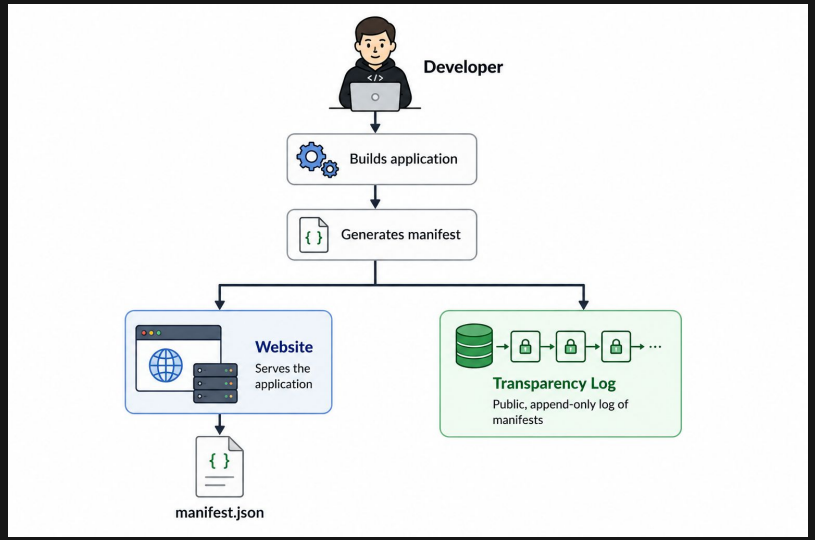
Consistency means that all users receive the same published application version.

And if we care, how do we check that the applications are consistent?



Transparency

Transparency means that every published application version is publicly visible and auditable.



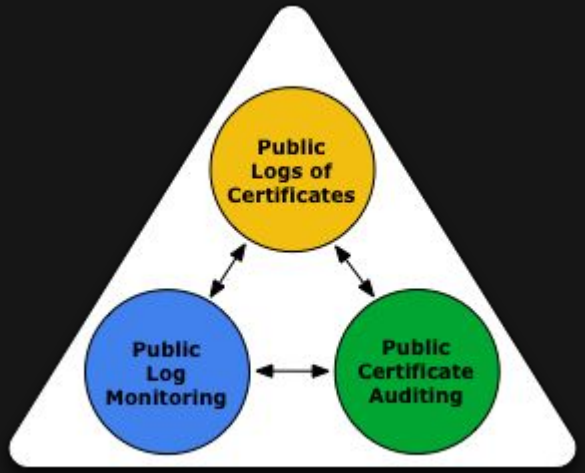
Transparency

gives us:

- Publicly verifiable log such that
  - Everyone can see it
  - Everyone can inspect it
  - Monitors can raise alerts
  - It can not exist only for one victim

Transparency Isn't a New Idea

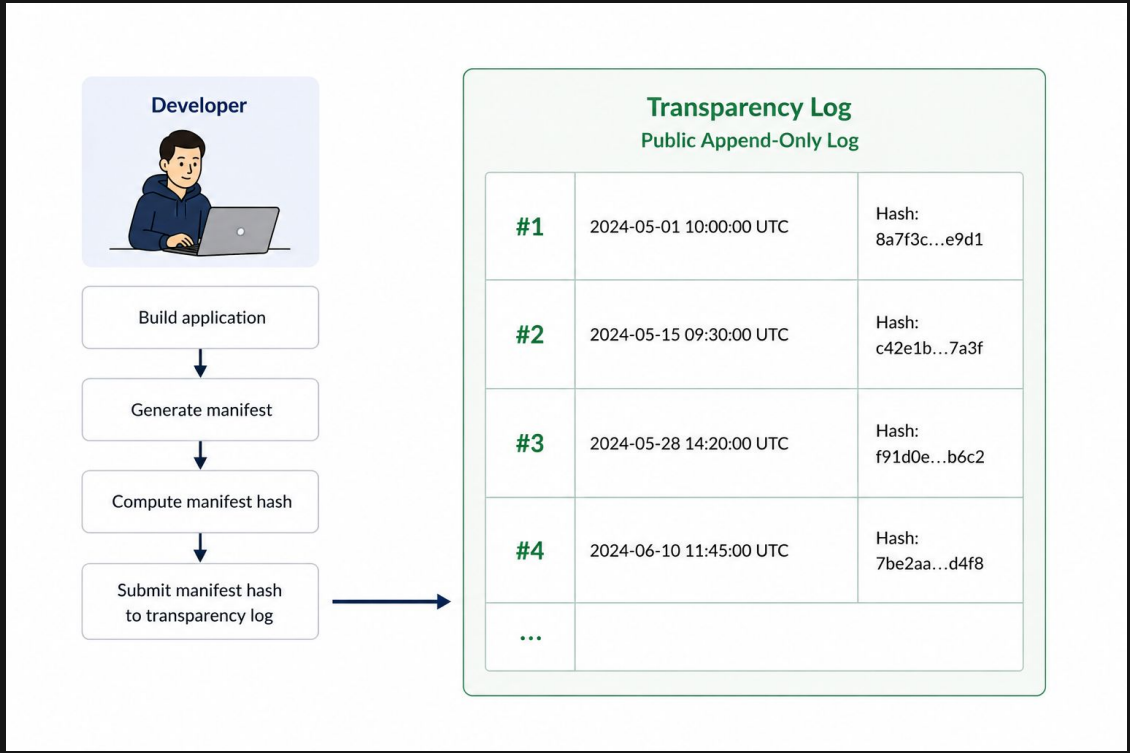
Since 2018, publicly trusted TLS certificates have been required to be logged in Certificate Transparency (CT) logs before browsers trust them.



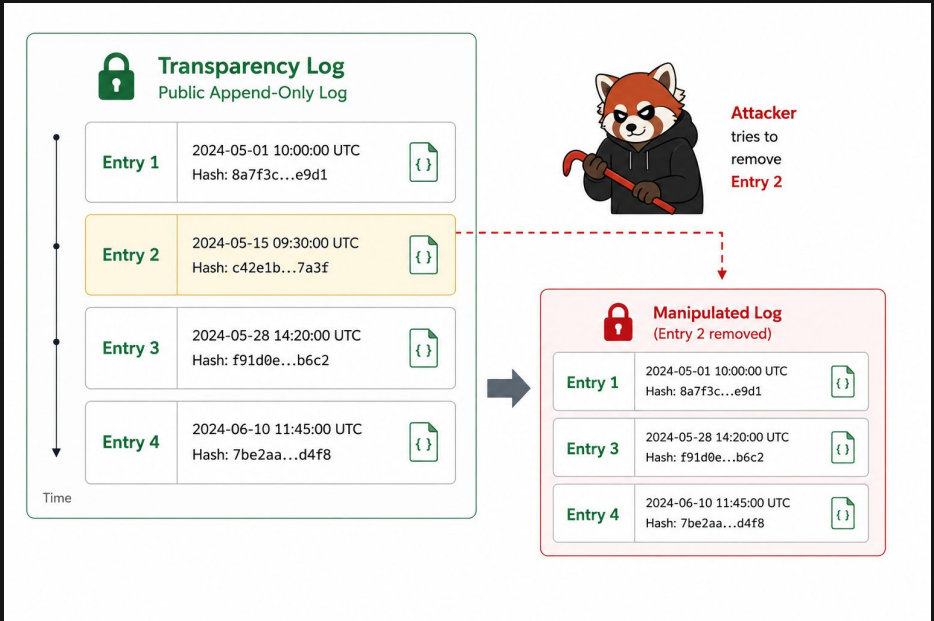
Transparency Isn’t a New Idea

Certificate Transparency	WAICT Transparency
Logs TLS certificates	Logs application manifests
Protects the public key infrastructure	Protects web application integrity
Detects unexpected certificates	Detects unexpected application versions
Public append-only log	Public append-only log

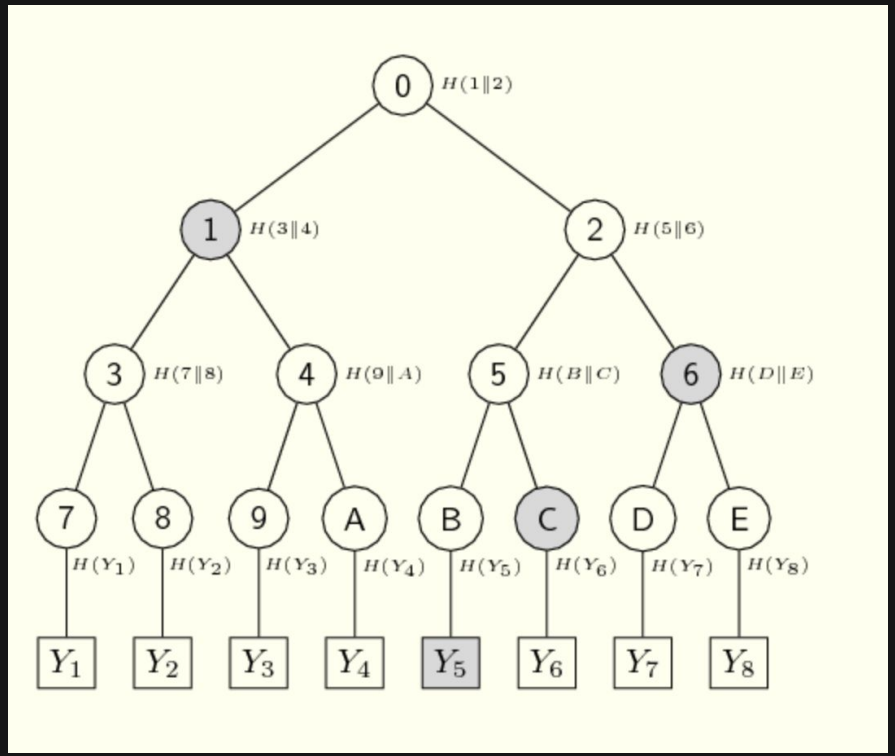
Public append-only log



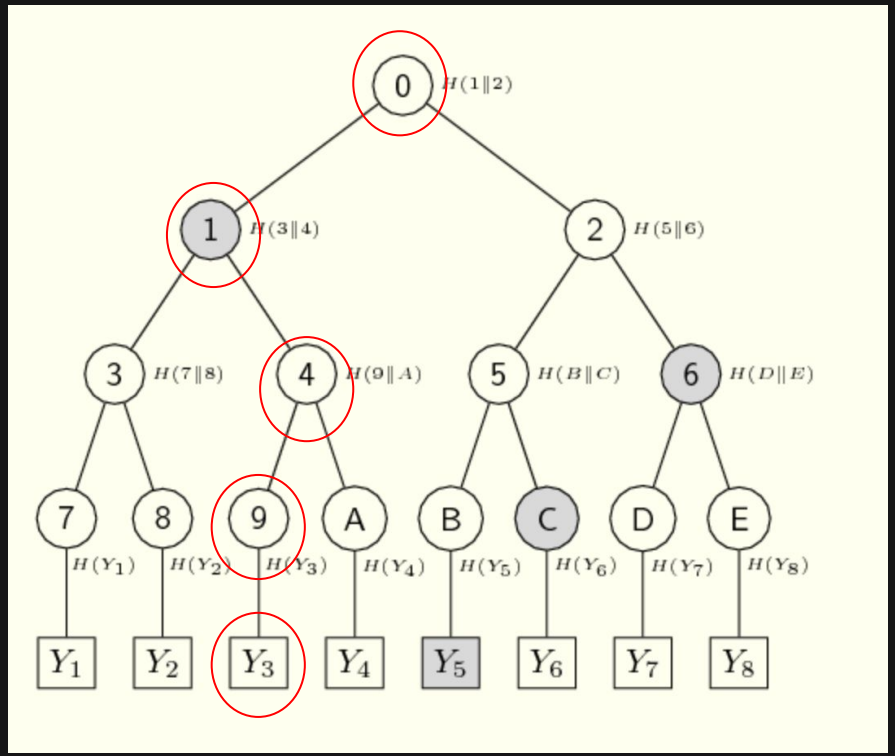
What stops the operator from deleting an embarrassing entry?



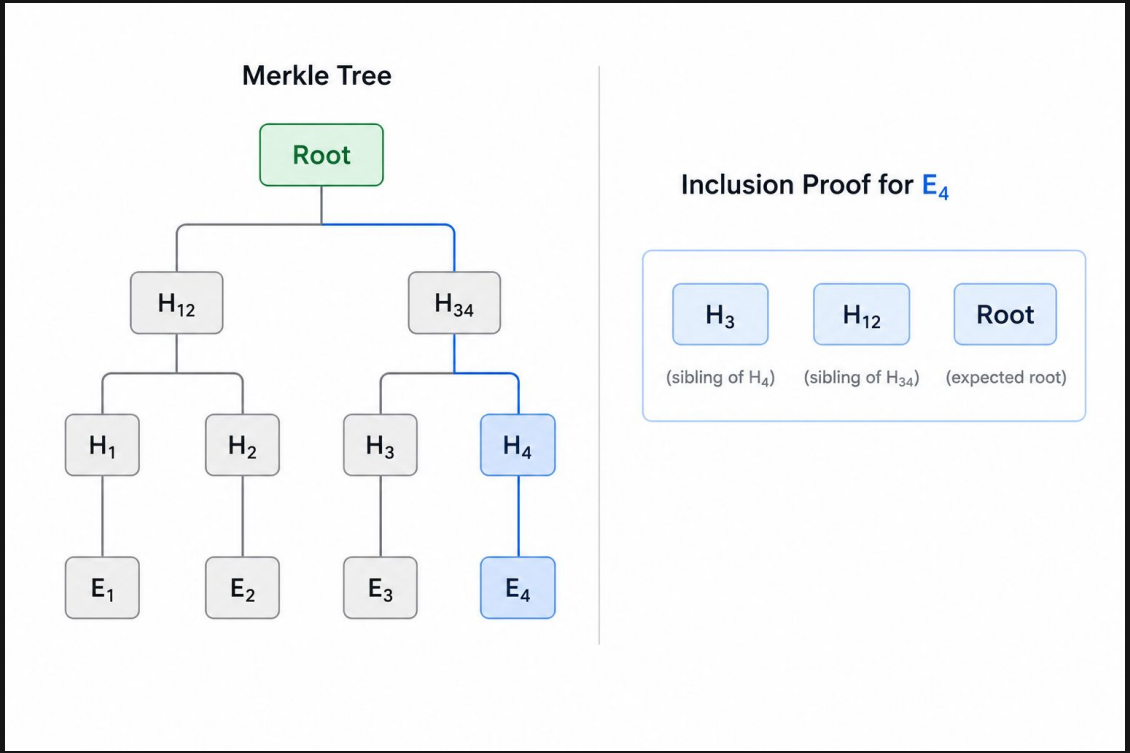
Merkle Tree


0
$H(1\|2)$
1
$H(3\|4)$
2
$H(5\|6)$
3
$H(7\|8)$
4
$H(9\|A)$
5
$H(B\|C)$
6
$H(D\|E)$
7
8
9
A
B
C
D
E
$H(Y_1)$
$H(Y_2)$
$H(Y_3)$
$H(Y_4)$
$H(Y_5)$
$H(Y_6)$
$H(Y_7)$
$H(Y_8)$
Y_1
Y_2
Y_3
Y_4
Y_5
Y_6
Y_7
Y_8

Merkle Tree

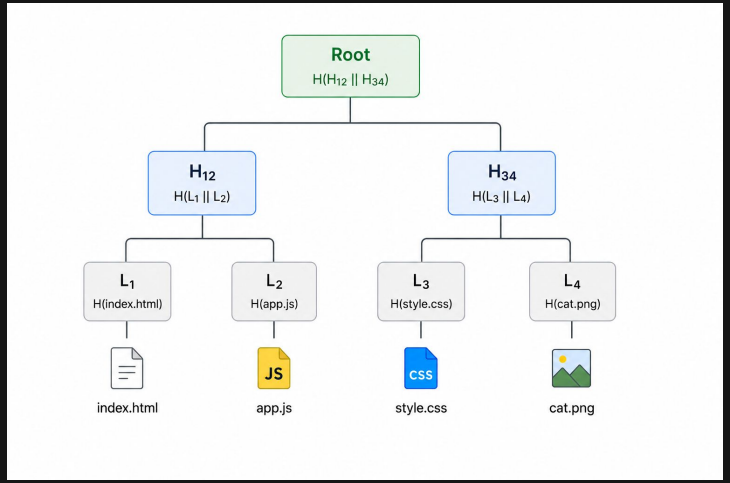

0
$H(1\|2)$
1
$H(3\|4)$
2
$H(5\|6)$
3
$H(7\|8)$
4
$H(9\|A)$
5
$H(B\|C)$
6
$H(D\|E)$
7
8
9
A
B
C
D
E
$H(Y_1)$
$H(Y_2)$
$H(Y_3)$
$H(Y_4)$
$H(Y_5)$
$H(Y_6)$
$H(Y_7)$
$H(Y_8)$
Y_1
Y_2
Y_3
Y_4
Y_5
Y_6
Y_7
Y_8

Inclusion proof

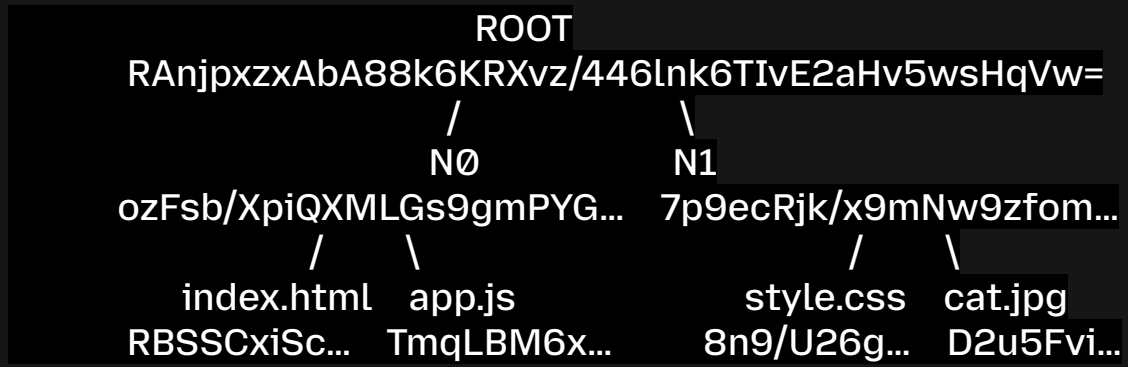


Exercise: Compute the Merkle Tree Root

Using the file hashes, compute the Merkle Tree root



Exercise: Compute the Merkle Tree root



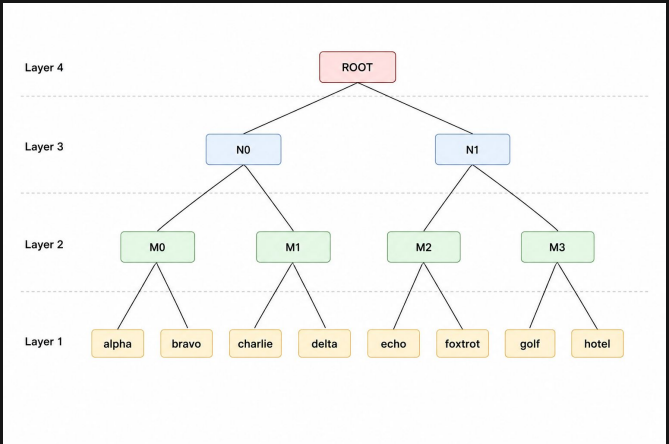
Exercise: Transparency verification

Received leaves — is it correct?

file.html 2h3r3J5EJsxPfwqx1S2LVQh15a2pysdp3vZf26bGb6Q=
newApp.js emHIRt51YcEu4V6fltCFUpjiOEZqcNv7R9kvkRAQbaM
newStyle.css lsqVdOL8JIQTk7d8KE27t1qHD9KpqmCrEvH8Azn+pIk=
AnotherCat.jpg QVnYzCmPQWJdT7C5Ouy59T4l3lkV2UKQTDor1O7K8UU=

Published root:
jHA5trmbuG4hk14KA4MH7KYibqtYWtBWvuEHCJQaU48=

Exercise: Transparency verification



Received inclusion proof — is it correct?

leaf foxtrot
ziDVfHb5JNtmWBbwtYEI8zI2IRZLzHsWTPh++V/klCY=

leaf echo
kQK/wfZTw40Tlof9ALVL60K1j8gp/4wMy927tVxIiJQ=

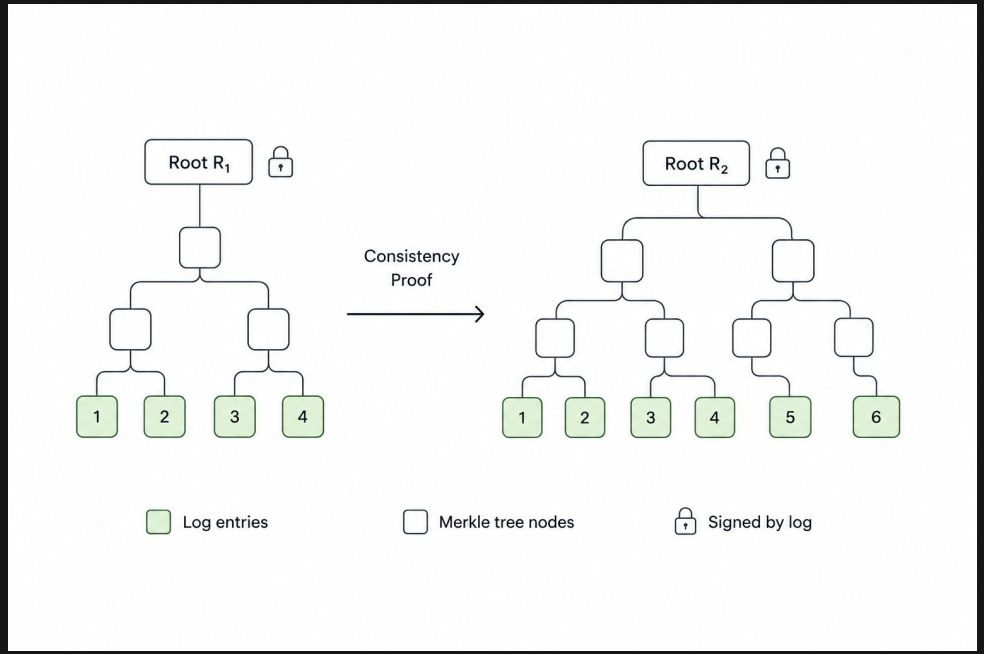
M3
myqAO3gjm+DH87YYa4LXYmlqCy74DS/HE1pKFa149Lk=

N0
p3apCNd035VD8quaSewFFl5HD5O9AW5MjkCd33hMDGU=

Published root:
hFF2zKpIOZJH6YAcuRWtxnmnPqLX/2qzvZ5A6L10v58=

Consistency proof

Has the log only grown, or has its history changed?

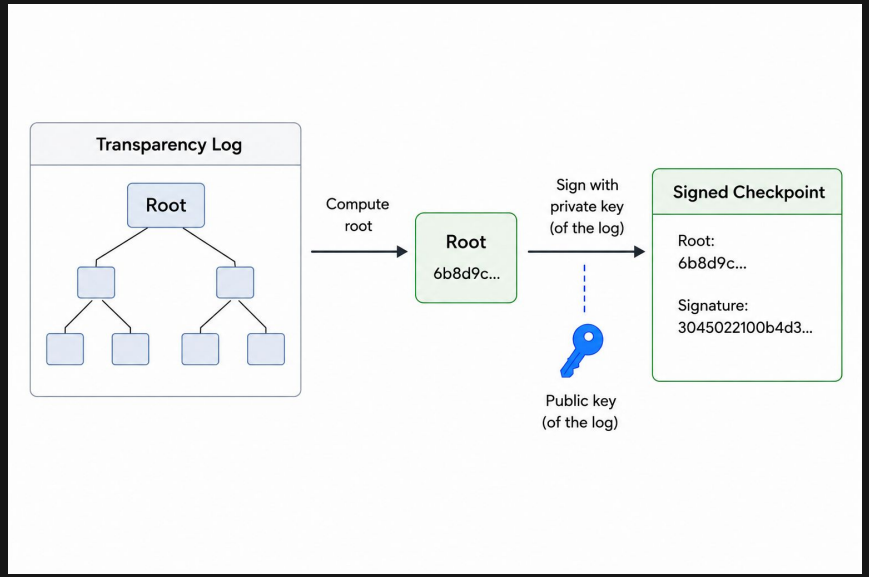


Why Can the Browser Trust the Root?

How can we make sure that the root was not replaced?

Why Can the Browser Trust the Root?

How can we make sure that the root was not replaced?



Why Can the Browser Trust the Public Key?

If the browser uses the public key to verify the signature, who verifies the public key?

Minor take away

What Does WAICT Protect Against?

Attack	Protected by WAICT?
CDN compromise	Yes
Build server compromise	Yes, detectable
Malicious insider	Yes, detectable
Man-in-the-middle (network attacker)	No (HTTPS already)
Cross-site scripting (XSS)	No
SQL injection	No
Stolen user password	No

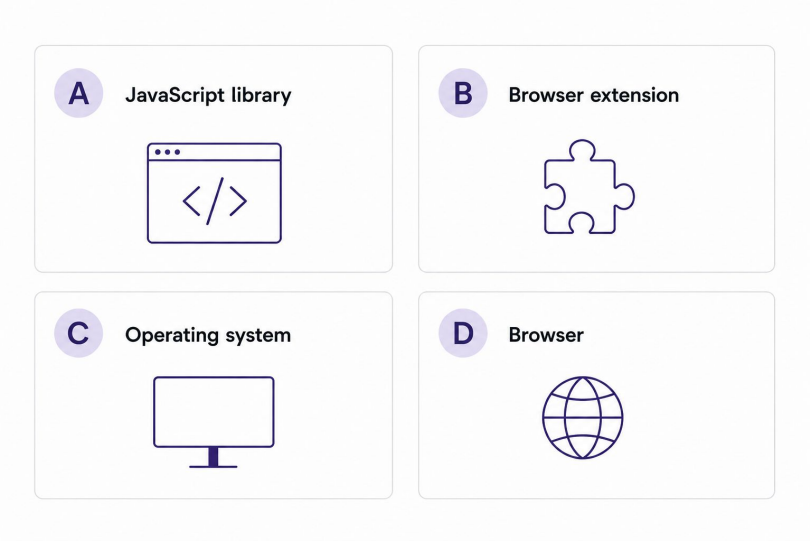
Main take away - Why WAICT?

Modern web applications increasingly behave like installed software.

Yet browsers have no way to verify that users receive the application the developer intended.

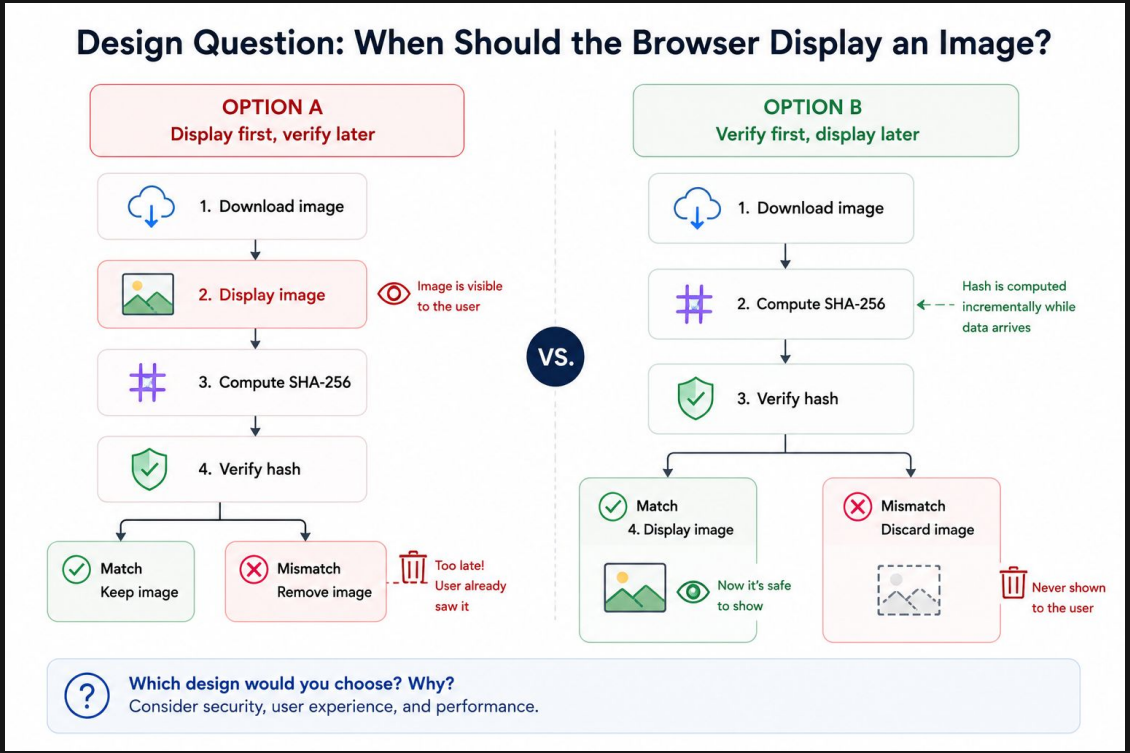
WAICT brings software-update style integrity and transparency to the Web.

Where Should WAICT Be Implemented?



How does WAICT work in Firefox?

The request lifecycle

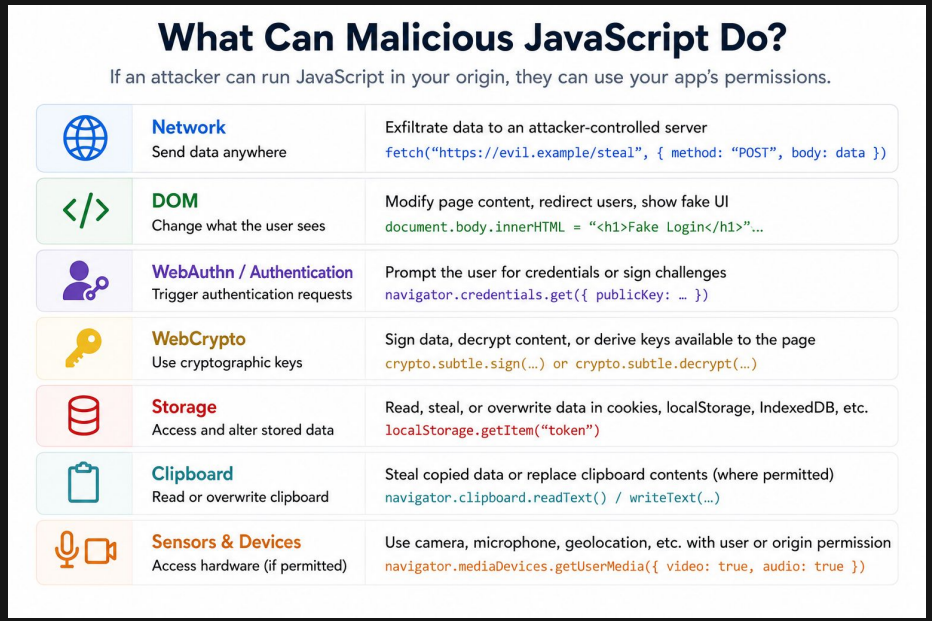

Design Question: When Should the Browser Display an Image?
OPTION A
Display first, verify later
1. Download image
2. Display image
Image is visible to the user
3. Compute SHA-256
4. Verify hash
Match
Keep image
Mismatch
Remove image
Too late! User already saw it
VS.
OPTION B
Verify first, display later
1. Download image
2. Compute SHA-256
Hash is computed incrementally while data arrives
3. Verify hash
Match
4. Display image
Now it's safe to show
Mismatch
Discard image
Never shown to the user
Which design would you choose? Why?
Consider security, user experience, and performance.

Would your answer be different for JavaScript?

Would your answer be different for JavaScript?

What about images? Fonts? CSS?

If you answered “load JavaScript”, this slide is for you



P.S. CSS?

FINGERPRINT THE USER

CSS can reveal characteristics of the user's environment without needing JavaScript.

1. Media queries

Detect screen size, resolution, aspect ratio, color scheme, contrast preference, HDR, prefers-reduced-motion, input capabilities, and more.

2. Available fonts

Detect which fonts are installed locally by measuring text rendering differences.

3. Rendering differences

Detect subtle differences in text metrics, anti-aliasing, subpixel rendering, and layout.

4. Browser & platform behavior

Exploit CSS features and bugs that vary across browsers, versions, and platforms.

How it works

A page includes CSS that applies different styles based on environment characteristics.
The page observes the result (e.g., via loaded resources or layout differences) to infer information.

Example: probing color scheme

```
@media (prefers-color-scheme: dark) {
  body { background-image: url("/dark.png"); }
}
@media (prefers-color-scheme: light) {
  body { background-image: url("/light.png"); }
}
```

The server sees which image was requested (dark.png or light.png) and learns the user's color scheme preference.

Why it matters

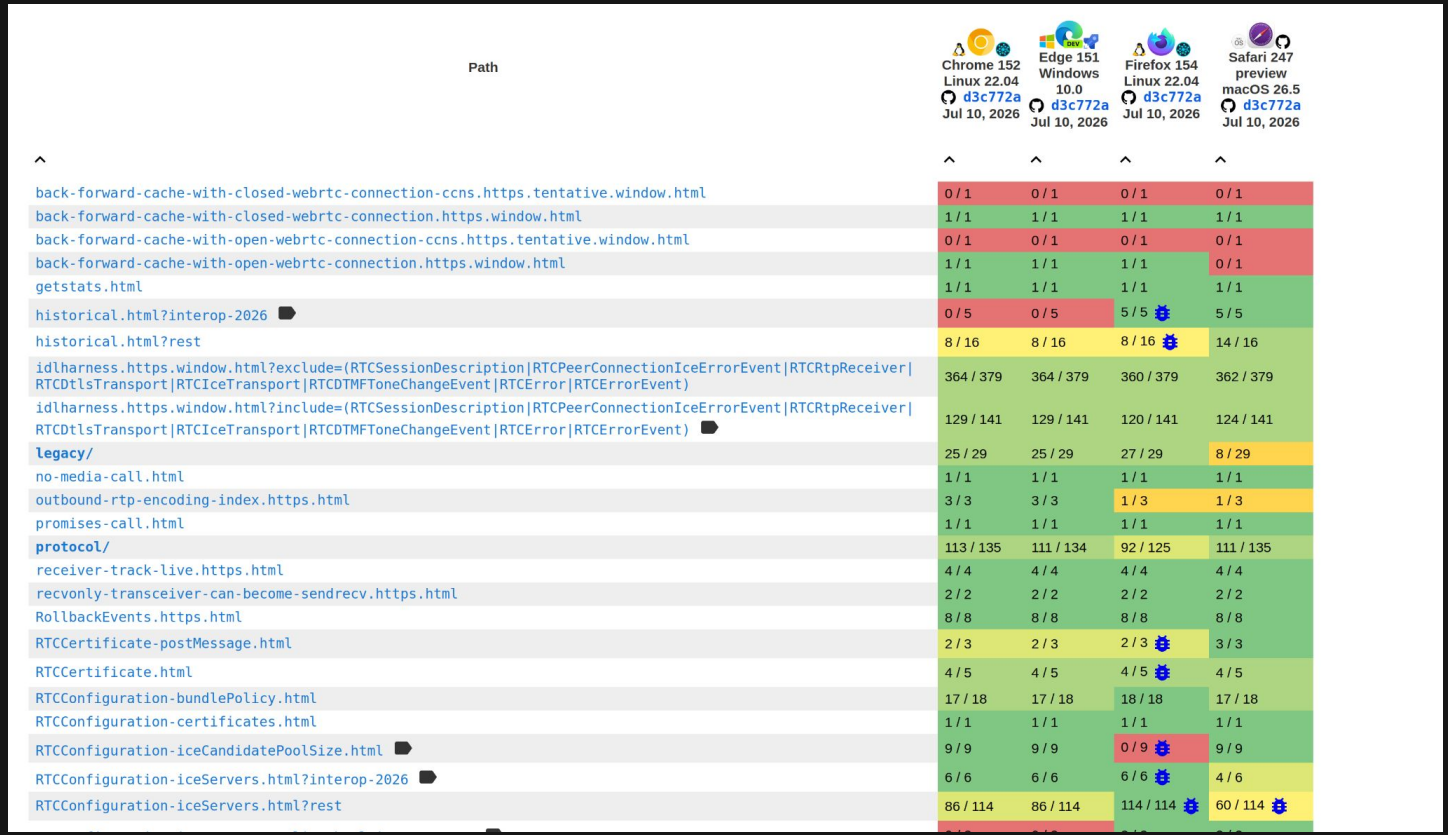
Fingerprinting enables tracking, bypasses cookie controls, and can deanonymize users even in privacy-focused browsers.

Thanks

Why Do APIs Exist?

Why Do APIs Exist?

Browsers expose capabilities through standardized APIs for exactly the same reason operating systems expose system calls or libraries: to provide a stable, portable, and secure interface between applications and the underlying system.



Path	Chrome 152 Linux 22.04 d3c772a Jul 10, 2026	Edge 151 Windows 10.0 d3c772a Jul 10, 2026	Firefox 154 Linux 22.04 d3c772a Jul 10, 2026	Safari 247 preview macOS 26.5 d3c772a Jul 10, 2026
back-forward-cache-with-closed-webrtc-connection-ccns.https.tentative.window.html	0 / 1	0 / 1	0 / 1	0 / 1
back-forward-cache-with-closed-webrtc-connection.https.window.html	1 / 1	1 / 1	1 / 1	1 / 1
back-forward-cache-with-open-webrtc-connection-ccns.https.tentative.window.html	0 / 1	0 / 1	0 / 1	0 / 1
back-forward-cache-with-open-webrtc-connection.https.window.html	1 / 1	1 / 1	1 / 1	0 / 1
getstats.html	1 / 1	1 / 1	1 / 1	1 / 1
historical.html?interop-2026	0 / 5	0 / 5	5 / 5	5 / 5
historical.html?rest	8 / 16	8 / 16	8 / 16	14 / 16
idlharness.https.window.html?exclude=(RTCSessionDescription|RTCPeerConnectionIceErrorEvent|RTCRtpReceiver|RTCDtlsTransport|RTCIceTransport|RTCDTMFToneChangeEvent|RTCError|RTCErrorEvent)	364 / 379	364 / 379	360 / 379	362 / 379
idlharness.https.window.html?include=(RTCSessionDescription|RTCPeerConnectionIceErrorEvent|RTCRtpReceiver|RTCDtlsTransport|RTCIceTransport|RTCDTMFToneChangeEvent|RTCError|RTCErrorEvent)	129 / 141	129 / 141	120 / 141	124 / 141
legacy/	25 / 29	25 / 29	27 / 29	8 / 29
no-media-call.html	1 / 1	1 / 1	1 / 1	1 / 1
outbound-rtp-encoding-index.https.html	3 / 3	3 / 3	1 / 3	1 / 3
promises-call.html	1 / 1	1 / 1	1 / 1	1 / 1
protocol/	113 / 135	111 / 134	92 / 125	111 / 135
receiver-track-live.https.html	4 / 4	4 / 4	4 / 4	4 / 4
recvonly-transceiver-can-become-sendrecv.https.html	2 / 2	2 / 2	2 / 2	2 / 2
RollbackEvents.https.html	8 / 8	8 / 8	8 / 8	8 / 8
RTCCertificate-postMessage.html	2 / 3	2 / 3	2 / 3	3 / 3
RTCCertificate.html	4 / 5	4 / 5	4 / 5	4 / 5
RTCConfiguration-bundlePolicy.html	17 / 18	17 / 18	18 / 18	17 / 18
RTCConfiguration-certificates.html	1 / 1	1 / 1	1 / 1	1 / 1
RTCConfiguration-iceCandidatePoolSize.html	9 / 9	9 / 9	0 / 9	9 / 9
RTCConfiguration-iceServers.html?interop-2026	6 / 6	6 / 6	6 / 6	4 / 6
RTCConfiguration-iceServers.html?rest	86 / 114	86 / 114	114 / 114	60 / 114