It Wasn't Me, It Was the Prototype!

Towards a Formal Model of JavaScript Prototype Pollution

Mohammad M. Ahmadpanah, David Sands, and Musard Balliu

KTH

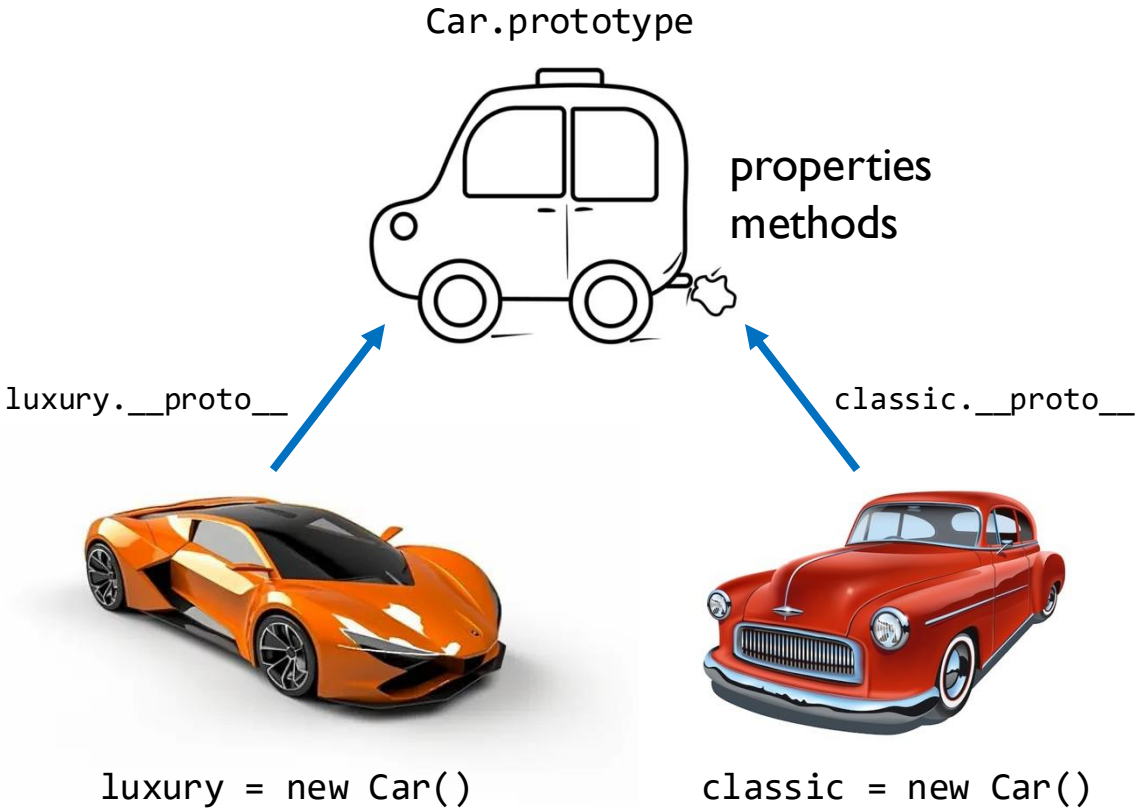
Chalmers

ShiftLeft Workshop, Gothenburg

October 27, 2025

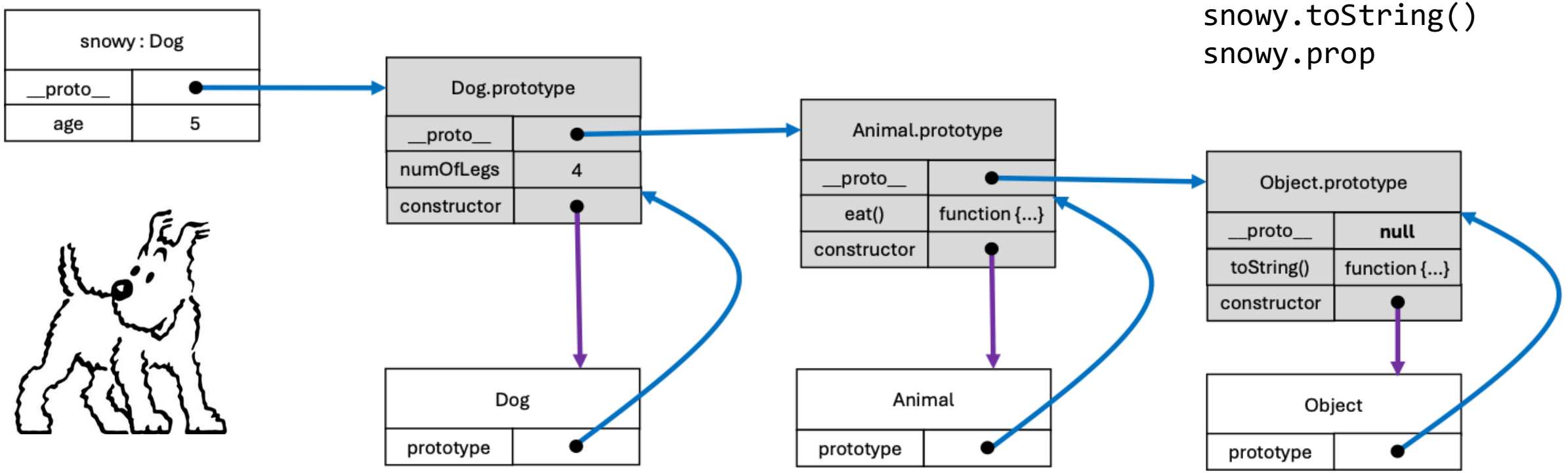
JavaScript objects

- Mutable collection of properties
  - Values evaluated at **runtime**
- **Prototype: object blueprint**
  - Reusing existing objects
  - An object with a set of properties and functions shared between all objects of the same type
  - Exposed as regular programming construct
    `luxury["__proto__"], classic.__proto__, Car.prototype`



Prototype-based Inheritance

```
snowy.age
snowy.numOfLegs
snowy.eat()
snowy.toString()
snowy.prop
```



Prototype pollution (PP)

brutalicious : Dog

__proto__	
age	100

```
brutalicious.__proto__.numOfLegs = 3
```

snowy : Dog

[illegible]	
	5

Poor snowy! ☹

Dog.prototype

__proto__	
numOfLegs	3
constructor	

Dog

prototype	

Animal.prototype

__proto__	
eat()	function {...}
constructor	

Animal

prototype	

Object.prototype

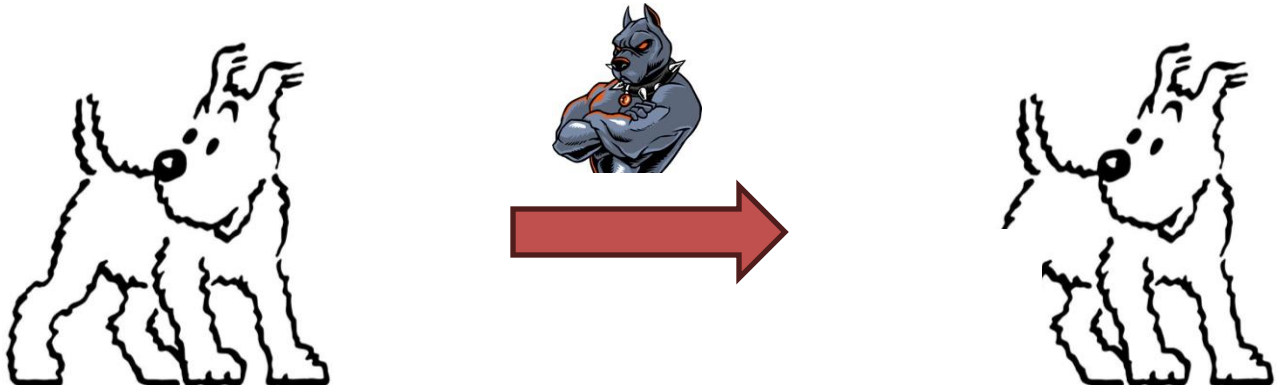
__proto__	**null**
toString()	function {...}
constructor	

Object

prototype	

Prototype pollution (cont.)

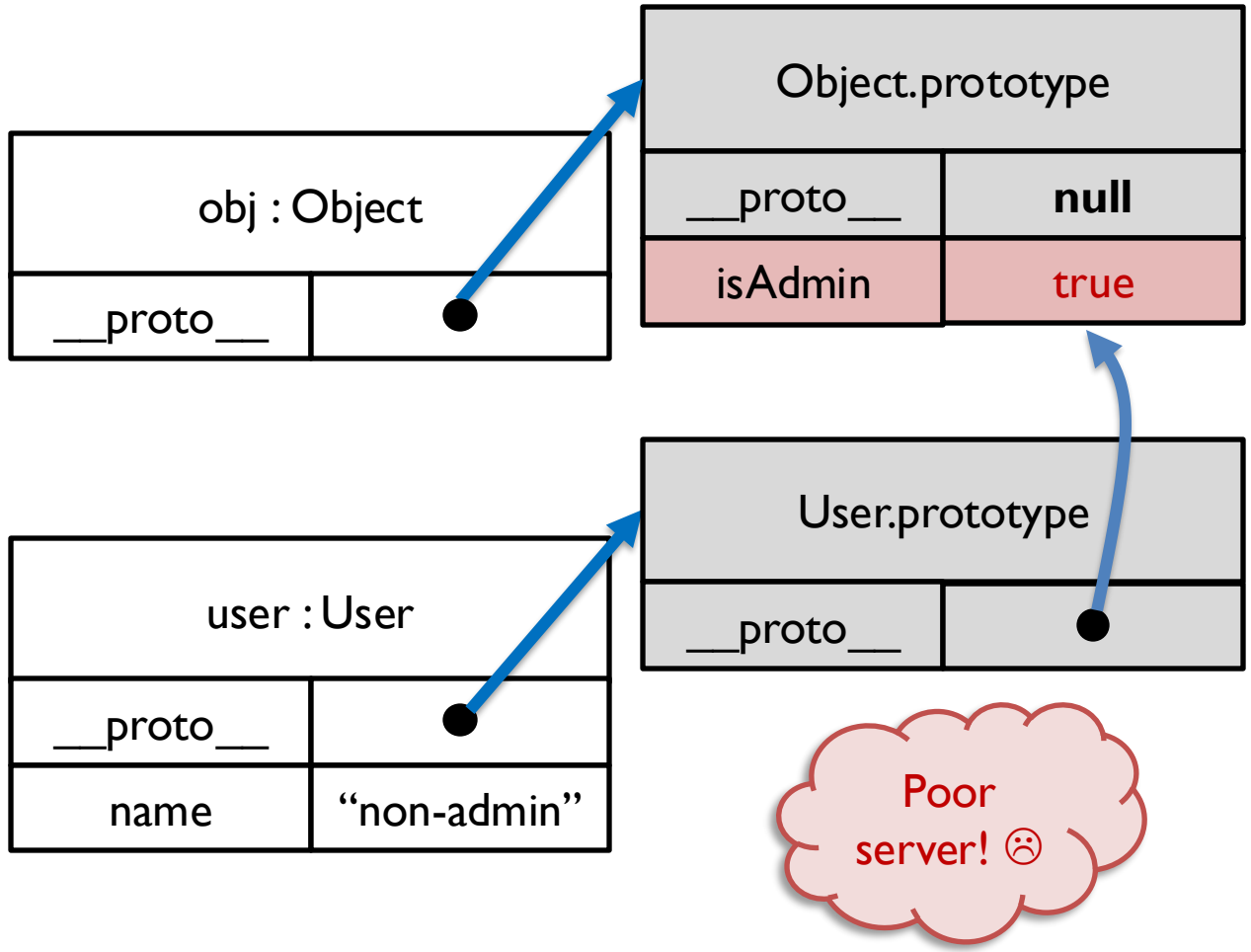
- A vulnerability where an attacker can modify an **object's prototype** at ***runtime***
  - May then be inherited by user-defined objects
  - May then result in **unintended behavior**



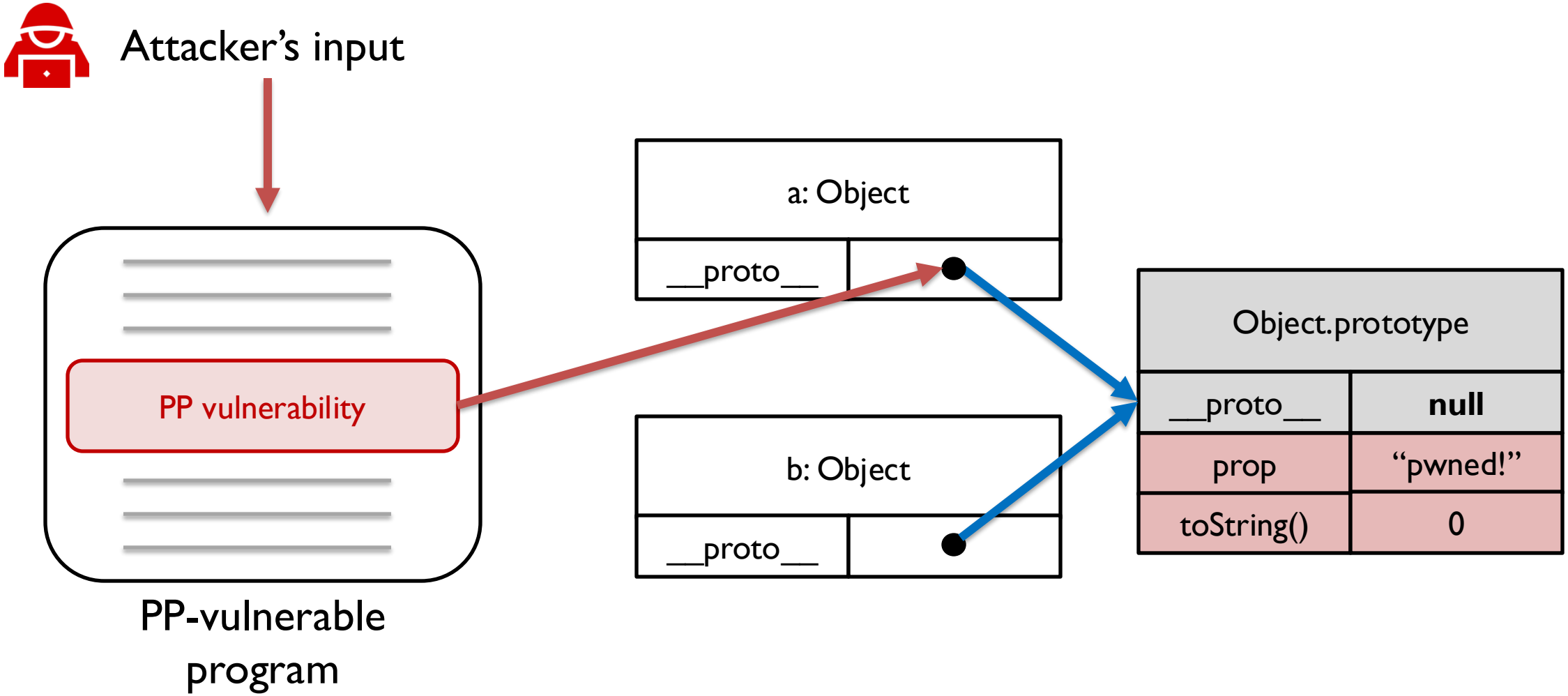
Prototype pollution (cont.)

```
{ "__proto__": { "isAdmin": true } }
```

```
function checkAdmin(user) {
  if (user.isAdmin) {
    console.log("Welcome, admin user!!");
  } else {
    console.log("Just a regular user...");
  }
}
...
let user = new User("non-admin");
...
let obj = JSON.parse(input);
checkAdmin(user);
```

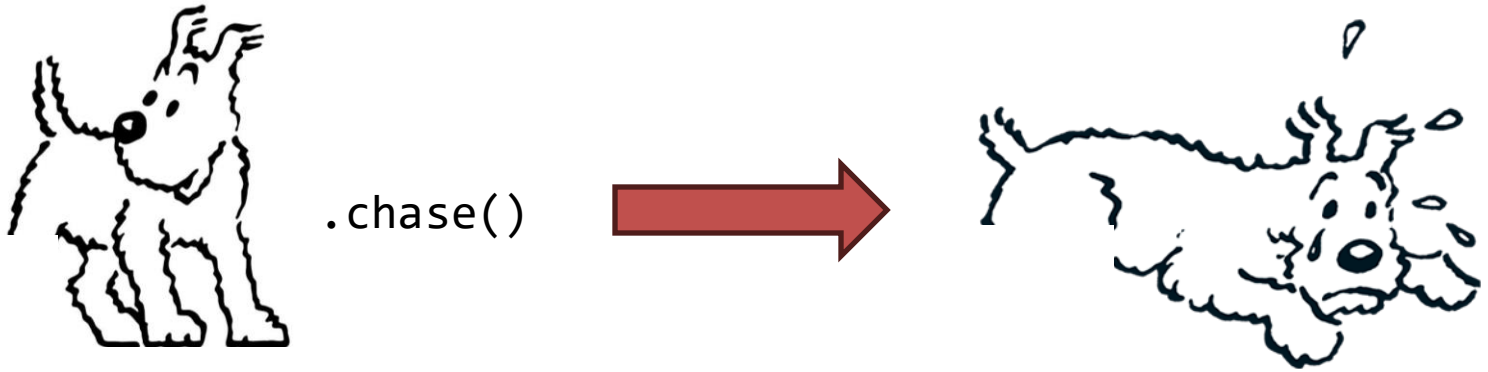


PP-vulnerable program

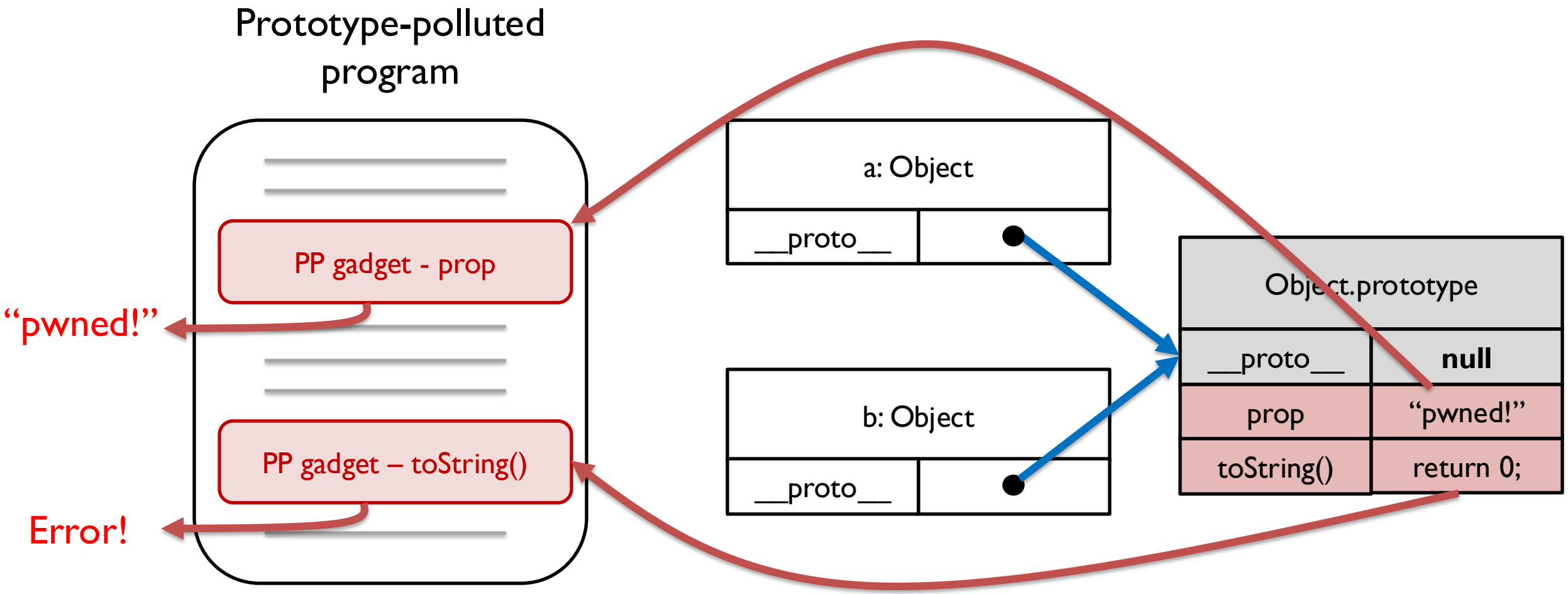


Gadget

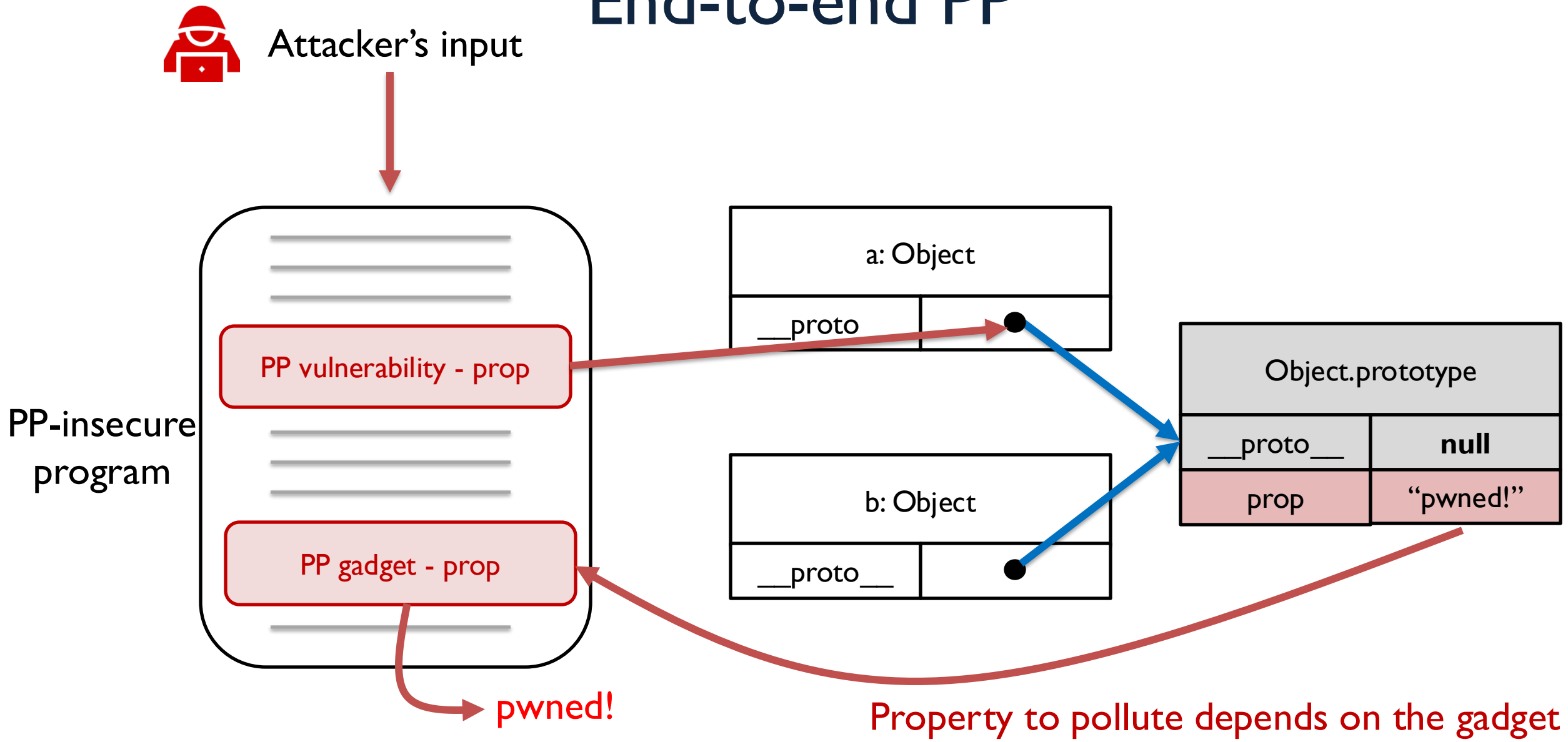
- PP's impact depends on the existence of **gadgets**
  - An ***otherwise benign*** piece of code which *inadvertently* **read from polluted properties** to execute security-sensitive operations
    - Triggering unintended behavior
- Examples: privilege elevation, reading secrets, RCE, DoS, log pollution



Gadget (cont.)


Prototype-polluted program
PP gadget - prop
PP gadget – toString()
"pwned!"
Error!
a: Object
__proto__
b: Object
__proto__
Object.prototype
__proto__
null
prop
"pwned!"
toString()
return 0;

End-to-end PP


Attacker's input
PP-insecure program
PP vulnerability - prop
PP gadget - prop
pwned!
a: Object
__proto
b: Object
__proto__
Object.prototype
__proto__
null
prop
"pwned!"
Property to pollute depends on the gadget

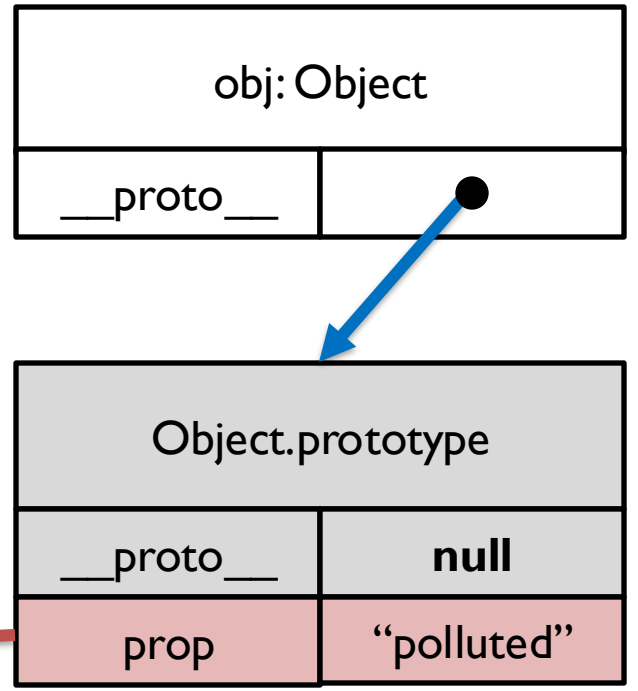
Example 1

```
var inProto = "__proto__"
var inProp = "prop"
var inVal = "polluted"

var obj = {}
var p = obj[inProto]
p[inProp] = inVal

console.log({}.prop) //"polluted"
```

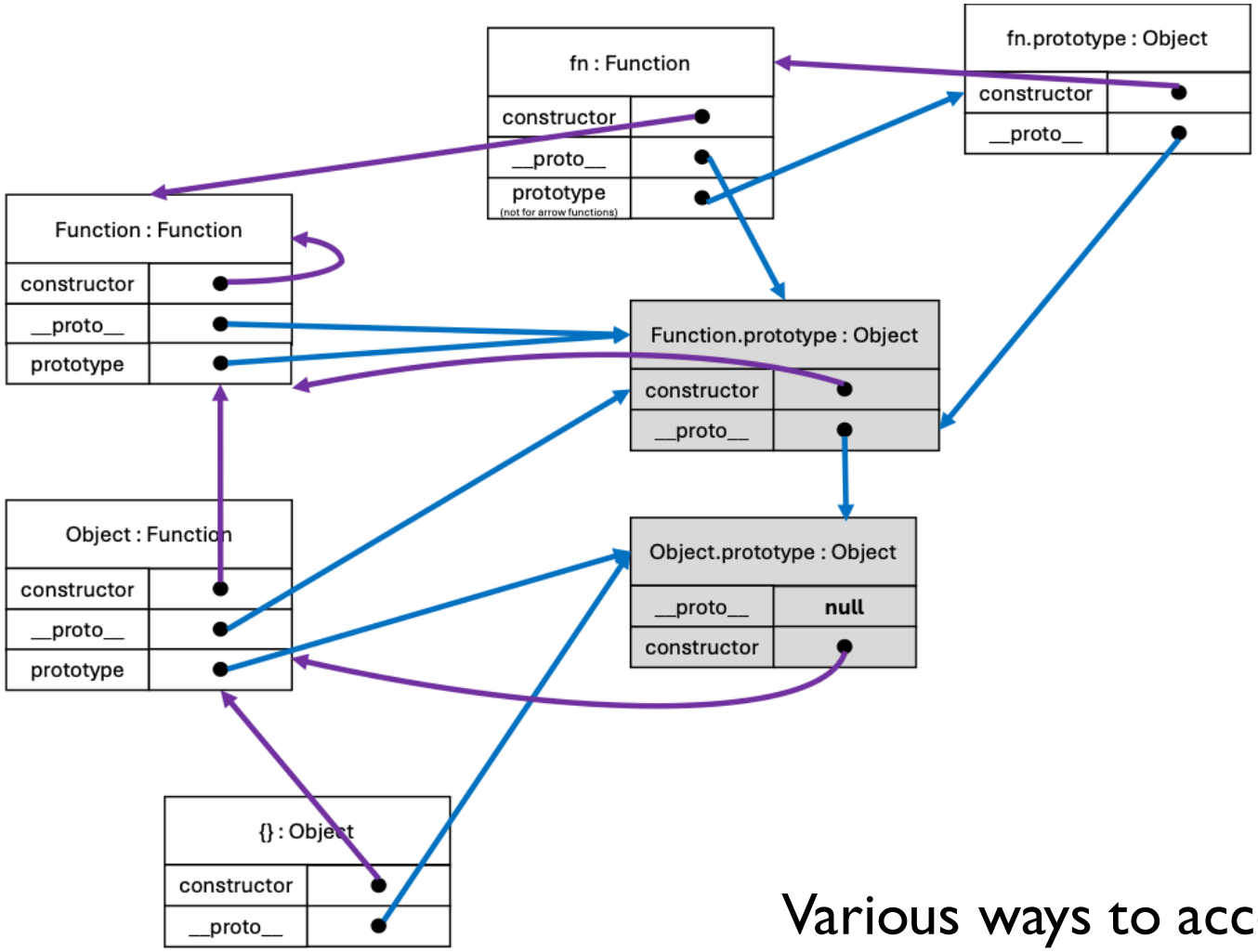
Malicious input

PP vulnerability

Gadget



Who says JavaScript is hard to learn?! ☺



Various ways to access object prototypes

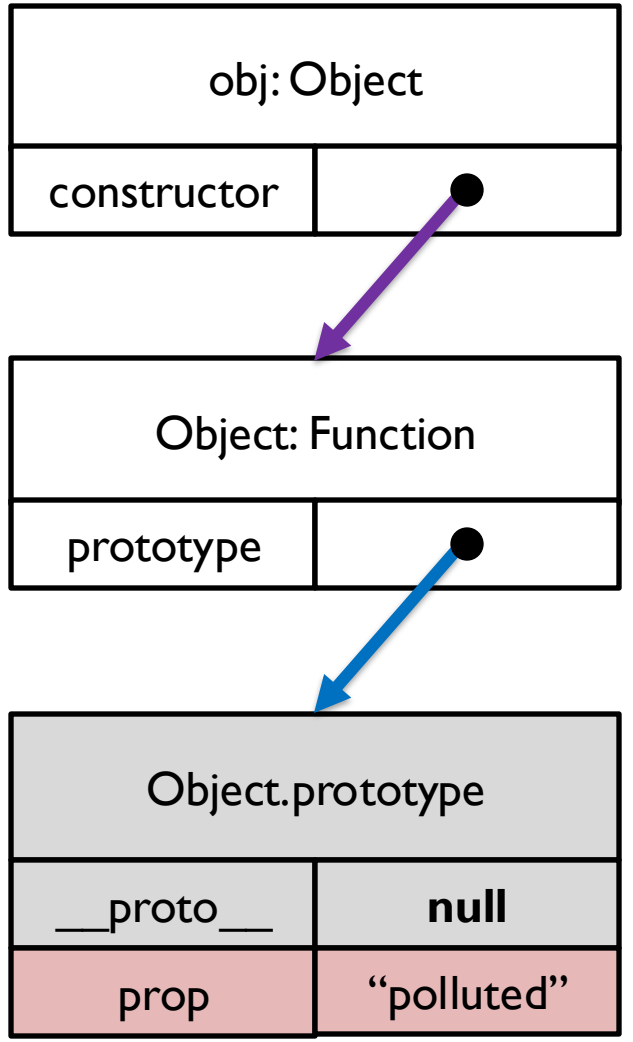
Example 2

```
var inConstr = "constructor"
var inProto = "prototype"
var inProp = "prop"
var inVal = "polluted"

var obj = {}
var c = obj[inConstr]
var p = c[inProto]
p[inProp] = inVal

console.log({}.prop) //"polluted"
```

Malicious input

PP vulnerability

Gadget



Example 3

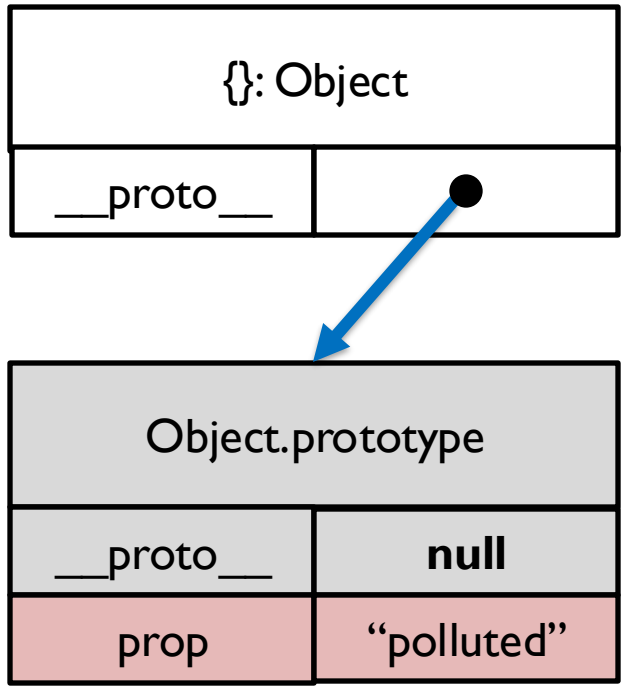
```
function merge(dst, src) {
  for (let key in src) {
    if (!src.hasOwnProperty(key)) continue;
    if (typeof dst[key] === 'object') {
      merge(dst[key], src[key]);
    } else {
      dst[key] = src[key];
    }
  }
}

var input = '{"__proto__": {"prop": "polluted"}}'
merge({}, JSON.parse(input));

console.log({}.prop); //"polluted"
```

PP vulnerability

Malicious input

PP vulnerability

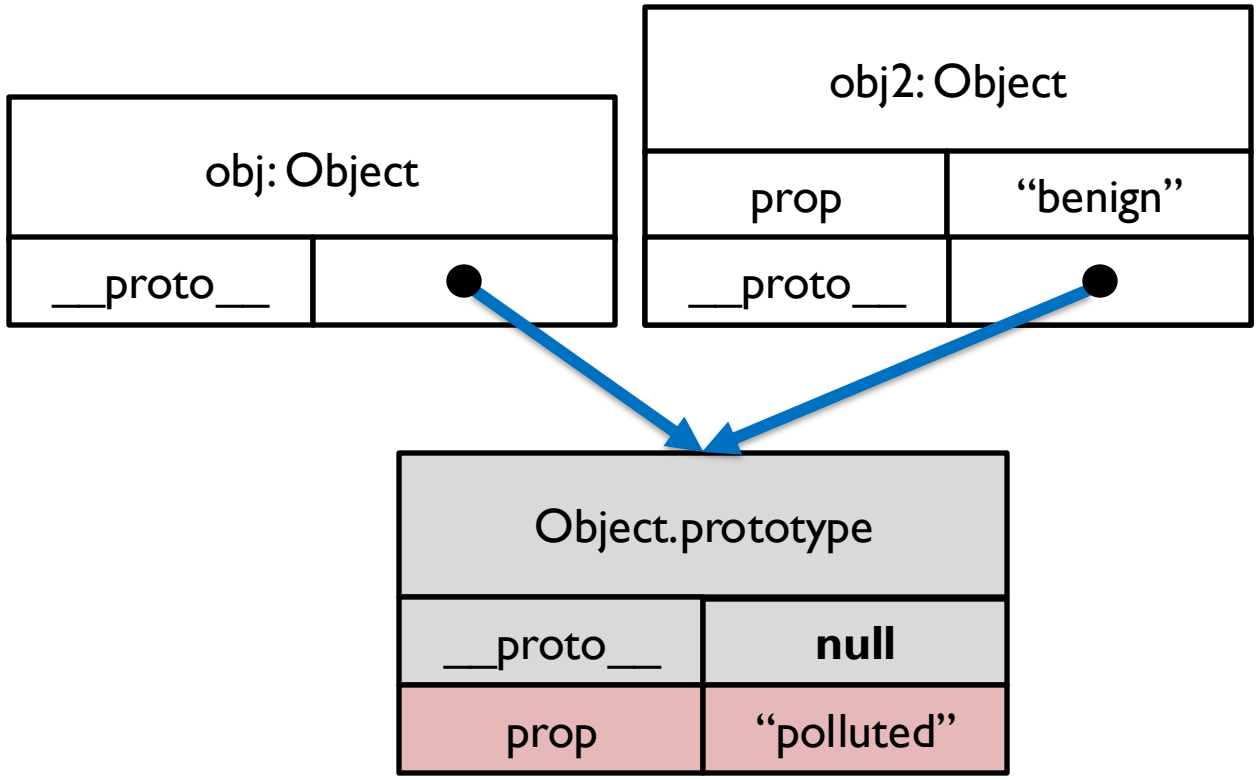
Gadget



Example 4

```
var inProto = "__proto__"
var inProp = "prop"
var inVal = "polluted"

var obj = {}
var p = obj[inProto]
p[inProp] = inVal

var obj2 = {}
obj2.prop = "benign"

console.log(obj2.prop) //"benign"
```

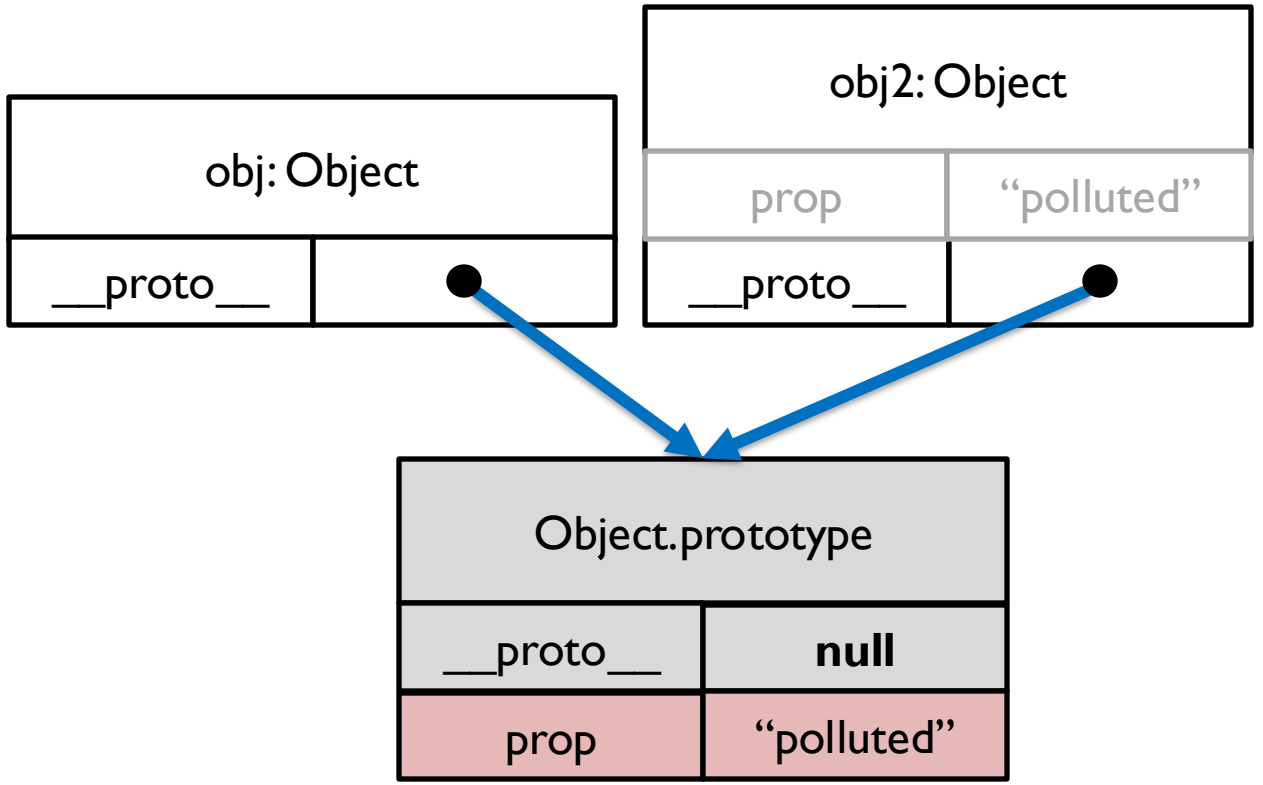
Observable behavior? 🤔



Example 5

```
var inProto = "__proto__"
var inProp = "prop"
var inVar = true

var obj, obj2 = {}
if (inVar)
  obj[inProto][inProp] = "polluted"
else
  obj2.prop = "polluted"

console.log(obj2.prop) //"polluted"
```

Observable behavior? 🤔



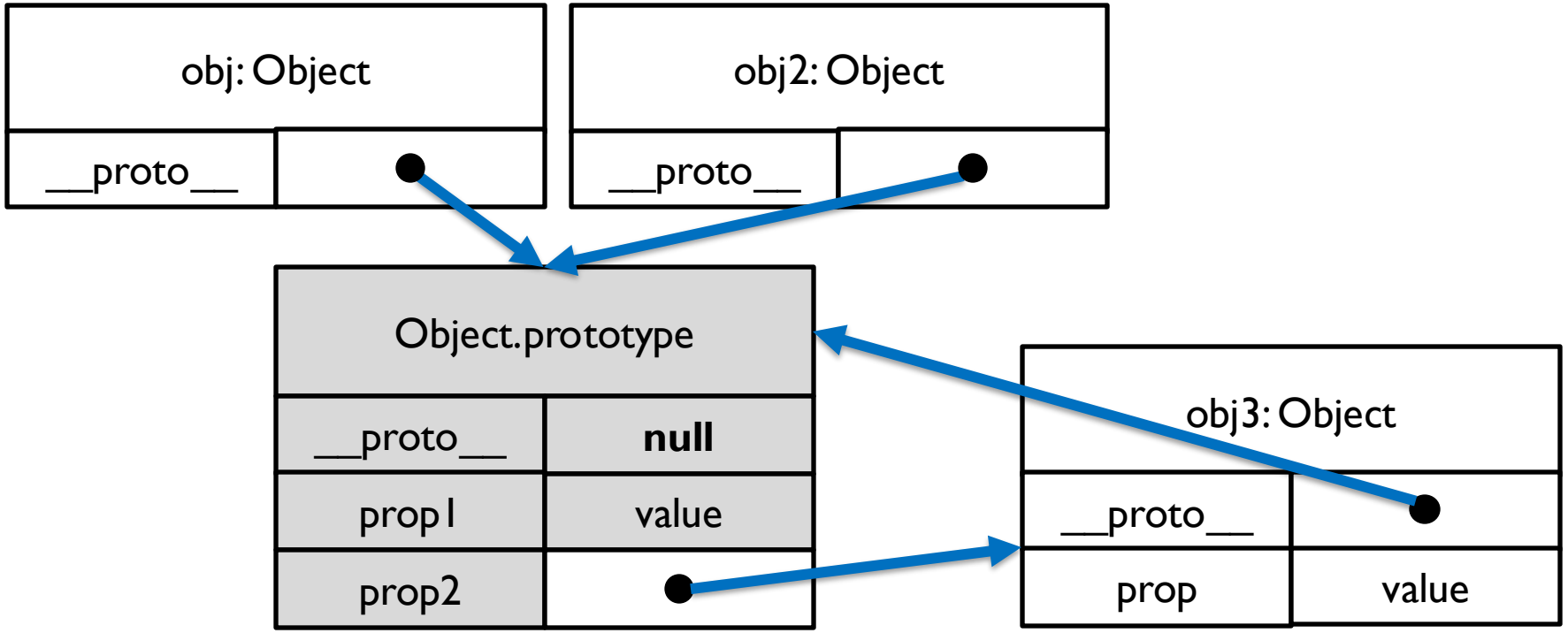
From PP to unintended behavior

- The literature focuses on *detection* rather than *defining* PP and gadgets *formally*
  - Any untrusted access to __proto__ is forbidden (*shallow definition*)
  - Any modification to an object property that is reachable from a prototype object is forbidden (*deep definition*)
- Gadgets
  - ACE, SSRF, privilege escalation, DoS, log pollution, cryptographic downgrade, …

What are the formal definitions of ***PP-secure*** and ***gadget-free*** programs?

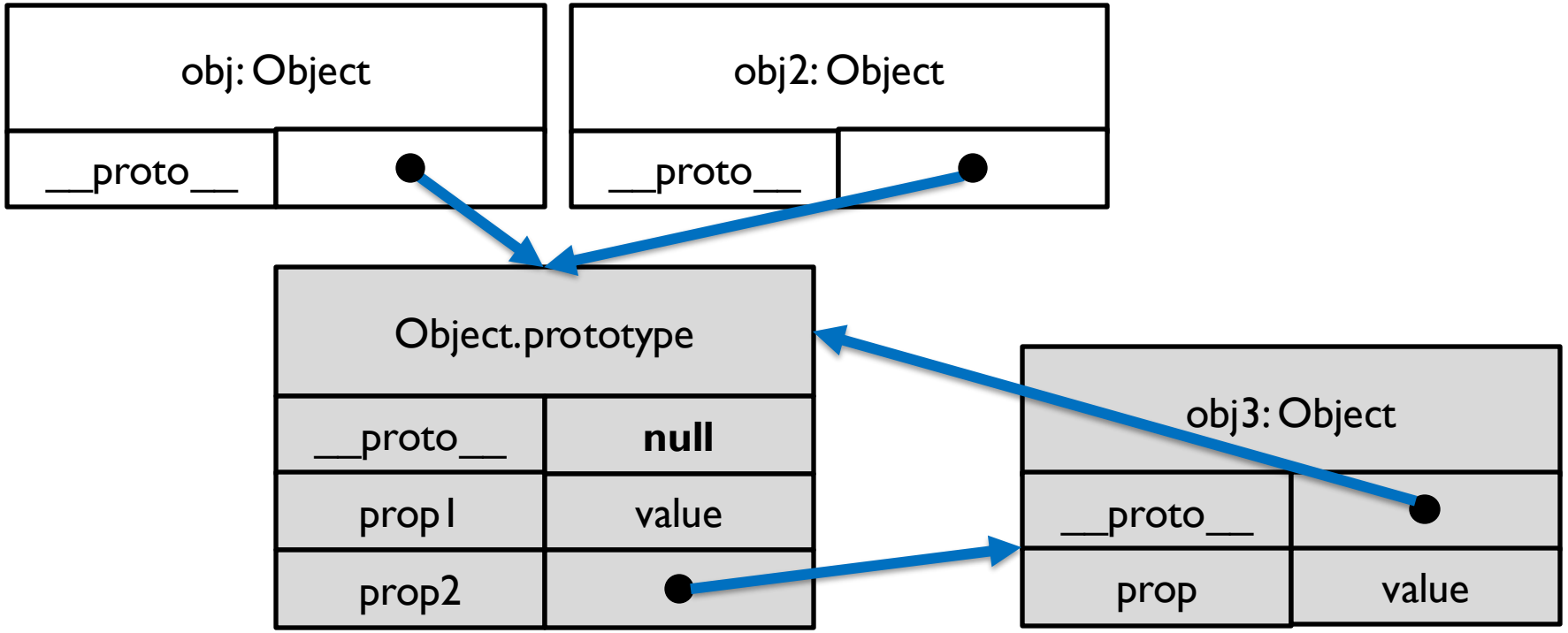
Shallow definition

The gray cells are expected to remain same for any untrusted input



Deep definition

The gray cells are expected to remain same for any untrusted input



Model language

A subset of the non-strict semantics of ECMA-262 standard

$$
\begin{aligned}
v &::= s \mid n \mid b \mid \mathit{undefined} \mid \mathit{null} \\
e &::= v \mid x \mid e \oplus e \mid e(e) \mid e[e] \mid \mathit{new}\, e(e) \mid \mathit{function}\,(x)\; c \\
i &::= x \mid i[e] \\
c &::= \mathit{skip} \mid i = e \mid \mathit{if}\ (e)\ c\ \mathit{else}\ c \mid \mathit{while}\ (e)\ c \mid c; c \mid \mathit{return}\ e \mid \mathit{out}_L(e)
\end{aligned}
$$

PP security

$$\textbf{\textit{PP-secure(c)}} \stackrel{def}{=} \forall E_1, E_2, H_1, H_2.$$

$$(E_1, H_1) \simeq_T (E_2, H_2) \wedge$$

$$\Gamma \vDash \langle c, E_1, H_1, \varnothing \rangle \longrightarrow^* \langle E_3, H_3, \tau_1 \rangle \wedge$$

$$\Gamma \vDash \langle c, E_2, H_2, \varnothing \rangle \longrightarrow^* \langle E_4, H_4, \tau_2 \rangle$$

$$\Longrightarrow \tau_1 \simeq^{proto}_{stut} \tau_2.$$

Shallow vs. deep

PP-Gadget freedom

$$\textbf{\textit{PP-Gadget-Free(c)}} \stackrel{def}{=} \forall E_1, E_2, H_1, H_2, H_3^P, H_4^P.$$

$$\Gamma \vDash \langle c, E_1, H_1, \varnothing \rangle \rightarrow_v^* \langle E_1', H_1', \tau_1 \rangle \wedge$$

$$\Gamma \vDash \langle c, E_2, H_2, \varnothing \rangle \rightarrow_{v'}^* \langle E_2', H_2', \tau_2 \rangle \wedge$$

$$v \sim_\beta v'$$

$$\Longrightarrow$$

$$\Gamma \vDash \langle c, E_1, H_1^O \uplus H_3^P, \varnothing \rangle \rightarrow_{v''}^* \langle E_1'', H_1'', \tau_3 \rangle \wedge$$

$$\Gamma \vDash \langle c, E_2, H_2^O \uplus H_4^P, \varnothing \rangle \rightarrow_{v'''}^* \langle E_2'', H_2'', \tau_4 \rangle \wedge$$

$$v'' \sim_\beta v'''.$$

End-to-end PP

$$\textbf{\textit{E2E-secure(c)}} \stackrel{def}{=} \forall E_1, E_2, H_1, H_2.$$

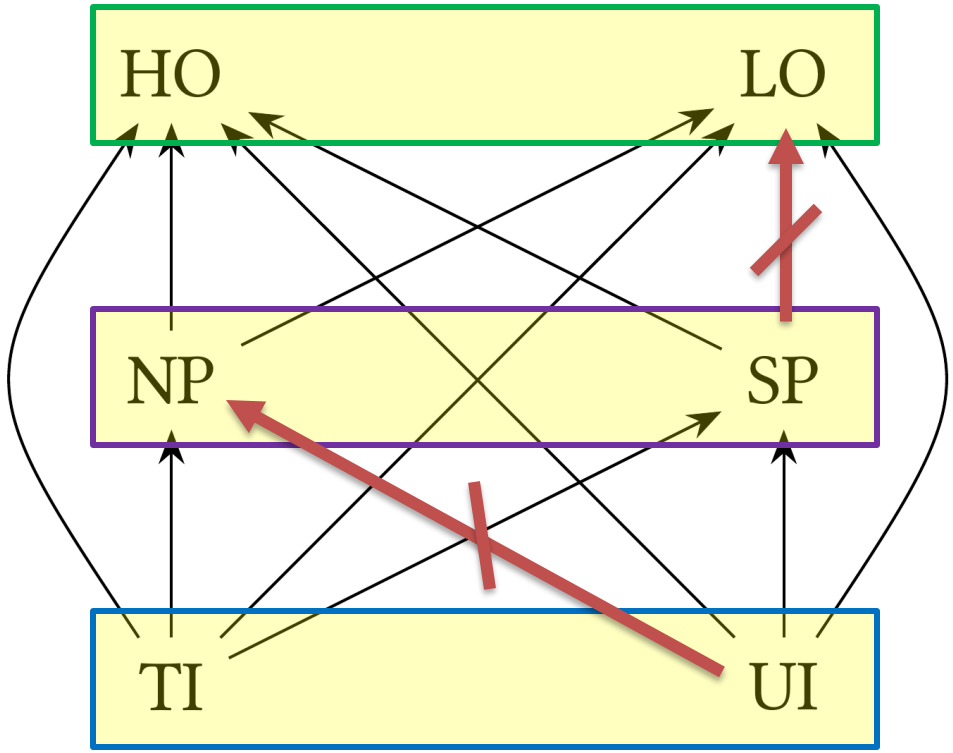
$$(E_1, H_1) \simeq_T (E_2, H_2) \wedge$$

$$H_1 \simeq^{\kappa, NP} H_2 \wedge$$

$$\Gamma \vDash \langle c, E_1, H_1, \varnothing \rangle \rightarrow_v^* \langle E_3, H_3, \tau_1 \rangle \wedge$$

$$\Gamma \vDash \langle c, E_2, H_2, \varnothing \rangle \rightarrow_{v'}^* \langle E_4, H_4, \tau_2 \rangle$$

$$\Longrightarrow v \sim_\beta^{LO} v' \vee \tau_1 \simeq_{stut}^{\kappa, SP} \tau_2.$$



Ongoing work

- Sound enforcement mechanisms
  - Runtime monitoring
- Non-root pollution
- Degree of exploitability and attacker's knowledge
- Modeling mitigations
  - Object.freeze(), Object.seal()
  - Use of new Set(), new Map()

Takeaways

Formal modeling of prototype-pollution security