

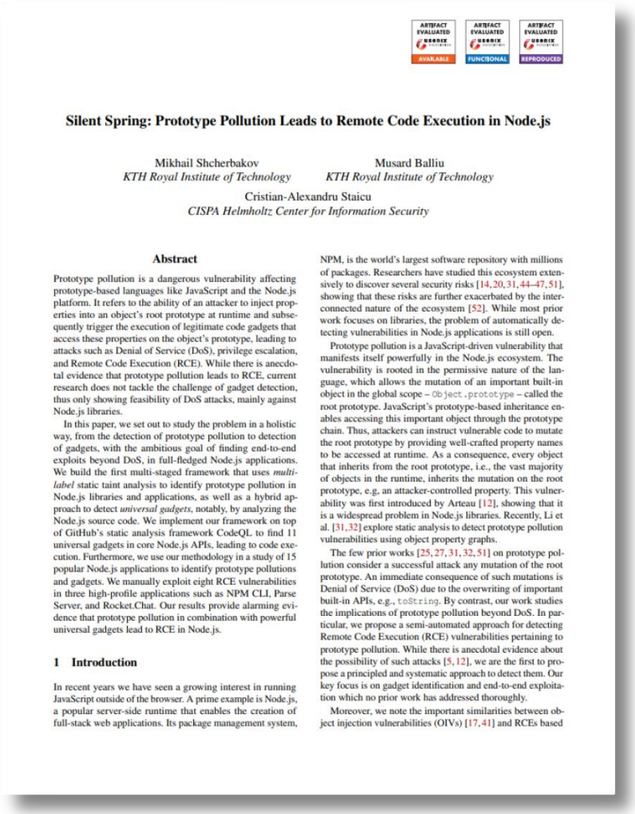


Silent Spring: Prototype Pollution Leads to Remote Code Execution in Node.js

Mikhail Shcherbakov and Musard Balliu
KTH Royal Institute of Technology

Cristian-Alexandru Staicu
CISPA Helmholtz Center for Information Security

Research overview



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

Mikhail Shcherbakov
KTH Royal Institute of Technology

Musard Balliu
KTH Royal Institute of Technology

Cristian-Alexandru Staicu
CISPA Helmholtz Center for Information Security

Abstract

Prototype pollution is a dangerous vulnerability affecting prototype-based languages like JavaScript and the Node.js platform. It refers to the ability of an attacker to inject properties into an object's root prototype at runtime and subsequently trigger the execution of legitimate code gadgets that access these properties on the object's prototype, leading to attacks such as Denial of Service (DoS), privilege escalation, and Remote Code Execution (RCE). While there is anecdotal evidence that prototype pollution leads to RCE, current research does not tackle the challenge of gadget detection, thus only showing feasibility of DoS attacks, mainly against Node.js libraries.

In this paper, we set out to study the problem in a holistic way, from the detection of prototype pollution to detection of gadgets, with the ambitious goal of finding end-to-end exploits beyond DoS, in full-fledged Node.js applications. We build the first multi-staged framework that uses *multi-label* static taint analysis to identify prototype pollution in Node.js libraries and applications, as well as a hybrid approach to detect *universal gadgets*, notably, by analyzing the Node.js source code. We implement our framework on top of GitHub's static analysis framework CodeQL to find 11 universal gadgets in core Node.js APIs, leading to code execution. Furthermore, we use our methodology in a study of 15 popular Node.js applications to identify prototype pollutions and gadgets. We manually exploit eight RCE vulnerabilities in three high-profile applications such as NPM CLI, Parse Server, and Rocket.Chat. Our results provide alarming evidence that prototype pollution in combination with powerful universal gadgets lead to RCE in Node.js.

1 Introduction

In recent years we have seen a growing interest in running JavaScript outside of the browser. A prime example is Node.js, a popular server-side runtime that enables the creation of full-stack web applications. Its package management system, NPM, is the world's largest software repository with millions of packages. Researchers have studied this ecosystem extensively to discover several security risks [14,20,31,44–47,51], showing that these risks are further exacerbated by the interconnected nature of the ecosystem [52]. While most prior work focuses on libraries, the problem of automatically detecting vulnerabilities in Node.js applications is still open.

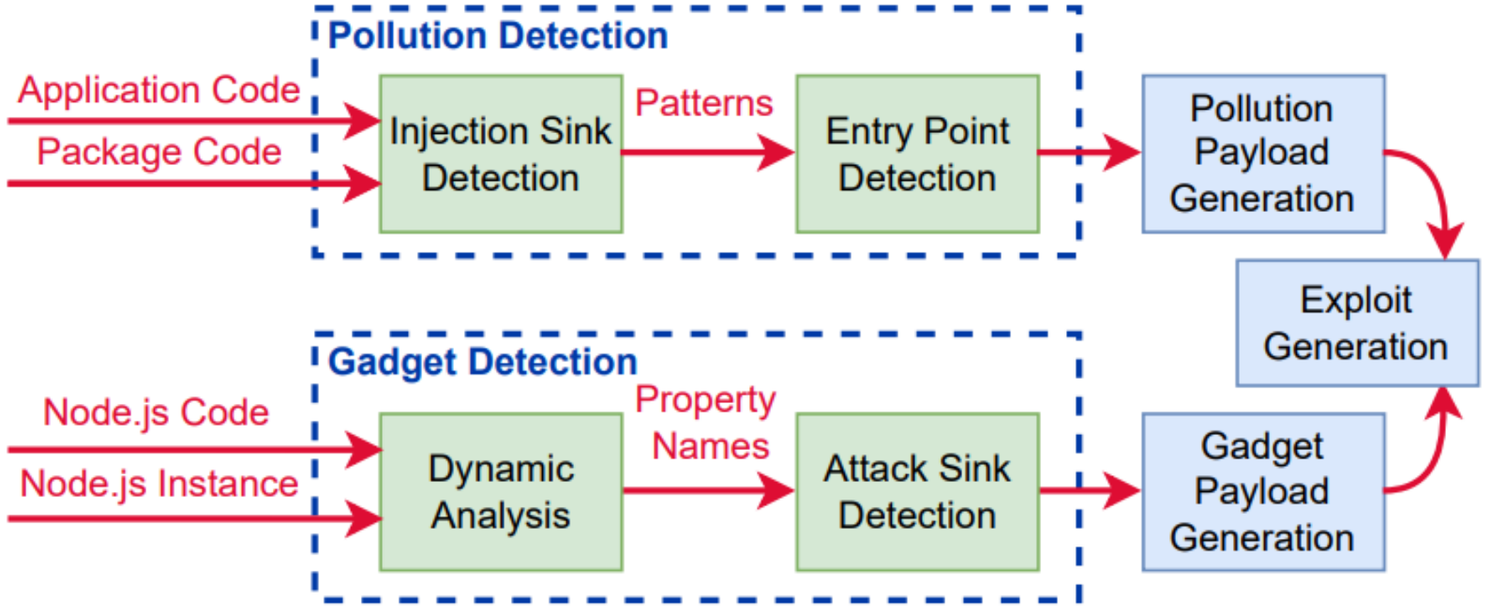
Prototype pollution is a JavaScript-driven vulnerability that manifests itself powerfully in the Node.js ecosystem. The vulnerability is rooted in the permissive nature of the language, which allows the mutation of an important built-in object in the global scope – `Object.prototype` – called the root prototype. JavaScript's prototype-based inheritance enables accessing this important object through the prototype chain. Thus, attackers can instruct vulnerable code to mutate the root prototype by providing well-crafted property names to be accessed at runtime. As a consequence, every object that inherits from the root prototype, i.e., the vast majority of objects in the runtime, inherits the mutation on the root prototype, e.g, an attacker-controlled property. This vulnerability was first introduced by Arteau [12], showing that it is a widespread problem in Node.js libraries. Recently, Li et al. [31,32] explore static analysis to detect prototype pollution vulnerabilities using object property graphs.

The few prior works [25,27,31,32,51] on prototype pollution consider a successful attack any mutation of the root prototype. An immediate consequence of such mutations is Denial of Service (DoS) due to the overwriting of important built-in APIs, e.g., `toString`. By contrast, our work studies the implications of prototype pollution beyond DoS. In particular, we propose a semi-automated approach for detecting Remote Code Execution (RCE) vulnerabilities pertaining to prototype pollution. While there is anecdotal evidence about the possibility of such attacks [5,12], we are the first to propose a principled and systematic approach to detect them. Our key focus is on gadget identification and end-to-end exploitation which no prior work has addressed thoroughly.

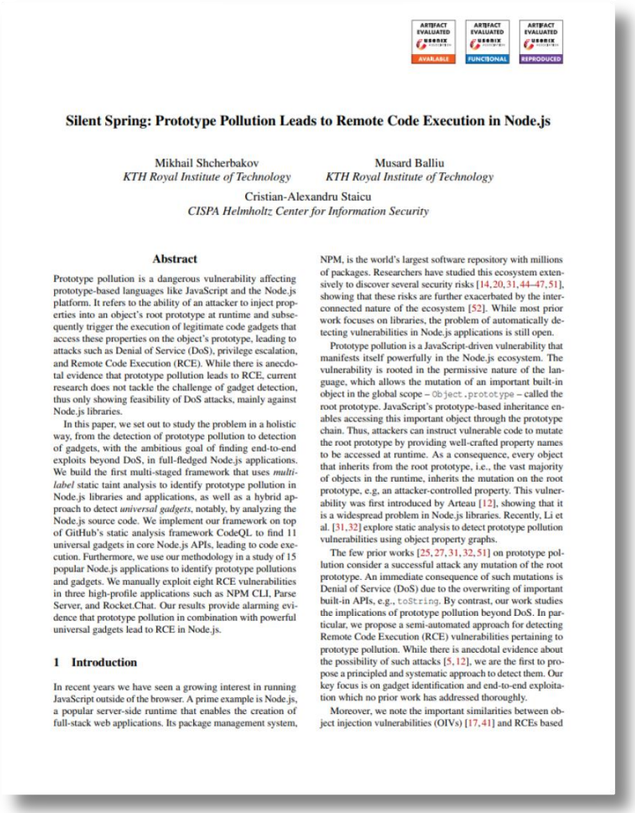
Moreover, we note the important similarities between object injection vulnerabilities (OIVs) [17,41] and RCEs based

https://github.com/yuske/silent-spring

Workflow: automated (green) and manual (blue) steps



Research overview



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

Mikhail Shcherbakov
KTH Royal Institute of Technology

Musard Balliu
KTH Royal Institute of Technology

Cristian-Alexandru Staicu
CISPA Helmholtz Center for Information Security

Abstract

Prototype pollution is a dangerous vulnerability affecting prototype-based languages like JavaScript and the Node.js platform. It refers to the ability of an attacker to inject properties into an object's root prototype at runtime and subsequently trigger the execution of legitimate code gadgets that access these properties on the object's prototype, leading to attacks such as Denial of Service (DoS), privilege escalation, and Remote Code Execution (RCE). While there is anecdotal evidence that prototype pollution leads to RCE, current research does not tackle the challenge of gadget detection, thus only showing feasibility of DoS attacks, mainly against Node.js libraries.

In this paper, we set out to study the problem in a holistic way, from the detection of prototype pollution to detection of gadgets, with the ambitious goal of finding end-to-end exploits beyond DoS, in full-fledged Node.js applications. We build the first multi-staged framework that uses *multi-label* static taint analysis to identify prototype pollution in Node.js libraries and applications, as well as a hybrid approach to detect *universal gadgets*, notably, by analyzing the Node.js source code. We implement our framework on top of GitHub's static analysis framework CodeQL to find 11 universal gadgets in core Node.js APIs, leading to code execution. Furthermore, we use our methodology in a study of 15 popular Node.js applications to identify prototype pollutions and gadgets. We manually exploit eight RCE vulnerabilities in three high-profile applications such as NPM CLI, Parse Server, and Rocket.Chat. Our results provide alarming evidence that prototype pollution in combination with powerful universal gadgets lead to RCE in Node.js.

1 Introduction

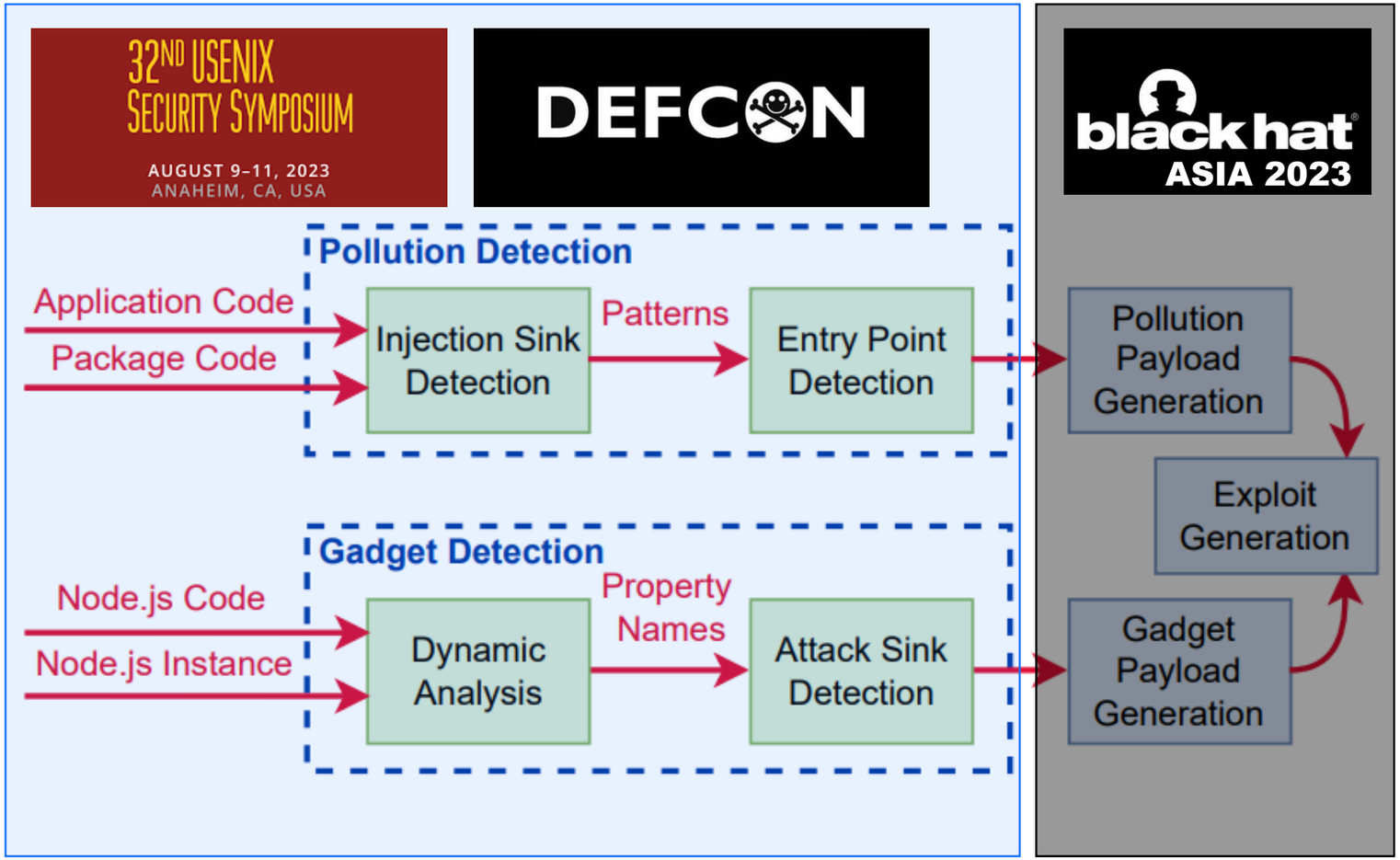
In recent years we have seen a growing interest in running JavaScript outside of the browser. A prime example is Node.js, a popular server-side runtime that enables the creation of full-stack web applications. Its package management system, NPM, is the world's largest software repository with millions of packages. Researchers have studied this ecosystem extensively to discover several security risks [14,20,31,44–47,51], showing that these risks are further exacerbated by the interconnected nature of the ecosystem [52]. While most prior work focuses on libraries, the problem of automatically detecting vulnerabilities in Node.js applications is still open.

Prototype pollution is a JavaScript-driven vulnerability that manifests itself powerfully in the Node.js ecosystem. The vulnerability is rooted in the permissive nature of the language, which allows the mutation of an important built-in object in the global scope – `Object.prototype` – called the root prototype. JavaScript's prototype-based inheritance enables accessing this important object through the prototype chain. Thus, attackers can instruct vulnerable code to mutate the root prototype by providing well-crafted property names to be accessed at runtime. As a consequence, every object that inherits from the root prototype, i.e., the vast majority of objects in the runtime, inherits the mutation on the root prototype, e.g, an attacker-controlled property. This vulnerability was first introduced by Arteau [12], showing that it is a widespread problem in Node.js libraries. Recently, Li et al. [31,32] explore static analysis to detect prototype pollution vulnerabilities using object property graphs.

The few prior works [25,27,31,32,51] on prototype pollution consider a successful attack any mutation of the root prototype. An immediate consequence of such mutations is Denial of Service (DoS) due to the overwriting of important built-in APIs, e.g., `toString`. By contrast, our work studies the implications of prototype pollution beyond DoS. In particular, we propose a semi-automated approach for detecting Remote Code Execution (RCE) vulnerabilities pertaining to prototype pollution. While there is anecdotal evidence about the possibility of such attacks [5,12], we are the first to propose a principled and systematic approach to detect them. Our key focus is on gadget identification and end-to-end exploitation which no prior work has addressed thoroughly.

Moreover, we note the important similarities between object injection vulnerabilities (OIVs) [17,41] and RCEs based

https://github.com/yuske/silent-spring



Background: inheritance in JavaScript

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

Terminal

```
> node app/index.js
42
>
```



node



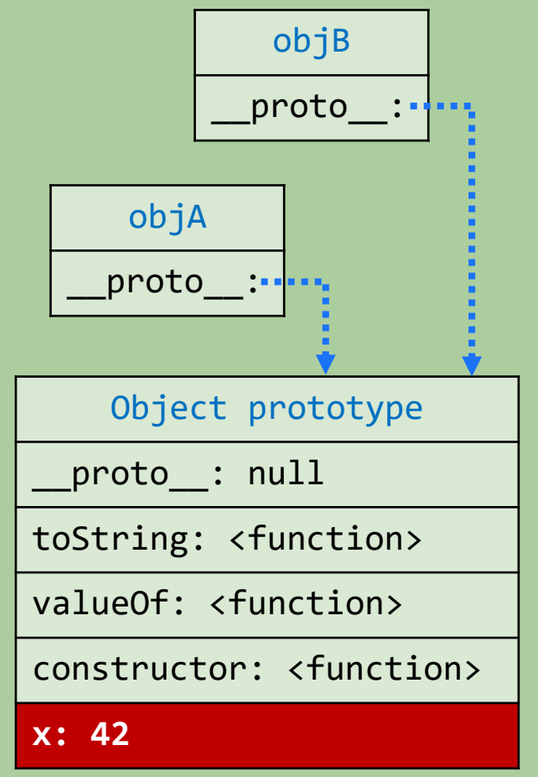
index.js

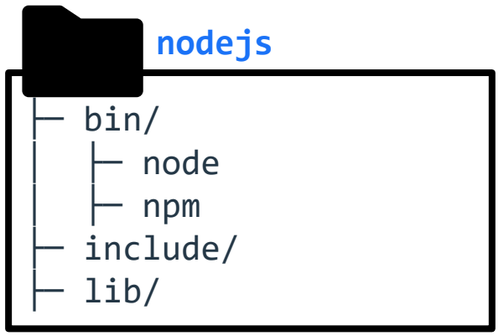
Code

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

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

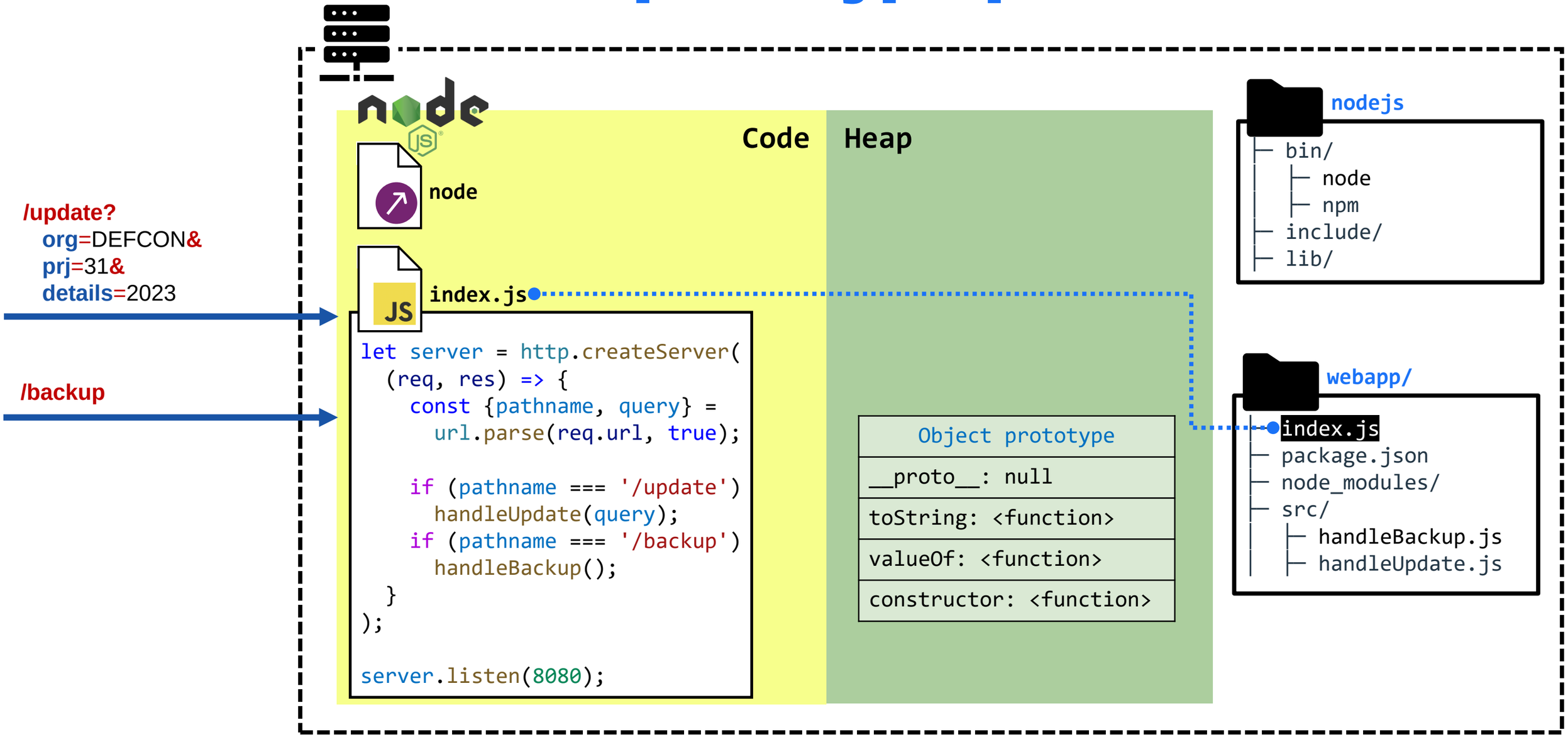
Heap



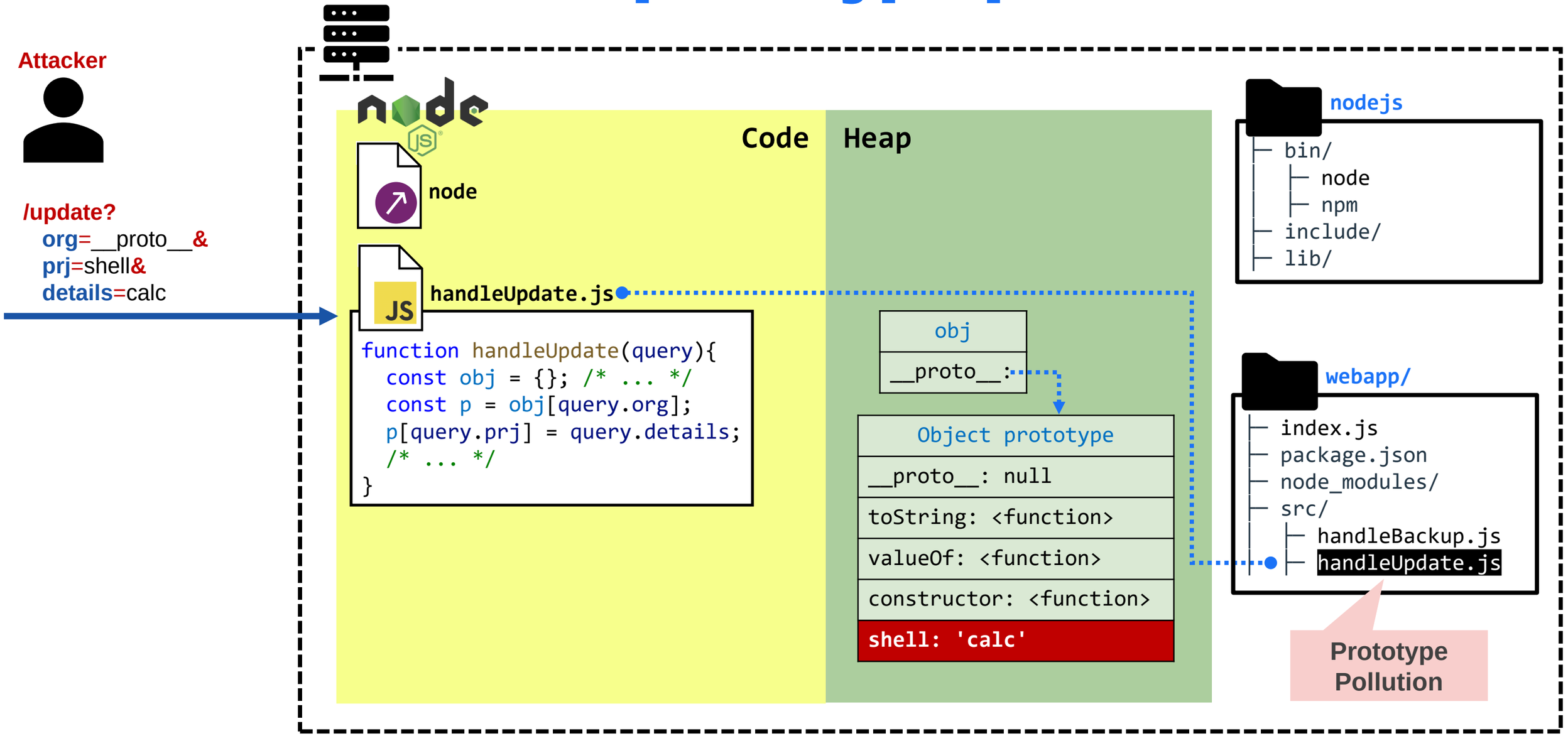




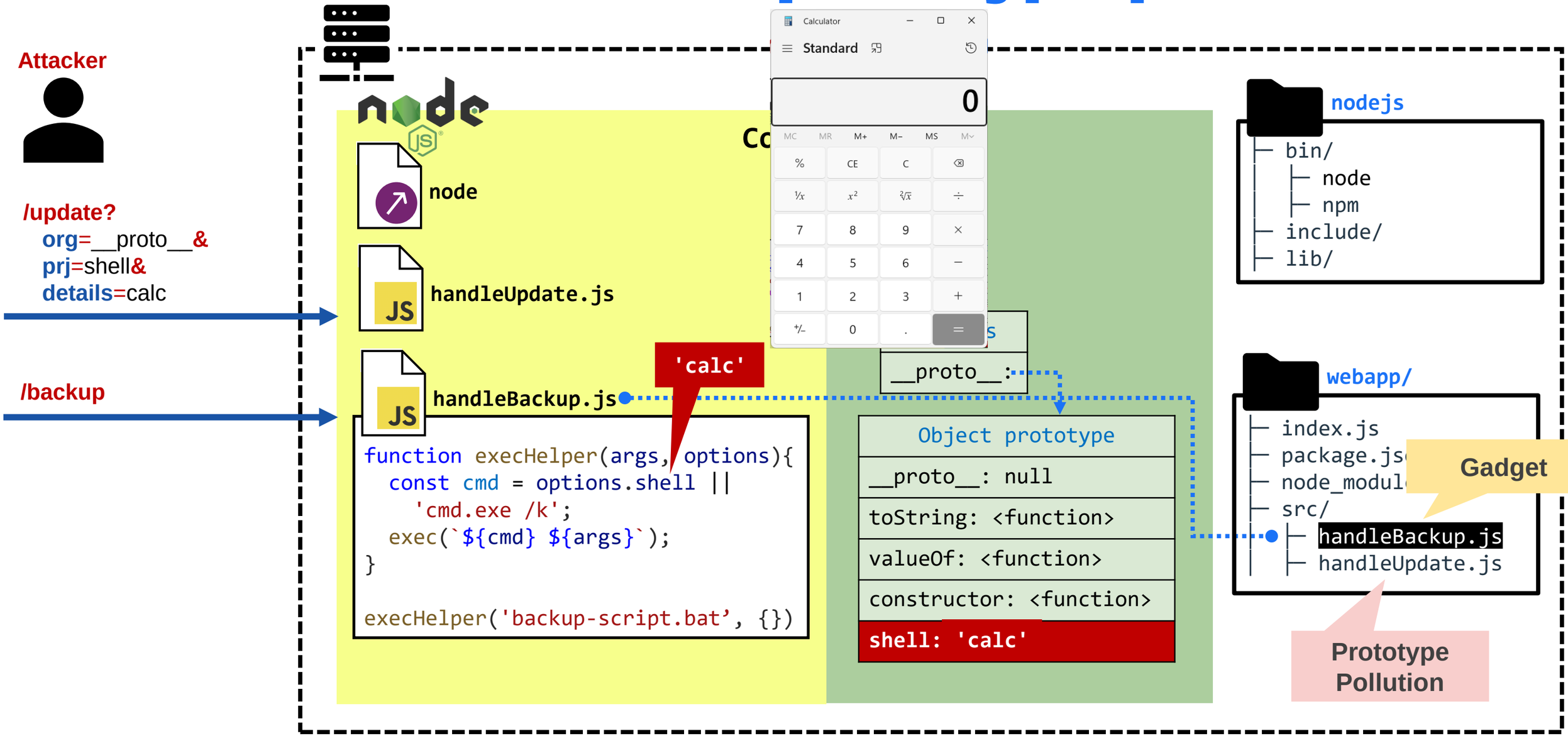
Problem: prototype pollution



Problem: prototype pollution



Problem: RCE via prototype pollution



Problem: Summary

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

Prototype Pollution

```
function handleUpdate(query){
  const obj = {};
  /* ... */
  const p = obj[query.org];
  p[query.prj] = query.details;
  /* ... */
}
```

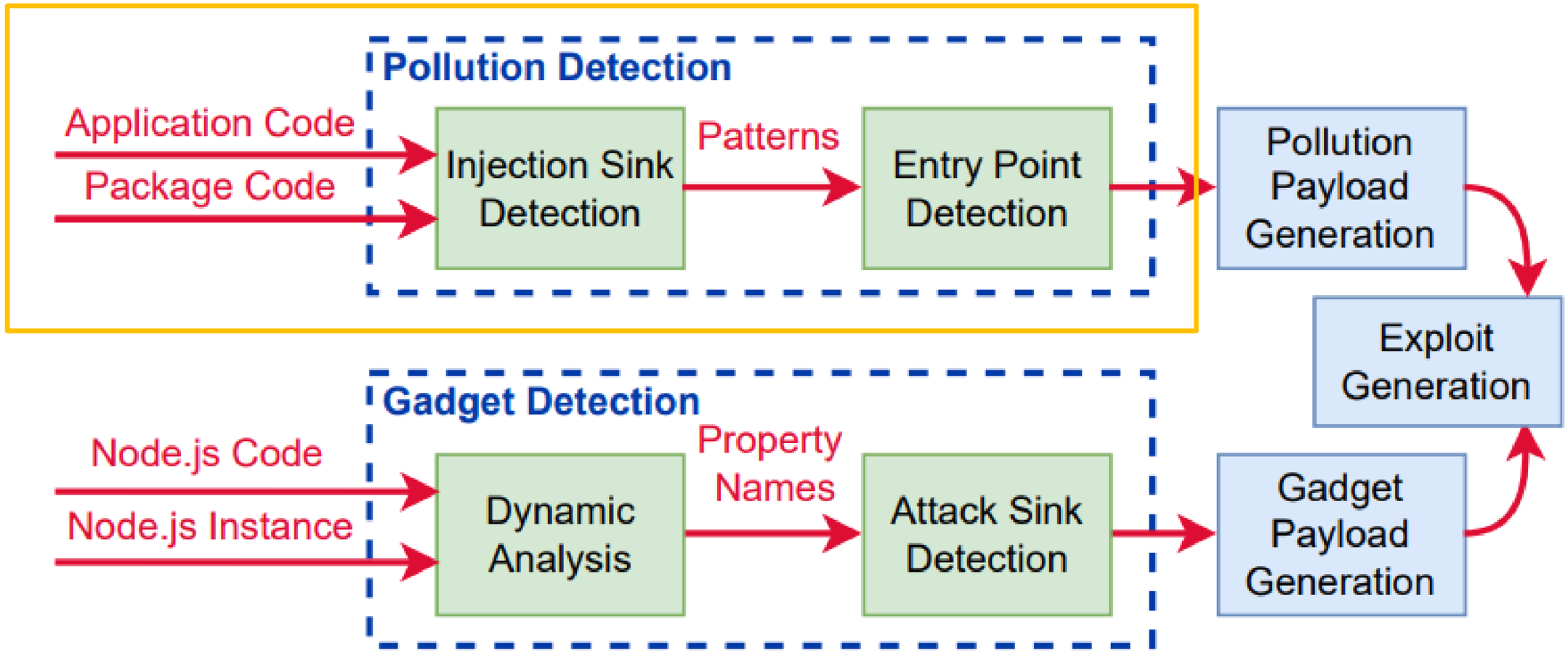
Gadget

```
function execHelper(args, options) {
  const cmd = options.shell ||
    'cmd.exe /k';
  return exec(`${cmd} ${args}`);
}

execHelper('backup-script.bat', {})
```

- **RQ1:** How to identify prototype pollutions in NPM packages and applications?
- **RQ2:** How to identify gadgets in Node.js APIs?
- **RQ3:** How to exploit RCE via prototype pollution in applications?

RQ1: How to identify prototype pollutions in NPM packages and applications?



Static taint analysis

Tracking how sensitive information flows from the sources to target sinks.

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

```
function handleUpdate(arg1, arg2, arg3) {
  const obj = {};
  const p = obj[arg1];
  p[arg2] = arg3;
  /* ... */
}
```



https://github.com/github/codeql

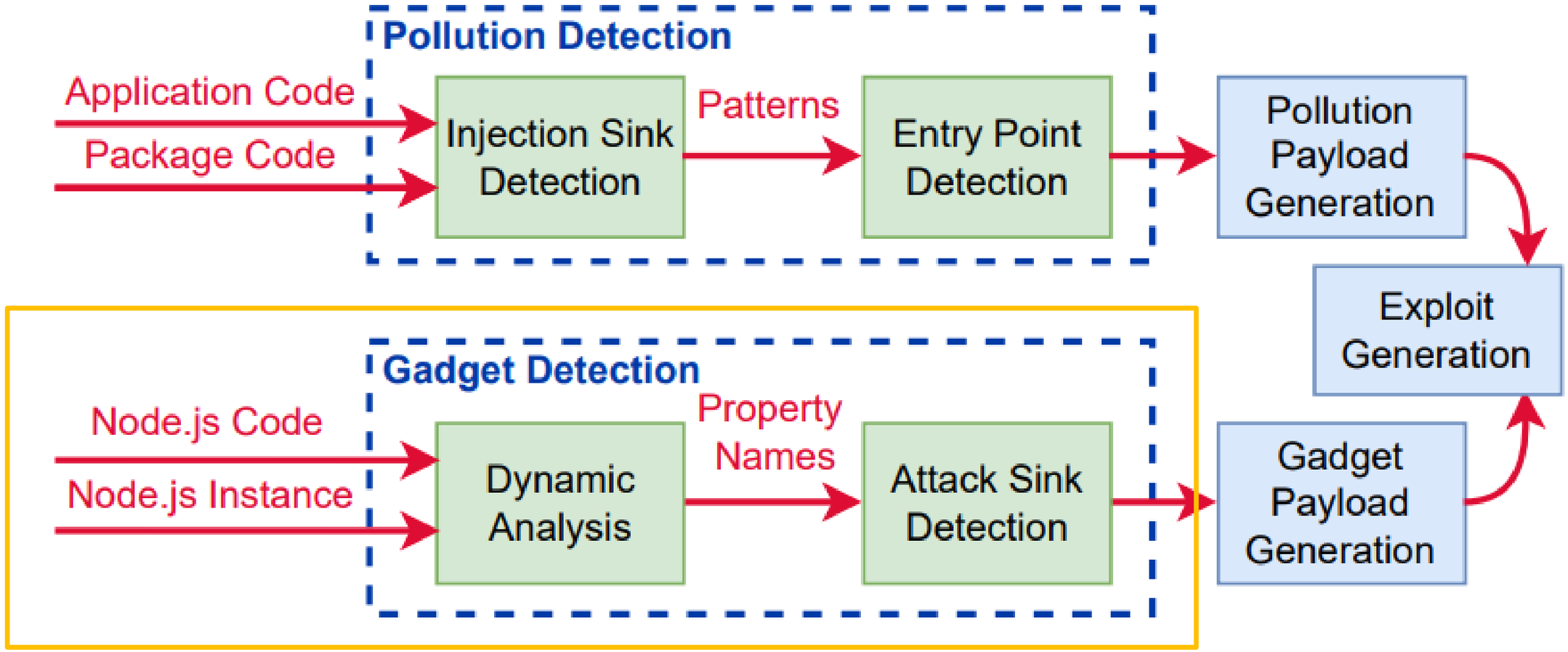
Evaluation of packages analysis

We built a new benchmark from 100 vulnerable Node.js packages and evaluate true positives and false positives metrics for each package.

Metrics	Baseline queries		Priority queries		General queries	
	Prototype Polluting Assignment	Prototype Polluting Function	Exported Functions	Any Functions	Exported Functions	Any Functions
Recall	42.1%	21.3%	82.2%	93.3%	88.4%	97.0%
Precision	46.1%	67.3%	49.6%	40.1%	35.3%	30.9%

The best result achieves 97% recall.

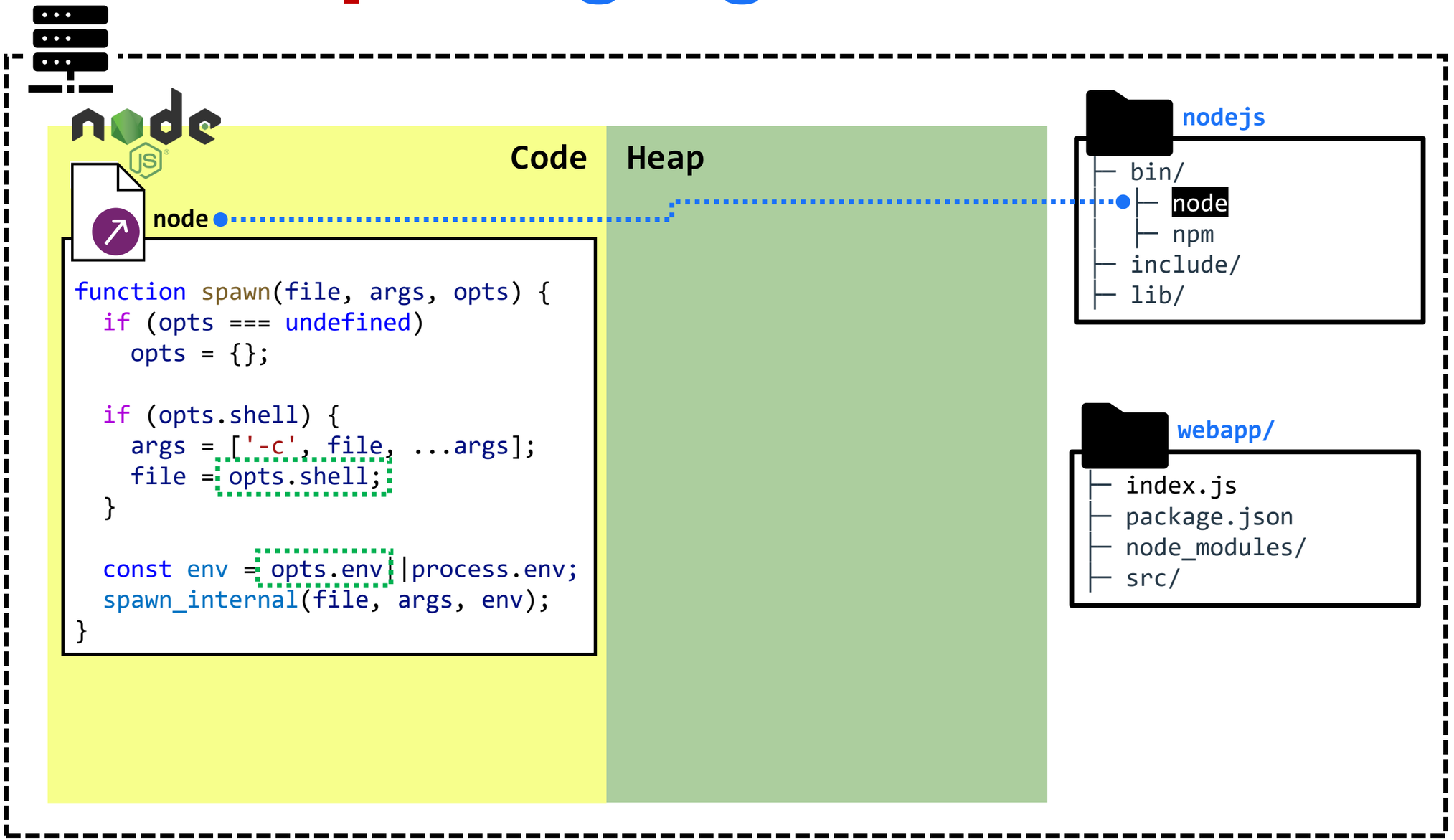
RQ2: How to identify gadgets in Node.js API?



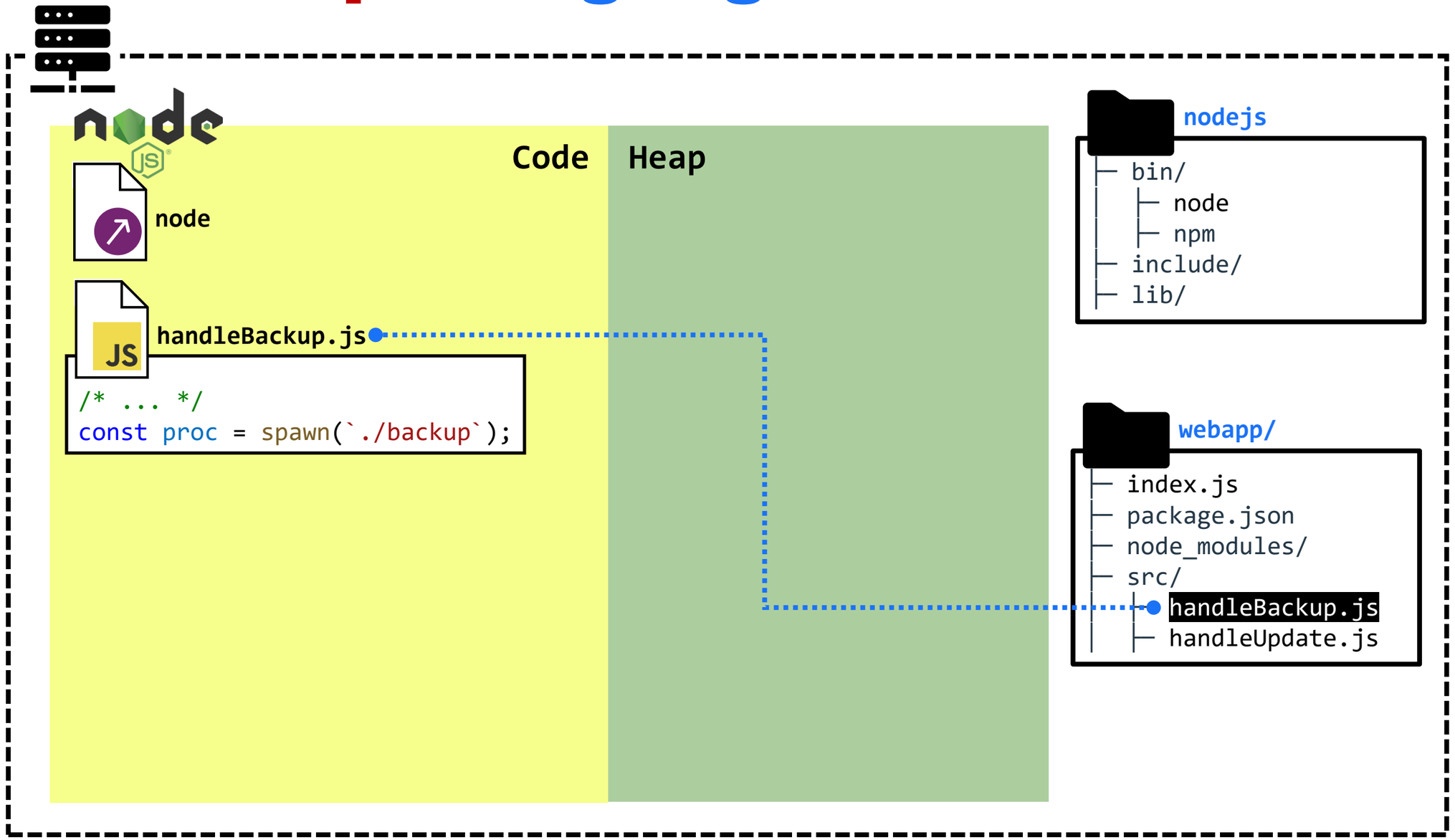
Universal gadgets

Universal properties	Trigger	Impact	OS
`shell, env`	Call command injection API	Execute an arbitrary command	L+W
`shell, env`	Call command injection API	Execute an arbitrary command	L
`shell, input`	Call command injection API	Execute an arbitrary command	W
`main`	Import a package without a declared "main"	Import an arbitrary file from the disk*	L+W
`main`	Require a package without a declared "main"	Require an arbitrary file from the disk*	L+W
`exports, 1`	Require a file using a relative path	Require an arbitrary file from the disk*	L+W
`'=C:'`	Resolve a file path	Resolve the path to a different file	W
`contextExtensions`	Require a file using a relative path	Overwrite global variables of the file	L+W
`contextExtensions`	Compile function in a new context	Overwrite function's global variables	L+W
`shell, env, main`	Require a package without a declared "main"	Execute an arbitrary command	L+W
`shell, env, exports, 1`	Require a file using a relative path	Execute an arbitrary command	L+W

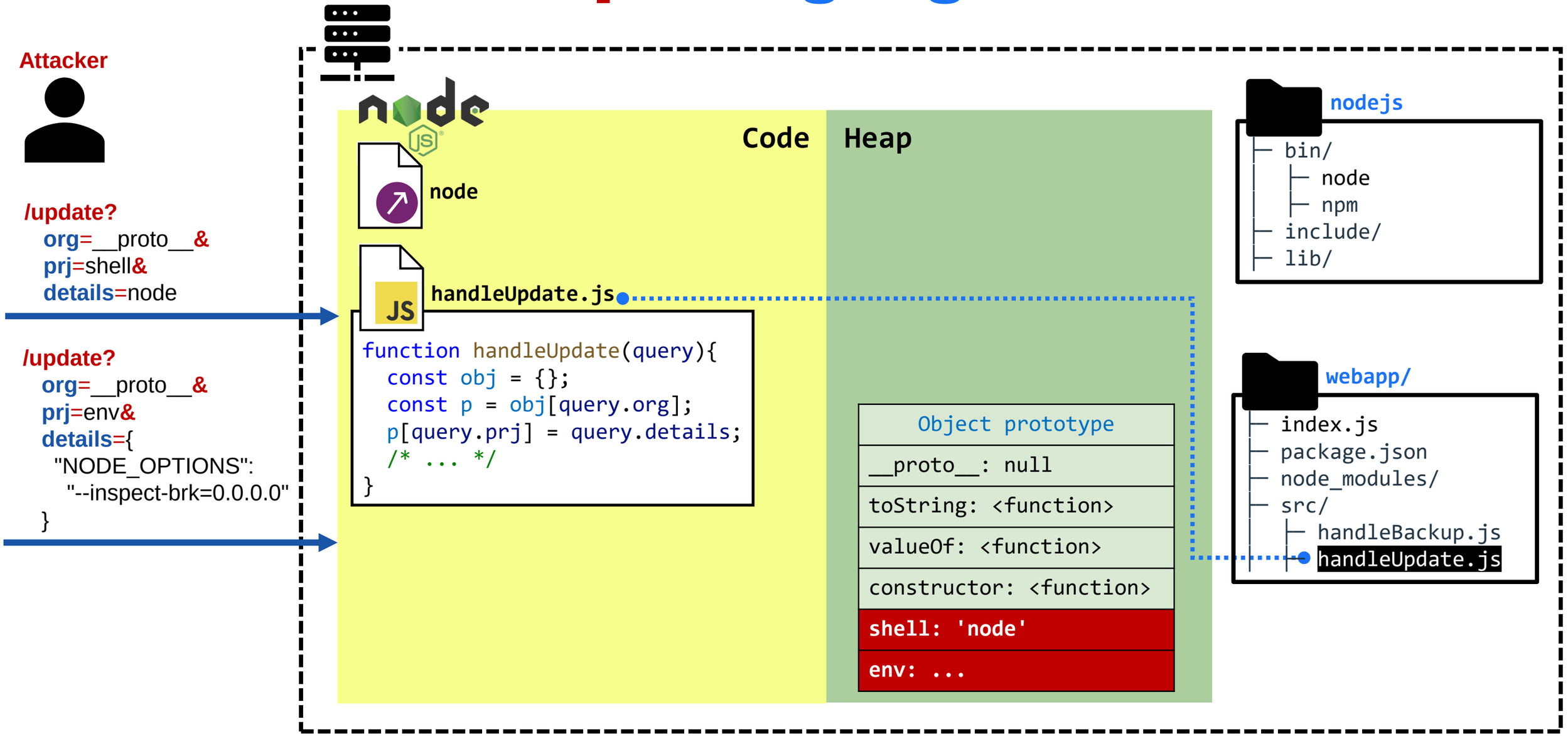
spawn gadget



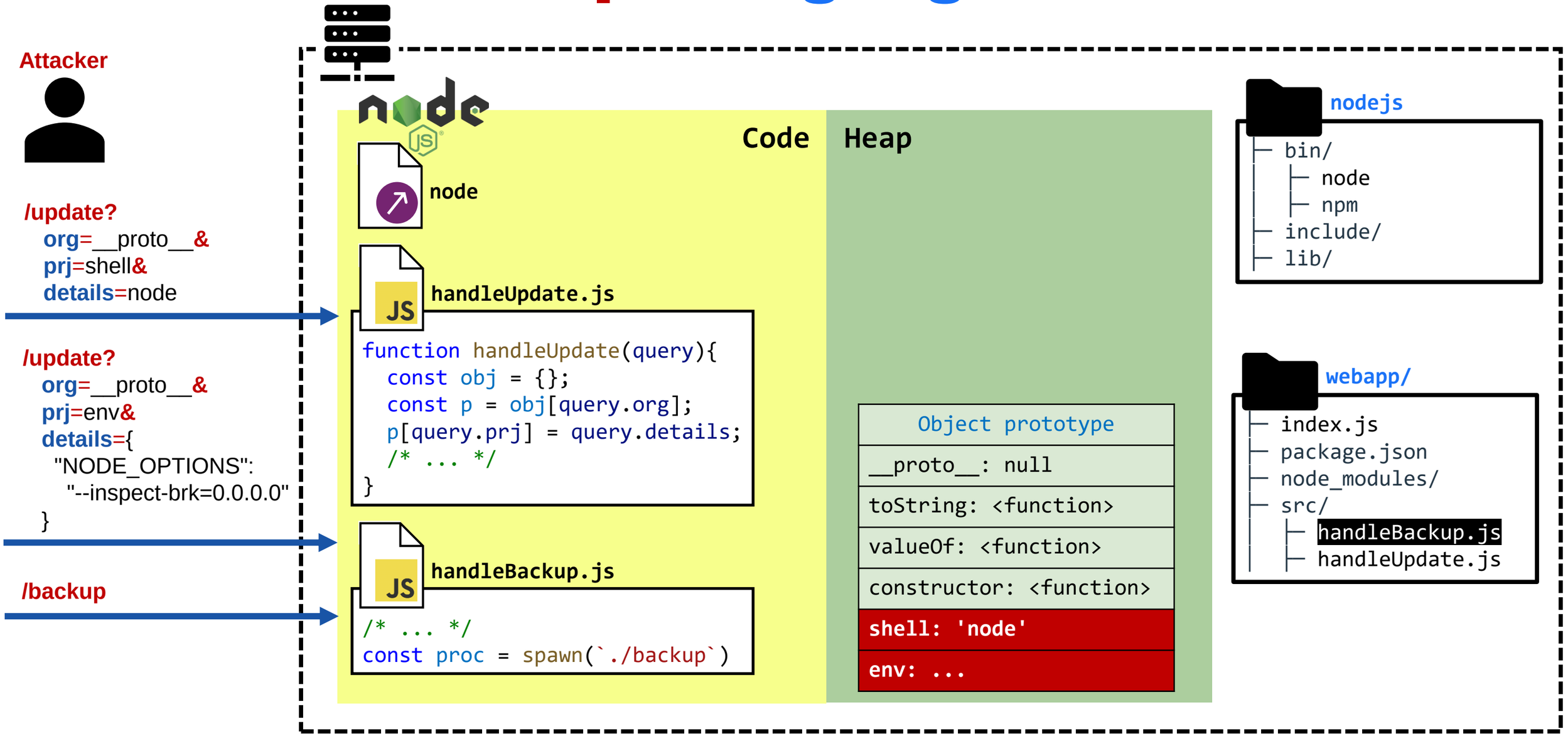
spawn gadget



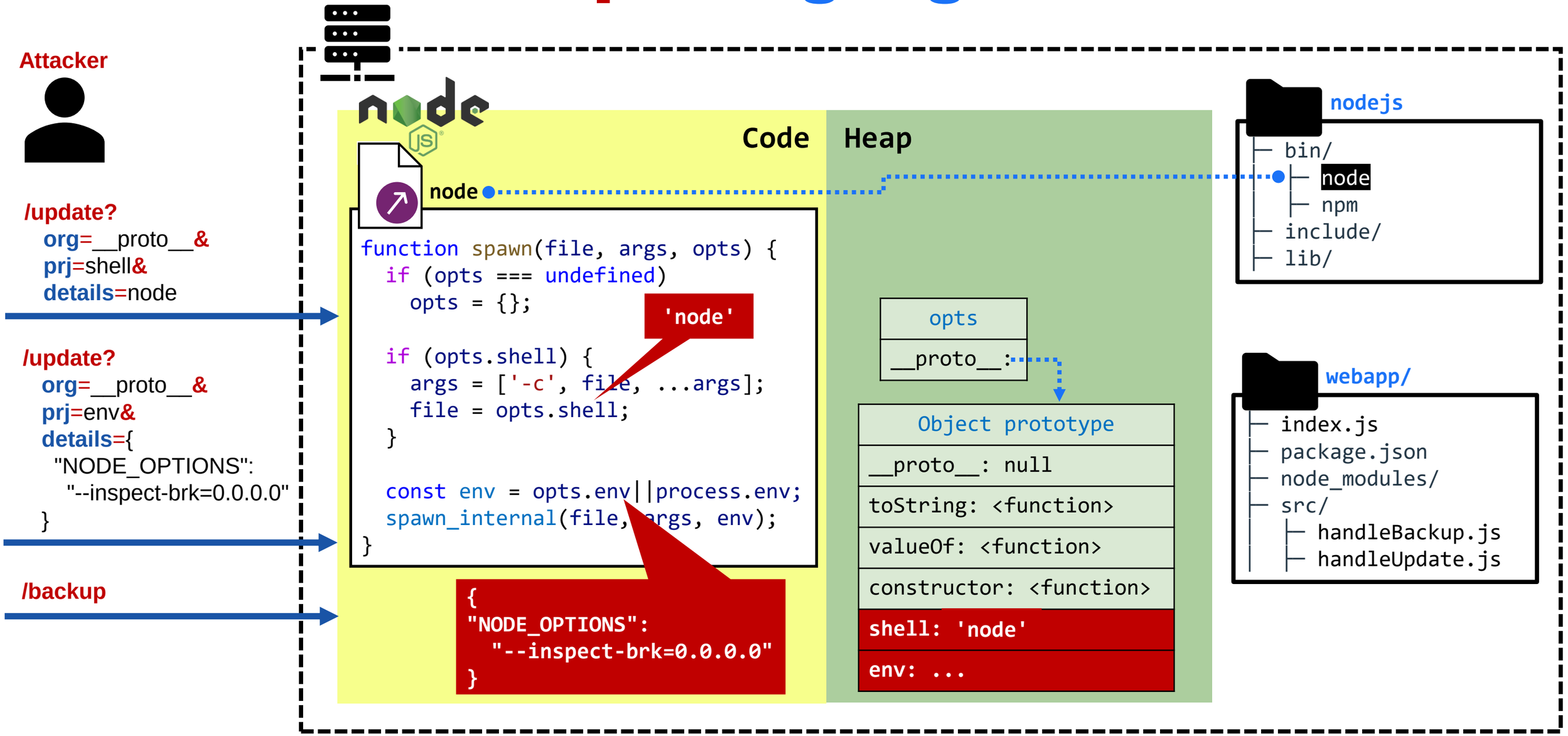
spawn gadget



spawn gadget



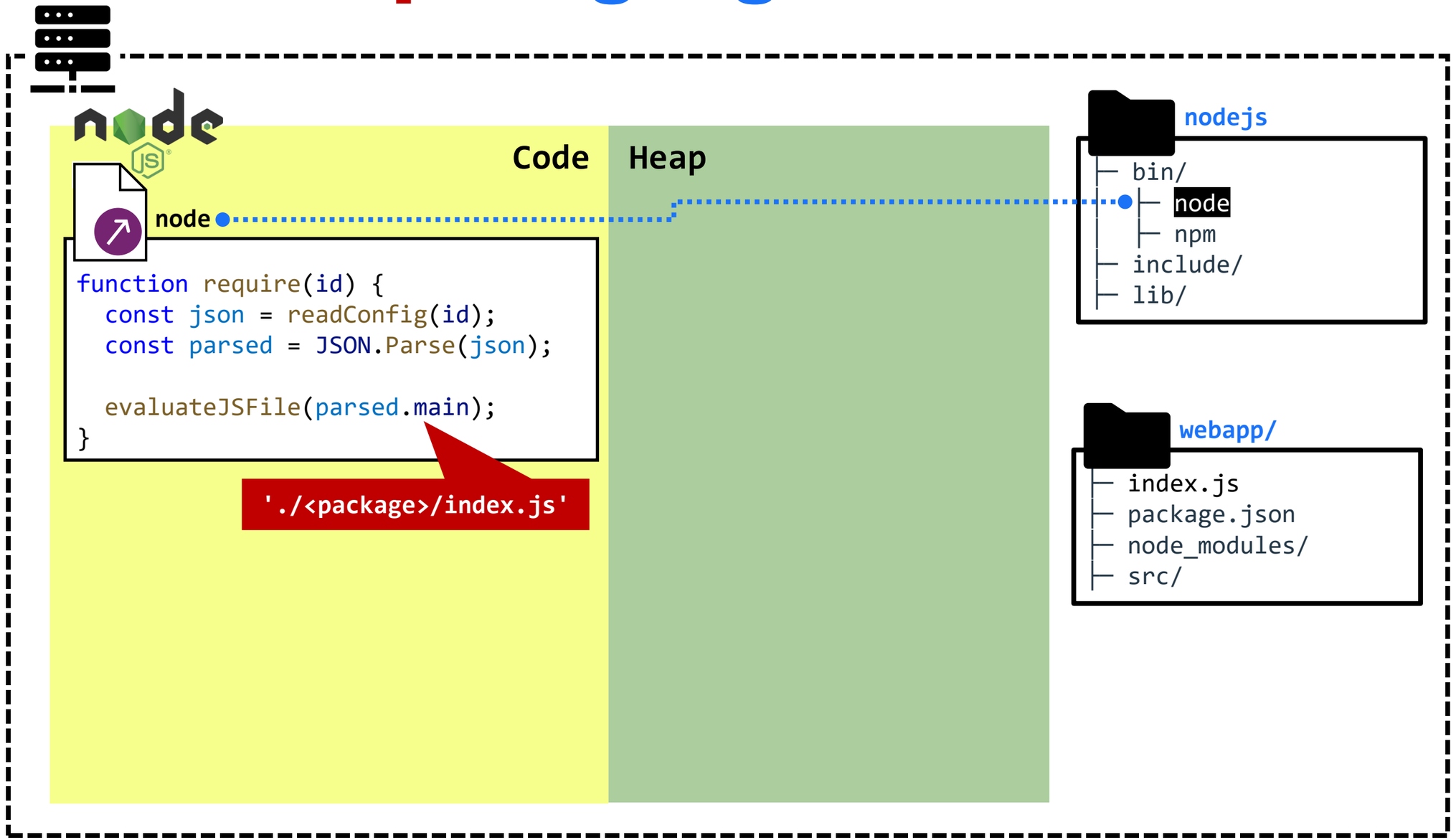
spawn gadget



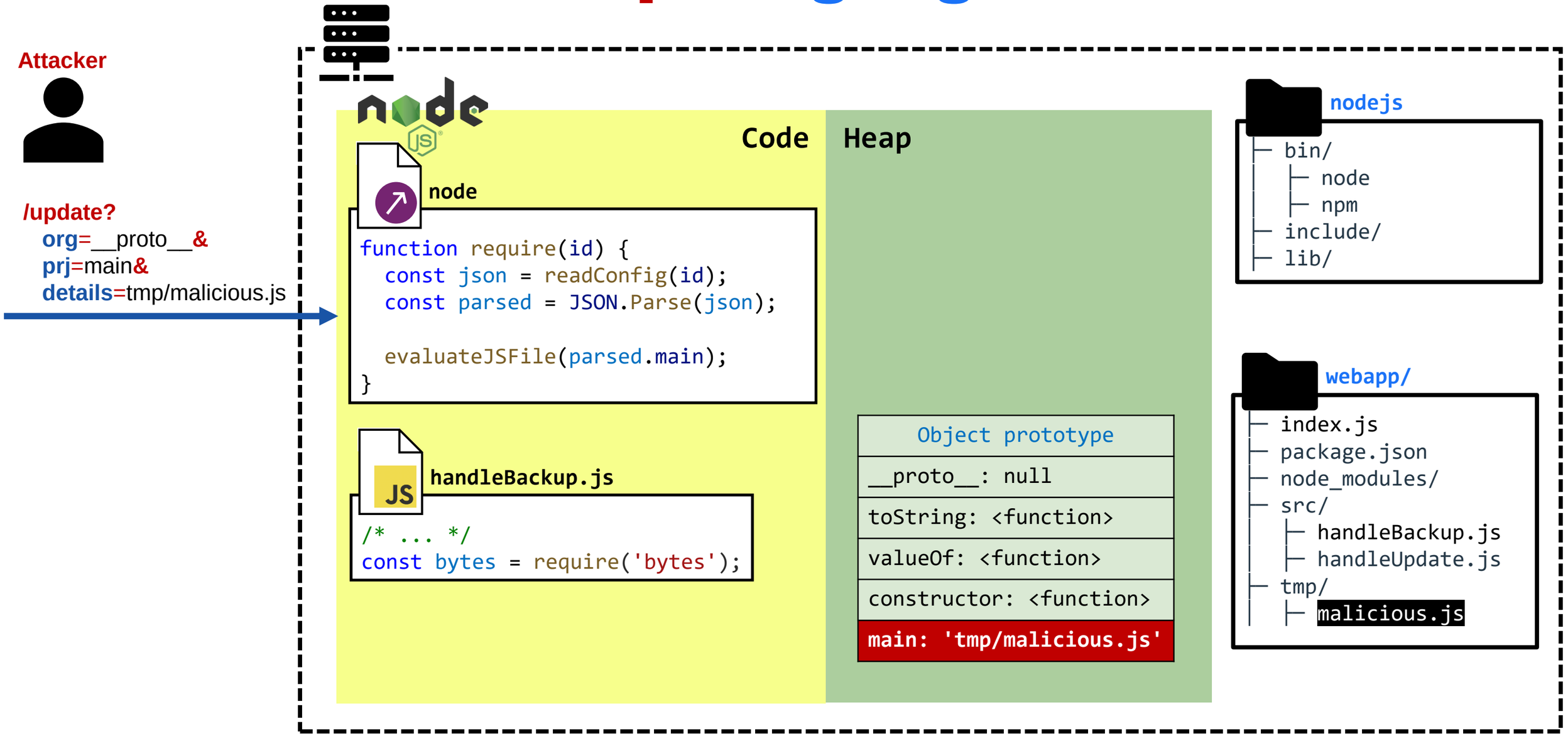
Universal gadgets

Universal properties	Trigger	Impact	OS
`shell, env`	Call command injection API	Execute an arbitrary command	L+W
`shell, env`	Call command injection API	Execute an arbitrary command	L
`shell, input`	Call command injection API	Execute an arbitrary command	W
`main`	Import a package without a declared "main"	Import an arbitrary file from the disk*	L+W
`main`	Require a package without a declared "main"	Require an arbitrary file from the disk*	L+W
`exports, 1`	Require a file using a relative path	Require an arbitrary file from the disk*	L+W
`'=C:'`	Resolve a file path	Resolve the path to a different file	W
`contextExtensions`	Require a file using a relative path	Overwrite global variables of the file	L+W
`contextExtensions`	Compile function in a new context	Overwrite function's global variables	L+W
`shell, env, main`	Require a package without a declared "main"	Execute an arbitrary command	L+W
`shell, env, exports, 1`	Require a file using a relative path	Execute an arbitrary command	L+W

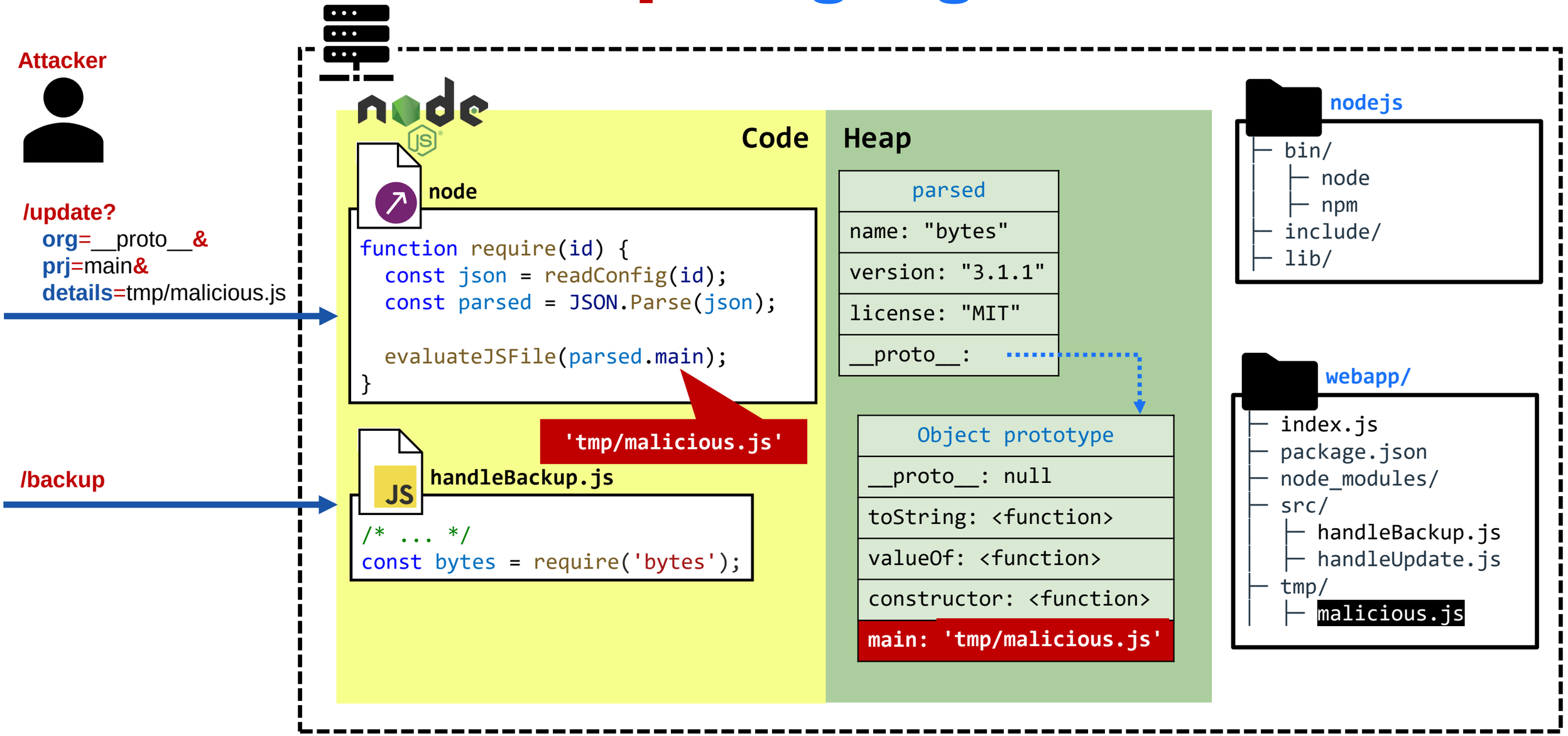
require gadget



require gadget



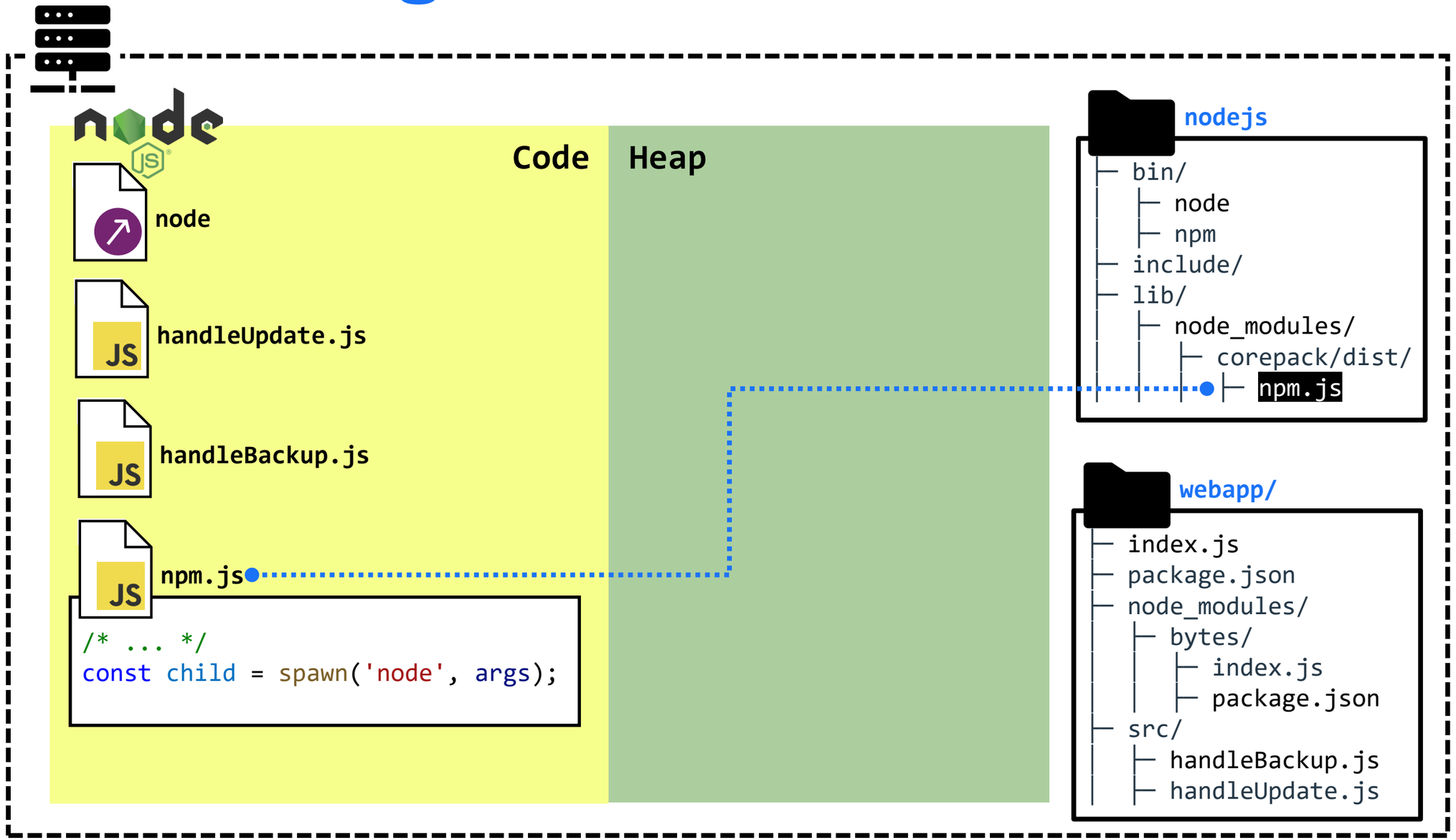
require gadget



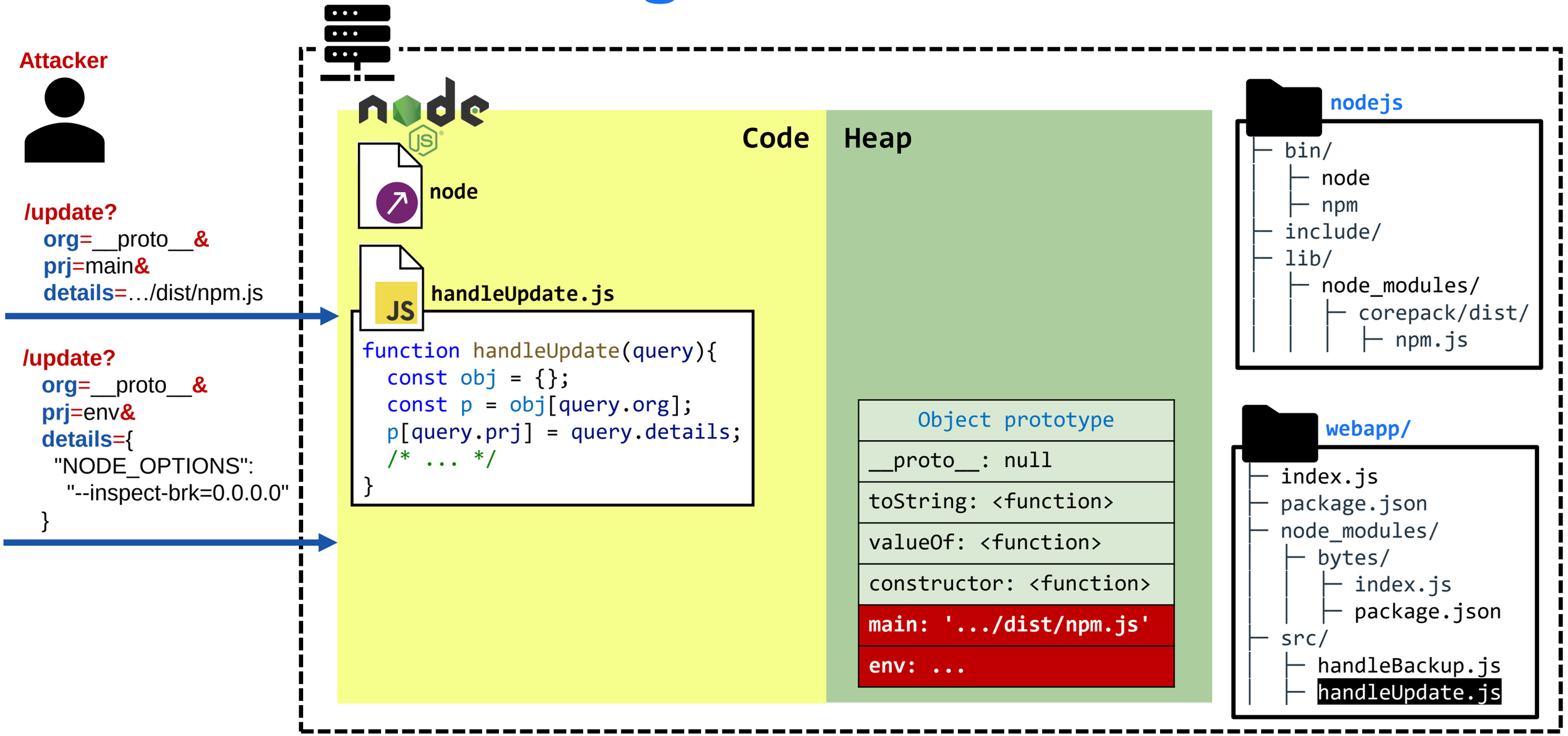
Universal gadgets

Universal properties	Trigger	Impact	OS
`shell, env`	Call command injection API	Execute an arbitrary command	L+W
`shell, env`	Call command injection API	Execute an arbitrary command	L
`shell, input`	Call command injection API	Execute an arbitrary command	W
`main`	Import a package without a declared "main"	Import an arbitrary file from the disk*	L+W
`main`	Require a package without a declared "main"	Require an arbitrary file from the disk*	L+W
`exports, 1`	Require a file using a relative path	Require an arbitrary file from the disk*	L+W
`'=C:'`	Resolve a file path	Resolve the path to a different file	W
`contextExtensions`	Require a file using a relative path	Overwrite global variables of the file	L+W
`contextExtensions`	Compile function in a new context	Overwrite function's global variables	L+W
`shell, env, main`	Require a package without a declared "main"	Execute an arbitrary command	L+W
`shell, env, exports, 1`	Require a file using a relative path	Execute an arbitrary command	L+W

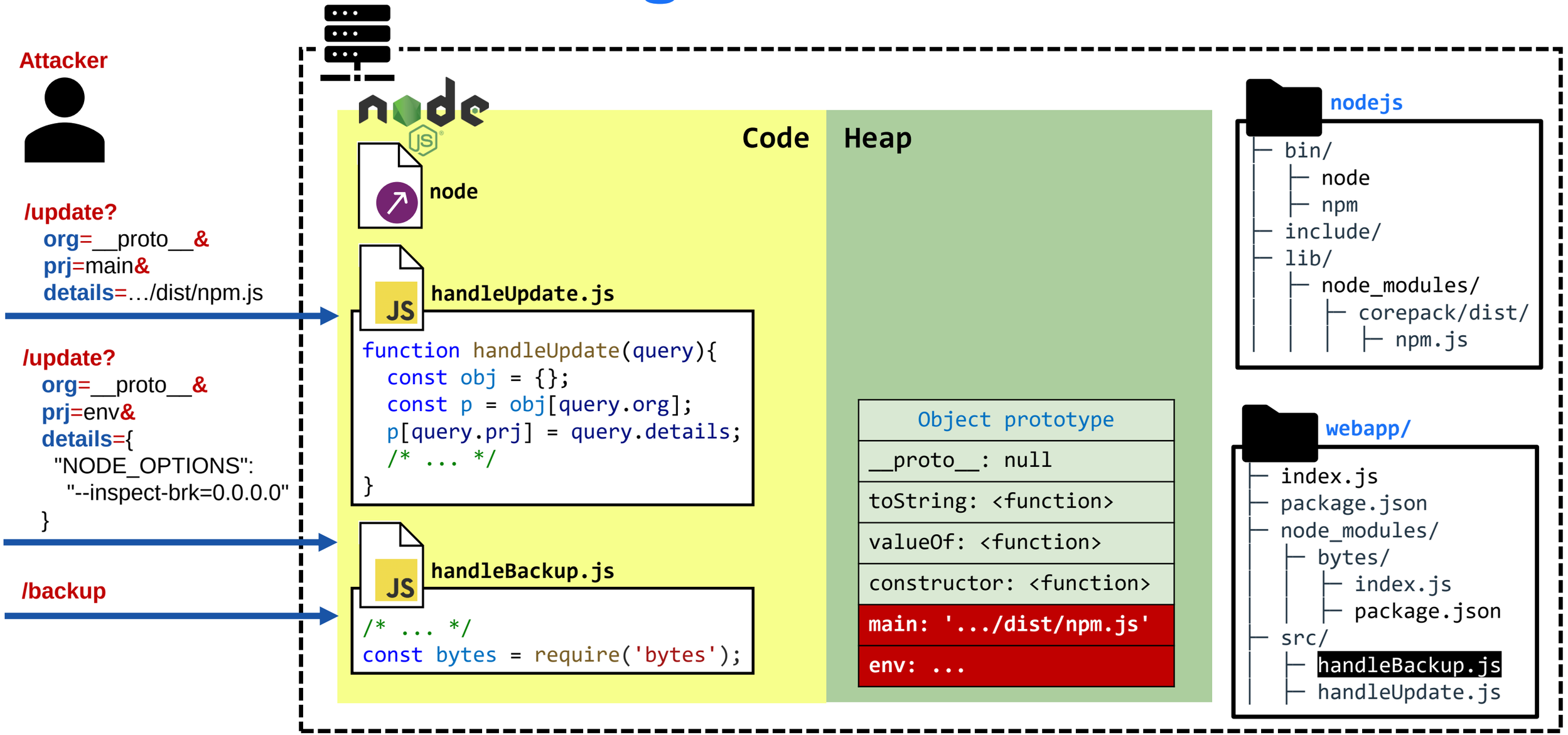
Gadgets cocktail



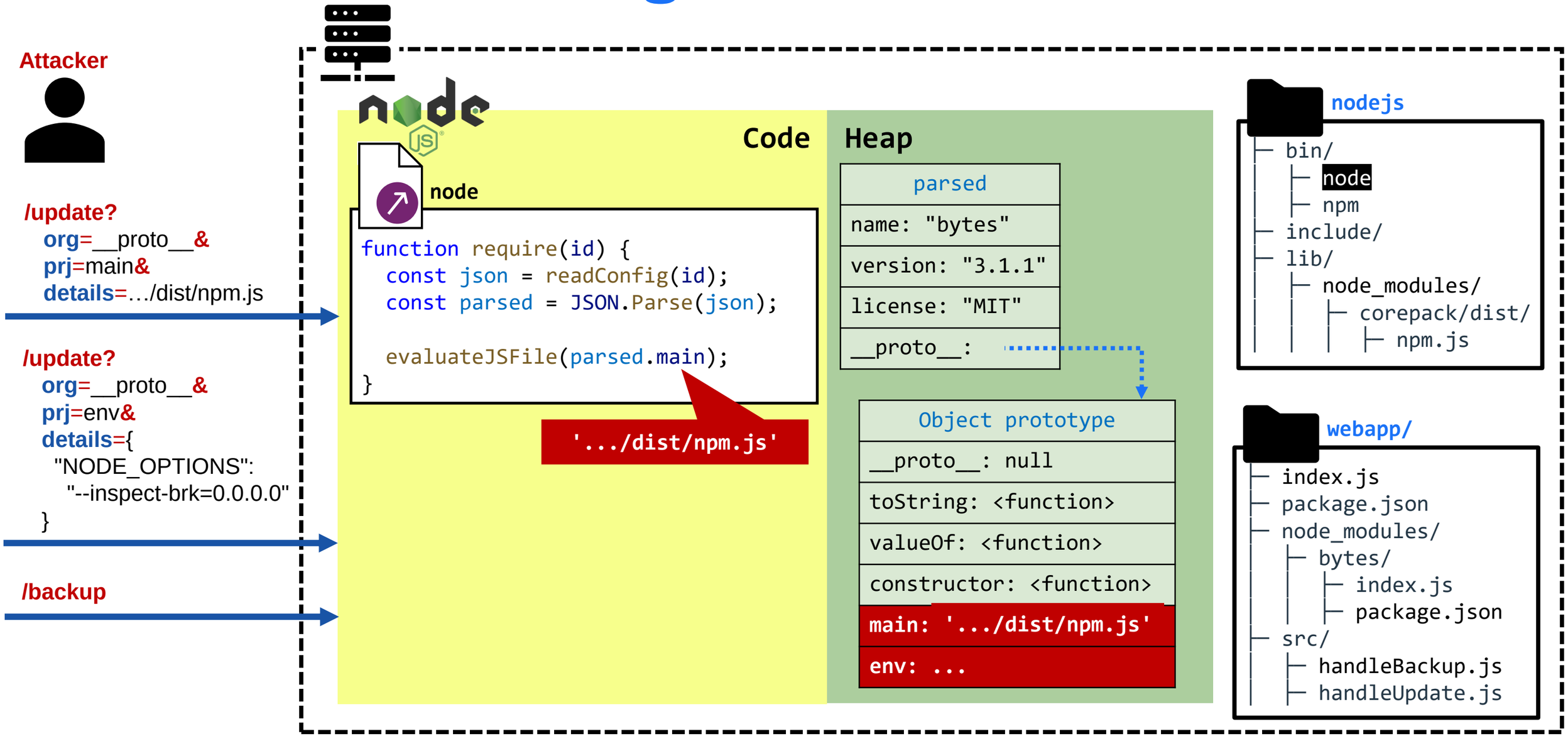
Gadgets cocktail



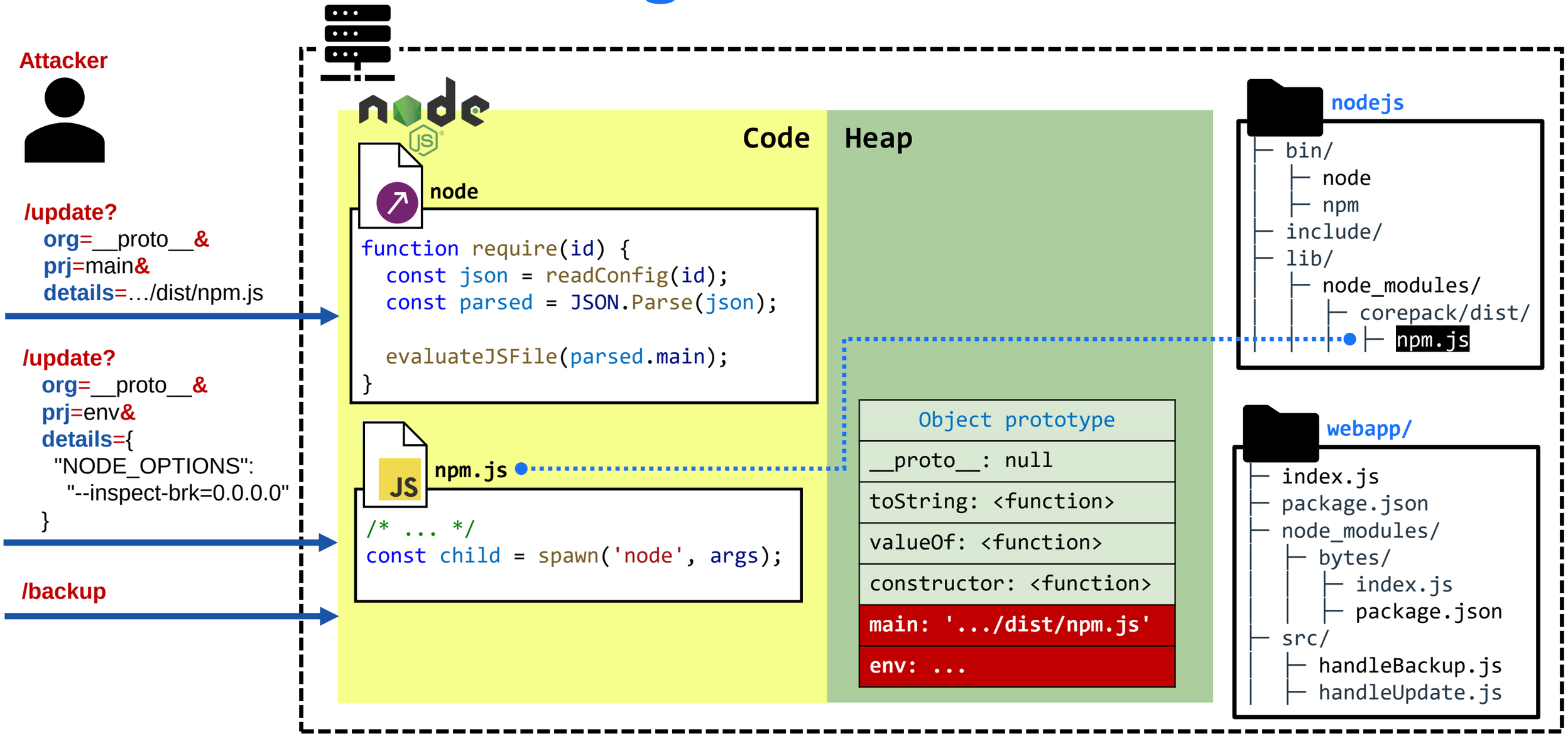
Gadgets cocktail



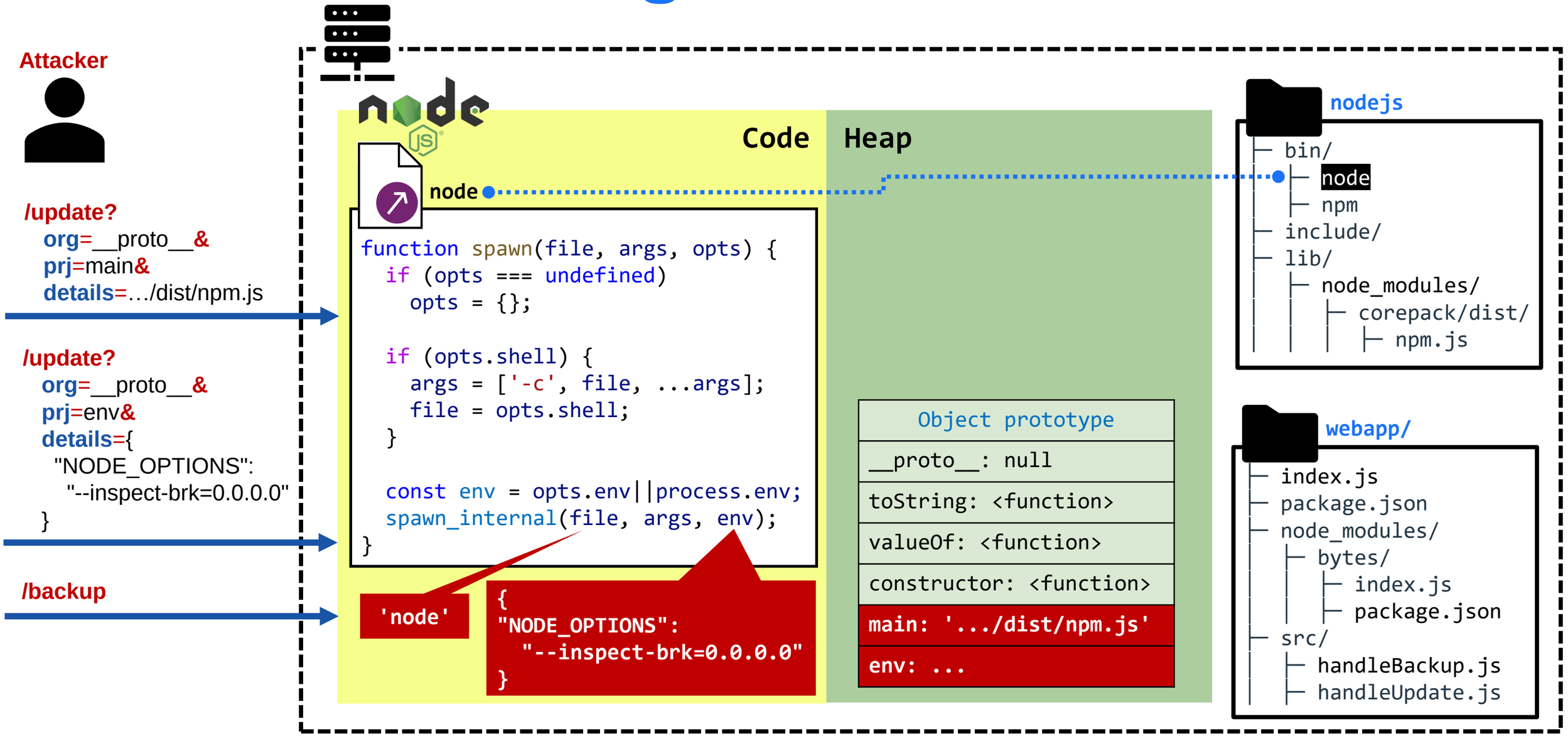
Gadgets cocktail



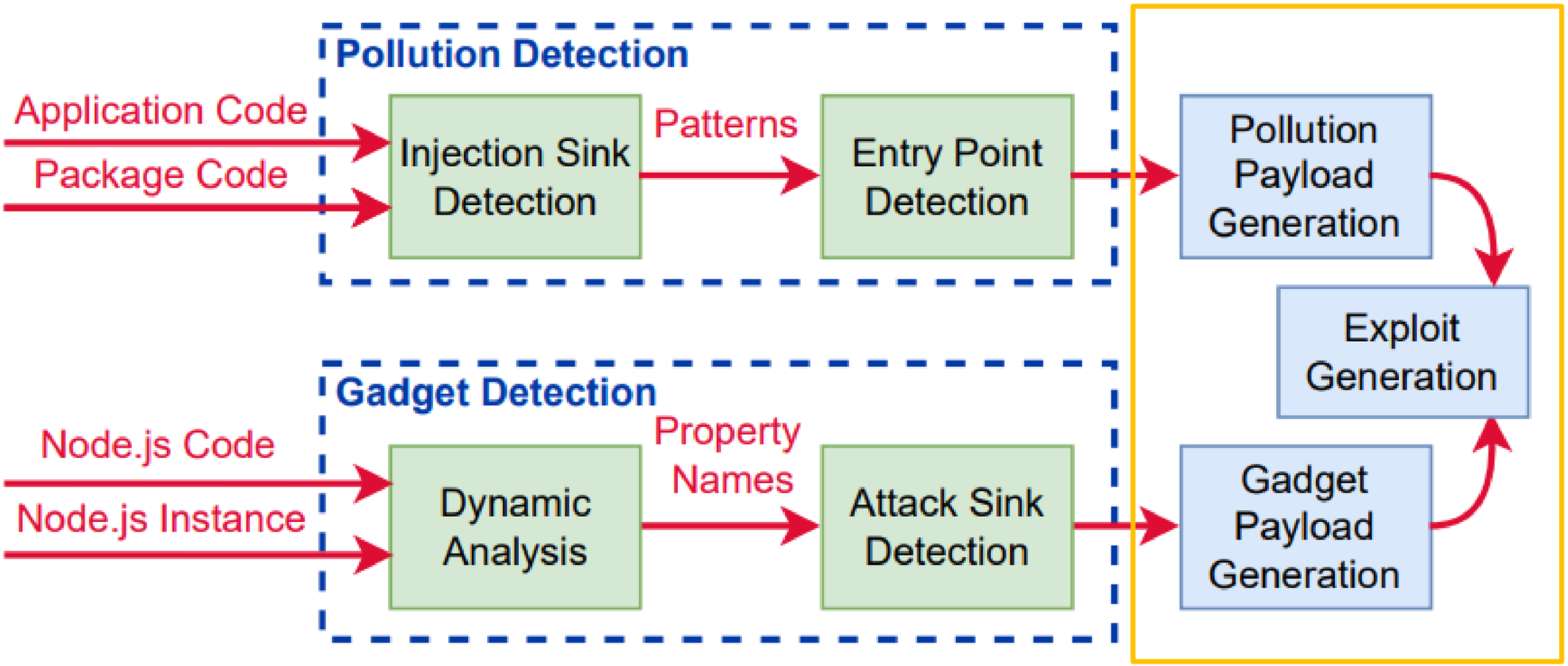
Gadgets cocktail



Gadgets cocktail



RQ3: How to exploit RCE via prototype pollution in applications?



RCE exploits

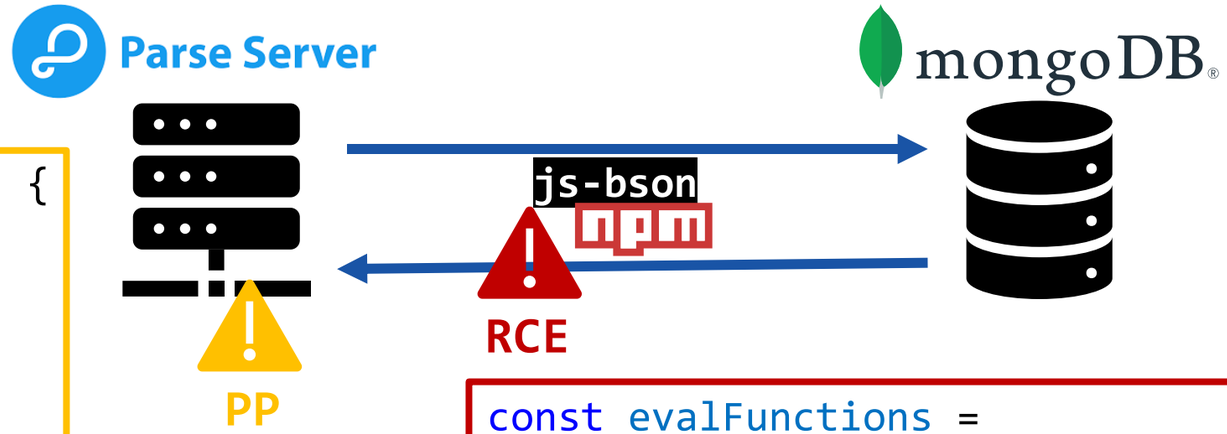
Application's Repository	Stars	Lines of code	Total		Exploitable		Suspicious		Testing Code		Client-Side Code		False Positives	
			Cases	Time	Cases	Time	Cases	Time	Cases	Time	Cases	Time	Cases	Time
typicode/json-server	57,257	2,374	0		-		-		-		-		-	
expressjs/express	54,883	14,450	0		-		-		-		-		-	
meteor/meteor	42,673	202,213	26	255	0		5	210	4	10	8	5	9	30
strapi/strapi	40,724	168,998	3	5	0		0		0		0		3	5
TryGhost/Ghost	38,944	125,696	4	55	0		1	50	0		2	3	1	2
hexojs/hexo	33,666	21,073	1	40	0		1	40	0		0		0	
sahat/hackathon-starter	32,431	2,326	0		-		-		-		-		-	
koajs/koa	31,910	4,596	0		-		-		-		-		-	
RocketChat/Rocket.Chat	31,059	242,949	5	1555	1	1500	3	50	0		1	5	0	
balderdashy/sails	22,085	24,445	0		-		-		-		-		-	
emberjs/ember.js	22,034	113,749	6	60	0		2	40	1	10	0		3	10
fastify/fastify	21,043	37,049	0		-		-		-		-		-	
parse-community/parse-server	19,045	107,909	7	3225	5	3220	0		0		0		2	5
docsifyjs/docsify	18,946	7,603	0		-		-		-		-		-	
npm/cli	5,371	713,648	15	603	2	360	6	230	1	3	0		6	10

- NPM CLI RCE (NO CVEs 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)

Parse Server attacker model

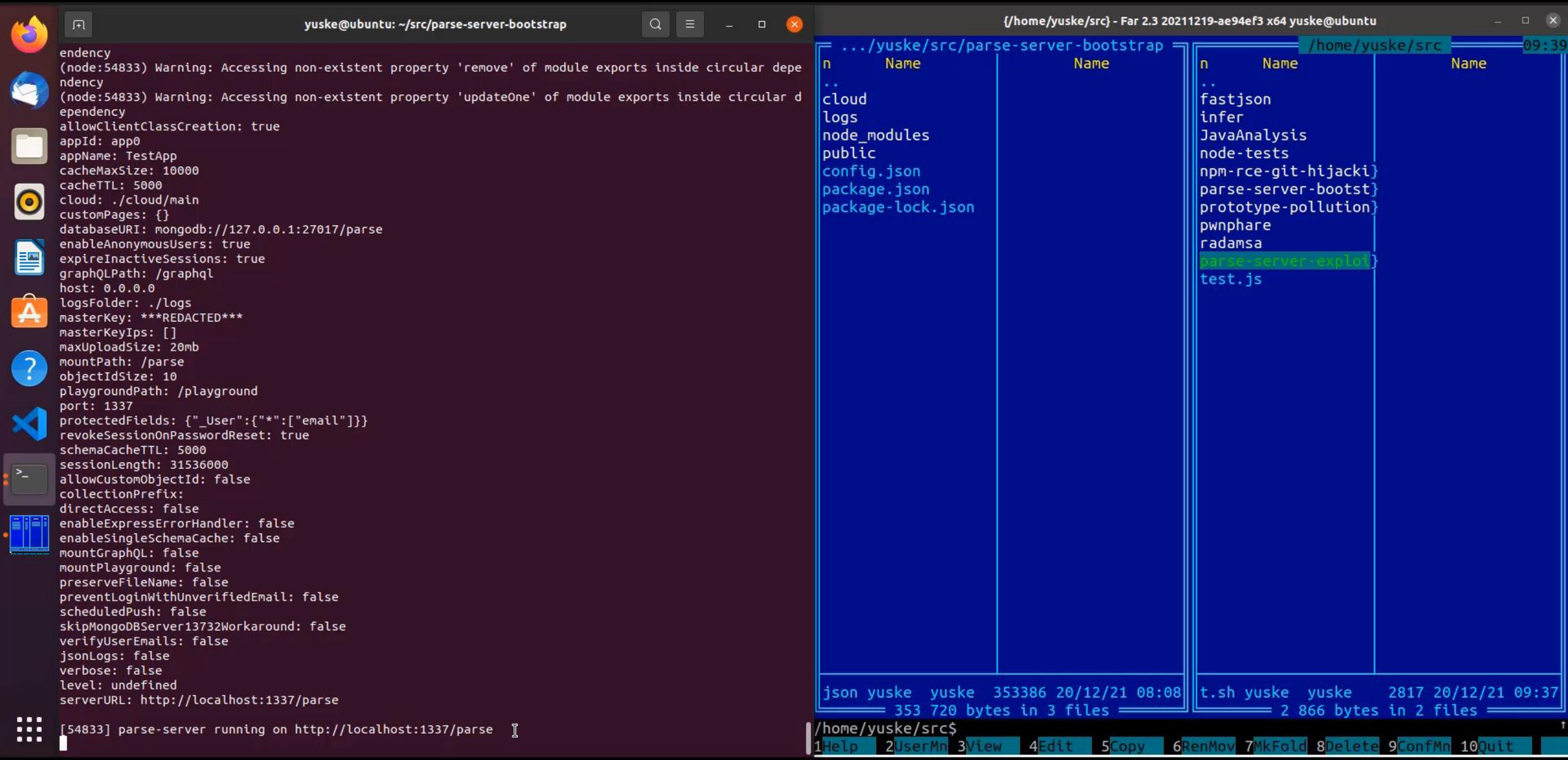
```
function expandResultOnKeyPath(obj, key, res) {
  if (key.indexOf('.') < 0) {
    obj[key] = res[key];
    return obj;
  }

  const path = key.split('.');
  const firstKey = path[0];
  const nextPath = path.slice(1).join('.');
  obj[firstKey] = expandResultOnKeyPath(
    obj[firstKey] || {},
    nextPath, res[firstKey]);
  return obj;
}
```




```
const evalFunctions =
  options['evalFunctions'] == null
    ? false
    : options['evalFunctions’];

if (evalFunctions) {
  eval(functionString);
}
```


yuske@ubuntu: ~/src/parse-server-bootstrap
endency
(node:54833) Warning: Accessing non-existent property 'remove' of module exports inside circular depe
ndency
(node:54833) Warning: Accessing non-existent property 'updateOne' of module exports inside circular d
ependency
allowClientClassCreation: true
appId: app0
appName: TestApp
cacheMaxSize: 10000
cacheTTL: 5000
cloud: ./cloud/main
customPages: {}
databaseURI: mongodb://127.0.0.1:27017/parse
enableAnonymousUsers: true
expireInactiveSessions: true
graphQLPath: /graphql
host: 0.0.0.0
logsFolder: ./logs
masterKey: ***REDACTED***
masterKeyIps: []
maxUploadSize: 20mb
mountPath: /parse
objectIdSize: 10
playgroundPath: /playground
port: 1337
protectedFields: {"_User":{"*":["email"]}}
revokeSessionOnPasswordReset: true
schemaCacheTTL: 5000
sessionLength: 31536000
allowCustomObjectId: false
collectionPrefix:
directAccess: false
enableExpressErrorHandler: false
enableSingleSchemaCache: false
mountGraphQL: false
mountPlayground: false
preserveFileName: false
preventLoginWithUnverifiedEmail: false
scheduledPush: false
skipMongoDBServer13732Workaround: false
verifyUserEmails: false
jsonLogs: false
verbose: false
level: undefined
serverURL: http://localhost:1337/parse
[54833] parse-server running on http://localhost:1337/parse
{/home/yuske/src} - Far 2.3 20211219-ae94ef3 x64 yuske@ubuntu
.../yuske/src/parse-server-bootstrap
n Name Name
..
cloud
logs
node_modules
public
config.json
package.json
package-lock.json
json yuske yuske 353386 20/12/21 08:08
353 720 bytes in 3 files
/home/yuske/src
09:39
n Name Name
..
fastjson
infer
JavaAnalysis
node-tests
npm-rce-git-hijacki}
parse-server-bootst}
prototype-pollution}
pwnphare
radamsa
parse-server-exploi}
test.js
t.sh yuske yuske 2817 20/12/21 09:37
2 866 bytes in 2 files
/home/yuske/src$
1Help 2UserMn 3View 4Edit 5Copy 6RenMov 7MkFold 8Delete 9ConfMn 10Quit

Conclusion and Future Work

- We designed a toolchain for detecting RCE vulnerabilities that are enabled by prototype pollution.

  https://github.com/yuske/silent-spring

- We found 11 universal gadgets in Node.js APIs source code.

  https://github.com/yuske/server-side-prototype-pollution

- We reported 8 RCEs in the popular open-source applications: NPM CLI, Parse Server and Rocket.Chat.

- What is a more efficient way to detect gadgets that allows us to analyze billions of lines of code?
- How many new gadgets are in NPM packages **dependencies**?
- How many new gadgets are in **Node.js core** source code?

Conclusion and Future Work

- We designed a toolchain for detecting RCE vulnerabilities that are enabled by prototype pollution.

  https://github.com/yuske/silent-spring

- We found 11 universal gadgets in Node.js APIs source code.

  https://github.com/yuske/server-side-prototype-pollution

- We reported 8 RCEs in the popular open-source applications: NPM CLI, Parse Server and Rocket.Chat.

- What is a more efficient way to detect gadgets that allows us to analyze billions of lines of code?
- How many new gadgets are in NPM packages **dependencies**?
- How many new gadgets are in **Node.js core** source code?

Thanks for your attention!

https://twitter.com/yu5k3