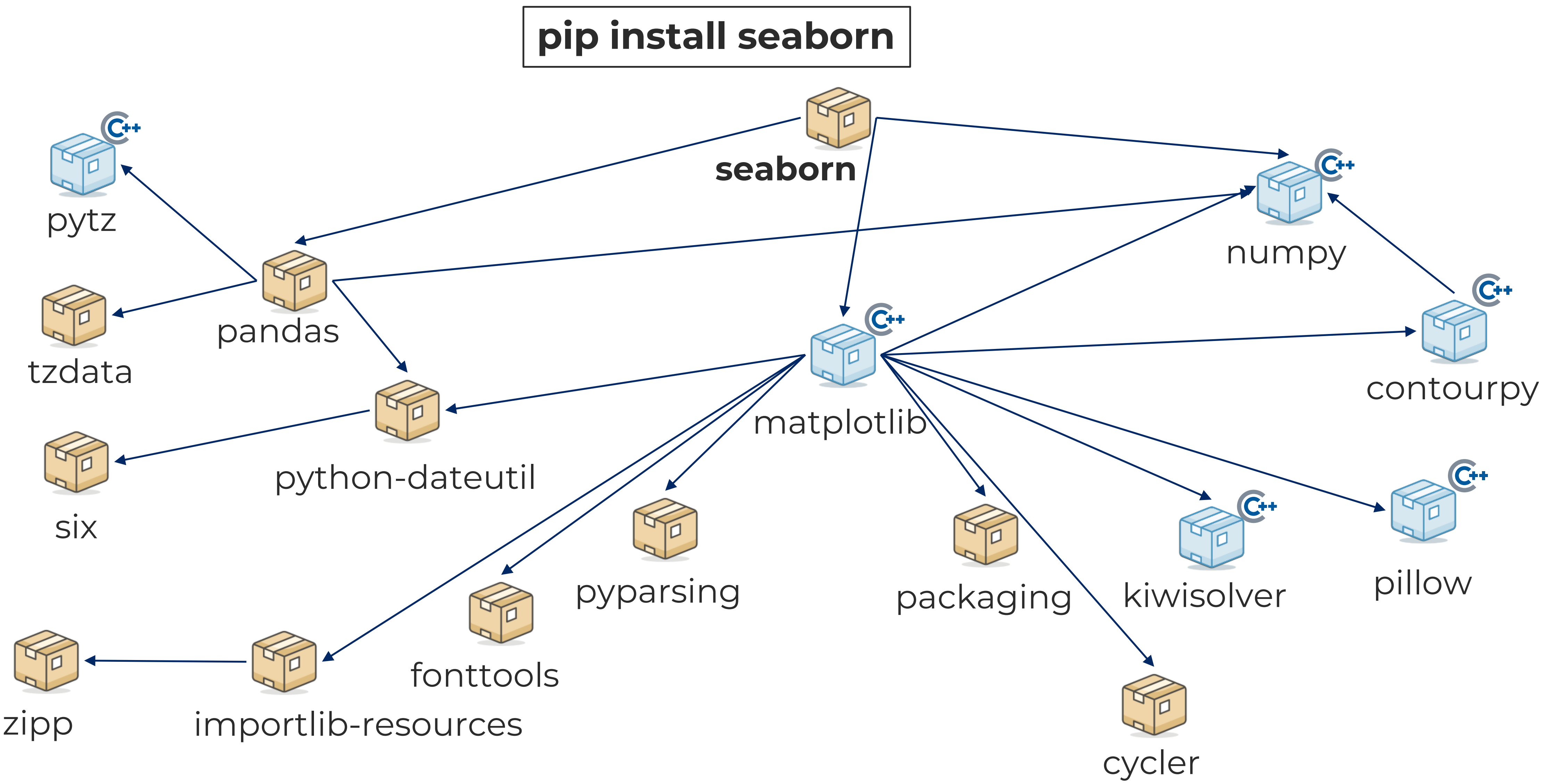
Bilingual Problems: Studying the Security Risks Incurred by Native Extensions in Scripting Languages

Joint work with Sazzadur Rahaman, Ágnes Kiss and Michael Backes

Cristian-Alexandru Staicu | USENIX Security 2023 | 11 August 2023

Modern software supply chains



https://deps.dev/pypi/seaborn/0.12.2/dependencies/graph

Native extensions

Why?

- High-performance code,
- Expose hardware capabilities,
- Mature, legacy code from a low-level language.

How?

- Compile at installation time,
- Often supports both C and C++ code via specialized bindings,
- Expose low-level code to scripting language via API calls,
- Often run inside the same process.

What can go wrong?

- Break guarantees of the scripting language,
- Inexperienced developers may **misuse native extensions**.

Relying on a package with native extension

Client application

```
let nlib = require(`nativepad’);
nlib(`foo’);          // returns “foopad”
nlib(`foo \0 bar’); // “foo” followed by three
uninitialized bytes
nlib(true); // four uninitialized bytes
nlib({toString : 42}); // segfault
```

JS

Third-party dependency

```
let addon = require(`bindings’)(`addon.node’);
module.exports = (str) => {
   if (!str)
      throw `Invalid string’;
   return addon.Pad(str);
}
```

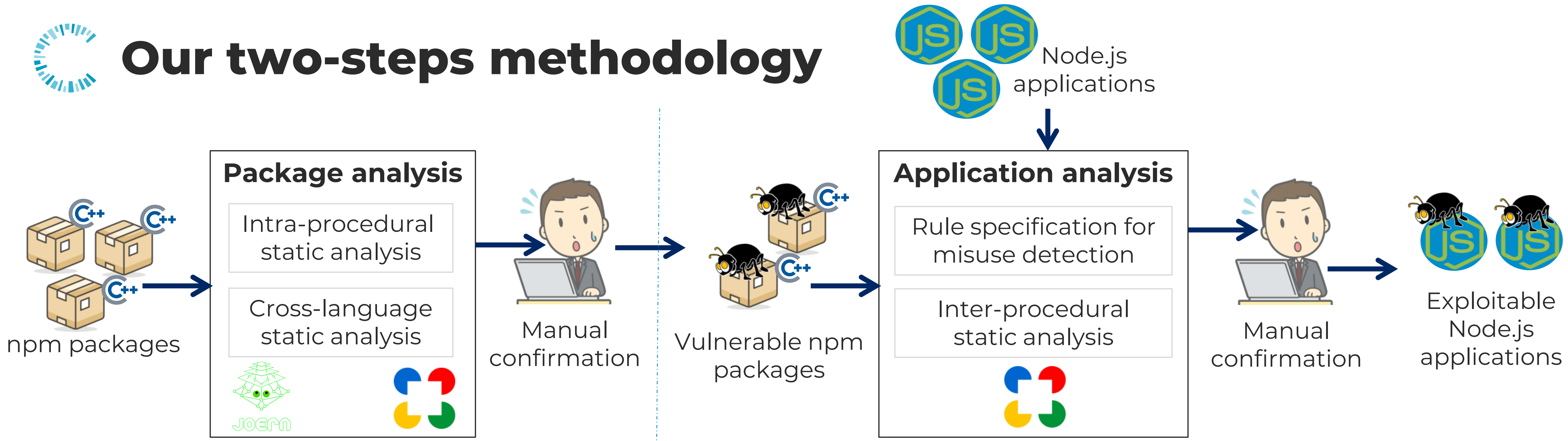
JS

```
napi_value Pad( napi_env env, napi_callback_info info) {
   napi_status status;
   size_t argc = 1, strSize;
   napi_value args[1], result;
   status = napi_get_cb_info(env, info, &argc, args,
NULL, NULL);
   assert(status == napi_ok);
   napi_get_value_string_utf8(env, args[0], NULL, NULL,
&strSize);
   strSize = strSize + 4;
   char myStr[strSize];
   napi_get_value_string_utf8(env, args[0], myStr,
strSize, NULL);
   strcat(myStr, "pad");
   napi_create_string_utf8(env, myStr, strSize,
&result);
   return result;
}
```

Study of misuses in different languages

Type	Id	Misuse	Node.js-N-API	Node.js-Nan	Python	Ruby	Severity
Errors	M_1	Not catching C++ exceptions	●	●	●*	N/a	Low
	M_2	Not handling runtime errors in C/C++	●	●	●	●	Medium
Arguments	M_3	Passing arguments with a wrong type	●	●	○	○	High
	M_4	Passing wrong number of arguments	●	◐	○	○	High
	M_5	Not accounting for different semantics of \0	●	●	○	○	High
	M_6	Passing arguments that overflow numeric types	●	●	○	○	High
Ret.	M_7	Missing return statement	●	○	◐	●	Low
	M_8	Declaring interface methods that return void	○	○	◐	○	Low
Mem.	M_9	Returning uninitialized memory values	●	○	●	○	Medium
	M_{10}	Mismanagement of cross-language pointers	●	○	●	○	Low
High-level	M_{11}	Producing unexpected side-effects in the runtime	○	○	○	●	High
	M_{12}	Blocking the runtime with slow cross-language calls	●	●	●	●	Medium
Low-level	M_{13}	Reading outside of an allocated buffer	●	○	○	○	High
	M_{14}	Using a pointer after it was freed	●	●	●	◐	High
	M_{15}	Freeing a pointer twice	◐	◐	◐	◐	High
	M_{16}	Failing to deallocate unused memory	●	●	●	●	Low
	M_{17}	Interpreting user input as format string	◐	◐	○	○	High

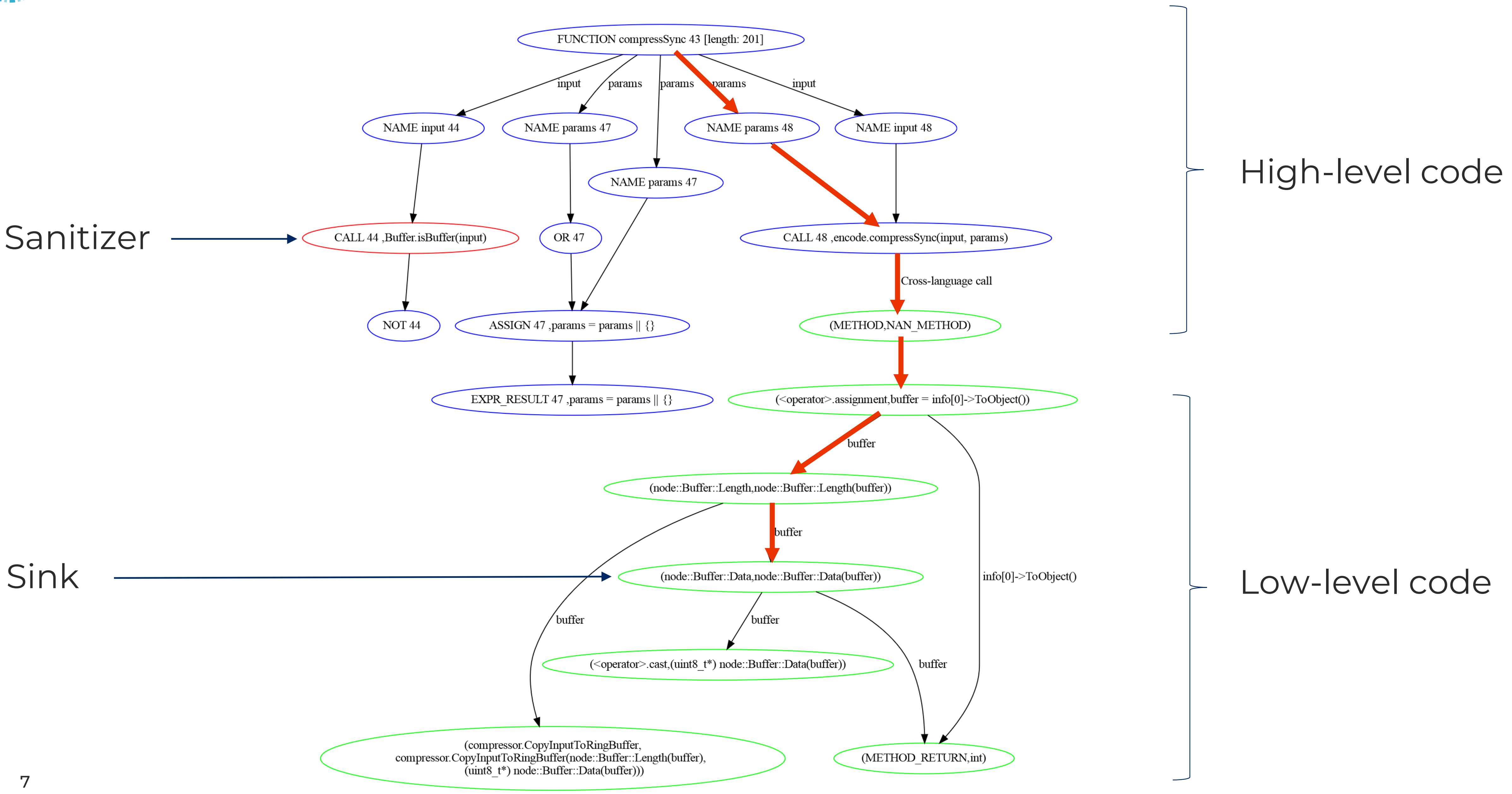
Our two-steps methodology


Node.js applications
Package analysis
Intra-procedural static analysis
Cross-language static analysis
JOERN
npm packages
Manual confirmation
Vulnerable npm packages
Application analysis
Rule specification for misuse detection
Inter-procedural static analysis
Manual confirmation
Exploitable Node.js applications

- Path-insensitive **data-flow analysis**,
- Nine sinks and five sanitizers,
- We **map cross-language calls** using native extensions' definition,
- For each confirmed vulnerability, we provide **a hard crash** of the runtime.

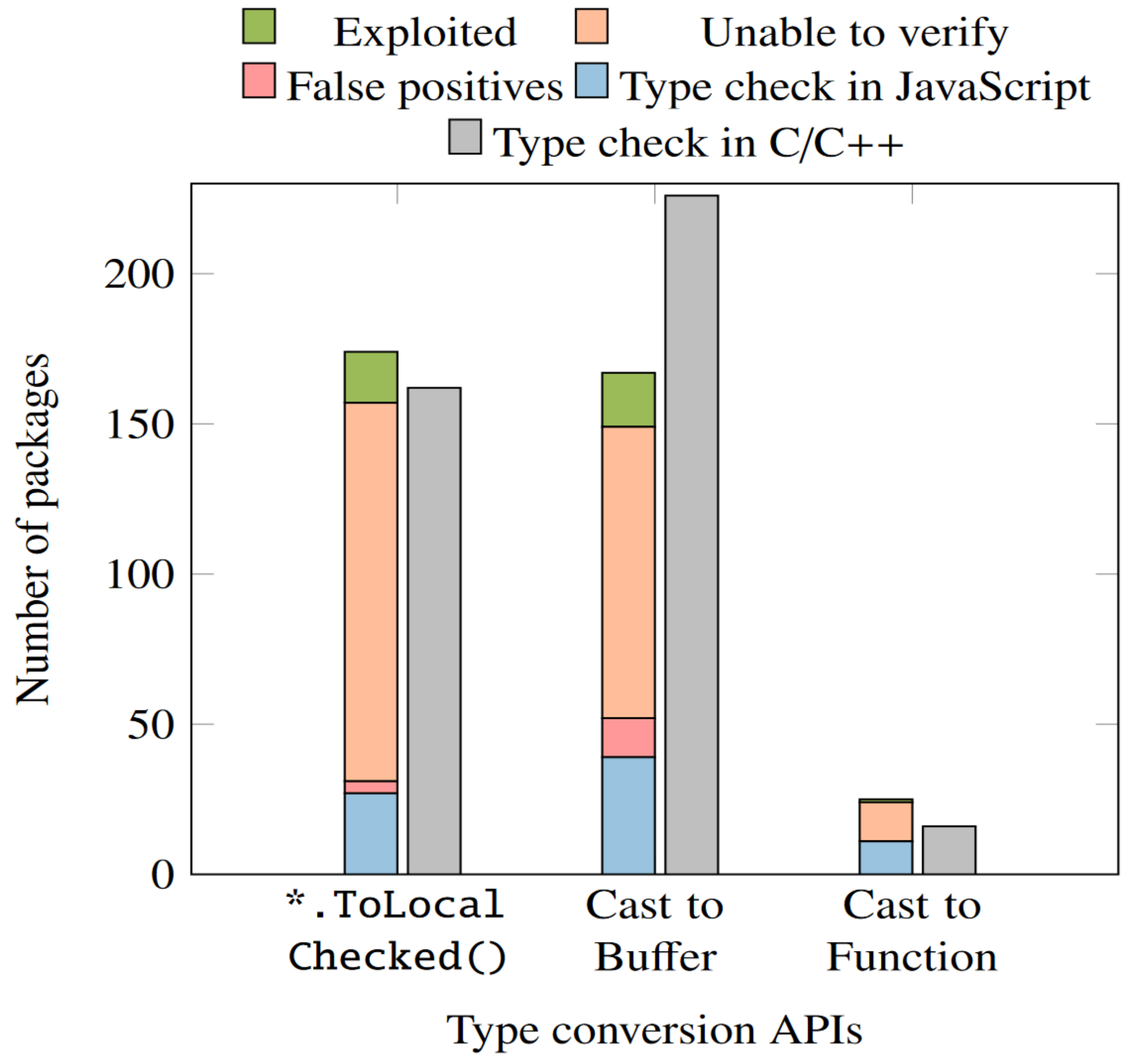
- Further analyzed vulnerable packages **confirmed by their maintainers**,
- **Demand-driven**, backward def-use analysis,
- Specify each confirmed vulnerability as a **rule for the static analysis**,
- Verify that the hard crash can be triggered remotely.

Cross-language data-flow graph



FUNCTION compressSync 43 [length: 201]
input
params
params
params
input
NAME input 44
NAME params 47
NAME params 48
NAME input 48
NAME params 47
Sanitizer
CALL 44 ,Buffer.isBuffer(input)
OR 47
CALL 48 ,encode.compressSync(input, params)
Cross-language call
NOT 44
ASSIGN 47 ,params = params || {}
(METHOD,NAN_METHOD)
EXPR_RESULT 47 ,params = params || {}
(<operator>.assignment,buffer = info[0]->ToObject())
buffer
(node::Buffer::Length,node::Buffer::Length(buffer))
buffer
Sink
(node::Buffer::Data,node::Buffer::Data(buffer))
info[0]->ToObject()
buffer
buffer
buffer
(<operator>.cast,(uint8_t*) node::Buffer::Data(buffer))
(compressor.CopyInputToRingBuffer, compressor.CopyInputToRingBuffer(node::Buffer::Length(buffer), (uint8_t*) node::Buffer::Data(buffer)))
(METHOD_RETURN,int)
High-level code
Low-level code

Intra-procedural analysis of type checks

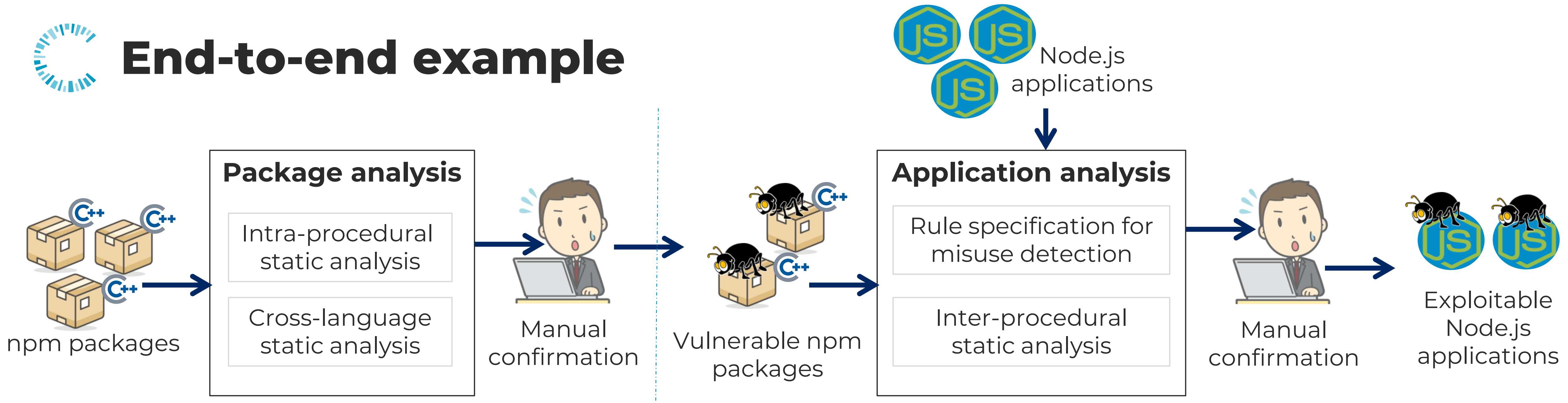


- Most type checks in C/C++,
- Some are done in JavaScript, justifying the need for **cross-language analysis**,
- False positives due to lack of inter-procedural reasoning,
- Most **native extensions are hard to install** (specialized hardware, legacy API version, hard-to-resolve software dependencies).

Manually confirmed vulnerabilities

Package name	#Downloads	Misuse	Remote exploitability	Status
bignum	5,091	M_3	Yes	CVE-2022-25324
ced	1,765	M_3	No	CVE-2021-39131
libxmljs	28,629	M_3	Yes	CVE-2022-21144
sqlite3	452,737	M_3, M_9	Yes	CVE-2022-21227
pg-native	92,436	M_3	Yes	CVE-2022-25852
utf-8-validate	917,251	M_3, M_4	No	Reported
@discordjs/opus	63,007	M_2, M_3, M_9	Yes	CVE-2022-25345
fast-string-search	25	M_3	Yes	CVE-2022-22138
time	1,701	M_3	Yes	Reported
bigint-buffer	159,067	M_3	No	Reported

End-to-end example


Node.js applications
npm packages
Package analysis
Intra-procedural static analysis
Cross-language static analysis
Manual confirmation
Vulnerable npm packages
Application analysis
Rule specification for misuse detection
Inter-procedural static analysis
Manual confirmation
Exploitable Node.js applications

PoC at package level

```
let sqlite3 = require("sqlite3")
let db = new sqlite3.Database("mem");
db.serialize(function() {
   db.run("CREATE TABLE lorem (info TEXT)");
   db.run(
     "INSERT INTO lorem VALUES (?)",
     [{toString: 23}]
   );
});
```

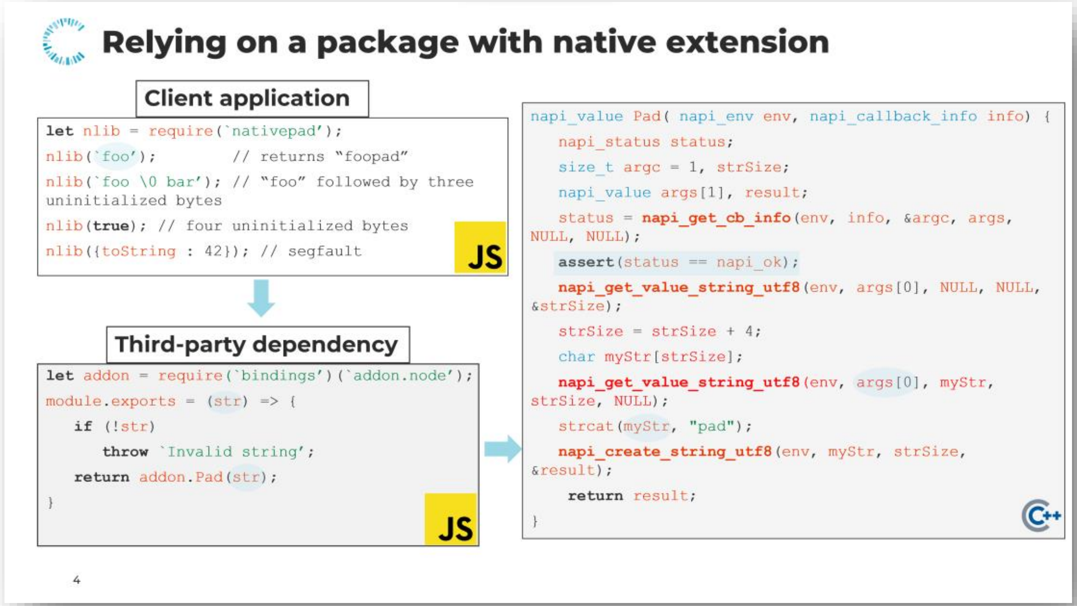
CVE-2022-21227

Vulnerable web application

```
server.post("/", (req , res) => {
   const {img, title, cat, desc, link} = req.body;
   const query = "INSERT INTO ideas (image, title,
cat, desc, link) VALUES (? ,? ,? ,? ,?)";
   const values = [img, title, cat, desc, link];
   db.run(query, values, function (err) {})
});
```

Post request with title = {toString: 23}

Conclusion



Relying on a package with native extension

Client application

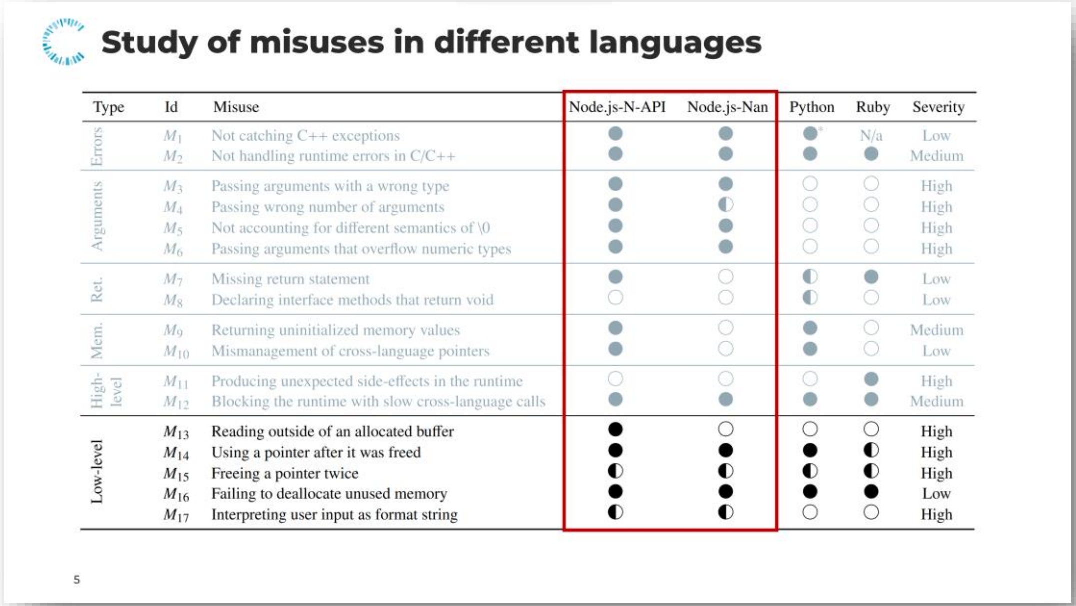
```
let nlib = require(`nativepad');
nlib(`foo');          // returns "foopad"
nlib(`foo \0 bar'); // "foo" followed by three
uninitialized bytes
nlib(true); // four uninitialized bytes
nlib({toString : 42}); // segfault
```

Third-party dependency

```
let addon = require(`bindings')(`addon.node');
module.exports = (str) => {
   if (!str)
      throw `Invalid string';
   return addon.Pad(str);
}
```

```
napi_value Pad( napi_env env, napi_callback_info info) {
   napi_status status;
   size_t argc = 1, strSize;
   napi_value args[1], result;
   status = napi_get_cb_info(env, info, &argc, args,
NULL, NULL);
   assert(status == napi_ok);
   napi_get_value_string_utf8(env, args[0], NULL, NULL,
&strSize);
   strSize = strSize + 4;
   char myStr[strSize];
   napi_get_value_string_utf8(env, args[0], myStr,
strSize, NULL);
   strcat(myStr, "pad");
   napi_create_string_utf8(env, myStr, strSize,
&result);
    return result;
}
```

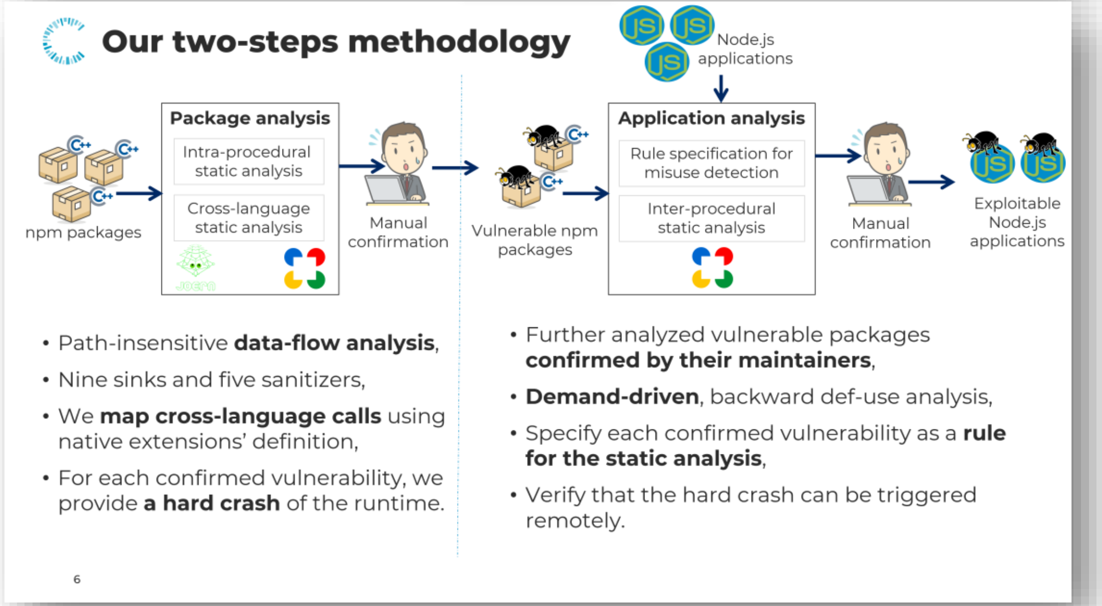
4



Study of misuses in different languages

Type	Id	Misuse	Node.js-N-API	Node.js-Nan	Python	Ruby	Severity
Errors	M_1	Not catching C++ exceptions	●	●	●*	N/a	Low
	M_2	Not handling runtime errors in C/C++	●	●	●	●	Medium
Arguments	M_3	Passing arguments with a wrong type	●	●	○	○	High
	M_4	Passing wrong number of arguments	●	◐	○	○	High
	M_5	Not accounting for different semantics of \0	●	●	○	○	High
	M_6	Passing arguments that overflow numeric types	●	●	○	○	High
Ret.	M_7	Missing return statement	●	○	◐	●	Low
	M_8	Declaring interface methods that return void	○	○	◐	○	Low
Mem.	M_9	Returning uninitialized memory values	●	○	●	○	Medium
	M_{10}	Mismanagement of cross-language pointers	●	○	●	○	Low
High-level	M_{11}	Producing unexpected side-effects in the runtime	○	○	○	●	High
	M_{12}	Blocking the runtime with slow cross-language calls	●	●	●	●	Medium
Low-level	M_{13}	Reading outside of an allocated buffer	●	○	○	○	High
	M_{14}	Using a pointer after it was freed	●	●	●	◐	High
	M_{15}	Freeing a pointer twice	◐	◐	◐	◐	High
	M_{16}	Failing to deallocate unused memory	●	●	●	●	Low
	M_{17}	Interpreting user input as format string	◐	◐	○	○	High

5

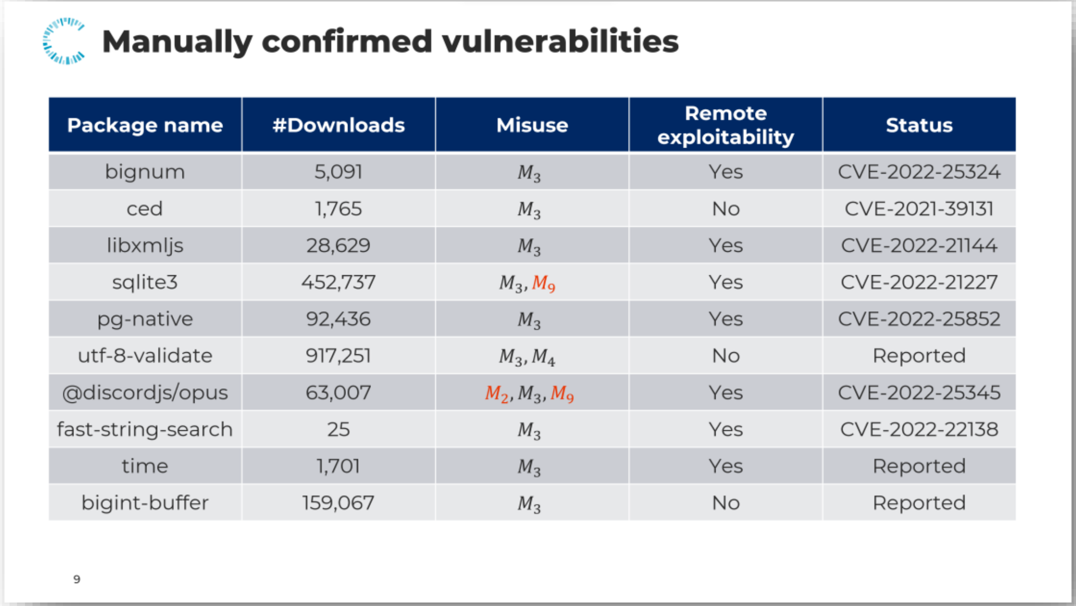

Node.js applications
npm packages
Package analysis
Intra-procedural static analysis
Cross-language static analysis
Manual confirmation
Vulnerable npm packages
Application analysis
Rule specification for misuse detection
Inter-procedural static analysis
Manual confirmation
Exploitable Node.js applications

Our two-steps methodology

- Path-insensitive **data-flow analysis**,
- Nine sinks and five sanitizers,
- We **map cross-language calls** using native extensions' definition,
- For each confirmed vulnerability, we provide **a hard crash** of the runtime.

- Further analyzed vulnerable packages **confirmed by their maintainers**,
- **Demand-driven**, backward def-use analysis,
- Specify each confirmed vulnerability as a **rule for the static analysis**,
- Verify that the hard crash can be triggered remotely.

6



Manually confirmed vulnerabilities

Package name	#Downloads	Misuse	Remote exploitability	Status
bignum	5,091	M_3	Yes	CVE-2022-25324
ced	1,765	M_3	No	CVE-2021-39131
libxmljs	28,629	M_3	Yes	CVE-2022-21144
sqlite3	452,737	M_3, M_9	Yes	CVE-2022-21227
pg-native	92,436	M_3	Yes	CVE-2022-25852
utf-8-validate	917,251	M_3, M_4	No	Reported
@discordjs/opus	63,007	M_2, M_3, M_9	Yes	CVE-2022-25345
fast-string-search	25	M_3	Yes	CVE-2022-22138
time	1,701	M_3	Yes	Reported
bigint-buffer	159,067	M_3	No	Reported

9