black hat®
EUROPE 2023

DECEMBER 4–7

EXCEL LONDON / UK

#BHEU @BlackHatEvents

black hat EUROPE 2023

HODOR: Reducing Attack Surface on Node.js via System Call Limitation

Speakers: Wenya Wang, Xingwei Lin

Contributors: Wenya Wang, Xingwei Lin, Jingyi Wang, Wang Gao, Dawu Gu

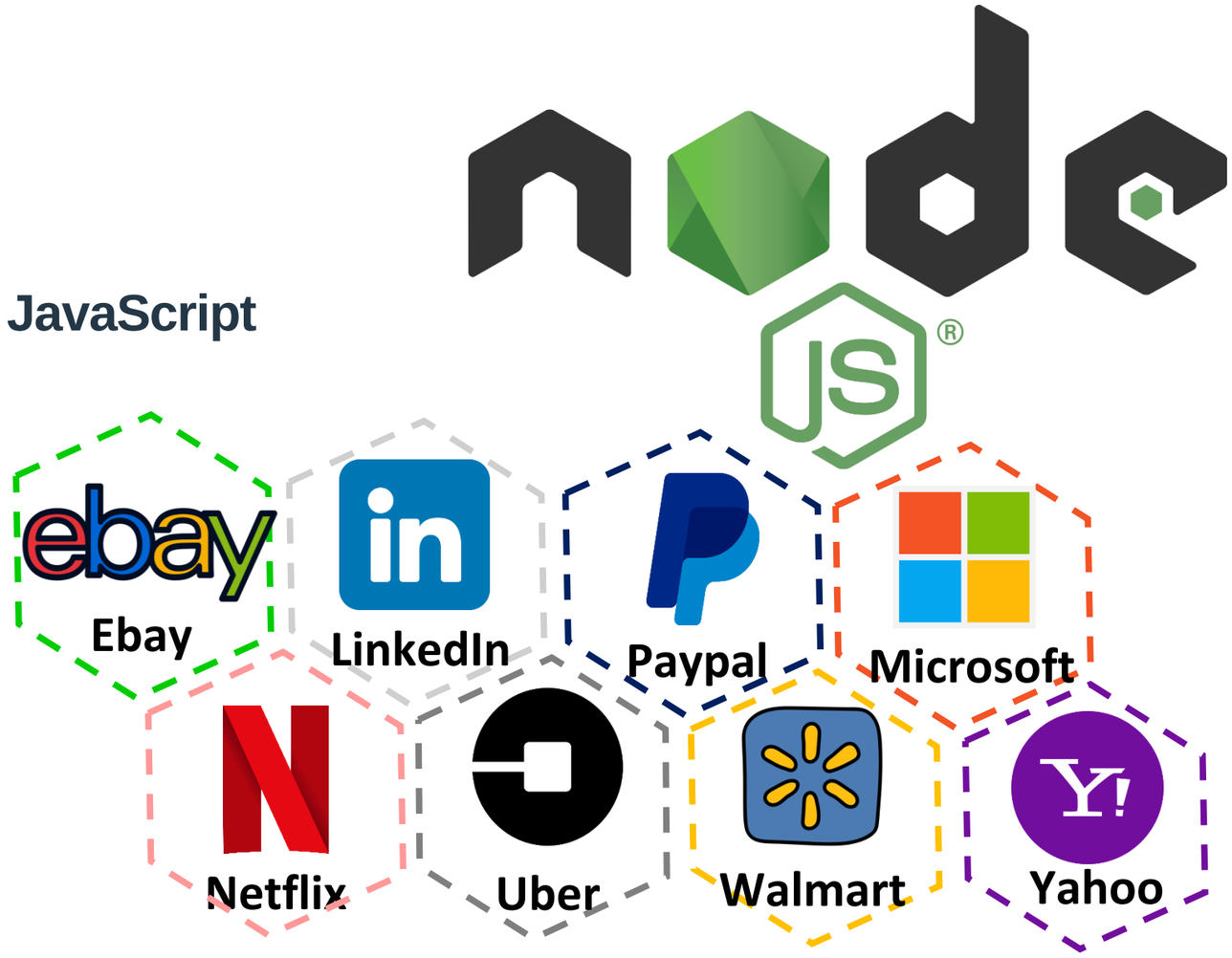
Agenda

- *Introduction*
- Previous work & Remaining challenges
- HODOR: system call level protection system for Node.js applications
- Evaluation
- Conclusion & Takeaways

Node.js

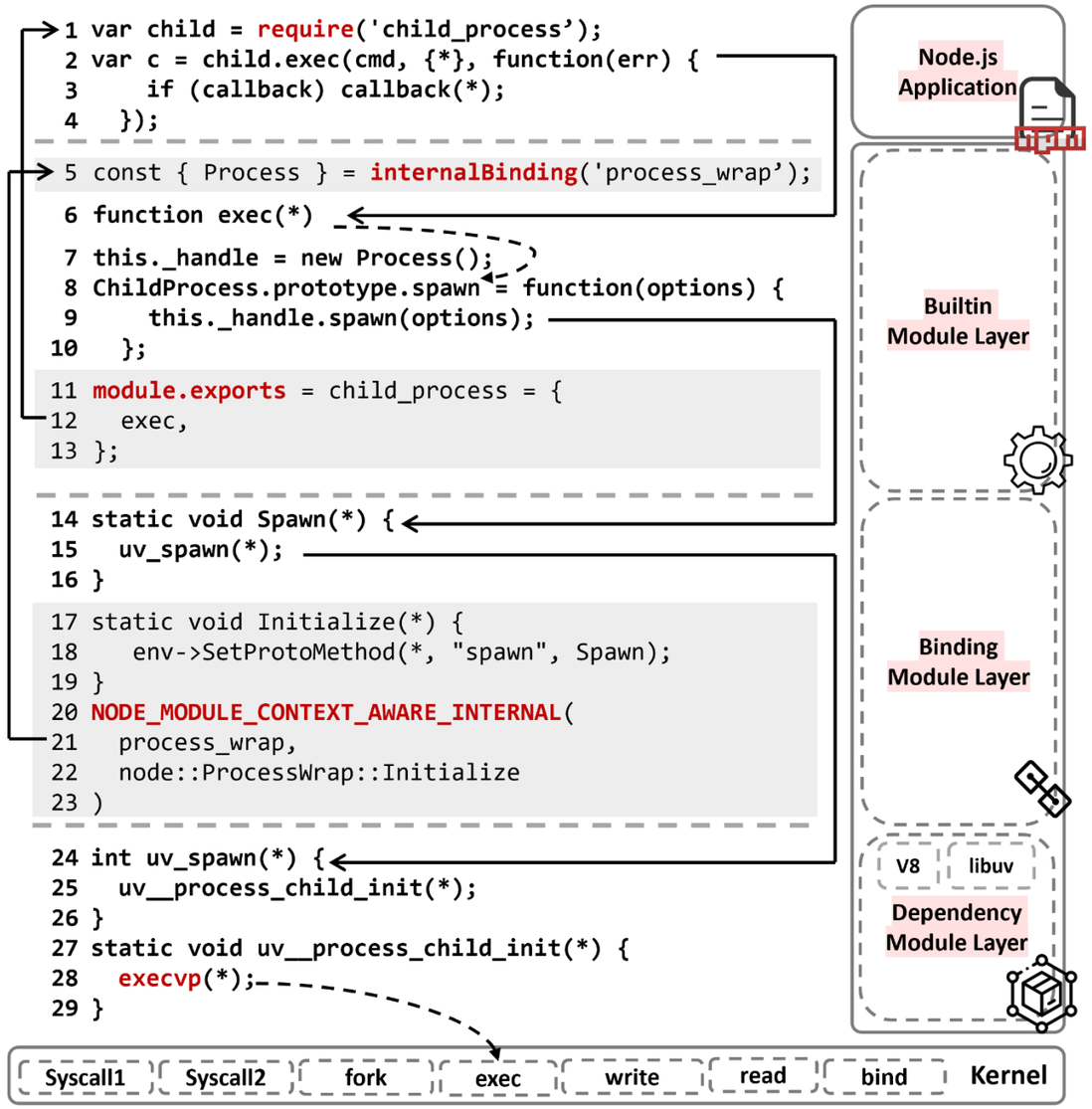
Node.js is an **open-source, cross-platform JavaScript runtime** environment.

- ✓ Asynchronous and Event-Driven
- ✓ Single-Threaded
- ✓ Cross-Platform
- ✓ NPM (Node Package Manager)
- ✓ JavaScript Everywhere



Node.js architecture

- ✓ Node.js Applications (JS)
- ✓ Built-in Module Layer (JS)
- ✓ Binding Module Layer (C++)
- ✓ Dependency Module Layer (C)



Motivation

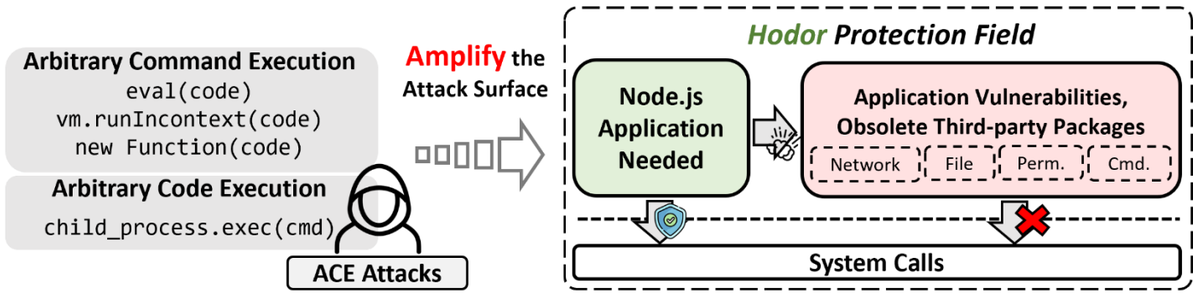
- NPM is a package manager with over 1 million packages →The key to the success of Node.js
- 19.63% of packages in the NPM ecosystem depend on vulnerable packages, such as gadget chain attacks, inject-related attacks, and supply chain attacks. → Most of them may lead to ACE attacks.

- Arbitrary Command/Code Execution: the attackers can perform arbitrary dangerous critical operations

- mail `cat / etc /passwd`
- mail `nc –l –e /bin/bash 8001`
- mail `su root`
- …

```
1  // main.js
2  var growl = require("growl");
3  var message = 'You have mail!';
4  growl(message);
5
6  // ./lib/growl.js
7  exports = module.exports = growl;
8
9  var exec = require('child_process').exec
10 cmd = { pkg: "notify-send" };
11
12 function growl(msg, *) {
13   args = [cmd.pkg];
14   args.push(quote(msg));
15   exec(args.join(''),…);
16 };
```

Growl Application (v1.8.0)



Agenda

- Introduction
- *Previous work & Remaining challenges*
- HODOR: system call level protection system for Node.js applications
- Evaluation
- Conclusion & Takeaways

How to reduce the attack surface of ACE attacks for Node.js applications?

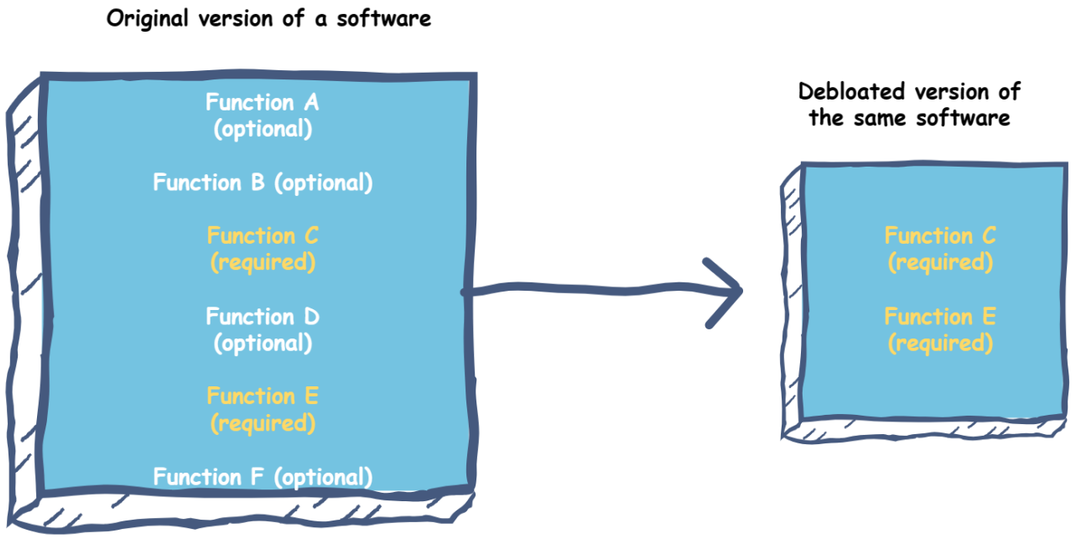
How to reduce the attack surface of ACE attacks for Node.js applications?

Threat Model

- ✓ Consider an attacker with *ACE ability*
- ✓ Not considered: preventing ACE, code vulnerabilities in binding layer/dependency layer, race condition, DOS attack, etc

Existing Works: Software Debloating

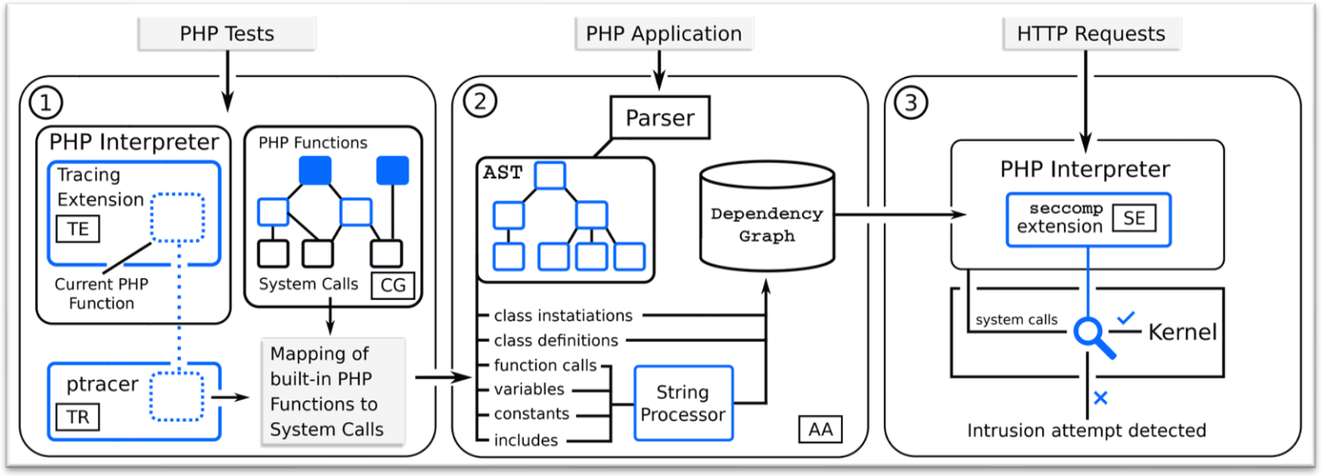
- Use program analysis to cut the ***useless code***
  - ✓ (USENIX Sec'19) RAZOR: A Framework for Post-deployment Software Debloating
  - ✓ (USENIX Sec'19) Less is More: Quantifying the Security Benefits of Debloating Web Applications
  - ✓ (Usenix Sec'20) Slimium: Debloating the Chromium Browser with Feature Subsetting
  - ✓ (RAID'20) Mininode: Reducing the Attack Surface of Node.js Application



What is software debloating? (educative.io)

Existing Works: System Call Limitation

- Restrict the system calls that can be used by the application
  - ✓ (USENIX Sec'20) Temporal System Call Specialization for Attack Surface Reduction
  - ✓ (RAID'20) Confine: Automated System Call Policy Generation for Container Attack Surface Reduction
  - ✓ (RAID'20) sysfilter: Automated System Call Filtering for Commodity Software
  - ✓ (PLDI'20) BlankIt Library Debloating Getting What You Want Instead of Cutting What You Don't
  - ✓ (USENIX Sec'21) Saphire: Sandboxing PHP Applications with Tailored System Call Allowlists



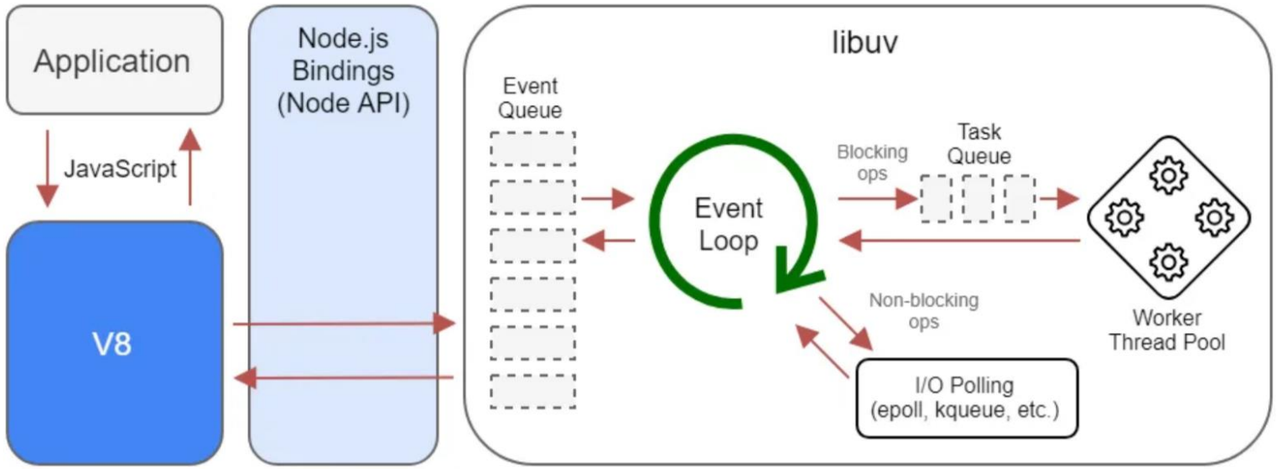
Remaining Challenges

1. Cross-language mapping requirement

✓ JS code layer & C/C++ code layer

2. Integration with Node.js framework

✓ Node.js runs in a single process that creates two kinds of threads.



https://medium.com/preezma/node-js-event-loop-architecture-go-deeper-node-core-c96b4cec7aa4

Problem Formulation

- The number of all system calls provided by the system:

$$S_{base} = |SYSCALL_{system}|$$

- The number of system calls in the whitelist:

$$S_{app} = |SYSCALL_{main\text{-}thread}| \cup |SYSCALL_{thread\text{-}pool}|$$

- The degree of attack surface reduction in the system call level:

$$SR = \frac{S_{app}}{S_{base}}$$

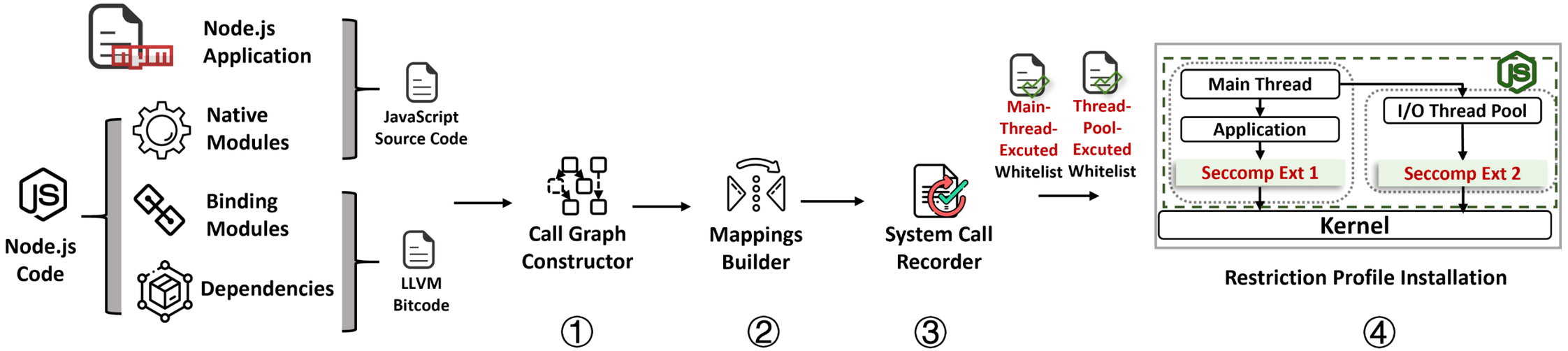
***Goal**: minimize the attack surface in the system call level to prevent malicious critical operations, while not affecting the application's normal execution*

Agenda

- Introduction
- Previous work & Remaining challenges
- *HODOR: system call level protection system for Node.js applications*
- Evaluation
- Conclusion & Takeaways

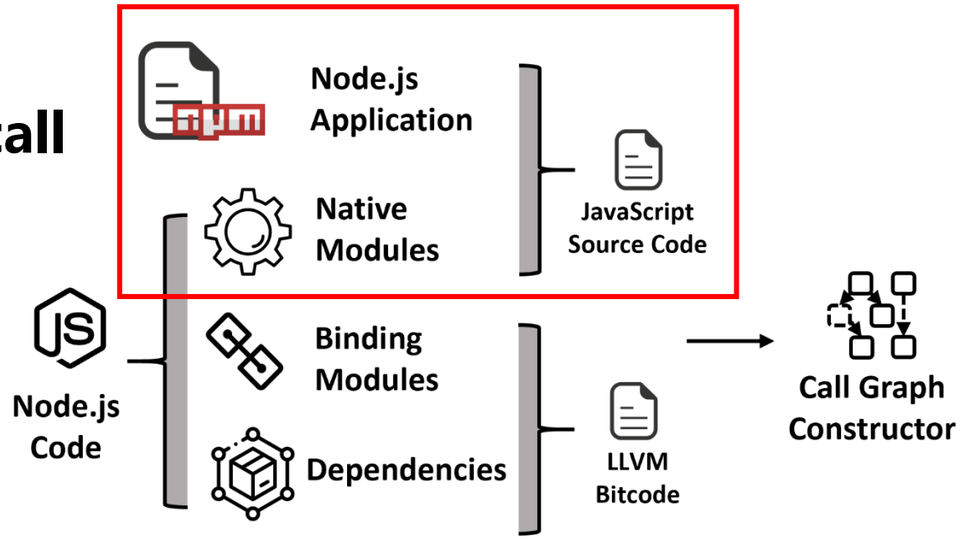
Our approach: Hodor

A lightweight runtime protection system.



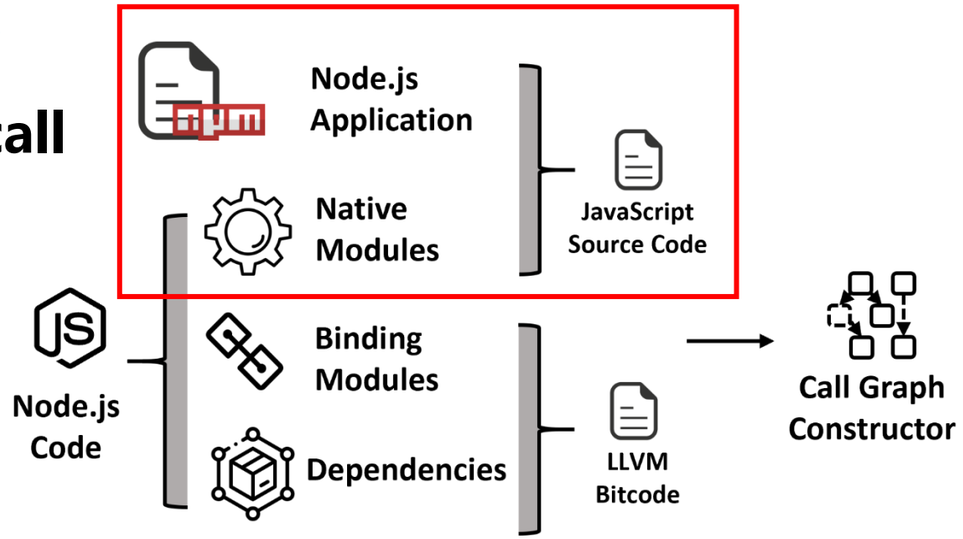
Step 1: Call Graph Construction

- **JS > complement missing nodes/edges/syscall**
  - ✓ **Code features of built-in methods**
    - [1,2,3].map(**x => x * 2**);
    - fs.readFile(filename, **CallbackFunc**);
  - ✓ **Dynamic Analysis Refiner**
    - let sum = new Function('a','b','return a+b');
    - eval("sum()");
  - ✓ **Dynamic Command Execution**
    - child_process.exec("touch new file");



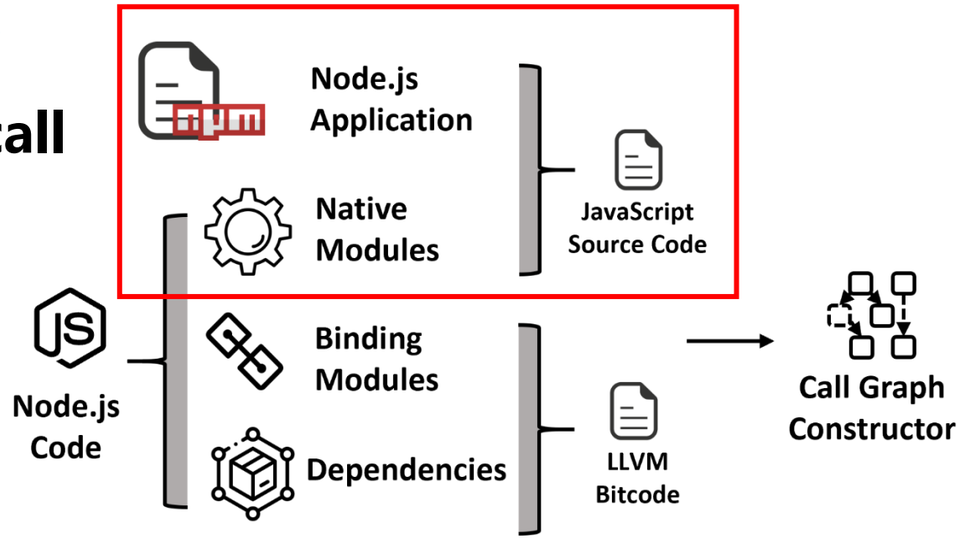
Step 1: Call Graph Construction

- JS > complement missing nodes/edges/syscall
  - ✓ Code features of built-in methods
    - [1,2,3].map(x => x * 2);
    - fs.readFile(filename, CallbackFunc);
  - ✓ Dynamic Analysis Refiner
    - let sum = new Function('a','b','return a+b');
    - eval("sum()");
  - ✓ Dynamic Command Execution
    - child_process.exec("touch new file");



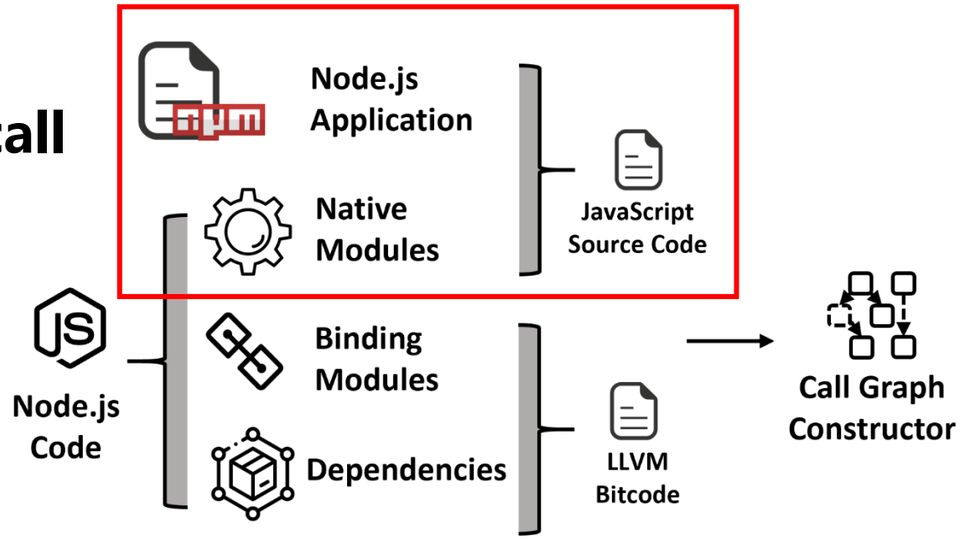
Step 1: Call Graph Construction

- **JS > complement missing nodes/edges/syscall**
  - ✓ **Code features of built-in methods**
    - [1,2,3].map(x => x * 2);
    - fs.readFile(filename, callbackFunc);
  - ✓ **Dynamic Analysis Refiner**
    - let sum = new Function('a','b','return a+b');
    - eval("sum()");
  - ✓ **Dynamic Command Execution**
    - child_process.exec("touch new file"); ⇨ **System Call Required**



Step 1: Call Graph Construction

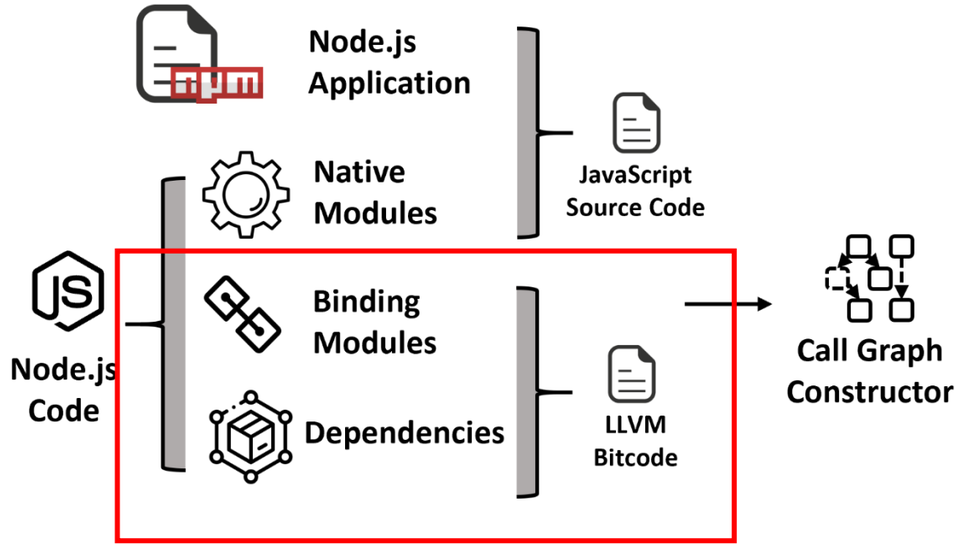
- **JS > complement missing nodes/edges/syscall**
  - ✓ **Code features of built-in methods**
    - [1,2,3].map(x => x * 2);
    - fs.readFile(filename, callbackFunc);
  - ✓ **Dynamic Analysis Refiner**
    - let sum = new Function('a','b','return a+b');
    - eval("sum()");
  - ✓ **Dynamic Command Execution**
    - child_process.exec("touch new file");
- **Implementation**
  - ✓ Reimplement **JAM** and add in proposed optimizations
    - ISSTA'21 Modular call graph construction for security scanning of node.js applications
  - ✓ Combine dynamic call graph tool ***Nodeprof*** and Linux ***strace*** utility



Step 1: Call Graph Construction

- **C/C++ call graph > eliminate non-existing nodes/edges**
  - ✓ *Partial context-aware* analysis for ***switch-case statement*** & function pointer parameter

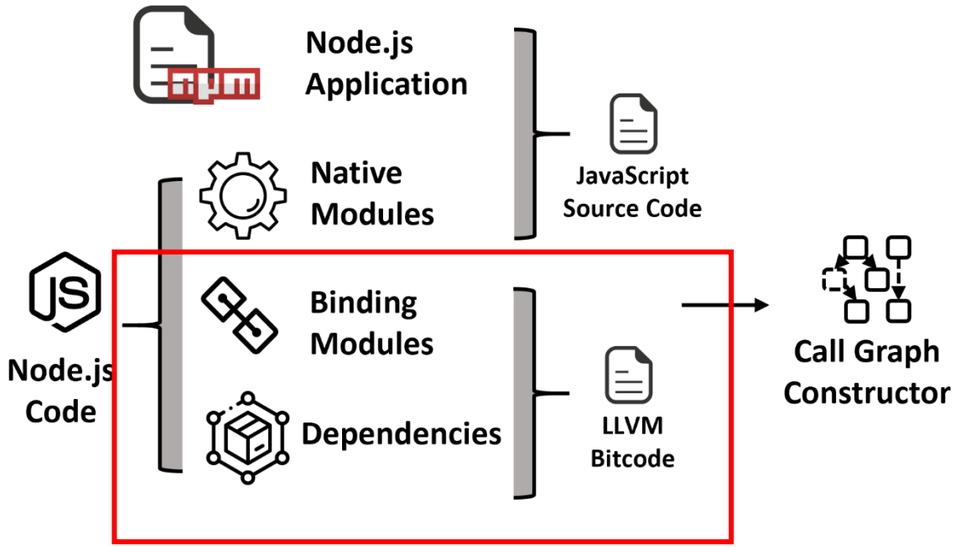
```
1  #define INIT(subtype) req->fs_type = UV_FS_ ## subtype;
2
3  int uv_fs_access(*) {
4    INIT(ACCESS);
5    POST;
6  }
7  int uv_fs_write(*) {
8    INIT(WRITE);
9    POST;
10 }
11
12 #define POST uv__fs_work(&req->work_req);
13
14 static void uv__fs_work(struct uv__work* w) { uv__fs_work.uv_fs_access
15   req = container_of(w, uv_fs_t, work_req);
16   switch (req->fs_type) {
17     X(ACCESS, access(req->path, req->flags));
18     X(WRITE, uv__fs_write_all(req));
19     …
20    default: abort();
21     }
22 }
```



Step 1: Call Graph Construction

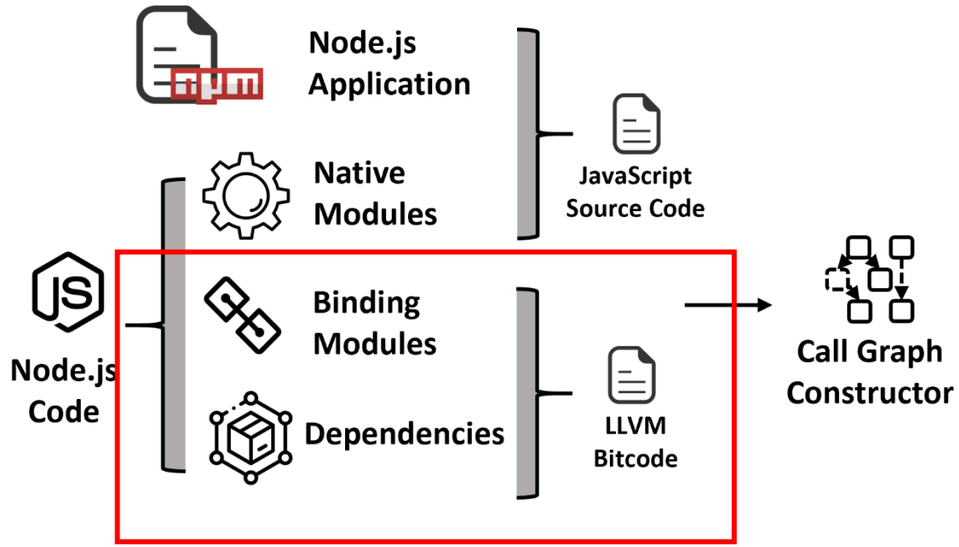
- **C/C++ call graph > eliminate non-existing nodes/edges**
  - ✓ *Partial context-aware* analysis for switch-case statement & ***function pointer parameter***

```
1  static void Read(const FunctionCallbackInfo<Value>& args) {
2    AsyncCall(..., uv_fs_read, ...);
3  }
4  static void Unlink(const FunctionCallbackInfo<Value>& args) {
5    AsyncCall(..., uv_fs_unlink, ...);
6  }
7  static void RMDir(const FunctionCallbackInfo<Value>& args) {
8    AsyncCall(..., uv_fs_rmdir, ...);
9  }
10 FSReqBase* AsyncCall(..., Func fn, Args... fn_args) { AsyncCall.uv_fs_read
     [ ... ]
11   return AsyncDestCall(..., fn, fn_args...);
12 }
13 FSReqBase* AsyncDestCall(..., Func fn, Args... fn_args) { AsyncDestCall.uv_fs_read
     [ ... ]
14   req_wrap->Dispatch(fn, fn_args..., after);
15 }
16 int ReqWrap<T>::Dispatch(LibuvFunction fn, Args... args) { Dispatch.uv_fs_read
     [ ... ]
17   CallLibuvFunction<T, LibuvFunction>::Call( fn, ...);
18 }
19 struct CallLibuvFunction<ReqT, int(*)(uv_loop_t*, ReqT*, Args...)> {
     [ ... ]
20   static int Call(T fn, uv_loop_t* loop, ReqT* req, PassedArgs... args) {
21     return fn(loop, req, args...);                          Call.uv_fs_read
22   }
23 };
```



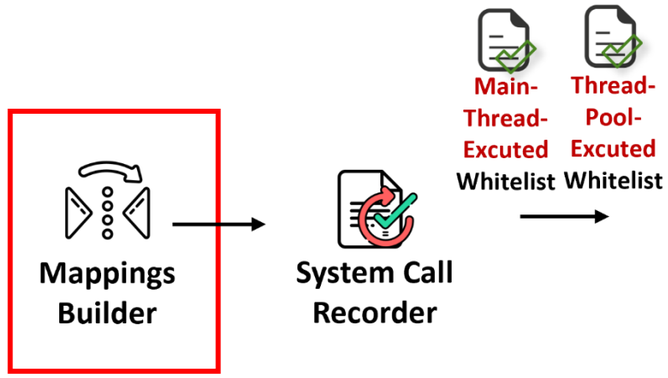
Step 1: Call Graph Construction

- **C/C++ call graph > eliminate non-existing nodes/edges**
  - ✓ *Partial context-aware* analysis for switch-case statement & function pointer parameter
- **Implementation**
  - ✓ clang with wllvm > llvm link > **SVF ++**



Step 2: Mapping Builder

- We build ***call graph traversal*** for call graphs of the Node.js application layer, Binding Module layer, and Dependency layer.
- We build ***LLVM Pass*** for the Built-in Module layer.
- We get mappings of different layers.



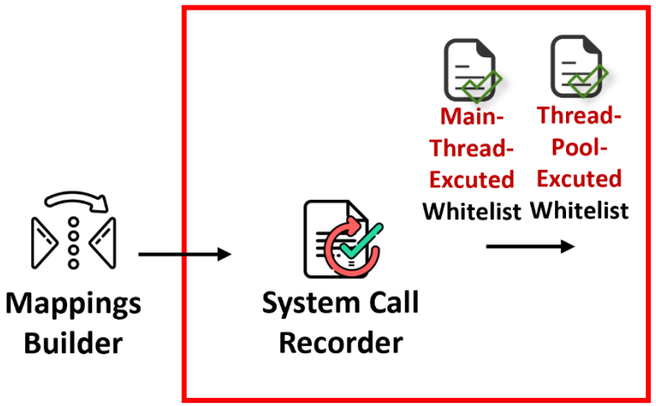
Algorithm 1: Mapping generation

Data: Call graphs of builtin modules *cg.builtin*, call graphs of binding modules and dependencies *cg_bottom*, call graphs of libc *cg.libc*, LLVM IR of binding modules *ir_bind*
Result: Output mapping dict *M*

```
M.builtin ← {} ;                          /* Mappings of builtin modules */
forall cg.Module ∈ cg_builtin do
    forall method ∈ module.exports do
        M.builtin.module.method ← {};
        Callers C invoked by the method by traversing cg.Module;
        forall c ∈ C do
            if c == internalBinding then
                M.builtin.module.method ← (module, method);
M.binding ← {} ;                          /* Mappings of binding modules */
forall ir.Module ∈ ir_bind do
    forall method_bind ∈ ir.Module do
        M.binding.module.method ← func;
M.depend ← {} ;                           /* Mappings of dependencies */
forall func ∈ M.bindings.module.method do
    M_depend.module.method ← {};
    Callers C invoked by the function by traversing cg.bottom;
    forall c ∈ C do
        if c == libc then
            M.depend.module.method ← libc;
forall libc ∈ M.depend.module.method do
    M.depend.module.method ← {};
    Callers C invoked by the function by traversing cg.libc;
    forall c ∈ C do
        if c ∈ syscall then
            M.depend.module.method.libc ← syscall;
return M.builtin, M.binding, M.depend;
```

Step 3: System Call Recorder

- Based on mappings, we calculate the ***system call whitelists*** for the Node.js application.
- We ***divide*** the system call list into the system call list of **main thread** and the system call list of **the thread pool**.



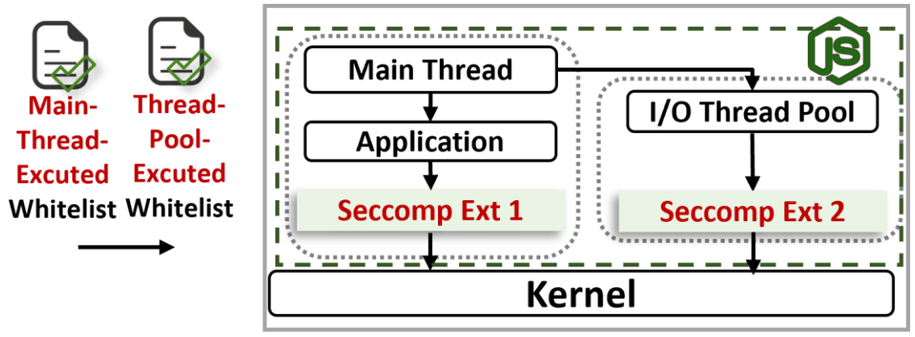
Algorithm 2: Whitelist generation

```
Data: Call graph of Node.js Application cg_app, mapping sets M
Result: Output whitelist W
wl.main ← {};
wl.pool ← {};
Callers C invoked application by traversing cg_app;
forall c ∈ C do
    if c ∈ M.native.c then
        forall b ∈ M.native.c do
            if f ∈ M.bindings.b then
                forall sys ∈ M.depend.f do
                    if b ∈ builtin_threadpool then
                        wl.pool ← sys;
                    else
                        wl.main ← sys;
return W
```

Step 4: Hodor Installation

- **Seccomp Implementation**
  - ✓ For ***thread pool required applications***, we ***first*** install the filter for the thread pool thread and ***then*** install the filter for the main thread to prevent the thread pool thread from inheriting the main thread filter.
  - ✓ For thread pool dis-required applications, we ***only*** load the main thread filter.

- **Read/write Permission Restrictions.**
  - ✓ ***Read*** and ***write system calls*** are widely used by Node.js engine.
  - ✓ ***Chroot*** mechanism and Switch the ownership.



Restriction Profile Installation

Agenda

- Introduction
- Previous work & Remaining challenges
- HODOR: system call level protection system for Node.js applications
- ***Evaluation***
- Conclusion & Takeaways

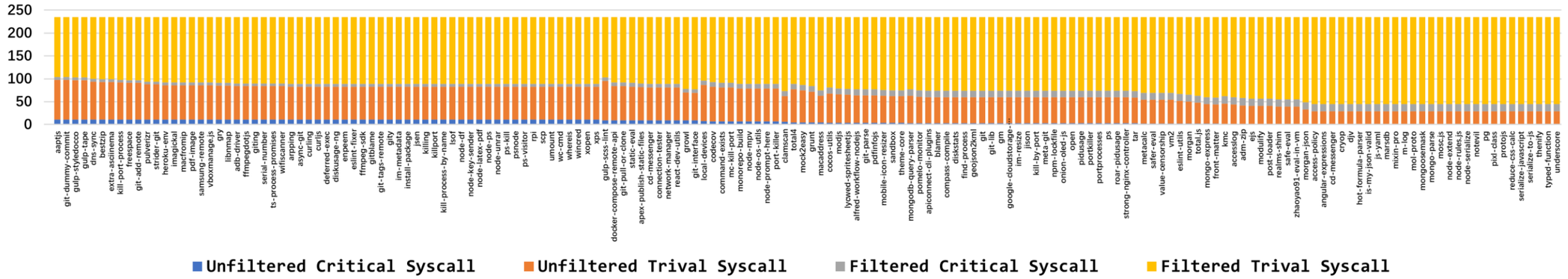
Evaluation

➢ Dataset

- ✓ 169 packages suffered from ACE attacks
- ✓ Three large-scale real-world applications (koa, express and json-server).
- ✓ Node.js core tests and 4 well-known web frameworks (koa, fastify, express, and connect).

➢ Total Result

- ✓ HODOR can **reduce the attack surface** of Node.js applications to *19.42%*.



Evaluation - Call Graph Construction and Resulting Protection

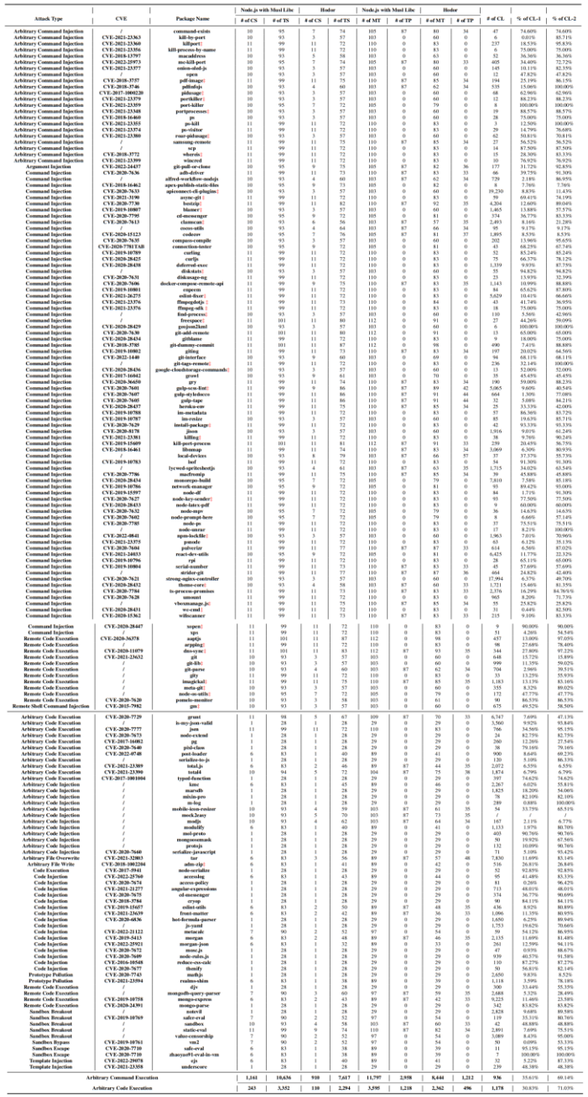
- ✓ The **optimization of JS call graph** construction helps identify hidden required system calls for **23.21%** packages.
- ✓ The **optimization of C/C++ call graph** construction further reduces the system call permissions by **71.02%**.
- ✓ HODOR **reduces the attack surface** for the **main thread** to **19.20%**, for the **thread pool thread** to **7.73%**, while ***not affecting*** the application's normal operation.

Type	# of Package	Node.js w/ Musl		Hodor	
		# of CS	# of TS	# of CS	# of TS
Arb Comm Exec	110	1,161	10,636	910	7,617
Arb Code Exec	58	243	3,352	110	2,294
Total	168	1,404	13,988	1,020	9,911

Type	# of Package	# of MT	# of TP	# of MT	# of TP
Arb Comm Exec	110	11,797	2,958	8,444	1,212
Arb Code Exec	58	3,595	1,218	2,362	496
Total	168	15,392	4,176	10,806	1,708

CS: Critical syscalls invocation; **TS**: Trivial syscalls invocation; **MT**: Main Thread system calls invocation; **TP**: Thread Pool system calls invocation;

Table 5: HODOR granularity of packages at system call level and thread level (RQ1).



Evaluation - Exploit Mitigation

- ✓ We construct **different advanced attack payloads** to simulate various dangerous behaviors of attackers, where a variety of critical system calls can be invoked.
- ✓ HODOR could effectively mitigate the execution of **73.59%** exploits.

Table 5: Vulnerability payloads.

Critical Syscall		JavaScript	C*
Cmd Execution	exec	child_process.exec	execve
	fork	child_process.fork	fork
Permission	setgit	process.setuid	setuid
	setuid	process.setgid	setgid
Network	bind	net.connect	bind
	connect	dgram.createSocket.bind	connect
	listen	server.listen	listen

*: We compile the C code into binary and execute the binary as a shell.

Table 7: Exploit execution for packages with arbitrary code execution attacks.

Package Name	Initial Attack	HODOR								MIR							
		Initial Attack	Cmd: Exec	Cmd: Fork	Permission: Setgid	Permission: Setuid	Net: Connect	Net: Listen	Net: Bind	Initial Attack	Cmd: Exec	Cmd: Fork	Permission: Setgid	Permission: Setuid	Net: Connect	Net: Listen	Net: Bind
accesslog	Print sSS	✘	✓	✓	✓	✓	✓	✓	✓				/				
access-policy	Print 123	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
adm-zip	Path traversal	✓	✓	✓	✓	✓	✓	✓	✓				/				
angular-expressions	Write file angular-expressions-success	✓	✓	✓	✓	✓	✓	✓	✓				/				
cd-messenger	Print JHU	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
crypp	Print defconrussia	✘	✓	✓	✓	✓	✓	✓	✓				/				
djv	touch HACKED	✓	✓	✓	✓	✓	✓	✓	✓				/				
ejs	Write file ejs-success	✓	✓	✓	✓	✓	✓	✓	✓				/				
eslint-utils	Write file eslint-utils-success	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
front-matter	Print 1	✘	✓	✓	✓	✓	✓	✓	✓				/				
grunt	Returns Date.now	✓	✘	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
hot-formula-parser	Write file test	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
is-my-json-valid	Execute cat /etc/passwd	✓	✓	✓	✓	✓	✓	✓	✓				/				
jscn	Write file malicious	✓	✘	✘	✘	✘	✘	✘	✘				/				
js-yaml	Returns Date.now	✓	✓	✓	✓	✓	✓	✓	✓				/				
kmc	Write file kmc-success	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
marsdb	Write file marsdb-success	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
metacalc	Print process	✘	✓	✓	✓	✓	✓	✓	✓				/				
mixin-pro	Print hacked	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
m-log	Print injected	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
mock3cazy	Write mock3cazy-success	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
modjs	Write modjs-success.txt	✘	✘	✘	✓	✓	✘	✘	✘	✓	✓	✓	✓	✓	✓	✓	✓
modulify	Print hacked	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
mol-proto	Write file mol-proto-success	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
mongodb-query-parser	touch test-file	✓	✓	✓	✓	✓	✘	✘	✘				/				
mongo-express	exec calculator	✓	✓	✓	✓	✓	✓	✓	✓				/				
mongoosemask	Print "my evil code was run"	✘	✓	✓	✓	✓	✓	✓	✓				/				
mongo-parse	Write file hacked	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
morgan	Write file morgui-success	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
morgan-json	Print GLOBAL CTF HIT	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
mosc.js	Write file Song	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
node-extend	Print 123	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
node-extend	Print 123	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
node-rules.js	Print 123	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
node-serialize	Execute ls	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
notevil	Print pwned	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
pg	Print process.env	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
pixl-class	Print 123	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
post-loader	Print ree	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
protojs	Write file protojs-success	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
realms-shim	Messed with Object.toString	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
reduce-css-calc	Read /etc/passwd	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
safe-eval	Return process	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
safer-eval	Print id	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sandbox	Print process.pid	✘	✘	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
serialize-javascript	Print 1	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
serialize-to-js	Execute ls	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
static-eval	Print hacked	✓	✘	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
tar	Overwrite file	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
thenify	Write file Song	✓	✓	✓	✓	✓	✓	✓	✓				/				
total.js	Touch HACKED	✘	✓	✓	✓	✓	✘	✓	✘				/				
total4	Touch HACKED	✓	✘	✘	✓	✓	✘	✘	✘				/				
typed-function	Execute whoami	✓	✓	✓	✓	✓	✓	✓	✓				/				
underscore	touch HELLO	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
value-censorship	Access the Function constructor	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
vm2	return process.env	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
zhaoyao91-eval-in-vm	return process.env	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
mobile-icon-resizer	Print hacked	✘	✘	✘	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

✘: Exploits are executed; ✓: Exploits are blocked;

Table 6: Exploit execution for packages with arbitrary command execution attacks.

Package Name	Initial Attack	HODOR							
		Initial Attack	Cmd: Exec	Cmd: Fork	Permission: Setgid	Permission: Setuid	Net: Connect	Net: Listen	Net: Bind
command-exists	Write command-exists	✓	✘	✘	✓	✓	✓	✓	✓
kill-by-port	Write success	✓	✘	✘	✓	✓	✓	✓	✓
killport	Write success	✓	✘	✘	✓	✓	✓	✓	✓
kill-process-by-name	Write success	✓	✘	✘	✓	✓	✓	✓	✓
macaddress	Write /tmp/poof	✓	✘	✘	✓	✓	✓	✓	✓
mc-kill-port	Write newFile.txt	✓	✘	✘	✓	✓	✘	✓	✓
onion-olod-js	Write success	✓	✘	✘	✓	✓	✓	✓	✓
open	Write /tmp/tada	✓	✘	✘	✓	✓	✓	✓	✓
pdf-image	Write /tmp/hacked	✓	✘	✘	✓	✓	✓	✓	✓
pdfinfo.js	Write a	✓	✘	✘	✓	✓	✓	✓	✓
pidusage	Execute /usr/local/bin/python	✓	✘	✘	✓	✓	✓	✓	✓
portkiller	Write success	✓	✘	✘	✓	✓	✓	✓	✓
port-killer	Write success	✓	✘	✘	✓	✓	✓	✓	✓
portprocesses	Write success	✓	✘	✘	✓	✓	✓	✓	✓
ps	Write success.txt	✓	✘	✘	✓	✓	✓	✓	✓
ps-kill	Write success	✓	✘	✘	✓	✓	✓	✓	✓
ps-visitor	Write success	✓	✘	✘	✓	✓	✓	✓	✓
roar-pidusage	Write success	✓	✘	✘	✓	✓	✓	✓	✓
samsung-remote	Write /tmp/malicious;	✓	✘	✘	✓	✘	✓	✓	✓
scp	nc localhost 4444;	✓	✘	✘	✓	✓	✓	✓	✓
whereis	Write /tmp/tada	✓	✘	✘	✓	✓	✓	✓	✓
wincred	Write success	✓	✘	✘	✓	✓	✓	✓	✓
git-pull-or-clone	Write /tmp/pwn3	✓	✘	✘	✓	✓	✓	✓	✓
adb-driver	Write a	✓	✘	✘	✓	✓	✓	✓	✓
alfred-workflow-nodejs	Write hacked	✓	✘	✘	✓	✓	✓	✓	✓
apex-publish-static-files	Write apex-publish-static-files	✓	✘	✘	✓	✓	✘	✓	✓
apiconnect-cli-plugins	Write Song	✓	✘	✘	✓	✓	✓	✓	✓
async-git	Write HACKED #	✓	✘	✘	✓	✘	✘	✘	✘
bestzip	Write bestzip	✓	✘	✘	✓	✓	✓	✓	✓
blamer	Write vulnerable	✓	✘	✘	✓	✓	✓	✓	✓
cd-messenger	Write JHU	✓	✘	✘	✓	✓	✓	✓	✓
clamscan	Write create.txt	✓	✘	✘	✓	✓	✘	✓	✓
cocos-utils	Write hacked	✓	✘	✘	✓	✓	✓	✓	✓
codecov	Write codecov	✓	✘	✘	✓	✓	✓	✓	✓
compass-compile	Write JHU	✓	✘	✘	✓	✓	✓	✓	✓
connection-tester	Write 111	✓	✘	✘	✓	✓	✘	✓	✓
curling	Write JHU	✓	✘	✘	✓	✓	✓	✓	✓
curljs	Write JHU	✓	✘	✘	✓	✓	✓	✓	✓
deferred-exec	Write JHU	✓	✘	✘	✓	✓	✓	✓	✓
diskstats	Write HACKED	✓	✘	✘	✓	✓	✓	✓	✓
diskusage-ng	Write Song	✓	✘	✘	✓	✓	✓	✓	✓
docker-compose-remote-api	Write vulnerable.txt	✓	✘	✘	✓	✓	✘	✘	✘
enpeem	Write create.txt	✓	✘	✘	✓	✓	✓	✓	✓
eslint-fixer	Write eslint-fixer	✓	✘	✘	✘	✘	✓	✓	✓
ffmpegdotjs	Write success	✓	✘	✘	✓	✓	✓	✓	✓
ffmpeg-sdk	Write success	✓	✘	✘	✓	✓	✓	✓	✓
find-process	Write /tmp/semicolon_file	✓	✘	✘	✓	✓	✓	✓	✓
freespace	Write /tmp/semicolon_file	✓	✘	✘	✓	✓	✘	✓	✓
geojson2kml	Write JHU	✓	✘	✘	✓	✓	✓	✓	✓
git-add-remote	Write Song	✓	✘	✘	✓	✓	✘	✓	✓
gitblame	Write JHU	✓	✘	✘	✓	✓	✓	✓	✓
git-dummy-commit	Write git-dummy-commit	✓	✘	✘	✓	✓	✓	✓	✓
giting	Write create.txt	✓	✘	✘	✓	✓	✓	✓	✓
git-interface	Write /tmp/pwned	✓	✘	✘	✓	✓	✓	✓	✓
git-tags-remote	Write /tmp/command-injection.test	✓	✘	✘	✓	✓	✓	✓	✓
google-cloudstorage-commands	Write JHU	✓	✘	✘	✓	✓	✓	✓	✓
growl	Write aaaa	✓	✘	✘	✓	✓	✓	✓	✓
gry	Write HACKED	✓	✘	✘	✓	✓	✓	✓	✓
gulp-scss-lint	Write create.txt	✓	✘	✘	✓	✓	✘	✘	✘
gulp-styledocco	Write Vulnerable	✓	✘	✘	✓	✓	✘	✘	✘
gulp-tape	Write JHU.txt	✓	✘	✘	✓	✓	✘	✘	✘
heroku-env	Write JHU	✓	✘	✘	✘	✘	✓	✓	✓
im-metadata	Write im-metadata	✓	✘	✘	✘	✘	✘	✘	✘
im-resize	Write create.txt	✓	✘	✘	✓	✓	✓	✓	✓
install-package	Write Song	✓	✘	✘	✓	✓	✓	✓	✓
jison	Write pwned	✓	✘	✘	✓	✓	✓	✓	✓
killing	Write success	✓	✘	✘	✓	✓	✓	✓	✓
kill-port-process	Write kill-port-process	✓	✘	✘	✓	✓	✘	✓	✓
libnmap	Write success.txt	✓	✘	✘	✓	✓	✓	✓	✓
local-devices	Makrk directory attacker	✓	✘	✘	✓	✓	✓	✓	✓
lsof	Write create.txt	✓	✘	✘	✘	✘	✓	✓	✓
lycwed-spritesheetjs	Write 111233 #	✓	✘	✘	✓	✓	✓	✓	✓
macfromip	Write JHU2	✓	✘	✘	✓	✓	✓	✓	✓
monorepo-build	Write JHU	✓	✘	✘	✓	✓	✘	✓	✓
network-manager	Write create.txt	✓	✘	✘	✓	✓	✘	✓	✓
node-dff	Write HACKED	✓	✘	✘	✓	✓	✓	✓	✓
node-key-sender	Write Song	✓	✘	✘	✓	✓	✓	✓	✓
node-latex-pdf	Write JHU	✓	✘	✘	✘	✘	✘	✘	✘
node-ospr	Write JHU	✓	✘	✘	✓	✓	✓	✓	✓
node-prompt-here	Write create.txt	✓	✘	✘	✓	✓	✘	✓	✓
node-ps	Write JHU	✓	✘	✘	✓	✓	✓	✓	✓
node-unrar	Write node-unrar	✓	✘	✘	✓	✓	✓	✓	✓
npm-lockfile	Write ree	✓	✘	✘	✓	✓	✓	✓	✓
pmode	Write success	✓	✘	✘	✓	✓	✓	✓	✓
pulverizr	Write Song	✓	✘	✘	✓	✓	✓	✓	✓
react-dev-utils	Write react-dev-utils	✓	✘	✘	✓	✓	✘	✓	✓
rpi	Write vulnerable.txt	✓	✘	✘	✓	✓	✓	✓	✓
serial-number	Write create.txt	✓	✘	✘	✘	✘	✓	✓	✓
strider-git	Write HACKED;	✓	✘	✘	✘	✘	✘	✘	✘
strong-nginx-controller	Write Song	✓	✘	✘	✓	✓	✓	✓	✓
theme-core	Write JHU	✓	✘	✘	✓	✓	✓	✓	✓
ts-process-promises	Write JHU	✓	✘	✘	✓	✓	✓	✓	✓
umount	Write Song	✓	✘	✘	✘	✘	✓	✓	✓
vboxmanage.js	Write HACKED	✓	✘	✘	✘	✘	✓	✓	✓
wc-cmd	Write JHU	✓	✘	✘	✓	✓	✓	✓	✓
wlbscanner	Write /tmp/evpkit.txt	✓	✘	✘	✓	✓	✓	✓	✓
xopen	Write JHU	✓	✘	✘	✓	✓	✓	✓	✓
xps	Write HACKED	✓	✘	✘	✓	✓	✓	✓	✓
aapt.js	Write aapt.js	✓	✘	✘	✓	✘	✓	✘	✓
arpping	Write HACKED	✓	✘	✘	✓	✓	✓	✓	✓
dns-sync	Write pwned	✓	✘	✘	✓	✘	✓	✓	✓
git	date	✘	✘	✘	✓	✓	✓	✓	✓
git-lib	Write HACKED;	✓	✘	✘	✓	✓	✓	✓	✓
git-parse	Write HACKED	✓	✘	✘	✓	✓	✓	✓	✓
gity	Write HACKED	✓	✘	✘	✓	✓	✓	✓	✓
imagickal	Write HACKED	✓	✘	✘	✓	✓	✓	✓	✓
meta-git	Write HACKED	✓	✘	✘	✓	✓	✓	✓	✓
node-os-utils	Write DUMMY_FILE	✓	✘	✘	✓	✓	✘	✓	✓
pomelo-monitor	Write Song	✓	✘	✘	✓	✓	✓	✓	✓
gm	Write gm	✓	✘	✘	✓	✓	✓	✓	✓

✘: Exploits are executed; ✓: Exploits are blocked;

Evaluation - Comparison with Other Techniques

- ✓ HODOR can defend against **a wider spectrum of attacks** (additionally covering arbitrary command execution) **with less runtime overhead**.

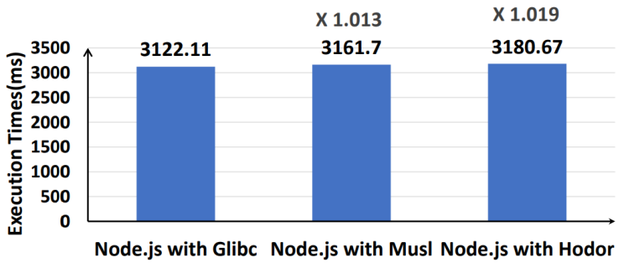
Table 7: Exploit execution for packages with arbitrary code execution attacks.

Package Name	Initial Attack	HODOR								MIR							
		Initial Attack	Cmd	Permission			Net			Initial Attack	Cmd	Permission			Net		
			Exec	Fork	Setgid	Setuid	Connect	Listen	Bind		Exec	Fork	Setgid	Setuid	Connect	Listen	Bind
accesslog	Print xSS	✘	✔	✔	✔	✔	✔	✔	✔					/			
access-policy	Print 123	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
adm-zip	Path traversal	✔	✔	✔	✔	✔	✔	✔	✔				/				
angular-expressions	Write file angular-expressions-success	✔	✔	✔	✔	✔	✔	✔	✔				/				
cd-messenger	Print JHU	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
cryop	Print defconrussia	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
djv	touch HACKED	✔	✔	✔	✔	✔	✔	✔	✔					/			
ejs	Write file ejs-success	✔	✔	✔	✔	✔	✔	✔	✔					/			
eslint-utils	Write file eslint-utils-success	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
front-matter	Print 1	✘	✔	✔	✔	✔	✔	✔	✔					/			
grunt	Returns Date.now	✔	✘	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
hot-formula-parser	Write file test	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
is-my-json-valid	Execute cat /etc/passwd	✔	✔	✔	✔	✔	✔	✔	✔					/			
jsen	Write file malicious	✔	✘	✘	✘	✘	✘	✘	✘					/			
js-yaml	Returns Date.now	✔	✔	✔	✔	✔	✔	✔	✔					/			
kmc	Write file kmc-success	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
marsdb	Write file marsdb-success	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
metacalc	Print process	✘	✔	✔	✔	✔	✔	✔	✔					/			
mixin-pro	Print hacked	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
m-log	Print injected	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
mock2easy	Write mock2easy-success	✘	✔	✔	✔	✔	✘	✘	✘	✔	✔	✔	✔	✔	✔	✔	✔
modjs	Write modjs-success.txt	✘	✘	✘	✔	✔	✘	✘	✘	✔	✔	✔	✔	✔	✔	✔	✔
modulify	Print hacked	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
mol-proto	Write file mol-proto-success	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
mongodb-query-parser	touch test-file	✔	✔	✔	✔	✔	✘	✘	✘					/			
mongo-express	exec calculator	✔	✔	✔	✔	✔	✔	✔	✔					/			
mongoosemask	Print "my evil code was run"	✘	✔	✔	✔	✔	✔	✔	✔					/			
mongo-parse	Write file hacked	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
morgan	Write file mongui-success	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
morgan-json	Print GLOBAL CTF HIT	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
mosc.js	Write file Song	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
node-extend	Print 123	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
node-extend	Print 123	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
node-rules.js	Print 123	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
node-serialize	Execute ls	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
notevil	Print pwned	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
pg	Print process.env	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
pixl-class	Print 123	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
post-loader	Print rce	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
protojs	Write file protojs-success	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
realms-shim	Messed with Object.toString	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
reduce-css-calc	Read /etc/passwd	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
safe-eval	Return proces	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
safer-eval	Print id	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
sandbox	Print process.pid	✘	✘	✘	✔	✔	✔	✔	✔	✔	✘	✘	✔	✔	✔	✔	✔
serialize-javascript	Print 1	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
serialize-to-js	Execute ls	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
static-eval	Print hacked	✔	✘	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
tar	Overwrite file	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
thenify	Write file Song	✔	✔	✔	✔	✔	✔	✔	✔					/			
total.js	Touch HACKED	✘	✔	✔	✔	✔	✘	✘	✘					/			
total4	Touch HACKED	✔	✘	✘	✔	✔	✘	✘	✘					/			
typed-function	Execute whoami	✔	✔	✔	✔	✔	✔	✔	✔					/			
underscore	touch HELLO	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
value-censorship	Access the Function constructor	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
vm2	return process.env	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
zhaoyao91-eval-in-vm	return process.env	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
mobile-icon-resizer	Print hacked	✘	✘	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

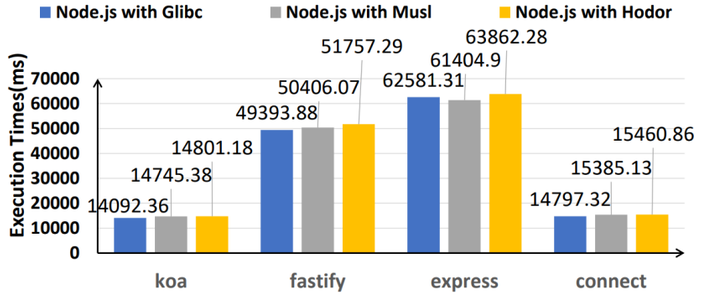
✘: Exploits are executed; ✔: Exploits are blocked;

Evaluation - Runtime Overhead

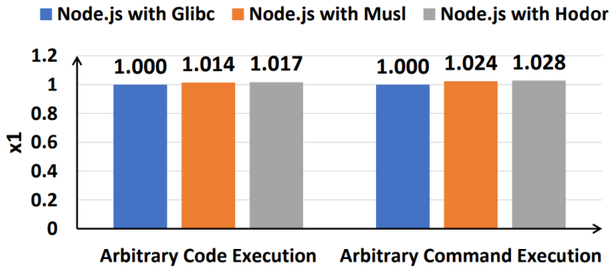
- ✓ The runtime overhead of HODOR is **0.61%** for **Node.js core tests**, **2.80%** for the **web framework**, and **0.39%** for all **168 packages**.



(a) Node.js Core Tests



(b) Web Framework



(c) Node.js Packages

Figure 8: Runtime overhead of Node.js core tests, web framework and applications under the protection of HODOR (RQ4).

Agenda

- Introduction
- Previous work & Remaining challenges
- HODOR: system call level protection system for Node.js applications
- Evaluation
- ***Conclusion & Takeaways***

Conclusion & Takeaways

1. Attendees will learn a new call graph building methods for JavaScript code and C/C++ code.

2. Attendees will gain knowledge of a novel protection mechanism for Node.js applications, focusing on thread-level and system call-level security.

3. Attendees will develop an understanding of the hazards associated with vulnerabilities in the Node.js application ecosystem, with a particular emphasis on system call-level vulnerabilities.

Thanks & Questions?

Wenya Wang, Xingwei Lin @xwlin_roy