Enable **seamless sandboxing** of native components your web application most likely depends upon

NatiSand: Native Code Sandboxing for JavaScript Runtimes

Marco Abbadini, Dario Facchinetti, Gianluca Oldani, **Matthew Rossi**, Stefano Paraboschi

BACKGROUND

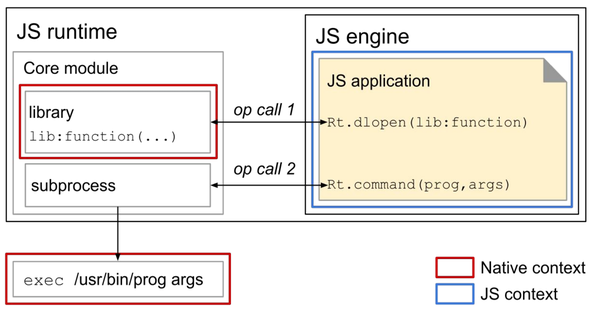
Originally meant to run in the browser to provide advanced dynamic interactions to web pages, JavaScript is now also being used on the server-side

To run JavaScript on the server-side, we use **JavaScript runtimes**

Node.js, Deno → V8 engine

Bun → JavaScriptCore

Despite their differences, here is how we can depict the architecture of JavaScript runtimes



They complement the standard JavaScript features available in browsers with additional functions necessary for the development of applications on the server-side

PROBLEM STATEMENT

Native code is executed outside of the isolated context prepared by the engine ➡ runs with the permissions of the whole app

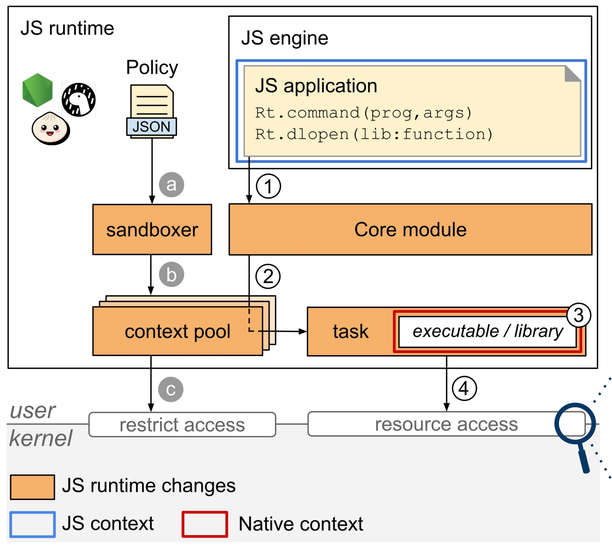
Native components are often written with **memory unsafe languages** and may come with **different** security **assumptions** ➡ one vulnerability away from a series of security problems

SECURITY RISKS

- **Violation** of the **integrity** and **confidentiality** of the application code and data
- **Escalation of privileges** by attempting confused deputy attacks on privileged system services
- Malicious network channels by opening **reverse shells**

SOLUTION

Improve the security of JS applications with the introduction of **ad hoc security contexts** for the execution of **native code**



RESULTS

- **Mitigation** of **CVEs** targeting executables and libraries widely used in web applications
- The cost of activating the sandbox is amortized with the increase in the test duration
- **Outperform state-of-the-art solutions** in the execution of common Linux utilities and image processing microservices **reduce latency by 5 to 10 ms**
- **Significantly outperform WebAssembly** in the execution of popular libraries (e.g., libpng, sqlite3), and microservices using them **reduce latency by 100 to 400 ms**

SUMMARY

- Execution of native code in JS runtimes represents a clear **security risk**
- Our proposal enables **native code sandboxing** for JavaScript Runtimes by providing fine-grained access control of system resources **without** requiring **changes to the application code**
- Experiments showcase the **mitigation** of exploits at the cost of **limited overhead**

WARNING: SCARY TECHNICAL DETAILS UP AHEAD

COMPONENTS FOR RESOURCE PROTECTION

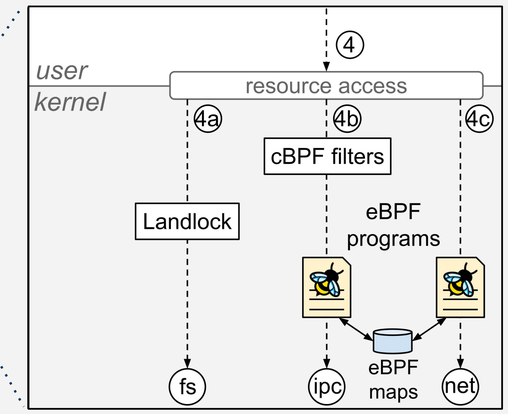
Seccomp is a mechanism provided by the Linux kernel to restrict the system calls available

It has only access to the values of the arguments passed to the system calls (e.g., configuration flags), and pointers cannot be dereferenced

eBPF permits to hook programs anywhere in the kernel to safely modify its functionality at runtime

Use cases: efficient networking, observability, tracing and security

Landlock is a Linux Security Module that enables unprivileged applications to restrict their filesystem permissions



Context lifecycle
uprobe/attach_policy
tp_btf/sched_process_fork
tp_btf/sched_process_exit

Access control	
IPC	Network
lsm/socket_bind	fentry/fifo_open
lsm/socket_create	lsm/socket_bind
lsm/socket_connect	lsm/socket_connect

IPC	Subclass	Linux system call	Seccomp	eBPF
Message queue	POSIX	mq_open, mq_getsetattr, mq_notify, mq_timedreceive, mq_timedsend, mq_unlink	✓	
	System V	msgctl, msgget, msgrcv, msgsnd	✓	
Pipe	Named	mknod, mknodat, open, openat	✓*	✓
Semaphore	POSIX	futex, mmap	✓*	
	System V	semctl, semget, semop, semtimedop	✓	
Shared memory	POSIX	mmap	✓*	
	System V	shmat, shmctl, shmdt, shmget	✓	
Signal	Standard	kill, pidfd_send_signal, tgkill, tkill	✓	
	Real-time	rt_sigqueueinfo, rt_tgsigqueueinfo	✓	
UNIX socket	Named	bind, connect, mknod, mknodat	✓*	✓

POLICY FORMAT & POLICY GENERATION

- **Easy** to understand policy **syntax**
- Writing a policy can be a complex time-consuming task
- Support **policy generation** with automatic discovery of filesystem, inter-process communication and network requirements

```
[{
    "name": "/usr/bin/curl",
    "type": "executable",
    "fs": {
            "read":   ["/usr/bin/curl", ...],
            "write":  ["response.json"],
            "exec":   ["/usr/bin/curl", ...]
    },
    "ipc": {
            "socket": true,
    },
    "net": [{
            "name":   "https://www.example.com",
            "ports":  [443]
    }]
}]
```

EXPERIMENTS