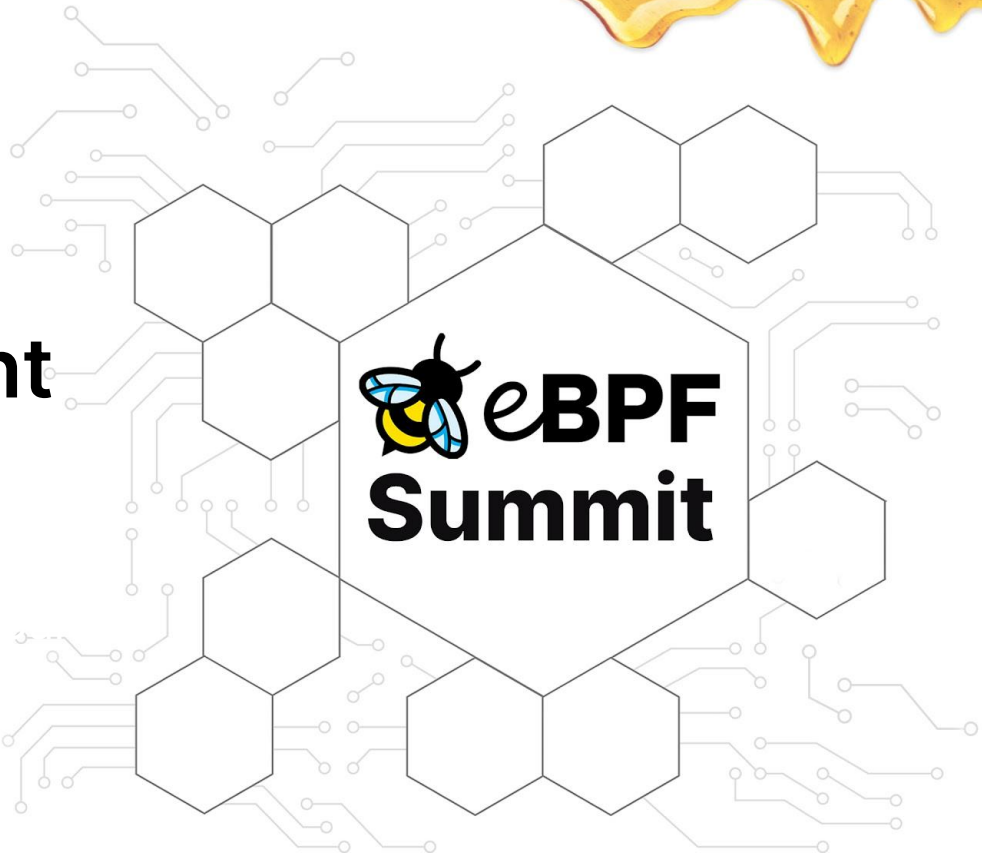
Container Security and Runtime Enforcement with Tetragon



eBPF Summit 2022
Sep 28 - 29

Djalal Harouni, isovalent.com

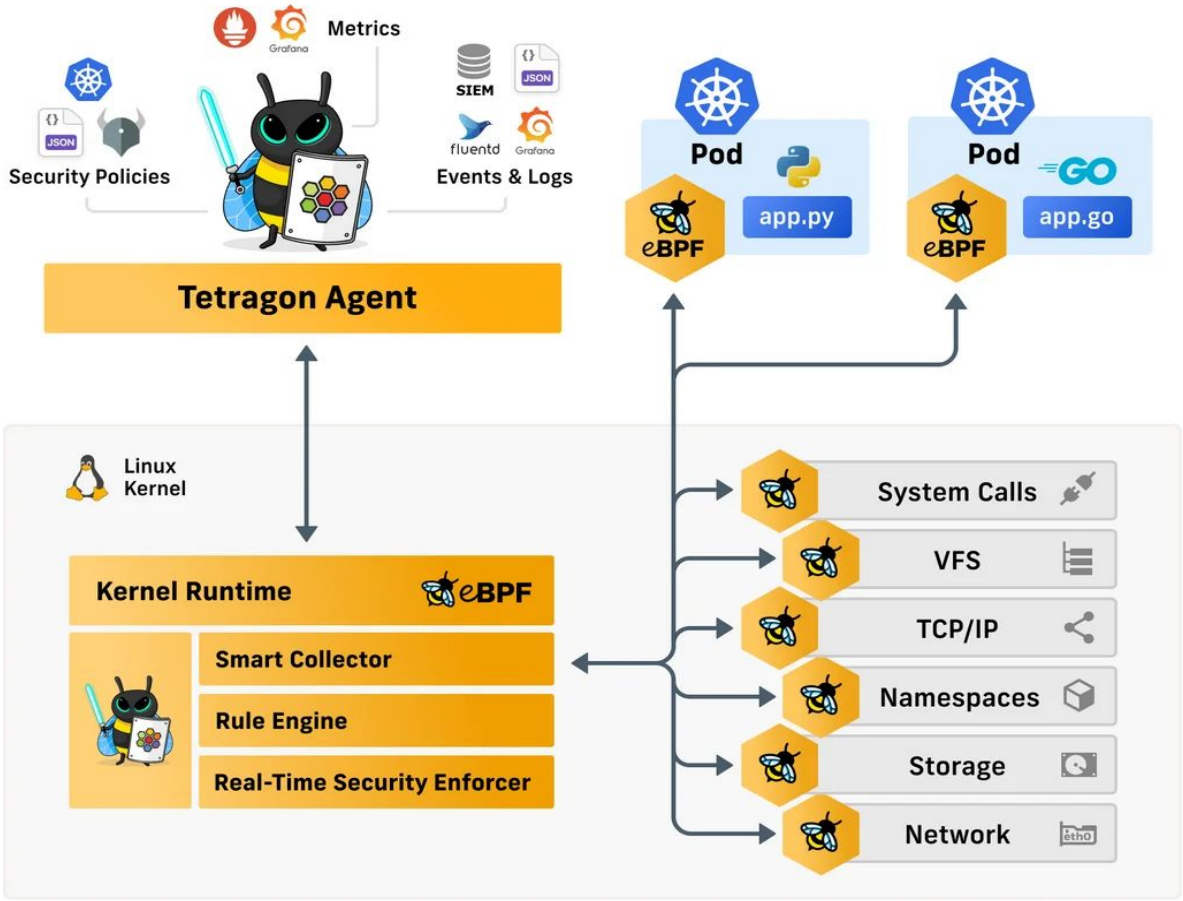
@tixxdz

Tetragon

github.com/cilium/tetragon

Tetragon

- eBPF-based Security Observability & Runtime Enforcement
  - Deep visibility achieved without requiring application changes and is provided at low overhead.
  - Runtime enforcement layer capable of performing access control on the system calls, **but also at other levels and subsystems of the Operating System:**
    - Network, File & I/O activity, VFS, Running Executables, Capabilities usage & changes, Namespaces access, etc

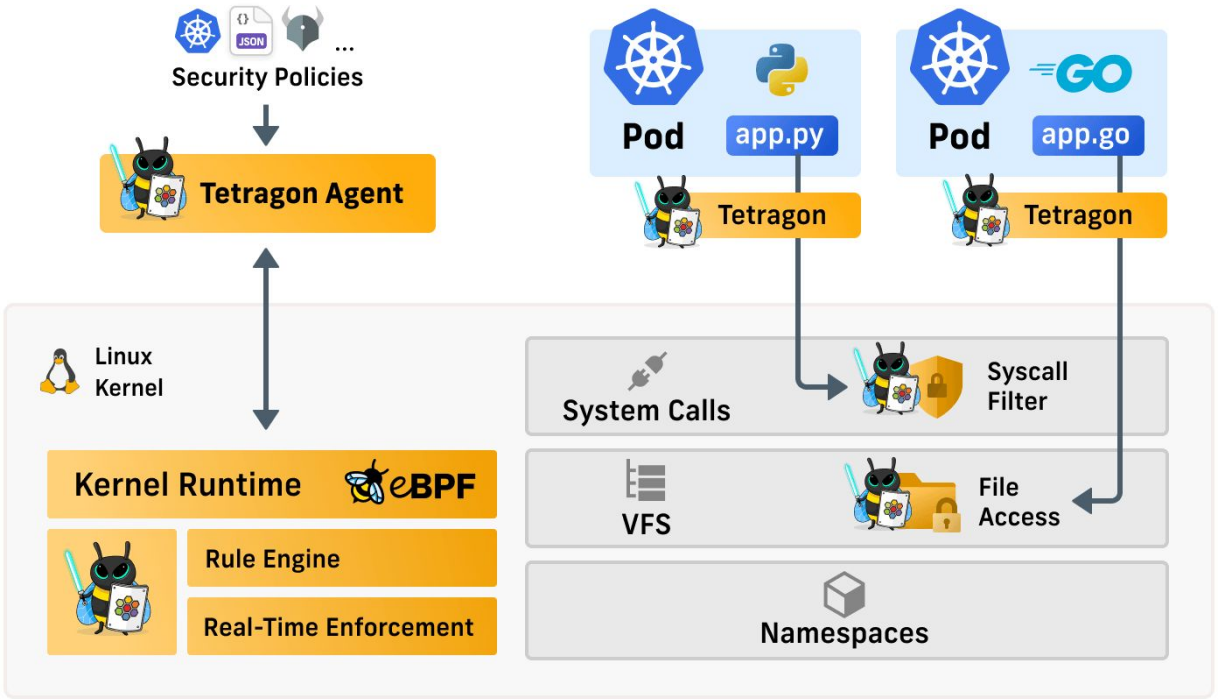

Metrics
Grafana
SIEM
JSON
fluentd
Grafana
Events & Logs
JSON
Security Policies
Pod
app.py
eBPF
Pod
app.go
eBPF
Tetragon Agent
Linux
Kernel
Kernel Runtime
eBPF
Smart Collector
Rule Engine
Real-Time Security Enforcer
System Calls
VFS
TCP/IP
Namespaces
Storage
Network
eth0

Tetragon BPF + BTF

- Using BPF CO-RE allows to run Tetragon on different Linux versions

- BTF (BPF Type Format) the metadata format that encodes the debug info related to BPF programs.

Kubernetes and Containers



Tetragon: Attach Anywhere

- Kprobes: dynamically break into any kernel routine, trap at almost any kernel code address and execute specific probes.
    - Add/Insert Capability checks without changing kernel code
    - Hardening kernel code without changing it by killing offending processes
    - Detect vulnerability exploitation: parameter overflows, etc
    - Attach to Linux Security Modules hooks

In future: bpf-lsm, fentry, etc

Trace Pods and Containers activity

- Trace process execution
- Trace system calls
- Trace Capability checks performed by the kernel on processes

Trace Pods and Containers activity

- Trace Capability checks performed by the kernel on processes

```
"process_kprobe": {
"process": {
        "pid": 101185,
        "uid": 1000,
        "cwd": "/home/testuser",
        "binary": "/bin/su",
        arguments": "-",
        "pod": {
                "namespace": "default",
                "name": "xwing",
                "container": {
                        “name”: “spaceship”,...
                }
        },
        "docker": "816bfe8c3663e4710fb1945f505d1e6"
},
```

```
"function_name": "cap_capable",
"args": [...
        "user_namespace_arg": {
                "level": 1,
                "owner": 1000,
                "group": 1000
        },
        {
        "capability_arg": {
                "value": 2,
                "capability": "CAP_DAC_READ_SEARCH"
        },
]
"return": {
        "int_arg": 0
},
```

System call filters

- Filter and block dangerous system calls like Seccomp
- Load new filters on the fly without a down-time or requiring a Container restart
- Support Docker default Seccomp profile based on an allow or deny list

Examples:

- Kill processes that load kernel modules
- Kill processes that perform raw I/O
- Kill processes that try to manipulate the clock
- ...

Sandbox

- Restrict Linux namespace usage
  - Prevent these kernel exploits and possible container escapes: CVE-2021-22555, CVE-2022-1015, CVE-2022-27666, CVE-2022-32250, CVE-2022-0185, ...

Restrict unprivileged user namespace

```
$ unshare -Urf
Killed
```



```
"process_kprobe": {
        "process": {
                "binary": "/usr/bin/unshare",
                "arguments": "-Urf",
                ...
        },
        "function_name": "create_user_ns",
        "action": "KPROBE_ACTION_SIGKILL"
        }
}
```

Sandbox

- Restrict Capabilities within namespaces

Kill CAP_SYS_PTRACE in user namespace

```
$ unshare -Urpf
# ps aux
Killed
```

```
"process_kprobe": {
        "process": {
                "binary": "/usr/bin/ps",
                "arguments": "aux",
                ...
        },
        "function_name": "cap_capable",",
        "action": "KPROBE_ACTION_SIGKILL"
        }
}
```

Sandbox

- Kill raising capabilities: setting new caps, execve programs with caps, etc
- Restrict loading kernel modules or autoloading “vulnerable” kernel modules

.

Tetragon and Linux Security Modules

Tetragon is able to:

- Attach to Linux Security Modules hooks on the fly
- Ability to kill processes during an LSM hook

Advantage of Tetragon (eBPF based solution):

- Update Kubernetes cluster policies by uploading new CRDs **without a downtime.**
- Does not require a Container or an Operating System restart.

Linux Security Modules

Tetragon and the Capabilities LSM

- Attach to the Capabilities LSM hooks
- Trace capabilities usage: raise, drop, gaining caps from file-exec, etc
- Kill processes that succeeded capabilities checks

Linux Security Modules

Kernel lockdown LSM – kernel image access prevention feature

- Prevents both direct and indirect access to a running kernel image

Tetragon is capable to:

- Attach to the lockdown LSM hooks
- Protect against unauthorized modification of the kernel image
- Restrict loading unsigned kernel modules
- Restrict access to ***/dev/mem, /dev/kmem*** and ***/dev/port***
- Restrict kexec of unsigned images

Linux Security Modules

Safesetid LSM

- An LSM module that gates the setid family of syscalls to restrict UID/GID transitions

Tetragon is capable to:

- Trace and kill setuid()/setgid() calls according to parameters
- Gate the setid family of syscalls based on an allowlist

Tetragon Features Summary

Ability to improve Containers sandbox and enforce runtime restrictions:

- Prevent Container escapes
- Prevent kernel exploits
- Easy to adapt to new techniques

Tetragon sandbox can prevent these exploits:

- CVE-2021-22555
- CVE-2022-0185
- CVE-2022-1015
- CVE-2022-27666
- CVE-2022-32250
- ...

Tetragon Features Summary

Tetragon is able to:

- **Syscalls and Seccomp**: trace and restrict system calls
- **Capabilities**: trace and restrict capabilities and namespaces usage
- **LSM**: attach to Linux Security Module hooks and perform security checks
- **Kernel hardening:** insert capability checks at random places without requiring kernel changes
- **VFS**: monitor and restrict access to sensitive files at the file system layer
- **Prevention**: attach anywhere in the kernel and kill offending processes
- **Network**: combine network and runtime visibility

Conclusion

- eBPF-based Security and Observability solutions are flexible
- Ability to adapt on the fly without a service downtime

Thanks to the Tetragon team for the feedback and ideas!

Djalal Harouni - @tixxdz