SYNACKTIV

Rootkit eBPF LinkPro impliqué dans une compromission AWS
CoRIIN 2026

Théo Letailleur

2026/03/31

TLP:CLEAR

eBPF

SYNACKTIV

- **eBPF :** *extended Berkeley Packet Filter*
  - Permet d'exécuter des programmes vérifiés et sandboxés dans l'espace noyau
- Orchestré depuis l'espace utilisateur
- Cas d'usage : observabilité, sécurité, réseau…
  - Exemple : CNI Cilium, Tetragon, Falco

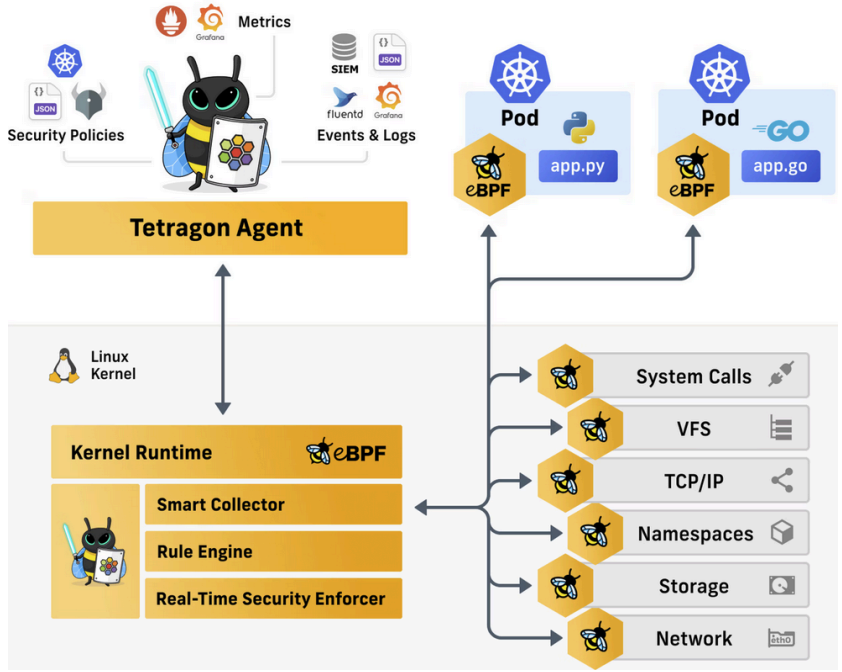


Schéma Tetragon : https://isovalent.com/blog/post/2022-05-16-tetragon/

TLP:CLEAR

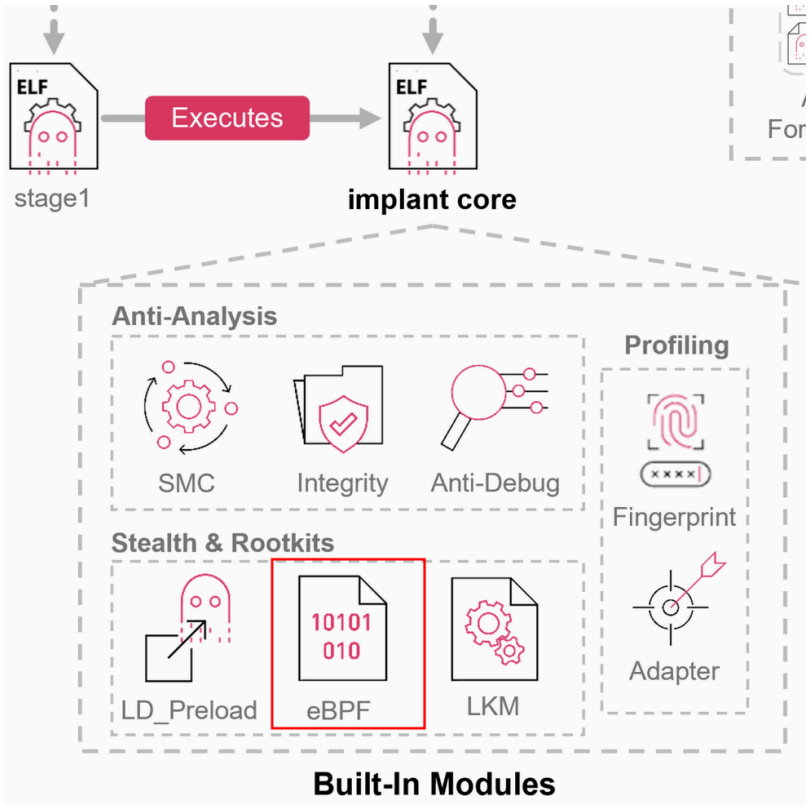
eBPF

Intégré dans les implants Linux modernes

SYNACKTIV

- **Bpfdoor (nouvelle variante)**

- **VoidLink (janvier 2026)**



Bpfdoor : https://www.rapid7.com/blog/post/tr-bpfdoor-telecom-networks-sleeper-cells-threat-research-report/
VoidLink : https://research.checkpoint.com/2026/voidlink-the-cloud-native-malware-framework/

TLP:CLEAR

eBPF

Nouvelle backdoor eBPF

SYNACKTIV

- **LinkPro :** rootkit eBPF Linux non documenté découvert pendant l'analyse d'une compromission AWS

TLP:CLEAR

Contexte

SYNACKTIV

- Réponse sur incident en été 2025
- **SI compromis :** infrastructure **Amazon Web Services**
- **Impact :** détournement de requêtes de paiement, vol de données et demande de rançon

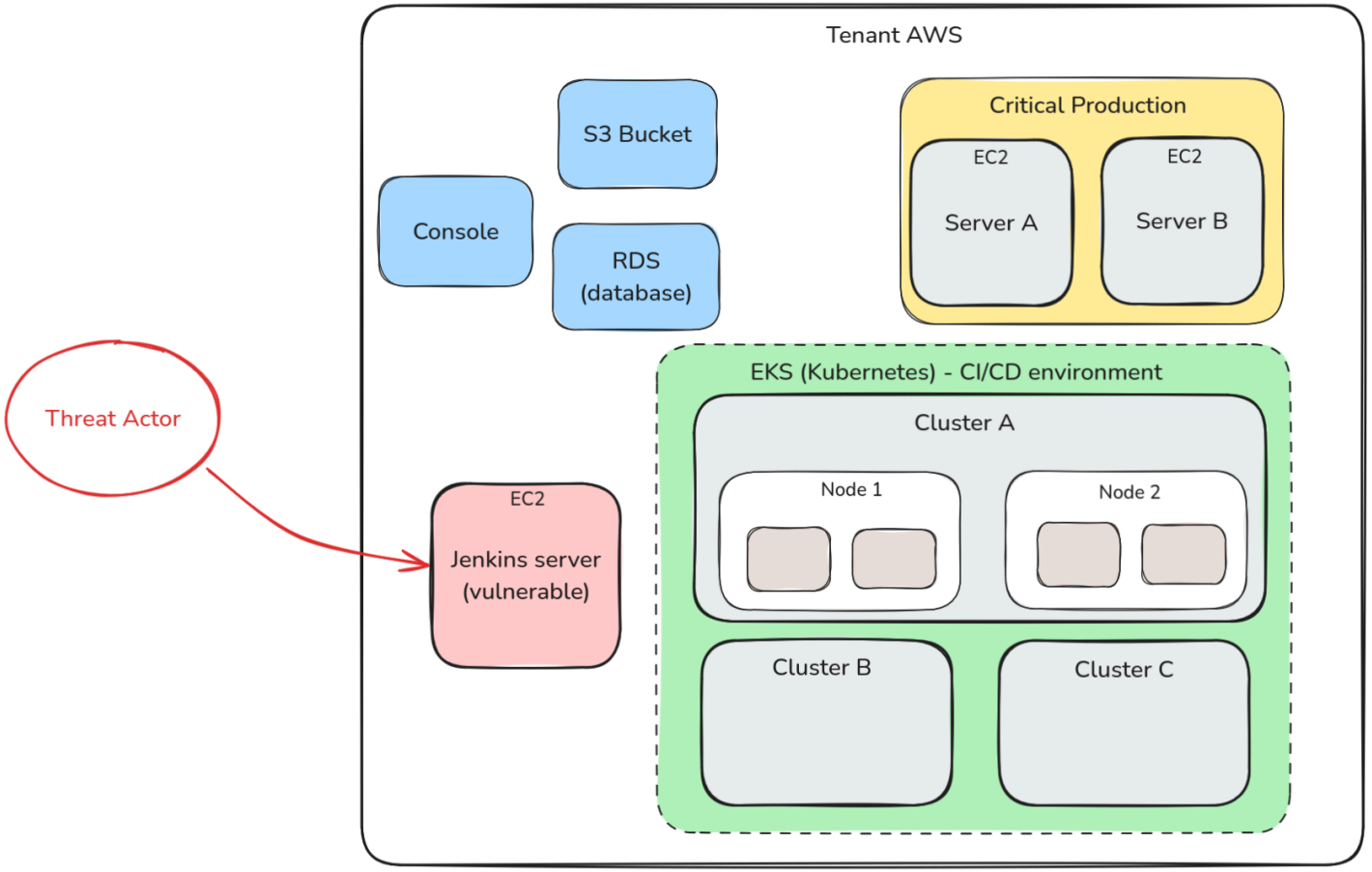
TLP:CLEAR

Chaîne d'infection

SYNACKTIV

Accès initial : Jenkins CVE-2024–23897

- Arbitrary file read vulnerability through the CLI can lead to RCE

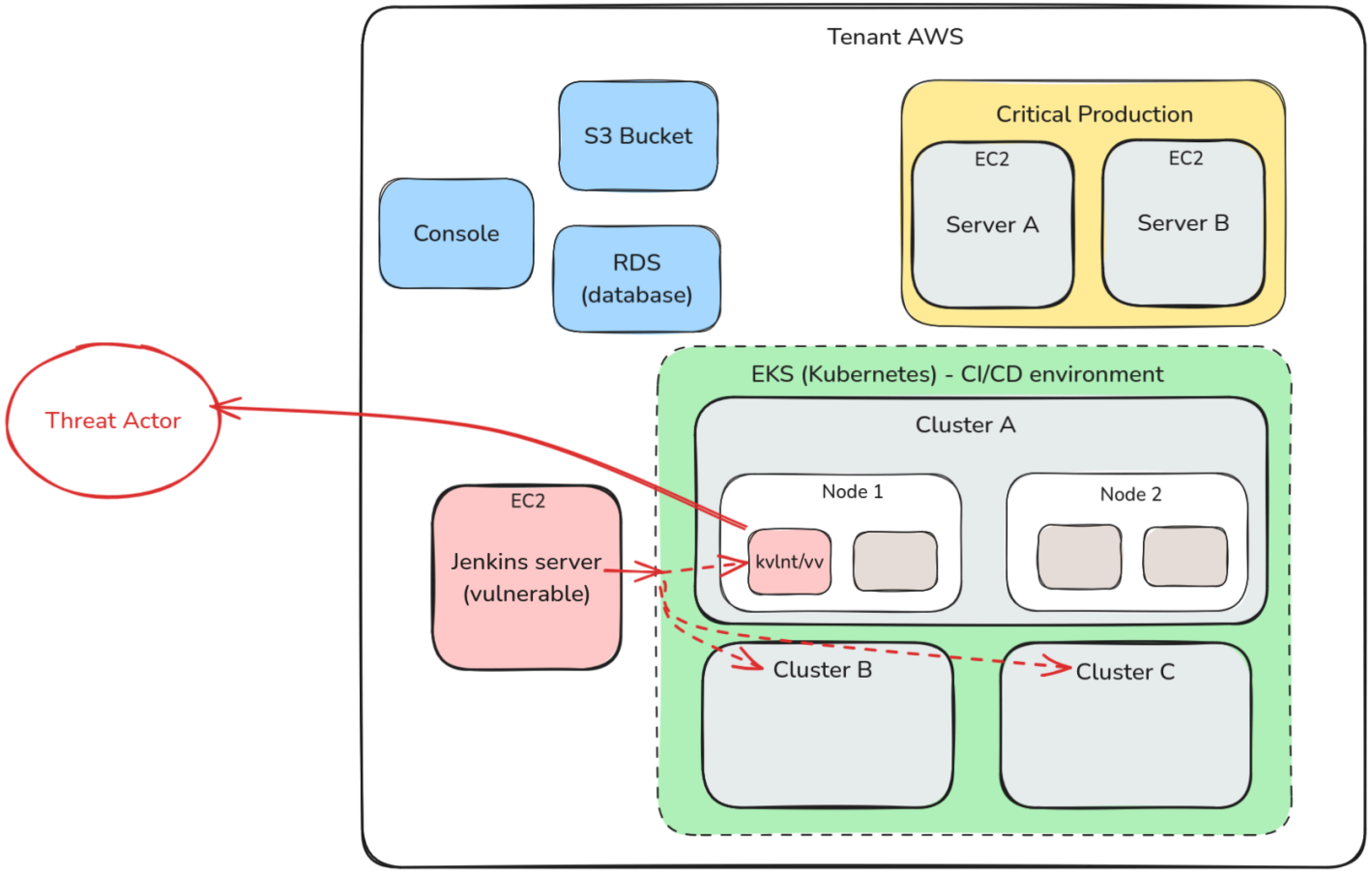


TLP:CLEAR

Chaîne d'infection

SYNACKTIV

- **Mouvement latéral :** Déploiement de **pods malveillants**
  - Nom de l'image docker : `kvlnt/vv` (hébergée sur DockerHub)
  - Exécution d'un VPN **vnt** et de la backdoor **vShell** (WebSocket)

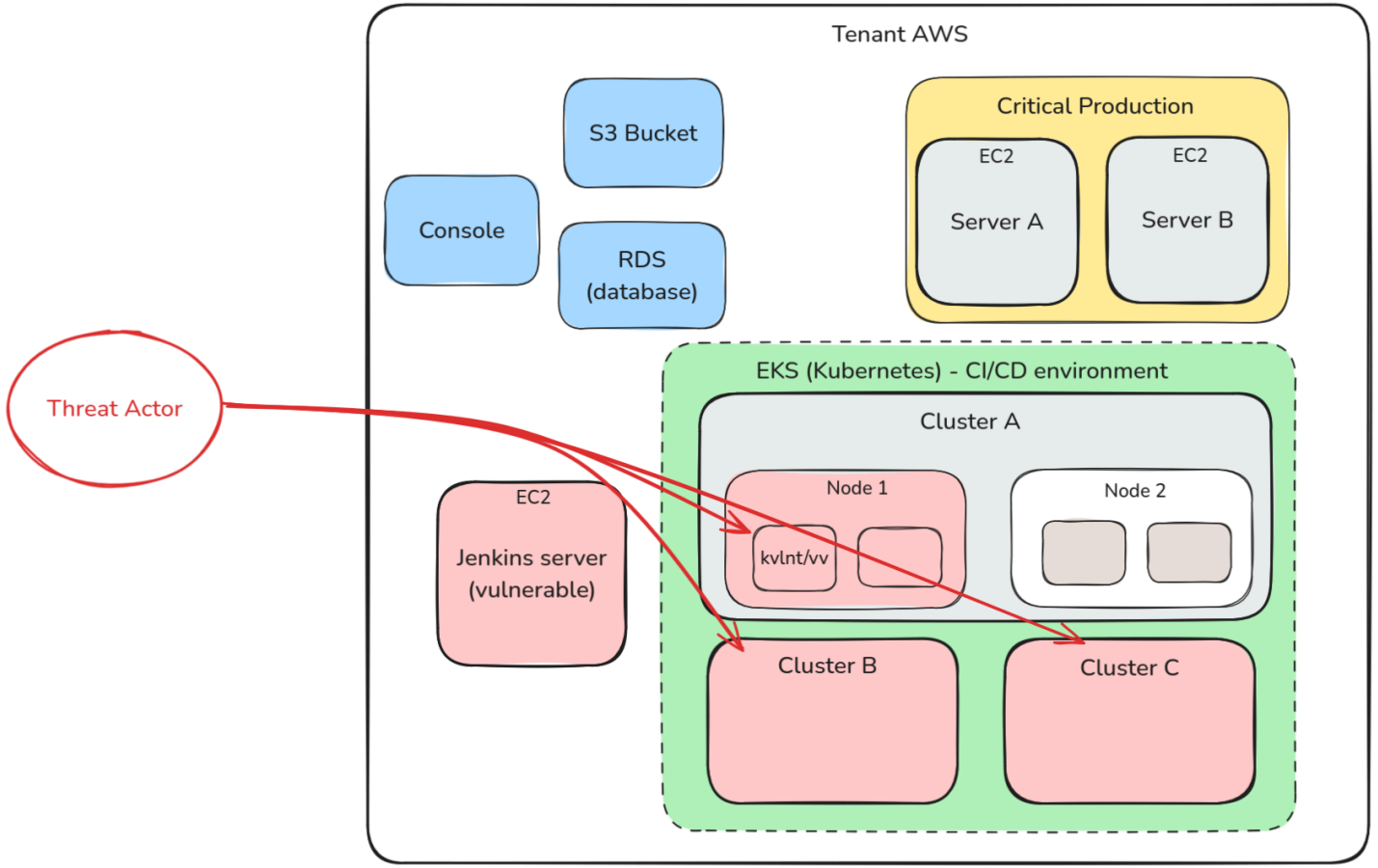


TLP:CLEAR

Chaîne d'infection

SYNACKTIV

- **Elevation de privilèges :**
  - Pods privilégiés → ~ `root` sur le nœud (toutes les Capabilities + `/dev`)
  - HostPath `/` : `/mnt` → Accès au FS du noeud en lecture/écriture



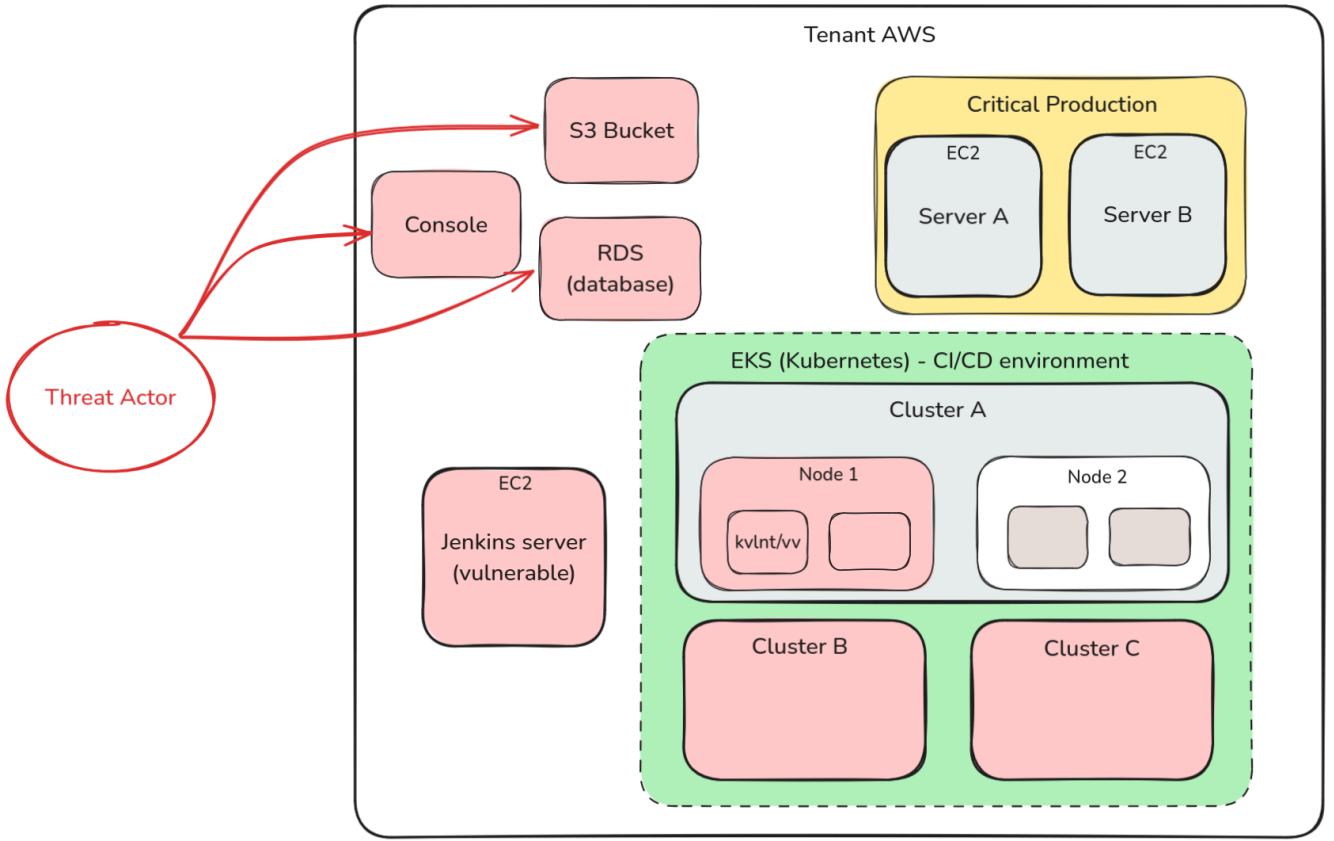
TLP:CLEAR

Chaîne d'infection

SYNACKTIV

Accès à des identifiants

- Lecture des volumes et secrets des autres services, AccessKey de comptes AWS
- Utilisation de **TruffleHog** : usurpation de **rôles AWS privilégiés**
  - Accès à la console, aux S3, aux RDS et aux EC2 : **exfiltration de données**



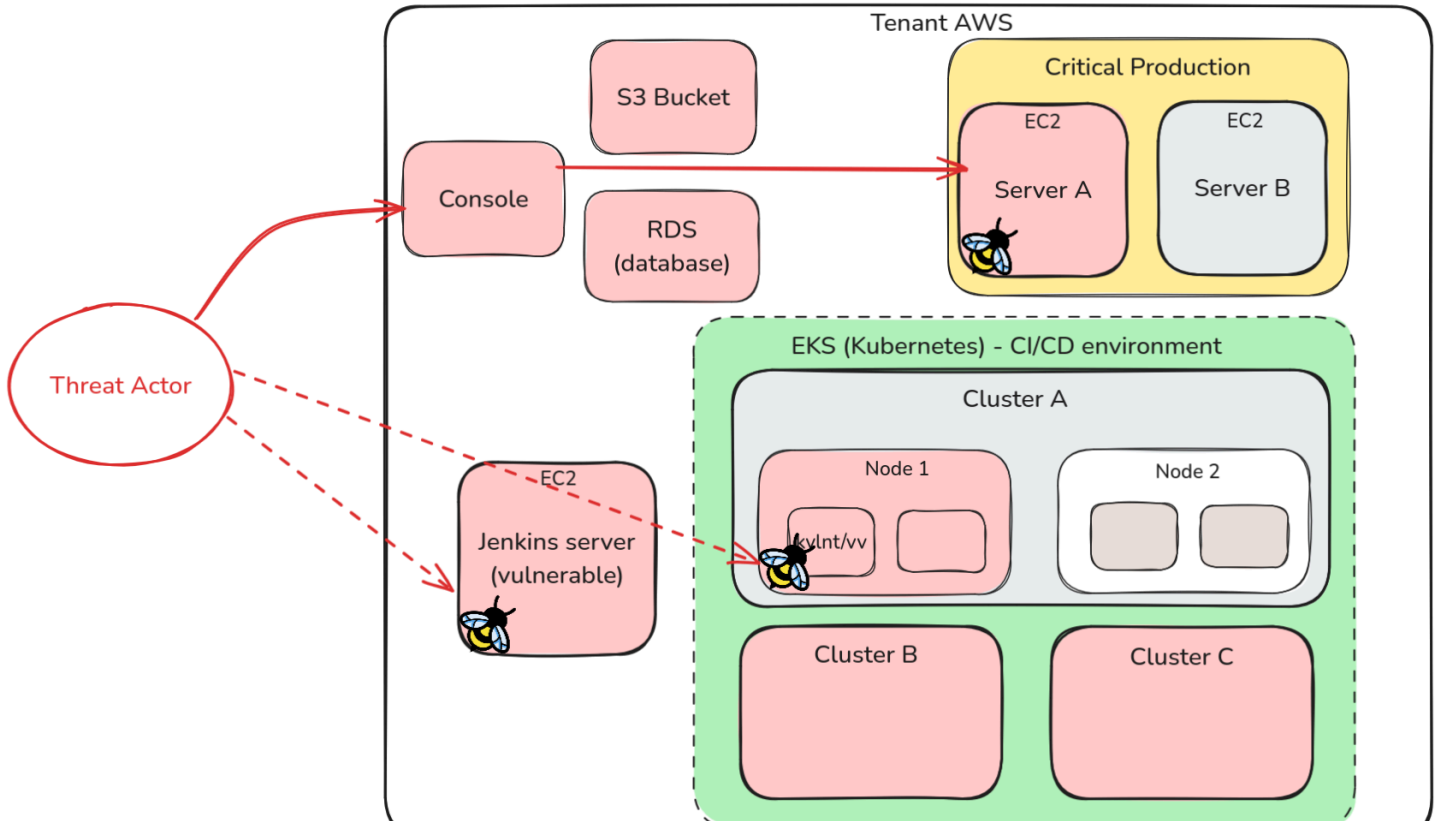
TLP:CLEAR

Chaîne d'infection

SYNACKTIV

ⓘ Trois semaines plus tard : déplacement vers des serveurs métiers

- **Dropper vShell** : chargé en mémoire (DNS *tunneling*)
- **Rootkit eBPF LinkPro** : seul implant persistant sur les systèmes (deux variantes)



TLP:CLEAR

SYNACKTIV

LinkPro

TLP:CLEAR

LinkPro

Modularité de son comportement

SYNACKTIV

- Nom de fichier **=~** `.tmp~data.*`
- **Golang** (mod `link-pro/link-client`)
- Modulation possible via des **arguments en cli**
- **Deux modes :** reverse ou forward
  - `reverse` (passif) : service HTTP
  - `forward` (actif) : HTTP, WS, UDP, TCP, DNS

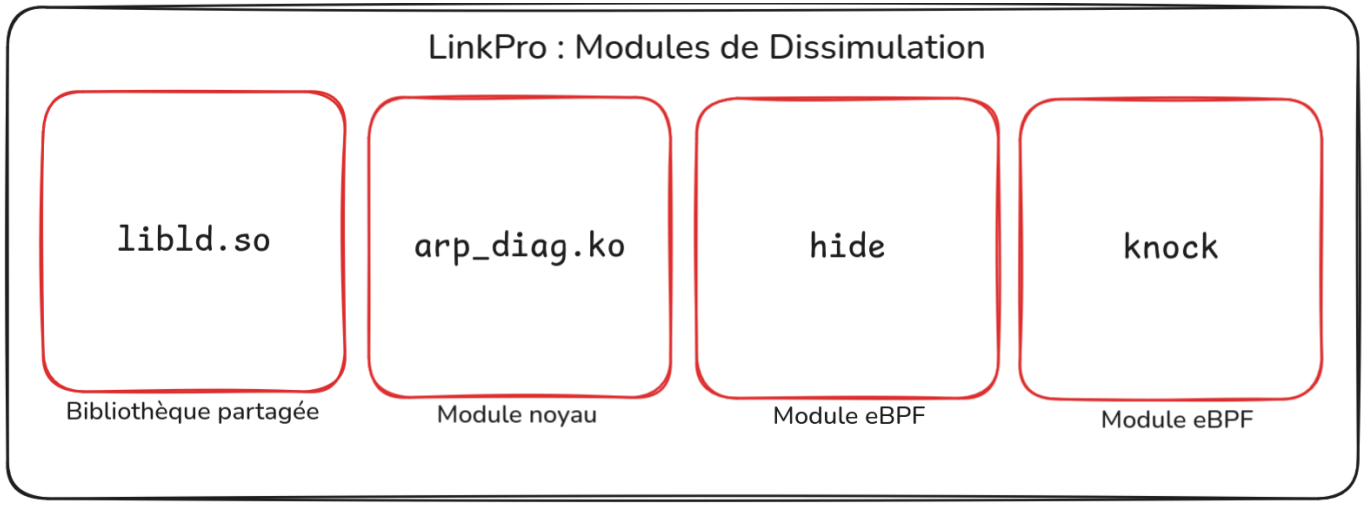
	`d5b2202b`	`1368f3a8`
	Mode passif HTTP	**Mode actif HTTP**
ServerAddress	1.1.1.1 *(non utilisé)*	18.199.101.111
ServerPort	6666	2233
SecretKey	0	3344
SleepTime	10	10
JitterTime	2	2
Protocol	http	http
DnsDomain	dns.example.com	dns.example.com
DNSMode	tunnel	tunnel
DnsServer	0	0
Debug	false	false
Version	1.0.0	1.0.0
ConnectionMode	reverse	forward
ReversePort	2233	2233

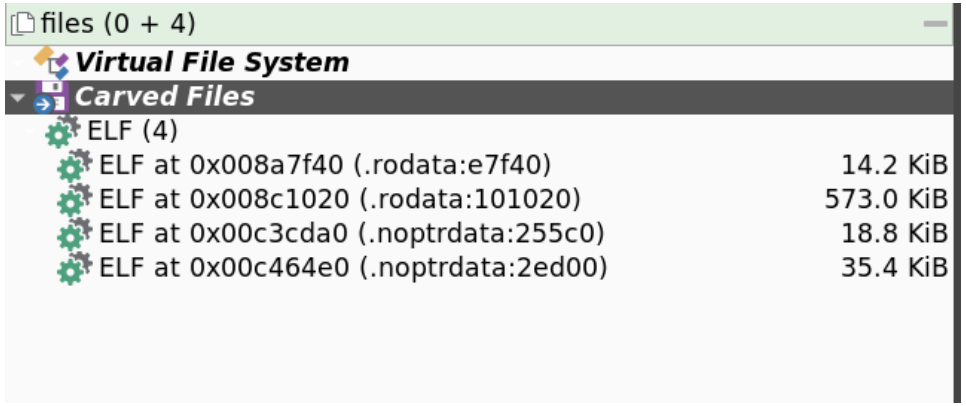
LinkPro

Dissimulation

SYNACKTIV

Quatre modules ELF embarqués





TLP:CLEAR

SYNACKTIV

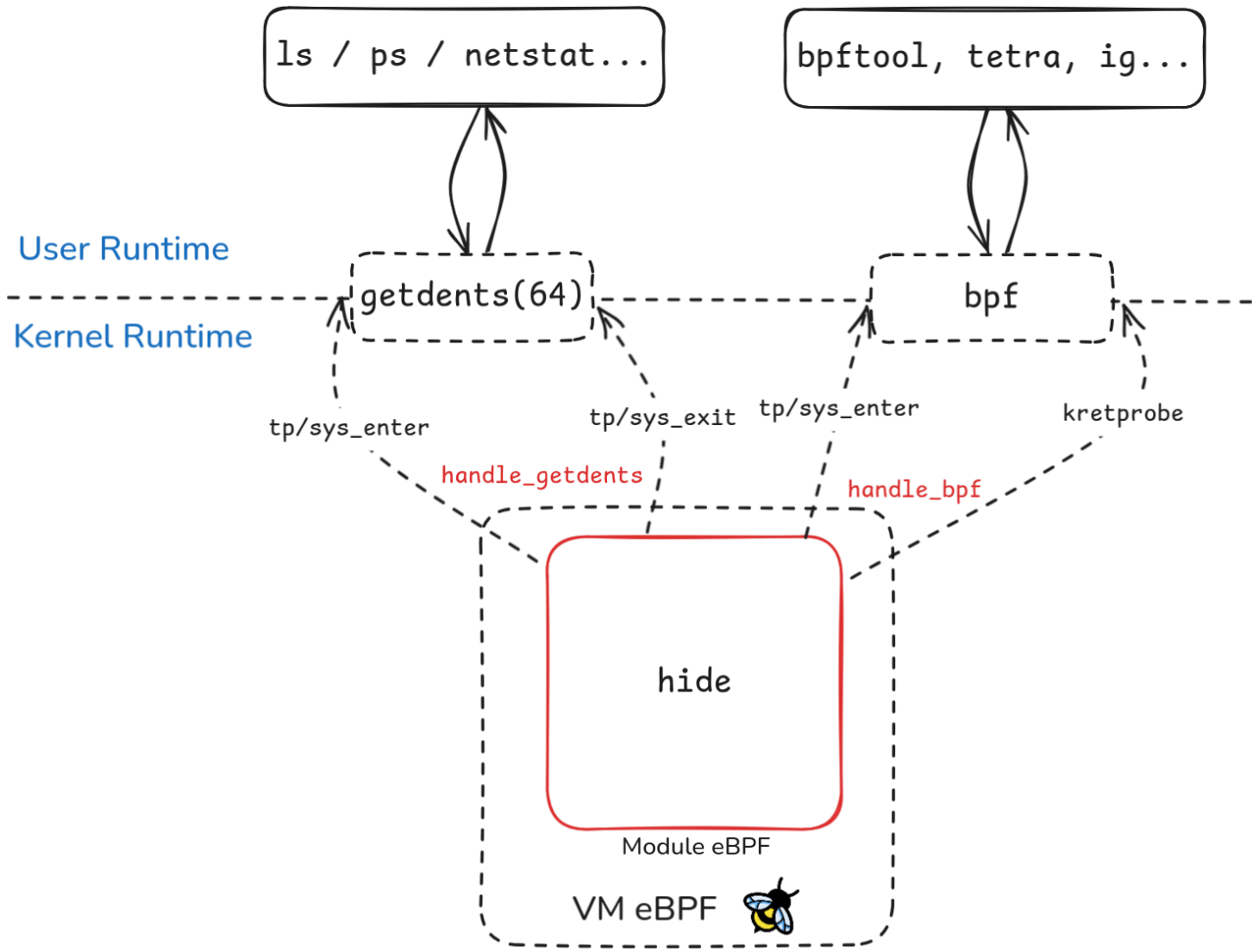
Module eBPF Hide

TLP:CLEAR

LinkPro

Module eBPF Hide

SYNACKTIV



TLP:CLEAR

LinkPro

Module eBPF Hide

SYNACKTIV

Hooks SYS_getdents

- `getdents(int fd, struct linux_dirent *dirp, size_t count);`

```
struct linux_dirent {
    unsigned long  d_ino; // Numéro d'inode
    unsigned long  d_off; //
    unsigned short d_reclen; // Taille de cette entrée dirent
    char           d_name[]; // Nom du fichier
}
```

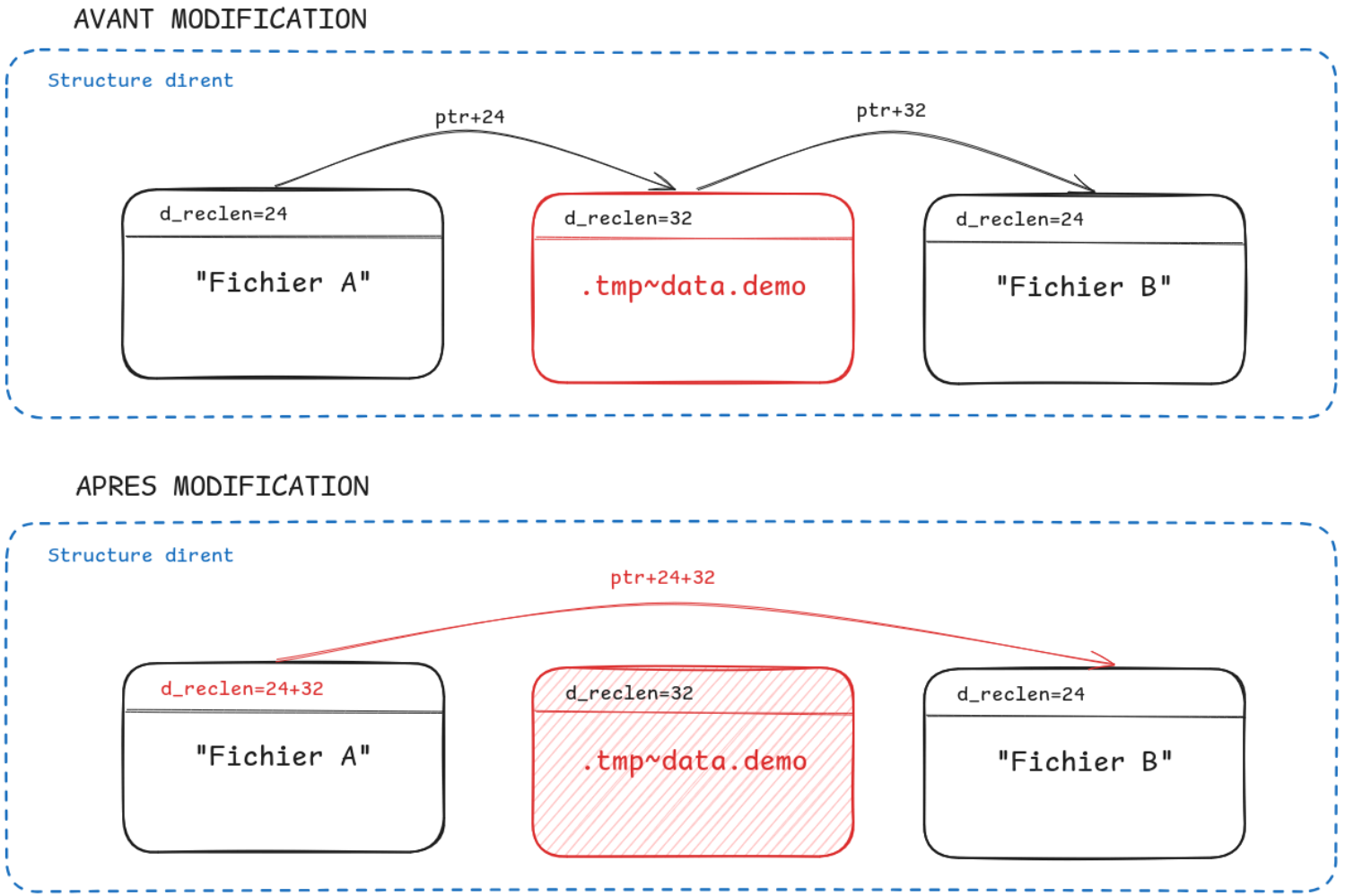
- **Objectif :** noms de fichiers **LIKE** `.tmp~data`, `.system` **OU** une liste de **PIDs** (`pids_to_hide`)
  - → Liste de `dirent` altérée
- Map eBPF `pids_to_hide`
  - Contient le **PID de LinkPro** et une liste spécifiée en argument (`-pid`)

TLP:CLEAR

LinkPro

SYNACKTIV

Module eBPF Hide - Hooks `SYS_getdents`



TLP:CLEAR

LinkPro

SYNACKTIV

Module eBPF Hide - Hooks `SYS_bpf`

- `int bpf(int cmd, union bpf_attr *attr, unsigned int size);`
  - `cmd` : définit le type d'opération à exécuter (ex : lister les programmes eBPF)
- Le hook surveille en entrée (`sys_enter`) les commandes BPF `BPF_PROG_GET_NEXT_ID` , `BPF_MAP_GET_NEXT_ID` , et `BPF_LINK_GET_NEXT_ID`

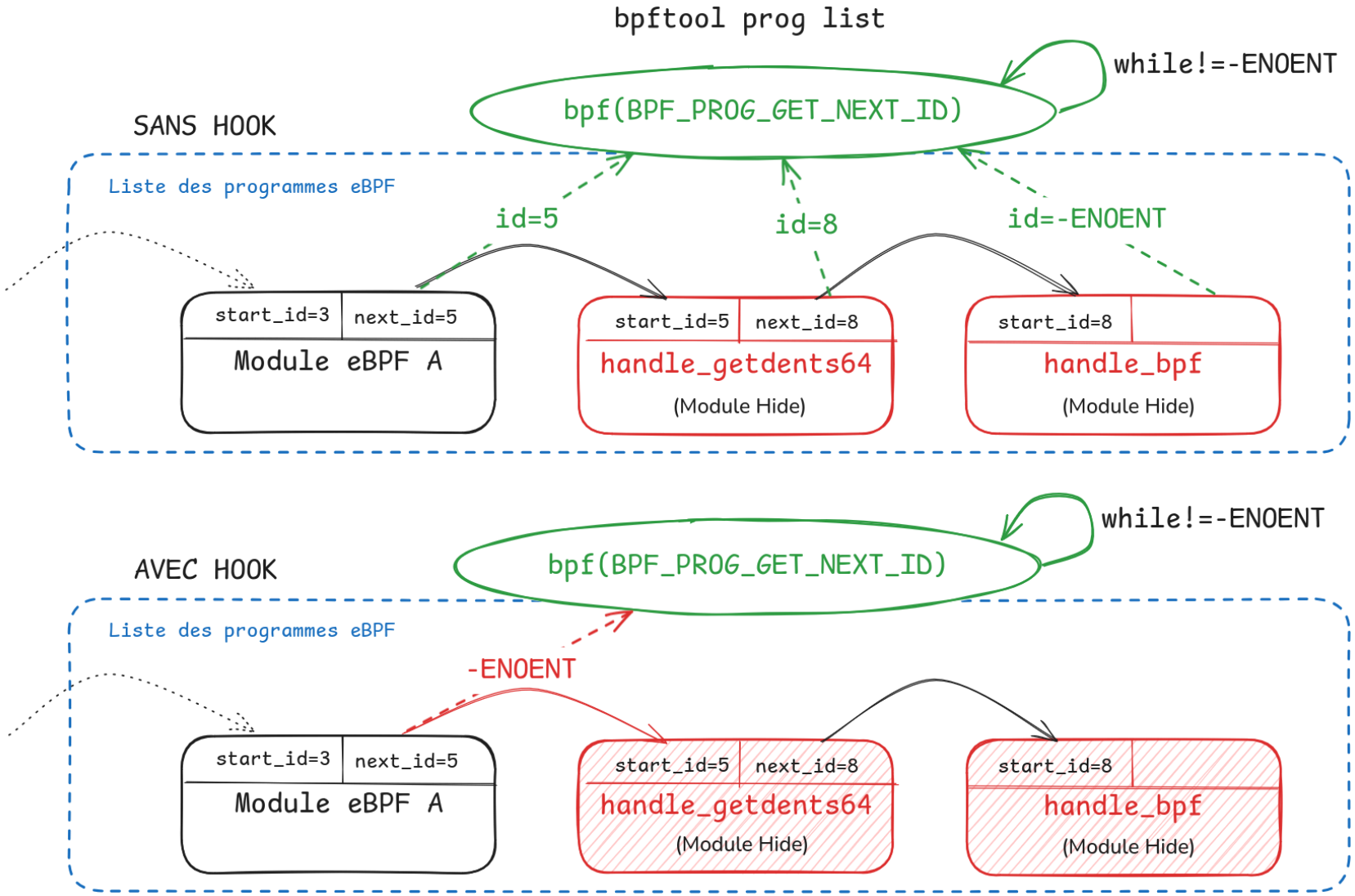
```
struct bpf_attr { // Cas cmd == BPF_*_GET_NEXT_ID
    __u32 start_id;
    __u32 next_id
};
```

- **Objectif :** modifier l'attribut `bpf_attr→next_id` retourné par `bpf()` pour dissimuler les programmes, maps, liens eBPF chargés par LinkPro (`hide` et `knock`) face à des outils de traçage tel que `bpftool`

LinkPro

SYNACKTIV

Module eBPF Hide - Hooks `SYS_bpf`



TLP:CLEAR

Démo eBPF Hide

Démo eBPF Hide

SYNACKTIV

```
root>
```

TLP:CLEAR

LinkPro

Module eBPF Hide

SYNACKTIV

```
bpf_override_return(ctx, -ENOENT);
```

⚠ CONFIG_BPF_KPROBE_OVERRIDE

- `bpf_override_return()` utilisé par le module eBPF `hide` (**kretprobe** sur `sys_bpf`)
- Disponible que si le noyau Linux a été compilé avec l'option de configuration `CONFIG_BPF_KPROBE_OVERRIDE=1`
- Certains noyaux Linux ne sont pas compatibles (ex : Debian stable)

- Besoin d'un mécanisme de repli :
  - module `libld.so` installé dans `/etc/ld.so.preload`

SYNACKTIV

Module eBPF Knock

TLP:CLEAR

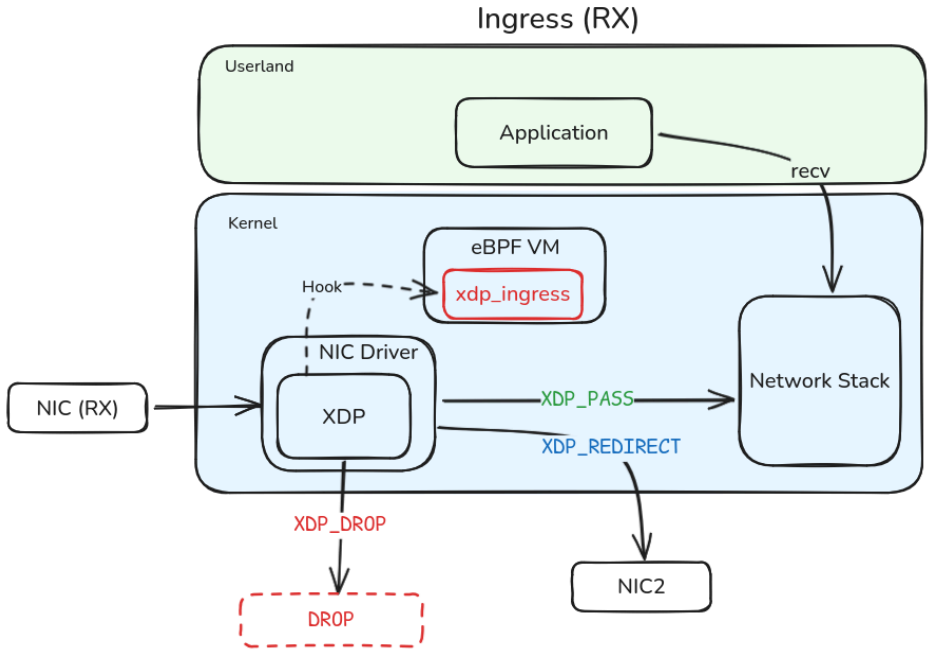
LinkPro

Module eBPF Knock

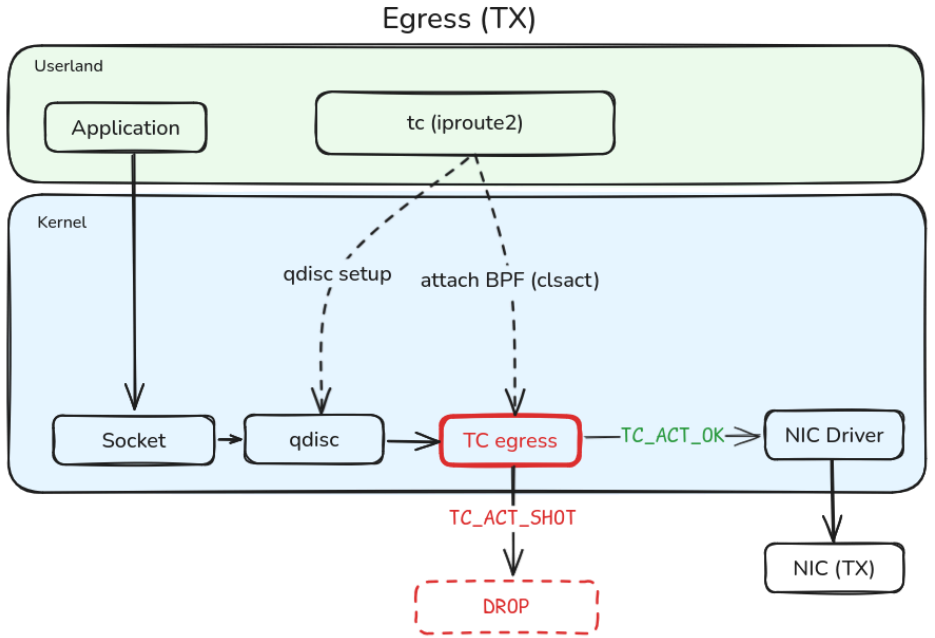
SYNACKTIV

Programmes XDP et TC

- *eXpress Data Path*



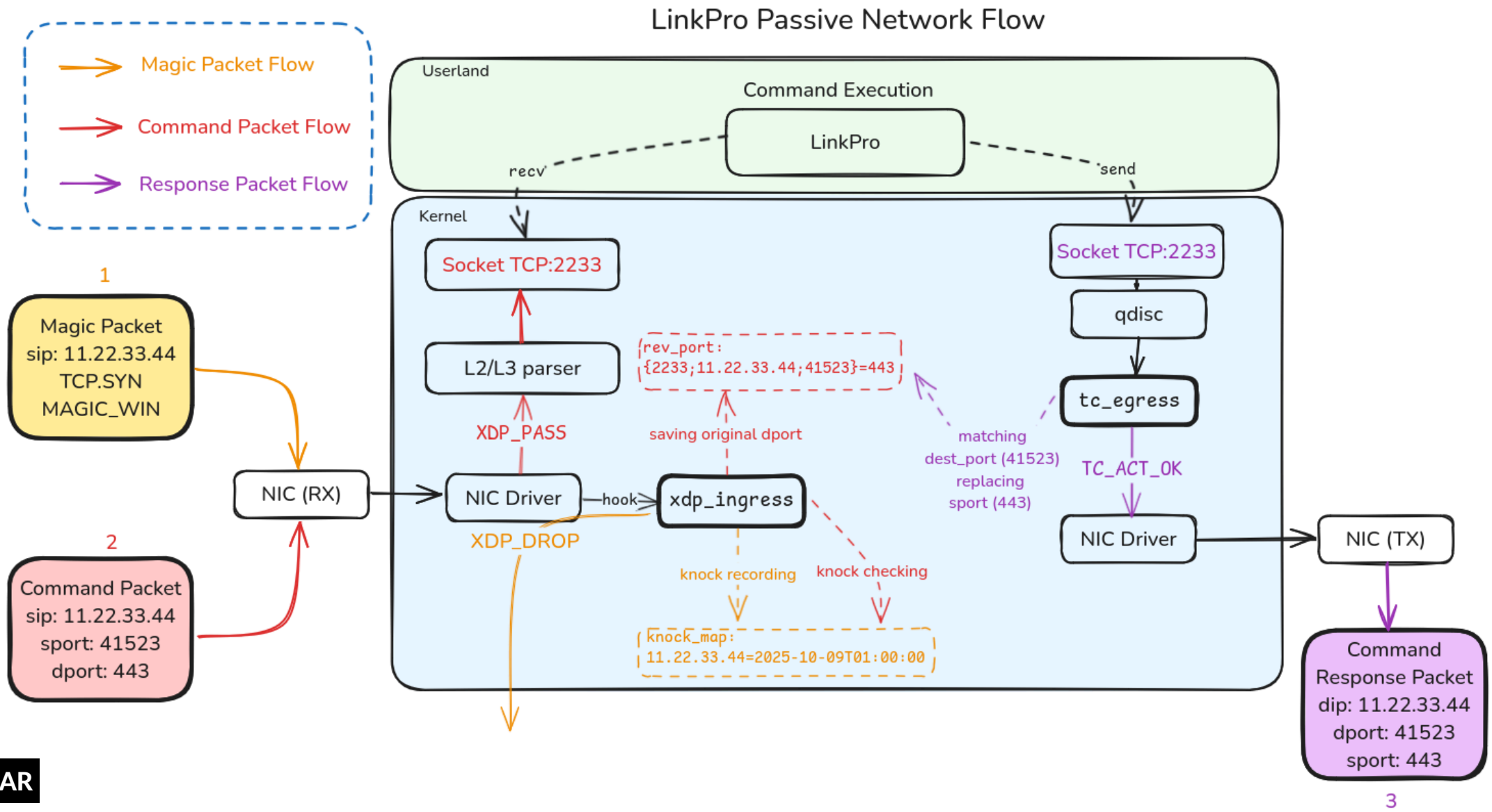
- *Traffic Control (egress)*



TLP:CLEAR

LinkPro - Module eBPF Knock

SYNACKTIV



TLP:CLEAR

SYNACKTIV

Démo eBPF Knock

TLP:CLEAR

Démo eBPF Knock

SYNACKTIV

TLP:CLEAR

LinkPro

SYNACKTIV

- **Commandes :**
  - Commandes shell arbitraires
  - Gestion de pty distant
  - Gestionnaire de fichier (download/upload/list/read/write/delete…)
  - Relai de tunnel proxy SOCKS5 (utilise `resocks`)
- **Persistance :** `systemd-resolveld`
  1. Se copie vers `/usr/lib/.system/.tmp~data.resolveld`
  2. Nouveau fichier unit **systemd** `/etc/systemd/system/systemd-resolveld.service`

TLP:CLEAR

Recap

SYNACKTIV

- **LinkPro :** implant Linux adaptatif
  - **Dissimulation :** hooks eBPF (`getdents` / `bpf`), mécanisme de repli (`ld.so.preload`)
  - **Interception réseau :** eBPF XDP/TC, activation sur réception d'un paquet magique, redirection vers le port d'écoute
- Pour aller plus loin :
  - Article : https://www.synacktiv.com/publications/linkpro-analyse-dun-rootkit-ebpf
  - Règles YARA : https://github.com/synacktiv/synacktiv-rules

SYNACKTIV

Détection des implants eBPF

TLP:CLEAR

Détection

Kernel logs

- Le module eBPF Hide utilise le helper `bpf_probe_write_user` pour modifier la liste de `dirent` retournée par `getdents` et `getdents64`.
- Problème, cela **génère un évènement noyau** à l'installation du module :

```
kernel: [   12.429867] .tmp~data.resol[474] is installing a program with bpf_probe_write_user helper that may corrupt user memory!
```

⚠ Keep in mind that this feature is meant for experiments, and it has a risk of crashing the system and running programs. Therefore, when an eBPF program using this helper is attached, a warning including PID and process name is printed to kernel logs.

eBPF.io documentation: https://docs.ebpf.io/linux/helper-function/bpf_probe_write_user/

Détection

Traçage eBPF

SYNACKTIV

- Les rootkits eBPF contournent les outils de supervision et d'investigation se reposant sur les appels système (`getdents64` , `bpf`)
- La **VM eBPF** peut aussi être utilisée à notre avantage !
  1. Supervision de l'appel système `bpf()` en amont
  2. Inspection du sous-système eBPF avec `bpftool`
  3. Analyse des divergences avec les **outils BCC** et la `libbpf`

BCC Tools : https://github.com/iovisor/bcc

Détection

SYNACKTIV

Supervision de l'appel système `bpf()`

Alerte Falco

Source

syscall

Hostname

minikube-m02

Priority

Critical

Rule

Unexpected eBPF Program Loaded

Output

```
12:18:34.892228990: Critical An unknown process is interacting with the eBPF subsystem!  (bin_path=/home/docker/.tmp~data.demo
command=.tmp~data.demo pid=114045 user=root bpf_cmd=BPF_PROG_LOAD fd_returned=38 container_id=host) container_id=host
container_name=host container_image_repository= container_image_tag= k8s_pod_name=<NA> k8s_ns_name=<NA>
```

Fields

container.id	host	container.image.repository		container.image.tag		container.name	host	evt.arg.cmd	BPF_PROG_LOAD	evt.rawres	38
evt.time	1772281114892228900	k8s.ns.name		k8s.pod.name		proc.cmdline	.tmp~data.demo	proc.exepath	/home/docker/.tmp~data.demo	proc.pid	114045
user.name	root										

Tags

ebpf host mitre_defense_evasion rootkit

```
$ ls -l /proc/114045/fd/38
lrwx------ 1 root root 64 Feb 28 12:19 /proc/114045/fd/38 -> anon_inode:bpf-prog
```

TLP:CLEAR

Détection

SYNACKTIV

Inspection du sous-système eBPF avec `bpftool`

```
$ cat /proc/114045/fdinfo/38
pos:    0
flags:  02000002
mnt_id: 18
ino:    1062
prog_type:      2
prog_jited:     1
prog_tag:       fab22b923bde9f72
memlock:        4096
prog_id:        61
run_time_ns:    0
run_cnt:        0
recursion_misses:       0
verified_insns: 65

$ bpftool prog list | grep 61
#Note : vide à cause du hook de LinkPro

$ bpftool prog show id 61
61: kprobe  name handleBpfExit  tag fab22b923bde9f72  gpl
        loaded_at 2026-02-28T12:18:34+0100  uid 0
        xlated 496B  jited 281B  memlock 4096B  map_ids 35,36,34
        btf_id 97

$ bpftool prog dump xlated id 61
int handleBpfExit(struct pt_regs * ctx):
; int handleBpfExit(struct pt_regs *ctx) {
   0: (bf) r6 = r1
; size_t pid_tgid = bpf_get_current_pid_tgid();
...
```

- `/proc/PID/fdinfo/`
  - informations des *file descriptors*
- `bpftool prog list`
  - liste des programmes eBPF chargés
- `bpftool prog show id 61`
  - informations d'un programme eBPF d'ID 61
- `bpftool prog dump ...`
  - code du programme (kretprobe sur bpf)

Détection

Analyse des divergences avec les outils BCC et libbpf

SYNACKTIV

- `syscount-bpfcc` : comptage d'appels systèmes par PID
  - Même caché, le rootkit doit interagir avec le système

```
$ syscount-bpfcc -P
PID    COMM               COUNT
122882   syscount-bpfcc      1706
94890    systemd-oomd         702
112563   sshd                 468
114045   .tmp~data.demo       458
102822   sudo                  30

$ ps aux |grep 114045 | grep -v grep
<vide>
```

- `getdents64` hook par les rootkits eBPF
- Kprobe sur `filldir64` : fonction noyau appelée par `getdents64`
  - Utilisation de `libbpf`

```
$ python3 kprobe_filldir64_ls.py

🥷  Interception de filldir64 en cours...
468.800335: bpf_trace_printk: Kernel generated: .
468.800350: bpf_trace_printk: Kernel generated: .local
468.800352: bpf_trace_printk: Kernel generated: .ssh
468.800353: bpf_trace_printk: Kernel generated: .bash_history
468.800353: bpf_trace_printk: Kernel generated: .bash_rc
468.800354: bpf_trace_printk: Kernel generated: .tmp~data.demo
468.800355: bpf_trace_printk: Kernel generated: ..
```

- Bindings Python pour enregistrer le hook et récupérer les logs

Supervision Cloud/Kubernetes

Recommandations

SYNACKTIV

- Surveiller les **programmes eBPF** chargés via l'appel système `bpf`
  - En particulier dans les environnements où BPF n'est pas censé être utilisé
- **Auditer les journaux noyau :** avertissement d'utilisation de `bpf_probe_write_user`
- **Kubernetes :** Implémenter des **politiques de sécurité des pods** (PSA) empêchant
  - l'utilisation de conteneurs privilégiés
  - le montage de la partition de l'hôte
  - l'accès à la socket Docker
- Intégrer **évènements K8s** et **journaux des conteneurs** dans la supervision

Pod Security Admission : https://kubernetes.io/docs/concepts/security/pod-security-admission/

SYNACKTIV

https://www.linkedin.com/company/synacktiv

https://x.com/synacktiv

https://bsky.app/profile/synacktiv.com

https://synacktiv.com