SYNACKTIV

Utiliser Suricata pour mieux se défendre en CTF

Ambre Iooss — SSTIC 2026

https://github.com/FCSC-FR/shovel

Introduction

SYNACKTIV

Joueuse à l'*European Cybersecurity Challenge* (supervisé par ENISA) puis coach.

- Conseille la *TeamFrance* sur la technique et stratégie pour les CTF Attaque-Défense.
- **Developpe depuis 4 ans un outil basé sur Suricata pour aider la TeamFrance.**

European Cybersecurity Challenge 2023 en Norvège

Capture-The-Flag Attaque-Défense

SYNACKTIV

CTF Jeopardy : un flag secret par tâche résolue, e.g. le challenge du SSTIC.

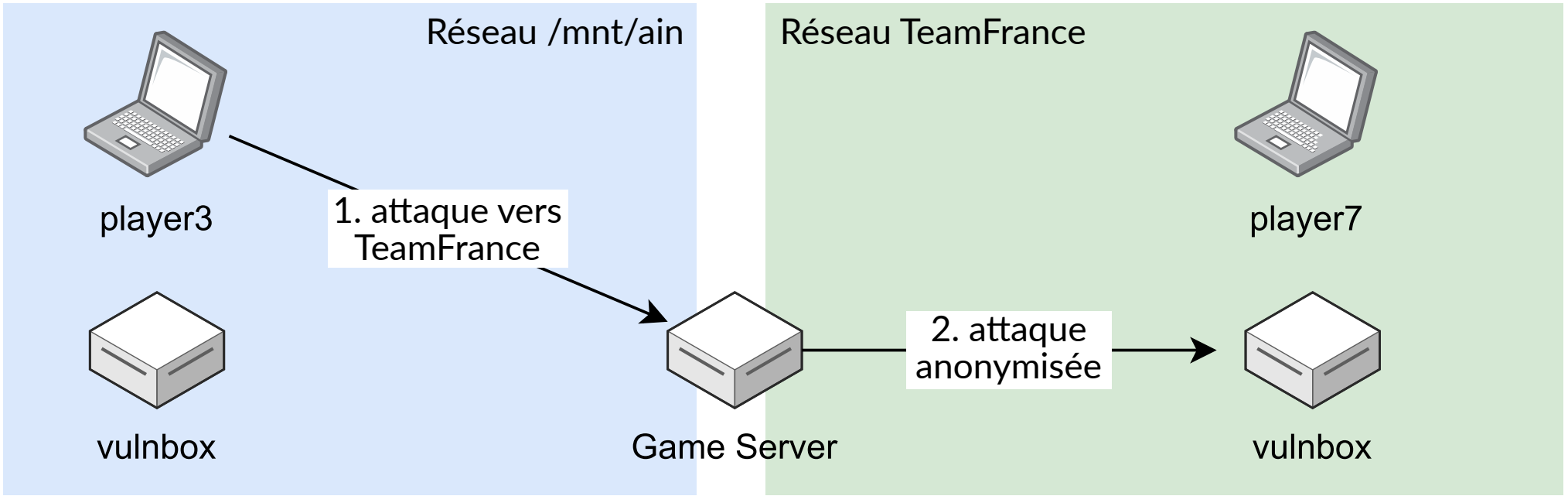
CTF Attaque-Défense :

- machines "*vulnbox*" pour chaque équipe, provisionnées à l'identique (souvent GNU/Linux).
- 4 à 10 services avec des vulnerabilitées intentionnelles.
- ~8 heures de réseau ouvert entre les équipes, avec anonymisation du trafic.

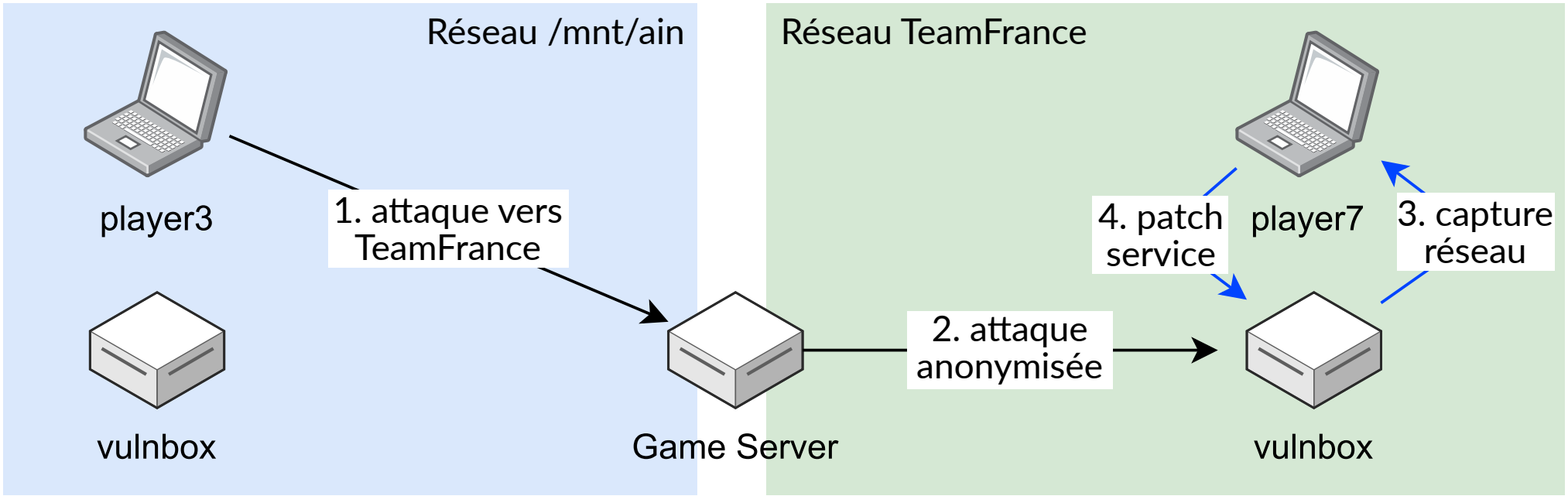
But de chaque équipe :

1. Protéger les flags stockés dans ses services.
2. Voler les flags des autres équipes en exploitant leurs services.
3. Garder une disponibilité élevé.

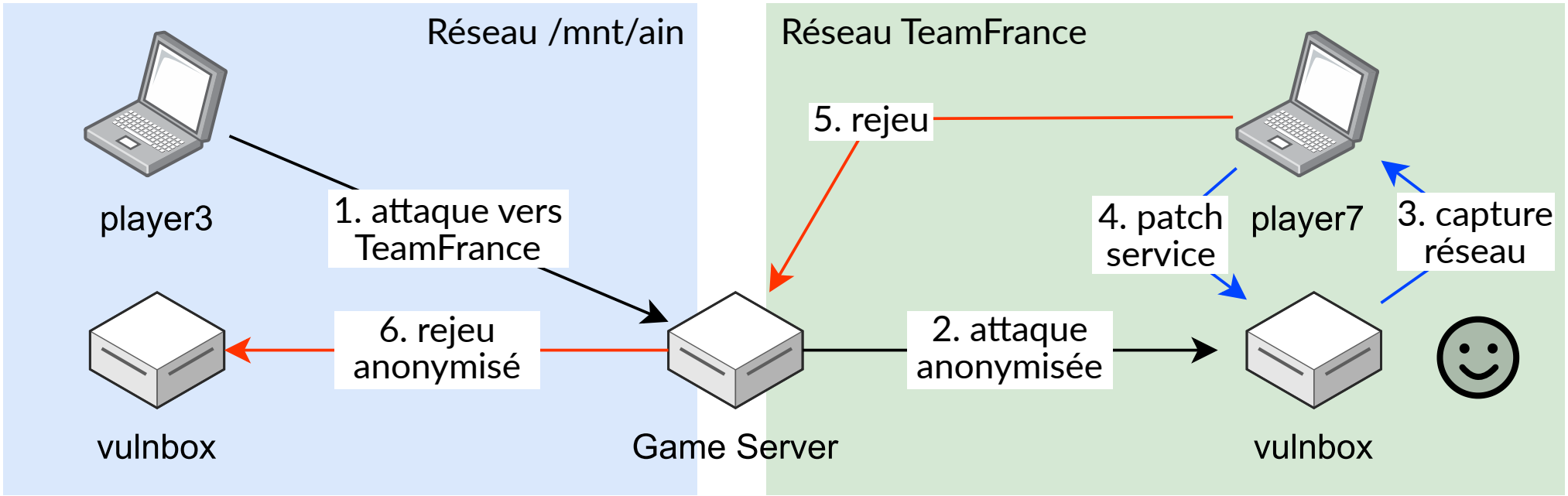
Quelques évènements publiques : FAUST CTF, ENOWARS, saarCTF, (La Nuit du Hack)



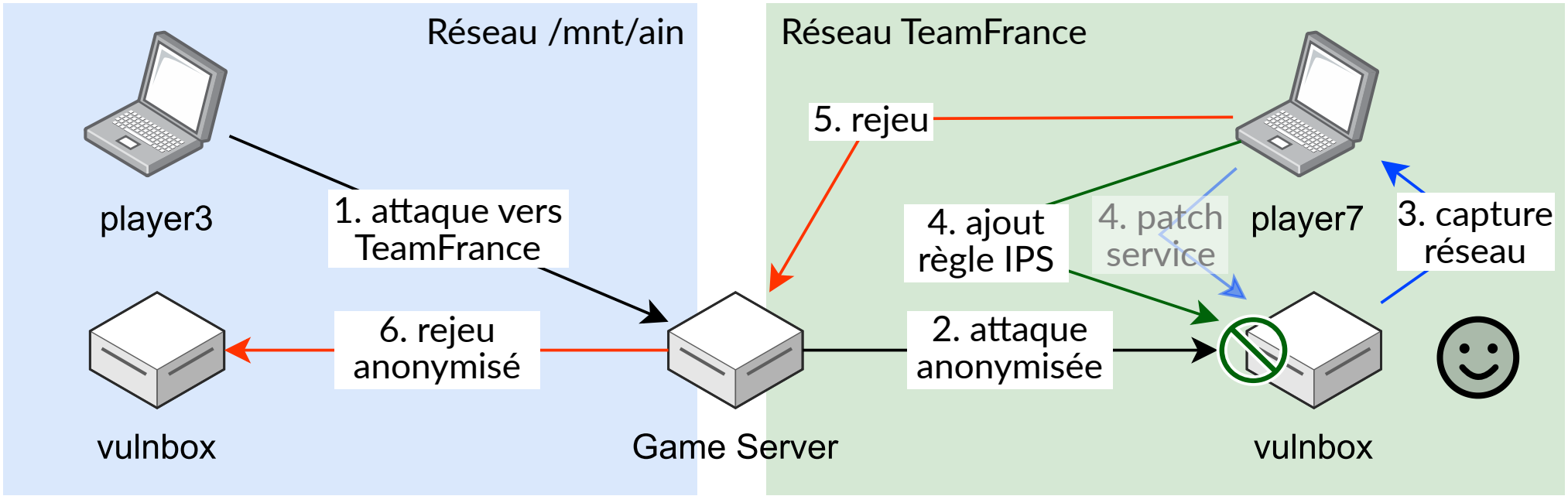
1. Pas de défense : perte de points.
2. Pas d'attaque.
3. Peut perdre de la disponibilité (e.g. exploit qui DoS).



1. Défense en patchant (Suricata IDS).
2. Pas d'attaque.
3. Perte de disponibilité (délicat de patcher).



1. Défense en patchant (Suricata IDS).
2. Attaque facile par rejeu.
3. Perte de disponibilité (délicat de patcher).



1. Défense par filtrage réseau (Suricata IPS).
2. Attaque facile par rejeu.
3. Disponibilité stable.

Démonstration

On perd des points, une équipe nous attaque !!

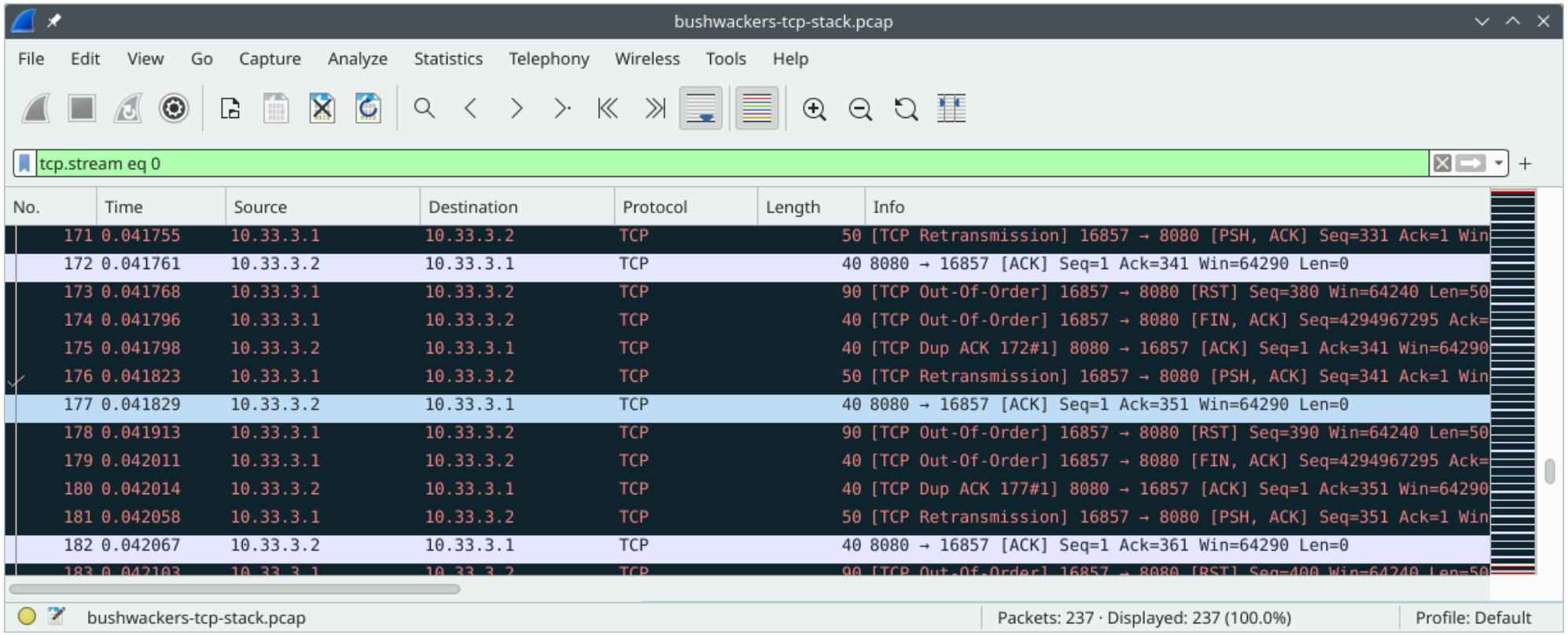
Comparaison aux autres outils

Pourquoi pas Wireshark ?

Pourquoi pas Wireshark ?

SYNACKTIV

Wireshark: analyse au niveau des paquets (trop verbeux !), reconstruction TCP fragile.



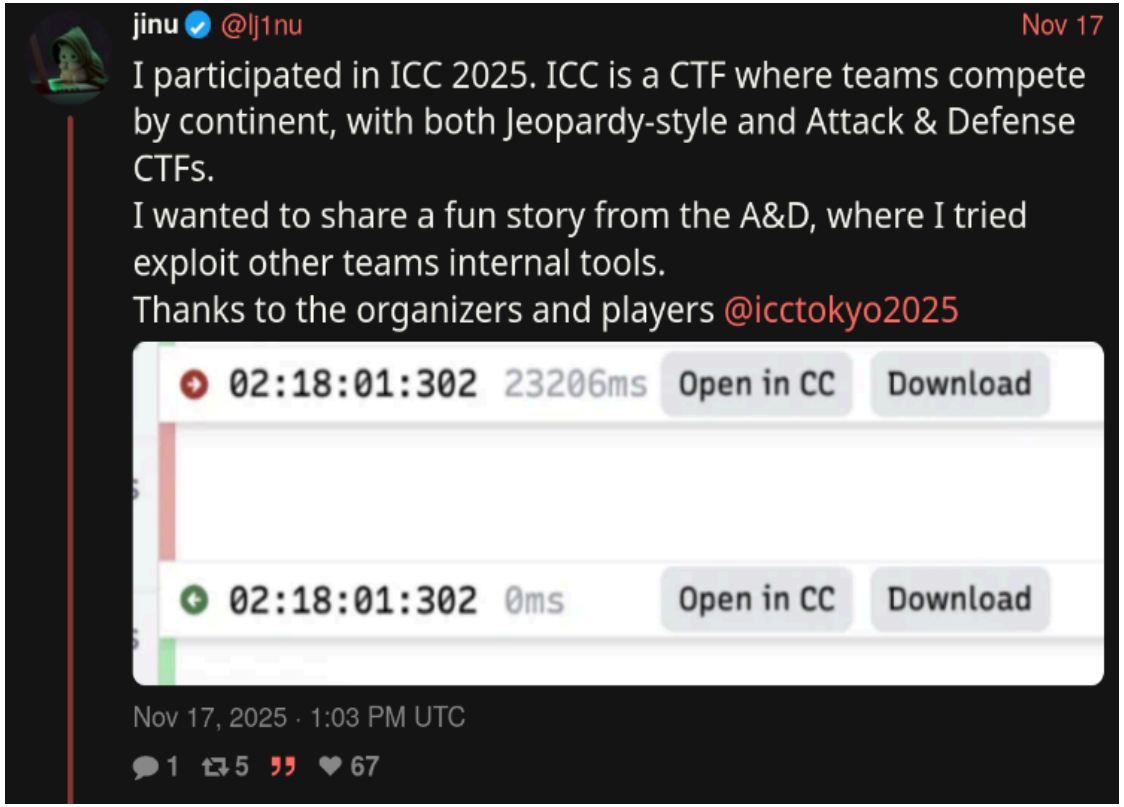
Pourquoi pas Tulip/Caronte/Flower ?

SYNACKTIV

- Flower (Ca' Foscari University)
  premier commit en 2018
- Caronte (TeamItaly)
  premier commit en 2020
- Tulip (TeamEurope)
  forked de Flower en mai 2022

Le suivi de flot est difficile à implémenter.

Plus robuste d'utiliser Suricata.



Suricata : journalisation Eve

SYNACKTIV

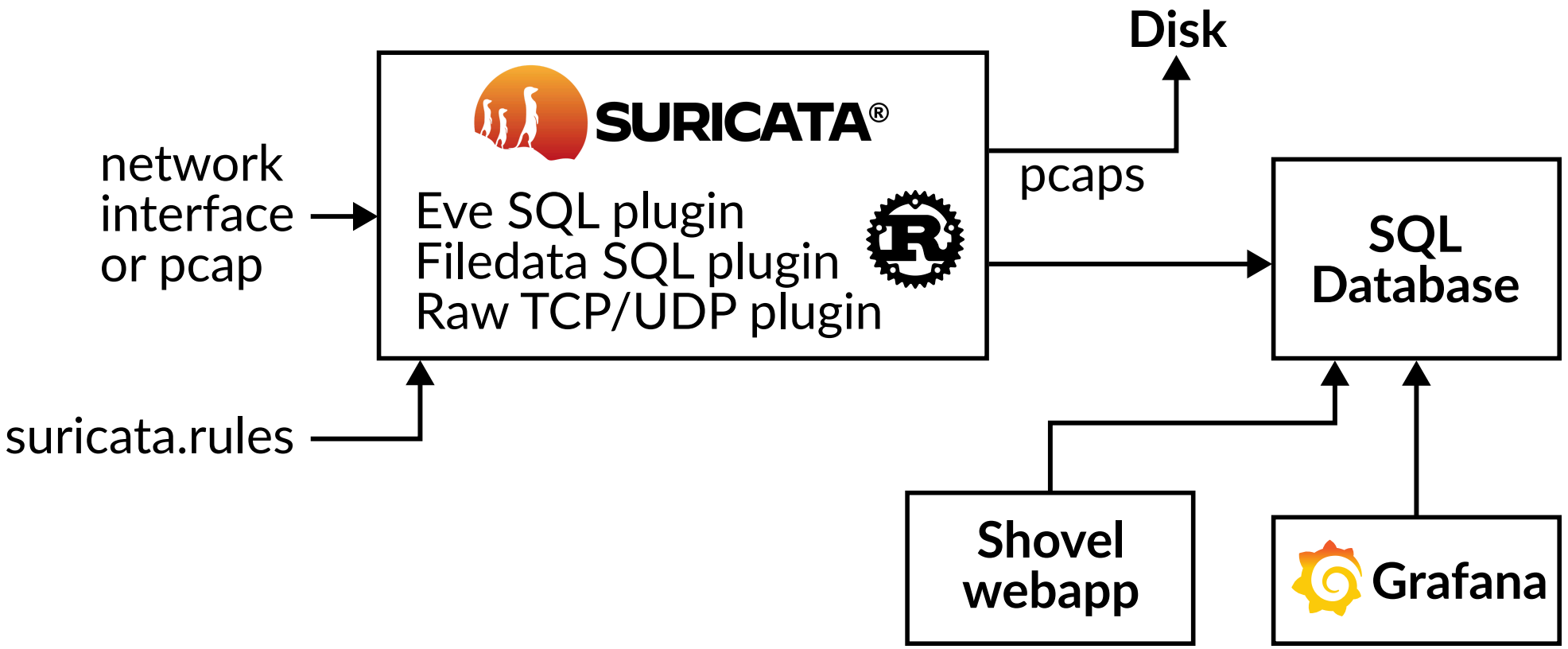

```
{"timestamp":"2024-11-30T13:41:20.371593+0000","flow_id":90877030211882,"event_type":"alert", ...}
{"timestamp":"2024-11-30T13:41:20.373418+0000","flow_id":90877030211882,"event_type":"http", ...}
{"timestamp":"2024-11-30T13:42:02.316193+0000","flow_id":1746857598270022,"event_type":"fileinfo", ...}
{"timestamp":"2024-11-30T13:42:02.316193+0000","flow_id":1746857598270022,"event_type":"http", ...}
{"timestamp":"2024-11-30T14:00:07.530540+0000","flow_id":90877030211882,"event_type":"flow", ...}
```


`eve.json` extract

Points forts : analyse par flot, beaucoup de dissecteurs.

Architecture : plugins Suricata + webapp

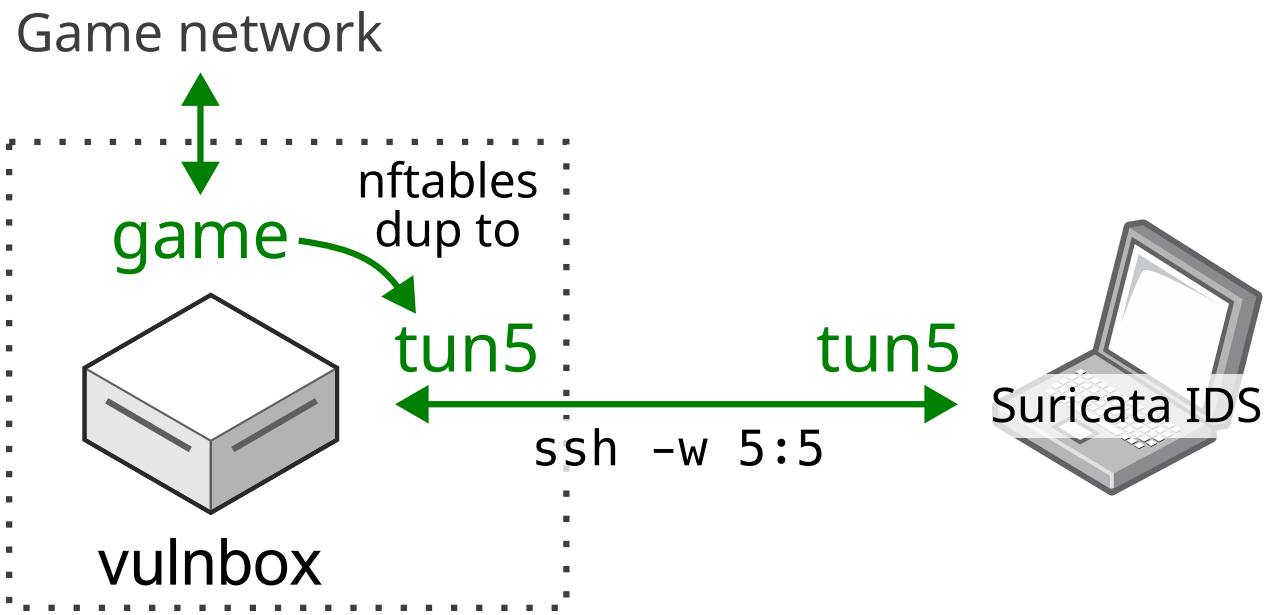
SYNACKTIV



Capture du trafic live via SSH

SYNACKTIV

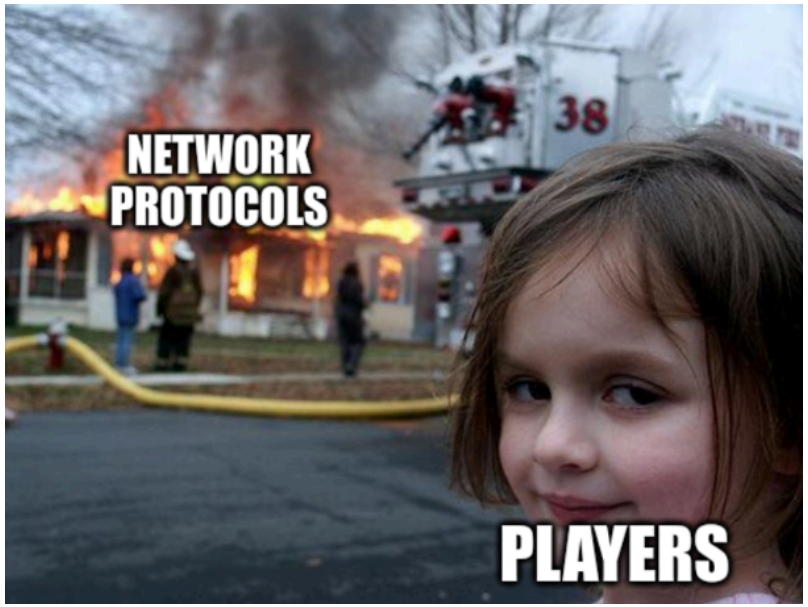
- Traffic mirroring plutôt que `tcpdump` + `rsync`.
- La *vulnbox* peut être compromise.
- Seulement accès SSH garanti.



Retombées sur le projet Suricata

SYNACKTIV

- Suricata plus accessible aux jeunes.
- Vulnérabilités trouvées et corrigées.
  - Évasion de détection.
  - Crashs investigués.
  - Anomalies de dissection HTTP.
- Betatesting plugin API Rust.



SYNACKTIV

Code source disponible sur https://github.com/FCSC-FR/shovel (GPL-2.0)

Testez l'outil sur https://shovel.elfique.org/

https://synacktiv.com

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

https://xcancel.com/synacktiv

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

Annexes

Anomalies HTTP trouvées

SYNACKTIV

Abnormal Content-Encoding header #7843

- `Content-Encoding: identity` in ASP.NET
- *SHOULD NOT be used in the Content-Encoding header -- RFC 2616, Section 3.5*

Response Header Repetition #7925

- Repeated `Set-Cookie` headers in Express, Django and Kestrel.
- Repeated `Vary` headers in Java Spring Boot.
- Repeated `Date` headers in Werkzeug.

Request Body Unexpected + Unable to Match Response to Request

- HTTP request confusion with `Content-Length` and `boundary`.

Suricata 8.0.0 segfault

SYNACKTIV

```
[216562 - W#15] 2025-07-19 18:46:57 Error: suricata: stacktrace:sig 11:PrefilterAddSidsResize+0x000000e2;...
```

The patch:

```
 uint64_t HSHashDb(const PatternDatabase *pd)
 {
-    uint64_t cached_hash = 0;
-    uint32_t *hash = (uint32_t *)(&cached_hash);
+    uint32_t hash[2] = { 0 };
     hashword2(&pd->pattern_cnt, 1, &hash[0], &hash[1]);
     for (uint32_t i = 0; i < pd->pattern_cnt; i++) {
         SCHSCachePatternHash(pd->parray[i], &hash[0], &hash[1]);
     }

-    return cached_hash;
+    return ((uint64_t)hash[1] << 32) | hash[0];
 }
```

Hints: Rust would have fixed the issue. CI/CD did not spot the crash.

Suricata 8.0.0 segfault

SYNACKTIV

The patch:

```
 uint64_t HSHashDb(const PatternDatabase *pd)
 {
-    uint64_t cached_hash = 0;
-    uint32_t *hash = (uint32_t *)(&cached_hash);
+    uint32_t hash[2] = { 0 };
     hashword2(&pd->pattern_cnt, 1, &hash[0], &hash[1]);
     for (uint32_t i = 0; i < pd->pattern_cnt; i++) {
         SCHSCachePatternHash(pd->parray[i], &hash[0], &hash[1]);
     }

-    return cached_hash;
+    return ((uint64_t)hash[1] << 32) | hash[0];
 }
```

> Hello! AFAIK the original code is not respecting Strict Aliasing, so it's an undefined behavior, and the compiler is free to do whatever it wants with it. So I believe the optimization is correct