RIPS: Robotics Intrusion Prevention System[1]

Enrique Soriano Salvador, Gorka Guardiola Múzquiz

GSYC, URJC

23 de mayo de 2024

[1] This work is funded by the Ministry of Science and Innovation of Spain, co-funded by the European Union, project PID2021-126592OB-C22 CASCAR/DMARCE.

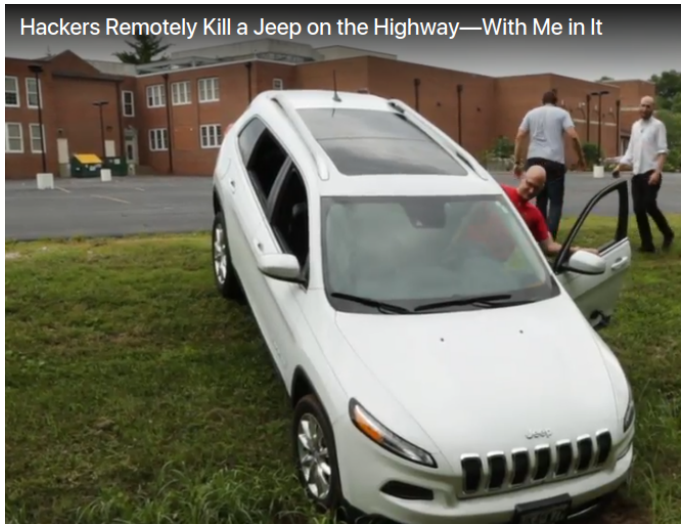
Robótica: ¿Ciberseguridad y seguridad?

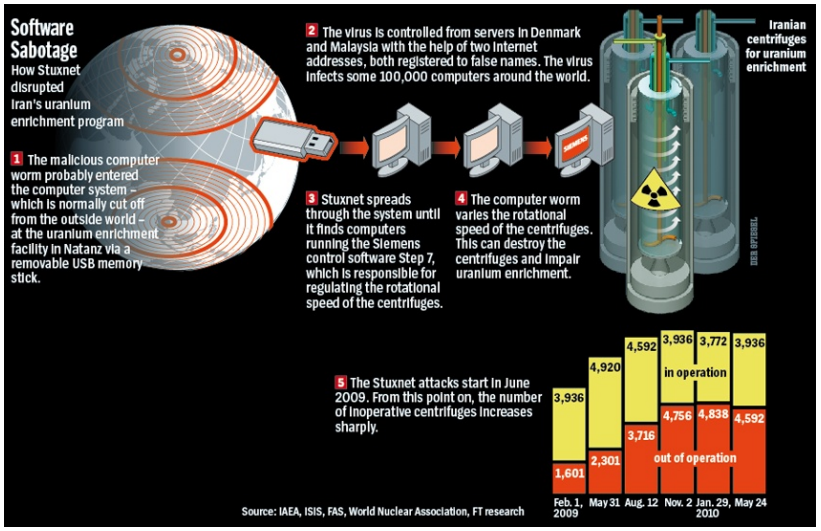
- "Seguridad" (Safety). Protección contra **fallos/accidentes** que ponen en peligro al entorno físico: daños a objetos, peligros para la salud, daños provocados a seres vivos, peligros medioambientales, etc.
- "Ciberseguridad" (Cybersecurity, Infosec, Security). Defensa ante **ciberataques** en sistemas y redes: ataques intencionados para dañar o robar datos, usar recursos de forma ilegítima, modificar el comportamiento del software o el hardware, violar la privacidad de las personas, etc.

$\neq$

¿Ciberseguridad y seguridad?

- Los robots autónomos, y en general todos los *sistemas ciber-físicos* (CPS), tienen que tener en cuenta las dos: los incidentes de ciberseguridad pueden desencadenar accidentes, fallos o acciones que provoquen daños físicos.





En robótica...

- **Safety**: siempre se ha tenido en cuenta para robots industriales, aplicaciones robóticas médicas, asistencia personal, entretenimiento, etc. Existen estándares como ISO 10218, ANSI R15.06-1999, ISO 13482, etc.
- **Security**: hasta hace pocos años, nadie se ha preocupado por la seguridad en los sistemas robóticos. Las principales herramientas para construir sistemas robóticos (p. ej. ROS) no proporcionaban mecanismos de seguridad (confidencialidad, autenticación, autorización, auditoría, etc.). Poco a poco, se van incorporando (p. ej. ROS 2).


Problemas

- Los IDS convencionales analizan las comunicaciones para detectar intrusiones y amenazas. Pueden analizar las capas de enlace, red, transporte y aplicación, pero **no pueden analizar aplicaciones robóticas**.
- Los IPS convencionales son capaces de prevenir ataques eliminando mensajes sospechosos de la red, configurando filtros en *firewalls*, emitiendo alarmas, etc. Aunque estas contramedidas y mitigaciones pueden ayudar en un entorno robótico, **no son suficientes**.

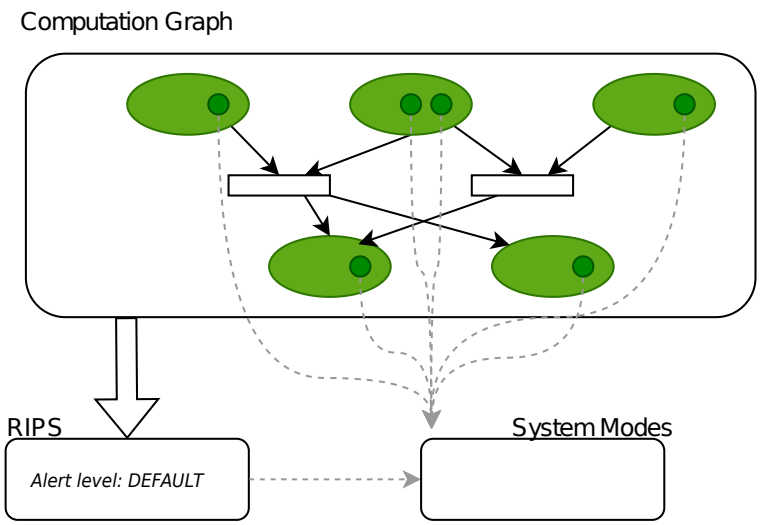
Objetivo: RIPS

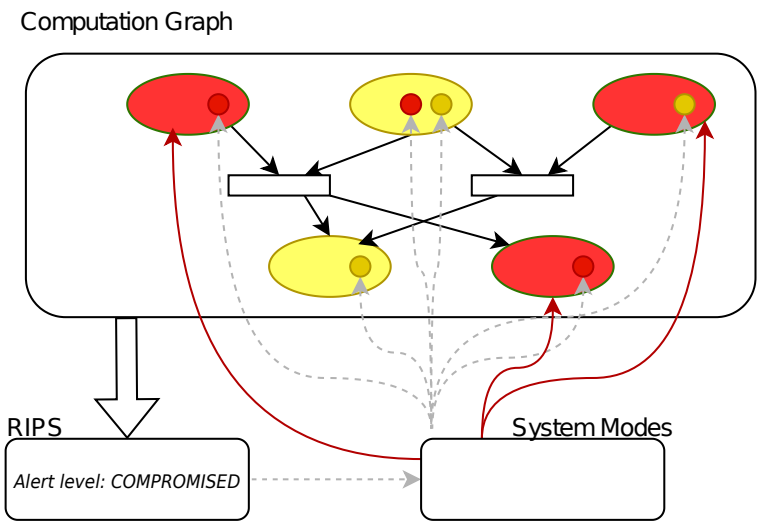
- Creación de un sistema **IDS (Intrusion Detection System) robótico** para sistemas ROS 2 (análisis de publicadores y subscriptores, tipos y subtipos de mensajes ROS 2, etc.).
- Creación de mecanismos de prevención de ataque que apliquen contramedidas y mitigaciones **fusionando Security+Safety**.

RIPS: ideas fundamentales

- Actúa a nivel de ROS 2: RIPS es un participante ROS 2 más.
- Inspecciona participantes y topics.
- Se subscribe a todos los topics que aparecen en el dominio y monitoriza los mensajes publicados, etc.
- Se puede integrar con un IDS de bajo nivel (p. ej. Snort).
- Define un lenguaje propio para definir las reglas de detección y prevención.
- Utiliza un framework de *safety* para poder reaccionar ante intrusiones: *system modes*.

RIPS: interacción con *system modes*





RIPS: ideas fundamentales

- El usuario definirá en el fichero de reglas un conjunto de niveles de alerta (que pueden ser mapeados a los niveles de los *system modes*).
- Por omisión, no se puede transitar de un nivel de alerta más severo a uno más suave sin la supervisión del administrador.
- RIPS es reactivo: reacciona a eventos detectados por su monitor:
  - Mensajes enviados a los topics.
  - Cambios en el grafo de computación.
  - Eventos externos (IDS, señales, etc.).

RIPS: ideas fundamentales

Una regla se compone de:

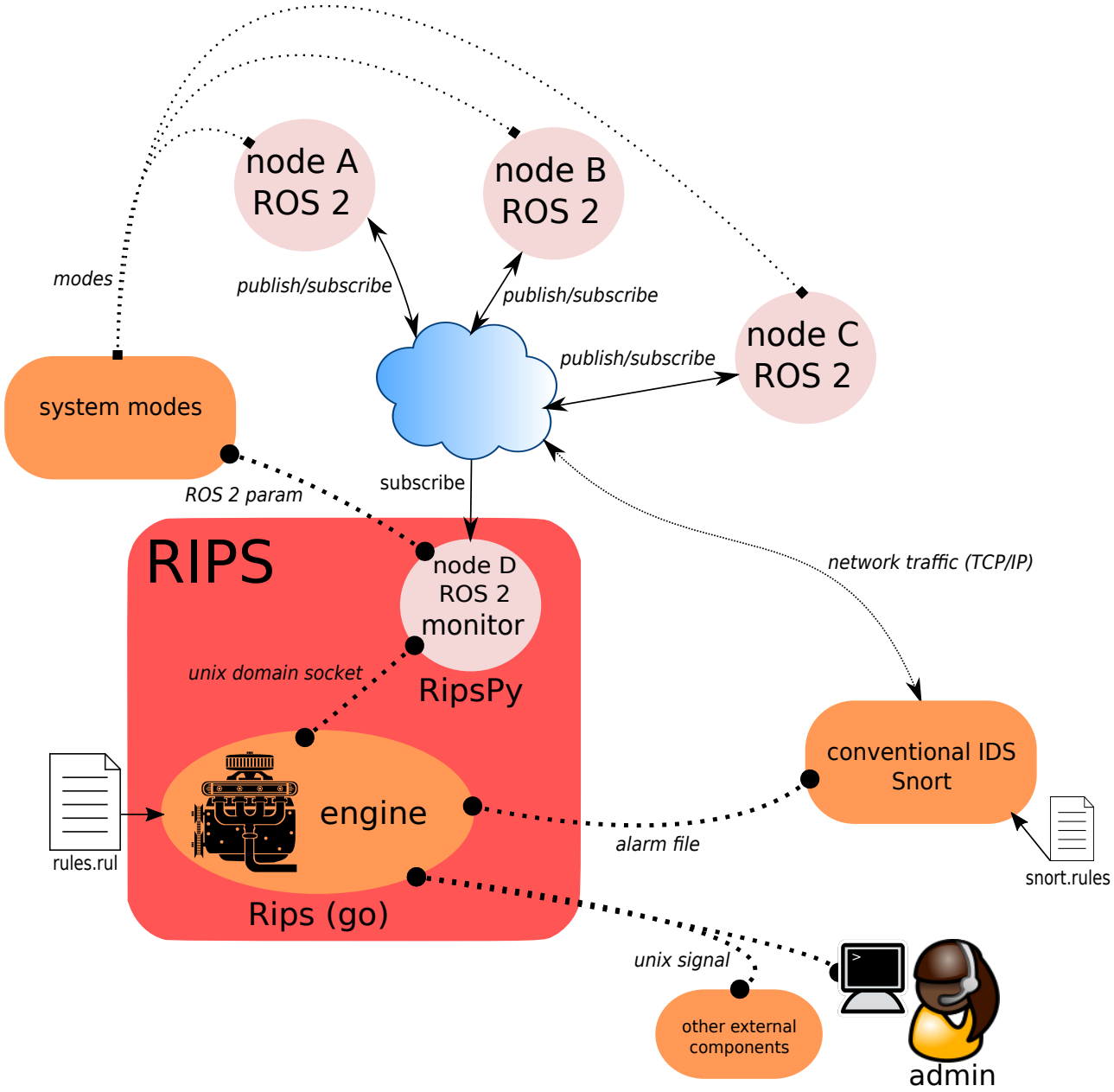
- **Nombre**: su identificador.
- **Expresión**: una expresión booleana que se evaluará cada vez que el motor recibe un cambio del contexto o un mensaje nuevo desde un topic. La expresión puede consultar los nodos, topics, subscriptores, publicadores, tipos de mensajes, etc.
- **Acciones**: una cadena de acciones que se ejecutarán cuando la expresión evalúa a *true*. Incluyen:
  - Emisión de alertas (sonoras, logs, alarmas, etc.).
  - Cambiar variables del sistema, que pueden ser usadas en las expresiones.
  - Ejecutar programas externos.
  - Transitar a un nivel de alerta nuevo con `trigger` (afectando a *system modes*).

RIPS: arquitectura

El primer prototipo de investigación está compuesto por los siguientes componentes:

- `RipsPy`: **monitor** de ROS 2, implementado como un nodo, que monitoriza los topics, participantes, etc. Está implementado en Python 3.
- `Rips`: **motor de detección** que interpreta un fichero de reglas escrito por el usuario y reacciona a los eventos proporcionados por el monitor. Está implementado en Go.

RIPS: arquitectura



Monitorización: RipsPy

- Clase Python 3 que hereda de `Node`.
- Configura un `timer` para hacer *polling* para detectar cambios en el grafo de computación (nodos descubiertos, topics, subscriptores de topics, etc.) usando métodos de `Node` como `get_node_names()`, `get_topic_names_and_types()`, etc.
- Mantiene una representación propia del grafo.
- Cuando detecta variaciones en el grafo, transmite el estado actual al motor.

Monitorización: RipsPy

- Cuando detecta un topic nuevo, se subscribe.
- Cuidado: hay que ignorar el topic `/rosout` para no entrar en un bucle de retroalimentación.
- Creamos una *inner function* como manejador para recibir del topic. La función manejará tanto el mensaje *deserializado* como el mensaje en crudo.
- Cada mensaje recibido se envía al motor.

Monitorización: RipsPy

```
...
params = self._context.params_of(topic)
...
msgtype = get_message(params[0])
def f(binmsg):
    pymsg = deserialize_message(binmsg, msgtype)
    self._update_context()
    self._send_msg_event(topic, message_to_yaml(pymsg), binmsg);

subscription = self.create_subscription(
    msgtype,
    topic,
    f,
    self.__QUEUE_DEPTH,
    raw = True
)
```

Monitorización: RipsPy

- La comunicación (bidireccional) con el motor es por un socket de dominio Unix. RipsPy dedica un thread para leer del socket.
- Se usa YAML para describir el contexto (grafo) y los mensajes recibidos.
- El motor comunica a RipsPy si se ha cambiado el nivel de RIPS. Cuando pasa esto, RipsPy cambia un parámetro que hace que reaccione: *system modes*.

Monitorización: RipsDash

- Dashboard en modo texto para monitorizar en vivo.
- Interpreta el mismo YAML que recibe el motor.

```
esoriano@blackbox: ~/prof/doc/prj/dmarce/ros_ws/src/ripspy/ripsdash

RIPSpy term dashboard                                          Tue Sep 19 10 35 15 2023
Current Level: __DEFAULT__

Topics
├─ ● /parameter_events
│  ├─ Publishers
│  │  ├─ corridorcamera
│  │  ├─ monitor
│  │  ├─ recorder
│  │  └─ rips
│  └─ Subscribers
│     ├─ corridorcamera
│     ├─ monitor
│     ├─ recorder
│     └─ rips
├─ ● /rosout
│  ├─ Publishers
│  │  ├─ corridorcamera
│  │  ├─ monitor
│  │  ├─ recorder
│  │  └─ rips
│  └─ Subscribers
├─ ● /videocorridor
│  ├─ Publishers
│  │  └─ corridorcamera
│  └─ Subscribers
│     ├─ monitor

Nodes:
├─ ● corridorcamera
│  └─ Services
│     ├─ /corridorcamera/describe_parameters
│     ├─ /corridorcamera/get_parameter_types
│     ├─ /corridorcamera/get_parameters
│     ├─ /corridorcamera/list_parameters
│     ├─ /corridorcamera/set_parameters
│     └─ /corridorcamera/set_parameters_atomically
├─ ● monitor
│  └─ Services
│     ├─ /monitor/describe_parameters
│     ├─ /monitor/get_parameter_types
│     ├─ /monitor/get_parameters
│     ├─ /monitor/list_parameters
│     ├─ /monitor/set_parameters
│     └─ /monitor/set_parameters_atomically
├─ ● recorder
│  └─ Services
│     ├─ /recorder/describe_parameters
│     ├─ /recorder/get_parameter_types
│     ├─ /recorder/get_parameters
│     ├─ /recorder/list_parameters
│     ├─ /recorder/set_parameters

msgs
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 337
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 338
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 339
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 340
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 341
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 342
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 343
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 344
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 345
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 346
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 347
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 348
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 349
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 350
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 351
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 352
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 353
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 354
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 355
✉ 10:35:15 msg to topic:/videocorridor String content: CORRIDOR CAMERA: SEQ 356

alerts
```

Motor: Rips

- Minilenguaje de reglas (parser descendente recursivo y Pratt parser, gramática limpia).
- Interpretado o compilado (generando Go que luego se compila).
- Tipado estricto y comprueba todo lo posible (Cuidadoso, la alerta cambia cosas. . .).
- Las reglas están en secciones que se corresponden con el evento que las dispara (llegada de un tipo de mensaje, evento externo. . .) y que se utiliza para el tipado.

Motor: Rips

Ejecuta:

- Las reglas correspondientes a cada mensaje que recibe del RipsPy (en YAML a través del socket).
  - El tipado te protege de programar reglas que miren campos de otro mensaje.
- Polling de alertas del IDS (fichero)/señales.

Contesta:

- YAML por el socket al RipsPy.
- Ejecuta comandos.

Ejemplo de fichero de reglas

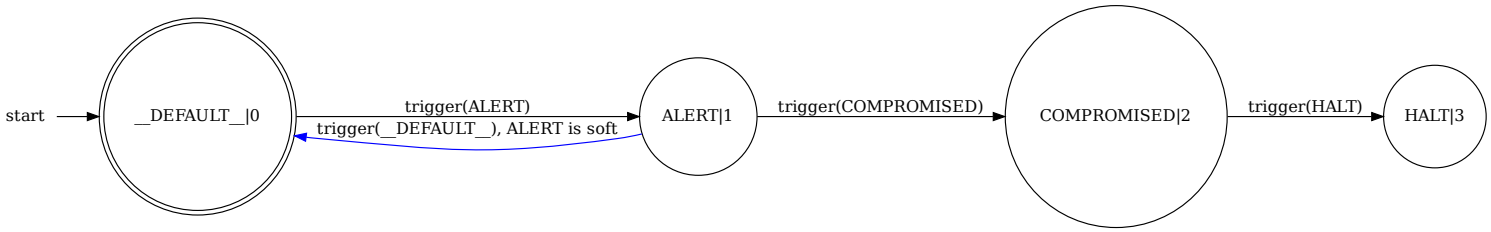
```
levels:
  __DEFAULT__;
  ALERT soft;
  COMPROMISED;
  HALT;

consts:
  MaxNodes int = 5; # rips and 4 participants

vars:
  descalated int = 0;

rules Graph:
   ! nodecount(1, MaxNodes) && CurrLevel == __DEFAULT__ ?
    alert("detected more than 4 nodes: too many nodes, entering level ALERT"),
    trigger(ALERT);

  nodecount(1, MaxNodes) && CurrLevel == ALERT ?
    set(descalated, descalated+1),
    alert("returning to default mode"),
    trigger(__DEFAULT__);

  descalated > 5 && (CurrLevel == ALERT || CurrLevel == __DEFAULT__)?
    alert("too many transitions to alert"),
    exec("/usr/bin/spd-say", "too many transitions to alert"),
    trigger(COMPROMISED);

  # should be: rips, monitor & recorder
  ! topicsubscribercount("/videocorridor", 0, 3) ?
```

Ejemplo de fichero de reglas (continúa...)

```
1        alert("videocorridor: too many subscribers"),
2        trigger(COMPROMISED);
3
4      # should be: corridorcamera
5      ! topicpublishercount("/videocorridor", 0, 1) ?
6        alert("videocorridor: too many publishers"),
7        trigger(COMPROMISED);
8
9      # should be: rips & recorder
10     ! topicsubscribercount("/videooffice", 0, 2) ?
11       alert("videooffice: too many subscribers"),
12       trigger(COMPROMISED);
13
14     # should be: officecamera
15     ! topicpublishercount("/videooffice", 0, 1) ?
16       alert("videooffice: too many publishers"),
17     trigger(COMPROMISED);
18
19   rules Msg:
20     topicmatches("/videocorridor") && !publishers("corridorcamera") && CurrLevel != HALT ?
21       alert("unauthorized publisher in corridorcamera"),
22       exec("/usr/bin/spd-say", "red code the system is totally compromised"),
23       exec("/usr/bin/spd-say", "halting the system"),
24       exec("/bin/sleep", "5"),
25       trigger(HALT);
```

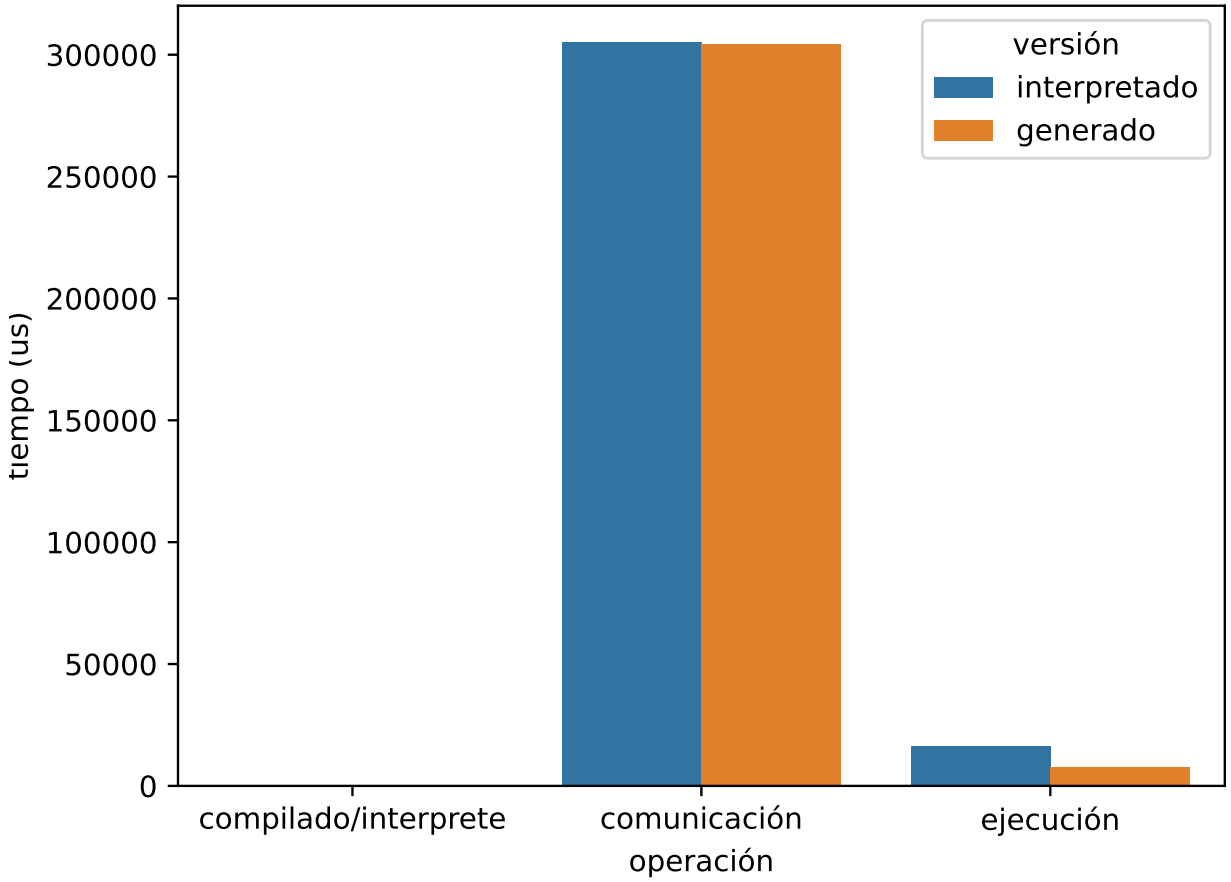
Estados

- Los cambios se disparan al ejecutar un `trigger` porque dispara una regla.
- Se pueden desescalar sólo los soft.
- Se comprueba que son alcanzables todos.
- Los cambios de estado se le comunican al RipsPy para que los ejecute.



Medidas iniciales

Casi todo el tiempo se consume en comunicación con RipsPy (decodificación mensajes, etc.).
Decodificación son centenares de ms. Ejecución decenas de ms.
Compilación/interpretado centenares de μs.



Medidas iniciales

- El compilador y el intérprete son muy muy rápidos (y compila expresiones regulares y reglas de Yara).
- Para hacer la ejecución más rápida, evitar el paso de mensajes: fusionarlo con RipsPy.

Trabajo futuro

- Pruebas en un entorno robótico real (robot TiaGO).
- Análisis del rendimiento.
- Dashboard gráfico.
- Implementación de producción en un programa monolítico (C++).
- ...