Offensive AI

A fun and not (so) explored field mixing **offensive cybersecurity** and **machine learning**.

Dorian BACHELOT

About Me!

Dorian Bachelot

Research Engineer

@LHS/CentraleSupélec/INRIA

#Malware Developer/Detection

https://dorianb.net

Table of Content

I. Introduction
1. Field Overview
2. Reminder on AI
3. Some Interesting Facts
4. LLMs...

II. OAI
1. Overview
2. The different subfields
3. Some History
4. Use Cases

III. OAI for C2 Piloting
1. C2 Piloting / Heuristics
2. C2 Piloting / RL
3. C2 Piloting / Modelisation

IV. OAI for Evasion
1. Offline Evasion
2. Online Evasion
3. Meta-optimization
4. Surrogate Model
5. Surrogate Model: Evasion
6. Surrogate Model: Visualization
7. Conclusion

V. Conclusion
1. OAI is larger!
2. Thanks!

Introduction

AI, ML, DL, OAI...

Field Overview

Definition

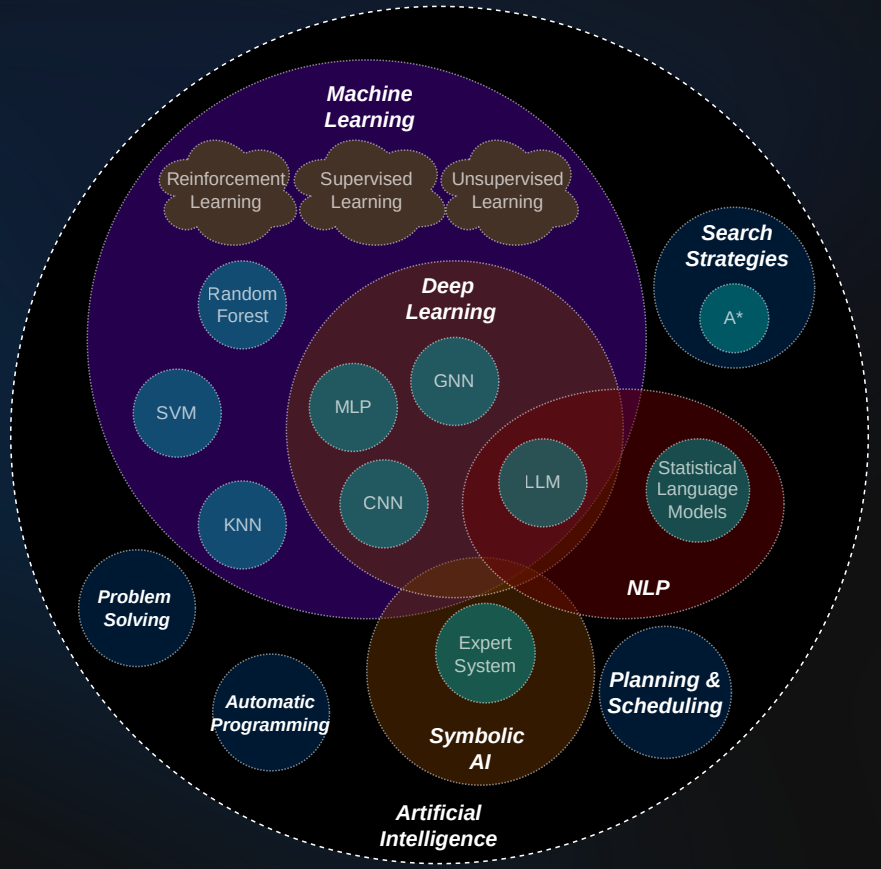
Offensive artificial intelligence (OAI) can be defined as the field in which artificial intelligence algorithms or models are used to **automate one or multiple parts of an offensive security task**.

This cover:

- Vulnerability scanning
- Exploit automation
- Attack path identification
- ...

Reminder on AI

We can classify as AI any system that solves a problem or achieve a goal and have some degree of adaptation and autonomy.

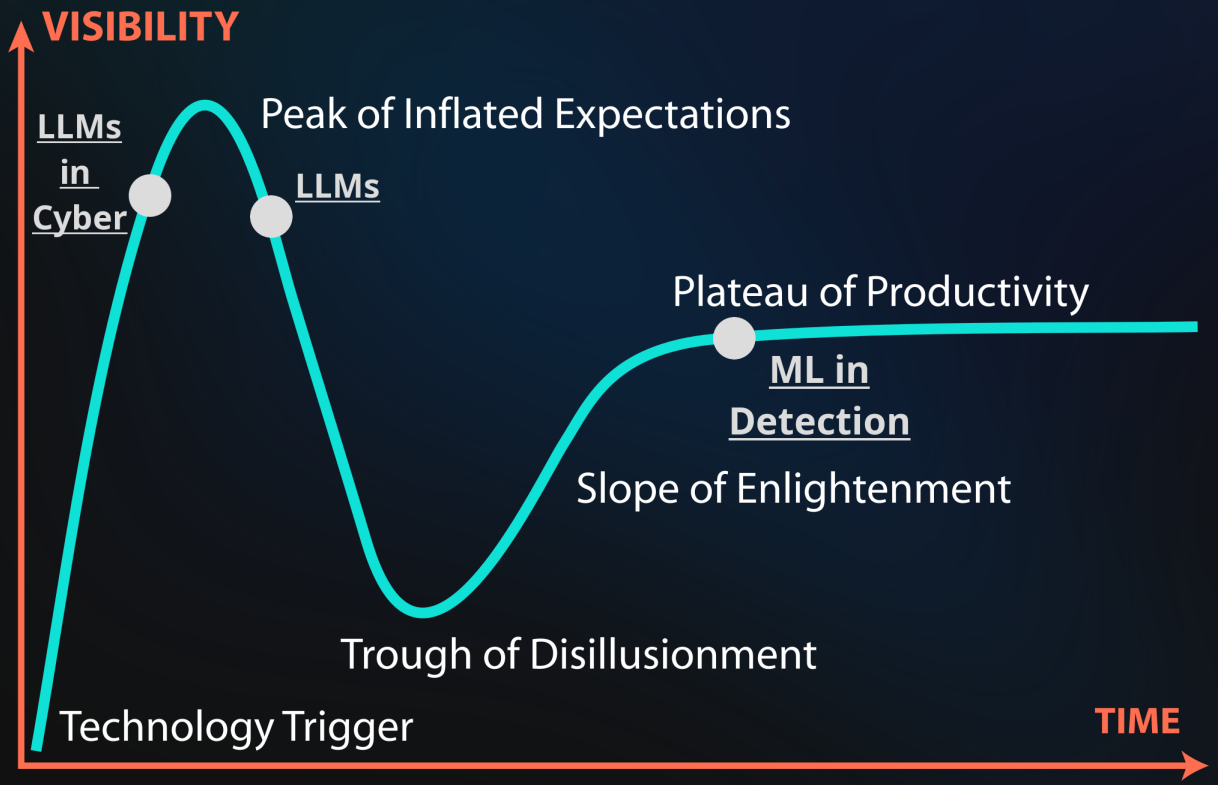


Machine Learning ≠ AI ≠ Deep Learning ≠ LLMs

Some Interesting Facts

- Majority of vulnerability scanners are using **AI but not ML**.
- YARA is an **Expert System**.
- Suricata is an **Expert System**.
- EDRs are mostly using **Expert Systems** (like Elastic EDR rule engine).
- ML models are subject to **adversarial attacks**, hence they have their own class of vulnerabilities.
- ...

LLMs...


VISIBILITY
LLMs in Cyber
Peak of Inflated Expectations
LLMs
Plateau of Productivity
ML in Detection
Slope of Enlightenment
Trough of Disillusionment
Technology Trigger
TIME

OAI

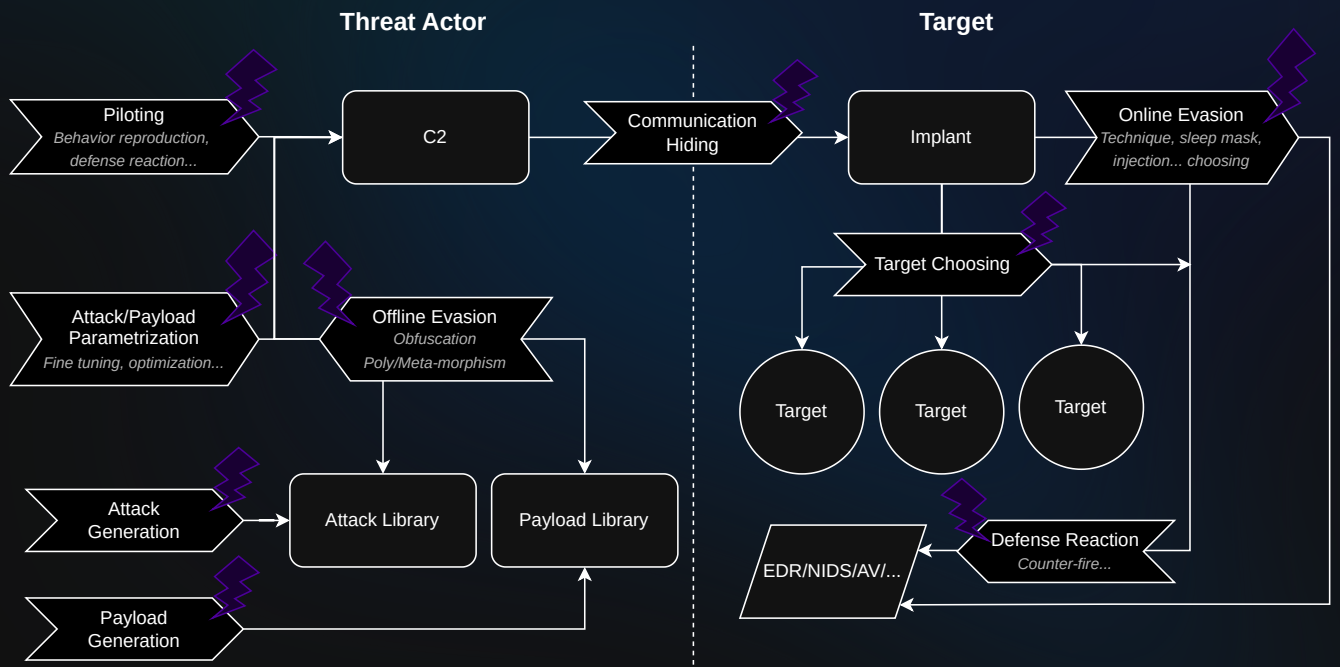
Global overview

Overview

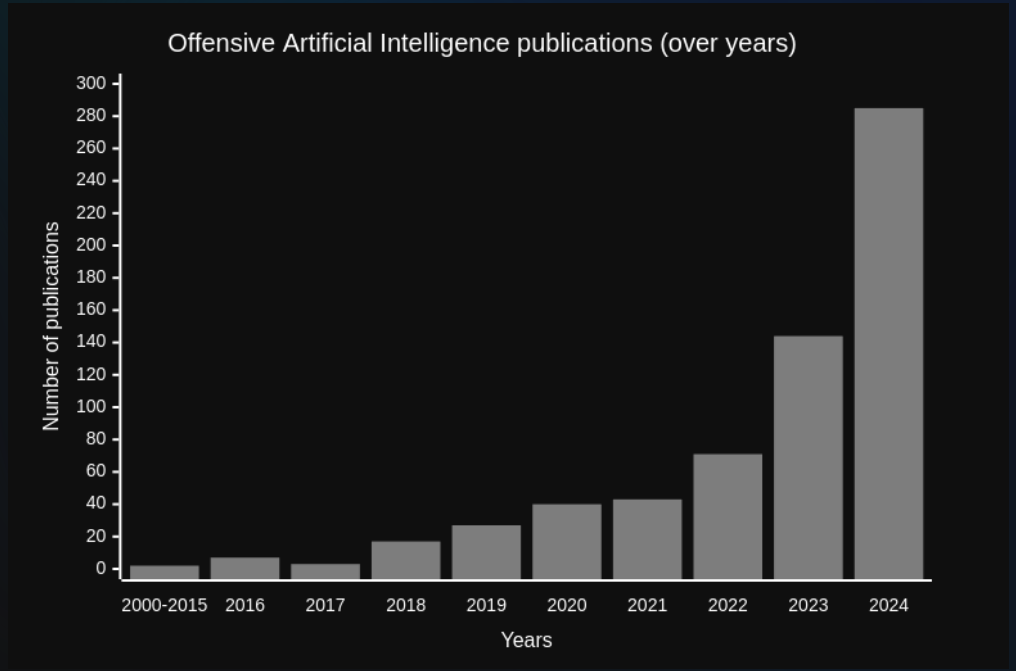
OAI can also be cut in **multiple subfields**, which correlate closely to some **use cases like pentest or red team automation**.

Automation ≠ Better than Humans

The different subfields



Some History

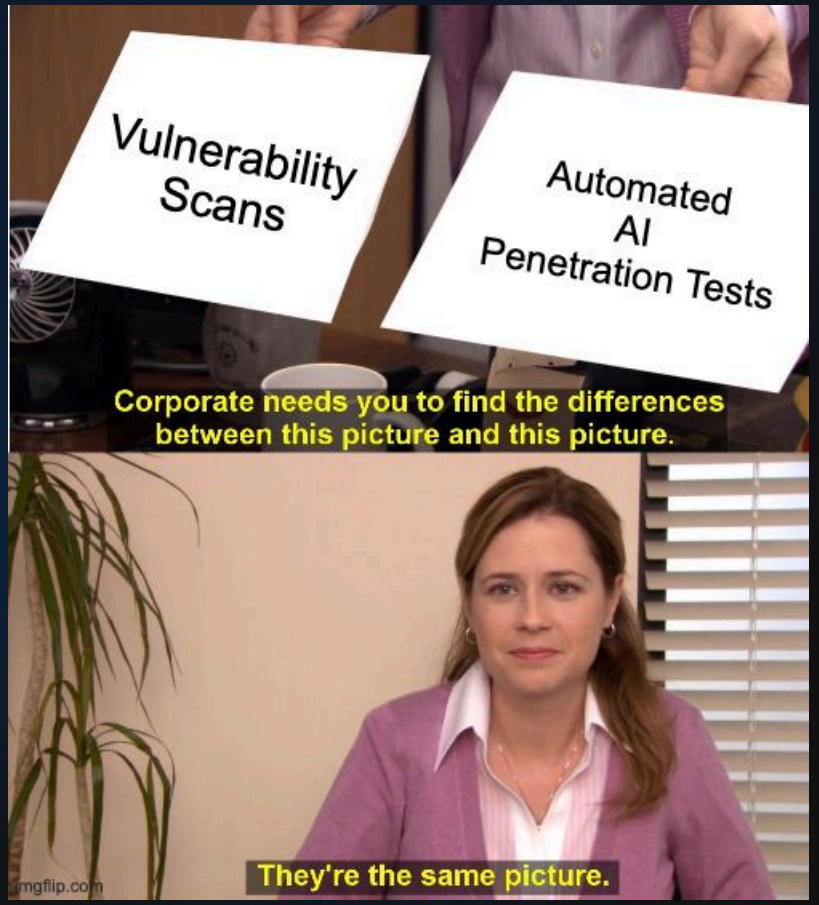


OAI is **not a new field**, but only got a "boom" recently **with LLMs** (easier prototyping).

Use Cases

Non-exhaustive list

- (*Partially*) automated pentest tools.
- (*Partially*) automated red team tools.
- Automated training systems (for Blue Team).
- New malware variants/techniques prediction.
- Auditing detection systems (EDRs).
- ...



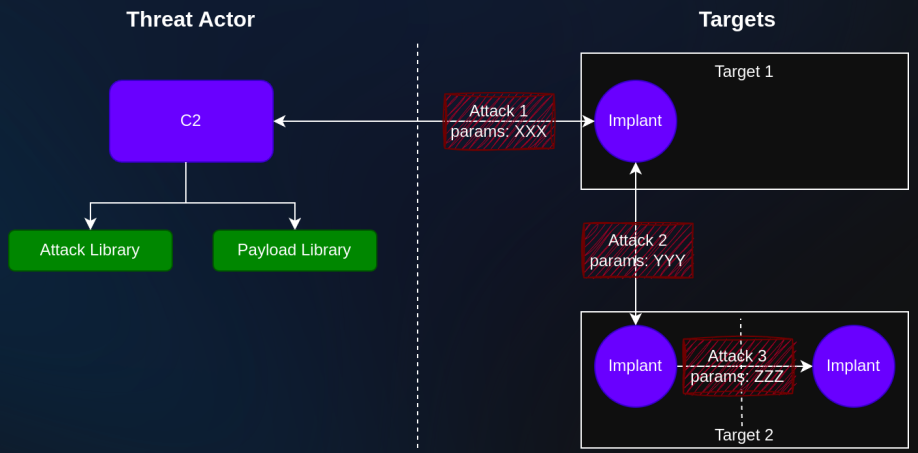
OAI for C2 Piloting

Target choosing, attack choosing...

C2 Piloting

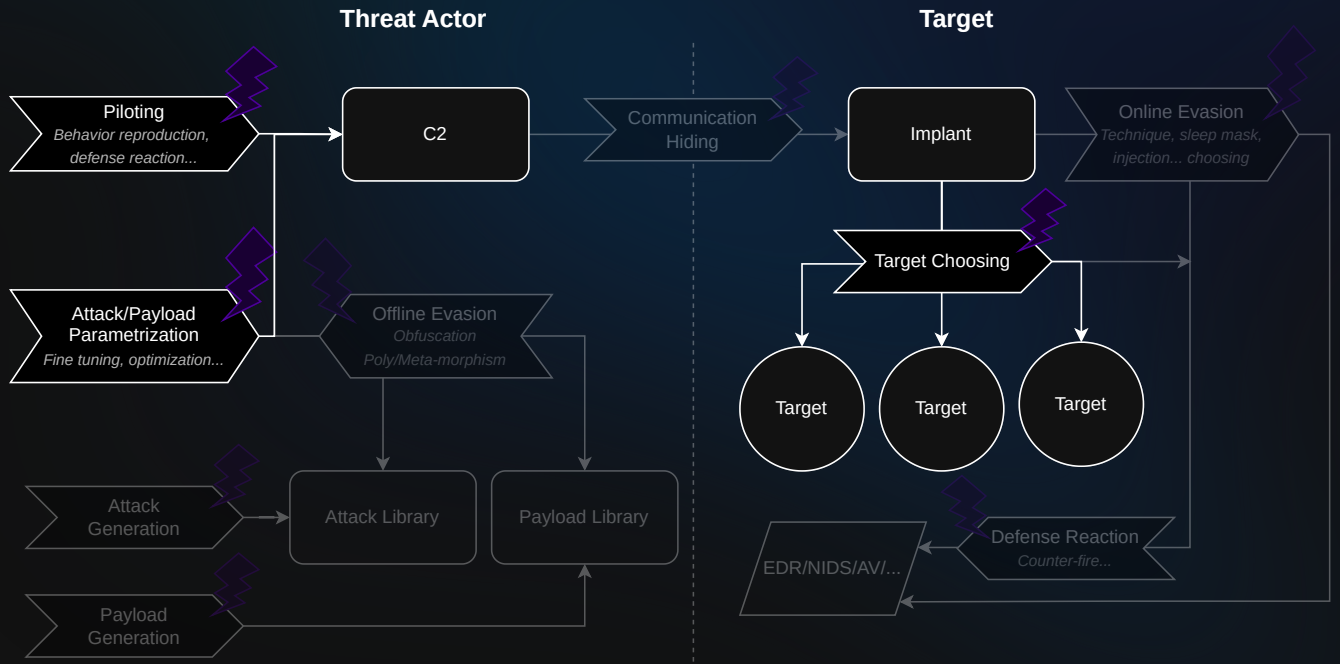
What is a C2?

- Pilot implants, that have infected some hosts.
- Implants can communicate directly with the C2, or can do it through another implant.
- Can issue commands, and send attacks (modules).
- Focus on evasion, hiding communication and using advanced implants.
- Used to pivot and deeply infect a network.
- Controlled by an operator (or AI!).



C2 Piloting / The Idea

Consists of selecting a target, an attack and parameters using a model/algorithm.



C2 Piloting / Possibilities

A lot of features can be augmented by AI just on the C2 piloting part:

- **Automated** solution, no human needed.
- **Speed of execution** (time until SOC can see a system log can be in minutes!).
- **Multiple path** (~networks) explored in parallel.
- **Counterfires** (e.g., making noise on a part of the network).
- Attack **parameters optimization**.
- Past context **change monitoring** (e.g., change in network).
- Actions **logging/reproducibility**.
- ...

C2 Piloting / What's Needed?

Recipe:

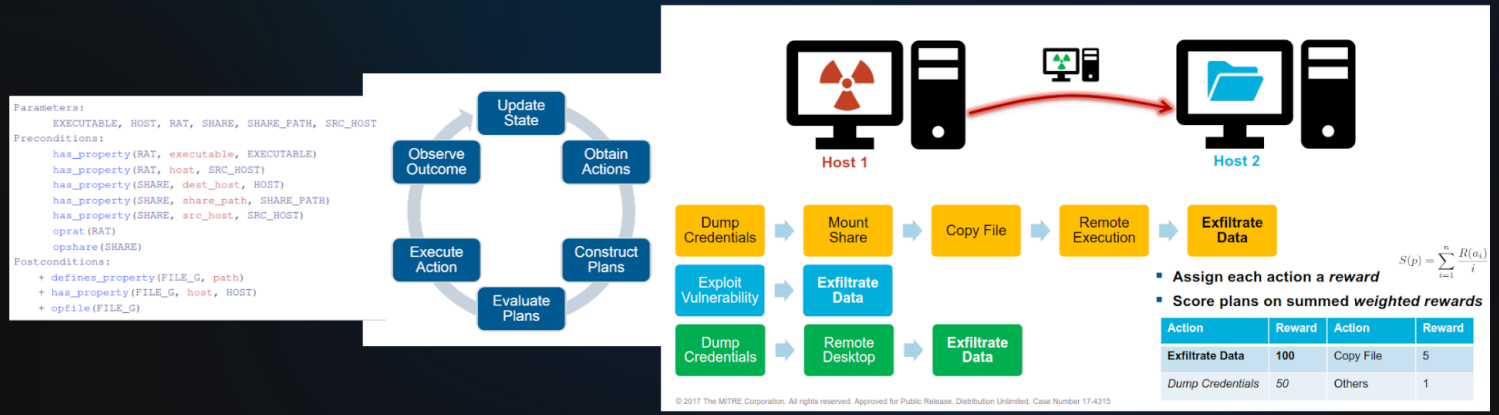
- Take a **C2 with an API** (giving automation power on ALL operations).
- Prepare an **attack library** (modules), with exposed parameters.
- Add some model/algorithm.
- Shake. Done!

(@chrisrohlf)

Not that simple in reality... A lot of AI-related issues.

C2 Piloting / Heuristics

Caldera[1], an automated framework for adversary emulation (by Mitre), rely on a **heuristic** to select attacks & targets.



Based on **pre & post conditions**, really basic and not very "smart" system.

C2 Piloting / Heuristics

When **preconditions** are met, will select the best action path (P) by using the **path reward function** $S(p)$:

$$p = \{a_0, a_1, ..., a_n\} \tag{1}$$

$$S(p) = \sum_{i=1}^{n} \frac{R(a_i)}{i} \tag{2}$$

$$P = \underset{p}{\operatorname{argmax}}\, S(p) \tag{3}$$

Where R is a function defining associated reward values per action and a_k an action in action space A.

This is close to using a finite horizon over a **Markov Decision Process (MDP)**.

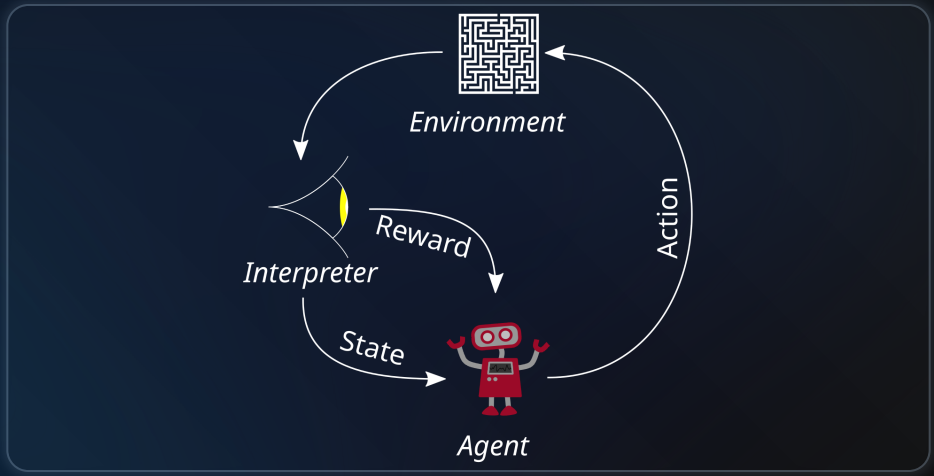
It can be difficult to model **complex behavior** with this approach.

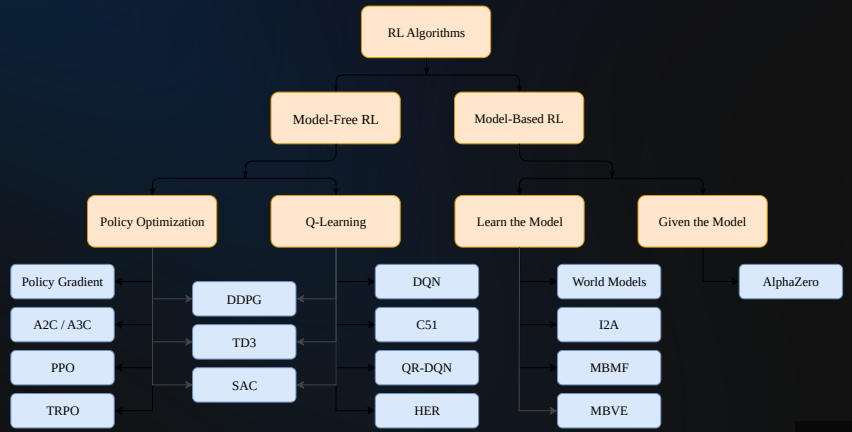
C2 Piloting / RL

Another approach used by the literature is Reinforcement Learning (ML).

- S is the set of all states,
- A is the set of all actions,
- $R : S \times A \times S \to \mathbb{R}$ is the reward function,
- For a step k, $r_k = R(s_k, a_k, s_{k+1})$ is the associated reward function,

Reinforcement learning is vast:





(OpenAI)

C2 Piloting / RL for Piloting

RL fit well for C2 piloting:

- S contains all network states (like machines, OSes, open ports...),
- A contains all the modules in the attack library (times possible parameters?!),
- R is possible to construct, has demonstrated with Caldera's heuristic (not that easy...).

Playing with R make it possible to complexify model behavior (e.g., valorize exploration vs exploitation, take defense into consideration...)

C2 Piloting / RL Environment

First issue, to use RL you need a training environment that'll simulate (mock) inputs/outputs [2][3]. This is called a simulation environment (or Gym).

Multiple environments are available, but are globally oriented toward Deep Learning, making them **very** basic:

Address	Compr.	Reach.	Disc.	Value	Access	linux	windows	proftpd	drupal	phpwiki	e_search	wp_ninja	mysql
(1, 0)	True	True	True	0.0	1.0	False	False	False	False	False	True	True	False
(3, 1)	False	True	True	0.0	0.0	False	False	False	False	False	False	False	False
(3, 0)	True	True	True	100.0	2.0	False	False	True	True	True	False	False	True
(4, 1)	False	True	True	0.0	0.0	False	False	False	False	False	False	False	False
(4, 0)	True	True	True	0.0	1.0	False	False	True	False	True	False	False	False

(3)

Most, if not all, are **oversimplifying** the problem, and some doesn't even provide realistic evaluation (emulation, or running in a real IS). For example, none allow **attack parametrisation**. And most are using a probability of failure per action (artificial realism).

C2 Piloting / Some RL Environments

Name	Release Date	Features
NASim	2019	Simulation only
CyberBattleSim	2021	Simulation only, Microsoft
CybORG (CAGE)	2022	Simulation & some level of emulation (~Metasploit C2)
NASimEmu	2023	Simulation & Emulation (Metasploit C2)

Writing such environment takes time since the **level of abstraction is very high**. There is also a **scaling issue**.

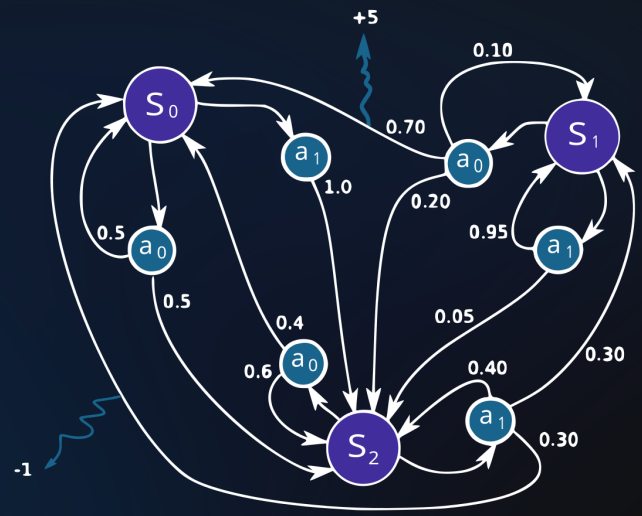
Metasploit is the **ONLY** used C2 across the literature.

C2 Piloting / Modelisation

Basic reinforcement learning is modeled as a Markov Decision Process (MDP), assuming that the agent will directly (& fully) observes the environment state.

- $P : S \times A \to \mathcal{P}(S)$ is the transition probability function,
- $P(s'|s,a)$ is the probability of transitioning into state s' from state s with action a.
- $\pi : S \times A \to [0,1], \pi(s,a) = P(a|s)$ is called a policy, used to choose an action a at a state s.

Aims is to find an optimal policy $\pi^*(s) = \text{argmax}_a\, Q^*(s,a)$ maximizing cumulative rewards.



(waldoalvarez)

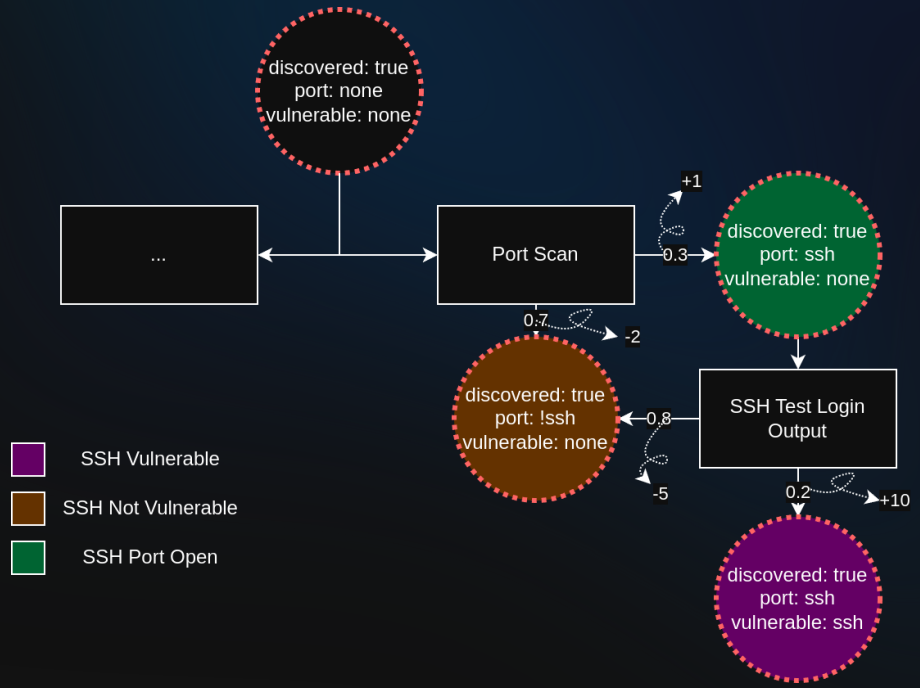
$$V^*(s) = \max_a \mathbb{E}_{s' \sim P}[R(s,a) + \gamma V^*(s')]$$
$$= \max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right]$$

$$Q^*(s,a) = \mathbb{E}_{s' \sim P}\left[R(s,a) + \gamma \max_{a'} Q^*(s',a') \right]$$
$$= R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$$

(Bellman equations)

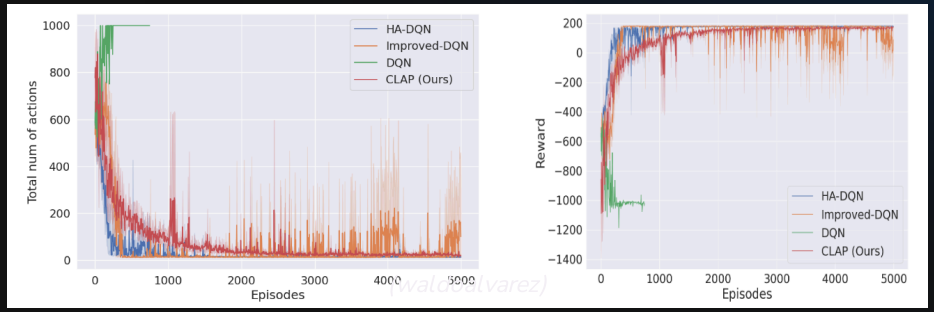
C2 Piloting / Modelisation

Bellman equations are for optimal solution, ML models try to estimate that function



C2 Piloting / Some RL Agents

- **Q-Learning**: learning an approximator $Q_\theta(s, a)$ of the optimal $Q^*(s, a)$ function. Example method: DQN.
- **Policy Optimization**: optimize a parameter θ (e.g., gradient ascent) for a policy $\pi_\theta(a|s)$. Example method: PPO.



Model	Technique	Year
HA-DRL[4]	Algebraic Action Decomposition	2021
NDSPI-DQN [5]	Q-Learning	2021
CLAP[6]	Policy Optimization	2022

Multiple issues with these approaches and the environment used (NASim):

- No generalization (no realism/real usage),
- Questionable hypothesis (fully observable),
- Imperfect reward function R/policy π,
- No attack parametrisation,
- Questionable performances.

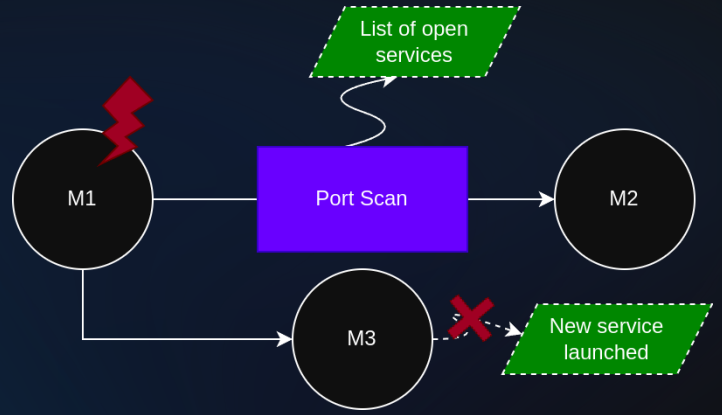
C2 Piloting / Naive Modelisation

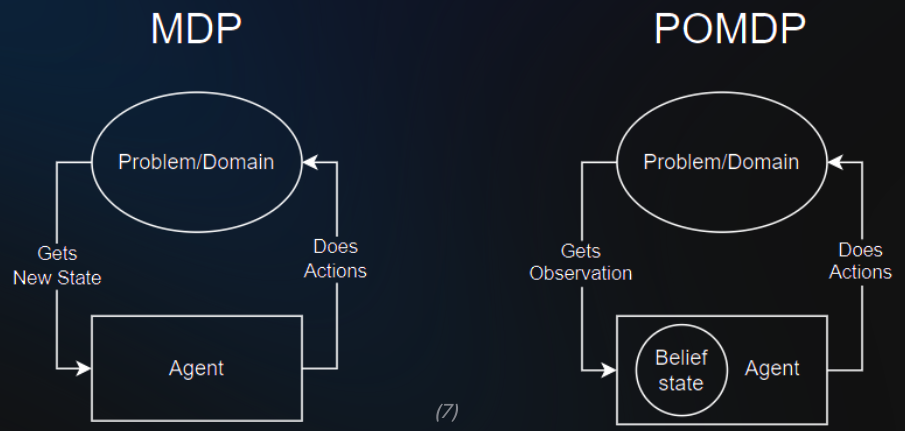
"assuming that the agent will directly (& fully) observes the environment state.": this is naive.

A network is dynamic and continuous, seeing it at fully observable is wrong. Hence, MDP is not the correct model.

This is model-free vs model-based problematic.

We need to introduce the notion of partial observations to the model, and therefore models the problem using POMDP (Partially Observable Markov Decision Process).





(7)

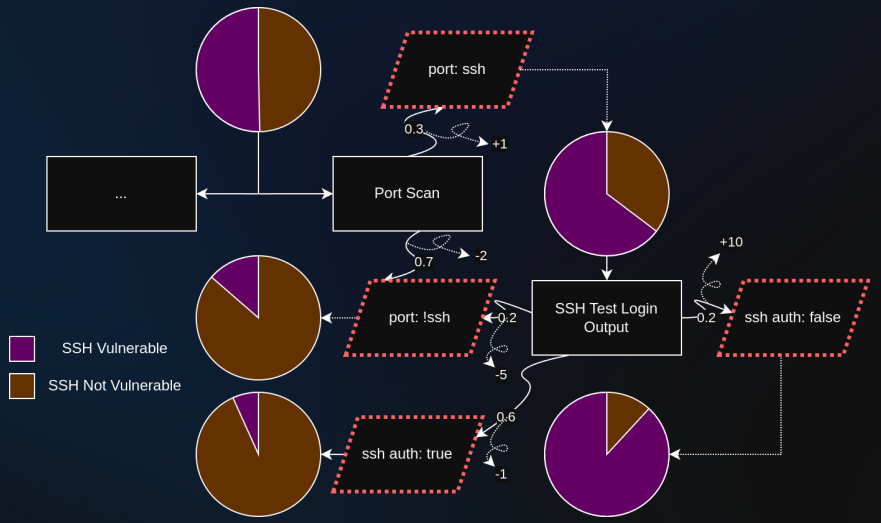
C2 Piloting / POMDP

Model-based solutions need to maintain a model of the environment, and update it regarding observations o.

- $O : S \times \Omega \to [0, 1]$ is a set of conditional observation probabilities, with $p(o|s) = O(s, o)$.
- b is the belief state, a posterior probability distribution over S.

Actions are chosen based on the belief state b (updated with observations o). After having selected an action a at state s, an observation o is "received" with the probability $O(o|s', a)$.

The new state s' is not known, and can only be estimated using the belief state b. $b(s)$ is the probability to be in state s.



Updating b to b' (knowing b, a and o / η is a normalizing constant):

$$\begin{aligned} b'(s') &= P(s'|o, a, b) \\ &= \eta O(o|s', a) \sum_{s \in S} P(s'|s, a) b(s) \end{aligned}$$

C2 Piloting / More RL Agents

Model	Technique	Complex Behavior	Parametrisation	Generalization	Year
POMDP[8]	POMDP	No	No	No	2013
LD-PenTesting [9]	POMDP	~Defender Behaviour	No	No	2020
EPPTA[10]	POMDP + PPO	No	No	No	2023
AutoRed [11]	POMDP	No	No	GNN	2024

Multiple issues with these approaches :

- Questionable generalization (except AutoRed),
- Imperfect reward function R/policy π (complexity),
- No attack parametrisation,
- Questionable performances,
- Low quality/high abstraction environments.

C2 Piloting / A Complex Problem

Multiple questions:

- **One or multiple** models to select the target, the attack and the parameters?
- Does the target selection model **see the whole network**, or just select the target with the best attack?
- Since a network can have a (virtually) **infinite number of machines** (targets), how to construct the input/context for the target selection model?
- How to create an adequate **reward function/policy**?
- How to capture the **complexity** of the problem (like taking into account defense reaction)?
- C2 Piloting is a **planning problem** (with goals & subgoals)?
- On a big network, with a lot of possible actions, each with many parameters, how **not to collapse**?

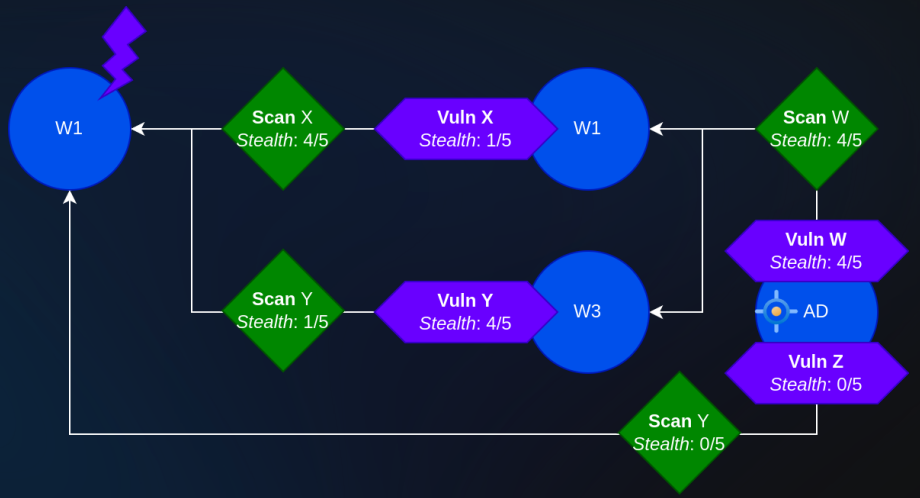
C2 Piloting / Reward & Policy

The reward functions R & policies π used are generally focusing on rewarding **shorter attack path** or big **number of compromised machines**. They are generally just a static table.

This is **naive** in the context of Red Team automation.

For Red Team, we want to be **stealth** and **evade** defenses.

There is an **exploration/exploitation** balance to optimize, and this is **not trivial**.



Example reward R in [5] & [2] (where $H \in S$ is the set of compromised hosts, and A the set of actions taken):

$$R = \sum_{h \in H} \text{value}(h) - \sum_{a \in A} \text{cost}(a)$$

This is not enough to capture **complex behaviors**! **Planning** and having **goals/subgoals** is a clue.

C2 Piloting / Generalization

Generalization is about training on a specific scenario (varying vulnerabilities & paths), and being able to **run on others** (e.g., with a bigger network).

As seen, many (close to all) agents **can't generalize**, and need a **retraining per network topology**. Some are giving up on the target selection, and focus on **attack selection per host** (mono-target).

POMDP helps since O can be "**network topology agnostic**".

Generalization is also very difficult regarding **parametrisation**.

C2 Piloting / Parametrisation

Complete action space $A^* = \{(a, p) \,|\, a \in A \,\text{and}\, p \in P_a\}$ where P_a is the set containing all parameters for action a and $P = \{P_{a_1}, P_{a_2}, ..., P_{a_n}\}$ is the set containing all parameters for all actions. This can be **huge**!

Some parameters have a common format/purpose (e.g., target IP, process to target, etc.) but are still linked to the associated attack.

How to avoid having **one model** per action/parameter? And how to **handle training** data/environment?

Currently, the best and "easiest" choice for parametrisation is to **use a heuristic**.

C2 Piloting / New Approach

Proposed approach

Issue	Solution	Description
Generalization	**GNN / Carrousel**	Ingest the network context independently from its size. Learn to retain only interesting information. Context-based ingestion.
Partially Observable	**POMDP + Dreamer** (*JEPA?*)	Internal environment representation, latent space reasoning.
Complex Behavior	**Complex R & Goals**	Complexify the reward function. Add goals & subgoals to complexify policies (planning).
C2 Automation	**New Specific C2**	Create a C2 with automation in mind.
RL Environment	**New Emulation First RL Environment**	Create a new RL environment centered around emulation (maximum realism).

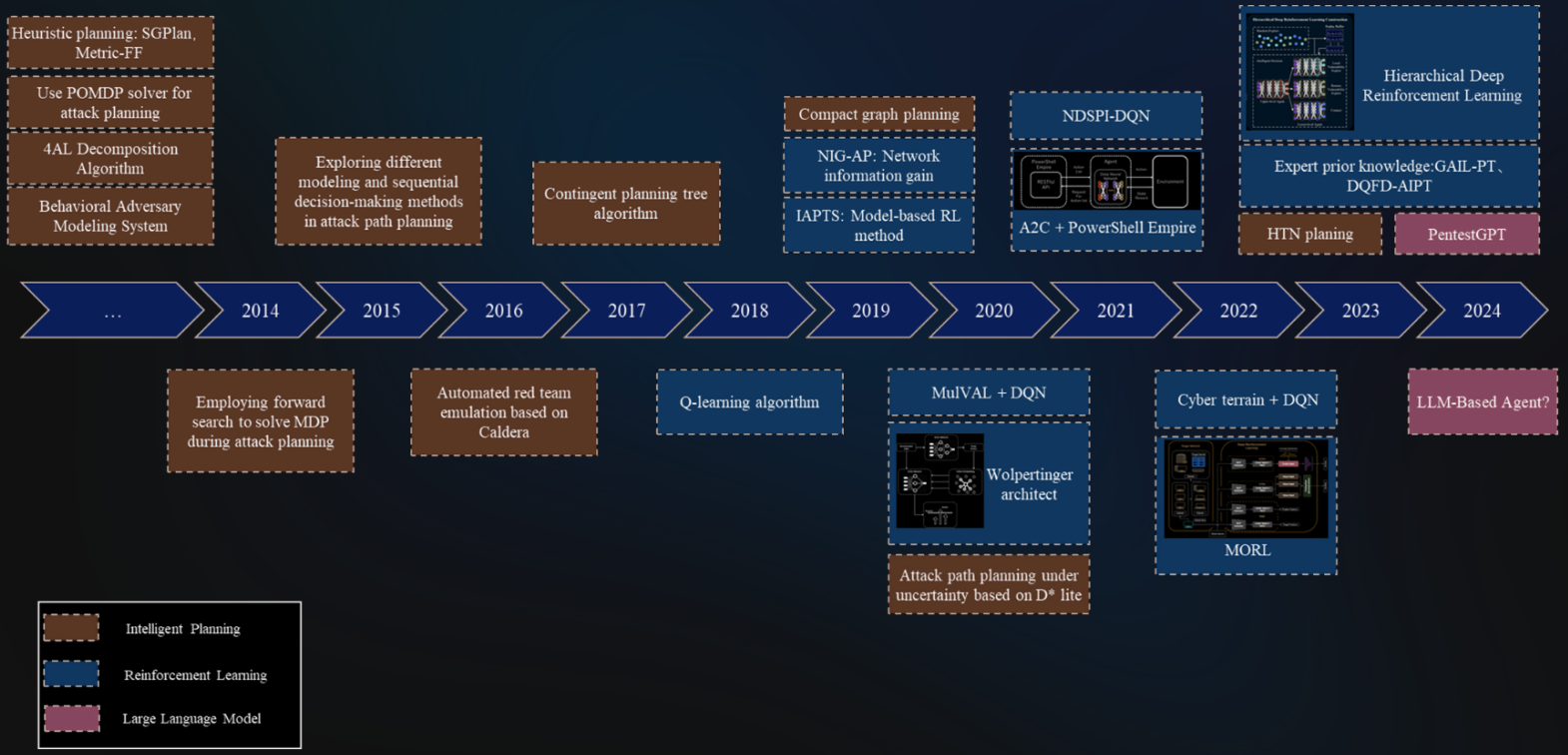
C2 Piloting / Other Approaches

Other approaches exist:

- Some people are trying to use **LLMs for C2 piloting** (me! [12]), but also end-to-end OAI. Results are not great, OAI is not really **hallucination-compatible** (e.g., commands needs to be exact, IPs also, impact if the wrong target...).
- RL is not the only learning method: **Self-Supervised Learning** is trending.
- **Heuristics** are not dead.
- **Expert systems** (analogy to the defensive side) should also be explored.

The dream is to have **end-to-end OAI** (target, attack, parameters choosing with attack generation).

C2 Piloting / Overview

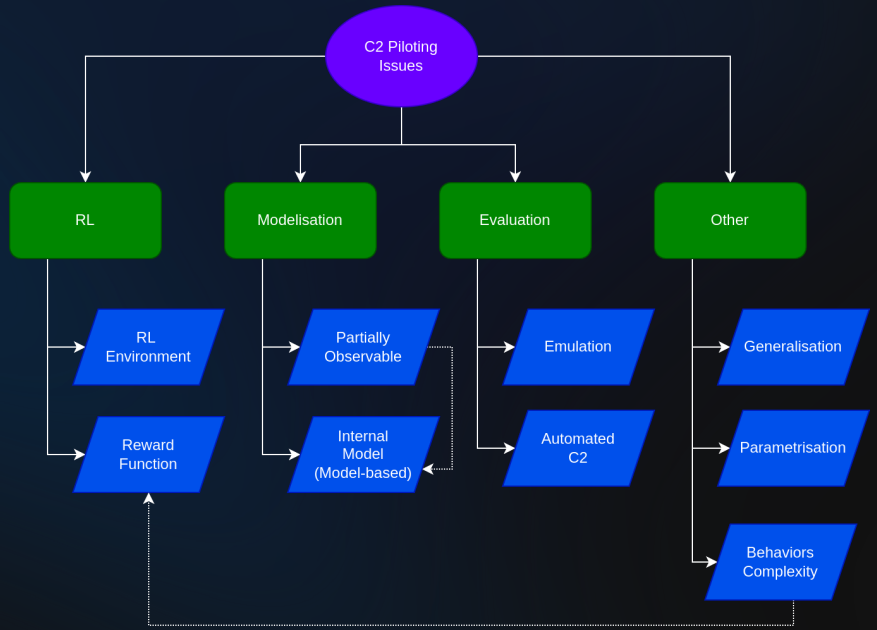


(13)

C2 Piloting / Conclusion

Some "Hot Takes":

- C2 Piloting is **hard**.
- This is not just a **model choice issue**.
- Past literature (minus some exception) are **focusing on the wrong problematic**.
- Linked **use cases** are huge.
- Ask many **fundamental** questions.
- When it will work, the **impact might be important** for defenses.
- (*LLMs are not the solution*)

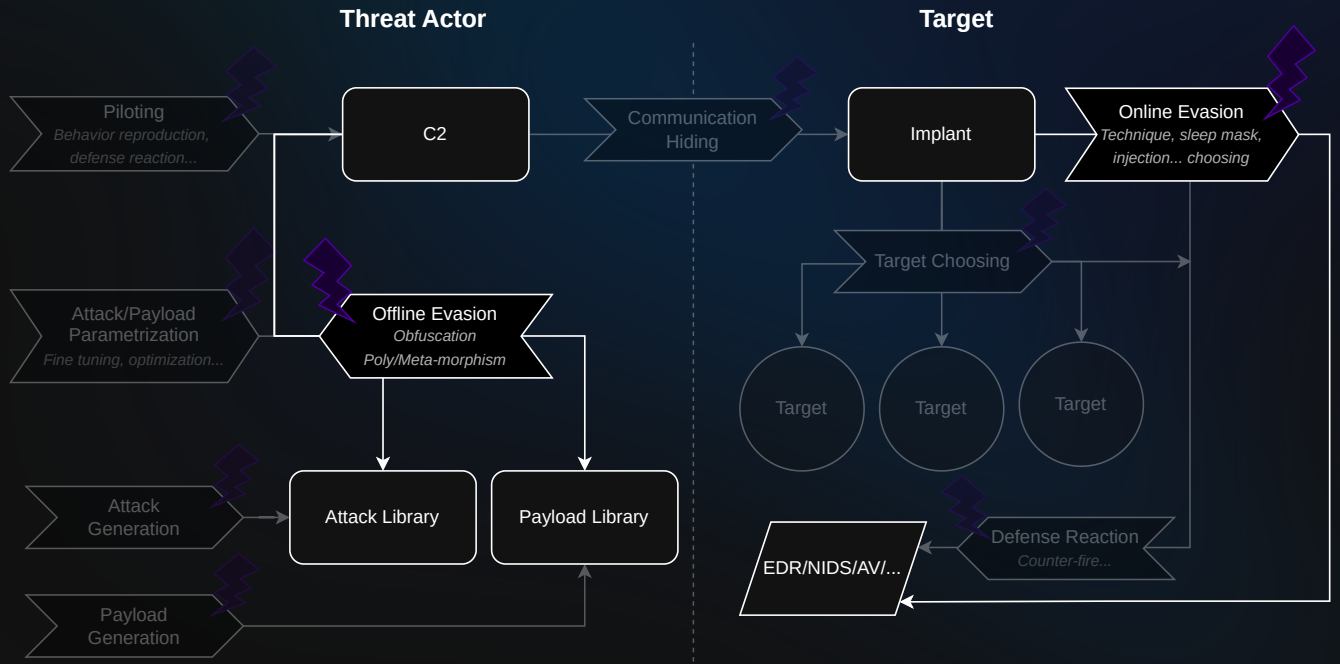


OAI for Evasion

Online & Offline evasion...

Evasion

Consists of **optimizing** (e.g., by selecting parameters) an evasion method.

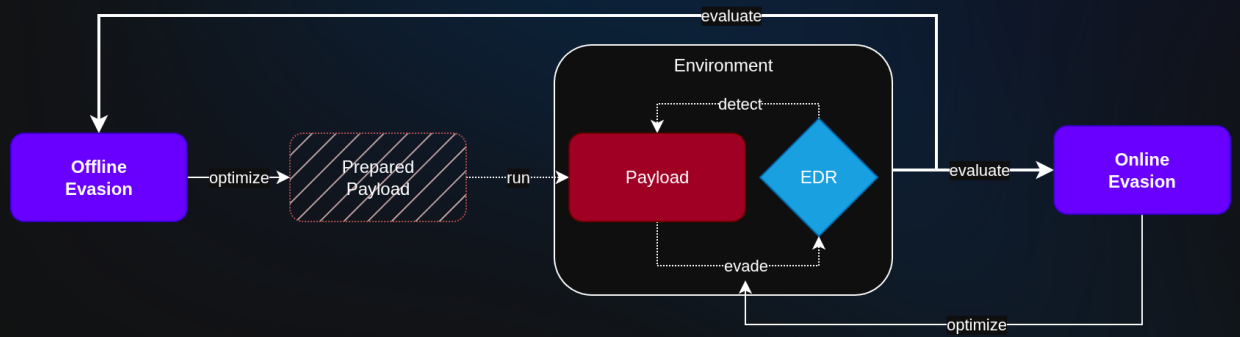


Evasion / The Idea

You want to **bypass defenses** (and here, specifically host based sensors, like EDRs).

Many evasion methods are available, but they generally have **parameters**.

Each sensor is generally **more vulnerable** to specific combination of evasion techniques, associated with specific parameters.



AI can be used to find these combinations and associated parameters, **optimizing evasion**.

Offline Evasion

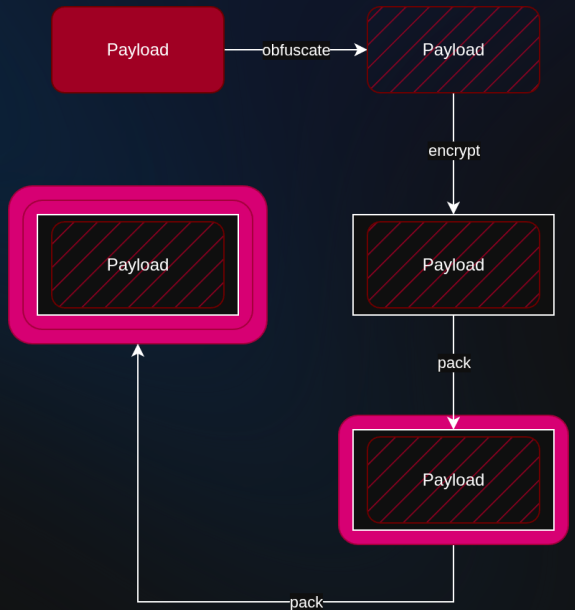
Offline evasion includes any method that reduces detection & evade defenses **statically** (e.g., modifying a payload before sending it). We want to **obfuscate a payload** not to be detected by static analysis.

This cover:

- Not having the same hash,
- Obfuscating code,
- Hidding strings,
- Packing,
- ...

Some techniques:

- Polymorphism
- Metamorphism
- Obfuscation (excluding *morphism)
- ...



Offline Evasion / *morphism

Objective: Complexify part of a binary to make detection and/or reverse engineering **harder**. This maintains same functionalities, but make **analysis more difficult**.

Automation Surface:

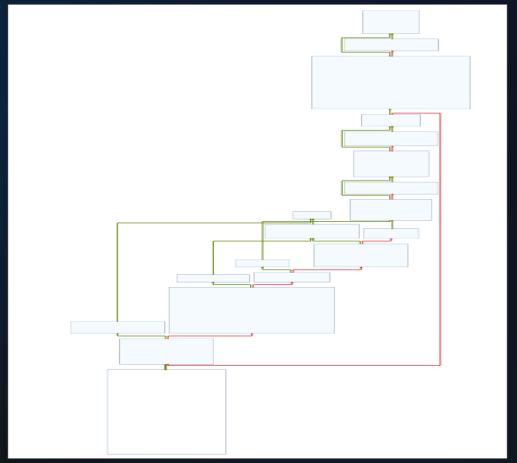
- Call graph obfuscation

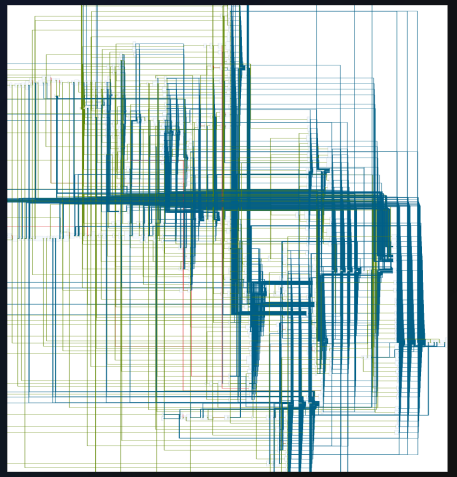
Examples:

- 1 Maximize binary diff
- 2 Maximize number of subroutines
- 3 Maximize number of calls between each subroutines

Automation Possiblities:

- Meta-heuristic (e.g., Genetic Algorithm): 1 2 3
- Reinforcement Learning: 1 2 3
- Heuristic: 2 3





Offline Evasion / Packing

Objective: Hide a payload by including it **into another**. At execution, the latter will **unpack** the payload and **pass execution** to it. Can include payload encryption & compression.

Automation Surface:

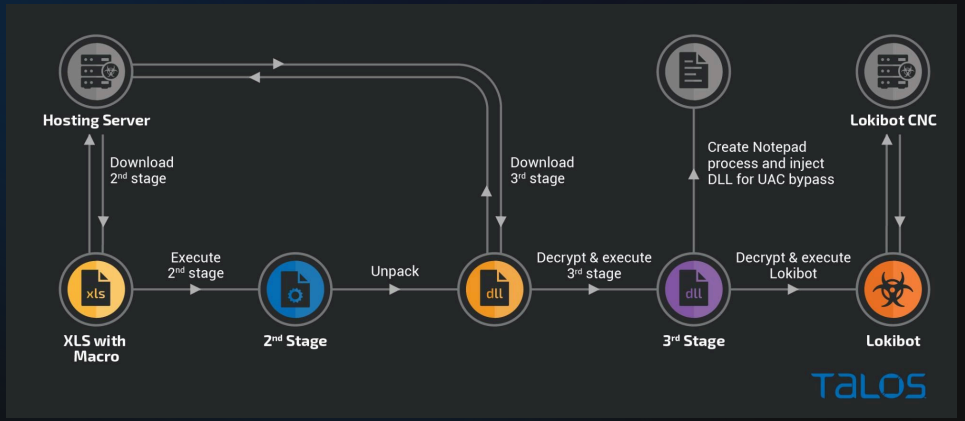
- Packer nesting
- Packer optimization

Examples:

- 1 Minimize packer entropy
- 2 Maximize waves diversity

Automation Possibilities:

- Meta-heuristic (e.g., Genetic Algorithm) 1 2
- Reinforcement Learning 1 2
- Heuristic: 1



Online Evasion

Definition

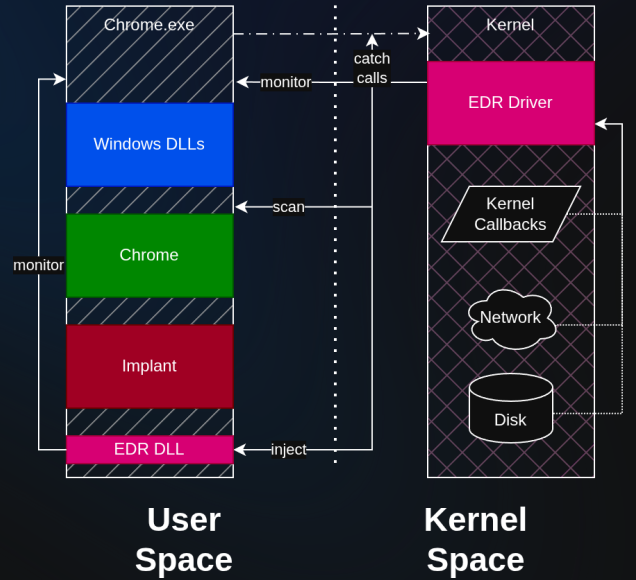
Online evasion includes any method that **reduce detection** & **evade** defenses **dynamically** (at runtime). We want to **hide traces** not to be detected by dynamic analysis.

This cover:

- Hide API calls,
- Hide in another process,
- Mask/Encrypt part of the payload in memory,
- Reproduce legitimate behaviors,
- ...

Some techniques:

- Sleep Obfuscation
- Memory Masking
- Process Injection
- Call Stack Spoofing
- ...



Online Evasion / Process Injection

Objective: Execute **malicous code** in the context of a **target process**.

This can turn any legitimate process into a malicious one.

Automation Surface:

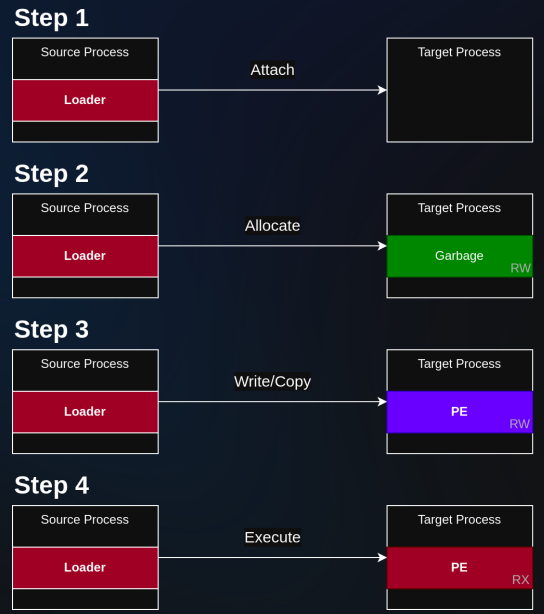
- Find API combination
- Choose target process

Examples:

- 1 Maximize behavior similarity of target process
- 2 Select best injection method (against specific defense)

Automation Possibilities:

- Meta-heuristic (e.g., Genetic Algorithm): 1 2
- Reinforcement Learning: 1 2
- Heuristic: 1



Online Evasion / Memory Masking

Objective: Try to **hide** part, or the entirety, of a payload **in memory**. This is useful against **memory scanners**, like YARA. **Sleep obfuscation** is a variant, where a payload will sleep & be encrypted in memory during that time.

Automation Surface:

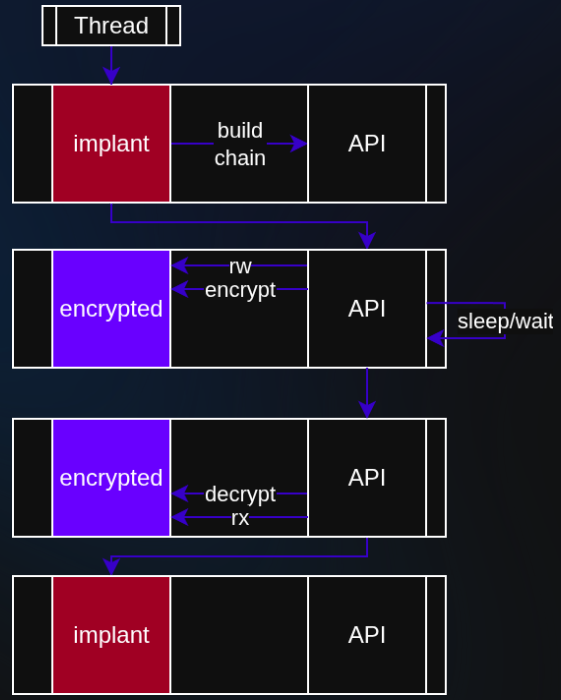
- Technique parameters
- API used

Examples:

- 1 Find "optimal" sleep/jitter value
- 2 Find "optimal" delay before remasking

Automation Possibilities:

- Meta-heuristic (e.g., Genetic Algorithm): 1 2
- Heuristic: 1 2



Online Evasion / Call Stack

Objective: Spoof or construct a **fake call stack**, to **hide original callers** and evade detection.

Automation Surface:

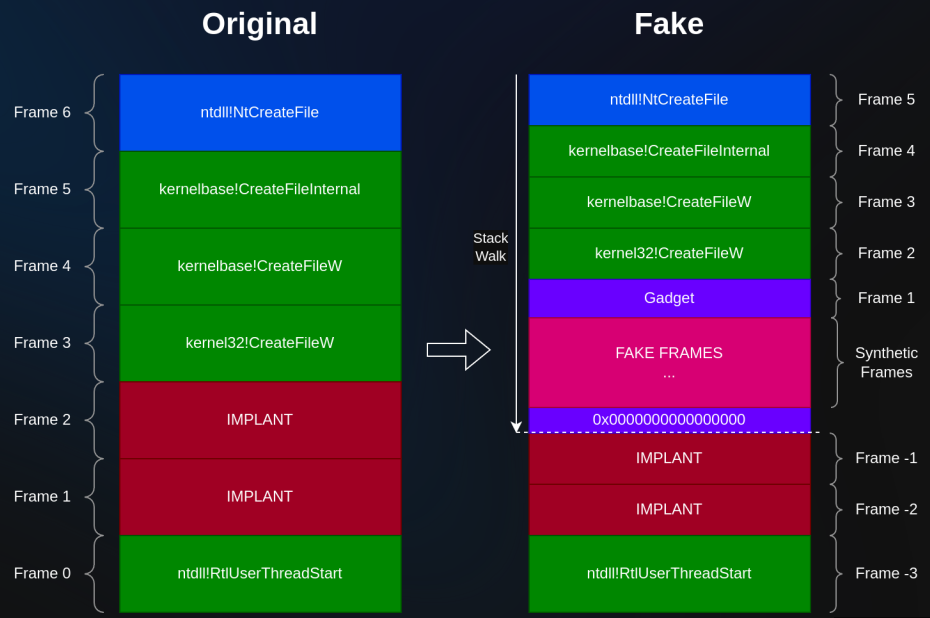
- Fake API choice

Examples:

- 1 Maximize similarity to injected process call stack
- 2 Find suspicious call stacks

Automation Possibilities:

- Meta-heuristic (e.g., Genetic Algorithm): 1 2
- Reinforcement Learning: 1 2
- Heuristic: 1



Meta-optimization

Objective: Each evasion technique has some variation available. Considering all variations of all techniques, this creates a lot of combination possibilities.

Automation Surface:

- Techniques combinations
- Techniques variations

Examples:

- 1 Minimize number of techniques
- 2 Find all combinations leading to bypass

Automation Possibilities:

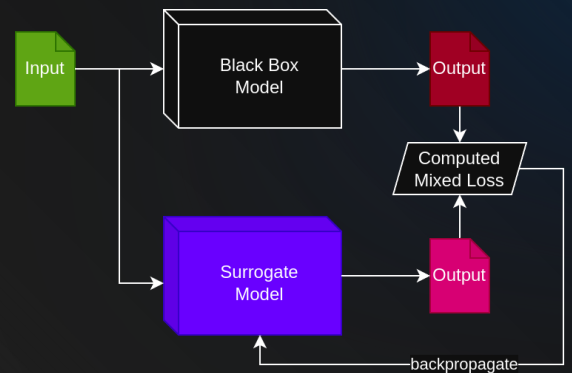
- Meta-heuristic (e.g., Genetic Algorithm): 1 2
- Reinforcement Learning: 1 2
- Surrogate Model: 2

Product	Call Stack	Process Injection	Metamorphism	Memory Masking
EDR1	Variant 2	"Explorer"	Variant 1	X
EDR2	Variant 2	X	Variant 1	X
EDR2 + Win11	Variant 3	"Google Chrome"	X	Variant 2

Surrogate Model

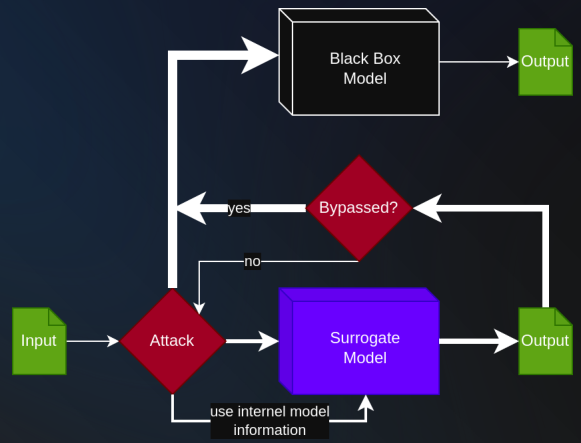
Definition

A surrogate model approximates a black box model by learning pairs of input/output.



Objective

We can then use the surrogate model to build adversarial inputs to force misclassification by the black box model. For that, we can use the surrogate model internals (like embedding or gradients).



Surrogate Model: Evasion

Usage for evasion

The black box model can be the classification algorithm of an EDR, the input a binary (offline evasion) or dynamic traces generated by it (online evasion) and the output the benign/malicious classification.

Possible attacks

- Add obfuscation, packing...
- Use process injection, call stack spoofing...
- Mask/Obfuscate memory...

All of these techniques are primitives that we can leverage to build adversarial binaries, using our surrogate model.

Formalization

With $E_{v,p}$ an evasion technique (of variant v and parameters p). $I(E_{v,p})$ is the "impact" of $E_{v,p}$ in the classification C given by the surrogate model (computed from gradients for example).

$$\epsilon_\Omega(x) = \bigcirc_{E_{v,p}\in\Omega} E_{v,p}(x) \tag{4}$$

where $\bigcirc$ denotes function composition $F_n \circ F_{n-1} \circ \cdots \circ F_1(x)$.

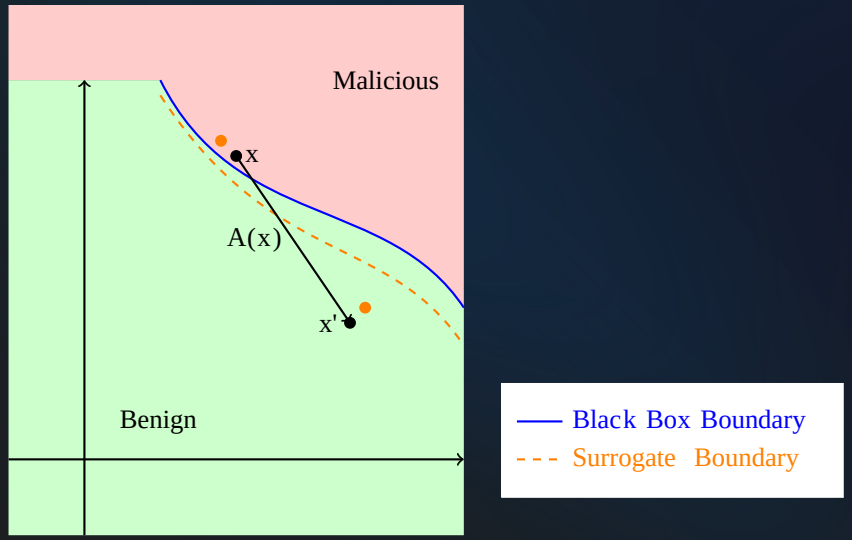
$$\mathcal{B} := \{\omega' \mid C(\epsilon_{\omega'}(x)) \neq C(x)\} \tag{5}$$

$$\mathcal{B}_{\min} := \{\omega' \in \mathcal{B} \mid |\omega'| = \min_{\omega'\in\mathcal{B}} |\omega'|\} \tag{6}$$

$$\Omega := \arg\max_{\omega'\in\mathcal{B}_{\min}} \sum_{E_{v,p}\in\omega'} I(E_{v,p}) \tag{7}$$

We could easily add a constraint to overcome the "burn" of some unique evasion techniques.

Surrogate Model: Visualization



Where $A(x) = \epsilon_\Omega(x)$.

Conclusion

Some "Hot Takes":

- Automation for evasion is **easier** than C2 piloting.
- Not much literature on the subject.
- Might not be explored by anyone?
- Multimodal $>$ Unimodal
- Linked **use cases** are important.
- When it will work, the **impact might be important** for defenses.
- (*LLMs are not the solution*)

Conclusion

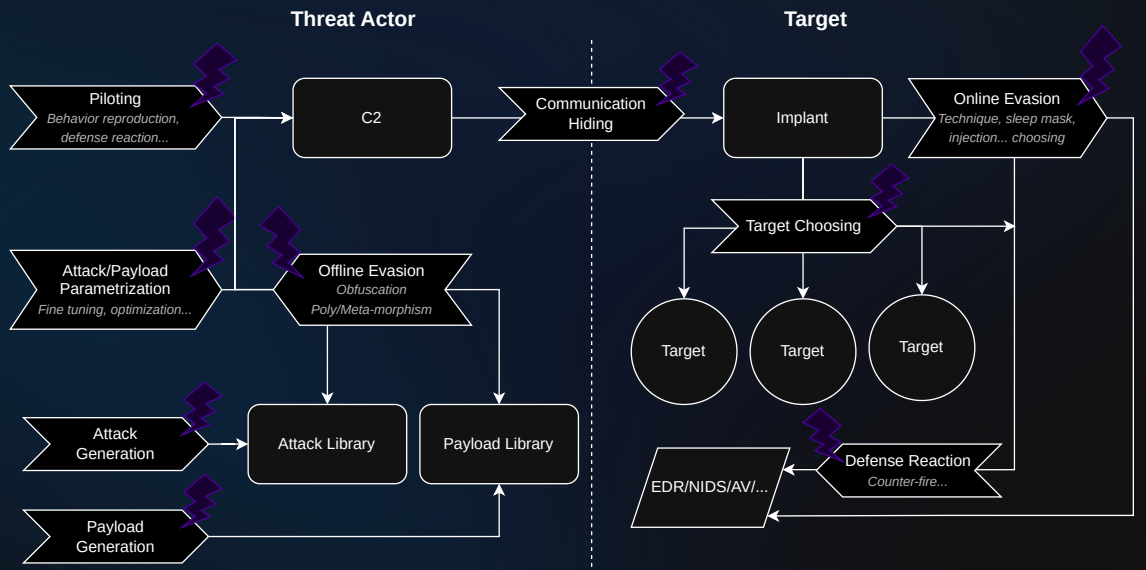
OAI is larger!

Not discussed, but:

- OAI for **attack generation**.
- OAI for **phishing**.
- OAI for **defense reaction**.
- OAI for **payload generation**.
- OAI for **communication hiding**.
- ...

The subject is **vast**!

But **not very explored** (in the literature).



Thanks!

A corresponding written blog post will be available soon https://dorianb.net/blog

Bibliography

- 13. Chen Z, Kang F, Xiong X, Shu H. A Survey on Penetration Path Planning in Automated Penetration Testing. Applied Sciences [Internet]. 2024;14(18). Available from: https://www.mdpi.com/2076-3417/14/18/8355 (37)