Intelligent Decision Support for Cybersecurity Incident Response Teams: Autonomic Architecture and Mitigation Search*

Camilo Correa[1], Jacques Robin[1], Raul Mazo[1,2], and Salvador Abreu[1,3]

[1] CRI, University of Paris 1 Panthéon-Sorbonne, Paris, France
{camilo.correa-restrepo,jacques.robin}@univ-paris1.fr
[2] Lab-STICC, ENSTA Bretage, Brest, France
raul.mazo@ensta-bretagne.fr
[3] NOVA-LINCS, University of Évora, Évora, Portugal
spa@uevora.pt

Abstract. Critical infrastructures must be able to mitigate, at runtime, suspected ongoing cyberattacks that have eluded preventive security measures. To tackle this issue, we first propose an autonomic computing architecture for a *Cyber-Security Incident Response Team Intelligent Decision Support System (CSIRT-IDSS)* with a precise set of technologies for each of its components. We then zoom in on the component responsible for proposing to the CSIRT, automatically ranked sets of runtime actions to mitigate suspected ongoing cyber-attacks. We formalize its task as a *Constraint Optimization Problem (COP)*. We then propose to implement it by a *Constraint Object-Oriented Logic Program (COOLP)* deployed as a containerized web service through the integration of three orthogonal extensions of *Logic Programming (LP)*: *Web Service Oriented LP (WSOLP)* [34] *Constraint LP (CLP)* [10] and *Object-Oriented LP (OOLP)* [23]. This integration supports seamlessly reusing platform and task independent cybersecurity ontological knowledge to dynamically build a mitigation action search COP that is customized to an input suspected cyberattack action set. This customization then allows the COP, to be solved by a generic CLP engine efficiently enough to propose mitigation actions to the CSIRT team while they can still be effective. To validate this approach, we implemented a prototype called *CARMAS (Cyber Attack Runtime Mitigation Action Search)* and ran scalability tests on simulated attacks with various COP construction strategies.

Keywords: Intelligent Decision Support System · Autonomic Computing · Cybersecurity Incident Response· Object-Oriented Logic Programming · Constraint Logic Programming.

* This work was partially funded by *Fundação para a Ciência e Tecnologia* under grant UIDB/04516/2020 (NOVA-LINCS) and by the project C4IIoT funded by the European Commission under Grant Agreement No. 833828. It reflects the views only of the authors. The funding agencies cannot be held responsible for any use which may be made of the information it contains.

1 Introduction

Defending critical network infrastructures in industrial environments has become a pressing need in recent years. The ongoing transition, from custom-built control systems to *Industrial Internet of Things (IIoT)* [29] based on low-cost, off-the-shelf, cloud connected devices, has considerably grown the attack surface exploitable by malicious agents [27]. Traditional approaches to cybersecurity emphasize attack prevention, laying out as many design-time defense mechanisms as is economically feasible. They sometimes add *post-mortem* analysis processes [7] to uncover and recover from attacks *after* they have occurred. These approaches are generally not automated and, in a way, come too late. This leaves a large window of opportunity for attackers to continue causing damage. Some attacks may remain undetected for up to several months [2]. To tackle this problem, new approaches are needed to detect an attack and apply countermeasures *swiftly* or even *before* it has been fully completed to best minimize its impact.

In this paper, we make two original contributions towards solving this problem. We first propose a concrete architecture for a CSIRT-IDSS shown in Fig. 1 as an *Unified Modeling Language (UML)* [30] class diagram[4]. This architecture refines, with a precise set of technologies, each abstract component of the classic autonomic computing architecture pattern ***MAPEK*** [17] shown on the top row of Fig. 1. It consists of a looping pipeline of *Monitor* (**M**), *Analyzer* (**A**), *Planner* (**P**) and *Executor* (**E**) components, sharing a common *Knowledge* (**K**) base, to autonomically manage a system.

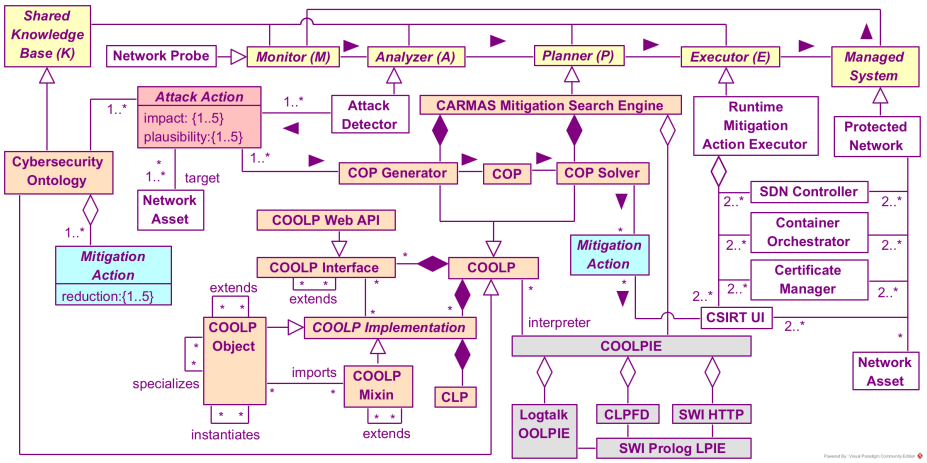

Fig. 1: MAPEK refinement for autonomic CSIRT-IDSS.

[4] To avoid clutter, this diagram omits the UML stereotypes «component» and «interface». It uses the following color coding: *yellow* for the abstract MAPEK components, *orange* for the refinements of which we are the sole (CARMAS) or main (Ontology) developer, *grey* for the third-party *Inference Engines (IE)* and libraries reused by CARMAS, *red* for its input, *blue* for its output and *white* for the refinements developed by C4IIoT project partners that either generate this input or act upon this output.

Our second contribution focuses on the **P** step, leaving the details of the **M, A** and **E** steps for other publications. We show how **P** can be formalized as a COP and propose a specific COOLP architectural pattern to solve it with practical performance. It consists of generating, at runtime, a COP customized to the input attack to mitigate, for then solving this COP using a generic CLP solver. Thanks to the taxonomic knowledge representation features of OOLP, the COP generator reuses knowledge from the task-independent cybersecurity ontology that refines the **K** component of the MAPEK loop.

We describe the prototype implementation of this mitigation engine search engine architectural pattern by the CARMAS COOLP. We also present scalability simulations comparing how various COP generation heuristics impact the performance of the combined *generation and solving* pipeline for attack action sets of increasing size and conceptual diversity. It shows that thanks to this approach, a declarative and generic CLP solver such as the CLP(FD) library of SWI-Prolog [32] can solve the generated COP efficiently enough to be usable in practice for cyberattack mitigation search.

2 Autonomic architecture for CSIRT-IDSS

Our proposed MAPEK refinement for a CSIRT-IDSS is shown in Fig. 1. It focuses on a single autonomic self-management capability: self-protection. It is part of a larger cybersecurity architecture framework under development within the H2020 project *C4IIoT (Cybersecurity for the Industrial Internet of Things)*. This framework includes a variety of preventive and post-mortem components developed by project partners that are beyond the scope of the MAPEK architecture shown in Fig. 1, as they do not directly interact with our CARMAS component.

The automation of the **K**-based **M, A, P** and **E** sub-task loop of an autonom***ic*** system self-management is very similar to the knowledge-based sense, reason, decide and act loop of autonom***ous*** agents [26]. However, while the latter aims to perform all decision steps without requiring any human intervention, the former rather provides a very high-level *User-Interface (UI)* to a human-in-the-loop. Through this UI, this human expert can partition decisions between, on the one hand, those that the autonomic system can take without supervision, and, on the other hand, those for which the autonomic system proposes alternative, automatically executable, high-level plans for the human to choose from. Autonomic computing is inspired from the nervous system where some functions are carried automatically without requiring conscious intervention (*i.e.,* heartbeat), whereas others are consciously triggered (*i.e.,* hyperventilating). This automation continuum it a particularly good fit for a CSIRT-IDSS.

As shown in the second row of 1, at its highest level, our MAPEK refinement for a CSIRT-IDSS proposes that (a) **M** consists of probes that snoop network packet traffic and host activities data (such as API and OS calls), (b) **A** consists of attack detection from the probe generated data, (c) **P** (CARMAS) consists of finding the network reconfiguration actions that can best mitigate the suspected attack, (d) **E** consists of executing these actions and (e) the shared **K** is a cybersecurity ontology. In this ontology, the choice of the bounded integer values ranging from 1 to 5 shown in Fig. 1 for the attributes of the classes rooting the attack and mitigation taxonomies were motivated by the requirement that they be easy to estimate and understand by the CSIRT members.

A outputs a set of suspected ongoing attack actions targeting a set of network assets that it passes to CARMAS. The plausibility attribute of each action comes from the attack detector while the impact attribute comes from the ontology. To mitigate the attack, CARMAS searches and ranks action sets and sends them to the CSIRT UI. Among these ranked action sets, the CSIRT then chooses which one to execute by calling sub-components of **E** (or none if it judges the attack alert input to CARMAS is a false positive). These sub-components are general management automation tools, already in place for normal network operations, that **E** reuses for its specific cyberattack runtime mitigation purposes. They include *Software Defined Network (SDN) controllers* [13] to reconfigure network parameters and routes, container orchestrators [25] to reconfigure technological stacks installed on network hosts with alternative, updated elements with no known vulnerabilities, and authentication certificate managers to reconfigure access rights to the hosted applications running on top of these stacks. Note that only the **E** component of our proposed MAPEK refinement depends on the use of these specific network management technologies. The CSIRT UI is also a sub-component of **E** since it controls the calls to its other sub-components.

3 IIoT Cybersecurity Ontology

The shared ontology, refining **K**, provides a reusable, task-independent and platform-independent reference conceptual model for the cybersecurity challenges of a given industry. The MAPEK refinement that we propose assumes that both the managed system and its autonomic self-protection components uniformly consist of containerized web services communicating through an SDN. The payload schema of the communication API between these services can be automatically generated from the ontology to support simultaneously rigorous and agile API evolution.

While the top-level concepts of a cybersecurity ontology, such as attack action, network asset, vulnerability and mitigation action are industry-independent, their refinements quickly become specific to a particular industry, a set of technologies and even a specific network, for the main operating mode of attackers is to spot the devil in the details. To illustrate such refinements, we show in Fig. 2 a snippet of the ontology under development for IIoT networks in the C4IIoT project. We are developing it in cooperation with the partners developing the refinements of the **M, A** and **E** components. It shows concepts from an attack action taxonomy (in red), a mitigation action taxonomy (in blue) and the many-to-many `mitigatedBy` role relations between them.

The C4IIoT ontology adopted a dual, synchronized representation: a high-level, graphical one in UML and a fully detailed, textual one as a COOLP. UML was a natural choice for both the overall architecture model of Fig. 1 and the shared conceptual model refining **K** because, while the components of the architecture run on different programming platforms, they all follow the OOP paradigm. Adopting a standard ontology language such as OWL [11], rather than UML, would have been a far steeper learning curve for a typical CSIRT. In addition, OWL's *Open-World Assumption (OWA)* is a semantic mismatch with OOP's *Closed-World Assumption (CWA)* [1]. However, while UML can intuitively represent taxonomies of concepts and local role relations between them, it cannot represent arbitrarily complex and possibly long-distance con-

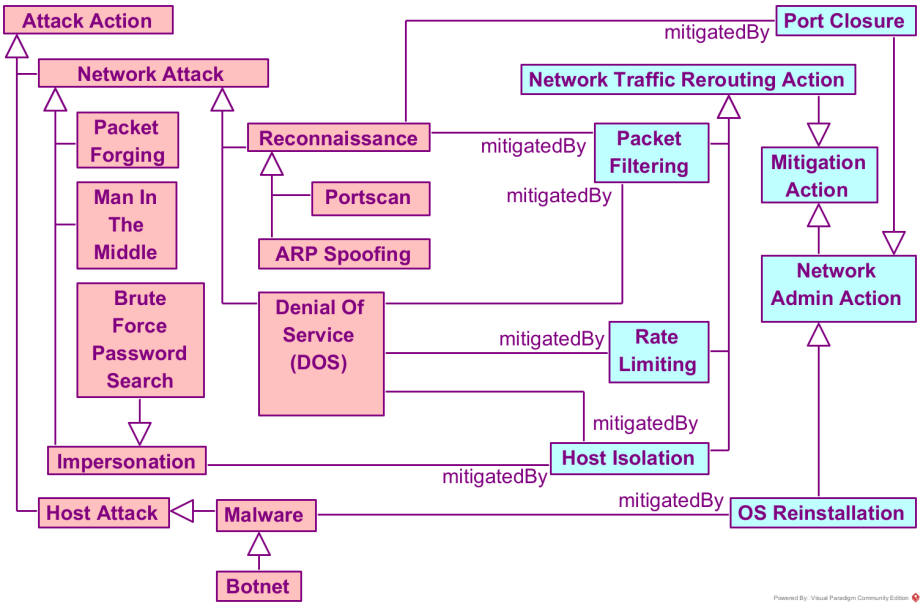

Fig. 2: Attack and mitigation action taxonomy snippet

straints among them. Nor does it provide formal semantics to support automated reasoning about them. This is why a refined, formal, executable, textual version of the ontology as a COOLP was also developed for components such as CARMAS that perform such reasoning. In the current implementation, we have so far synchronized the UML and COOLP versions of the ontology manually. We have however, specified a UML profile for COOLP as a first step towards automating it. The common OOP paradigm and CWA shared by the two languages make this translation far easier than a translation from UML to OWL would have been.

The snippet shown in Fig. 2 only aims to provide some illustrative context for the CARMAS running example presented in Section 6. The current attack taxonomy is primarily inspired by concepts proposed by MITRE [22] and Hindy et al. [14]. The current mitigation taxonomy is mostly inspired by the current focus of the project on (a) SDN controlled traffic rerouting actions and (b) network administration scripts remotely executable by the CSIRT.

4 Designing CARMAS as a COOLP web service

Let us now detail the design of CARMAS. Its main requirements are the following: (R1) solve a COP, formalized in Section 5, with practical performance, (R2) reuse the taxonomic knowledge of **K** for inheritance reasoning, (R3) be able to clearly explain its reasoning to the CSIRT and (R4) be deployed as a containerized web service inter-operating with the **A** and **E** components refinements. Each of these requirements suggest a different paradigm as the best fit to satisfy them: *Constraint Programming*

(CP) [10] for R1, *Object-Oriented Programming (OOP)* for R2, rule-based LP [10] for R3 and *Web Service Oriented Programming (WSOP)* for R4. This led us to develop CARMAS as a COOLP integrating those four paradigms.

As its input is a suspected attack action set of *unknown size*, CARMAS cannot just instantiate a predefined parametric COP. We therefore decomposed CARMAS into a pipeline of two components as shown in Fig. 1. The first generates a COP problem instance from the output of the attack detector specializing **A**. The second solves this COP and passes the solution as input to the runtime mitigation action executor specializing **A**. As the **P** component of the MAPEK pattern, CARMAS has access to knowledge encapsulated in the **K** component, the cybersecurity ontology. Both its COP Generator and COP Solver components thus also have access to this knowledge through inheritance and composition. Both are declaratively implemented as a COOLP interpreted by a *COOLP Inference Engine (COOLPIE)* that answers COOLP queries.

The multi-paradigm COOLPIE that interprets both COP generation and COP solving COOLP queries is assembled by three orthogonal extensions to the LP engine SWI-Prolog [34]: the CLP library CLP(FD) [32], the WSOLP library HTTP [35] and the OOLP extension Logtalk [23]. This is seamlessly done by putting Prolog module import directives `use_module(library(clpfd))` and `use_module(library(http))` in a Logtalk source file.

Logtalk is an object-oriented extension of Prolog. It transpiles Logtalk programs and queries into Prolog programs and queries and reuses the Prolog engine to answer the queries given the program. Logtalk is itself implemented in Prolog. As shown in Fig. 1, a Logtalk program is composed of three main top-level code entities: *objects*, and what OOP languages usually call *interfaces* and *mixins* (respectively called *protocols* and *categories* in Logtalk). Faithful to the spirit of OOP, a Logtalk object or mixin represents both a concept in a generalization taxonomy and a unit of code encapsulation. But rather than encapsulating both a computation state and a set of state altering operations as in imperative OOP, Logtalk encapsulates a declarative LP, or a CLP when used with a CLP library. Both objects and mixins can realize interfaces. Logtalk objects and mixins can thus both be viewed as extensions of Prolog modules. Mixins can be imported into objects to support a simple form of reuse by composition. Like JavaScript, Logtalk supports both class-based and prototype-based inheritance by providing *specializes, instantiates* and *extends* relations between objects. It does not make *class* a first-class concept. Instead, it makes it a *role* that an object plays in an *instantiates* relation with another object, which allows Logtalk to easily support, like Python, both meta-classes and multiple inheritance. Similarly, the concept of *prototype* is a role that an object plays in an *extends* relation with another object. Logtalk also supports inheritance down extension hierarchies defined over interfaces and mixins. Logtalks thus allows combining JavaScript and/or Python style OOP for structural code with the declarative, formal yet executable rule-based specification of CLP for behavioral code.

Using COOLP rather than CLP alone, allows simultaneously leveraging the built-in inheritance reasoning services provided by OOLP, down the cybersecurity ontology taxonomies, with the built-in heuristic optimization reasoning services provided by CLP. It also allows following a model-driven, object-oriented development methodology [11]. This is handy in the software engineering context of CARMAS, as a component in

an heterogeneous, multi-platform framework such as C4IIoT. The web server built in SWI Prolog's HTTP library and its seamless integration with Logtalk eased the deployment of CARMAS as a web service. Its built-in JSON serialization and deserialization predicates eased programming the data exchange between the inward-facing Logtalk interfaces of CARMAS and its outward-facing Web API interface to the other C4IIoT components. CARMAS is provided as a Docker image stacking two COOLPs, the COP Generator and the COP Solver, on top of Logtalk, CLP(FD) and SWI-HTTP, themselves on top of SWI-Prolog itself on top of Linux.

5 Formalizing attack mitigation search

The following definitions formalize attack mitigation search.

- Set definitions:
  1. A: the possible attack actions;
  2. $A_p \subset A$: input attack actions to a given mitigation search problem p;
  3. T: possible network assets targeted by members of A;
  4. $T_p \subset T$: target assets for a given problem p;
  5. M: the possible mitigation actions;
  6. $M(a_i) \subset M$ with $a_i \in A_p$, the possible actions to mitigate a given attack action a_i
  7. $M(t_i) \subset M$ with $t_i \in T_p$, the possible actions to mitigate the attack actions targeting a given asset t_i.
  8. $A_p(t_i) \subset A_p$ with $t_i \in T_p$, the attack actions targeting a given asset t_i.
- Function definitions:
  1. $at : A_p \rightarrow T_p$, asset target of a given attack action;
  2. $cl_A : A \rightarrow \mathscr{P}(A)$ (respectively $cl_M : M \rightarrow \mathscr{P}(M)$, $cl_T : T \rightarrow \mathscr{P}(T)$) ontological class of a given attack action (respectively mitigation action, target asset);
  3. $p : A_p \rightarrow \{1,...,5\}$ plausibility of a given attack suspicion;
  4. $i : \mathscr{P}(A) \times \mathscr{P}(T) \rightarrow \{1,...,5\}$ negative business impact, if left unmitigated, of an attack of a given action class targeting a given asset class;
  5. $r : \mathscr{P}(M) \times \mathscr{P}(A) \times \mathscr{P}(T) \rightarrow \{1,...,5\}$: negative business impact *reduction* of mitigation actions from a given class for a given attack class targeting a given asset class;
  6. $u : \mathscr{P}(M) \times \mathscr{P}(A) \rightarrow \mathbb{N}$: utility of mitigation action set $M' \subset M$ for attack action set $A' \subset A$ s.t.

$$u(M',A') = \sum_{m_i \in M', a_j \in A'} 5 - |p(a_j) - r(cl_M(m_i), cl_A(a_j), cl_T(at(a_j)))|.$$

- Relation definition: $mtg \subset \mathscr{P}(M) \times \mathscr{P}(A) \times \mathscr{P}(T)$, the mitigation actions of a given class can mitigate attack actions of a given class targeting network assets of a given class.
- Definition of integrity constraints for input problem instance and the ontology:
  1. $\forall a_i \in A_p, \exists (x,y) \in \{1,...,5\}^2 \ \left[x = p(a_i) \wedge y = i(cl_A(a_i), cl_T(at(a_i)))\right]$ (each attack action must have a plausibility and an impact);

2. $\forall a_i \in A_p, \exists (m_j, x) \in M \times \{1,...,5\}\ x = r(cl_M(m_j), cl_A(a_i), cl_T(at(a_i)))$ (each attack action must have mitigation actions with an impact reduction);
3. $\forall a_i \in A_p, \forall m_j \in M, \exists (x,y) \in \{1,...,5\}^2\ [mtg(cl_M(m_j), cl_A(a_i), cl_T(at(a_i))) \wedge x = r(cl_M(m_j), cl_A(a_i), cl_T(at(a_i))) \wedge y = i(cl_A(a_i), cl_T(at(a_i)))] \Rightarrow x < y$ (the impact reduction of a mitigation action cannot exceed the impact of the attack action that it reduces).

– Definition of output solution sets:
1. $\mathscr{F}_M(A_p) = \{M_{sat} \subset M \mid \forall a_i \in A_p, \exists! m_j \in M_{sat} [mtg(cl_M(m_j), cl_A(a_i), cl_T(at(a_i)))]\}$ family of all solution mitigation action sets;
2. $M_{max}(A_p) = \underset{M_{sat} \in \mathscr{F}_M(A_p)}{\arg\max}\ u(M_{sat}, A_p)$ solution that maximizes utility of the mitigation actions.

The formal notation just defined relates to the the UML model elements of Fig. 1 as follows. The Attack Action set that the Attack Detector component passes as input to the COP Generator is formalized by the sets A_p and T_p, and the functions at and p. The knowledge encapsulated in the ontology that enriches this input is formalized by the sets A, T and M, the functions cl_A, cl_M, cl_T, i and r, the relation mtg and the three integrity constraints. The Mitigation Action set output by the COP Solver is formalized by the $M_{max}(A_p)$ function.

The u function is constructed by the COP Generator and used by the COP Solver. Its formula favors choosing mitigation actions with higher (respectively mid-range, lower) reduction impact r for suspected attack actions with higher (respectively mid-range, lower) plausibility p. The rationale behind this choice is that mitigations with higher attack impact risk reduction generally tend to come with a higher negative impact on other factors such as service availability. Consider for example a DOS attack; the ontology snippet of Fig. 2 shows three possible mitigations for it: host isolation, rate limiting and packet filtering. While the first one has a higher attack risk impact reduction that the two others, it also more severely reduces the availability of the service running on the host suspected as the attack source. Thus, when the plausibility of this host being the source of a DOS attack targeting another network host is low, choosing packet filtering and/or rate limiting over host isolation is better for the overall availability of the network services. The u formula allows CARMAS to implicitly carry out such trade-off while still only building and solving a single objective COP, rather than a more complex multi-objective COP.

COP generation starts by creating the variable set:

$$V = D \cup R \cup P \cup \{u_p\}$$

where:

$$D = \{d(m_j, a_i) \in \{0,1\} \mid a_i \in A_p \wedge m_j \in M'\}$$

$$M' = \{m_j \in M \mid a_i \in A_p \wedge mtg(cl_M(m_j), cl_A(a_i), cl_T(at(a_i)))\}.$$

$$P = \{p(a_i) \in \{1,...,5\} \mid a_j \in A_p\}$$

$$R = \{ r(cl_M(m_j), cl_A(a_i), cl_T(at(a_i))) \in \{1, ..., 5\} \mid a_i \in A_p \wedge m_j \in M' \wedge mtg(cl_M(m_j), cl(a_i), cl(at(a_i))) \}$$

$$u_p = \sum_{\delta \in D, \rho \in R, \pi \in P} \delta * (5 - |\pi - \rho|)$$

In the set definitions above, $d(m_j, a_i) = 1$ when the COP solver selects m_j to mitigate a_i among the set M' of candidates, and $d(m_j, a_i) = 0$ otherwise. D is thus a set of *decision* Boolean variables that represents the COP solver's mitigation action choices. P and R are sets of integer variables with values in $\{1, ..., 5\}$. P is a set of *parameter* variables containing the plausibility of each input suspected attack action to mitigate. R is a set of *parameter* variables that comes from the ontology and contains the expected business impact reduction of each mitigation action on the attack action it is selected to mitigate. u_p is a positive integer *optimization* variable that the COP solver tries to maximize to guide its choices of value for the decision variables. With the variable defined above, the COP consists in choosing the allocation of *decision variables D* that maximizes the *optimization variable* u_p given the values of *parameter* variables P and R.

The worst-case complexity of a Boolean COP is 2 to the power of the number of its decision variable (*cf.* [26] p.199). The most straightforward COP generation strategy yields a COP with $\sum_{a_i \in A_p} |M(a_i)|$ decision variables. Its complexity is thus: $SF = O(2^{\sum_{a_i \in A_p} |M(a_i)|})$.

To improve the performance of the COP solver we propose two simple heuristics to generate equivalent COPs with less variables. The first is *asset-based decomposition*. It consists of building one sub-COP per targeted asset in the input, solving each of them and taking the union of the solutions as the global COP solution. With this heuristic, the number of variables for each sub-COP is upper-bounded by $\max_{t \in T_P} |M(t)|$. The total complexity of the union of all the sub-COPs is thus upper-bounded by: $ABD = O((2^{\max_{t \in T_P} |M(t)|}).(\sum_{a_i \in A_p} |M(a_i)|)/(\max_{t \in T_P} |M(t)|)))$ (*cf.* [26] p.199 again) whose growth rate is less than that of SF since $\max_{t \in T_P} |M(t)| \leq \sum_{a_i \in A_p} |M(a_i)|$. This is because the set of actions attacking a given asset is a subset of the whole attack action input set. Thus, the size of the search space of the sub-problems will necessarily be smaller than or equal to that of the whole problem. This heuristic can be used by CARMAS because, in the COP that it must generate and solve, the value choice for any given decision variable is independent from the choices for all other decision variables.

The second heuristic is *uniform mitigation* where the following constraint is added to the COP: all actions that mitigate attack actions of a given class must all belong to the same mitigation class. Or formally:

$$\forall a_i \in A_p, \forall m_j \in M' \Big[\big[d(m_j, a_i) = 1 \wedge cl_M(m_j) = c_m \wedge cl_A(a_i) = c_a \big] \Rightarrow \forall a_l \in A_p, \forall m_q \in M' \setminus \{m_j\} \big[(d(m_q, a_l) = 1 \wedge cl_M(m_q) = c_m) \Rightarrow cl_A(a_l) = c_a \big] \Big]$$

It assumes that if a given mitigation action is appropriate for an attack action of a given attack class, actions of the same mitigation class are likely to be effective for all other actions of that attack class. It dramatically reduces the number of decisions to be

taken by the COP solver from $\sum_{a_i \in A_p} |M(a_i)|$ to just
$d = \sum_{k_A \in \{cl(a_i) | a_i \in A_p\}} |\{k_M | \exists t \in T_p[mtg(k_M, k_A, cl_t(t))]\}|$, where $(k_M, k_A) \in \mathscr{P}(M) \times \mathscr{P}(A)$; that is, the sum, over the classes of attacks given as input, of the number of classes that can mitigate them. The resulting COP complexity is thus $UM = O(2^d)$. The price to pay for this performance gain is to give up the guarantee that the solver always return at the top of its list of proposed mitigation action sets, the one that maximizes the reduction of the attack impact.

6 CARMAS Running Example

To illustrate how our CARMAS prototype implementation works, at a very high-level, we now provide and explain four code listings. Listing 1.1 shows a snippet of the top-level CARMAS COOLP web service object that generates an HTTP response from an HTTP request. Listing 1.2 shows a request payload: a JSON object containing an example input set of two reconnaissance attack actions, one targeting an Android tablet and the other a Linux server. Listing 1.3 contains a snippet from a logged trace of the logical variable bindings resulting from the COOLPIE evaluating the goals of line 7 in listing 1.1. Listing 1.4 shows the JSON payload generated by CARMAS as answer to the request. It contains the actions to mitigate the attack actions from the client's request.

Let us now explain each listing a bit further in turn. In listing 1.1, lines 1 and 11 shows that the CARMAS web server is encapsulated in a COOLP object. Line 2 routes the request URL to the appropriate predicate handler of that object. Lines 4 to 10 shows the COOLP rule defining this predicate. Lines 5 and 6 extract the JSON payload from the request and convert it to a SWI-Prolog logical term. On line 7, the web server invokes the COP generating then solving predicates of the `carmas` COOLP object. The arguments `DVars` and `U` of these predicates respectively contain the decision variable set D and the optimization variable u_p defined in Section 5. Line 8 builds a term containing the mitigation action set chosen by the decision variables and line 9 converts it to a JSON object. Line 10 sends the object as payload of the HTTP response.

In listing 1.3, lines 1-4 show the `COP` logical variable once bound by the COOLPIE by evaluating the `gen_cop` goal predicate of the `carmas` object on line 7 of listing 1.1. It contains a set of candidate actions, retrieved from the ontology, to mitigate the input attack action set shown in listing 1.2. Each such candidate action is associated with a pair of logical variables, one, `DVarN` for the decision to choose this action, and the other, `IVarN`, for the contribution of this choice to the optimization variable. In this example, `COP` contains two candidate actions to mitigate each attack action from listing 1.2: one port closure and one packet filtering. Continuing with listing 1.3, lines 5 and 8 of show `DVars` before and after the COOLPIE evaluates the `solve_cop` goal predicate of the `carmas` object on line 7 of listing 1.1. Also in listing 1.3, lines 6-7 show the mitigation action set logical variable `Ms` bound by the COOLPIE by evaluating the `solve-cop` goal predicate of the `carmas` object on line 7 of listing 1.1. `Ms` then contains the choice of a port closure action to mitigate the reconnaissance targeting the Android tablet and of a packet filtering action to mitigate the reconnaissance targeting the Linux server.

Listing 1.1: CARMAS COOLP web service code snippet

```
:- object(server).
  {:- http_handler(root(ap), [Request]>>(server::handle_attack(Request)),
      ↪ [])}.
  ...
  handle_attack(Request) :-
    {http_json:http_read_json(Request, Data, [json_object(dict),
        ↪ value_string_as(atom)])},
    serialization::deserializeAttackActionSet(Data, AASetTerm),
    carmas::gen_cop(AASetTerm, COP, DVars, U), carmas::solve_cop(DVars,U),
    carmas::selectMitigationActions(COP, Ms), carmas::mitigationTerm(Ms,
        ↪ MASetTerm),
    serialization::serialize(MASetTerm, MADict),
    {http_json:reply_json(MADict, [status(200), json_object(term)])}.
:- end_object.
```

Listing 1.2: CARMAS COOLP web service JSON input example

```
{"attack_actions": [
    {"plausibility": 4, "attack_type": "reconnaissance", "attack_id": 1,
     "target": {"target_id": "tt", "os": "android", "ip":"1.1.1.1", "
         ↪ target_type":"tablet"}},
    {"plausibility": 5, "attack_type": "reconnaissance", "attack_id": 2,
     "target": {"target_id": "td", "os": "linux", "ip": "1.1.1.2", "
         ↪ target_type":"appServer"}}],
 "attack_id": "ap1"}
```

Listing 1.3: CARMAS COOLP internal logical variable binding log snippet for example input/output

```
COP = [[portClosure(tablet(tt, "1.1.1.1", android), 1, reconnaissance)-(
    ↪ DVar1-IVar1),
        packetFiltering(tablet(tt, "1.1.1.1", android), 1, reconnaissance)
            ↪ -(DVar2-IVar2)],
       [portClosure(appServer(td, "1.1.1.2", linux), 2, reconnaissance)-(
           ↪ DVar3-IVar3),
        packetFiltering(appServer(td, "1.1.1.2", linux), 2, reconnaissance
            ↪ )-(DVar4-IVar4)]],
DVars = [DVar1, DVar2, DVar3, DVar4],
Ms = [portClosure(tablet(tt, "1.1.1.1", android), 1, reconnaissance),
      packetFiltering(appServer(td, "1.1.1.2", linux), 2, reconnaissance)
          ↪ ],
DVars = [1, 0, 0, 1], U = 23,
```

Listing 1.4: CARMAS COOLP web service JSON output example

```
{"mitigationActions": [
    {"mitigation_action_type": "portClosure", "attack_type": "
        ↪ reconnaissance", "attack_id": 1,
     "target": {"target_id": "tt", "os": "android", "ip":"1.1.1.1", "
         ↪ target_type":"tablet"}},
    {"mitigation_action_type": "packetFiltering", "attack_type": "
        ↪ reconnaissance", "attack_id": 2,
     "target": {"target_id": "td", "os": "linux", "ip": "1.1.1.2", "
         ↪ target_type":"appServer"}}],
 "id": "mp1"}
```

7 Experimental Evaluation

To empirically evaluate the scalability of CARMAS, we carried out a set of execution time measurements with synthetic input attack sets of growing size in terms of number of attack actions and/or number of assets targeted by these attacks. Those two numbers are not always aligned. Although we assume that each input attack action only targets a single asset, multiple input attacks actions can nevertheless target the same asset. The synthetic attack sets were randomly generated from the classes of the attack taxonomy while ensuring that they satisfied all the integrity constraints of the ontology. To draw practical usability conclusions from the measurements presented below, one must note that in the current version of C4IIoT, CARMAS is used in a *reactive* manner, searching for a mitigation action set as soon as an attack is suspected and assuming that some of these actions will be immediately carried out by the CSIRT. Therefore, it does not reason on a *history* of attack actions, but rather only considers newly arriving attack actions. This considerably limits the current practical size of its input. However, we felt it was interesting to assess its scalability to larger inputs, if it becomes required to take as input the attack history in a future version of C4IIoT.

In the first experiment, we compared the time CARMAS takes to propose to the CSIRT from one to six alternative action sets to mitigate attack action sets of growing size. For an IDSS such as CARMAS performing heuristic search on the basis of an uncertain input, providing a few alternatives to human experts to choose from is important. However, too many alternatives is counterproductive to support a swift choice by the CSIRT. For single attack inputs, CARMAS took 0.06s to propose a one solution and 0.13s to propose six alternative solutions. Its response time remains practical up to about 10 attack inputs with 5.26s for one solution and 13.32s for six solution.

In the second experiment we compared execution time without and with the uniform-mitigation heuristic described in Section 5. Unsurprisingly, both these times grow exponentially with input size but with a different exponent. Using this heuristic reduces the response time for input of size 1, 5, 10 and 15 from 0.06s, 0.44s, 5.26s and 51s down to 0.07s, 0.74s, 0.105s and 0.134s respectively.

Table 1: Asset-based decomposition heuristic performance

Actions/Asset	Number of Assets				
	1 Asset	2 Assets	5 Assets	10 Assets	15 Assets
1 Action	0.001(s)	0.002(s)	0.004(s)	0.009(s)	0.117(s)
2 Actions	0.002(s)	0.004(s)	0.011(s)	0.019(s)	0.090(s)
5 Actions	0.020(s)	0.041(s)	0.112(s)	0.121(s)	0.222(s)
10 Actions	0.545(s)	1.977(s)	3.260(s)	6.692(s)	7.355(s)
15 Actions	18.746(s)	25.360(s)	181.974(s)	293.632(s)	476.140(s)

In the third experiment we compared execution time with the asset-based decomposition heuristic described in Section 5 for both a growing number of target assets and a growing number of attack actions for each target. The results are shown in Table. 1. The

main take away from this table is that with this heuristic, CARMAS responds quickly even for inputs much larger than expected in practice.

8 Related work

To show the originality of our proposal, we now compare it with previous work closely related to either (a) its application, runtime cyberattack mitgation or (b) its approach, leveraging ontological knowledge to dynamically build a COP so as to solve it efficiently by a generic solver.

Concerning (a), we only found one previous work (Nespoli et al. [24]) to propose and evaluate an approach to search a many-to-many mapping space from attack action classes to mitigation action classes. It presents a runtime mitigation search system based on the artificial immune system metaphor. The search is done by an iterative algorithm that refines and modifies mitigation actions that fit detected attacks on a class of a system.

All the other previous works tackle simpler search problems by only focusing on a single mitigation class (Marsa-Maestre et al. [21]), a one-to-one mapping from attack action classes to mitigation action classes (Sandor et al. [28]) or a one-to-many mapping from the former to the latter (Lysenko et al. [19], Huertas-Celdran et al. [5], [20]). While Gonzales-Granadillo et al. [12], Kanoun et al. [16], Javornik et, al. [15], Yuan et al. [36], and Vieira et al. [33] make other original proposals, they either do not discuss their implementation or do not evaluate it through an experiment or simulation, making it difficult to assess their practical applicability.

Concerning (b) we also only found a single previous work (Zhu et al. [37]) similar to our approach. Taking as input an F-Logic [18] ontology that models a database schema evolution, their approach translates it into a Prolog and *Constraint Handling Rules* [9] CLP that declaratively models, as a COP, a wrapper allowing database queries following a new schema to be run on historical data following an old schema. The similarity with our approach is that both their ontology and ours are specified in an OOLP language under CWA, F-Logic for them and Logtalk for us. The key design difference between these two languages is that in F-Logic, *objects appear inside rules* taking the place of Prolog atomic formulas, whereas in Logtalk, it is the other way around, as *rules appear inside objects* which replace Prolog modules. Another difference is that while we leverage the seamless *integration* of OOLP with CLP provided by Logtalk and CLP(FD), they, in contrast, implemented a *translation* from OOLP to CLP. Furthermore, they build COPs for a completely different application than ours.

In all other previous works leveraging ontologies and COPs in concert, the respective roles of the former, the latter and the combination differed notably from our proposal. Camacho et al. [4] use an ontology to represent the state of a smart home and specify as a COP the resolution of user preferences conflicts on such states. Fowler et al. [8] follows a similar approach for a mechanical engineering design assistant. In these works, ontological knowledge merely instantiates parameters of a fixed structure COP rather than dynamically generating that structure as we do. Chesani et al. [6] use an ontology for pre-processing and validating requests, formulated as COPs, to aid in the

management of food items in a smart warehouse. Torta et al. [31] encode constraints directly into their ontology, to filter geographic information system queries.

9 Conclusion

In this paper, we made four original contributions to advance the state-of-the-art in CSIRT-IDSS. The first is a detailed architecture pattern for such system with runtime attack detection and mitigation capabilities. It is a refinement of the classic **MAPEK** autonomic architecture in which **M** is a set of network traffic and host activity data probes, **A** is an attack detector, **P** is a mitigation search engine, **E** is a mitigation executor that reuses SDN, container orchestration and certificate management tools, and **K** is a cybersecurity ontology. In this architectural context, we then made three more contributions focused on the mitigation search engine. The first is a formal definition of its task as a COP. The second is its decomposition into a COP generator, that heavily relies on knowledge from the ontology, pipelined into a COP solver. The third is the proposal of COOLP as a parsimonious unifying paradigm implementing both the generator and the solver. We showed the practical efficiency of this approach through a set of scalability experiments for runtime attack mitigation searches.

One limitation of our approach is that it formalizes mitigation search as a single objective COP: maximizing the business impact reduction of the attack. In future work, we intend to rethink this task as a *multi-objective* COP taking into account other criteria to choose among mitigation actions such as their impact on the availability of the various services deployed on the network to protect and the cost of their execution.

Another limitation is that both the attack action input set and the mitigation action output set are unstructured. In future work, we intend to extend our cybersecurity ontology to include temporal and causal relations between these actions, as well as spatial relations in terms of network topology. This will allow the CSIRT-IDSS to represent and reason about the spatio-temporal progression of a multi-stage attack, hypothesize the plan behind the observed anomalies and in some cases mitigate some plan actions before they are executed. This is a major endeavor since leveraging such richer ontology will require both extending the attack detector with a probabilistic adversarial plan recognition capability and extending CARMAS with a probabilistic adversarial planning capability [3].

References

1. Baset, S., Stoffel, K.: Object-oriented modeling with ontologies around: A survey of existing approaches. International Journal of Software Engineering and Knowledge Engineering **28**(11n12), 1775–1794 (2018)
2. Bilge, L., Dumitraş, T.: Before we knew it: an empirical study of zero-day attacks in the real world. In: Proceedings of the 2012 ACM conference on Computer and communications security. pp. 833–844 (2012)
3. Braynov, S.: Adversarial planning and plan recognition: Two sides of the same coin. In: Secure Knowledge Management Workshop. vol. 3, pp. 67–70 (2006)

4. Camacho, R., Carreira, P., Lynce, I., Resendes, S.: An ontology-based approach to conflict resolution in home and building automation systems. Expert systems with applications **41**(14), 6161–6173 (2014)
5. Celdrán, A.H., Karmakar, K.K., Mármol, F.G., Varadharajan, V.: Detecting and mitigating cyberattacks using software defined networks for integrated clinical environments. Peer-to-Peer Networking and Applications pp. 1–16 (2021)
6. Chesani, F., Cota, G., Lamma, E., Mello, P., Riguzzi, F., et al.: A decision support system for food recycling based on constraint logic programming and ontological reasoning. In: 33rd Italian Conference on Computational Logic. vol. 2214, pp. 117–131. CEUR-WS. org (2018)
7. Cichonski, P., Millar, T., Grance, T., Scarfone, K., et al.: Computer security incident handling guide. NIST Special Publication **800**(61), 1–147 (2012)
8. Fowler, D.W., Sleeman, D., Wills, G., Lyon, T., Knott, D.: The designers' workbench: Using ontologies and constraints for configuration. In: International Conference on Innovative Techniques and Applications of Artificial Intelligence. pp. 209–221. Springer (2004)
9. Frühwirth, T.: Constraint handling rules. Cambridge University Press (2009)
10. Frühwirth, T., Abdennadher, S.: Essentials of constraint programming. Springer (2003)
11. Gaševic, D., Djuric, D., Devedžic, V.: Model driven engineering and ontology development. Springer Science & Business Media (2009)
12. Gonzalez-Granadillo, G., Doynikova, E., Kotenko, I., Garcia-Alfaro, J.: Hypergraph-driven mitigation of cyberattacks. Internet Technology Letters **1**(3), e38 (2018)
13. Goransson, P., Black, C., Culver, T.: Software defined networks: a comprehensive approach. Morgan Kaufmann (2016)
14. Hindy, H., Brosset, D., Bayne, E., Seeam, A.K., Tachtatzis, C., Atkinson, R., Bellekens, X.: A taxonomy of network threats and the effect of current datasets on intrusion detection systems. IEEE Access **8**, 104650–104675 (2020)
15. Javorník, M., Komárková, J., Husák, M.: Decision support for mission-centric cyber defence. In: Proceedings of the 14th International Conference on Availability, Reliability and Security. pp. 1–8 (2019)
16. Kanoun, W., Cuppens-Boulahia, N., Cuppens, F., Araujo, J.: Automated reaction based on risk analysis and attackers skills in intrusion detection systems. In: 2008 Third International Conference on Risks and Security of Internet and Systems. pp. 117–124. IEEE (2008)
17. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003)
18. Kifer, M., Lausen, G.: F-logic: a higher-order language for reasoning about objects, inheritance, and scheme. In: Proceedings of the 1989 ACM SIGMOD international conference on Management of data. pp. 134–146 (1989)
19. Lysenko, S., Savenko, O., Bobrovnikova, K., Kryshchuk, A.: Self-adaptive system for the corporate area network resilience in the presence of botnet cyberattacks. In: International Conference on Computer Networks. pp. 385–401. Springer (2018)
20. Maati, B., Saidouni, D.E.: Ciotas protocol: Cloudiot available services protocol through autonomic computing against distributed denial of services attacks. Journal of Ambient Intelligence and Humanized Computing pp. 1–30 (2020)
21. Marsa-Maestre, I., Gimenez-Guzman, J.M., Orden, D., de la Hoz, E., Klein, M.: React: Reactive resilience for critical infrastructures using graph-coloring techniques. Journal of Network and Computer Applications **145**, 102402 (2019)
22. MITRE: Att&ck® for industrial control systems (2021), `https://collaborate.mitre.org/attackics/index.php/Main_Page`
23. Moura, P.: Logtalk-Design of an Object-Oriented Logic Programming Language. Ph.D. thesis, Department of Computer Science, University of Beira Interior, Portugal (2003)
24. Nespoli, P., Mármol, F.G., Vidal, J.M.: A bio-inspired reaction against cyberattacks: Ais-powered optimal countermeasures selection. IEEE Access (2021)

25. Rice, L.: Container Security: Fundamental Technology Concepts that Protect Containerized Applications. O'Reilly Media (2020)
26. Russel, S., Norvig, P.: Artificial Intelligence; A Modern Approach. Pearson, 4th edn. (2020)
27. Sadeghi, A.R., Wachsmann, C., Waidner, M.: Security and privacy challenges in industrial internet of things. In: 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC). pp. 1–6. IEEE (2015)
28. Sándor, H., Genge, B., Szántó, Z., Márton, L., Haller, P.: Cyber attack detection and mitigation: Software defined survivable industrial control systems. International Journal of Critical Infrastructure Protection **25**, 152–168 (2019)
29. Serpanos, D., Wolf, M.: Industrial Internet of Things, pp. 37–54. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-69715-4_5, `https://doi.org/10.1007/978-3-319-69715-4_5`
30. The Object Management Group: Unified modeling language (UML) version 2.5.1. Standard (Dec 2017), [Online; accessed 21-April-2021]
31. Torta, G., Ardissono, L., Fea, D., La Riccia, L., Voghera, A.: A semantic approach to constraint-based reasoning in geographical domains. In: International Joint Conference on Knowledge Discovery, Knowledge Engineering, and Knowledge Management. pp. 202–227. Springer (2018)
32. Triska, M.: The finite domain constraint solver of SWI-Prolog. In: FLOPS. LNCS, vol. 7294, pp. 307–316 (2012)
33. Vieira, K.M., Schubert, F., Geronimo, G.A., de Souza Mendes, R., Westphall, C.B.: Autonomic intrusion detection system in cloud computing with big data. In: Proceedings of the International Conference on Security and Management (SAM). pp. 173–178. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp) (2014)
34. Wielemaker, J., Huang, Z., Van Der Meij, L.: Swi-prolog and the web. Theory and practice of logic programming **8**(3), 363–392 (2008)
35. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theory and Practice of Logic Programming **12**(1-2), 67–96 (2012)
36. Yuan, E., Malek, S., Schmerl, B., Garlan, D., Gennari, J.: Architecture-based self-protecting software systems. In: Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures. pp. 33–42 (2013)
37. Zhu, H., Madnick, S.E., Siegel, M.D.: Reasoning about temporal context using ontology and abductive constraint logic programming. In: International Workshop on Principles and Practice of Semantic Web Reasoning. pp. 90–101. Springer (2004)