D. Demmler, D. Krupka, H. Federrath. (Hrsg.): INFORMATIK 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 1531

Modeling Security Controls and System Assets As Autonomous Planning Tasks

Mahmoud Khalaf[1], Ludger Peters[2], Karl Waedt[3]

Abstract:

Safety I&C (Instrumentation & Control) and Operational I&C programmable digital systems are growing in complexity at a rapid pace while system designers, project architects, and cyber-security engineers work tirelessly to ensure the safety of the systems by complying with long lists of rules and regulations dictated by relevant regional & international standards [20; BQB17]. These standards are updated, withdrawn, replaced by a revised edition, and amended fairly frequently. Guaranteeing new and existing I&C system assets are still compliant is arduous, expensive, and time-consuming. In this paper, we propose an approach that assists in security modeling and system design by formulating the security controls and I&C assets in PDDL domain (Planning Domain Definition Language by D. McDermott et al. [D 98]). A domain-independent general purpose planner can explore the *state space* and provide a *deterministic plan* that transforms the *initial state* into the *goal state*. The *initial state* in this context can be the security threats from which the system needs protection. The *goal state* can be reaching a specific security degree (S1, S2, S3), satisfying risk management requirements, availability requirements, performance requirements, or a combination of them Tellabi et al. [Te18].

Keywords: Security Controls; I&C System Assets; AI Planning; PDDL; First-order Logic

1 Motivation

> "When you are writing a document for a human being to understand, a human being will look at it and nod his head: 'Yeah, this makes sense'. But then there are all kinds of ambiguities and vagueness that you do not realize until you try to put it into a computer." — Donald Knuth

2 Background

PDDL has gone a long way in recent years and is being incorporated in numerous fields (e. g. penetration testing, security assessment, robot mapping, urban planning, and traffic control). Mainly due to it being the lingua franca for the International Planning Competition[4], and as a result, there is a rich eco-system of tools and services that help the modelers and practitioners to define tasks in a declarative manner intuitively.

[1] Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau, Germany khalaf@cs.uni-freiburg.de
[2] Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau, Germany lpeters@informatik.uni-freiburg.de
[3] Framatome GmbH, Erlangen, Germany Karl.Waedt@framatome.com
[4] https://www.icaps-conference.org/competitions/

doi:10.18420/inf2022_131

3 Planning Semantics

Here we will primarily focus on using *Fast Downward* Helmert [He06]. *Fast Downward* is a domain-independent classical planning system and considered to be the state-of-the-art planner, usable both for research and practical applications Nebel; Mattmüller [NM19].

Fast Downward can be substituted for another PDDL planner with minimal-to-no modification to the previously-defined domain. This might be desirable to improve runtime performance or describe the premise in a more expressive syntax.

3.1 Formal Definition

Every task here is being tackled from the perspective of classical planning. Deterministic dynamics, full observability, and finite horizon. A deterministic planning task in finite-domain representation is a 4-tuple: $\Pi = \langle V, I, O, \gamma \rangle$ where V is a finite set of finite-domain state variables, I is an initial state over V, O is a finite set of finite-domain operators over V, and γ is a formula over V describing the goal states[5].

3.2 Planner Input

The `problem.pddl` file can be machine-generated. Such that objects (V) can be populated from an external source[6]. The initial state (I) of the problem statement is generated based on an aggregation of sensor data from the target environment, which maps to a list of initially true state variables. The operator can manually specify the goal state (γ).

The `domain.pddl` file for a specific standard is a one-time investment that requires efficient knowledge representation on both the security field and PDDL modeling. Ideally, cybersecurity experts and PDDL modelers work together to encode a standard into the target domain, possibly with the aid of some custom high-level constructs[7].

Each security degree should have its own `domain.pddl` file. the differences between them are mostly in the precondition of the *actions*. Lower security degrees usually have fewer preconditions. A modeler will start with the highest security degree and comment out the precondition in a new `domain.pddl` file according to each security requirement.

3.3 Plan Validation

A validator (`VAL` by Howey et al. [HLF04]) can be used to make sure that previously generated plans are still valid in case of an update to the `domain.pddl` file. For every new

[5] Multiple plans that reach the same *goal state* may exist.
[6] e. g. database or file
[7] e. g. state-space visualization, action usability detection

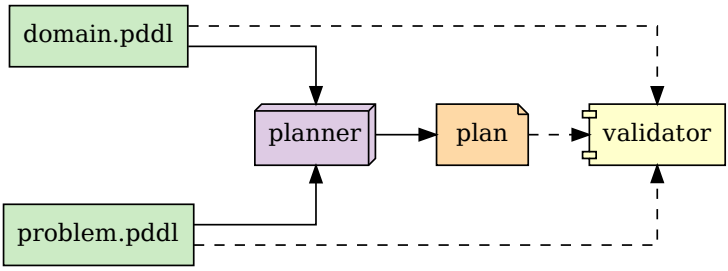

Abb. 1: Plan synthesis overview

amendment in the standard. Accordingly, the `domain.pddl` file needs to updated. The next step will be to check if the plan used to reflect the status quo is still compliant with the amends. The validator succeeds if it can reach the goal state from the initial state using the steps in the plan given the updated domain. If the validator fails, then a new plan needs to be generated from the planner and adjusted in the environment to reflect the new plan accordingly.

4 PDDL Encoding

Before starting any planning task; types, predicates and *actions* have to be defined for the planner in the `domain.pddl` file. Types should follow a reasonable hierarchy to provide logical separation and improve grounding performance. Predicates describe the current premise by accepting zero or more typed parameters. These predicates represent the attributes and relations between their arguments. Any undeclared predicate is considered false[8]. *Actions* are the possible transitions the planner considers to reach the goal state.

4.1 Type Hierarchy

Each I&C system asset in the environment will correspond to an object in the `problem.pddl` file. For example, A computer is a `DevelopmentResource`; Therefore, any action or predicate that accepts a `DevelopmentResource` as one or more parameters; will also accept any descendants of a `DevelopmentResource` in the type hierarchy.

[8] Closed world assumption

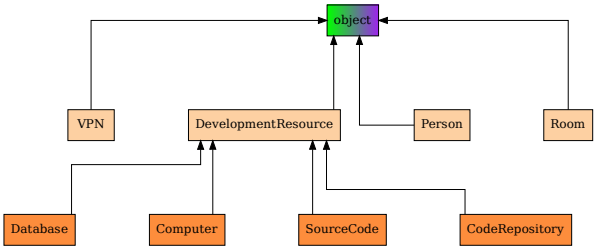

Abb. 2: Type Hierarchy
`domain.pddl`

```
(:objects
    alice bob       - Person
    mysql1          - Database
    thinkpad440     - Computer
    open_sw         - SourceCode
    gitlab          - CodeRepository
    internalIKED    - VPN
)
```

List. 1: objects
`problem.pddl`

4.2 Predicate Definition

A modeler can describe the current premise implicitly or explicitly. `allow_share_resource alice codev1` and `forbid_share_resource alice codev1` are used to tell the planner that proper countermeasures need to be taken to prevent sharing of confidential resources from an authorized party to an unauthorized party. Failing to define the `allow*` or `forbid*` predicates in the `problem.pddl` implies that there is a security risk of sharing confidential resources with unauthorized parties. Moreover, the modeler needs to allow or forbid it explicitly. Otherwise, the modeler might have accidentally overlooked a potential security risk, and the planner will act as a reminder.

In other cases, the `forbid` can be implicit. for example `allow_room_access bob controlroom`. all personnel defined in the `problem.pddl` file don't have access to the control room except bob, and there is no need to explicitly `forbid` each and every individual.

4.3 Transitive Relations

It is a matter of time before running into a situation where the modeler might prefer encoding a transitive relation implicitly. For example, if you access a version control system, you implicitly access the source code on it. Such relation can be represented as a derived predicate (Edelkamp; Hoffmann [EH04]):

```
(:derived
  (derived_authorized_resource_access ?p - Person ?c - DevelopmentResource)
  (and
      (exists
          (?d - DevelopmentResource)
          (and
              (stored_in ?c ?d)
              (or
                  (authorized_resource_access ?p ?d)
                  (derived_authorized_resource_access ?p ?d)))))
)
```

List. 2: Derived Authorized Resource Access

4.4 Action Schema

Actions consist of typed parameters, preconditions, and effects. When the current state satisfies the precondition of a particular action, the planner may consider this action and execute a *state transition* by mutating the predicates in effect. This process keeps repeating until reaching the goal state or a dead end[9].

4.4.1 Blue Team Perspective

Blue teams aim to maintain the security of assets and comply with the relevant standards. An *Action* in this context can be implementing a specific security protocol, upgrading an existing asset, or eliminating a particular threat. Failure to reach the *goal state* might indicate the absence of critical security elements that need to be defined from the security catalog.

The plan in List. 4 guides the blue team to follow the proper protocol for installing or upgrading a software inside an environment operating under strict security constraints:

```
(define (problem install_upgrade_sw)
    (:domain IAEA_MRT_330)
    (:objects
        sw                          - Software
        engineer                    - Engineer
        technician                  - Technician
        licensedOperationsStaff     - LicensedOperationsStaff
        computerSecuritySpecialist  - ComputerSecuritySpecialist
        flashdrive                  - RemovableMedia
        removableMediaIssuer        - RemovableMediaIssuer
    )
    (:init
        (not (is_deliver_sw sw))
        (has_cabient_key licensedOperationsStaff)
    )
    (:goal
        (and (installed_sw sw)) )
)
```

List. 3: Install Software Protocol (`problem.pddl`)

```
0 : (deliver_sw sw engineer)
1 : (verify_hash sw technician)
2 : (test_plan sw computerSecuritySpecialist)
3 : (regression_test sw engineer)
4 : (request_media sw flashdrive engineer)
5 : (issue_media flashdrive engineer removableMediaIssuer)
6 : (transfer_sw sw technician flashdrive)
7 : (scan_sw sw technician flashdrive)
8 : (cabinet_key sw engineer licensedOperationsStaff)
9 : (unblock_cabinet engineer)
10 : (install_sw sw engineer)
11 : (functional_test sw engineer)
12 : (lock_cabinet sw engineer)
13 : (return_keys engineer licensedOperationsStaff)
14 : (scan_media flashdrive engineer)
15 : (return_media flashdrive engineer)
16 : (sanitize_media flashdrive removableMediaIssuer)
17 : (done_install_sw sw flashdrive)
```

List. 4: Install Software Protocol (plan)

4.4.2 Red Team Perspective

Red teams aim to evaluate the security of a system by identifying weaknesses and potential security threats and providing feedback to the blue team to mitigate any security risk. An *action* in this context can be exploiting a misconfigured asset, disclosing confidential resources, or elevating access to an unprivileged user. Failure to reach the *goal state* indicates that the planner cannot find[10] a plan that exploits the system's security.

[9] unsolvable task
[10] under the given assumptions

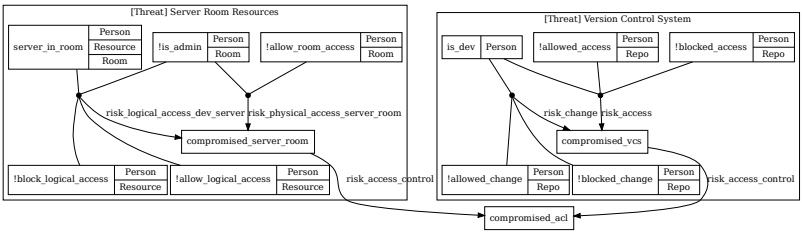

Abb. 3: Compromising Access Control (sample)

```
(:action risk__logical_access_dev_server
    :parameters (?p - Person ?s - DevelopmentServer)
    :precondition (and
        (exists (?r - Room)
            (and
                (server_in_room ?s ?r)
                (not (is_administrator ?p ?r))
                (not (block_logical_access ?p ?s))
                (not (allow_logical_access ?p ?s))
            )))
    :effect (and (compromised_server_room)))
```

List. 5: Logical Access Compromise Action

```
(:action risk__physical_access_server_room
    :parameters (?p - Person ?r - Room)
    :precondition (and

        (not (is_administrator ?p ?r))
        (not (allow_room_access ?p ?r))
        (has_room_key ?p ?r)

    )

    :effect (and (compromised_server_room)))
```

List. 6: Physical Access Compromise Action

Abb. 3 shows some possible ways a malicious actor can compromise a server room according to the standard [20]. Each node corresponds to a predicate that needs to be satisfied. Inside each node are the parameter types that the predicate accepts. Each edge is an action the planner might consider to set the next predicate to true. `compromised_acl` is a predicate that take no parameters and can be reached when either by `compromised_server_room` or `compromised_vcs` is true.

List. 5 and List. 6 are the PDDL representation of the possible ways an attacker can compromise the access control policy according to the standard[20].

In List. 8 the planner managed to detect a breach in the protocol and is warning the user that an explicit confirmation that `alice` cannot share or transfer data to `bob` is needed:

> Authorized personnel must not share or transfer the source code or application software with unauthorized individuals. — *IEC 63096; Nuclear Power Plants – Instrumentation, Control and Electrical Power Systems Security Controls* [20]

```
(define (problem attack_acl)
  (:domain IEC_63096_ACCESSCONTROL_9112_S1)
  (:objects
      alice bob - Person
      mysql1 - Database
      thinkedge450 - Computer
      open_sw - SourceCode
      gitlab - SourceCodeRepository
      internalIKED - VPN
  )
  (:init
      (has_2FA internalIKED)
      (authorized_resource_access alice mysql1)
      (authorized_resource_access bob mysql1)
      (forbidden_resource_access alice thinkedge450)
      (authorized_resource_access bob thinkedge450)
      (authorized_resource thinkedge450)
      (stored_in gitlab thinkedge450)
      (stored_in open_sw gitlab)
      (stored_in mysql1 thinkedge450)
  )
  (:goal
      (and (compromised_access_control)))
)
```

List. 7: Access Control Analysis (`problem.pddl`)

```
0 : (risk__resource_share_transfer alice bob thinkedge450)
1 : (risk__access_control)
```

List. 8: Risk Data Leak (plan)

The difference between the planner output in the case of the red team vs. in the case of the blue team reflects the asymmetric nature of cyber warfare where the defender needs to do everything right; in contrast, the attacker only needs to succeed once.

5 Related Work

There is no shortage of literature that focus on specialized tasks from the red team point of view: Automate attacks against networks with hundreds of machines Boddy et al. [Bo05], Obes et al. [OSR13] and Ron Alford et al. [Ro22]. From the blue team point of view: Role-based access control optimization Benedetti; Mori [BM19].

A successful attempt[11] has been made to port *Fast Downward* to WebAssembly (`WASM`) using Emscripten by Tran et al. [Tr20] with a small performance penalty[12]. This significantly reduces the need for server-side setup, facilitates integration with other modern web-based tools, and improves data privacy considerably as the planning process and output never leave the user's browser.

6 Summary

The proposed approach is flexible enough to be utilized by red and blue teams to secure industrial automation and control systems (IACS) and comply with the most recent version of any particular standard. Red teams can uncover potential threats on an organizational level, and blue teams check a high-level guide before introducing a new system asset to the environment.

[11] `https://project.cispa.io/fd-in-browser/`
[12] depending on the web browser

Literatur

[20] IEC 63096; Nuclear Power Plants – Instrumentation, Control and Electrical Power Systems Security Controls. International Electrotechnical Committee, 2020, ISBN: 978-2-8322-8837-5.

[BM19] Benedetti, M.; Mori, M.: On the Use of Max-SAT and PDDL in RBAC Maintenance. Cybersecurity 2/1, S. 19, Dez. 2019, ISSN: 2523-3246.

[Bo05] Boddy, M.; Gohde, J.; Haigh, T.; Harp, S.: Course of Action Generation for Cyber Security Using Classical Planning./, S. 10, 2005.

[BQB17] Bochtler, J.; Quinn, E. L.; Bajramovic, E.: Development of a New IEC Standard on Cybersecurity Controls for I&C in Nuclear Power Plants – IEC 63096./, S. 11, Juni 2017.

[D 98] D. McDermott; M. Ghallab; A. Howe; Craig A. Knoblock; A. Ram; M. Veloso; Daniel S. Weld; D. Wilkins: PDDL - The Planning Domain Definition Language./ , 1998.

[EH04] Edelkamp, S.; Hoffmann, J.: PDDL2. 2: The Language for the Classical Part of the 4th International Planning Competition./Technical Report 195, University of Freiburg, Jan. 2004.

[He06] Helmert, M.: The Fast Downward Planning System. Journal of Artificial Intelligence Research 26/, S. 191–246, Juli 2006, ISSN: 1076-9757, arXiv: 1109.6051 [cs].

[HLF04] Howey, R.; Long, D.; Fox, M.: VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In: 16th IEEE International Conference on Tools with Artificial Intelligence. IEEE Comput. Soc, Boca Raton, FL, USA, S. 294–301, 2004, ISBN: 978-0-7695-2236-4.

[NM19] Nebel, B.; Mattmüller, R.: Principles of AI Planning - PDDL, Albert-Ludwigs-Universität Freiburg, Nov. 2019.

[OSR13] Obes, J. L.; Sarraute, C.; Richarte, G.: Attack Planning in the Real World./, S. 8, Juni 2013.

[Ro22] Ron Alford; Lukas Chrpa; Mauro Vallati; Andy Applebaum: Knowledge Reformulation and Deception as a Defense Against Automated Cyber Adversaries. Proceedings of the ... International Florida Artificial Intelligence Research Society Conference 35/, Mai 2022.

[Te18] Tellabi, A.; Zid, I. B.; Bajramovic, E.; Waedt, K.: Safety, Cybersecurity and Interoperability Aspects in Modern Nuclear Power Plants./, S. 12, 2018.

[Tr20] Tran, N.; Speicher, P.; Kuennemann, R.; Backes, M.; Hoffmann, J.; Torralba, A.: Planning in the Browser./, S. 3, 2020.