Operational Design for Advanced Persistent Threats

Tithnara Nicolas Sun
Lab STICC UMR6285,
ENSTA Bretagne
Brest, France
tithnara.sun@ensta-bretagne.org

Ciprian Teodorov
Lab STICC UMR6285,
ENSTA Bretagne
Brest, France
ciprian.teodorov@ensta-bretagne.fr

Luka Le Roux
Lab STICC UMR6285,
ENSTA Bretagne
Brest, France
luka.leroux@ensta-bretagne.fr

Abstract

The *Advanced Persistent Threats* (APT) are sophisticated and well-resourced attacks targeting valuable assets. For APTs both the attack and the defense require advanced planning and strategies similar to military operations. The existing cyber-security-aware methodologies achieve valuable results for regular cyber-threats, however they fail to adequately address APTs. The armed forces around the world use the Operational Design methodology to plan actionable strategies for achieving their military objectives. However, this conceptual methodology lacks the tools and the automation needed to scale to the complexity of todays advanced persistent cyber-attacks. In this paper we propose a tool-supported Operational Design-based methodology for cyberspace mission planning. Our approach relies on a structural modeling language, used by the French armed forces, that is extended with behavioral specifications for modeling the operational situation. The APT objectives are captured through temporal logic specifications. The expert is assisted by model-checking tools to perform the typical capacity-based operation design. The approach is illustrated by studying a mission on a water pumping station. After capturing its partial understanding of the system, the attacker formalizes the mission objectives and explores the design space defined around its five operational capabilities.

CCS Concepts: • **Security and privacy** → *Formal security models*; **Systems security**.

Keywords: advanced persistent threat, cyber-security, model-checking

ACM Reference Format:
Tithnara Nicolas Sun, Ciprian Teodorov, and Luka Le Roux. 2020. Operational Design for Advanced Persistent Threats. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion), October 18–23, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3417990.3420044

1 Introduction

Cyber-physical systems are the lifeblood of todays industrial, military, and civilian infrastructures. As the degree of automation and connectivity increased, these systems became vulnerable to cyber attacks. Moreover, armed forces around the world recognise the importance of the cyberspace as an operational environment to leverage in their missions. *Advanced Persistent Threats* (APT) emerged, in this context, as sophisticated and well-resourced attacks targeting specific valuable assets [4]. These attacks target long running systems, on which neither the attacker nor the defender have complete knowledge. Thus, they require advanced planning, strategies and coordination similar to military operations.

Operational Design [7] is a conceptual methodology largely used in the military context to plan the mission strategy (the operational approach - in military terminology). From a systemic point of view, operational design can be seen as similar to systems engineering, in a context where : *a)* the operational environment, besides not being entirely known/understood, can change drastically; *b)* the solution (the operational approach) highly depends on the available capabilities; *c)* the preferred solution focuses on stealthy influences on the existing environment, instead of the introduction of a new "system" in the environment.

Extensively used in traditional warfare, the Operational Design methodology can be adapted to address cyber operations planning [10], either offensive or defensive. Moreover, Operational Design is better suited for addressing APT than the traditional risk-based systems design methodologies, which suppose a holistic point of view, and selectively address high-risk threats only. The "Direction Générale de l'Armement" (DGA) of the French ministry of armed forces created a language and methodology for cyberspace mission planning, named Pimca [15]. This language provides a rich relation vocabulary, which enables abstract structural modeling of the operational environment to ease the attack surface analysis. Despite being already used operationally, the Pimca methodology lacks the tools and the automation needed to scale to the complexity of todays advanced persistent cyber-attacks.

In this paper we propose a tool-supported Operational Design-based methodology for cyberspace mission planning. Our approach extends the Pimca framework with behavioral specifications for modeling the operational environment dynamics. Using the dynamic Pimca (DyPimca) specification as the current operational environment, we frame the Operational Design problem as a model-checking analysis. The desired operational environment, capturing the mission objectives, are formalised as temporal logic specifications. The design process itself becomes an iterative approach where the planner alters the model, based on the available capabilities, until the mission objectives are satisfied.

To illustrate our approach we consider a mission on a water pumping station. The structural viewpoint is created using the Pimca framework, the behavioral viewpoint is captured using a guarded-command language. The mission goals are captured using Linear Temporal Logic (LTL). The iterative nature of the process is emphasized by incrementally considering the attacker capabilities.

Section 2 discusses APTs, overviews the Pimca formalism, and introduces the Operational Design methodology. Section 3 introduces the methodology and the formal framework needed. Section 4 presents the case-study. Section 5 concludes this paper discussing some future research directions.

2 Background & Related Works

As cyber security threats on the industry grow increasingly sophisticated, conventional security approaches tend to become less relevant. In this paper, we propose a new point-of-view on such threats. We argue that complex cyber threats can be modeled as strategist envisioning sophisticated operations using military methodology.

2.1 Addressing APTs

Our context requires forward planning and strategy conception. This relates to cyber security advanced persistent threats. The US National Institute of Standards and Technology [13] defines an Advanced Persistent Threat (APT) as : "An adversary that possesses sophisticated levels of expertise and significant resources which allow it to create opportunities to achieve its objectives by using multiple attack vectors (e.g., cyber, physical, and deception). These objectives typically include establishing and extending footholds within the information technology infrastructure of the targeted organizations for purposes of exfiltrating information, undermining or impeding critical aspects of a mission, program, or organization; or positioning itself to carry out these objectives in the future. The advanced persistent threat: (i) pursues its objectives repeatedly over an extended period of time; (ii) adapts to defenders' efforts to resist it; and (iii) is determined to maintain the level of interaction needed to execute its objectives".

Several instances of APTs in a wide range of industrial domains have been reported [4, 5]. From the infamous Stuxnet [11] to political espionage [12], these domains include but are not limited to shipping, finance, energy, water. Thus APTs pose a significant threat to any kind of industrial systems and must be considered in security analysis.

As opposed to regular threats, APTs exhibits four distinguishing features identified by Chen et al.[4]: (i) specific targets and clear objectives; (ii) highly organized and well-resourced attackers; (iii) a long-term campaign with repeated attempts; (iv) stealthy and evasive attack techniques. Chen et al.[4] specify the cyber kill chain for typical ATP attacks: (1) reconnaissance and weaponization; (2) delivery; (3) initial intrusion; (4) command and control; (5) lateral movement; (6) data exfiltration.

Due to the complexity of such threats, traditional defense mechanisms are insufficient. Some security approaches have been proposed to address APTs [4, 8, 20, 21], but all can agree that there is no single solution to the problem. APTs require multi-faceted responses. While many security approaches tackle specific phases of the cyber kill chain as advised by Brewer [2], it appears that technical solutions focus on the latter phases of the kill chain. Indeed the early *"reconnaissance and weaponization"* stage differs in nature with the others stages, it involves the attacker gathering data and studying the system in order to formulate a strategy before proceeding with then more straight-forward phases. It requires to better understand the APTs' strategy as a whole.

Some approaches try to address the issue from this holistic point of view. Rass et al. [14] use a game theory approach to better understand the strategy. The approach may identify specific weaknesses in the system architecture through a game between attackers and defenders. However it requires an extensive understanding of the targeted system to begin with. Hutchins et al. [9] propose an intelligence-driven network defense approach to understand the strategy. It exploits the repeated attempts of intrusions of APTs to identify patterns of behavior. However the approach is based on data gathered and organized over time.

We argue that to defend against APT, taking the attacker's point of view can be valuable. It avoids issues such as completeness of the system model and requirements for previous data. In this article, we showcase how our approach can model the strategy development process of an APT.

2.2 Pimca

Pimca [15] is a language dedicated to high-level, structural modeling of systems with cyber security concerns. It is intended to be used as a tool for communication amongst designers and experts, as well as to improve the definition of requirements. Pimca meta-model ensures that instances provide valuable insights on the system to perform structural cyber security analysis. This section introduces the main concepts of the language.

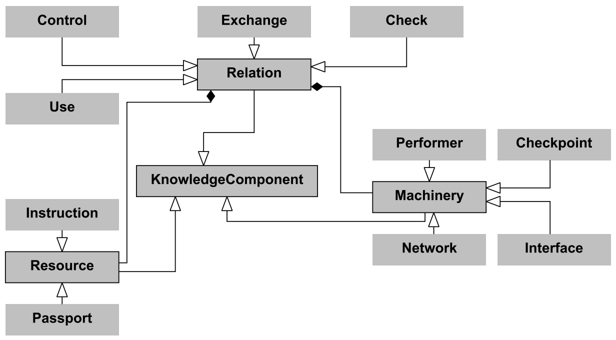

Figure 1. Pimca class diagram

Figure 1 presents an excerpt of Pimca's meta-model. A Pimca element is called a *knowledge component* and can be of three types: *1) machinery*, an active component following a specific behavior. It directly affects other knowledge component through its *relations*; *2) resource*, a passive component used to represent valuable assets that an attacker may want to specifically target; *3) relation*, an explicit link between two *knowledge components*. It is always specialized to represent the conceptual link between components, *e.g.* a physical connection, a control relation, etc.... Each of them offer various specializations that fit a cyber security context, such as the *checkpoint* machinery and the related *passport* resource. More examples are shown figure 1.

A Pimca model focuses on the structural properties of the system under study. Its strong cyber security connotation, especially regarding relations, allows for the definition of generic analysis dedicated to this context. As such, a given model's attack surface can be highlighted from various points of view depending on the possible adversaries (distant, in situ, internal threat, etc...) [15].

Pimca does not support the capture of dynamic behaviors and thus has limited use later in the system development cycle. As it will be shown sections 3 and 4, the approach presented in this paper complements Pimca and extends its usefulness throughout conception.

However, it is important to note that while our contribution is built upon Pimca concepts, introduced this section, if need be it could be bound to a different system modeling language, such as SysML.

2.3 Operational Design

Operational Design is defined in the Joint Publication (JP 5-0), Joint Operation Planning, as "the conception and construction of the framework that underpins a campaign or operation and its subsequent execution" [17]. When a commander implements the strategy, Operational Design is the highest level of implementation. It rests on operational art defined as "cognitive approach by commanders and staffs–supported by their skill, knowledge, experience, creativity, and judgment–to develop strategies, campaigns, and operations to organize and employ military forces by integrating ends, ways, and means" (JP 5-0).

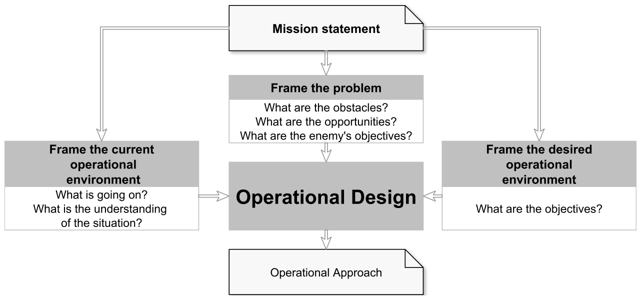
Mission statement
Frame the problem
What are the obstacles?
What are the opportunities?
What are the enemy's objectives?
Frame the current operational environment
What is going on?
What is the understanding of the situation?
Operational Design
Frame the desired operational environment
What are the objectives?
Operational Approach

Figure 2. Operational Design methodology, inspired by JP 5-0

Operational Design methodology is detailed in several military publications [3, 6, 7, 17, 19]. This high-level abstract methodology aims at developing an operational approach, *i.e.* a strategy to achieve a goal given an environment and obstacles. It revolves around three continuous concurrent and recursive processes presented in Figure 2.

- Frame the **C**urrent **O**perational **E**nvironment (**COE**).
- Frame the problem.
- Frame the **D**esired **O**perational **E**nvironment (**DOE**).

Current operational environment framing: This part is detailed in the Joint Intelligence Preparation of the Operational Environment (JIPOE) process [18]. It revolves around the identification and characterization of the operational area. This includes the previously acquired intelligence, information, hypothesis, limits and gaps in knowledge. Furthermore the commander fixes the level of granularity of the approach depending on the feasibility and time available. Finally they may submit requests for information to support further analysis. The mission objective is similarly analyzed and captured by framing the **desired operational environment**.

Problem framing: This part revolves around understanding and reviewing tendencies and potential actions of all relevant actors in the environment. This helps finding the root causes that prevents the Operational Environment (OE) from reaching the desired state. Problem framing requires several aspects of understanding: determine the strategic context and systemic nature of the problem(s), synthesize strategic guidance, identify strategic trends, identify gaps in knowledge and assumptions about the problem(s) and identify the operational problem(s) [16].

Operational approach: This part describes how the COE should be changed to the desired end state. It describes the mission task-by-task: who, what, when, where, and why. [16] The Operational Approach is heavily dependent on the Operational Design framework. Thus the approach can take

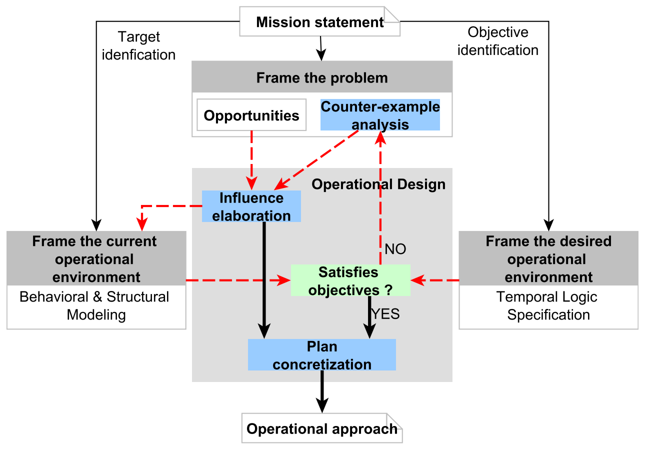

Figure 3. Approach Overview.

many forms from graphical to textual but clarity remains of the utmost importance.

Operational Design and military planning as a whole is relevant to consider in the context of APTs. Given the fact that such threats are well-organised and resourceful, APTs have the means and expertise to design complex strategy to reach their targets. In other words, APTs methodology to attack is akin to operational design methodology. Therefore, Operational Design can offer a new significant perspective in security modeling in the context of APTs.

The Pimca language was created by the DGA to enable the use of the Operational Design methodology in the context of cyberspace operations planning (both defense and offense). However, currently the Pimca formalism lacks behavioral specifications, which limits its use. This paper solves this problem. Furthermore, the methodology proposed here eases the operational design process by automating goal satisfaction through model-checking.

3 Operational Design for Cyberspace Missing Planning

This section introduces an Operational Design methodology for cyberspace operations planning, which uses model-checking for checking if the influences integrated in the COE satisfy the DOE. This approach is based on a behavioral extension of the structural Pimca formalism, which is described in section 3.2.

3.1 Operational Design Methodology

The purpose of our methodology is to offer the cyberspace operation designer a high-level environment through which they can abstractly capture the mission characteristics, design the needed operational influences on the current environment, and quickly check if the operational opportunities available suffice to achieve the desired effects in the OE.

Figure 3 overviews our Operational Design methodology. The starting point of the Operational Design process is the reception of a high-level mission statement, which should be "transformed" into an realistic actionable strategy within the bound of the available military capabilities. The first steps proposed by our methodology match standard Operational Design practice. The mission statement is analyzed to identify three different aspects of the mission: *a)* the target, abstractly representing the perimeter of the mission; *b)* the objective identification, representing the goal of the mission; *c)* the problem, representing the elements which prohibits the target from achieving the goal. Trivially, the process will stop if, after this preliminary analysis, the designer concludes that the COE already possess the needed characteristics for achieving the objectives.

Once the target is identified, we capture the structural and behavioral viewpoints of the COE (left box in Figure 3). The structural viewpoint correspond to the interacting actors in the OE, which are in the perimeter of the identified target. Standard Operational Design methodologies tend to rely exclusively on this structural viewpoint. In our case, the behavioral viewpoint complements the understanding of the actors interactions by abstractly specifying their behavior as state-machines. For framing the OE the designer relies on the available intelligence reports on the target. These reports can be incomplete, contradictory, or even wrong. The use of a tool supported modeling language for capturing the target environment eases the designers work, helping her to identify potential inconsistencies. Through model manipulation tools, such as simulation, debug, test, the designer along with her colleagues can better share their understanding of the mission target.

To formalize the DOE (right box in Figure 3), which captures the mission goals, we propose using temporal logic specifications. Similar to the target modeling, temporal logic specifications can serve as a golden model, a high-level understanding of the behavior of the DOE. Moreover, through different refinement mappings [1] (refinement of the propositional variables), the temporal specifications can be bound to different target operational environment models. In this paper we use LTL for specifying the mission goals.

The problem is framed as a set of tensions between the COE and the DOE ("Frame the problem" in Figure 3). The designer knowledge of the available capabilities and the experience guide her towards the identifications of some opportunities, which can be exploited to push the COE towards the mission goals. In our proposal, the problem framing process is complemented with counter-examples obtained by model-checking the COE against the DOE specifications. These counter-examples objectively capture the behaviors which are detrimental to the mission objective.

Our methodology identifies three important processes central to the operational design (middle gray box in Figure 3): *1)* Objective satisfaction, represents the process of checking if the current operational environment satisfies the mission goals. If the desired operational environment is not satisfied, the counter-example produces feeds the problem framing process. In our case this process is automated through model-checking, which produces behavioral counter-example traces, describing the sequence of steps that lead to failure. On the other hand, if the objective is satisfied, the plan concretization process can be started. *2)* The influence elaboration task uses the opportunities identified by the designer and the counter-examples produced by model-checking to devise the operations to be applied on the COE to achieve the mission goals. In our case, the influence elaboration process acts on the behavioral specification of the target by adding, removing or modifying its behavior as needed. *3)* The plan concretization process is enabled when the mission objectives are satisfied by the "influenced" COE. The purpose of this process is to map the influences, applied on the COE, to the concrete actions of the operational approach.

3.2 Behavioral Modeling And Verification

This section introduces the underlying formal framework that supports the methodology described section 3.1. A classic guarded-command (GC) meta-model is introduced as the basic building blocks to capture behaviors. Then, model elements are linked to Pimca nodes as an anchor for a behavioral model to its structural counterpart.

A guarded-commands model is a tuple $(\mathbb{V}, \mathbb{A}, \mathbb{S})$ where $\mathbb{V}$ is a set of variables, $\mathbb{A}$ is a set of guarded-commands and $\mathbb{S}$ is a set of synchronisation channels. In the following, the set of possible valuations over $\mathbb{V}$ is denoted $val_{\mathbb{V}}$. A guarded-command is a tuple $(u : \mathbb{B}, s : \mathbb{S} \cup \{\textbf{none}\}, g : val_{\mathbb{V}} \to \mathbb{B}, c : val_{\mathbb{V}} \to val_{\mathbb{V}})$ where u is a boolean denoting if the rule is urgent or not, s denotes its synchronisation channel (or lack thereof), g denotes its guard (boolean expression) and c denotes its command (statement). A synchronisation channel is a tuple $(d : \{\textbf{in}, \textbf{out}\}, id)$ where d denotes its *direction* and id is an unique identifier.

Illustrations can be found section 4. The concrete syntax of a guard-command used then is as follows:

```
GC_name :
    urgent ?
    ( channel ( ? | ! ) ) ?
    [guard] ? /
    (command ;) *
```

While the guards (boolean expressions) and commands then illustrated are straight-forward, the following focuses on clarifying *urgent* and synchronisation semantics.

Intuitively, if conditions are met for any urgent guarded-command in $\mathbb{A}$, all non-urgent ones are not available for execution. In the following, the term $hasUrgent_{\mathbb{A}} : val_{\mathbb{V}} \to \mathbb{B}$ is introduced to denotes such a case. Formally, $\forall \rho \in val_{\mathbb{V}}$:

$$\begin{aligned} & hasUrgent_{\mathbb{A}}(\rho) \Leftrightarrow \\ & \quad \exists (u, \textbf{none}, g, \_) \in \mathbb{A}, u \wedge g(\rho) \\ \vee & \quad \exists (u_1, (\textbf{out}, id), g_1, \_), (u_2, (\textbf{in}, id), g_2, \_) \in \mathbb{A}, \\ & \qquad (u_1 \vee u_2) \wedge g_1(\rho) \wedge g_2(\rho) \end{aligned}$$

The semantics of synchronised and urgent guarded-commands of a given $(\mathbb{V}, \mathbb{A}, \mathbb{S})$ model is presented Figure 4 as inference rules. $\langle \mathbb{A}, \rho_1 \rangle \to \rho_2$ denotes an execution step where ρ_1 and ρ_2 are the variables valuations prior and post execution. **Urgent:** Rules $single_u$ and $sync_u$ have priority over rules *single* and *sync* (the later two require $\neg hasUrgent_{\mathbb{A}}(\_)$). **Single:** Rules *single* and $single_u$ apply to guarded-commands with no synchronisation channel (**none**). Such rules are executed *alone*. **Synchronisation:** Rules *sync* and $sync_u$ apply to pairs of guarded-commands with a common synchronisation channel id but different *directions* (i.e. $(\textbf{out}, id)$ and $(\textbf{in}, id)$). It is important to note that 1. such rules can only be executed if both guards are satisfied $(g_1(\rho_1) \wedge g_2(\rho_1))$; 2. the channel outgoing command is executed before the channel incoming command $(c_2(c_1(\rho_1)) = \rho_2)$; 3. $g_2(c_1(\rho_1))$ is not evaluated, in other words guard g_2 is not required to still hold after command c_1 execution; 4. a synchronisation is considered *urgent* if any of the two guarded-commands is $(u_1 \vee u_2)$.

Bindings to Pimca nodes and relations. The guarded-commands meta-model introduced above is meant to capture behaviors for a given structural Pimca specification. As such, variables and guarded-commands are bound to nodes from the structural model (machineries). Variables referenced by a rule (guard or command) do not have to be bound to the same Pimca node, but if bound to a different node than the rule then a Pimca relation is expected to exist between the two nodes. Similarly, pairs of synchronised guarded-commands are expected to be bound to different Pimca nodes linked by a Pimca relation.

In other words, Pimca nodes act as *scopes* for behavioral elements, *global* declarations are not allowed and Pimca relations act as *dependencies* between those scopes.

Model-checking. This paper comes with its own model and language to capture the system under study. Several approaches can run a model-checking analysis on such an entry point. Among them, the **OBP model-checker** allows and encourages the definition of a *language plugin* dedicated to the user needs. Such a plugin is, arguably, easier to write and maintain than a *compiler* (model transformation) to another tool input language.

An OBP *plugin* requires the following functionalities: 1. **load** a model; 2. get the set of **initial states**; 3. get the set of **outgoing transitions** from a given source state; 4. get the set of **target states** from a given source state and a transition; 5. evaluate **atomic propositions** over a transition, its source and target states. This API is sufficient to conduct the analysis. It should be noted, however, that some care should be given to how states and transitions are displayed.

$$single_u \colon \frac{\forall (u, \textbf{none}, g, c) \in \mathbb{A}, \forall \rho_1, \rho_2 \in val_{\mathbb{V}} \quad u \wedge g(\rho_1) \wedge c(\rho_1) = \rho_2}{\langle \mathbb{A}, \rho_1 \rangle \longrightarrow \rho_2}$$

$$single \colon \frac{\forall (u, \textbf{none}, g, c) \in \mathbb{A}, \forall \rho_1, \rho_2 \in val_{\mathbb{V}} \quad \neg hasUrgent_{\mathbb{A}}(\rho_1) \wedge \neg u \wedge g(\rho_1) \wedge c(\rho_1) = \rho_2}{\langle \mathbb{A}, \rho_1 \rangle \longrightarrow \rho_2}$$

$$sync_u \colon \frac{\forall (u_1, (\textbf{out}, id), g_1, c_1), (u_2, (\textbf{in}, id), g_2, c2) \in \mathbb{A}, \forall \rho_1, \rho_2 \in val_{\mathbb{V}} \quad (u_1 \vee u_2) \wedge g_1(\rho_1) \wedge g_2(\rho_1) \wedge c_2(c_1(\rho_1)) = \rho_2}{\langle \mathbb{A}, \rho_1 \rangle \longrightarrow \rho_2}$$

$$sync \colon \frac{\forall (u_1, (\textbf{out}, id), g_1, c_1), (u_2, (\textbf{in}, id), g_2, c2) \in \mathbb{A}, \forall \rho_1, \rho_2 \in val_{\mathbb{V}} \quad \neg hasUrgent_{\mathbb{A}}(\rho_1) \wedge \neg(u_1 \vee u_2) \wedge g_1(\rho_1) \wedge g_2(\rho_1) \wedge c_2(c_1(\rho_1)) = \rho_2}{\langle \mathbb{A}, \rho_1 \rangle \longrightarrow \rho_2}$$

Figure 4. Synchronised and urgent guarded-commands semantics

Although optional, this step provides much needed feedback to the users.

In the case of this section guarded-commands model, the plugin implementation can found along the provided case-study source code linked section 4.

As for properties, OBP uses **Generic Property Specification Language**[1] (GPSL). It provides boolean and temporal operators (LTL, Buchi automata) to combine atomic propositions into higher level constructs. Those atomic propositions are to be evaluated by the language plugin. Their expressiveness and syntax can thus easily be tuned as needed.

4 Illustration: Water Pumping Station

In this section, we will showcase our approach on a case study. Due to the sensitivity of real-life military use cases, we chose to illustrate our methodology on a hypothetical realistic case of a water pumping station targeted by an Advanced Persistent Threat.

The complete case study is available at https://github.com/Lawyne/dypimca-secure-mde-2020 along with the experimental setup.

4.1 Initial Operational Environment Framing

The APT has a rough understanding of the targeted system. The system is a water pumping Supervisory Control And Data Acquisition (SCADA) plant that possess several water pumping stations. Previous intelligence operations have shown that the system uses a Programmable Logic Controller (PLC) to regulate the level of a water tank. The PLC controls a sensor to check the water level at all times and two actuators a pump and an inflow valve. In addition the outflow of the water tank is conditioned by a manual valve

[1]The full description of the GPSL language is available at http://www.obpcdl.org/properties/2019/05/09/buchi/

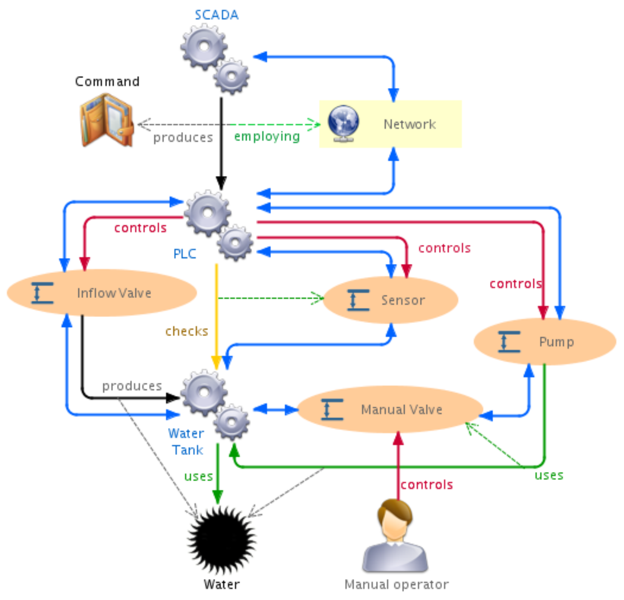

Figure 5. Supposed water pumping station structure

that is actionable by an on-site operator. The PLC follows a command given by the SCADA central through the plant network. To the APT's understanding, the command raises the water level by switching the pump off and opening the inflow valve. It lowers the water level by switching the pump on and closing the inflow valve. The command must also stipulate that the PLC regularly relays the sensor's measures to the SCADA central.

Using the knowledge of our hypothetical APT, we model the system according to our methodology. Figure 5 showcases the Pimca model of the system with a focus on a single water tank. Every element previously stipulated is represented as either a *machinery* (SCADA, network, PLC, inflow valve, sensor, pump, water tank, manual valve and operator) or a *resource* (command, water).

These elements are linked through the rich *relations* of Pimca language [15]. Physical connections are represented by *exchange* relations. These are represented by unlabelled bidirectional relations in Figure 5. More complex and richer relations are labelled. For instance, the PLC *checks* the water level of the Tank through the sensor. This is captured using a specific relation in Pimca shown in Figure 5. In our case, the objectives of the APT are: (i) overflowing a water tank; (ii) remaining undetected. Following the Operational Design methodology, the APT develops the framework for the desired operational environment.

4.2 Sub-Objective 1: Overflow the Water Tank

Mission statement: To overflow the water tank, the APT targets the physical components of the system, namely the water tank, the inflow valve, the pump and the manual valve. To the APT's understanding, the inflow valve is the only element directly responsible for raising the water level. Thus in the desired operational environment, the inflow valve must be switched on. Conversely, the pump and the manual valve are responsible for lowering the water level, so at least one among them must be closed.
Problem framing: Several elements can hinder the APT's attempts: the PLC that directly controls the inflow valve and the pump; the sensor that relays the water level to the PLC so that it can make informed decision; the operator who can open the manual valve and allow the water to flow out of the tank.
Current operational environment refinement: This analysis allows for a behavioral refinement of the current operational environment framework, *i.e.* guarded-command modeling. The APT binds behavioral elements or *execution units* to the different machineries identified in the problem framing process. These execution units are written using the guarded command semantic previously defined. Table 1 shows the different execution units and their respective guarded-commands representing the current behavioral model pertaining to the mission.

For instance, the goal of the PLC is to maintain the water level in the tank between a predefined range of a low threshold and a high threshold. To do so, it controls the inflow valve and the pump through "open" and "close" messages allowing water to flow in or out the tank. The sensor notifies the PLC of the new water level in the tank triggering the *update* guarded-command. This guarded-command forces the PLC to make an urgent decision based on the current water level. The decision has three different cases:

- If the water level if above the acceptable upper threshold, the PLC commands the inflow valve to close. It also switches the pump on so that the water level can lower overall.
- If the water level if below the acceptable lower threshold, the PLC commands the inflow valve to open. It also switches the pump off so that the water level can raise overall.
- If the water level is within the accepted range, the PLC does not command any particular behavior in the actuators.

The entirety of the guarded-command model can be found on our open repository [2]. For the purpose of this paper and its readability we use the syntax as introduced section 3.2.

Formally the PLC GC are as follows:

```
update:
    updateLevel?/
    plcWaterLevel := sensorWaterLevel;
    TriggerDecision := true;

regular:
    urgent
    [TriggerDecision ∧
    plcWaterLevel < plcUpperThreshold ∧
    plcWaterLevel > plcLowerThreshold]/
    TriggerDecision := false;

highThres:
    urgent
    [TriggerDecision ∧
    plcWaterLevel ≥ plcUpperThreshold]/
    TriggerDecision := false;
    TriggerCloseValve := true;
    TriggerOpenPump := true;

lowThres:
    urgent
    [TriggerDecision ∧
    plcWaterLevel ≤ plcLowerThreshold]/
    TriggerDecision := false;
    TriggerOpenValve := true;
    TriggerClosePump := true;

valveOff:
    urgent
    commandValveOff!
    [TriggerCloseValve]/
    TriggerCloseValve := false;

valveOn:
    urgent
    commandValveOn!
```

[2]https://github.com/Lawyne/dypimca-secure-mde-2020

```
        [TriggerOpenValve]/
        TriggerOpenValve:=false;

pumpOff:
        urgent
        commandPumpOff!
        [TriggerClosePump]/
        TriggerClosePump := false;

pumpOn:
        urgent
        commandPumpOn!
        [TriggerOpenPump]/
        TriggerOpenPump:=false;
```

The sensor follows a simpler behavior. Once the water tank changes level, it notifies the sensor through the *measure* channel. This triggers a update in the sensor which then propagates to the PLC. Formally the sensor GC are as follows:

```
update:
        measure?/
        sensorWaterLevel := value;
        TriggerSensor := true;

refreshPLC:
        urgent
        updateLevel!
        [TriggerSensor]/
        TriggerSensor := false;
```

The water tank, inflow valve, manual valve and pump are also bound to an execution unit to model their respective behaviors as shown in Table 1.

Intuitively the inflow valve can either be open or closed. When the valve is open, the *flowOut* GC can trigger and increase the water level in the tank. The *open* and *close* GC are responses to commands from the PLC.

Similarly the pump can either be open or closed through commands from the PLC. When the pump is open, the *flowIn* GC can trigger and decrease the water level in the tank, should the manual valve allow it.

The manual valve can be opened or closed by an human operator. Intuitively it is the an intermediary node between the pump and the water tank. As such it transmits the flow of water between these two elements. If the pump trigger its *flowIn* GC, the receiving GC in the manual valve is *flowOut*. The *flowOut* GC then trigger the urgent *flowIn* GC which decrease the water level in the tank.

Finally the water tank handles the evolution of its level. It has a receiving *flowIn* GC responding to the inflow valve and a receiving *flowOut* GC responding to the manual valve. Both GC change the water level and therefore they trigger the *refeshSens* GC which notifies the sensor of the new water level. Furthermore the APT adds two urgent GCs to measure specific states of the water tank. The *overflow* urgently triggers when the water level is above the tank capacity and the *underflow* urgently triggers when the tank is empty.

Note that the network and the SCADA central are bound to empty execution units. This symbolizes the fact that the APT did not frame these machineries as problems. Therefore their behaviors are not relevant to the current sub-objective. From this behavioral model, the APT is able to run a simulation of the targeted system and they can observe the water level in the tank raising and lowering periodically between two thresholds.

Desired operational environment framing: The APT's desired operational environment has an overflowing water tank. This can be expressed in LTL in the following form: $<> |waterOverflow|$ where $waterOverflow$ is a boolean variable initialized at false and becoming true whenever the water level reaches higher than $Threshold_{HIGH}$. The water tank's *overflow* GC is the only GC able to change this value.

Identified opportunities: Judging from the identified obstacles, the APT's opportunities are to force the inflow valve open, to bribe the operator into closing the manual valve or to block the pump flow. The PLC is deemed too risky to be tempered with given its inherent degree of sophistication.

These opportunities are captured through guarded-commands. Specifically, on the inflow valve, the APT generates a *forceOpen* GC in response to the APT's guarded-command to attack. Once *forceOpen* is triggered the valve can no longer be closed. Thus the APT adds a condition to the guard of the *close* GC preventing it from being triggered if *forceOpen* already has.

On the manual valve, the APT adds a condition to the guard of the *close* GC allowing it from being triggered by the bribed operator.

On the pump, similarly to the inflow valve, the APT generates a *block* GC in response to the APT's guarded-command to attack. Once *block* is triggered the pump can no longer be switched on by the PLC. To implement this aspect, the APT models two *switchOn* GCs: the regular response to the PLC command (*switchOn1*) and an altered response to the PLC command where the pump does not open (*switchOn2*). The guard of *switchOn1* is initially true and becomes false when *block* is triggered. Conversely the guard of *switchOn2* is initially false and becomes true when *block* is triggered. Splitting the *switchOn* GC in two is necessary because it is a receiving GC responding to the PLC command. The command must have a valid GC response to be triggered in order to ensure that the model works correctly.

The Operational Approach to meet the sub-objective 1 is developed via model-checking. The executable approach is formally verified using the model-checker OBP2. Table 2 presents our results. Each column represents a combination of opportunities used by the APT and their success or failure to achieve a particular sub-objective.

Water Tank	PLC	Inflow Valve	Manual Valve	Pump	Sensor	Network	SCADA
flowIn	update	flowOut	flowIn	flowIn	update		
flowOut	regular	open	flowOut	switchOn	refreshPLC		
refreshSens	highThres	close	open	switchOff			
overflow	lowThres		close				
underflow	valveOn						
	valveOff						
	pumpOn						
	pumpOff						

Table 1. Current execution units and their guard-commands in the operational environment model to overflow the water tank

Solely for the sub-objective 1, the model-checker shows that at least two opportunities have to be exploited to achieve the overflow. Forcing the inflow valve open and either closing the manual valve or blocking the pump are required. Based on this results, the APT can formulate a course-of-action to succeed in its operations. However, the APT has an other sub-objective to complete their mission.

4.3 Sub-Objective 2: Remain Undetected

The APT refines its previous operational design to take the second sub-objective into account.

Desired operational environment refinement: To remain undetected while achieving the overflow, the SCADA central must not be aware of the current state of the water level. Therefore, at least one element in the communication chain Sensor-PLC-Network-SCADA must be tempered with.

The APT's desired operational environment is constrained by a new LTL property: $[]!|scadaAlert|$ where *scadaAlert* is a boolean varuable tied to the SCADA central initialized at false and becoming true whenever a dangerous level is reported by the PLC for too long. The mission overall is constrained by the following LTL property: $<>|waterOverflow| \wedge []!|scadaAlert|$

Problem identification: Several elements are involved in the communication chain that can trigger an alert at the SCADA central, namely the Sensor, the PLC, the Network and the SCADA.

Current operational environment refinement: We take the APT's second objective into account by refining our previous operational environment model. Table 3 details the guarded-commands added to the model.

In principle, the PLC now notifies the SCADA through the network whenever it updates the water level. If *regular* is triggered, the PLC sends the *regularSig* GC into the network. Otherwise if either *highThres* or *lowThres* are triggered, the PLC sends a *dangerSig* into the network.

The network acts as an intermediary node between the PLC and the SCADA. Whenever the PLC sends a signal through a GC, the network receives it with the *receive* GC. It then trigger the urgent *send* GC to transmit the signal to the SCADA Central.

The SCADA is the final receiver in the model. Whenever the SCADA receives two dangerous signals (*dangerLvl*) in a row, it raises an alert through the urgent *alert* GC, otherwise it goes back to its normal state whenever it receives a regular signal (*regularLvl*).

Identified opportunities: According to previous reports, the PLC and the SCADA central are too risky to be tempered with. Thus the APT only has two opportunities to achieve their second sub-objective: jamming the network or disabling the sensor. The opportunities are captured through guarded-commands.

The APT injects a *jam* GC in the network that is triggered by an attack. Whenever the network is jammed, the *send* GC can not resolve. Thus the APT adds a condition to the guard of *send* so that it can only trigger when *jam* has yet to trigger.

Similarly to the previous opportunities, the APT injects a *disable* GC in the sensor, preventing the legitimate *refreshPLC* GC from triggering once *disable* has been done. Just like with the attack on the manual valve, the sensor requires *refreshPLC* to be guarded by an additional condition. Formally the new sensor's behavior is described as follows:

```
update:
    measure?/
    waterLevel := value;
    triggerSensor := true;

refreshPLC:
    urgent
    updateLevel!
    [triggerSensor ∧ !disabledSensor]/
    triggerSensor := false;

disable:
    corruptSensor?/
    disabledSensor := true;
```

The Operational Approach to meet the sub-objective 2 is developed via model-checking. Table 2 details the execution results of the model simulation and verification. Several combinations can achieve both sub-objectives, for example

Force (open) inflow valve		•			•	•			•		•	•
Close manual valve			•			•		•				•
Block pump				•	•			•			•	
Jam network							•	•	•		•	•
Disable sensor										•		
Sub-objective 1	X	X	X	X	O	O	X	X	X	O	O	O
Sub-objective 2	X	X	X	X	X	X	O	O	O	O	O	O

Table 2. Model-checking of the water pumping station (O: success, X: failure)

PLC	Network	SCADA
regularSig	receive	regularLvl
dangerSig	send	dangerLvl
		alert

Table 3. Addition to the operational environment to remain undetected

forcing the inflow valve open, closing the manual valve and jamming the network can result in a successful mission. As expected, tempering with the network or the sensor achieve the sub-objective 2. However, jamming the network does not affect the sub-objective 1, thus previously exhibited opportunities are still required to achieve both objectives. On the contrary, one particular execution case is when the APT only disables the sensor. In doing so, both sub-objectives can be achieved. Using our methodology, the APT identifies a single opportunity required to achieve the mission objectives. From this high-level system and opportunities modeling, the APT can formulate the best course-of-action.

4.4 Discussions

This case study showcases the proposed approach on a hypothetical water pumping station targeted by an APT. Following the Operational Methodology, the APT devises their strategy to stealthily overflow the water tank of the system. First they frame their current operational environment using previously gathered intelligence. In doing so, the APT builds the framework to plan for the operational approach from the mission statement. We showcase how the iterative processes interact with each other to further the APT reasoning.

Our approach revolves around the point of view of an opportunity-driven APT. An abstract model of the system is refined in iterative loops to capture those opportunities. Compared to the related works in high-level APT modeling [9, 14], we argue that our approach presents several benefits. As opposed to a hypothetical attacker who has an extensive understanding of the targeted systems structure and behavior, we propose an abstract system model that represents the partial knowledge of the attacker. As such the approach avoids exhaustivity and completeness issues raised by other works. In addition our approach is also low-investment in that extensive knowledge of the system or security data previously gathered are not required to proceed. Moreover military planning offers a insightful new perspective on APT strategy as a whole.

While our approach models an APT strategy planning, the approach does not prescribe any defensive solution. It highlights scenarios of success for the APT and consequently scenarios of failure for the system. As such, it serves as a diagnosis tool and requires linking results to more traditional security techniques. It also relies on the cyber security expert's knowledge and judgment for the quality of the analysis. We are currently working on applying the proposed methodology and tools to other case studies (a cruise-control and a distant car rental systems among others). This experience provides valuable insight to better formalize the methodological steps and address the aforementioned issues. In particular, guidelines and patterns for opportunity modeling are emerging as ground work for future contributions.

5 Conclusion

This paper presented a methodology for cyberspace operations planning in the context of APTs. Inspired by the military Operational Design methodology, our contribution proposes a model-based methodology enabling incremental design of the operational approach. The core of the approach is based on a formal model of the current operational environment and of the mission objectives. This formalization enables the automation of the goal satisfaction, through model-checking. If the mission goals are not reached, the generated counter-examples provides valuable insight to the designer for the next iteration. The approach was illustrated on a mission on a water pumping station showing that it can be valuable not only for knowledge sharing between experts, but also for systemic exploration of the design-space. Currently, we are working on designing a user study to both refine and better characterize the adequacy of our methodology to the needs of the French armed forces. In the future we plan to formalize the problem framing and the influence elaboration tasks to clearly isolate the needed "influences" from the initial formalization of the operational environment.

References

[1] Martín Abadi and Leslie Lamport. 1991. The existence of refinement mappings. *Theoretical Computer Science* 82, 2 (1991), 253 – 284. https://doi.org/10.1016/0304-3975(91)90224-P

[2] Ross Brewer. 2014. Advanced persistent threats: minimising the damage. *Network security* 2014, 4 (2014), 5–9.

[3] Canadian Joint Operations Headquarters. 2009. Systemic operational design: Freeing operational planning from the shackles of linearity.

[4] Ping Chen, Lieven Desmet, and Christophe Huygens. 2014. A Study on Advanced Persistent Threats. In *15th IFIP International Conference on Communications and Multimedia Security (CMS) (Communications and Multimedia Security, Vol. LNCS-8735)*. Springer, Aveiro, Portugal, 63–72. https://doi.org/10.1007/978-3-662-44885-4_5

[5] Michael K Daly. 2009. Advanced persistent threat. *Usenix, Nov* 4, 4 (2009), 2013–2016.

[6] Dale C Eikmeier. 2010. *Redefining the center of gravity.* Technical Report. NATIONAL DEFENSE UNIV WASHINGTON DC.

[7] Thomas Graves and Bruce E Stanley. 2013. Design and operational art: A practical approach to teaching the army design methodology. *Military Review* 93, 4 (2013), 53.

[8] Thoufique Haq, Jinjian Zhai, and Vinay K Pidathala. 2017. Advanced persistent threat (APT) detection center. US Patent 9,628,507.

[9] Eric M Hutchins, Michael J Cloppert, Rohan M Amin, et al. 2011. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research* 1, 1 (2011), 80.

[10] Muhammer Karaman, Hayrettin Catalkaya, Ahmet Zeki Gerehan, and Kerim Goztepe. 2016. Cyber operation planning and operational design. *International Journal of Cyber-Security and Digital Forensics* 5 (2020/4/22/ 2016), 21+.

[11] Ralph Langner. 2011. Stuxnet: Dissecting a Cyberwarfare Weapon. *IEEE Security Privacy* 9, 3 (2011), 49–51.

[12] Frankie Li, Anthony Lai, and Ddl. 2011. Evidence of Advanced Persistent Threat: A case study of malware for political espionage. In *2011 6th International Conference on Malicious and Unwanted Software*. IEEE, Puerto Rico, 102–109.

[13] National Institute of Standards and Technology (NIST). 2011. Managing Information Security Risk Organization, Mission, and Information System View.

[14] Stefan Rass, Sandra König, and Stefan Schauer. 2017. Defending against advanced persistent threats using game-theory. *PloS one* 12, 1 (2017), e0168675.

[15] Tithnara N. Sun, Bastien Drouot, Joël Champeau, Fahad R. Golra, Sylvain Guérin, Luka Le Roux, Raùl Mazo, Ciprian Teodorov, Lionel Van Aertryck, and Bernard L'Hostis. 2020. A Domain-Specific Modeling Framework for Attack Surface Modeling. In *Proceedings of the 6th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*. INSTICC, SciTePress, Malta, 341–348. https://doi.org/10.5220/0008916203410348

[16] US Air Force. 2016. Annex 3-0 Operations and Planning.

[17] US Joint Operation Planning. 2006. Joint Publication (JP) 5-0.

[18] US Joint Operation Planning. 2014. Joint Publication 2-01.3 Joint Intelligence Preparation of the Operational Environment (JIPOE).

[19] US Joint Staff, J and Suffolk, Virginia. 2011. 7. Planner's Handbook for Operational Design.

[20] Nikos Virvilis and Dimitris Gritzalis. 2013. The Big Four - What We Did Wrong in Advanced Persistent Threat Detection?. In *2013 International Conference on Availability, Reliability and Security*. IEEE, Germany, 248–254.

[21] Nikos Virvilis, Dimitris Gritzalis, and Theodoros Apostolopoulos. 2013. Trusted Computing vs. Advanced Persistent Threats: Can a defender win this game?. In *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*. IEEE, Italy, 396–403.