Threat-driven Dynamic Security Policies for Cyber-Physical Infrastructures

Joseph Hallett[1], Simon N. Foley[2], David Manda[1], Joseph Gardiner[1], Dimitri Jonckers[3], Wouter Joosen[3], and Awais Rashid[1]

[1] University of Bristol
[2] Norwegian University of Science and Technology
[3] KU Leuven

Abstract. How does one defend Cyber-Physical Systems (CPSs) in the face of changing security threats without continuous manual intervention? These systems form a critical part of national infrastructure and exist within an ever changing dynamic threat environment. The static security policies programmed into systems cannot react quickly to changing threats. We describe how a MAPE-K-based threat-centered dynamic policy framework could reconfigure CPSs in the face of attack. The framework encodes threats, their mitigations, and a threat posture as a mechanism for deciding how best to react. We describe our framework, a prototype implementation and show how it be integrated within a testbed CPS demonstrator. Threat centered dynamic policies allow us to harden a system as the threat assessment evolves, reconfiguring a system and how on the basis of policy and perceived threat.

1 Introduction

The large Cyber-Physical Systems (CPSs) used in Supervisory Control And Data Acquisition (SCADA) systems, power grids, water treatment and high-value manufacturing, are increasingly connected to other enterprise systems. This enhances the visibility of the physical processes and allows for improved remote monitoring, optimization and gleaning greater business intelligence from the production processes. However, security wasn't a core design feature of the software, hardware and networking protocols in these systems. Their long lifespans, legacy devices and increased connectivity, therefore, expose them to a range of attacks. In the last few years we have seen high profile attacks against the Florida's water treatment plants, the Ukrainian Power Grid and a German Steel Mill [6,27,29], which alongside attacks like Stuxnet [26,25,12] have become international news.

Attacks, such as the Triton malware [20], show that the threats against such systems are continuously evolving and becoming more sophisticated. Securing such CPSs, therefore, presents a unique set of challenges. The devices employed in them are often resource-constrained in terms of available energy, processing power and bandwidth. Critically, they have strong real-time constraints: it is not always feasible to deploy resource-intensive security policies—for example

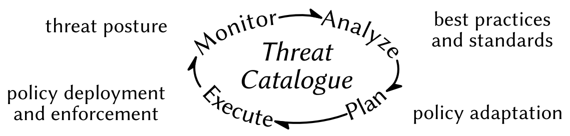

Fig. 1. MAPE-K loop for threat-driven dynamic policies.

continuous active monitoring of a Programmable Logic Controller (PLC) in a SCADA system, heavyweight cryptographic functions or authentication services all risk overloading the PLC and compromising its core monitoring and control functions. Whilst a large CPS has at any time a number of security vulnerabilities or compromised elements, it is not desirable in most cases to suspend the system's operation to fix them. For example, shutting down or restarting a power grid is a non-trivial undertaking leading to societal disruption and business losses. CPSs need to operate in a risky environment, where it is infeasible to deploy traditional mitigations all of the time.

We propose moving from traditional static policy management to *dynamic threat-centered policies*. Instead of describing the policy in terms of *what the administrator wants the system to do*, we describe it in terms of *the threats to the system and the appropriate actions to mitigate them.* Suppose that an operator becomes aware that an attacker is attempting to remotely upload malicious firmware to CPSs within a city's electrical grid: the operator wishes to disallow remote updates whilst the attack is ongoing. When there are only a few such systems in a city, the feasibility of this manual change is okay—but the number of CPS is growing: there were almost 400 electricity power stations (each with similar control systems) in New York in 2018 alone [31], and these approaches are barely feasible on a city-wide, let alone regional or national scale.

Our approach provides a *threat-centered* view on policy administration based on a MAPE-K [23] loop (Figure 1). Given a catalog of threat mitigations, when an attack occurs the approach uses the catalog to provide options for policy change—whether chosen by users in the policy administration loop, or through automation. Following the MAPE-K loop, we *Monitor* changes to the efficacy of the security policy, our threat posture and any changes in threat catalogs (*Knowledge*); *Analyze* based on best practices and standards as encoded in the policy catalog and decide whether we need to take action; *Plan* how to adapt the policy in order to mitigate the threats, and *Execute* by deploying the policy and enforcing its rules through an Event Condition Action (ECA)-based approach [28]. By managing policy knowledge in terms of threats [15,39], the threat-centered approach provides a framework in which security policies can be dynamically administered, rapidly and on-demand, in the face of unfolding attacks on a CPS.

Our contributions are as follows:

– A *threat-centered* policy framework for adapting CPS configuration on the basis of active threats, rather than a static administrator-driven policy.

– An implementation of our policy framework showing how policies can be enforced by reconfiguring a CPS testbed dynamically.

2 Threat-Centered Security Policies

2.1 Scenario

To describe our threat-centered policy framework, we first introduce a hypothetical water treatment plant to illustrate it, similar to other water treatment plants such as the one in *Oldsmar, Florida* that suffered a cyber attack in 2021 [6]. Water treatment plants take in waste water and sewage, and produce clean water. The job of the plant is to ensure that only clean water is output from the plant. If the plant fails at its task then the local area's water supply will be poisoned by sewage and chemicals. Alternatively, if the plant is shutdown then swathes of the population will be impacted by the disruption to the water supply.

The plant consists of a tank, with sensors, pumps and valves. Sensors monitor the bacterial level in the water treatment tank and filters. The pumps' rate is also controlled by a sensor and the PLC. All sensors send the raw data to a PLC which normalizes the data and stores it in a writable register. The value in the writable register is copied to the readable registers every hour. Whilst the values are written to the writable register frequently and instantaneously, the copying from the writable register to the readable register occurs only periodically as the control system is computationally limited. Faster refresh rates are possible, but have a computational and power cost, and so are generally avoided. Data from the PLC is available from an onsite Human-Machine Interface (HMI), and remotely via offsite analytics.

To attack the plant an attacker might aim to disrupt its functioning and, ideally, cause the water output to be poisoned or trigger a shutdown. Doing so would cause the plant operator to need to spend additional resources and money to protect and repair their plant. Alternatively, attacking a plant may disrupt the water supply to the local area. They might attempt this by tampering with the sensors to add more or less chlorine to the water, or by overloading the pumps and damaging the plant physically[4].

To defend against attack the plant has policies (that is, rules determining what should and is allowed to happen) in place which describe how the plant should operate and how it should react when certain threats occur. These policies can be related to functionality (e.g. the pump should run continuously), safety (e.g. if the pumps reach 80% of capacity then switch off to prevent damage), or security (e.g. only engineers can interact with the HMI)—the policies define how the plant works. They are typically enforced statically: the policies are hard coded into the devices and mechanisms within the plant. If one wanted to change the PLC's safety cut off, then the PLC would have to be physically

[4] One of the earliest known cyber attacks is on Maroochy Water Services in Australia which led to sewage spilling over a large area causing disruption and environmental damage.

reprogrammed[5]. Our goal in this paper is to introduce a policy framework that allows this reprogramming to be automated in predictable ways based on the currently identified threats.

2.2 Policy Framework

Our policy framework is based on modal logic [1], borrowing the syntax of Prolog to express rules and facts. Variables are denoted by symbols beginning with an upper-case letter, constants and predicates with a lower-case letter. A single underscore denotes a wildcard.

A policy is a collection of rules that defines whether it is permitted to engage an operation in a context. Principals assert rules, whereby ϕ`(p,`$\cdots$`)` means that principal `P` asserts logic statement ϕ, defined in terms of system attributes, holds. A description of all of the predicates allowed in our framework is given in Table 1. For example, in our water treatment plant, the operator is always allowed to shutdown the plant. This could be expressed using the policy framework by the following assertion:

```
permit(operator, shut_down).
```

Sometimes, however, an action is unsafe to do. We declare these as *threats* to an action. For example, the operator can declare that it would be unsafe to open the valve releasing water from the plant back into the system if the filters show a high level of bacteria in their output.

```
threat(operator, open_valve, safety)
  ← says(plc, _, high_bacteria_in_filter).
```

Not all threats are as important all the time—we have a *threat posture* that states what threats need to be dealt with should they arise. Some threats are always critical, for example, an operator should always care about safety.

```
posture(operator, safety).
```

Othertimes an operator's posture may be dependant on other events: for example, an operator may only care about the threat of physical tampering with a device if there are signs that the site has had a break-in recently.

```
posture(operator, tampering)
  ← threat(operator, _, break_in).
```

If there are threats to actions a principal cares about, then actions should be taken to ensure appropriate countermeasures against the threats. These actions could involve doing nothing but acknowledging the threat: for example, an operator might acknowledge that there has been a break-in.

```
mitigate(operator, _, break_in).
```

[5] More accurately a physical key would have to be turned to put the PLC into programming mode and allow any remote loading of logic into the PLC (e.g., in the Triton attack [22]).

Alternatively the operator might want to take some corrective action, for example, in the case of high levels of bacteria in the filter, the operator may want to flush the system and add additional chlorine to the tank:

```
mitigate(operator, open_valve, safety,
  [flush_system, increase_chlorine]).
```

What happens then when a principal (such as the Operator) attempts to perform an action in general. We check the default rule: an action is only permitted if all threats currently present and in the principal's threat posture have been mitigated, alongside a basic access control check (`responsible`) to check that the principal is allowed to perform that action:

$$\frac{\texttt{responsible}\,(P, A) \quad \left(\begin{array}{c}\forall T \text{ s.t. } \texttt{posture}\,(P, T) \wedge \\ \texttt{threat}\,(P, A, T) : \\ \exists\, \texttt{mitigate}\,(P, A, T, o)\end{array}\right) \quad o \in O}{\texttt{permit}\,(P, A, O)}$$

Predicate	Description
$\texttt{posture}\,(P, T)$	P believes that they consider T to be a threat (that T is in their *threat posture*).
$\texttt{threat}\,(P, A, T)$	P believes that the presence of threat T would make the action objective A impossible without taking steps to mitigate the danger.
$\texttt{mitigate}\,(P, A, T, O^*)$	P believes that the danger presented by threat T to action objective A is mitigated by completing the obligations O (that is, P says if you do O then the threat T to the action A will go away).
$\texttt{says}\,(P, P', F, O^*)$	P believes that P' (a second principal) asserts that a fact F is true, if some additional obligations O are completed.
$\texttt{permit}\,(P, A, O)$	P believes that the action objective A is safe to do if the obligations O are completed.
$\texttt{responsible}\,(P, A)$	P is responsible for deciding if action objective A is allowed
$\texttt{test}\,(E)$	The last result of the efficacy test E was successful.

Table 1. Predicates used to describe policies for CPSs. In both the `mitigate` and `says`, the obligations parameter O (marked with a star) is optional, and can be omitted if there are no additional actions that need to be performed.

A policy catalogue is created by collecting these statements and handing them to a Policy Decision Point (PDP). The PDP is responsible for analyzing the policy and deciding if actions are permitted and orchestrating any configuration changes required, based on the evaluated threats. The policy author would not typically write rules for the `permit` predicate or the `responsible` predicate, rather they are predefined on the PDP.

2.3 Policy Management

The policy catalog may change over time, as a consequence of threat reports, and principals add and/or revoke policy assertions as new knowledge about threats and controls emerges. An operator's threat posture may also change. Furthermore, it may be determined that a control has failed and, therefore, its associated threat is no longer properly mitigated in the current policy. In these cases the operator may wish to adapt the system configuration by deploying alternative controls from the catalog in order to ensure mitigation of the risk posture. Alternatively, the operator may decide to accept the threat in some form by revising their threat posture or updating the knowledge in the catalog.

The configuration of the policy itself may also need to change in response to threats. We stated earlier that the monitoring of registers in the PLC was configurable. In risk and threat management frameworks [39,37,16] , the efficacy of a control (in mitigating a threat) is monitored via procedure tests and the failure of a procedure test indicates the presence of a potential threat. Procedure tests may execute to a schedule, be triggered by system or external events, or be continually running. For example, a failing test for water quality indicates that the purification control is not effective, and that an alternate should be selected.

The term `test(ids)` is the outcome of an Intrusion Detection System (IDS) test on the consistency between the replicated PLC registers. If this test fails then there is a threat to the consistency of the system. Intrusion detection systems can experience false positives (a well-known limitation of anomaly detection systems). In reaction to this the operator may chose to add rules to automatically update the polling frequency of the test (how frequently the result of `test(ids)` is updated) when the threat is detected.

```
threat(operator, consistency, bad_register)
  ← ¬ test(ids).

mitigate(operator, consistency, bad_register,
  [increase_ids_polling_frequency]).
```

By using obligations we can control settings outside of the logic of the framework—including how frequently the framework is updated and checked. This allows the framework to not only define *security policies* which establish how to react to newly identified threats, but also *monitoring policies* that define how to check for new threats that need a reaction. In general, a threat-centered policy comprises of assertions:

```
threat(Principal, Action, Threat).

mitigate(Principal, Action, Threat, Controls)
  ← test(T).
```

Failing procedure tests correspond to attack identification: the control is no longer effective at mitigating the identified threat.

2.4 Dynamic Policies

The policy catalog provides a *declarative* definition of security. It defines *what* an acceptable policy deployment is, but it does not define *how* to change (adapt) a policy. In principle, as a result of an attack, the catalog can be searched for an optimal replacement policy. However, in practice we may wish to provide an *operational* definition of *how* the policy should change. In this case ECA [28] style operational rules are used whereby procedure test failures match the event-conditions and the triggered consequence is an action to update the policy for the prescribed control. Typically, operational rules take the form:

```
mitigate(Principle, action, threat, Controls)
  ← ¬ test(something).
```

In this case we assume there is a mechanism outside of the policy that will update the desired controls on the basis of decisions made and policy queries. This could be supported through obligations or by a dynamic policy language in which changes can be explicitly encoded in the rules [5].

Actions are the operational means to affect the system's behavior based on its policies. Threat-centered policies are often highly domain-specific, incorporating knowledge about CPS operation and the underlying controlled process. Hence, many actions will impact this process; for example, by applying a specific strategy. The policies themselves can also be changed as a result of actions, in response to failed procedure test or updated risk postures (possibly based on operator input). The deployment and updating of a policy may in turn result in further actions to be carried out.

2.5 Example

With the policy framework outlined, we illustrate how it might be used in practice. An *Engineer* maintains a water treatment plant and uses the framework to manage threats to the system. The plant has an IDS that works by monitoring a register in a PLC, as well as a guard who walks past the plant daily to check for signs of a break-in. The Engineer is concerned about the monitoring failing (`bad_register`) and a break-in to the plant (`break_in`). The policy is shown below, but can be summarized as: if the IDS is triggered, and there is no current risk to the operations of the plant, then increase monitoring; the guard can identify if there has been a break-in; and if there is a break-in, then call a repairman for the fence to mitigate the threat and until the fence is repaired be aware that there may be an extra threat of sabotage or tampering.

```
mitigate(engineer, valve_open, bad_register,
  [increase_ids_polling_frequency])
  ← ¬ test(ids),
    ¬ threat(engineer, operations, _).

threat(engineer, integrity, break_in)
  ← threat(guard, integrity, break_in).

threat(engineer, valve_open, bad_register)
  ← ¬ test(ids).

mitigate(engineer, integrity, break_in, [call_repairman, add(
    threat(engineer, operations, sabotage))]).

posture(engineer, bad_register).

posture(engineer, break_in).
```

During operation of the treatment plant, the IDS identifies that the registers in the PLC appear to be faulty, and the result of the test is updated. The system monitors the changing threats and establishes that there is a new threat to the consistency of the system and the safe operation of the valve. Since there is no other threat to the operation of the plant the system adapts by increasing the monitoring frequency of the IDS. If the IDS returns to a good state, then the fault may have been a transient one. The treatment plant is allowed to return to normal behavior once the security threat has gone. Alternatively, the IDS returning to a good state could indicate that an attacker has breached the security system and is starting to cover their tracks. In this case the operator may choose to monitor additional sensors and actuators for signs of misbehavior until the threat has passed.

Updating the security policy is not just limited to low-level technical controls but can also combine facts taken from experts on site. The water treatment plant has a guard who regularly patrols the site. The Guard reports that a hole has been cut in the fence, and updates the policy with the additional threat; changing the threat posture as the policy is monitored and analyzed. To mitigate this threat the Engineer's policy calls the repairman. The repair will take time but will eventually be fixed. Until the repair is complete, the Engineer is wary that the plant may have been tampered with, and that there may be a security risk. The Engineer believes that there is a threat of *sabotage* to the plant. Now if there is a bad register this may be another glitch or it could be a sign that the plant is being tampered with, and the output of the plant poisoned. Since this could lead to the town being supplied with dangerous water, the Engineer can modify the policy to add an additional rule that mitigates these risks. If the IDS triggers when the plant has already established that there is a threat of sabotage then the secure thing to do may be to slow the flow of water through the plant, add additional filtration and chlorination schemes and, *if all else fails*, then consider shutting the plant as a last resort.

```
mitigate(engineer, valve_open, bad_register,
  [additional_chlorination, slow_flow])
```

```
← ¬ test(ids),
   threat(engineer, operations, sabotage).
```

Our policy is reconfigured on the basis of the changing threat posture and the catalog of threats. The framework allows us to describe CPS security policies on the basis of identified threats and react to them accordingly—whether by increasing refresh rates, shutting down plants and notifying towns, or any other mitigations or security controls the policy designers and engineers might wish to describe. We do not need to hard-code our security policies. We change them dynamically as threats are identified.

2.6 Implementation

Our implementation of the threat-centred policies framework consists of the decision protocol (a principal grants all actions they are responsible for if there is an acceptable mitigation in place for all the threats they identify) and knowledge base implemented in Prolog, alongside a server to enable remote querying and database acting as an archivist. To help integrate the policy framework with a real CPS, we also implemented a custom IDS—looking at Modbus[6] communication messsages (using Scapy [7]). We also implemented a bridge between our database and Kepware—a commercial data historian used in industrial automation and Internet of Things (IoT) systems—so that policies can incorporate data from existing analytics.

The implementation has been undertaken to enable us to explore and iterate on how policies could be written and to test the framework's expressivity. If threat-centered policies are to defend CPSs, then we must be able to express the threats to these systems. Due to our focus on exploring expressivity, the current implementation has several limitations: whilst it can analyze policies, decide whether actions are permitted, and list actions and policy changes that *should* be made, it has no means to enforce them. If a policy could permit an action through multiple policy rules, then we return the first we find rather than making a decision about which is the most appropriate (or easiest to enact). Future iterations of the implementation will address these limitations as we develop it further.

3 Evaluation on a CPS Testbed

To further explore what dynamic threat-centered policies can offer we integrated it with a CPS test bed to explore whether we express policies that can defend a CPS, and how our framework would react when an attack occurs. We integrated the prototype implementation into a national large-scale cyber-physical systems testbed [19]. The testbed includes a number of scaled physical processes: a water treatment plant, a factory and a building management system as well as PLCs,

[6] Modbus is a standard PLC communication protocol.

[7] `https://scapy.net/`

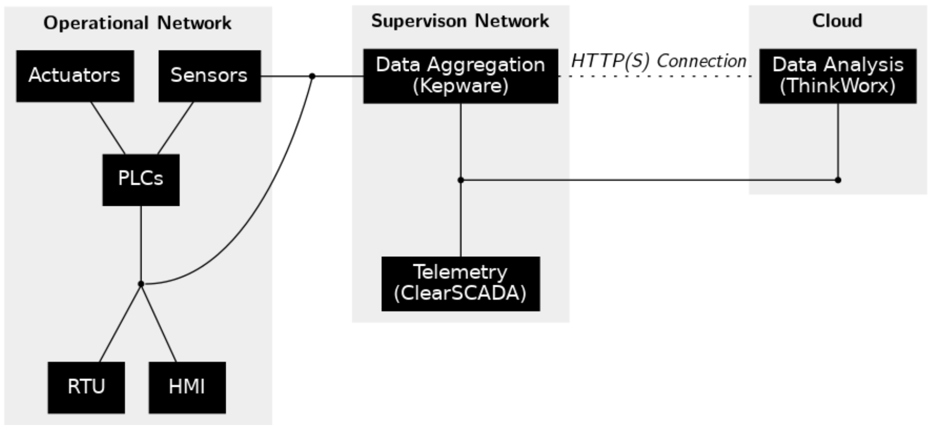

Fig. 2. Network of the CPS testbed.

RTUs[8], HMIs[9], sensors and actuators as well as software platforms from a range of vendors that can be deployed in different architectural configurations for security analyses. As well as realistically modeling CPS architectures (and physical processes), the testbed also has a range attack chains that can be run against it when evaluating various security countermeasures.

For our evaluation, we utilize the attack chain from a national IoT demonstrator that we have previously developed [35]. The demonstrator involves a factory conveyor using commercial industrial IoT control systems, and which aims to be a realistic simulation of a production Industrial Control System (ICS). The deployed configuration includes an operational network consisting of the physical process (the factory conveyor), with PLCs, sensors, an RTU and an HMI. The operational network is connected to a local supervisory network, providing data aggregation and telemetry, which is itself connected to the cloud for data analysis (Figure 2).

The attack proceeds as follows [35]:

1. The attacker compromises the Cloud-based data analytics platform via an unpatched exploit and a network takeover attack on the connection between the data aggregation and the data analysis.
2. The attacker compromises the supervision network via an engineer opening a malicious PDF (in this case, the help manual opened in response to an error displayed via the manipulated data analysis).
3. A command channel is set up through the connection between the aggregation and analysis software.
4. The attacker scans the Operational network looking for PLCs and recovers the logic from them.

[8] Remote Terminal Unit: the interface between sensors and a SCADA system.
[9] Human Machine Interface: local controls for an engineer.

5. The attacker exploits the RTU, blinding the control room from the unfolding attack.
6. The HMI and safety systems are blinded by manipulating the PLC's memory, and the PLC logic is overwritten.
7. The process is run outside of safe operational parameters, and the safety cut offs do not take effect. The process destroys itself.

Threat From	Threat To	Detected By
Possible attack start	Continued operation	IDS detects NMAP scan
Possible attack start	Continued operation	IDS detects ET200S enumeration
Possible attack start	Continued operation	IDS detects S7-1200 enumeration
Lack of analysis	Continued operation	Interrupted connection to Thingworx
PLC logic exposed	Continued operation	IDS detects PLC logic dump
Running blind	Continued operation	IDS detects RTU is offline
Running blind	PLC updates	IDS detects RTU is offline
Malicious update	Continued operation	IDS detects an unauthorized PLC logic update

Table 2. Threats presented in the attack on the demonstator.

To decide how our framework might help defend the demonstrator, we first consider the possible threats to the system, and what they might threaten (Table 2). Each of these threats is then coded up using the framework, Prolog and through querying the IDS database and Kepware. For example, one route to detecting the attack starting is letting the IDS search for signs of an S7 enumeration (a common protocol for ICSs). To detect the scan we query the database (within a time window) for signs of an S7-1200 enumeration. Similar queries are built for NMAP scans (looking for SYN scans) and ET200S enumerations. If we detect the signs of a scan or enumeration, then we establish the presence of the threat. Whether we react to it or not is controlled by the threat posture, though that is also dynamically configurable. For instance, we only start to look for signs of a malicious update to a PLC if we are aware of an attack starting:

```
posture(op, malicious_write) ←threat(op, _, attack_start).
```

Rules were written for each of the threats we identified as part of the attack and each threat presented at the appropriate point as the attack progressed.

When implementing the policy, if a check required more data or a specific technique to establish a threat (for example the network monitoring above) then we implemented this as an external tool and allowed our policy to query (and reconfigure) these tools dynamically. The expressivity of our policy language comes from the ability to combine existing (as well as new) checks and tooling around establishing threats. We don't build any new checks and tools into our framework, rather we give the ability to organize and combine the results from existing tools into threat-centered policies. By identifying threats to a real system, encoding them using our framework, and then being able to use the encoded policy to detect when a real attack is underway suggests that our framework is expressive enough to capture and encode threats to real CPS systems. In other words, if one has a mechanism to establish when a threat is present, then one can use the framework to write a rule to react appropriately.

4 Related Work

4.1 Threats to CPSs

The risks involved with connecting networked computer systems to the internet are well established. When the computer systems form part of national infrastructure, however, the risks can be particularly dangerous. These systems and infrastructures exist in a vulnerable state in many sectors: Radmand et al. noted their prevalence in the oil and gas sector [34], and Giraldo et al. noted their prevalence in many others including smart grids, hospitals, manufacturing, transport and aircraft sectors [21]. Pal et al. suggest that collaboration between threat intelligence and threat response is vital for resilient security mechanisms but that these applying these principles to CPSs is often difficult.

As the attack on the water treatment plant in Florida [6] emphasizes, the threats to these systems are not necessarily very sophisticated. In that plant the attack consisted of a brief anomalous login with stolen credentials that was noted but not acted on early in the morning, followed by another login at lunch time that increased levels of lye in the water to dangerous levels. The change was noted by an operator and remote access was disabled, but we shouldn't only rely on vigilant operators to prevent attacks. An anomalous login should have placed the plant at a higher threat level when it first happened automatically. That the attack happened in the days preceeding a major sports event should have altered the plants risk posture. The plant did use pH level sensors that would have eventually noticed the increased levels of lye, but the polling rate and sensitivity of these sensors could have been adjusted automatically in response to the increased threats as we have proposed.

In 1999 Furnell and Warren suggested that networked infrastructure could be an attractive target for terrorists and hackers and that we need to take care to avoid a scenario like the *millennium bug*—where we have to take rapid remedial action to avoid significant disruption from something that could and should have been avoided [17]. A 2012 survey of 15 SCADA and critical infrastructure incidents showed, however, that infrastructure had been continuously attacked, mostly through malware, malicious (ex-) employees and compromised accounts [29].

DiMase et al. analyzed threats to CPS resilience [10]. They noted several persistent threats but did not look at the dynamic threats that came and went periodically. Cherdantseva et al. surveyed several different mechanisms for assessing risks in CPSs [8]. They noted several frameworks which could be used to establish risks to a system and others which could be used to establish threats, however they did not look at what to do when the threats have been established. Our threat-centered policies go further by considering not just how to establish threats but also how these threats change dynamically and what to do with that knowledge.

Gao et al.'s work on Security Information and Event Management (SIEM) based policy monitoring focuses on detecting various events and how they could be chained into taking actions based on policy [18]. They give the example of how if a recently fired employee (a threat) accesses the server room (a further threat), then this ought to trigger an alert. The approach treats specific threats with specific mitigations (send alert if fired employee accesses server) rather than the threat-first approach we propose (a fired employee creates a threat of retribution, an out-of-date login system gives a threat of mistaken authorization—mitigate both by updating accounts and logging all accesses).

Knowles et al. note that industrial approaches to risk assessment could be improved, and proposed a framework for performing assessments dynamically, and producing a *measure of confidence in security* [24]. As the systems have become more distributed new attacks have appeared. Petit and Shladover identified the potential for attacks on collaborative *self-driving* cars and that the systems could be seriously disrupted if given incorrect information [33].

4.2 Policy Frameworks

To specify our policies and threats we build on ideas from access control languages. Da Silva et al. described a role-based access control scheme that adapted to user's misusing their permissions dynamically [36], and Cheng et al. described an access control scheme that included a *risk level* as part of the policy [7]. By parameterizing the policy with risk Cheng et al. could allow users greater access if there were exceptional circumstances.

Our proposal for threat-centered dynamic policies is applicable to domains other than CPS; but, in this paper, we focus on the CPS domain because it is particularly suited to this threat-centered approach as these systems are harder to maintain and more dangerous when attacked than other types of systems.

Threat-centered dynamic policies can be described as a self-adaptive system based on threats. Other work in this area includes Pandey et al.'s work on developing hybrid planning for self-adaptive systems that allows it to choose between strategies for mitigating problems based on how urgently the problem needs to be resolved [32]. Adapting a system at run time, especially in adversarial environments, can lead to the system becoming inconsistent. To help detect if this happens Barbosa et al. created a tool to verify self-adaptive systems based on a probabilistic execution model [4], while [15] catalogs firewall rule configurations, according to the threats that they mitigate, using a searchable ontology that is used to generate suitable firewall deployments. Other techniques such as IDSs can also be used to help protect CPSs by looking for intrusion events [9]. The threat-centered policies we propose are more general and can handle a greater variety of threats than just intrusions.

4.3 Self-Adaptive Systems

Whilst this work focuses on threat-centered policies it also builds upon earlier work developing self-adaptive systems for more general environments, as well as the more general application of MAPE-loops. Elkhodary and Whittle surveyed 4 self-adaptive application security approaches and developed a framework for evaluating them [11]. Tsuchida et al. used a MAPE-loop mechanism to alter an embedded system's behavior in response to its changing environment [38]. Whilst not specifically targeted at managing threats and security Tsuchida et al.'s work shows that MAPE loops can be applied effectively to reconfigure low-power devices such as embedded systems and CPSs. Arcaini et al. described a formal modeling techniques for verifying systems that use MAPE-K loops to ensure that the loops could not interfere with each other and were correct [3].

In summary, existing work has focused on how to adapt to problems as they appear. Our work takes a different approach: given a catalog of *potential* threats and possible future problems in a changing, and adversarial environment, how can we monitor, select and mitigate them automatically. By mitigating threats automatically we can avoid threats escalating, and security becoming fatally compromised. Additionally, further work is needed to empirically evaluate the benefits of a dynamic threat centered policy framework. Whilst many dynamic MAPE-based adaptive systems have been proposed for various domains; empirically demonstrating that they provide a measurable improvements is often overlooked [30]: the benefits of a threat centered policy must be proven beyond all doubt to see adoption.

5 Discussion

5.1 Threat Assessment and Policies

When exploring threat-centered policies on the demonstrator we closely tied our policy to a table of threats (Table 2). This helped tie the existing security risk assessments about the system to the policy adaptation necessary to defend the attacks as they unfold.

Most policy frameworks take a *goal-oriented* approach: an action is allowed if an explicit set of conditions is met. Policy frameworks infrequently include a notion of threats to a system [13,14,2], and it is unusual to express the policy wholly in terms of them (other applications have limited to networks and autonomous systems). This is somewhat surprising as when writing the policies it feels natural to organize a system's security policy around the threat assessment that inspired it. For CPSs in particular, where operators are resistant to dynamic configuration but comfortable with threat assessments, using threat-centered policies is a helpful stepping stone: providing mechanisms for organizing existing security controls around threats. After all we do threat modeling in terms of threats; by describing our adaptation policies in terms of threats too, we help ensure the link between them is clear, giving confidence to operators that the policy is correct and appropriate.

5.2 The Cost of Dynamism

Threat driven dynamic security policies allow us to configure CPSs to react to a changing threat landscape. The flexibility that is achieved through dynamic policies is not without cost, however. By allowing these systems to reconfigure themselves, even on the basis of a policy written by a specialist, there is a potential increase in attack surface. For example, if an attacker could see the policies that reconfigured a system then they may be able to see which checks and tests they need to trigger to put the system into a state that is beneficial for them. To illustrate this, consider again a PLC with an IDS that is dependent on register polling. The polling speed is configurable, but higher polling rates take resources from other functionality, effectively slowing the PLC's functionality for the benefit of better detecting another threat. If the attacker's goal is to make the PLC more expensive to run and they can exploit the policy to consistently force the PLC into the increased polling state then they can attack the system. Defending against these kinds of threats means still having humans *in-the-loop* to make changes, and update the policy so that automatic re-configurations reflect the current threat model.

In practice, it is no longer feasible to build static systems—and this will continue to be the case as we build more complex dynamic interconnected systems. Dynamic policies are an essential step towards better security. While the complexity of CPSs and the volume of deployed systems has gone up, we are still using static, hard-coded and even manual solutions to defend them. There is a growing awareness that existing systems are insufficiently secured. By carefully describing and testing threat-centered policies, we enable a path to more resilient systems that can react to dynamic threats. In even larger scale (massive) CPSs, such as smart city environments and automated transportation systems, the environment itself is highly dynamic—with devices connecting to each other as they traverse the CPS environment leading to dynamic service composition. In such massive scale CPS environments, static security policies can neither be feasibly constructed nor deployed as the environment is not under the control of a single stakeholder. Threat centered policies need further evaluation in these large scale environments to see if they can help bring dynamic adaptation between different systems and organizations.

6 Conclusion

Threat-centered dynamic policies let us describe policies from the perspective of threats and help cyber-physical infrastructures reconfigure themselves dynamically, on the basis of policy, to mitigate the attacks as they are identified. Existing techniques for securing CPSs do not scale and cannot react to the changing threat landscape. Threat-centered dynamic policies give us a mechanism to reconfigure the monitoring rules automatically to ensure, for example, that we make appropriate tradeoffs for security only when needed; and that we only require human intervention if the policy fails, or if new threats are presented.

Dynamic threat-centered policies, are a stepping stone towards adaptive security in such environments in order to ensure that they remain resilient in the face of unfolding threats and continue safe operation even when parts of the infrastructure are compromised. Future work will explore how combinations of different policy mitigations can be selected to mitigate all threats, and integrate cost-functions with deploying them.

Acknowledgement

CHIST-ERA project: DYPOSIT (EPSRC grants EP/N021657/1, EP/N021657/2).

References

1. Abadi, M.: Logic in access control. In: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science, 2003. pp. 228–233. IEEE, IEEE, Ottawa, Canada (2003)
2. Ahmed, M.S., Al-Shaer, E., Taibah, M.M., Abedin, M., Khan, L.: Towards autonomic risk-aware security configuration. In: IEEE Network Operations and Management Symposium. pp. 722–725. IEEE, IEEE, Piscataway, NJ (2008)
3. Arcaini, P., Riccobenne, E., Scandurra, P.: Formal design and verification of self-adaptive systems with decentralized control. ACM Transaction on Autonomous Adaptive Systems **11**(4), 25:1–25:35 (2017)
4. Barbosa, D.M., de Moura Lima, R.G., Maia, P.H.M., Junior, E.C.: Lotus@Runtime: a tool for runtime monitoring and verification of self-adaptive systems. In: IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 24–30. IEEE, Buenos Aires, Argentina (2017)
5. Becker, M.Y., Fournet, C., Gordon, A.D.: SecPAL: Design and semantics of a decentralized authorization language. Journal of Computer Security **18**(4), 619–665 (2010)
6. Cervini, J., Rubin, A., Watkins, L.: Don't drink the cyber: Extrapolating the possibilities of oldsmar's water treatment cyberattack. In: Proceedings of the 17th International Conference on Information Warfare and Security (2022)
7. Cheng, P.C., Rohatgi, P., Keser, C.: Fuzzy MLS: An experiment on quantified risk-adaptive access control. IEEE Symposium on Security and Privacy pp. 222–230 (2007)
8. Cherdantseva, Y., Burnap, P., Blyth, A., Eden, P., Jones, K., Soulsby, H., Stoddart, K.: A review of cyber security risk assessment methods for SCADA systems. Computers & security **56**, 1–27 (2016)
9. Chromik, J.J., Remke, A., Haverkort, B.R.: Bro in SCADA: dynamic intrusion detection policies based on a system model. In: 5th International Symposium for ICS & SCADA Cyber Security, ICS-CSR 2018. pp. 112–121. British Computer Society, Hamburg, Germany (2018)

10. DiMase, D., Collier, Z.A., Heffner, K., Linkov, I.: Systems engineering framework for cyber physical security and resilience. Environment Systems and Decisions **35**(2), 291–300 (2015)
11. Elkhodary, A., Whittle, J.: A survey of approaches to adaptive application security. In: Software Engineering for Adaptive and Self-Managing Systems, 2007. ICSE Workshops SEAMS'07. International Workshop on. pp. 16–16. IEEE (2007)
12. Falliere, N., Murchu, L.O., Chien, E.: W32.Stuxnet dossier. Tech. rep., Symantec Security Response (2011)
13. Fitzgerald, W.M., Neville, U., Foley, S.N.: MASON: Mobile autonomic security for network access controls. Journal of Information Security and Applications **18**(1), 14–29 (2013)
14. Foley, S.N., Fitzgerald, W.M.: An approach to security policy configuration using semantic threat graphs. In: IFIP Annual Conference on Data and Applications Security and Privacy. pp. 33–48. Springer (2009)
15. Foley, S.N., Fitzgerald, W.M.: Management of Security Policy Configuration using a Semantic Threat Graph Approach. Journal of Computer Security **3**(19) (2011)
16. Foley, S.N., Moss, H.: A risk-metric framework for enterprise risk management. IBM Journal of Research and Development **54**(3), 3 (2010). https://doi.org/10.1147/JRD.2010.2043403
17. Furnell, S.M., Warren, M.J.: Computer hacking and cyber terrorism: The real threats in the new millennium? Computers & Security **18**(1), 28–34 (1999)
18. Gao, Y., Xie, X., Parekh, M., Bajramovic, E.: SIEM: policy-based monitoring of SCADA systems. In: Informatik 2016. pp. 559–570. Gesellschaft für Informatik eV, Bremen, Germany (2016)
19. Gardiner, J., Craggs, B., Green, B., Rashid, A.: Oops i did it again: Further adventures in the land of ics security testbeds. In: Proceedings of the ACM Workshop on Cyber-Physical Systems Security & Privacy. pp. 75–86. CPS-SPC'19, ACM, New York, NY, USA (2019). https://doi.org/10.1145/3338499.3357355, `http://doi.acm.org/10.1145/3338499.3357355`
20. Gibbs, S.: Triton: hackers take out safety systems in 'watershed' attack on energy plant. The Guardian (December 2017), `https://www.theguardian.com/technology/2017/dec/15/triton-hackers-malware-attack-safety-systems-energy-plant`
21. Giraldo, J., Sarkar, E., Cardenas, A.A., Maniatakos, M., Kantarcioglu, M.: Security and privacy in cyber-physical systems: A survey of surveys. IEEE Design & Test **34**(4), 7–17 (2017)
22. Higgins, K.J.: Schneider Electric: TRITON/TRISIS attack used 0-day flaw in its safety controller system, and a RAT (2018)
23. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer **36**, 41–50 (2003)
24. Knowles, W., Prince, D., Hutchinson, D., Ferdinand, J., Disso, P., Jones, K.: Towards real-time assessment of industrial control systems (ICSs): A framework for future research. In: Proceedings of the 1st International Symposium for ICS & SCADA Cyber Security Research. pp. 106–109. Leicester, UK (2013)
25. Kushner, D.: The real story of stuxnet. IEEE Spectrum **50**(3), 48–53 (2013)
26. Langner, R.: Stuxnet: Dissecting a cyberwarfare weapon. IEEE Security & Privacy **9**(3), 49–51 (2011)
27. Lee, R.M., Assante, M.J., Conway, T.: German steel mill cyber attack. SANS, Technical Report 2014 `https://ics.sans.org/media/ICS-CPPE-case-Study-2-German-Steelworks_Facility.pdf` (2014)

28. McCarthy, D., Umeshwar, D.: The architecture of an active database management system. ACM Sigmod Record **18**(2), 215–224 (1989)
29. Miller, B., Rowe, D.: A survey SCADA of and critical infrastructure incidents. In: Proceedings of the 1st Annual conference on Research in information technology. pp. 51–56. ACM (2012)
30. Montemaggio, A., Iannucci, S., Bhowmik, T., Hamilton, J.: Designing a methodological framework for the empirical evaluation of self-protecting systems. In: 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C). pp. 218–223. IEEE (2020)
31. New York Independent System Operator, Inc: 2018 load & capacity data "gold book". Tech. rep., ISO (2018)
32. Pandey, A., Ruchkin, I., Schmerl, B., Cámara, J.: Towards a formal framework for hybrid planning in self-adaptation. In: IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 109–115. IEEE (2017)
33. Petit, J., Shladover, S.E.: Potential cyberattacks on automated vehicles. IEEE Transaction on Intelligent Transportation Systems **16**(2), 546–556 (2015)
34. Radmand, P., Talevski, A., Petersen, S., Carlsen, S.: Taxonomy of wireless sensor network cyber security attacks in the oil and gas industries. In: 2010 24th IEEE International Conference on Advanced Information Networking and Applications. pp. 949–957. IEEE (2010)
35. Rashid, A., Gardiner, J., Green, B., Craggs, B.: Everything is awesome! or is it? cyber security risks in critical infrastructure. In: International Conference on Critical Information Infrastructures Security. pp. 3–17. Springer (2019)
36. da Silva, C.E., da Silva, J.D.S., Paterson, C., Calinescu, R.: Self-adaptive role-based access control for business processes. In: IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 193–203. Buenos Aires, Argentina (2017)
37. of Sponsoring Organizations of the Treadway Commission (COSO), C.: Enterprise Risk Management-Integrated Framework. Jersey City, NJ (2004)
38. Tsuchida, S., Nakagawa, H., Tramontana, E., Fornaia, A., Tsuchiya, T.: A framework for updating functionalities based on the MAPE loop mechanism. In: 42nd IEEE International Conference on Computer Software & Applications. pp. 38–47 (2018)
39. Waltermire, D., Quinn, S., Scarfone, K., Halbardier, A.: The Technical Specification for the Security Content Automation Protocol: SCAP Version 1.2. Recommendations of the National Institute of Standards and Technology, NIST-800-126 (September 2011)