Towards Using Inductive Learning to Adapt Security Controls in Smart Homes

Kushal Ramkumar*, Wanling Cai†, John McCarthy‡, Gavin Doherty†, Bashar Nuseibeh§, Liliana Pasquale*
* Lero@University College Dublin, Ireland
† Lero@Trinity College Dublin, Ireland
‡ Lero@University College Cork, Ireland
§ The Open University, UK

***Abstract*—Smart home users often lack the technical expertise required to secure their devices and could benefit from the automated selection of security controls. In this paper, we explore the capabilities of inductive learning to adapt the requirements and system specification of a smart home system to identify security controls. We present preliminary results from using Inductive Learning via Answer Set Programming (ILASP) to learn how to produce (1) an updated system specification that enables benign behaviours while excluding malicious ones and (2) updated security requirements that the system should satisfy. We encode traces of benign and malicious execution traces from two smart home attack datasets (CICIoT2023 and IoT-23) into ILASP's language. ILASP could learn updated system specifications (to prevent DoS/Botnet attacks), new security requirements (to check for malware uploads and insecure protocols), and other integrity constraints that could be indicators of compromise. However, challenges remain when ILASP cannot perform the learning due to its sensitive syntax or complex system behaviour that lead to a large analysis space. Finally, we discuss how these limitations can be addressed in future work.**

***Index Terms*—Adaptive security, Inductive Learning, Security Controls, Smart Homes**

I. Introduction

Smart home devices, such as smart speakers and cameras, have become increasingly widespread, with millions of households worldwide integrating them into their daily lives for convenience, safety, and efficiency. However, smart homes are an attractive target for cybercriminals due to the insecurity of IoT devices, improper device configuration, and the significant value of the assets that can be harmed (e.g., sensitive information, people's safety) [1]. New vulnerabilities and threats are continually emerging, with cyberattacks exploiting these weaknesses and causing significant harm.

Users often lack the technical expertise to respond to attacks on their home networks, creating a need for automated tools to help secure evolving smart homes [2]. Identifying security controls in such evolving environments requires reasoning about newly discovered attacks and identifying updated requirements and system specifications that could mitigate them. Although traditional machine learning (ML) approaches could be used to identify security controls, they often require retraining when new attacks are discovered. Inductive learning, a subset of symbolic learning, has been used to adapt goal models [3], evolve requirements specifications [4], and learn network security policies from runtime traces [5]. There is an opportunity to use inductive learning to adapt security requirements and specifications to protect systems from cyberattacks.

In this paper, we present preliminary results from using Inductive Learning via Answer Set Programming (ILASP) to learn adapted system specifications and security requirements from smart home execution traces. ILASP learns an adapted system specification from traces that satisfy benign behaviour while excluding malicious ones. It learns security requirements predominantly from the malicious traces we want to avoid. We chose ILASP since it learns Answer Set Programs, which are expressive [6], have ample tooling support for model checking and logic-based learning [7], and are suitable for representing partial models in dynamic environments. To represent the execution traces, we considered the benign behaviour and anomalies (malicious behaviour) detected in the CICIoT2023 [8] and IoT-23 [9] smart home attack data sets. An Answer Set Programming (ASP) representation of these benign and malicious behaviours is available publicly [10]. We encode it into ILASP's language to learn updated predicates (system specification) and integrity constraints (security requirements) that we use to adapt a pre-defined smart home model and manually derive the security controls from the adapted model.

Our primary contribution lies in enabling adaptive system modelling by learning an updated system specification and security requirements and deriving security controls. Using ILASP, we learned secure system predicates (updated specification) to counteract attacks like Denial of Service (DoS) and Botnets. Additionally, we identified new security requirements, including integrity constraints preventing malware uploads and using insecure protocols. We also identified other integrity constraints that could serve as indicators of compromise (IOC) requiring further analysis, such as network packets looping back to a device. However, we faced challenges with learning system specification or security requirements from complex system behaviour, which potentially created a large analysis space for ILASP. The tool's highly sensitive syntax also required careful consideration in the specification of the system behaviour since it could lead to non-termination. Finally, we discuss how these limitations can be addressed in future work.

This research was funded in part by Science Foundation Ireland grant 13/RC/2094_P2 with additional support from UKRI.

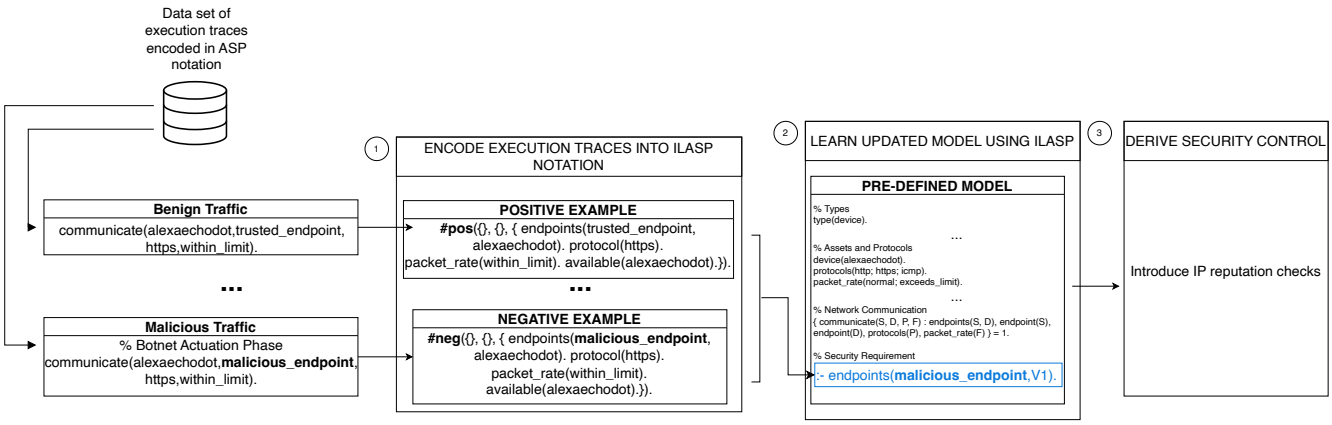

Fig. 1: Overview of the technique

II. Related Work

The aim of our work to dynamically identify security controls to mitigate new attacks aligns with existing research on adaptive security [11]–[13]. Landuyt et al. [14] suggest evolving the representation of threats based on the automated derivation of changing architectural system models from runtime and operational system artefacts. Calo et al. [15] propose an approach to dynamically generate access control policies when the environment changes, assuming that constraints associated with the resources to be accessed are satisfied. Although relevant, this work has not provided a systematic solution to identify security controls in response to new attacks [13]. Similarly to previous work on security requirements engineering [16]–[22], we reason about the satisfaction of the system security requirements to identify relevant security controls. However, this work typically requires a complete system model and has not been designed to support requirements evolution. Other approaches [23], [24] trigger requirements evolution only considering pre-defined rules when security properties are violated.

To address new attacks, previous work has suggested adapting the system specification and security requirements from execution traces indicating the presence of new attacks [25]. Inductive learning can address this need by learning logic-based system representations from examples (execution traces), supporting model adaptation. For example, a goal-oriented requirements modelling framework has been proposed to adapt requirements when environment conditions change [3]. Employing counterexample-guided learning generates goal model adaptations while ensuring correctness and minimal changes. However, the method focuses on goal refinement structures and static adaptations during development time, limiting its application to dynamic or runtime scenarios where adaptive systems require ongoing learning and revision of specifications. A probabilistic rule-learning technique, NoMPRoL, has been developed to revise behavioural models in adaptive systems based on execution traces [4]. This method effectively integrates observations to improve domain models and generate reactive plans. However, the approach assumes static feedback cycles and primarily focuses on probabilistic transitions, with limited emphasis on symbolic explainability. Recent work highlights the benefits of symbolic learning for anomaly detection, emphasising explainability and efficiency [5]. This work dynamically learns security policies from data while adapting to distribution shifts. However, this work centres on learning high-level security policies and does not extend to nuanced system specifications or requirements for diverse attack scenarios.

III. Adaptive Security with Inductive Learning

As shown in Fig. 1, our approach uses smart home execution traces extracted from the CICIoT2023 [8] and IoT-23 [9] data sets [10], which include the most commonly performed against smart homes using real and simulated devices [26]. The malicious traces were identified using anomaly detection algorithms (e.g., isolation forest) on the datasets, and were encoded using the ASP notation [26]. The traces are categorised into two types: *benign* and *malicious traffic*. The benign traffic traces represent typical network traffic, such as communication with trusted endpoints using standard protocols. The malicious traces capture potentially malicious activity, such as botnet actuation phases involving communication with malicious endpoints.

To conduct our study, we first convert the ASP-encoded traces into a format compatible with ILASP (Section III-B). Second, we use the ILASP framework to learn an adapted logic-based model from the converted traces (Section III-C). In Fig. 1, the learned output specifies rules that differentiate benign from malicious traffic, such as identifying endpoints associated with malicious activity. We then update a pre-defined smart home model (discussed in Section III-A) with the newly learned model. For example, we can learn a new security requirement that forbids network communication with a malicious source. Finally, we derive security controls (Section III-D) from the adapted model. For instance, we can use IP reputation checks to identify malicious endpoints and forbid network traffic originating from them. The steps to replicate

our results, the pre-defined model of the smart home and the ILASP encoding, are provided in the replication package[1].

A. Pre-Defined Model

The pre-defined model of a smart home is encoded in ASP and resembles the architecture of the smart homes considered in the CICIoT2023 and IoT-23 data sets. It represents the asset types (devices), network communication, and one sample security requirement. The smart home has a flat network structure where the devices connect directly to the router (Fig. 2). The attacks are simulated in the data sets using a malicious actor within the home network that communicates directly with the devices.

We define the assets of the smart home as a *type* in ASP, which is then used to define the domain of objects, such as the devices in the home. Listing 1 shows an example of an asset (i.e. an Alexa Echo Dot device).

```
% Define Types
type(device).

% Devices within a smart home
device(alexaechodot).
```

Listing 1: System Assets

We specify the network traffic using a predicate, a function that takes one or more arguments and evaluates to true or false depending on whether the arguments satisfy certain conditions. We name this predicate *communicate*, as shown in Listing 2, since it models the network communication in the smart home. The source (S) and destination (D) endpoints, protocol (P) used, and packet rate (F) are the body of the predicate, i.e, arguments that represent network traffic features. The *communicate* predicate is true only when S and D are specified as *endpoints*, P is of type *protocol*, and F is of type *packet_rate*.

```
{ communicate : endpoints(S, D), protocols(P),
    packet_rate(F) } = 1.
```

Listing 2: Predicate to model Network Traffic

The execution traces are specified by replacing the predicate's arguments with constants. Listing 3 shows an example of a communication trace from the data set used in our study.

```
communicate(alexaechodot,trusted_endpoint,https,
    within_limit).
```

Listing 3: Atom depicting Benign Network Traffic

We model the system's security requirements using integrity constraints, i.e. rules the system must satisfy. Listing 4 shows an example of a security requirement that ensures that a device is not made unavailable by a Denial-of-Service (DoS) attack. Here, *unavailable* is a predicate that takes a device type as input and represents the case where the device is not operational or no longer reachable on the network.

```
:- unavailable(S), device(S).
```

Listing 4: Integrity Constraint

[1] https://github.com/kushalramkumar/adapting-security-controls-with-inductive-learning

B. Encode Execution Traces Into ILASP Notation

To learn an updated system model, it is necessary to encode the benign and malicious traces from the dataset into the ILASP notation, also called *mode bias*. It supports the definition of constants and variables. The constants can define atoms, whereas the variables are used to learn predicates. We use them to describe the system assets to ILASP. Listing 5 shows how the *malicious_endpoint* and *alexaechodot* are constants assigned to the variable named *endpoint*.

```
#constant(endpoint, malicious_endpoint).
#constant(endpoint, alexaechodot).
```

Listing 5: Constants and Variables in ILASP

We provide positive examples that represent benign traffic to learn updated predicates. Each example consists of an inclusion list (permitted predicates), an exclusion list (excluded predicates), and a set of execution traces in ASP which specify additional context. Listing 3 shows a benign trace from the dataset. In ILASP notation, this is decomposed into endpoints (source, destination), protocol, and packet_rate, and additional contextual predicates like device availability (Listing 6). *communicate* is in the inclusion list because we would like it to appear in the head of the learned predicate.

```
#pos({communicate}, {}, { endpoints(trusted_endpoint
    , alexaechodot). protocol(https). packet_rate(
    within_limit). available(alexaechodot).}).
```

Listing 6: A positive example in ILASP

The example illustrates that the Alexa Echo Dot can receive network traffic from a trusted source using HTTPS, with packet rates within permissible limits, provided the device is available. It also shows that the *communicate* predicate is permitted under the specified context in the third argument.

The ILASP syntax of negative examples is similar to that of positive examples. A negative behaviour to prevent is the Alexa Echo Dot communicating with a malicious endpoint, encoded in ASP as shown in Listing 7.

```
communicate(malicious_endpoint,alexaechodot,https,
    within_limit). available(alexaechodot).}).
```

Listing 7: A negative example in ASP

Similar to creating positive traces, we decompose the predicate from the data set into its arguments to convert it to ILASP notation shown in Listing 8.

```
#neg({}, {}, { endpoints(malicious_endpoint,
    alexaechodot). protocol(https). packet_rate(
    within_limit). available(alexaechodot).}).
```

Listing 8: A negative example in ILASP

C. Learn Updated System Model Using ILASP

We use *mode declarations* to instruct ILASP on what we want it to learn. The body declaration, *modeb*, indicates that the object or predicate must appear in the body of the learned Answer Set Program (Listing 9). The first argument of *modeb* is the number of times the predicate can appear in the learned ASP model. We set it to 1 since the endpoints only need to

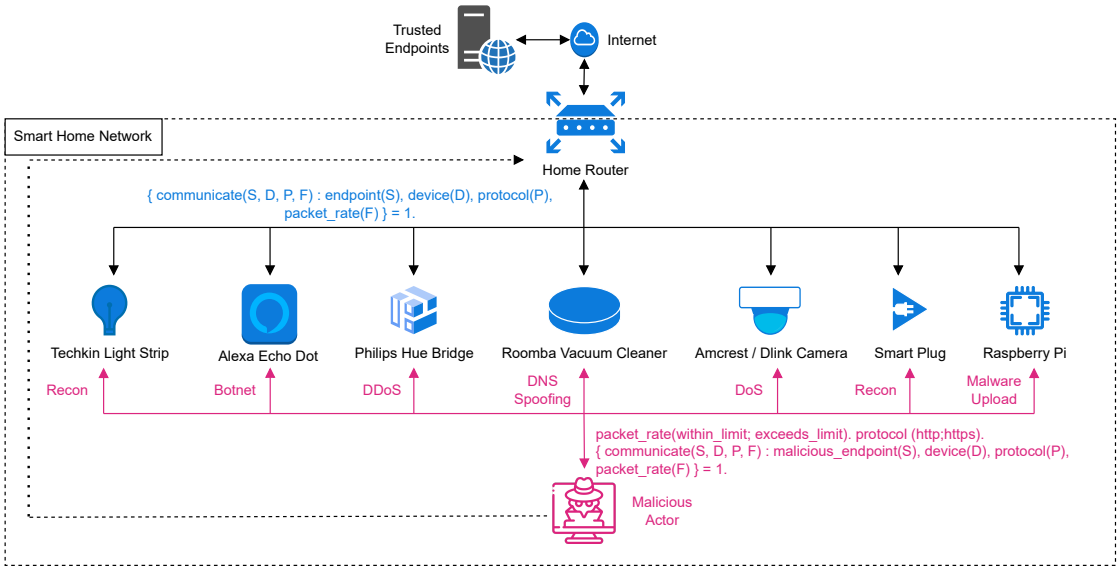

Fig. 2: Pre-Defined Model of the Smart Home

occur once. The second argument defines the composition of the predicate to be learned, i.e., whether the predicate must contain constants, variables, or a combination of both.

```
#modeb(1, endpoints(var(endpoint),var(endpoint))).
#modeb(1, endpoints(var(endpoint),const(endpoint))).
#modeb(1, endpoints(const(endpoint),var(endpoint))).
```

Listing 9: Body mode declaration

ILASP can be used to learn both predicates and integrity constraints. The predicates are useful in implementing secure system configurations (e.g., HTTPS encrypted network communication in transit). In contrast, the integrity constraints specify the security requirements that must not be violated (e.g., no communication with malicious endpoints).

Providing sufficient positive examples in the format of Listing 6 enabled us to learn an ASP predicate specifying permitted communication in the smart home (Listing 10).

```
communicate :- endpoints(V1,V2); protocol(https);
    not unavailable(V2).
```

Listing 10: Example of a learned predicate

This predicate differs from the one in Listing 2, where the *communicate* predicate's head contained variables. These differences must be addressed when updating the pre-defined model with ILASP-learned predicates. For example, knowing V1 and V2 represent the source and destination of network traffic, we manually extrapolate the updated predicate shown in Listing 11. Before updating the model, conflicts between the learned system specification, security requirements and the existing model must be resolved, as incorrect specifications could break model satisfiability. Although it is possible to perform satisfiability checking to identify conflicts between the learned integrity constraints and the pre-defined system specification, validation of the modified security requirements must be performed by a security expert.

```
{ communicate(S, D, P, F) : endpoints(S, D),
    endpoint(S), endpoint(D), protocols(P), P =
    https, not unavailable(D), packet_rate(F) } = 1.
```

Listing 11: Example of an updated predicate

Providing sufficient negative examples similar to that in Listing 8 enabled ILASP to learn a constraint preventing communication with a malicious endpoint. Constants like *malicious_endpoint* improve readability but can represent any value (e.g., IP address or domain). This approach allows ILASP to learn predicates and constraints by controlling the provided examples.

We now discuss the predicates and integrity constraints that ILASP learnt for each attack in the dataset.

Denial-of-service (DoS) and Distributed Denial-of-service (DDoS) attacks overwhelm a target with network traffic, rendering it unavailable [27]. A DoS attack originates from a single source, while a DDoS attack involves multiple sources. These attacks are detected by monitoring for unusually high packet flow rates from malicious or unverified endpoints [28]. We represent (D)DoS traces using the *endpoints*, *packet_rate*, and *unavailable* predicates (Listing 12). To learn an updated system specification that prevents such attacks, the *communicate* predicate is included in the inclusion list.

```
#neg({communicate}, {}, {unavailable(alexaechodot).
    endpoints(multiple_endpoints,alexaechodot).
    protocol(https). packet_rate(exceeds_limit).}).

% DoS attack trace
#neg({communicate}, {}, {unavailable(amcrestcamera).
     endpoints(single_endpoint,amcrestcamera).
    protocol(https). packet_rate(exceeds_limit).}).
```

Listing 12: (D)DoS Attack Traces in ILASP

The number of endpoints (*multiple_endpoints*) can be determined by analysing network traffic. Normal packet rate bounds can be estimated using statistical methods (e.g., IQR method).

Botnet attacks occur in three stages: (1) the upload phase, where malware is sent to a compromised device; (2) the actuation phase, where the device receives commands from a Command & Control (C&C) server; and (3) the execution phase, involving a DoS/DDoS attack with a flood of packets [8], [9]. Upload and actuation phases are detected by monitoring communication with known C&C servers or malicious endpoints, modelled as shown in Listing 13. To identify new security requirements preventing communication with malicious endpoints, only negative traces were provided, without any inclusion or exclusion lists. The execution phase aligns with the (D)DoS attack described earlier.

```
#neg({}, {}, { endpoints(malicious_endpoint,
    alexaechodot). protocol(https). packet_rate(
    within_limit). available(alexaechodot).}).
#neg({}, {}, { endpoints(alexaechodot,
    malicious_endpoint). protocol(https).
    packet_rate(within_limit). available(
    alexaechodot).}).
```

Listing 13: Malware Upload Attack Traces in ILASP

We label the endpoint as *malicious_endpoint*, identified in practice through IP reputation checkers [29] and C&C blacklists [30]. The IoT-23 dataset includes various botnet attacks, all characterised by the same execution phases, so we do not distinguish between botnet types in this study.

Recon attacks involve probing a system to gather information [8], often through repeated requests (e.g., TCP SYN) to observe responses. These attacks require devices to remain available and responsive. The CICIoT2023 dataset includes port scanning attacks by malicious actors with local IP addresses, making malicious IP lists ineffective. Detection relies on identifying spurious requests from unknown hosts, as specified in ILASP (Listing 14).

```
#neg({communicate}, {}, {available(amazonplug).
    endpoints(rpi,amazonplug). protocol(https).
    packet_rate(exceeds_limit).}). %Recon
```

Listing 14: Recon Attack Traces in ILASP

We could not learn updated system specifications or security requirements for these attacks. Despite providing (D)DoS and upload attack traces, distinguished only by the target device's availability, ILASP successfully identified *packet_rate* exceeding limits as a (D)DoS attack but reported an unsatisfiable model for Recon traces.

Table I summarises the updated system specifications and new security requirements generated by ILASP for each attack type. Providing sufficient traces of DoS, DDoS, and the execution phase of Botnet attacks, we learned two updated system specifications that can prevent these attacks. ILASP learnt the updated Answer Set program in Row 1 using only positive examples, showing that communication from the Alexa Echo Dot and Amazon Plug must use HTTPS and occur only when devices are available. ILASP inferred that the variable $V2$ represents the destination device since availability is specified in the traces only for the device, not the originating endpoint. However, ILASP failed to learn that *packet_rate* exceeding the limit must be disallowed, despite such examples being provided in the exclusion list (Listing 12). ILASP learnt the updated Answer Set program in Row 2 using both inclusion and exclusion lists for positive traces (Listing 15), enabling ILASP to generate multiple constraints on predicates rather than a single system specification as in Row 1.

```
% Inclusion Criteria
#pos({communicate}, {}, { endpoints(trusted_endpoint
    , alexaechodot). protocol(https). packet_rate(
    within_limit). available(alexaechodot).}).
...

%Exclusion Criteria
#pos({}, {communicate}, { endpoints(alexaechodot,
    malicious_endpoint).}).
...
```

Listing 15: Providing inclusion and exclusion criteria for the learned predicate

In these predicates, the device is identified by $V1$, indicating ILASP detected more traces where the device was the first argument. These rows illustrate how varying example types for the same attack yield different system specifications. Row 3 demonstrates a learned security requirement preventing malicious endpoints from communicating with devices, based on traces discussed in Listings 8 and 13. Row 4 shows how secure protocols like HTTPS can be identified from traces. ILASP learned that communication must use HTTPS, though this is not limited to one protocol. Here usage of HTTP indicates a misconfigured device or malware-enabling man-in-the-middle (MitM) attacks. Row 5 extends Rows 1 and 2 by learning the requirement for devices always to be available. This was achieved by providing negative examples of device unavailability from DoS attack traces. Row 6 introduces a security requirement preventing network traffic from looping back to the same device. Such traffic may be an indicator of compromise (IOC) requiring further analysis, potentially due to Recon attacks, MitM attacks, or DNS/Routing Table poisoning in the router rather than the device itself.

D. Select Security Controls

In this section, we discuss how security controls can be chosen based on the adapted system model (Table I). We discuss the security controls without implementing them.

Rows 1, 2, and 5 correspond to (D)DoS attacks or the botnet execution phase. Rows 1 and 5 share the requirement for devices to always be available. These requirements can be satisfied through periodic availability checks (e.g., network pings), monitoring live traffic, or user reports on device usability [28]. The updated specification in Row 2 requires network traffic to stay within permissible limits while ensuring device availability. This can be achieved with packet filtering alongside availability checks, using techniques to determine rate limits (e.g., IQR thresholds), detect when thresholds are exceeded, and filter packets during suspected attacks (e.g., via firewall rules) [31]. The security requirement in Row 3 addresses malware uploads and the botnet actuation phase, enforcing a hard constraint against communication with *malicious_endpoint*. This can be achieved using IP blacklists [30]

Index	What ILASP learned	Output Answer Set Program	Attack	Security Controls
1	Updated Specification	communicate :- endpoints(V1,V2); protocol(https); not unavailable(V2).	DoS/DDoS/Botnets (Exec)	Periodic device availability checks
2	Updated Specification	communicate :- packet_rate(within_limit). communicate :- endpoints(V1,V2); available(V1).	DoS/DDoS/Botnet (Exec)	Configure packet rate thresholds
3	Security Requirement	:- endpoints(malicious_endpoint,V1). :- endpoints(V1,malicious_endpoint).	Botnet (C&C), Malware Upload	Introduce IP reputation checks
4	Security Requirement	:- not protocol(https).	Device Vulnerability	Flag or prevent insecure communication
5	Security Requirement	:- unavailable(V1).	DoS	Periodic device availability checks
6	Security Requirement	:-endpoints(V1,V1).	IOC (MitM, malware, recon), DoS, or Routing Table Poising / ARP Spoofing	Diagnosing Indicator of Compromise

TABLE I: Updated System Specification and Security Requirements learned by ILAPS and Security Controls derived manually.

and IP reputation checkers [29] to block known botnet servers or malicious IPs.

The security requirement in Row 4 enforces a hard constraint against using insecure network protocols. This can be addressed by patching device vulnerabilities or securing device configurations, requiring guidance from security engineers. Router manufacturers often provide similar support that also offer access to expert communities [32]. The security requirement in Row 6 represents a constraint on the network traffic looping back to a device. This requires checking for an IOC by detecting packets looping back to a source within the network [33]. Like Row 4, Row 6 also requires diagnosis of the IOC by security engineers to determine if an attack is underway and how it can be mitigated.

IV. Discussion

Our approach is useful for dynamically learning and updating system models, whether to address evolving security requirements or respond to unknown attacks with identified symptoms but unknown root causes. A symptom could be a network execution trace indicating a successful DoS attack. Our approach could be deployed in a smart home as part of a router security solution that monitors network activity, detects anomalies, and mitigates attacks. Although we successfully learned updated system specifications and security requirements for various attacks, we encountered significant challenges in some instances.

We could not learn updated specifications or requirements for the Recon attack. Unlike DoS, which renders the device unavailable, Recon keeps it available. Despite providing additional traces with varying devices and positive and negative examples, ILASP reported the model as unsatisfiable, likely due to relying only on four network features that were insufficient to distinguish benign from anomalous behaviour. In future work, we plan to create a testbed to simulate attack-specific network traffic, generating traces with varied packet rates and additional features like inter-arrival time (IAT).

We found that ILASP has a sensitive syntax and sometimes fails to terminate. Without tool feedback, it is difficult to determine if it requires a more detailed mode bias or additional examples. While symbolic methods are more explainable than traditional machine learning techniques [34], further exploration is needed to (1) explain the learned ASP program and (2) automate security control generation from the adapted model. Large Language Models (LLMs) have been used to explain intrusion detection alerts [35]. Thus, there is also an opportunity to apply them to explain the outcome of symbolic learning techniques. Although LLMs enhance explainability, relying on them to fully automate security control generation could risk over-reliance, as cautioned by OWASP LLM Top 10 [36].

We manually created the initial system model based on common security requirements in smart homes [27]. In future work, we will learn the entire model from scratch from execution traces using neuro-symbolic approaches [5], which have been used to learn security policies from network traces. While we detailed how to specify mode bias to learn different model adaptations, security engineers replicating our approach must define mode bias to specify what to learn (e.g., system specifications or security requirements) and how to learn (e.g., constants or variables in the ASP program).

V. Concluding Remarks

In this paper, we provided our preliminary results on using inductive learning to adapt pre-defined security requirements and system specifications from ASP-encoded execution traces. We also showed how security controls can be derived from the adapted model. Our approach demonstrates several key advantages of leveraging inductive learning, making it a compelling alternative to traditional machine learning techniques. From ILASP encoding of execution traces, we learned a nuanced system specification and security requirements. ILASP does not require as many traces as traditional ML techniques to update the system requirements and specification. For example, it only needed seven examples for the DoS attacks. In addition, symbolic learning techniques use logic-based representations that are more readable and explainable than traditional machine learning approaches [34], which enabled us to derive the corresponding security controls for the system. We discussed the challenges we faced, particularly in learning attacks represented using complex system behaviours and issues with ILASP's syntax and tractability. We also provided our plans for addressing these limitations in future work.

REFERENCES

[1] J. I. I. Araya and H. Rifa-Pous, "Anomaly-Based Cyberattacks Detection for Smart Homes: A Systematic Literature Review," *Internet of Things*, vol. 22, p. 100792, 2023.

[2] E. Zeng, S. Mare, and F. Roesner, "End user security and privacy concerns with smart homes," in *Thirteenth Symp. Usable Privacy and Security (SOUPS 2017)*, 2017, pp. 65–80.

[3] D. Alrajeh, A. Cailliau, and A. van Lamsweerde, "Adapting requirements models to varying environments," in *2020 IEEE/ACM 42nd Int. Conf. Software Engineering (ICSE)*, 2020, pp. 50–61.

[4] D. Sykes, D. Corapi, J. Magee, J. Kramer, A. Russo, and K. Inoue, "Learning revised models for planning in adaptive systems," in *2013 35th Int. Conf. Software Engineering (ICSE)*, 2013, pp. 63–71.

[5] A. Drozdov, M. Law, J. Lobo, A. Russo, and M. W. Don, "Online symbolic learning of policies for explainable security," in *2021 Third IEEE Int. Conf. Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, 2021, pp. 269–278.

[6] R. Koitz-Hristov and F. Wotawa, "Applying algorithm selection to abductive diagnostic reasoning," *Applied Intelligence*, vol. 48, pp. 3976–3994, 2018.

[7] R. Kaminski, T. Schaub, and P. Wanko, "A tutorial on hybrid answer set solving with clingo," *Reasoning Web. Semantic Interoperability on the Web: 13th Int. Summer School 2017, London, UK, July 7-11, 2017, Tutorial Lectures 13*, pp. 167–203, 2017.

[8] E. C. P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, and A. A. Ghorbani, "CICIoT2023: A real-time dataset and benchmark for large-scale attacks in IoT environment," *Sensors*, vol. 13, p. 5941, 2023.

[9] S. Garcia, A. Parmisano, and M. Jose Erquiaga, "IoT-23: A labeled dataset with malicious and benign IoT network traffic (Version 1.0.0)," 2020, https://www.stratosphereips.org/datasets-iot23 [Accessed: 12.06.2023].

[10] K. Ramkumar, W. Cai, J. McCarthy, G. Doherty, B. Nuseibeh, and L. Pasquale, "More Than Zero - Diagnosing Unknown Attacks Using Abductive Reasoning [Data set]," 2024, https://doi.org/10.5281/zenodo.14380178 [Accessed: 11.12.2024].

[11] M. Salehie, L. Pasquale, I. Omoronyia, R. Ali, and B. Nuseibeh, "Requirements-driven adaptive security: Protecting variable assets at runtime," in *2012 20th IEEE Int. Requirements Engineering Conf. (RE)*, 2012, pp. 111–120.

[12] E. Yuan, N. Esfahani, and S. Malek, "A Systematic Survey of Self-Protecting Software Systems," *ACM Trans. Autonomous and Adaptive Systems (TAAS)*, vol. 8, no. 4, pp. 1–41, 2014.

[13] G. Tziakouris, R. Bahsoon, and M. A. Babar, "A Survey on Self-Adaptive Security for Large-Scale Open Environments," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, pp. 1–42, 2018.

[14] D. van Landuyt, L. Pasquale, L. Sion, and W. Joosen, "Threat Modeling at Run Time: The Case for Reflective and Adaptive Threat Management (NIER track)," in *2021 Int. Symp. on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2021, pp. 203–209.

[15] S. Calo, D. Verma, S. Chakraborty, E. Bertino, E. Lupu, and G. Cirincione, "Self-Generation of Access Control Policies," in *Proc. 23nd ACM Symp. Access Control Models and Technologies*, 2018, pp. 39–47.

[16] L. Liu, E. Yu, and J. Mylopoulos, "Security and Privacy Requirements Analysis within a Social Setting," in *Proc. 11th IEEE Int. Requirements Engineering Conf., 2003.*, 2003, pp. 151–161.

[17] A. Van Lamsweerde, "Elaborating Security Requirements by Construction of Intentional Anti-Models," in *Proc. 26th Int. Conf. Software Engineering*. IEEE, 2004, pp. 148–157.

[18] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, "Modeling Security Requirements through Ownership, Permission and Delegation," in *Proc. of the 13th Int. Conf. on Req. Eng.*, 2005, pp. 167–176.

[19] C. Haley, R. Laney, J. Moffett, and B. Nuseibeh, "Security Requirements Engineering: A Framework for Representation and Analysis," *IEEE Trans. Software Engineering*, vol. 34, pp. 133–153, 2008.

[20] Y. Yu, V. N. Franqueira, T. T. Tun, R. J. Wieringa, and B. Nuseibeh, "Automated Analysis of Security Requirements through Risk-Based Argumentation," *Journal of Sys. and Softw.*, vol. 106, pp. 102–116, 2015.

[21] L. Pasquale, P. Spoletini, M. Salehie, L. Cavallaro, and B. Nuseibeh, "Automating Trade-off Analysis of Security Requirements," *Requirements Engineering*, vol. 21, pp. 481–504, 2016.

[22] S. Türpe, "The Trouble with Security Requirements," in *Proc. of the 25th Int. Requirement Engineering Conf.*, 2017, pp. 122–133.

[23] G. Bergmann, F. Massacci, F. Paci, T. T. Tun, D. Varró, and Y. Yu, "A Tool for Managing Evolving Security Requirements," in *Proc. 23rd Int. Conf. Advanced Information Systems Engineering Forum*. Springer, 2011, pp. 49–56.

[24] J. Bürger, J. Jürjens, and S. Wenzel, "Restoring security of evolving software models using graph transformation," *Int. J. Software Tools for Technology Transfer*, vol. 17, pp. 267–289, 2015.

[25] T. T. Tun, M. Yang, A. K. Bandara, Y. Yu, A. Nhlabatsi, N. Khan, K. M. Khan, and B. Nuseibeh, "Requirements and specifications for adaptive security: Concepts and analysis," in *2018 IEEE/ACM 13th Int. Symp. Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2018, pp. 161–171.

[26] K. Ramkumar, W. Cai, J. Mccarthy, G. Doherty, B. Nuseibeh, and L. Pasquale, "Diagnosing unknown attacks in smart homes using abductive reasoning," 2024. [Online]. Available: https://arxiv.org/abs/2412.10738

[27] R. Heartfield, G. Loukas, S. Budimir, A. Bezemskij, J. R. Fontaine, A. Filippoupolitis, and E. Roesch, "A taxonomy of cyber-physical threats and impact in the smart home," *Computers & Security*, vol. 78, pp. 398–428, 2018.

[28] Cloudflare, "What is a DDoS attack?" 2024, https://www.cloudflare.com/en-gb/learning/ddos/what-is-a-ddos-attack/ [Accessed: 08.12.2024].

[29] EasyDmarc, "IP/Domain Reputation Check," 2024, https://easydmarc.com/tools/ip-domain-reputation-check [Accessed: 18.01.2024].

[30] Amazon, "Working with trusted IP lists and threat lists," 2024, https://docs.aws.amazon.com/guardduty/latest/ug/guardduty$_{u}pload - lists.html[Accessed : 08.12.2024]$.

[31] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "Optimal load distribution for the detection of vm-based ddos attacks in the cloud," *IEEE Trans. Services Computing*, vol. 13, pp. 114–129, 2017.

[32] Netgear, "Netgear Security," 2024, https://www.netgear.com/security/ [Accessed: 09.12.2024].

[33] T. OConnor, D. Jessee, and D. Campos, "Through the spyglass: Towards iot companion app man-in-the-middle attacks," in *Proc. 14th Cyber Security Experimentation and Test Workshop*, 2021, pp. 58–62.

[34] B. P. Bhuyan, A. Ramdane-Cherif, R. Tomar, and T. Singh, "Neuro-symbolic artificial intelligence: a survey," *Neural Computing and Applications*, pp. 1–36, 2024.

[35] M.-T. Bui, M. Boffa, R. V. Valentim, J. M. Navarro, F. Chen, X. Bao, Z. B. Houidi, and D. Rossi, "A Systematic Comparison of Large Language Models Performance for Intrusion Detection," *Proc. ACM Networking*, vol. 2, pp. 1–23, 2024.

[36] OWASP, "OWASP Top 10 for LLM Applications 2025," 2024, https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/ [Accessed: 14.12.2025].