Towards a Natural Perspective of Smart Homes for Practical Security and Safety Analyses

Sunil Manandhar, Kevin Moran, Kaushal Kafle, Ruhao Tang, Denys Poshyvanyk, Adwait Nadkarni
William & Mary, Williamsburg, VA, USA
{sunil, kpmoran, kkafle, rtang, denys, nadkarni}@cs.wm.edu

***Abstract*—Designing practical security systems for the smart home is challenging without the knowledge of *realistic* home usage. This paper describes the design and implementation of Hϵlion, a framework that generates *natural home automation scenarios* by identifying the regularities in *user-driven* home automation sequences, which are in turn generated from routines created by end-users. Our key hypothesis is that smart home event sequences created by users exhibit inherent semantic patterns, or *naturalness* that can be modeled and used to generate valid and useful scenarios. To evaluate our approach, we first empirically demonstrate that this naturalness hypothesis holds, with a corpus of 30,518 home automation events, constructed from 273 routines collected from 40 users. We then demonstrate that the scenarios generated by Hϵlion seem valid to end-users, through two studies with 16 external evaluators. We further demonstrate the usefulness of Hϵlion's scenarios by addressing the challenge of policy specification, and using Hϵlion to generate 17 security/safety policies with minimal effort. We distill 16 key findings from our results that demonstrate the strengths of our approach, surprising aspects of home automation, as well as challenges and opportunities in this rapidly growing domain.**

I. Introduction

The smart home market is driven by the consumer demand for seamless automation, wherein devices react to the user's environment. Popular platforms such as SmartThings [1] and NEST [2] enable automation through trigger-action programs known as *routines*, which help users cause *action* events when a *trigger* condition is satisfied; *e.g.*, when the user gets home (*i.e.*, trigger) turn the security camera OFF (*i.e.*, action). Routines are the building blocks of home automation.

Prior research has analyzed routines, and more specifically, *IoT apps* published in marketplaces such as the SmartThings Official Repository [3], to understand the security, safety, and privacy properties of home automation. For instance, researchers have analyzed routines to detect the consequences of chains of routines (*e.g.*, Soteria [4] and IoTMon [5]), enable contextual integrity (*e.g.*, ContexIoT [6]), provide provenance information (*e.g.*, ProvThings [7]), and track data leaks (*e.g.*, Saint [8]). While prior research provides a useful estimate of potential problems, its findings would be actionable *iff* users deploy a specific combination of IoT apps, and execute them in a particular order. Without this critical insight, it is difficult to put the findings of prior work into perspective.

More importantly, without considering realistic home automation usage, designing or evaluating *practical* systems is challenging. For instance, consider the problem of specifying security/safety policies. A researcher creating policies for a system such as Soteria [4] or IoTGuard [9] must manually formulate use/misuse cases for smart home devices. This manual approach (1) requires significant effort, (2) is limited by the ability of the researcher to enumerate scenarios, and (3) may not reflect the use/misuse scenarios that may *naturally* occur in end-user homes. Unfortunately, we have no recourse other than this approach for policy specification, as we do not know what scenarios may realistically occur in end-user homes. For this same reason, security systems built for smart homes are often evaluated with random events as input (*e.g.*, ContexIoT [6] and IoTSAN [10]), which may not reflect the practical performance of the system. Thus, we observe a critical gap that limits the practicality of research in smart home security: *the lack of natural home automation scenarios*, *i.e.*, event sequences likely to occur in end-user homes. Such scenarios can be effective for addressing critical design/deployment problems, *i.e.*, they can be *analyzed* to discover safety, security, or privacy issues for *policy specification*, or *executed* as *test cases* to evaluate the security, performance, or usability of the system.

Consider this scenario generated during our policy specification case study (Sec. VIII):

user comes home → it is evening → gas range turns ON
→ speaker turns ON → motion is detected
→ security camera takes picture.

In this scenario, given that the user has arrived home, most events that follow are realistic, *e.g.*, turning the gas range ON or motion being sensed. However, we also see a subtle *privacy violation*: while the camera is configured to take pictures for home monitoring, in this particular instance, the user is the most likely cause of the motion, and hence the subject of the unintended picture. This privacy violation motivates the policy of turning the camera OFF when the user is at home.

The availability of such scenarios would enable and simplify the design and evaluation of practical security/safety systems for the smart home. The question is, how would we obtain such scenarios? A straightforward source would be *real execution traces* from end-user homes. However, execution traces are significantly noisy and privacy invasive, as we discuss in detail in Section II-B. Therefore, we consider a practical alternative: *automatically generating synthetic but realistic scenarios*.

The key argument in this paper is that natural scenarios may be generated by learning the regularities in *user-driven* home automation, *i.e.*, automation resulting from routines that end-users configure through interactive user interfaces (UIs)

provided by platforms (*e.g.*, the SmartThings [1]) or third-parties (*e.g.*, Yeti [11] and Yonomi [12]). User-driven routines are a realization of the "end-user programming" paradigm in smart homes, as users can assign triggers and actions without writing a single line of code. Users are empowered to craft their own routines that directly represent their requirements, without relying on developer-defined IoT apps. In fact, IoT apps may not represent all user requirements, *i.e.*, out of the 273 routines (233 unique) created by our 40 users (Sec. IV-A), more than 42.49% were not represented by any of the 187 SmartThings marketplace apps [3]. This mismatch between developer-provided apps and user needs motivates our focus on user-driven routines for modeling home automation.

We propose a novel approach that uses statistical language modeling [13] to identify the regularities in user-driven home automation, and leverages them to generate scenarios. Our approach builds upon a key result from the domain of Natural Language Processing (NLP), which states that while a natural language such as English may be extremely expressive in theory, *in practice*, the use of the language by people is "natural", *i.e.*, generally exhibits certain patterns and is, thus, predictable. This result was extended by Hindle et al., who demonstrated that source code, just like natural language, is the result of human effort, and thus, contains patterns that make it predictable [14]. Given that user-driven routines are effectively expressions of programs created by humans, we can leverage Hindle et al.'s insight to predict home automation scenarios. Specifically, we define the notion of a *home automation sequence*, which is the ordered set of routines that the user has scheduled to execute in their home, analogous to functions invoked in a program, and test the following **core hypothesis**:

> Home automation sequences created by humans are implicitly ***natural***, *i.e.*, they exhibit semantic patterns that make them predictable. Thus, we can use statistical language modeling to analyze corpora of sequences and predict ***useful*** home automation scenarios that enable the design and evaluation of security systems.

Contributions: We present a framework that enables a natural perspective for Home automation security EvaLuatION (Hϵlion). We initialize Hϵlion with routines collected from users. Moreover, we observe that the order in which routines execute may have different, even contradictory, security implications. Hence, for a precise characterization, Hϵlion obtains clues from users about the potential execution of individual routines, *i.e.*, *execution indicators*, and uses them to schedule routines, leading to ordered *home automation sequences*. Hϵlion uses the n-gram language model to learn patterns from such sequences, *i.e.*, obtains a "natural perspective" of home automation. Using this model, Hϵlion generates scenarios that are *realistic*, *i.e.*, reasonably likely to occur in a user's home. These scenarios help us identify real safety/security/privacy problems, a prime example being the privacy violation scenario presented earlier, *which was generated by Hϵlion.* Our contributions are summarized as follows:

- **Design of Hϵlion:** We contextualize statistical language modeling to the domain of home automation, and the problem of generating natural and useful scenarios. Several aspects of this design are novel: (1) its focus on *user-driven routines*, (2) abstractions such as *execution indicators*, and (3) the use of model *flavors* for generating diverse scenarios.
- **Naturalness of home automation:** We empirically test the naturalness hypothesis over a home automation corpus (called HOME) consisting of 30,518 events, split among event sequences from 40 users, created using 273 routines, and show that home automation is indeed *natural*, even more so than natural language and software corpora.
- **Validity of predicted scenarios:** We demonstrate the validity of our scenarios from the perspective of end-users in two studies with 16 additional external evaluators, and perform a third study to demonstrate the effectiveness of our predictive approach at the task of generating scenarios, over the baseline of using a formal (*i.e.*, graph-based) model.
- **Usefulness of scenarios:** We demonstrate the usefulness of the scenarios by using Hϵlion to generate security and safety policies for home automation. Hϵlion's approach automates the use/misuse case analysis workflow by *eliminating the need to imagine scenarios*, and reduces manual effort relative to a formal graph-based model. We semi-automatically discover 27 unsafe scenarios that motivate 17 policies.

Finally, we distill *16 key findings* ($\mathcal{F}_1 \rightarrow \mathcal{F}_{16}$) from our results that demonstrate the strengths of Hϵlion, surprising aspects of home automation, as well as challenges and opportunities in this new research domain.

II. Motivation and Background

The goal of this paper is to model home automation and generate realistic scenarios that may be useful at every step in the design and evaluation of security, safety, and privacy systems for the smart home. For a tractable analysis, we focus on using scenarios to address one of the most important problems in the design of security systems: *policy specification*.

A. Motivating Example: Effective Policy Specification

A common trait of the home security/safety systems proposed by prior work [15], [8], [4], [10], [5], [16] is their reliance on policies for analysis or enforcement, which researchers generally specify based on their understanding of the home. For example, Soteria [4] has a policy which states that "the refrigerator and security system must always be on", which is motivated by the safety and security consequences of the two devices being OFF. Moreover, these policies are often inherited by future research, *e.g.*, IoTSAN [10] and IoTGuard [9] use and extend Soteria's policies. Thus, effective policy specification is critical for the design of current *and* future systems. This 2-part example illustrates a gap in the process of policy specification that scenarios can address:

Example Part 1 – *Manual* approach (status quo): Consider Alice, a security researcher who is building a smart home security system. Alice relies on use/misuse-case requirements engineering (*e.g.*, as in prior work [4], [9]) to specify the

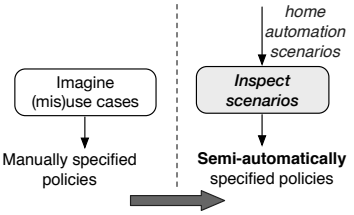

Fig. 1. The subjective and tedious task of imagining use/misuse cases can be replaced by a semi-automated approach of inspecting natural scenarios, for practical policy specification.

policies for this system, a process that consists of three steps: **Step 1:** *Recognize Assets.* Alice comes up with a list of assets she cares about, *e.g.*, home states that may have safety/security implications, such as the user being away, or a fire alarm. **Step 2:** *Imagine use/misuse cases.* Alice uses her domain knowledge to come up with a set of home automation use and misuse cases, involving devices as well as environmental factors in the smart home. Assessing what behavior constitutes use or misuse is often contextual, *i.e.*, recall the example used in Sec. I; the camera taking a picture is normal and necessary for security when the user is away, but a privacy violation when the user is home. Enumerating all potential contexts, relying solely on imagination, **is a hard problem**. **Step 3:** *Create Policies.* Alice transforms the use/misuse cases to functional requirements or constraints, *i.e.*, policies.

Generating the use and misuse cases *manually* (*Step 2*) is hard, and costs Alice tremendous effort as well. We show how scenarios can make Alice's task significantly easier.

Example Part 2 – *Semi-automatic* policy specification using scenarios: As shown in Fig. 1, we intend to remove *Step 2*, *i.e.*, *instead of imagining use/misuse cases, Alice analyzes realistic scenarios to create policies*. Alice uses a simple state model to track the states of different assets in the home as each event in a test scenario plays out. Only when an interesting event occurs (*e.g.*, the camera takes a picture), does Alice interrupt the scenario and *inspect* the state of the home. Alice creates a policy if the inspected state indicates a safety, security, or privacy problem (*e.g.*, the camera taking a picture when the user is home). That is, we *reduce the problem* from manually specifying use/misuse cases, to simply *inspecting* a few home states when an interesting event occurs. Moreover, if the scenarios are natural, *i.e.*, reasonably likely in the wild, then the resultant policies can be said to be less subjective.

B. Realistic Scenarios vs. Real Execution Traces

One approach to obtain natural scenarios would be to extract them directly from execution traces of user homes (*e.g.*, the CASAS project [17]). Indeed, our initial attempt involved executing routines with real devices and platforms to collect traces. However, we observed that the traces often contained *noise*, *i.e.*, superfluous events arising from intricacies in the platform/device implementation. Given the fragmentation of smart home ecosystems, filtering such noise may be infeasible. Thus, real traces may not exhibit *qualitative* patterns agnostic to platform/device brands, which are necessary for understanding the *general* regularities in home automation.

Another concern with real traces is *privacy*. At a time when privacy concerns are already a roadblock in smart home adoption [18], [19], collecting traces from homes would be extremely invasive, as they are not mere preferences, but *evidence* of user activity. In comparison, synthetic scenarios grounded in user expectations would be significantly less invasive. Indeed, we later demonstrate the feasibility of generating realistic scenarios that seem valid to end-users (Sec. VII), without ever looking at exact traces from user homes.

Now that we understand how Alice can benefit from synthetically generated but realistic scenarios, the key question is: *how can we generate natural home automation scenarios?*

C. Intuition: User-driven Home Automation is Natural

One way to generate natural scenarios is to build a model that can *predict events that are probable in the future, given the events that have already occurred* (*i.e.*, the *history*). Statistical language models (LMs) enable exactly this type of prediction.

Our core hypothesis (Sec. I) that would enable us to use LMs in our context is an extension of one of the key arguments in NLP: while languages themselves are tremendously expressive, their *actual use* by humans is often repetitive enough to be predictable. For example, given the sentence "You only live <missing word>.", it is easy to guess that the missing word should be "once", despite the fact that syntactically, "barely", "infinitely" or any other adverb would be just as correct. LMs trained on a corpus of common "utterances" in English would make the same prediction, based on the statistical probability of the word "once" *given the preceding phrase*.

In a similar vein, home automation events generally follow predictable patterns. Imagine the following sequence of events in a room where a light is controlled by a motion sensor: "sensor detects motion, lights are turned ON, motion is not detected for a while, <missing event>". Intuitively, the concluding event would most likely be "lights turn OFF". We use LMs to leverage this naturalness and make useful predictions.

D. Background: Statistical Language Modeling

This section provides the intuition behind statistical LMs, and specifically, the n-gram LM approach. A more mathematical discussion of these concepts can be found in Appendix E.

LMs measure the probability of a sentence $s = w_1^m = w_1 w_2 ... w_m$, given the probabilities of the individual words in the sentence (*i.e.*, w_1^m), as previously estimated from a training corpus. This ability enables prediction, *i.e.*, of predicting the next most probable word that can follow a sequence of words. In the context of modeling smart home routines, we define a "sentence" to represent a sequence of home automation events, wherein the "words" (a.k.a *tokens*) are smart home events.

The n-gram language model: In practice, however, there are often too many unique sequences to properly estimate the probability of tokens given long histories, even with large training corpora. Thus, we make use of the *n-gram* LM, which assumes the *Markov property*, *i.e.*, instead of computing the conditional probability given an entire history, we can *approximate* it by considering only a few tokens from the past. This

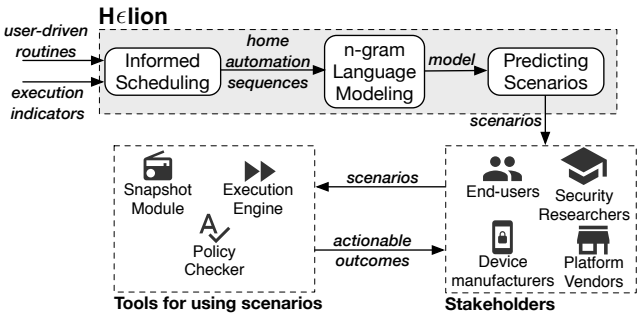

Fig. 2. An overview of the Hϵlion framework, which models home automation sequences to construct natural scenarios. Stakeholders use tools that analyze or execute scenarios to obtain actionable outcomes.

practical approach is valued in NLP and software analysis, as it enhances the model's predictive power by providing it with more examples to condition token probabilities, *i.e.*, as shorter sequences are more likely to occur in training corpora. Moreover, n-gram LMs may be an even better fit for analyzing home automation event sequences, due to the localized *causal* relationships between triggers and actions (and also event chains [20]), which are more relevant than the presumably weaker correlations with events from the distant past.

Evaluating the naturalness of a corpus: Our approach of modeling home automation using n-gram LMs will be effective only if our naturalness hypothesis holds. Thus, we must answer the question: *Are user-driven home automation event sequences natural?* The naturalness of token sequences can be measured according to a trained model's *perplexity* (or its log-transformed version, *cross-entropy*) on unseen data, which is a standard metric for assessing the viability of statistical language modeling for modeling any corpora. That is, a model will be "perplexed" upon observing a new sequence if it finds the sequence surprising, *i.e.*, unlike any sequence observed in the corpora. If the perplexity of a model built on corpora from a domain is low, it means that the domain exhibits naturalness, and that the model can identify regularities in the corpora, and predict events from that domain with significant confidence.

III. The Hϵlion Framework

Figure 2 shows Hϵlion, a data-driven framework that models the regularities of user-driven home automation, generates natural home automation scenarios, and provides stakeholders with tools to use the scenarios and obtain actionable outcomes. As shown in the figure, the process begins by collecting *user-driven routines*, as well as the corresponding *execution indicators*, *i.e.*, clues about when or how frequently the routines may be scheduled to execute, from end-users (Sec. III-B). Hϵlion transforms the routines and the corresponding execution indicators into home automation *event sequences*, using a process called *informed scheduling* (Sec. III-C). The resulting event sequences serve as a training data set that is represented using n-gram language modeling (Sec. III-D). Hϵlion's language model captures the patterns among the sequences, and can confidently generate new natural events that follow a home's previous history (Sec. III-E), forming a scenario.

We envision a set of tools – developed to leverage Hϵlion's scenarios – that enable key applications for several stakeholders. In this paper, we explore our motivating example of policy specification (Sec. II-A), by developing a tool called the *snapshot module*. This module allows security researchers to examine scenarios by capturing an evolving "snapshot" of the home for analysis, for every successive event in a scenario (see Sec. VIII for details). In addition to our main focus of policy specification, we also prototype an *execution engine* (see Sec. VIII), and discuss its use by platform and device vendors to test their products under realistic scenarios, with real devices (Sec. VIII-C). Finally, while we do not explore this aspect, Hϵlion's scenarios may be used with existing *policy checkers* [4], [10] to help end-users identify potential problems in their homes. We now describe the general threat model for Hϵlion's scenarios and use cases, followed by the design of its individual components.

A. Threat Model

Hϵlion generates scenarios that denote *realistic* behavior of the smart home. However, realistic scenarios may lead the system into an unsafe state (*i.e.*, in terms of the access control safety problem [21]). In most cases, an unsafe state may be reached both (1) *accidentally* (*e.g.*, similar to accidental data disclosure [22] or safety violations [9]), or (2) *due to a malicious adversary*. Consider this example from Sec. VIII: a gas range turns ON during a fire, which may make matters worse (*e.g.*, cause an explosion). Such an event may occur accidentally, *e.g.*, if the range is set to turn ON when the user wakes up (*e.g.*, to heat water), which happens to be when there is a fire. However, such an event may also be triggered by an adversary who can commandeer global state variables such as "awake/asleep", or "home/away". Indeed, our prior work [16] demonstrates how an attacker could manipulate such variables via compromised low-security devices, to indirectly control high-security devices. Finally, we also attempt to generate *unnatural* (*i.e.*, unlikely) scenarios to mimic accidents or adversarial circumstances (Sec. III-E).

The general threat model described above applies when scenarios are *analyzed*, *e.g.*, for the policy specification use case. However, we also envision adversary models that are specific to the use case, particularly when the scenarios are *executed* by stakeholders. For instance, a platform vendor may use scenarios to dynamically test the behavior of partner devices under realistic circumstances, in which case, the partner devices may be assumed to be the adversary (*i.e.*, they may violate the platform's security goals). On the contrary, a device/app developer may want to dynamically test their own product under natural scenarios, in order to weed out accidental violations of their own security/safety/privacy claims. Hϵlion may enable several such use cases that improve the overall health of the home automation ecosystem.

B. *Collecting* User-driven *Routines*

Hϵlion deviates from prior work as it is grounded in user-driven routines that reflect user requirements *directly*, as opposed to IoT apps that may only represent the developers'

perspective. Thus, a critical step in instantiating Hϵlion is the *collection* and *representation* of the events that form routines into semantically meaningful tokens suitable for modeling.

1. Collecting routines from users: To collect routines from users, we use a survey methodology that is conceptually similar to prior work by Ur et al. [23] (see Sec. IV-A for details). The raw data collected from the surveyed users consists of two components: (i) routines specified in a structured natural language format, and (ii) *execution indicators*, *i.e.*, clues for when these routines typically execute. For modeling, the natural language routines must be converted into semantically equivalent tokens. We illustrate this conversion with the following (raw) routine from our dataset:

IF the motion is detected **THEN** camera takes a picture

2. Representing smart home events as tokens: In the context of this paper, tokens are home automation events parsed from structured natural language routines. An event can denote a change in the state of a device (*e.g.*, door locked) or the home (*e.g.*, the user is away). To model the varying attributes of home automation events, we express our tokens as ordered information lists called tuples. The design of the tokens must strike a balance between encoding enough salient information to be descriptive, while still representing semantically similar events in equivalent tokens to capture meaningful patterns in the data. We define Hϵlion's home automation event token as:

$$e_i := < device_i, attribute_i, action_i >$$

where $device_i$ represents the device (*e.g.*, door lock, camera), the $attribute_i$ corresponds to one of a predefined set of device attributes (*e.g.*, the *lock* attribute for the door lock, which can take the values LOCKED/UNLOCKED), and $action_i$ represents the change of state, and hence, the current value of $attribute_i$. Using this design, the example routine discussed previously (*i.e.*, the motion sensor/camera) would be tokenized as:

$$< motion\_sensor,\ motion,\ \text{DETECTED}>,$$
$$< security\_camera,\ image,\ \text{TAKE}>^{1}$$

Note that to represent an event that causes a change to the overall state of the home (*e.g.*, "If the user is home"), we do not use the $device_i$ field, but only the attribute (*i.e.*, "home") and the specific change (*i.e.*, HOME or AWAY), to generate a token as follows: $< \phi, locationMode,$HOME$>$.

Of the several considerations that went into the design of this token, the most important was the decision to exclude a classification of an event as a *trigger* or an *action*. This choice is logical as our goal is to model home automation "event sequences" rather than a set of routines, and several events can be triggers or actions, depending on where they are used in a routine; *e.g.*, the trigger condition "if a picture is taken", and the event resulting from the action "take a picture", deal with the same device, attribute, and action, and hence the same event. This decision prevents unnecessary sparseness in the data, as including a trigger/action classification in the token would generate two separate tokens for the same event.

[1]We use terminology from the SmartThings capabilities reference [24].

C. Generating Event Sequences with Informed Scheduling

Hϵlion transforms the tokenized routines specified by a particular user into a home automation event sequence, *i.e.*, an approximate *ordered* representation of how the routines would execute in the user's home. This importance of order is apparent in natural language, *i.e.*, while single words carry with them isolated meaning, words combined into a sentence with a specific, intentional, ordering form a more precise, collective, meaning. The same can be said about home automation, *i.e.*, where events are like words in a sentence whose order is bound to affect the "meaning" (*i.e.*, implications) of the sequence. Consider a simple sequence that illustrates this point:

motion detected $\rightarrow$ camera takes picture

Intuitively, we can interpret this sequence to *mean* that that the camera takes a picture because motion is detected. However, if the order was reversed, the sequence would not have a logical meaning. Thus, scheduling routines in the right (or best-effort) order is important for generating home automation sequences. The question is: *How can we obtain this order?*

1. Introduction to Execution Indicators: We propose a novel abstraction for users to stipulate the approximate order in which routines may execute, *i.e.*, routine-specific *execution indicators*. We exploit the possibility that end-users have some intuition regarding *when* certain routines execute, based on when certain device or environmental events may generally occur. For instance, blinds are usually opened in the morning, and closed at night. As a result, a user may order a routine triggered by the opening of the blinds *before* another triggered by their closing. Similarly, users may be able to describe when they perform certain personal tasks which trigger home automation, *i.e.*, when they come home, go to work, bed, cook, or do laundry. Execution indicators allow us to capture such factors, which we then leverage to schedule routines to create home automation sequences. This is why we define the approach as informed scheduling, as the *scheduling mechanism is informed by the user's understanding of their own home use*.

2. Specifying Execution Indicators, Informed Scheduling: Execution indicators constitute the time and frequency of the potential execution of a routine. As users may not be able to specify precise values, we allow users to pick broad ranges of values organized into three types: (1) the *time-range* indicator (*e.g.*, early morning, noon, and night), (2) the *day-range* indicator (*e.g.*, mostly on weekdays, and mostly on weekends), and (3) the *frequency* indicator (*e.g.*, many times a day, few times a day, few times a month). As mentioned earlier execution indicators are collected from users for each routine during the data collection survey (see Sec. IV-A). We then use these execution indicators to generate a month-long time-series for each user, where each routine may occupy one or more one-hour time-slots (*i.e.*, depending on frequency), using the following algorithm for informed scheduling:

We initialize a month-long time-series, with hourly slots that can hold routines. We first place the routines triggered at specific times (*e.g.*, at 8AM, open the blinds) as defined. For each remaining (*i.e.*, un-placed) routine, we identify the

potential slots for placement, based on its time-range indicator. The frequency indicator of the routine determines how many instances of the routine to uniformly distribute among those slots. This distribution is also adjusted, based on the day-range indicator, *i.e.*, skewed in favor of weekdays or weekends. For the few routines without execution indicators (*i.e.*, when users are unsure), we randomly distribute them throughout the remaining slots in the month. Finally, once all the routines have been scheduled in the time series, we extract the ordered set of routines from the time series as the execution sequence.

We empirically demonstrate that users can confidently supply execution indicators for most routines, only being unsure for generally unpredictable events such as CO leaks (Sec. IV). *Accurately* scheduling all possible routines is a broader research challenge that is beyond the scope of this paper, as we discuss in Sec. X.

D. Modeling Event Sequences

Hϵlion uses the n-gram model to learn the regularities in user-driven home automation sequences, as described previously in Sec. II-D. This section describes certain intricacies that affect the performance of the model.

1. Why do we need $n \geq 3$?: Recall that when estimating the probability of a sequence of length n, the n-gram model computes the probability of the n^{th} token appearing after the $n-1$ previous tokens (*i.e.*, the *history*). The intuition behind looking back at the $n-1$ events is that they provide the *context* as to why the n^{th} event is being scheduled. Given this intuition, one thing is clear: when choosing n for modeling home automation sequences, we can rule out values of $n < 3$. That is, $n = 1$ will only estimate the probability of individual events, completely ignoring the context. Choosing $n = 2$ is only slightly better, as it may only capture simple relationships that are already observable from data, *i.e.*, trigger-action routines we collect from users. Only with larger values of n, *i.e.*, $n \geq 3$, the model can learn non-obvious regularities in home automation corpora. To illustrate this point, consider the following sequence whose probability is being estimated using a 4-gram model, *i.e.*, $n = 4$:

user comes home $\rightarrow$ lights ON $\rightarrow$ it is evening $\rightarrow$ door locks.

Here, the factors such as the user being home, the lights being ON and the time of the day being the evening provide the context for the occurrence of the next event, *i.e.*, the locking of the door. As a result, examining the last $three$ events certainly helps. However, there is a caveat: considering too much of the event history (*i.e.*, a very large n) may actually hurt the predictive power of the model. That is, an event that occurred earlier during the day (*e.g.*, the user going to work) would likely not share any semantic relationship to the given sequence, and hence, would not really encapsulate any regularities. Moreover, longer sequences may be relatively uncommon in the wild, even if they are realistic and useful for uncovering serious security/safety flaws. As a result, the choice of n directly impacts the ability of the model to capture the existing relationships between events in the corpus, especially if they belong to the same, high-level, user-activity.

2. The need for smoothing: Selecting $n \geq 3$ may seem to intuitively lead to a better model, however, for higher orders of n there will inherently be fewer sequences to learn from, as longer sequences are often unique. That is, the sequence:

the user leaves the home $\rightarrow$ the door locks

may often occur in a corpus, whereas the sequence:

user leaves the home $\rightarrow$ door locks $\rightarrow$ motion is detected

is likely to be less common. This leads to a data sparsity problem, wherein it is likely that a history queried during prediction may have not been observed in a training corpus. As a result, a naive model will be unable to predict the next event, if this entire history is not present in the training corpus, even if it may have seen subsequences of this history (*e.g.*, user leaving & door locking). We empirically demonstrate this lack of prediction in our comparison with a formal graph-based baseline (Sec. VII-C). To allow the LM to assign probabilities to previously un-observed sequences, and hence predict higher-order n-grams with sufficient statistical rigor, we rely on *smoothing* [25]. Smoothing is a well-known technique in NLP where the model assigns some probability distribution to rare or unobserved sequences. We consider two smoothing methods: (1) *backoff* and (2) *interpolation*. At a high level, backoff smoothing techniques simply revert to predictions based on lower order n-grams when histories of higher order are unobserved. Conversely, interpolation combines token probabilities for lower-order n-grams when making predictions. In our instantiation of Hϵlion, we elected to use interpolated n-grams due to their demonstrated ability to perform well with lower-order (*i.e.*, 3-4 gram) models [25].

E. Generating Expressive Scenarios for Security

Hϵlion generates scenarios by treating the model as a *sequence generator* that can produce an arbitrarily long series of events. That is, given a history, the n-gram model examines the previous $n-1$ events, and predicts the next *most probable* event, *i.e.*, the subsequence provided by a majority of the users (see Appendix E for details). The newly predicted event now becomes a part of the history for the next prediction. We can continue these predictions to get arbitrarily long scenarios.

Our threat model (Sec. III-A) describes how scenarios may be interpreted for security policy specification, *i.e.*, as *realistic* event sequences that are reasonably likely to occur in end-user homes, but which may indicate unsafe situations, whether accidentally or adversarially triggered. However, we also need to generate scenarios that denote *unnatural* situations. For instance, when specifying policies (Sec. II-A), Alice also needs highly unlikely scenarios that demonstrate stress tests, or rare but unsafe situations. To support such usage, Hϵlion can be configured to generate two *flavors* of scenarios:

1. The *up* flavor, *realistic* scenarios: This flavor is the default, *i.e.*, where our model generates highly probable event sequences, given a history using the natural probability distribution over tokens in the training corpus.

2. The *down* flavor, *unrealistic* scenarios: The *down* flavor corresponds to an *unnatural* distribution over tokens, *i.e.*, Hϵlion's LM generates *down* scenarios by sorting the model's most probable token predictions given some history, and then reversing this order, such that the *most improbable* token is given by the model as the prediction. The *down* scenarios may be interpreted as a system under constant attack, or where all devices are simultaneously malfunctioning.

IV. Data Collection and Initial Findings

This section describes our data collection methodology, approach for constructing the HOME corpus consisting of home automation sequences for use with Hϵlion, and important findings that can be directly gleaned from the data ($\mathcal{F}_1 \rightarrow \mathcal{F}_5$).

Data Availability and IRB Approval: All our user studies were performed with IRB approval. Anonymized datasets and code for the Hϵlion framework are available at [26].

A. Methodology for Collecting Data from Users

We used a survey methodology for collecting data for constructing the HOME corpus. The 40 users surveyed were generally from the Computer Science (CS) academic population: 37 were current graduate and undergraduate students, and 3 had PhDs. The majority owned at least one smart home device (24 or 60%), while a significant minority had created routines before (17 or 42.5%). To make the task of specifying routines and indicators easier, we split the survey into 3 steps:

1. Selecting devices: First, participants selected devices that they could envision (or already have) in their smart home. To enable this step, we provided the participants with a broad *device list* consisting of 70 unique types of devices available in the market. We constructed this list using resources such as websites and mobile apps of all the device partners of the popular Samsung SmartThings [27] and Google NEST [28], popular technology websites, and technology forums. Fig. 4 in Appendix A shows our device selection screen.

2. Creating routines: Participants were then given a short tutorial on routines, and asked to create routines using the devices that they had previously selected, along with general smart home variables such as home/away, temperature, and time. We asked the participants to provide triggers and actions in a *plain English text to allow them to express any functionality desired without enforcing any artificial constraints*.

We provided an interactive form for creating routines (Fig. 6 in Appendix A). We also provided functional information about devices (*e.g.*, the "lock" attribute for the door lock) to help participants focus more on the task of creating routines than imagining device functions (Fig. 5 in Appendix A). To enable this approach, we created a *device-attribute map* by systematically assigning one or more of the 110 attributes obtained from popular platforms (*i.e.*, NEST [28] and SmartThings [27]) to each of our 70 devices.

3. Specifying Execution Indicators: After creating routines, participants specified the time-range, day-range and frequency indicators for their routines (Fig. 7$\rightarrow$ 9 in Appendix A). Participants could select from predetermined ranges, indicate "anytime" for routines that could occur at any time (*i.e.*, for the time and day-range), or "not sure". We also collected additional information that may assist in understanding user-driven home automation (Fig. 10$\rightarrow$ 13 in Appendix A).

B. Constructing the HOME Corpus

We transformed the routines from plain English into an intermediate trigger-action format, and then tokens, using the syntax described in Sec. III-B. We also considered two additional situations when tokenizing: (1) if the trigger/action consisted of a conjunction of events, we combined the events into a single token (in the alphabetical order by device), as those events would be expected to execute simultaneously, and (2) for attributes with continuous values (*e.g.*, temperature), we abstracted the user-provided values into ranges (*e.g.*, low, medium, or high), to create semantically unique tokens. Two authors independently verified the correctness of the tokens. Finally, we constructed a *month-long* home automation event sequence for each participant using informed scheduling (Sec. III-C), creating the HOME corpus.

C. Initial Findings from Survey Data

The HOME corpus consists of 30,518 events, from 40 month-long sequences (*i.e.*, 40 users), generated from 273 routines (233 unique) and their execution indicators. Our analysis of the survey data led to the following findings:

Finding 1 **($\mathcal{F}_1$): Routines are *important* to users. Moreover, users leverage most available devices for creating routines.** When asked how important routines were to them, 4 users or 10% indicated that routines were "very important", 20 or 50% indicated "important", while 16 or 40% indicated "somewhat important". None indicated that routines were unimportant.

Further, our participants used 61/70 devices (or 87.14%) in at least one routine. Devices related to lighting and temperature control were the most popular for automation (*i.e.*, selected by 23 and 21 users respectively), closely followed by security devices such as cameras. This finding indicates a strong user preference for integrating device functions via routines.

Finding 2 **($\mathcal{F}_2$): SmartApps do not represent a significant number of user-driven routines.** Out of the 233 unique routines created by our 40 end-users, only 134 or 57.51% could be represented by SmartApps from the SmartThings market [3], *i.e.*, *42.49% were not*. For instance, the following user-driven routine was not represented by any SmartApp: **IF** the doorbell rings **THEN** turn the security camera ON. This routine represents a straightforward use-case, *i.e.*, it ensures that the camera stays OFF for most of the time the user is home (for privacy), but turns ON when significant events happen, such as when someone is at the door. Finally, only 40/187 SmartApps or 21.39% accounted for the represented routines, *i.e.*, our users would potentially not use the remaining 78.6%.

Finding 3 **($\mathcal{F}_3$): Users indicate a strong preference for controlling/creating their own routines.** Most users (32 or

TABLE I
SELECTION OF SPECIFIC RANGES, "ANYTIME", OR "NOT SURE", FOR EXECUTION INDICATORS, BY NUMBER OF ROUTINES.

Execution Indicator	Specific Range	Anytime	Not Sure
Time-range	155 (56.78%)	107 (39.2%)	11 (4.02%)
Frequency	256 (93.77%)	0	17 (6.23%)
Day-range	59 (21.61%)	186 (68.13%)	28 (10.26%)

80%) indicated *personal requirements* as their source of ideas for creating routines Moreover, when asked about their preference for vendor-controlled vs user-controlled automation, most users (20 or 50%) preferred a combination, a significant minority (17 or 42.5%) preferred a purely user-controlled setup, whereas only 3 (or 7.5%) preferred full vendor control.

Finding 4 **($\mathcal{F}_4$): Users may perceive any device as security sensitive, based on the context.** We asked users to select devices (out of 70) they considered to lead to harm if compromised or malfunctioning (*i.e.*, are security/safety-sensitive). As expected, the security/safety-focused devices were primarily selected, such as the security alarm (36 users), door lock (35), or the camera (35). More surprisingly, *every single device was marked as sensitive by at least 3 users*; *i.e.*, users also considered scenarios where their well-being may depend on a device that may not be considered security-focused. For example, the smart outlet was selected by 8 users, potentially because security-sensitive devices may be connected to it.

Finding 5 **($\mathcal{F}_5$): Users can confidently specify execution indicators aside from certain unpredictable triggers.** Table I summarizes the options chosen by our users, *i.e.*, selected one of the ranges offered, or "anytime", or "not sure", for each of the three execution indicators (*i.e.*, time-range, frequency, day-range). Users confidently specified indicators for most of their routines and were not sure in very few cases (*i.e.*, *at most 10.26%, for the day-range*). Moreover, users selected specific time-range and frequency values for a majority of routines, *i.e.*, for time-range (*i.e.*, 56.78%) and frequency (*i.e.*, 93.77%). This clearly demonstrates that users are able to confidently supply execution indicators for most of their routines. On further analyzing the "not sure" cases, we discovered that most are caused by triggers that are unpredictable by nature (*e.g.*, CO leaks), which explains why users could not specify them.

Our initial findings expose the evident need to analyze user-driven routines ($\mathcal{F}_1 \rightarrow \mathcal{F}_3$), and additional challenges, such as unpredictable execution indicators ($\mathcal{F}_5$). Sections V$\rightarrow$VIII will explore this data further to test naturalness of the home corpus, as well as the validity, and usefulness of Hϵlion's scenarios.

V. RESEARCH QUESTIONS (RQS)

We formulate 4 RQs to address our core contributions:

- **RQ**$_1$: How *natural* is home automation corpora?
- **RQ**$_2$: Do the scenarios generated by Hϵlion seem to be *valid* (*i.e.*, realistic) to the end-user?
- **RQ**$_3$: Can Hϵlion's scenarios be applied in *useful* ways to improve the security and safety of home automation?
- **RQ**$_4$: Can a formal (*e.g.*, graph-based) modeling approach generate scenarios as effectively as Hϵlion?

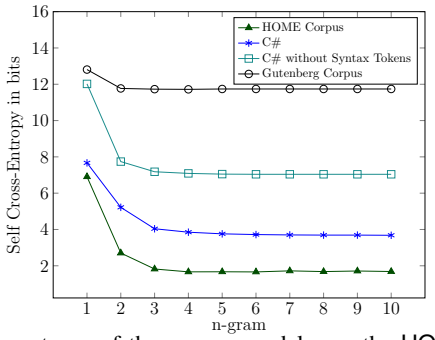

Fig. 3. Cross-entropy of the n-gram model over the HOME, Gutenberg, the C# corpora, and the C# corpus *without syntactic tokens*.

To answer RQ_1, we test our *naturalness hypothesis* for our HOME corpus, which is the foundation of Hϵlion (Sec. VI). To answer RQ_2 and evaluate the perceived validity of the scenarios Hϵlion generates, we perform two experiments with *external evaluators*, *i.e.*, 16 users who were not a part of the HOME corpus (Sec. VII). To answer RQ_3, we revisit the policy specification problem from Sec. II, and demonstrate how Hϵlion's scenarios enable an expert to predict security and safety policies with manageable effort (Sec. VIII). We also explore how stakeholders may gain insight into the security problems in platforms/devices, by executing the scenarios in a SmartThings-based execution engine. Finally, for answering RQ_4, we create a formal (graph-based) approach as a baseline to compare against Hϵlion, and discuss how (and why) Hϵlion's predictive approach enables it to generate valid scenarios where the baseline fails (Sec. VII-C).

VI. EVALUATING THE NATURALNESS OF HOME (RQ_1)

We test our naturalness hypothesis by measuring the cross-entropy of the HOME corpus, *i.e.*, the measure of how perplexed a model built from a training corpus is when exposed to other sequences from the same domain (Sec. II-D). We use the MITLM toolkit [29] to measure cross-entropy, with 10-fold cross validation. Since naturalness is relative, we compare the cross-entropy of HOME with that of the Gutenberg corpus (*i.e.*, English), as well as a the C# corpus, *i.e.*, the best cross-entropy of all programming corpora [14]. The raw cross entropy values for C# and Gutenberg are from [30].

Results: As seen in Fig. 3, the cross-entropy for the HOME corpus starts at a high of 6.9 bits for a unigram model, drops to 2.70 bits for the bigram model, and then stays close to 1.7 bits for the rest of the values of n from 3 up to 10. There is an explanation for this trend, which is observed in the Gutenberg and C# corpora as well: a unigram entropy is expected to be much higher, as the unique token frequencies may be heavily skewed. However, as the model considers more history (*e.g.*, for the bigram), the entropy drastically reduces.

Finding 6 **($\mathcal{F}_6$): The HOME corpus is natural, relative to English language and software corpora, *without any syntactic glue*.** As seen in Fig. 3, the HOME corpus has low cross-entropy values relative to C# and Gutenberg corpora, and hence can be said to be more natural, which satisfies our *naturalness hypothesis* (RQ_1). Further, there is another

interesting aspect of the HOME corpus: the user-driven home automation sequences in the HOME corpus are purely semantic in nature, *i.e.*, have zero syntax involved, and only capture the functionality that the user desires from the automation. On the contrary, C# or other software corpora may appear more natural than they are because of the common "syntactic glue" [30], which when removed, causes a steep drop in naturalness (see "C# without Syntax Tokens" in Fig. 3). That is, while the naturalness of software corpora drops when the syntactic glue is removed, the HOME corpus is unaffected, which indicates its amenability to being predicted.

The low cross-entropy of the HOME corpus demonstrates that Hϵlion's model (*i.e.*, Sec. III-D) would be able to capture the regularities in it, and hence, can generate *natural* scenarios. However, are such scenarios *valid* according to end-users? The next section addresses this question.

VII. Evaluating the Validity of Scenarios (RQ_2)

We assessed the ability of Hϵlion to generate valid scenarios with 16 *evaluators*, *i.e.*, a separate set of users who were not included in the initial 40, but from the same user population. This evaluation consists of *three* studies. First, we conduct the *routine comparison study* (Sec. VII-A), *i.e.*, we evaluate whether Hϵlion can generate routines that appear to be *just as valid as the routines collected from users*. Second, we perform a *sequence generation study*, in which we generate scenarios (*i.e.*, sequences) using *histories given by the external evaluators*, and test their perceived validity under various model configurations (Sec. VII-B). Finally, we create a graph-based model similar to those built by prior work for policy checking [4], [9], [10], and compare its ability to generate valid scenarios with that of Hϵlion (Sec. VII-C).

A. Routine Comparison Study

We envision that many applications of Hϵlion will rely on its ability to generate reasonable bigrams, *i.e.*, pairs of events which may be analyzed as routines. Using the routines previously created by end-users as our baseline (Sec. IV-A), this study answers the following key question: Can Hϵlion generate routines that are <u>*as valid as*</u> those created by users?

Generating Routines: Recall that Hϵlion is a sequence generator that predicts a sequence, one event at a time, given a history (Sec. III-E). We generate routines using an approach similar to 10-fold cross validation: *i.e.*, we split the HOME corpus randomly into 90%/10% sequences, train on the 90%, and sample histories from the remaining 10% (*i.e.*, random subsequences of "odd" lengths) to generate routines. Given a history with an odd length, the first prediction Hϵlion makes is likely to be the completion (*i.e.*, the action) of a routine, with the trigger being the last event in the history. Hence, we assume the second and third generated events as a new routine.

We randomly generated 40 such unique routines through the following configurations of the model (*i.e.*, varying the n-gram and *up*/*down* modalities): (i) *up/3-gram* (10 routines), (ii) *up/4-gram* (10 routines), (iii) *down/3-gram* (10 routines), and (iv) *down/4-gram* (10 routines). We chose the *up* flavor to answer the validity question, but we also chose the *down* flavor to understand if really unnatural routines (*i.e.*, down) are also perceived by users as invalid. Additionally, we randomly selected 10 unique routines from 10 different users from the data collected in Sec. IV-A, *i.e.*, (v) *survey* (10 routines).

TABLE II
Summary of the evaluator ratings for the routine comparison study (% computed out of 160 total ratings)

Routine Study (Bigram Generation)					
Model Config	**N-Gram**	**Strongly Agree**	**Somewhat Agree**	**Somewhat Disagree**	**Strongly Disagree**
up	3	**50.63%**	19.38%	11.87%	18.13%
	4	**69.38%**	13.12%	4.38%	13.12%
down	3	5.0%	11.87%	15.0%	**68.13%**
	4	0.63%	11.87%	19.38%	**68.13%**
– (survey)	–	**63.75%**	25%	3.75%	7.5%

Methodology: We conducted this study in-person, wherein a proctor explained the study procedure to the evaluators, and noted down any comments. Evaluators were asked to rate each of the 50 unique routines described previously, in terms of how valid (*i.e.*, reasonable) they seemed, according to a modified Likert Scale (*i.e.*, with options: Strongly Agree, Somewhat Agree, Somewhat Disagree, and Strongly Disagree). The order of routines was randomly shuffled for each evaluator to mitigate inductive bias. Additionally, evaluators were given the opportunity to provide additional feedback after the task.

Results: Table II summarizes the ratings for the 10 routines in each configuration by 16 evaluators (*i.e.*, 160 ratings per config). As we see in the table, routines generated with the *up* flavor are generally rated as valid, whereas the *down* routines are generally rated invalid. We now describe our findings:

Finding 7 **($\mathcal{F}_7$): Routines generated using Hϵlion are as valid as routines created by users.** Our evaluators rate over half (*i.e.*, 50.63%) of the *up/3-gram* routines and 69.38% of the *up/4-gram* routines as strongly agree. In the latter case, Hϵlion scores higher than the *survey* routines (63.75% strong agreement). Moreover, the total agreement (*i.e.*, strongly plus somewhat agree) is just over 70% for *up/3-gram* routines, and 82.5% for the *up/4-gram* routines, the latter being reasonably valid relative to the 88.75% agreement for the *survey* routines.

Finding 8 **($\mathcal{F}_8$): Unnatural routines generated by Hϵlion are perceived as invalid by users, and sometimes as unsafe/insecure.** Both *down/3-gram* and *down/4-gram* routines are overwhelmingly rated as invalid (68.13% strong disagreement), *i.e.*, unnatural sequences generated using Hϵlion are also considered invalid by users. More importantly, some users commented that these routines were unsafe; *e.g.*, one user said that "*...automations are a hazard. If I'm not home I don't want to gas stove to turn on.*" This finding indicates the potential of the *down* flavor for modeling misuse cases or anomalies.

Finding 9 **($\mathcal{F}_9$): The notion of validity may vary, even among users from the same population.** Evaluators rated 7.5% of the *survey* routines as strongly disagree, and another 3.75% as somewhat disagree. This indicates that even within the same population, there is diversity in terms of what users consider to be a reasonable routine, potentially due to personal

TABLE III
A SUMMARY OF THE RATINGS FOR THE SEQUENCE GENERATION STUDY. PERCENTAGES ARE OUT OF 160 RATINGS IN TOTAL.

Event Sequence Generation Study					
Training Data	**N-Gram**	**Strongly Agree**	**Somewhat Agree**	**Somewhat Disagree**	**Strongly Disagree**
HOME	3	**41.25%**	**21.25%**	15.63%	21.88%
	4	**37.5%**	**25.63%**	12.5%	24.38%
HOME + Evaluator's data	3	**43.13%**	**33.75%**	10.63%	12.5%
	4	**38.75%**	**28.75%**	16.88%	15.63%

preference. For example, one evaluator said that they needed additional conditions to be stated for a routine to be valid: "*...For the routine if time is Morning, turn the Coffee Maker on, the user should also be at home.*" In some cases, evaluators did not agree with a particular device being a part of the trigger or action, even if the rest seemed reasonable. These preferences also caused many of the "somewhat" ratings.

B. Sequence Generation Study

The sequence generation study measures the ability of Hϵlion to produce valid sequences of lengths $\geq$3, based on histories provided by the external evaluators. Further, we also test if the validity of the generated scenarios improves once Hϵlion knows more about the user's home automation context (*i.e.*, if we collect home automation sequences from the evaluators and include them in the model), just as NLP-based auto-complete approaches improve in their predictions after learning from the user's data. Therefore, prior to the study, we collected user-driven routines and execution indicators from the evaluators and computed home automation sequences for them. Also, we asked the evaluators to provide us with two histories, *i.e.*, event sequences that could happen in their home.

Methodology: This study was conducted using a methodology similar to the routine comparison study, with a few differences. First, all four model configurations were of the *up* flavor,[2] *i.e.*, the general *up/3-gram* and *up/4-gram* models trained on the HOME corpus, and *evaluator-specific up/3-gram* and *up/4-gram* models, trained on the HOME corpus as well as the evaluator-specific home automation sequences. Second, each model configuration was used to generate 5 successive events, for the two histories provided by each evaluator. The evaluator rated the validity of a generated event based on the history used to generate it, rating 40 events in total.

Results: Table III summarizes the evaluator ratings for the scenarios generated by Hϵlion (*i.e.*, 160 ratings per configuration). Evaluators mostly agreed with Hϵlion's scenarios, and the agreement was much higher when including evaluator-specific sequences. Further, evaluators justified scenarios as reasonable with additional explanation in most cases.

Finding 10 **($\mathcal{F}_{10}$): Hϵlion is able to generate reasonable scenarios, *despite previously unseen tokens* in evaluators' histories.** The histories provided by 8/16 evaluators contained tokens that were not present in the HOME corpus. Tokens outside model's vocabulary are generally known to hurt the predictive capabilities of LMs. However, Hϵlion delivers a reasonable worst case performance, *i.e.*, achieving a validity rating of over 62% in both 3 and 4-gram configurations.

Finding 11 **($\mathcal{F}_{11}$): Including the user's data improves the validity.** As shown in Table III, the models including the evaluator's event sequences perform much better, with the overall agreement for the 3-gram and 4-gram models rising to 76.88% and 67.5% respectively, from just over 62% previously. Note that whether training with or without the evaluator's data, the histories provided by the evaluators were mostly outside the scope of the model, *i.e.*, only 4/32 histories provided had subsequences of size $\geq$3 represented in the training sequences. Deployment of Hϵlion for end-users will benefit from this trend, as users may want to personalize the model, leading to improved predictions.

Finding 12 **($\mathcal{F}_{12}$): Context-shifts may lead to lower ratings.** Hϵlion can generate event sequences of arbitrary lengths; however, in reality, many scenarios of varying sizes may occur one after the other in the home. In many cases, users rated the first event of a new contextual scenario low, as they could not reason about it in terms of the history. For example, consider the following scenario, the first 3 events of which are the user-provided history: {it is night $\rightarrow$ temperature decreases $\rightarrow$ security system ON $\rightarrow$ air quality low $\rightarrow$ air purifier ON}. This quote from the user illustrates the context-shift: "*The first sequence does not make much sense to me because the air quality detection and the security system/night mode are not clearly related...second sequence makes perfect sense, as you would want the air to be purified if it was detected to be of suboptimal quality.*". As Hϵlion generates sequences that may span discrete contexts, it may be useful for a longitudinal evaluation of the home. Moreover, the challenge of capturing these context-shifts exposes a promising research direction.

C. Baseline Comparison: Hϵlion vs Graph-based Modeling

Prior work has used model checking (*e.g.*, using linear temporal logic (LTL)) to evaluate home automation routines against security or safety policies [4], [9], [10]. Such a model is constructed directly from a set of IoT apps, and can be generally represented as a directed graph, in which the vertices are home automation events, and edges represent *causal* relationships among the events (or state transitions, if the vertices represented device states). For example, given the routine **IF** user home **THEN** camera OFF, the graph will contain the edge (user home, camera OFF).

While prior research checks policies using this graph model, we intend to use it as a baseline approach for generating scenarios that is more grounded in user data than random sequences. Hence, we (1) create a graph-based model, called GraphHome (or GH) using the HOME corpus, and (2) use it to generate scenarios for the same histories provided to Hϵlion.

Methodology: We constructed GH from the HOME corpus (*i.e.*, from 40 users). In doing so, we added *causal* edges, *i.e.*, between triggers and actions of individual routines, and *temporal* edges, *i.e.*, among routines as they appear consecutively in a home automation sequence, from the action of

[2]We avoided the *down* flavor to reduce the number of questions.

the preceding routine to the trigger of the next. Note that the temporal edges are a concept that Hϵlion introduces (*i.e.*, with execution indicators), and adding them makes GH denser, which may improve its predictive ability. We also created GH_{56}, an instance of GraphHome that includes sequences from the 16 evaluators as well (*i.e.*, 56 users).

To query GH for generating a scenario, *i.e.*, predicting the n^{th} event using a given history of $n-1$ events, we search for a path in the graph that corresponds to the history (*i.e.*, up to the $n-1^{th}$ vertex), and predict the next vertex in the path. Using this approach, we tried to make one prediction for each of the 32 histories provided by the 16 evaluators, using both GH and GH_{56}, in both 3-gram and 4-gram configurations.

Results: The graph GH contains 233 vertices and 2,066 edges, while GH_{56} contains 291 vertices and 2,977 edges.

For most histories, GH simply did not return a prediction, *i.e.*, it was able to successfully predict an event for only 3/32 evaluator-provided histories (3-gram) and 1/32 histories (4-gram). GH_{56} performed slightly better for the 3-gram case, *i.e.*, it could predict for 6/32 histories (3-gram), but it could also only predict for 1/32 histories for the 4-gram case.

Finding 13 **($\mathcal{F}_{13}$): A predictive approach such as Hϵlion is better suited to generate valid scenarios based on user-provided histories.** GH and GH_{56} analyze event-sequences *verbatim*, *i.e.*, search for the exact path corresponding to the history in the model, and fail to predict when it is not found. The poor prediction rate of these models (*i.e.*, 3/32 predictions in the best case) is certainly not due to data sparsity. In fact, GH and GH_{56} are *denser* than Hϵlion's corpus; *e.g.*, GH contains 2,066 bigrams (vs 1,319 in HOME), 16,459 trigrams (vs 9,588), and 188,945 4-grams (vs 89,205).[3] Rather, the primary reason for the failure to predict is that human evaluators may often provide arbitrary histories that may not be in the corpus, and more importantly, may contain *unseen tokens*. Therefore, while GH_{56} performed slightly better than GH as it included the evaluators' data (*i.e.*, given the property discussed in $\mathcal{F}_{11}$), it mostly fails due to user-provided tokens/sequences not in the corpus. On the contrary, Hϵlion's *predictive* approach, and specifically its use of *smoothing* (Sec. III-D), allows it to extrapolate beyond the corpus and make reasonable predictions even when faced with arbitrary histories (as seen in $\mathcal{F}_{10}$).

Finally, while Hϵlion's predictions are precise, and point to the most probable event, approaches such as GraphHome are agnostic to event probabilities, and will generate a large range of potential candidates. This fundamental difference in approach allows Hϵlion to significantly reduce human effort in tasks that require evaluation of scenarios (*e.g.*, security policy specification), as we discuss further in Section VIII-B.

VIII. Evaluating the Usefulness of Hϵlion (RQ_3)

We now know that Hϵlion's scenarios are drawn from a *natural* corpus ($\mathcal{F}_6$), and are judged as *valid* by end-users ($\mathcal{F}_7$, $\mathcal{F}_{10}$, $\mathcal{F}_{11}$). This section demonstrates the usefulness of scenarios for security and safety. We focus on enabling security researchers to specify policies grounded in natural home automation (*i.e.*, implement the motivating example). Further, we explore the execution of scenarios via an execution engine, to provide initial insight into how platform/device vendors may benefit from Hϵlion. These are not the only ways to use Hϵlion; indeed, given Hϵlion's generative ability, the use cases for various stakeholders are boundless.

We develop two tools to enable these use cases, and foresee additional tools being developed with community support:

1. Snapshot Module: This module tracks the evolution of states of individual devices and the home, as events are executed in the home (*e.g.*, the "locked" state of the door lock, the home/away mode). The module starts with a default snapshot represented as a JSON object containing the various global states of the home as well as states of individual devices (see Appendix G for an example JSON object). As each event in the scenario is examined, the snapshot evolves, *i.e.*, one or more of the states in the JSON changes. Our implementation generates n successive snapshots for a scenario of length n. The analyst can examine a chosen subset of these snapshots (*i.e.*, the ones for security-sensitive events) for security, safety, or privacy violations. We plan to release a GUI-version of the snapshot module that will visualize the state of the home, and allow the analyst to dynamically change parameters such as flavors or the n-gram order (see [26] for a preview).

2. Execution Engine: To allow the execution of the scenarios predicted by Hϵlion, we built an execution engine on top of the SmartThings platform. We created device handlers (*i.e.*, software proxies for devices) for all the devices represented in the HOME corpus, by adapting existing handlers from the SmartThings repository [3] and creating new handlers when needed. Moreover, we instantiated each token in terms of one or more SmartThings events, a one-time effort for each unique token. The HOME corpus was represented with 158 unique events, using 45 SmartThings capabilities.

We connected 15 IoT devices to our execution engine (see list in Appendix F). Moreover, we provisioned virtual devices using the SmartThings IDE [31] to represent the devices not available in our lab. We created a *scheduler* SmartApp that has access to all capabilities, and executes scenarios using two parameters: (1) a sequence of tokens (*i.e.*, scenario), and (2) intervals between tokens. The scheduler executes tokens at specified intervals, logging all the events it executes (*i.e.*, the scheduler logs). The analyst can examine the logs provided by the SmartThings platform, along with the scheduler logs, to identify anomalies; *e.g.*, events that were executed by the scheduler but did not execute in reality, or vice versa.

Note that such a platform-specific execution module has a general limitation: certain tokens may represent events that may not be supported by the platform, and hence, may not be executable. We only observed 20/158 non-executable tokens; *e.g.*, tokens for medical IoT devices such as blood pressure monitors, which are unsupported in SmartThings, but which our users chose to integrate with other home automation.

[3]The graph models have more sequences than HOME, as edges in graphs can cause new sequences outside the corpus (see Appendix D for example).

A. Helping Researchers Generate Policies

Recall the motivating example in Section II-A, in which Alice's key problem was having to manually specify use/misuse cases, *i.e.*, not only does this process take significant effort, but Alice has no way to determine if her policies are indicative of real problems found in the wild. We demonstrate how using scenarios generated by Hϵlion addresses these challenges.

Enacting the motivating example and predicting policies: We had a security researcher with prior experience creating smart home policies using the use/misuse case requirements engineering approach (and also an author), try out Hϵlion's scenarios instead. The researcher used Hϵlion (4-gram model) to generate scenarios in four configurations: the *up* and *down* scenarios described in Sec. III-E, as well as two new *hybrid* configurations, *i.e.*, *up-down*, which predicts 1-3 *down* events for every 10 events in an *up* scenario, and *down-up*, which does the inverse of *up-down*. We sampled 260 unique histories from the sequences of 5 new users (*i.e.*, outside the HOME corpus) as input for Hϵlion. Each scenario consisted of 13 events, *i.e.*, 3 events from the history and 10 predicted events.

Results: The researcher analyzed each scenario using the snapshot module, detected 27 unique violations, and generated 17 policies from them. They spent only a few seconds on scenarios that had no interesting (*i.e.*, security/safety sensitive) events, and 1-3 minutes on interesting scenarios, *i.e.*, about 10 hours for the entire process. We discuss three salient policies below (see Appendix C for the full list):

- **Preventing a fire ($\mathbf{Pol}_1$):** In one scenario, we discovered that the *gas stove* was ON when the user was away. This safety violation could cause a fire, and is addressed with the policy: *the gas stove should be OFF when the user is away.*
- **Preventing an explosion ($\mathbf{Pol}_2$)** We discovered that the gas stove was switched ON after the smoke/CO detector detected smoke, *i.e.*, *when there is already a fire*. A gas leak and fire together could lead to a disastrous explosion, regardless of whether the user is home or away (*i.e.*, the violation is unsolvable using Pol_1).
- **Preventing an accidental privacy violation ($\mathbf{Pol}_3$)** We discovered that the indoor security camera would stay ON, even after the user got home, which is a privacy violation. Certain vendors (*e.g.*, NEST [32]) mitigate this problem by turning the camera OFF when the user is at home.

We now discuss our core findings from this experiment:

Finding 14 **($\mathcal{F}_{14}$): Scenarios significantly reduce effort, relative to a fully manual approach.** Specifying policies with Hϵlion required far less effort than a fully-manual approach of use/misuse case requirements engineering. To quote the researcher from this experiment: "*It's very convenient. The advantage of having the sequences is that they set up a likely or unlikely scenarios without me having to invent it.*"

Finding 15 **($\mathcal{F}_{15}$): All of Hϵlion's configurations, *i.e.*, *up, down, up-down and down-up*, are useful.** Each of our configurations contributed to the policies generated. In fact, half of the policies were obtained using a single configuration, which indicates the value of the *flavors* implemented in Hϵlion.

B. Baseline Comparison with GraphHome

We now compare the usefulness of Hϵlion for policy specification, relative to GraphHome (Sec. VII-C).

Methodology: We created a new instance of GraphHome, GH_{45}, which includes the sequences from the 5 new users (which were also used to create histories (Sec. VIII-A). GH_{45} contains 271 vertices and 2,576 edges. Further, we query both GH and GH_{45} for the 260 unique histories used in Sec. VIII-A.

Results: GH could predict for 32/260 histories. GH_{45} could predict for all 260 histories, which is expected, because the histories were directly sampled from *known sequences*, *i.e.*, the sequences of the 5 test users. However, these predictions were imprecise, *i.e.*, both GH and GH_{45} predicted on average *13 possible next events* for each history (in the worst case, 51 predictions by GH for a single history, and 68 by GH_{45}).

Finding 16 **($\mathcal{F}_{16}$): A predictive approach allows for precise event generation, and reduces effort, compared to a probability-agnostic approach.** Even in the average case, GH_{45} would return 13 possible predictions for each query, all equally likely according to the model, as the graph-based approach is agnostic to event probabilities. Hence, even in the best case scenario, a researcher would have to evaluate 13 potential cases for every history when using GraphHome to generate policies, instead of the single event generated by Hϵlion. This imprecision would only compound, as more future predictions are made in parallel for each of the 13 potential paths in the graph. That is, the number of potential future predictions is *13x or multiplicative at best*, and exponential at worst. Note that Hϵlion would not display such performance, as it would require all potential sequences (*i.e.*, unique 4-grams in the corpus) to be equally probable. Such a situation is highly unlikely, considering the diversity of the corpus, which is drawn from different users, and consists of month-long sequences generated using informed scheduling of user-specific execution indicators (*i.e.*, ensuring *non-uniform* repetition). In other words, *relative to a probability-agnostic graph-based approach, Hϵlion's predictive approach reduces the researcher's effort by 13x*, by generating the most probable (*up*) or improbable (*down*) event.

C. Detecting flaws in platforms and devices

We performed an initial investigation into the usefulness of scenarios in helping vendors evaluate platforms or partner devices in realistic situations. On executing a random set of less than 10 scenarios in our execution engine, we discovered three flaws (two platform and one device):

- **Dropped actions ($\mathbf{Flaw}_1$):** On seemingly random occasions, the SmartThings platform was not executing certain events, including security-sensitive events such as locking the door.
- **Zombie SmartApps ($\mathbf{Flaw}_2$):** We observed that routines that were previously "deleted" using the SmartThings mobile app [33] were persistently executing in the background,

and could only be deleted from the Web IDE [31]. This can lead to disastrous consequences if the routine were vulnerable, or a malicious SmartApp.

- **Unsafe Auto-relock default (Flaw$_3$):** We discovered that the Yale lock [34] would stay in the unlocked state after the door closed, unless manually re-locked. This *auto-relock* feature is standard in keypad locks, but implemented as an inconsistent default in most smart lock brands, and worse, not presented to the user during setup with SmartThings.

While we independently discovered Flaw$_1$ and Flaw$_2$, they were also previously reported on the SmartThings forum by users [35], [36], which further speaks to the promise of Hϵlion's scenarios in uncovering naturally occuring problems.

IX. Related Work

Home automation is a complex domain, where home automation sequences *created* by users manifest themselves in a highly contextual manner. Inspired by prior work in software engineering [14], we attempt to demystify this domain, by developing an approach that capture the non-obvious regularities in user-driven home automation, and generates actionable scenarios, which are complementary to prior and future research in this domain.

For instance, prior systems designed to evaluate or enforce the safety/security of home automation have one trait in common: their reliance on safety/security properties that are manually specified, either by users (*e.g.*, in case of Menshen [37] and AutoTAP [38]) or domain experts (*e.g.*, IoTSAN [10], Soteria [4], IoTGuard [9], or ProvThings [7]). The practical policy specification approach enabled by Hϵlion's scenarios (Sec. VIII-A) would assist experts in deploying such systems, by helping them develop policies that are relevant in realistic scenarios, with significantly less effort.

Similarly, recent work has explored the nature of end-user error within user-driven routines [39], [40], and developed methods to correct them [41], [42]. Hϵlion has a symbiotic relationship with this research, as our scenarios could lead to the discovery of new bugs/faults not yet considered by researchers, while Hϵlion will benefit from the resultant improvements in user-driven routines and the processes to specify them.

The natural perspective provided by Hϵlion may benefit an array of general security systems and analyses proposed for the smart home [6], [8], [5]. For instance, ContexIoT [6], which evaluates the frequency of its prompts using random events generated by fuzz testing of IoT apps, could use Hϵlion's scenarios as more realistic input. Additionally, Hϵlion's *down* flavor can directly contribute to benchmarks such as IoT-Bench [4]. Finally, Hϵlion's scenarios may be executed as test cases to dynamically measure private information exposure by IoT devices [43] in realistic environments.

X. Lessons Learned

The value of this work is in demonstrating the *feasibility* of generating natural, valid, and useful scenarios of home automation. While we acknowledge that our data (and hence findings) may or may not be representative of all users (*e.g.*, scenarios of $n \geq 3$ may seem natural to our users, but could be rare in the wild), our investigation provides evidence to motivate targeted, large-scale, studies into each of our 16 findings, which we distill into the following lessons:

Lesson 1: Home automation is natural, and can be leveraged to generate valid scenarios: User-driven home automation is implicitly natural and exhibits predictable patterns ($\mathcal{F}_6$), which can be modeled with statistical LMs to generate scenarios that are reasonably valid ($\mathcal{F}_7$, $\mathcal{F}_{10}$, $\mathcal{F}_{11}$). These results serve as a strong foundation for applying advanced statistical modeling techniques for home automation security.

Lesson 2: A predictive approach is better suited for generating scenarios: A probability-agnostic formal model-based approach such as GraphHome may enable accurate policy *checking* [4], [10], [9]; however, for a *generative* task such as policy specification, a predictive approach may be more suitable, due to its ability to capture the regularities in home automation, and extrapolate on them to provide precise predictions ($\mathcal{F}_{10}$, $\mathcal{F}_{16}$) under arbitrary user input ($\mathcal{F}_{13}$).

Lesson 3: Hϵlion can be used to generate useful policies and detect real flaws: Hϵlion's scenarios lead to useful policies ($\mathcal{F}_{14}$). Moreover, all prediction flavors prove useful ($\mathcal{F}_{15}$), and our experiments indicate the inclination of *down* towards generating unsafe scenarios ($\mathcal{F}_8$). Finally, executing Hϵlion's scenarios may be useful for detecting platform/device-level flaws that occur in the wild (Sec. VIII-C).

Lesson 4: Understanding user-driven home automation enables new research opportunities: Our investigation reveals new problems, such as understanding context-shifts in the home ($\mathcal{F}_{12}$), or capturing unpredictable execution indicators ($\mathcal{F}_5$), which motivate further research; *e.g.*, future work could attempt to understand such shifts via a large-scale analysis of user-driven routines, or model them by encoding temporal information in home automation sequences.

Lesson 5: We need to study more than IoT apps: Our empirical analysis confirms that routines are important for users ($\mathcal{F}_1$), and that users express a strong preference for creating their own routines ($\mathcal{F}_3$). Importantly, we observe a significant mismatch between the available IoT apps and routines created by our users ($\mathcal{F}_2$), which indicates that users may favor easily-created routines over IoT apps. Thus, user-driven home automation may introduce a set of largely unexplored security implications that may not be apparent from just analyzing IoT apps. This paper presents an important take away for future research in home automation security: unlike past research in app-based platforms, the methodology of characterizing the environment *solely* based on marketplace apps is not viable for characterizing home automation.

Acknowledgements

The authors have been supported in part by the NSF-1815336 grant and the William & Mary Summer Research Award. Any opinions, findings, and conclusions expressed herein are the authors' and do not reflect those of the sponsors.

References Cited

[1] SmartThings Support, "Routines in the SmartThings Classic app," https://support.smartthings.com/hc/en-us/articles/205380034-Routines-in-the-SmartThings-Classic-app, Accessed June 2018.

[2] Nest Labs, "Nest Developers," https://developers.nest.com///, Accessed June 2018.

[3] Samsung, "Samsung SmartThings SmartApp Public Repository," https://github.com/SmartThingsCommunity/SmartThingsPublic, 2018.

[4] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated IoT Safety and Security Analysis," in *2018 USENIX Annual Technical Conference (USENIX ATC)*, 2018, pp. 147–158.

[5] W. Ding and H. Hu, "On the Safety of IoT Device Physical Interaction Control," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Oct. 2018, pp. 832–846.

[6] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, A. Prakash, and S. J. Unviersity, "ContexIoT: Towards providing contextual integrity to appified IoT platforms," in *Proceedings of the 2017 Network and Distributed System Security Symposium (NDSS)*, 2017.

[7] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and Logging in the Internet of Things," in *Network and Distributed Systems Symposium*, 2018.

[8] Z. B. Celik, L. Babun, A. K. Sikder, H. Aksu, G. Tan, P. McDaniel, and A. S. Uluagac, "Sensitive Information Tracking in Commodity IoT," in *Proceedings of the 27th USENIX Security Symposium (USENIX)*, Aug. 2018.

[9] Z. B. Celik, G. Tan, and P. McDaniel, "IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT," in *2019 NDSS Symposium*, 2019, *To appear.*

[10] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. M. Colbert, and P. McDaniel, "IotSan: Fortifying the Safety of IoT Systems," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2018, pp. 191–203.

[11] Yeti, "Yeti - Simplify the control of your smart home," https://getyeti.co/, Accessed June 2018.

[12] Yonomi, "Yonomi app – Yonomi," https://www.yonomi.co, Accessed June 2018.

[13] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts: The MIT Press, 1999.

[14] A. Hindle, E. Barr, Z. Su, M. Gabel, and P. Devanbu, "On the naturalness of software," in *International Conference on Software Engineering (ICSE'12)*, 2012, pp. 837–847.

[15] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *Security and Privacy (SP), 2016 IEEE Symposium on*, May 2016, pp. 636–654.

[16] K. Kafle, K. Moran, S. Manandhar, A. Nadkarni, and D. Poshyvanyk, "A Study of Data Store-based Home Automation," in *Proceedings of the 9th ACM Conference on Data and Application Security and Privacy (CODASPY)*, Mar. 2019.

[17] D. J. Cook, A. S. Crandall, B. L. Thomas, and N. C. Krishnan, "Casas: A smart home in a box," *Computer*, vol. 46, no. 7, pp. 62–69, 2012.

[18] N. Lindsey, "Consumers Still Concerned About IoT Security and Privacy Issues," https://www.cpomagazine.com/data-privacy/consumers-still-concerned-about-iot-security-and-privacy-issues/, Accessed June 2019.

[19] J. Sattler, "Privacy Concerns Cooling IoT Adoption in the US and Europe," https://blog.f-secure.com/privacy-concerns-cooling-iot-adoption-us-europe/, Accessed June 2019.

[20] M. Surbatovich, J. Aljuraidan, L. Bauer, A. Das, and L. Jia, "Some Recipes Can Do More Than Spoil Your Appetite: Analyzing the Security and Privacy Risks of IFTTT Recipes," in *Proceedings of the 26th International Conference on World Wide Web*, Apr. 2017, pp. 1501–1510.

[21] M. Harrison, W. Ruzzo, and J. Ullman, "Protection in Operating Systems," *Communications of the ACM*, vol. 19, no. 8, pp. 461–471, Aug. 1976.

[22] A. Nadkarni and W. Enck, "Preventing Accidental Data Disclosure in Modern Operating Systems," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, Nov. 2013.

[23] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman, "Practical trigger-action programming in the smart home," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014, pp. 803–812.

[24] SmartThings Developer Documentation, "Capabilities Reference," https://docs.smartthings.com/en/latest/capabilities-reference.html, Accessed December 2018.

[25] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech & Language*, vol. 13, no. 4, 1999.

[26] Sunil Manandhar, "Hєlion Data and Source Code," https://github.com/helion-security/helion, 2020.

[27] Smartthings Developers, "Documentation," http://developer.smartthings.com/, Accessed June 2018.

[28] Nest Labs, "Works with Nest," https://nest.com/works-with-nest//, Accessed June 2018.

[29] Bo-June Hsu, "MIT Language Modeling Toolkit," https://github.com/mitlm/mitlm, Accessed December 2018.

[30] M. Rahman, D. Palani, and P. Rigby, "Natural software revisted," in *Proceedings of the 41st International Conference on Software Engineering Companion*, ser. ICSE '19. Montreal, QC Canada: IEEE Press, 2019, p. to appear.

[31] SmartThings, "SmartThings Web IDE," https://graph.api.smartthings.com, Accessed February 2019.

[32] Nest Labs, "Meet the Nest app," https://nest.com/app/, Accessed Feb 2019.

[33] SmartThings, "SmartThings Classic App," https://play.google.com/store/apps/details?id=com.smartthings.android, Accessed February 2019.

[34] ——, "Yale Assure Lock with Bluetooth (Zigbee)," https://www.smartthings.com/products/yale-assure-lock-with-bluetooth-zigbee, Accessed December 2018.

[35] SmartThings Community, "Execution Times Increasing," https://community.smartthings.com/t/execution-times-increasing/19979, Accessed December 2018.

[36] ——, "Help! Old SmartApp automation still running," https://community.smartthings.com/t/help-old-smartapp-automation-still-running/70213, Accessed December 2018.

[37] L. Bu, W. Xiong, C.-J. M. Liang, S. Han, D. Zhang, S. Lin, and X. Li, "Systematically ensuring the confidence of real-time home automation iot systems," *ACM Trans. Cyber-Phys. Syst.*, vol. 2, no. 3, Jun. 2018.

[38] L. Zhang, W. He, J. Martinez, N. Brackenbury, S. Lu, and B. Ur, "Autotap: Synthesizing and repairing trigger-action programs using ltl properties," in *Proceedings of the 41st International Conference on Software Engineering*, 2019, pp. 281–291.

[39] W. Brackenbury, A. Deora, J. Ritchey, J. Vallee, W. He, G. Wang, M. L. Littman, and B. Ur, "How users interpret bugs in trigger-action programming," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019.

[40] M. Palekar, E. Fernandes, and F. Roesner, "Analysis of the susceptibility of smart home programming interfaces to end user error," in *IEEE Workshop on the Internet of Safe Things (SafeThings)*, ser. SafeThings'19. New York, NY, USA: ACM, 2019.

[41] C. Nandi and M. D. Ernst, "Automatic trigger generation for rule-based smart homes," in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*. ACM, 2016, pp. 97–102.

[42] F. Corno, L. De Russis, and A. Monge Roffarello, "Empowering end users in debugging trigger-action rules," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 388:1–388:13.

[43] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach," in *Proceedings of the Internet Measurement Conference*, 2019, pp. 267–279.

Appendix

A. Additional Survey Questions

Aside from collecting routines and execution indicators, we asked users additional questions during the survey, illustrated in Figures 12, 11, 10, and 13.

B. Survey Instrument for the Routine Comparison and Sequence Generation Studies

This section provides the survey instrument for the routine comparison (Sec. VII-A) and sequence generation (Sec. VII-B) studies. Specifically, Figure 14 illustrates how routines were shown to evaluators in Sec. VII-A. Similarly, Figure 15 shows a sequence that one of the evaluators was shown, in Sec. VII-B. Finally, we asked evaluators to provide additional feedback regarding why they rated some routines/sequence as reasonable, as shown in Figure 16.

C. Policies generated using Hϵlion

Table IV provides the security/safety policies generated using Hϵlion.

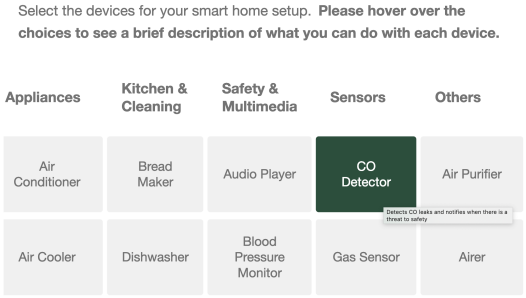

Fig. 4. Device Selection Screen

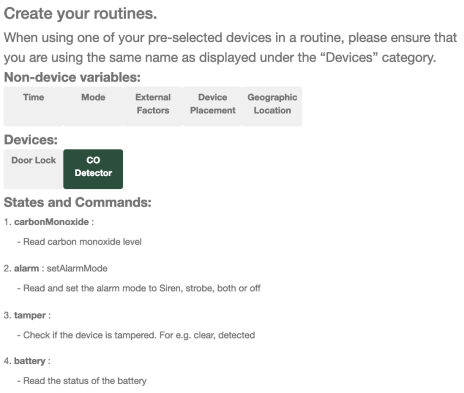

Fig. 5. Additional Information about each device

D. Number of n-grams in GraphHome vs Hϵlion

This section illustrates why the GraphHome models (*i.e.*,GH, GH_{56}, and GH_{45}) are generally denser than Hϵlion's model, even when both are built from the same set of home automation sequences (*e.g.*, the HOME corpus).

Consider a corpus composed of the two sequences below:

$$\mathbf{S}_1\text{: } A \rightarrow B \rightarrow C$$
$$\mathbf{S}_2\text{: } C \rightarrow D \rightarrow E$$

Figure 17 shows the directed graph that we would create from the example corpus (using the methodology in Section VII-C), for use with GraphHome. If we examine the

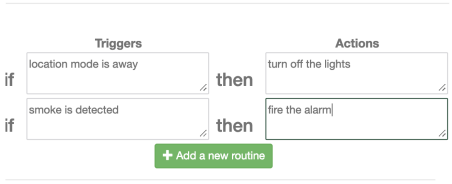

Fig. 6. Routine Creation

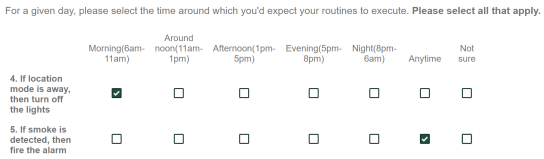

Fig. 7. Execution Indicator: Time of the Day

corpus as a collection of individual sentences, as Hϵlion does, we compute the following number of n-grams:

bigrams: 4 (AB, BC, CD, DE)
trigrams: 2 (ABC, CDE)
4-grams: zero

However, if we compute the numbers from the graph (*i.e.*, as we would query GraphHome), we get

bigrams: 4 (AB, BC, CD, DE)
trigrams: **3** (ABC, BCD, CDE)
4-grams: **2** (ABCD, BCDE)

4 bigrams as well, but **3** trigrams, and **2** 4-grams, which is more than what Hϵlion would observe. Intuitively, the reader may realize that the increased density is because the independent sentences may become intertwined in the graph (*i.e.*, connected due to the subsequences B→C from $\mathbf{S}_1$ and C→D from $\mathbf{S}_2$ in this case).

This increased density should increase the predictive power of GraphHome, as it can observe more higher-order n-grams. However, our analysis in Sec. VII-C demonstrates that GH and GH_{56} fail to predict for most user-provided histories ($\mathcal{F}_{13}$), as they analyze the graph verbatim, and do not extrapolate to predict for n-grams that may not be present in the graph.

Finally, while the number of bigrams seen by Hϵlion or GraphHome remain the same in this section, in reality, the bigrams (*i.e.*, edges) in the graph may increase due to tokens that consist of a conjunction of events (*i.e.*, as described previously in Sec. IV-B). That is, such conjunctions may cause tokens to be supersets of other tokens, in a way (*e.g.*, a token <A & X> is a superset of another token that is simply <A>). As a result, if there is an edge/sequence A→B in the corpus, then we also add an extra edge <A & X>→<B> to the graph, because if <A & X> is true, it means that <A> is also true. For example, if the conjunction "Light is ON *and* motion is detected" is true, then the individual event "Light is ON" is also true at the same time, meaning that any edges from the latter to other vertices must also be created from the former (*i.e.*, the conjunctive token), leading to extra bigrams.

TABLE IV
LIST OF SMART HOME VIOLATIONS, THE CORE PROBLEM THEY REPRESENT, AND THE CORRESPONDING POLICIES.

Violation	Problem	Policy	Flavor
- Gas stove on when user away. - Gas stove is on when user is on vacation.	Gas stove is on when user is not at home.	**(Pol$_1$)** Gas stove should not be on when the user isn't home.	*down, up-down, down-up*
- Gas stove is on when smoke has been detected previously.	Gas stove is on after smoke detector detects smoke.	**(Pol$_2$)** Gas stove should not be on when smoke is detected.	*up*
- Security camera is taking images when motion is detected. However, the user is home, which justifies motion as well as potentially violates user's privacy. - Security camera on when user is home. - Security camera is on and taking images when presence sensor detects that user is present at home	Security camera is on when the user is at home.	**(Pol$_3$)** Security camera should be off when the user is home to preserve privacy.	*up, down, up-down, down-up*
- Door is opened when user is away, but the user doesn't receive a notification. - Door is opened when user is in vacation, but user isn't notified. - Door is opened but the presence sensor detects that user isn't present at home and user is not notified.	Door sensor senses that the door is open when user is not at home.	**(Pol$_4$)** User should be notified if the door sensor senses the door is open when the user is away.	*up, up-down*
- Window opened when user away, but the user isn't notified.	Contact sensor senses that the window is open when user is not at home.	**(Pol$_5$)** User should be notified if the contact sensor senses the window is open when the user is away.	*up, down*
- Water valve closed when fire sprinkler on.	Water valve is closed when fire sprinkler turns on.	**(Pol$_6$)** Water valve should not be closed if the fire sprinkler is on as a reaction to fire/smoke.	*down, down-up*
- Air purifier turns off when CO is detected	Air purifier turns off when CO detector detects CO.	**(Pol$_7$)** Air purifier should not turn off automatically if CO is detected.	*down*
- Shades/Blinds are opened in the morning, but the user is away. - Shades/Blinds opened in vacation mode.	Shades/Blinds open when user is away.	**(Pol$_8$)** Window shades should not open when user is not home.	*up, down, down-up*
- Door is unlocked after gas level is detected to be high or alarm going off, supposedly as a safety measure, but user is away. - Door unlocked in vacation mode. - Door is unlocked state but presence sensor is detecting that user isn't present at home.	Door is unlocked when user is not at home.	**(Pol$_9$)** User should be prompted before the door is unlocked automatically for any reason, when user is away.	*down, up-down, down-up*
- Door stays unlocked even after user leaves home.	Door lock stays unlocked when user leaves home.	**(Pol$_{10}$)** Door should lock automatically when mode changes from home to away.	*down*
- Door remains unlocked when the sleep monitor detects that the user is sleeping.	Door is unlocked when user is sleeping.	**(Pol$_{11}$)** Door should be locked when the bedroom sleep monitor detects that user is sleeping to ensure safety.	*up-down*
- The sleeping monitor detects the user as sleeping, and the garage door is open.	Garage door is open when user is sleeping.	**(Pol$_{12}$)** Garage door should be closed when bedroom's sleep monitor detects that user is sleeping.	*up-down*
- The induction cooktop is on after the sleep monitor detects that the user is sleeping.	Electric appliance which is a potential fire hazard is on when user is sleeping.	**(Pol$_{13}$)** Induction cooktop should not be on when user is sleeping.	*up-down*
- Garage door opened when user is away. - Garage door opened when user in vacation mode.	Garage door is open when user is not at home.	**(Pol$_{14}$)** Garage door should be closed when user is not home.	*down, up-down, down-up*
- Glass break is detected when user is away but user is not notified. - Glass break is detected in vacation mode but user isn't notified.	Glass break is detected but the user is not notified.	**(Pol$_{15}$)** User should be notified when glass break is detected.	*down-up*
- Security alarm turned off when smoke is detected and user is away.	Security system turned off when smoke detector detects smoke.	**(Pol$_{16}$)** Security alarm should not be off when user isn't home.	*down-up*
- Fire sprinkler on when there is no fire.	Fire sprinkler is on for no reason.	**(Pol$_{17}$)** Fire sprinkler should only be on when there's fire detected in the home.	*up-down, down-up*

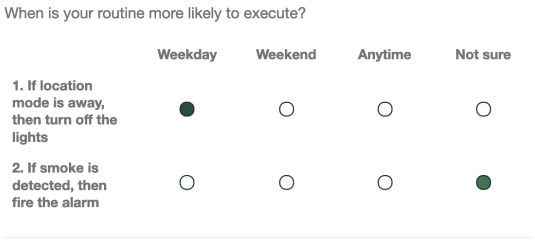

Fig. 8. Execution Indicator: Time of the week

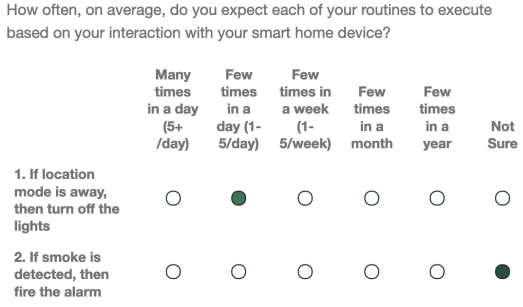

Fig. 9. Execution Indicator: Frequency

E. Background on Statistical LMs as used in Hɛlion

A statistical LM, fundamentally, measures the probability of a sentence, given the probabilities of the individual words in the sentence, previously estimated from a training corpus. That is, traditionally, a statistical LM is defined as a probability estimation over units of written/spoken language, which measures the probability of a sentence $s = w_1^m = w_1 w_2 ... w_m$ based on word probabilities. This ability can also enable prediction, *i.e.*, by predicting the next most probable word that can follow a sequence of words. In the context of modeling smart home routines, we define a "sentence" to represent a sequence of home automation events, wherein the "words" (a.k.a *tokens*)

Fig. 10. Importance of routines to the users

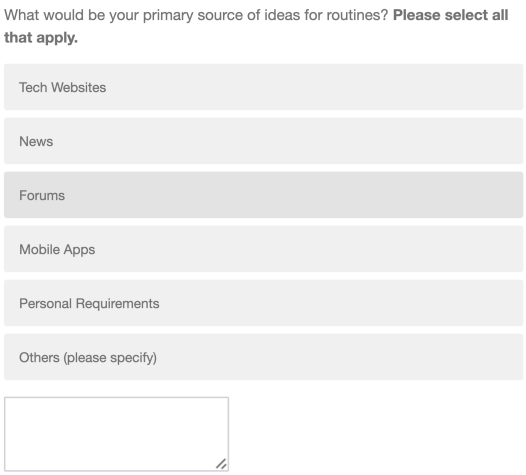

Fig. 11. Sources of ideas for routines

are atomic smart home events (*e.g.*,<LightBulb, switch, ON>).

Thus, our approach will measure the probability of an event sequence $s = e_1^m = e_1 e_2 ... e_m$ according to its constituent events e, relying on the chain rule of probability, as follows:

$$\begin{aligned} p(e_1^m) &= p(e_1)p(e_2|e_1)p(e_3|e_1^2)...p(e_m|e_1^{m-1}) \\ &= \prod_{i=1}^{m} p(e_i|e_1^{i-1}) \end{aligned} \tag{1}$$

The n-gram language model: In practice, however, there are often too many unique sequences to properly estimate the probability of tokens given long histories, even with large training corpora. Thus, we make use of the *n-gram* language model, which assumes the *Markov property*, *i.e.*, instead of computing the conditional probability given an entire event or language history, we can *approximate* it by considering only a few tokens from the past. The intuition behind n-gram language models applied to natural language is that shorter sequences of words are more likely to co-occur in training corpora, thus providing the model with more examples to condition token probabilities, enhancing its predictive power. While n-gram models are valued for their practicality in NLP and software analysis contexts, they are arguably an even more intuitive fit for analyzing home automation event sequences. This is due to the organic semantic relationships smart-home events exhibit, *i.e.*, the localized *causal* relationship between triggers and actions (*e.g.*, a trigger preceding a corresponding action or set of actions), which are more relevant than presumably weaker correlations with events from the distant past. Using the n-gram model, we estimate the probability of the event sequence $s = e_1^m = e_1 e_2 ... e_m$ as follows:

$$p(e_1^m) = \prod_{i=1}^{m} p(e_i|e_1^{i-1}) \approx \prod_{i=1}^{m} p(e_i|e_{i-n+1}^{i-1}) \tag{2}$$

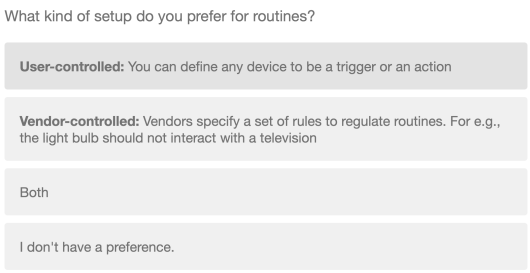

Fig. 12. Setup preferred by the users to create routines

Evaluating the naturalness of a corpus: The effectiveness of n-gram modeling in the context of smart home events holds only if our intuition regarding the naturalness home automation event sequences drawn from user-directed routines is correct. Thus, we must answer the question: *Are such event sequences natural?* Fortunately the naturalness of token sequences can be measured according to a trained model's *perplexity* (or its log-transformed version, *cross-entropy*) on unseen data. These are standard metrics used to test the viability of statistical language modeling for modeling any corpora. A trained model will be "perplexed" upon observing a new sequence if it finds the sequence surprising, *i.e.*, unlike any sequence observed in the corpora. Thus, if a domain is natural, then the perplexity of a model built on corpora from the domain (*e.g.*, home automation event sequences from a population of users) when applied to new sequences from the same domain should be measurably low. That is, the model should be able to identify regularities in the event sequences, and hence, predict new sequences with confidence. The cross-entropy H of an n-gram model M can be computed as follows:

$$H_M(e_1...e_n) = -\frac{1}{n}\log_2 P_M(e_1...e_n) \tag{3}$$

where H_M is the average negative log probability that M assigns to each event e_n in a test sequence. Perplexity is 2^{H_M}.

Predicting with an LM, with conflicting data: As Hєlion collects routines from users, it is indeed possible that multiple users might create conflicting routines (*e.g.*, opposite actions on the same trigger). If literally analyzed (*e.g.*, in a graph-based model such as GraphHome), these routines would lead to conflicting predictions. However, conflicting predictions would be highly unlikely in Hєlion due to (1) its process of informed scheduling, and (2) due to the nature of statistical LMs. That is, first, even if two users have conflicting routines, the execution indicators and informed scheduling introduce further variance in the sequences generated for each user (otherwise, the HOME corpus would not have a non-zero entropy). Second, statistical LMs assign sequence probabilities based on how many times a sequence appears in a corpus, i.e.,

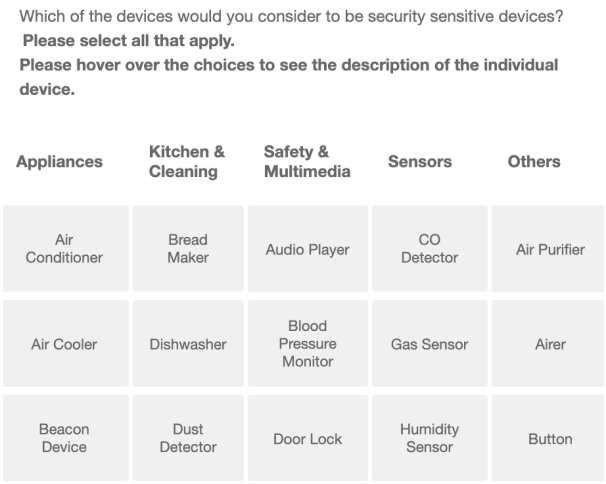

Fig. 13. Device-selection screen to select security-sensitive devices

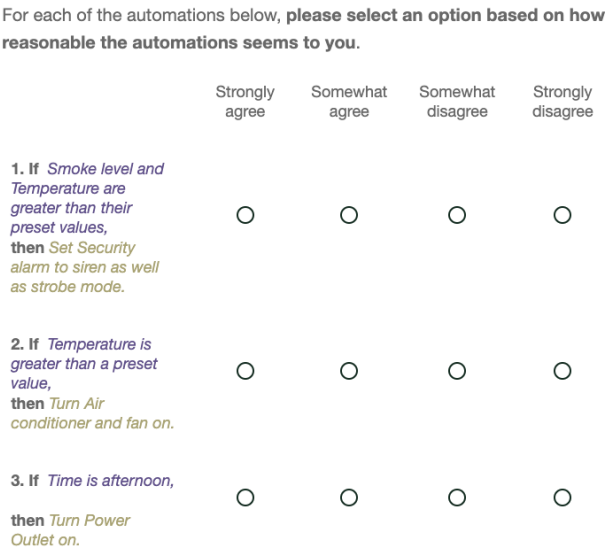

Fig. 14. Screen to validate routines

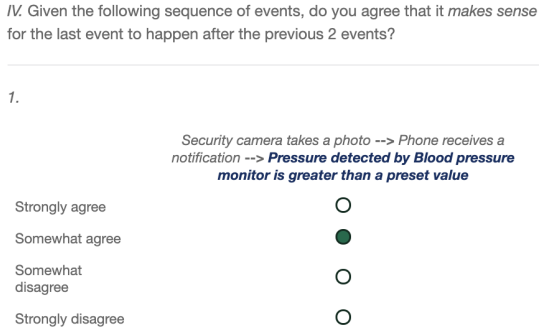

Fig. 15. Screen to validate sequences

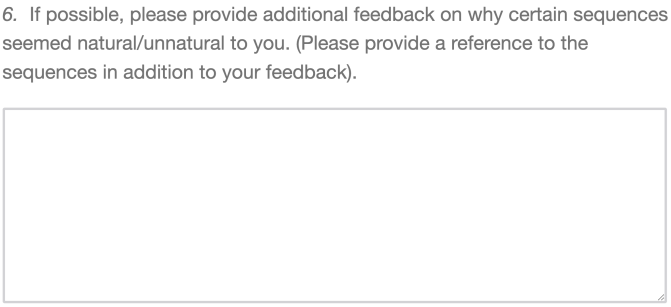

Fig. 16. Screen to provide additional feedback

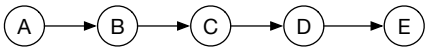

Fig. 17. An example of a directed graph created for GraphHome

the majority sequence always wins.

F. Execution Engine Device List

We integrated the following devices in our SmartThings-based execution engine: (1) AeoTec MultiSensor, (2) Amazon Alexa, (3) Arlo Security Camera, (4) First Alert Smoke and CO detector, (5) Fortrezz Alarm, (6) Google Home, (7) iHome Switch, (8) Philips Hue Light, (9) Ring DoorBell, (10) SmartThings Motion Sensor, (11) SmartThings MultiPurpose Sensor, (12) SmartThings Switch, (13) Tp Link-Kasa Switch, (14) Wemo SWITCH, and (15) Yale Assure Lock. The devices were chosen primarily in terms of their popularity, and to ensure a diverse collection of device types.

G. Example JSON object used in the Snapshot Module

Figure 18 shows an example of a JSON object used in the snapshot module. The states unaffected by the scenario are


```
{
  "null": {
    "locationMode": "night"
  },
  "Dishwasher": {
    "mode": "auto"
  },
  "Lighting_Remote": {
    "switch": "on"
  },
  "Thermostat": {
    "thermostat": "auto"
  },
  "Presence_Sensor": {
    "presence": "present"
  },
  "Door_Lock": {
    "lock": "locked"
  }
}
```


Fig. 18. An example JSON object used in Hєlion's snapshot module.

not shown in the object for brevity and ease of analysis. In the future, we plan to release a UI-version of the module that visualizes this object, and also allows the analyst to dynamically change model parameters such as the n-gram, or the flavors, and affect prediction (see [26] for a preview of the planned UI).