Array 14 (2022) 100146

Contents lists available at ScienceDirect

Array

journal homepage: www.elsevier.com/locate/array

Enhancing cybersecurity by generating user-specific security policy through the formal modeling of user behavior

Arwa AlQadheeb [a], Siddhartha Bhattacharyya [a,*], Samuel Perl [b]

[a] Computer Engineering and Sciences, Florida Institute of Technology, Melbourne, FL, 32901, USA
[b] Software Engineering Institute , Carnegie Melon University, Pittsburg, PA, 15213, USA

ARTICLE INFO

Keywords:
Zero Trust
Automatic verification
Correctness
Cybersecurity policy
Security policy
Formal methods
User behavior
Automated security policy generation
Finite-State Automata
Timed Computation Tree Logic

ABSTRACT

Organizations today are faced with the difficult challenge of balancing the embrace of new and emerging technology, and securing their systems and data that support critical business functions. Although there have been significant advances in security enforcement technology, attackers are still able to compromise organizations and access. The impacts of computer intrusions have become so untenable that many organizations are looking at a drastic rethinking of their approach to the security of internal networks. This approach is called *Zero Trust* and it seeks to remove all notion of a trusted internal network boundary. The benefits of *Zero Trust* include significantly increasing the work that attackers would need to perform to achieve their objectives. But *Zero Trust* will also increase the management complexity for internal security teams. These teams will need a way to collect data and enforce policy decisions based upon analysis. This process will need to be done for all organizational systems, and data, and it will need to be done in all access contexts.

Our approach uses formal methods to model and examine end-users security-related behaviors. Researchers have found that the users' security decisions correlate with factors including demographics, personality traits, decision-making styles, and risk-taking preferences. We describe these behaviors by using Finite-State Automata (FSA). This allows for the automated formulation of linear-time security properties based on Timed Computation Tree Logic (TCTL). The logic is then used to check the satisfaction of collected and observed security behaviors against policy. This formal behavioral analysis could be combined with other security and network data during the context analysis process that needs to occur for each *Zero Trust* access request. Other network or host security data could include address identifiers, tokens, event data, packet inspection, running process data, cyber threat intelligence, and much more. Our method allows organizations that embrace a *Zero Trust* philosophy to generate context specific security policies that can be automatically verified for correctness and completion.

1. Introduction

Organizations today are faced with the difficult challenge of balancing the embrace of new and emerging technology, and securing their systems and data that support critical business functions. Although there have been significant advances in security enforcement technology, attackers are still able to compromise organizations and access. Perhaps more now than ever before, Organizations are required to embrace new technical innovations including advances in the Internet of Things, Machine Learning and Statistical Analysis, and Cloud Computing. As these new devices and services become available, organizations have a business need to connect them to their business networks. If they do not, they risk becoming overtaken in their market by competitors. This further tests their security teams ability to prevent new intrusions because the attack surface changes quickly and the software supply chain becomes ever more complex.

Security professionals are developing new security models to be better prepared to Prevent, Detect, Respond, and Recover from cyberattacks. In 2010, John Kindervag, a Security and Risk Principal Analyst at Forrester Research Inc., developed a *Zero Trust* security model that radically prioritizes a classic security principles; "never trust, always verify" [1]. The rules of operation for how a *Zero Trust* network operates are different from traditional security models. They reflect lessons learned from over a decade of dealing with intrusions and observing the adversaries tactics, techniques, and procedures.

Organizations considering *Zero Trust* will also need to pay attention to the human behaviors [2]. The natural structure of humans

* Corresponding author.
E-mail address: sbhattacharyya@fit.edu (S. Bhattacharyya).

https://doi.org/10.1016/j.array.2022.100146
Received 10 June 2021; Received in revised form 15 February 2022; Accepted 1 April 2022
Available online 9 April 2022

psychology, the limitation of humans' information processing capacity, and their reliance on previous experiences to take future action can impede their security choices [3]. The Ponemon Institute study based on interviewing 507 companies across the globe reveals that accidents among users is the leading cause of 24% of data breaches, which are worth, on average, around $3.5 million of financial damage. Unintentional and accidental breaches result from so-called inadvertent insiders who had experienced a successful phishing attack or had their devices infected, lost, or stolen. According to the Ponemon Institute, human errors in cybersecurity takes organizations around 181 days and 61 days to identify and contain, respectively, a data breach that is related to such careless human action [4].

Most computer systems are designed based on the general perception of that all users in an access group will behave similarly. This overlooks the idea that individuals security behaviors differ from each other [5]. Researchers in human factors, have examined the psychology of humans to understand users' different behaviors toward privacy and security. Some works show that individual differences in demographics and psychological constructs (e.g., personality traits, decision-making styles, and risk-taking preferences) have a significant relationship with security behaviors and privacy attitudes. One research study found Norwegians and Japanese tend to browse the Internet more cautiously than their counterparts in Italy and Spain, thus lower their chances of randomly encountering malware-infected websites [6].

The prominent hacker Kevin D. Mitnick, states that the most effective technique to break into a company's system is to try exploiting the weakest link that is humans. From his point of view, the only reliable approach to overcome this problem is to combine security technologies with firm security policies along with proper education programs and training sessions for users [7]. As long as users are the weakest link in cybersecurity, cunning adversaries will continue to seek out and exploit users' vulnerabilities in almost any information security system via every devious possible way. The issue with Mitnick's solution is how to automate the process of having security policies that help with users' security misbehavior and poor decisions. In our research, we offer an ideal approach to impose more control over the user by analyzing the security behavior and then generating user-specific policy within a *Zero Trust* environment. Our goal is to generate user specific policies based upon security behaviors measured with scales in existing academic literature studies. The question we address in our research is:

Problem Statement How to generate explicit security policy after observing and analyzing end user security behaviors along with other assumed *zero trust* security enforcement decisions?"

Our method demonstrates the use of Formal Method based framework for generating user specific security policy that suggests a new contribution to the extant literature.

The remainder of this research is organized as follows: Section 2 describes in greater depth related work. Section 3 presents the research methodology, outlining the sequential development process for the automated approach. Section 4 discusses the selection of user security behavior. Section 5 discusses the model considerations and the selected modeling paradigm. Section 6 discusses the overall experiment performed with selection and modeling of the security behaviors in Section 6.1, Section 6.2 explains the checking of satisfaction of user security behavior. Section 6.3 defines the rule of generating a user-specific policy. Section 7 discusses the final results obtained by using our framework. Finally, Section 8 states our conclusions and future work.

2. Related work

Formal method based approach has been applied to the modeling and analysis of user behavior, with user models modeled in cognitive architecture, as described by Curzon [8]. With this approach the researchers were able to capture potential erroneous interactions between devices and humans. The focus of the research by Curzon was in the modeling of human cognition following the principles of cognition. The formal models where verified using Theorem Provers, not model checkers. As a result, it is a more manual process than the automated method used by model checkers. Finally, the work by Curzon discussed an approach to the verification but did not implement the approach. Degani [9] discusses an approach to the formal modeling and analysis of the interface between the human and machine in the avionics domain to capture errors in the design of the interface. Once the user models were developed, a correspondence table was developed to understand how a user would respond to an event. Finally, the composed model was developed by generating the masked synchronous product of the user model with the autopilot model. This work was also a theoretical study not implemented using any tool. Bolton [10] surveyed formal verification based approaches that was applied to verify human automation interaction to identify errors in the design of the interface. But, all of these approaches where focused on applying formal methods to identify errors during design time, none of the approaches discussed how formal user models can be integrated as part of analyzing real time data, as well as, for security policy generation for cybersecurity. Our approach demonstrates how formally modeled cybersecurity behavior of users can be analyzed in real time to then generate security policies. Houser [11] discussed developing mental user model to capture what the user thinks the present status is of the system, the actual state of the system and then identifying if there are any vulnerabilities. Houser's model does not capture the actual behavior exhibited by the user, which we are modeling by reviewing safe and unsafe behavior exhibited by users based on research conducted by Egelman [12], which can be used to generate security policies. One similar work is that of Enoch, Simon Yusuf. "Dynamic cybersecurity modeling and analysis." (2018) [1]. Enoch uses network state input to generate an adaptable security model based upon constraints. Changes from the state of a dynamic network are used as input to generate modified graphical security models but does not take user behavioral input explicitly. The work is also directed at a network that can dynamically adjust network configurations (also called Active Defense) rather than assuming a Zero Trust environment. The models used to generate and evaluate attack scenarios focus upon possible attacker behavior (attack trees) rather than upon evaluating the risk of end user behaviors, choices, and device settings.

There are four additional works for generating policy that are more specifically applied to a Zero Trust environment. Chen, Baozhan, et al. [13] and Mandal, Sudakshina, et al. [14] both focus on dynamically generating access control decisions but neither explicitly includes recent user security behaviors, choices, or device settings as decision-making as input or as model features. Eidle, Dayna, et al. [15] adjusts access policy for 8 different trust levels dynamically via firewall Access Control Lists (ACLs) and Authentication Gateways. The model takes alert data from firewalls, authentication gateways, and other network devices as input but does not include recent user security behaviors that do not trigger alarms. Lastly, Dean, Erik, et al. [16] use automated device health attestation results as input for some of the user devices in their network during session authentication in a Zero Trust environment. This maps to the properties in Device Securement, and Updating and may even map to Passwords or ProActive Awareness. However, the work does not list the specific device parameter checks that are performed in the "Health Check". Another key difference is that the work does not use reachability analysis to apply different levels of security permissions for slightly higher risk but still acceptable device configurations.

3. Research methodology

Research methodology, describes the approach used in addressing the problem of generating user specific security policy following the Zero trust philosophy.

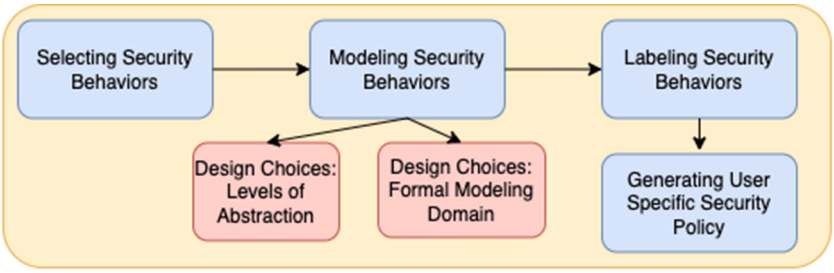

Fig. 1. Research methodology.

Our approach uses results from the study of human intentions, attitudes, norms, and resulting cybersecurity decision making behaviors. There are many studies showing that usable security is a hard problem. Users often ignore explicit warnings and misunderstand the likelihood of many risks and their resulting impacts [17]. Users also rationally reject warnings because they have experienced so many false alarms in their past experience [18,19]. Human and Computer interaction interfaces are hard to design and inherent cognitive weaknesses can negatively affect security decisions [3,20]. Some studies find that individual users behave differently when presented with security decisions based upon background, culture, and attitudes [2,5,21].

Because human behaviors are varied, difficult to predict, and open ended, we limit the security behaviors in our study to those with an existing scale that has been validated in previous studies. Egelman and Peer created a Security Behavior Intentions Scale (SeBIS) [12] to evaluate users' ability to adhere to computer security advice based on self-declared information. The scale focuses on four constructs which are: device securement, password generation, proactive awareness, and updating. Gratian et al. [2] substantiate the accuracy of SeBIS and broaden SeBIS to examine the correlations of personality traits and demographics with security intentions.

Our Formal Method guided User-Specific Policy Generation Framework (FMUSPGF) in Fig. 1 sets out the structure of this research as a whole, the figure shows the development process as a sequence of four stages.

The first step in FMUSPGF involves selecting security behaviors, this was achieved by conducting a survey of existing literature to identify what are the expected security behaviors. Once the security behaviors are selected, the second step involves modeling these behaviors, where we implemented a formal method-based approach. After the security behaviors are modeled, the next step involves labeling the security behaviors, this was accomplished by executing queries designed using formal specification. Finally, user specific security policies are generated guided by the labeling in the previous step.

Our main contribution is in the design of a formal method based framework to support the process of user specific policy generation for cybersecurity. In the process, we demonstrated how to identify and select the essential characteristics that define users security behavior. Then, we modeled the identified security behaviors as formal models to enable automated reasoning. Finally, we were able to detect weakness in users security behavior and then propose relevant policies.

4. Selecting security behaviors

In this section, we focus on selecting user's security-related behavior by taking into account the security decisions made by that user. It describes the scope of data collection, comparison, and selection in order to construct a reliable knowledge base that represents the requirements for developing the formal model, which we accomplished after reviewing a wide range of possibilities.

Previous studies have comprehensively investigated the correlations between individual differences in demographics, personality traits, decision-making styles, and risk-taking preferences and their influence on users' security-related behaviors in cyberspace. We chose to focus on decision-making because it has experimentally been proven to be a stronger predictor of security practices than demographics, personality traits, and risk-taking [2,12].

Table 1
Comparison between [5] and [2] psycho-metrics.

Metric	Study [5]	Study [2]
Consideration for Future Consequence (CFC)	✓	✘
Security Behavior Intentions Scale (SeBIS)	✓	✓
Domain Specific Risk Attitude (DoSpeRT)	✓	✓
International Personality Item Pool (IPIP)	✘	✓
General Decision Making Style (GDMS)	✓	✓
Barratt Impulsiveness Scale (BIS)	✓	✘
Need For Cognition (NFC)	✓	✘

The selection of predictors for this research is derived from the observation of two particular earlier studies, which were conducted by Egelman and Peer [5] and Gratian, Et al. [2], respectively. Both pieces of research aim to develop tailored security defenses, taking into account differences between individuals, in order to limit users' errors and their resulting consequences. The sample population of [2] included higher education participants from a large public university located in the USA, while Egelman and Peer surveyed a group of individuals from Amazon Mechanical Turk (MTurk) who are over the age of 18. They both made use of SeBIS to carry out their studies; however, Egelman and Peer's area of interest regarding security was investigating the correlations between SeBIS and decision-making psycho-metrics. Gratian et al. did an extended version of [5] such that they substantiate the accuracy of SeBIS, broaden the use of SeBIS to examine the correlations of personality traits and demographics with security intentions.

From Table 1 comparing Egelman et al. [5] and Gratian et al. [2] we identified that SeBIS, DoSpeRT, and GDMS are used for the same purpose of identifying the relation between security intentions, risk-taking, and decision-making. The process of determining how to respond to a particular situation requires risk evaluation, considering future consequences, and exploring possible alternatives. The natural structure of humans psychology, the limitation of humans' information processing capacity, and their almost absolute reliance on previous experiences stand in the way of making the right choices [3]. Failing to determine what is the right thing to do and vice versa is a critical matter in cybersecurity. If some individuals decided to leave their device unattended in public, set up a password that is easy to guess or hack, rush to download an anonymous email attachment, or underestimate the importance of software updates, they are more at risk to be victimized by a crafty individual.

We consider modeling security behaviors with each user's behavior as a sequence of different connected decisions. Each decision correlates with particular SeBIS construct: device securement, password generation, proactive awareness, or updating. Using the findings that we collected from [2,5]. We chose to use Egelman and Peer's results in this work to develop the formalized users' security-related behaviors according to specific security decisions as the behaviors were identified as realistic and user security behavior.

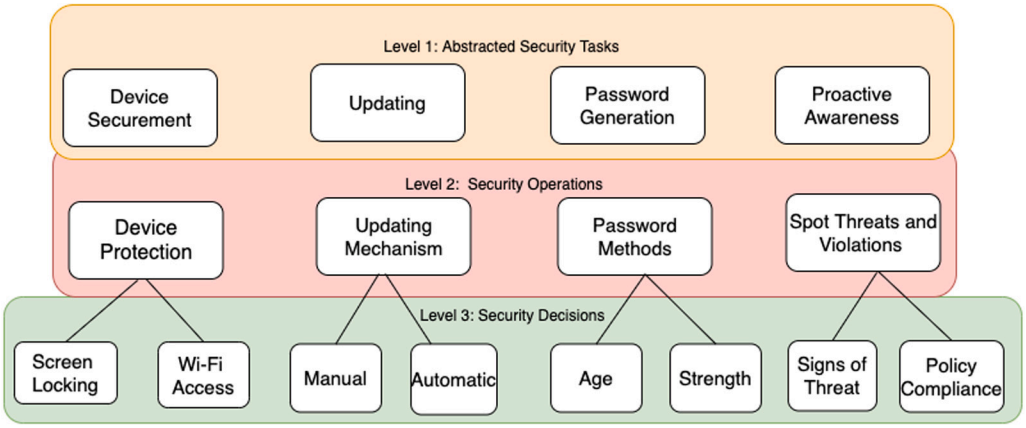

Fig. 2. Cybersecurity user behavior knowledge representation architecture.

5. Modeling user security behavior

After reviewing studies on how the decision-making process related to security behavior, the appropriate knowledge base was built, and the most relevant user behaviors were selected to develop the formal model. The content of this section focuses on the design choices that were made pertaining to the architecture with levels of abstraction, and the modeling paradigm that was selected.

5.1. Design choice: Levels of abstraction

It is challenging to model human–machine interactions, because of the complexity of human behavior and the broad set of knowledge requirements. Although, we chose a specific knowledge base, it is still challenging to model and examine every aspect of a users security-related behavior that is relevant to device securement, password generation, proactive awareness, and updating. To model users behavior across the different aspects of SeBIS, we applied principles of decomposition inherited from software/systems engineering to architect the structure of the model. the architecture of the model is as shown in Fig. 2.

The architecture modeling the representation of knowledge about the security behavior includes different levels of abstractions, in order to ease the debugging, increase readability/maintainability and lessen the complexity. We decomposed the structure into different sorts of security services on multiple layers, starting from **(1)** as the highest level of abstraction and ending with **(3)** as the lowest level of abstraction. By doing so, we eliminate a fair bit of confusion around which security aspect we employed for a specific SeBIS dimension.

Here, we illustrate each level of abstraction in detail.

- **Layer (1)** is the most abstract of the four security service check layers. It assimilates the SeBIS concepts: device securement, password generation, proactive awareness, and updating that are specified by Egelman and Peer. These correspond to the highest level of representation of the security task that needs to be executed.
- **Layer (2)** breaks down SeBIS concepts according to key parameters within each category as identified at Level 1 for example, device protection, update mechanism, password methods, and attention to threats and violations. Then, it is further decomposed into possible additional sub-services based on the user decisions on the actions they want to take.
- **Layer (3)** has a sub-tree that descends from the specifics of Layer 2 for example screen-locking feature for device protection and is enabled by what the user chooses.

5.2. Design choices: Modeling paradigm selection

Although, there are several approaches that can be used to model user behavior such as, Markov chains, architectural representations, we selected formal method based approach. We decided that the most appropriate method of representing user behavior is through the use of a Finite-State Automata (FSA) [22] because it allows us to perform automated analysis early in the design phase, which would empower us to reason about the logical representations of the user's behavior to evaluate alternative design options in case there were profound implications. It also allows graphical representation to visualize user's behavior easily. It enables the use of well-defined tools too.

In order to choose the correct platform for the purpose of designing and checking satisfiability with formal verification the formal model of user's behavior, several formalisms such as NuXMV [23], Uppaal [24], PVS [25], and Z3 [26] were considered carefully. We chose Uppaal [27,28][24], due to its ability to model timing aspects that are critical for cybersecurity, as well as, its ability to generate and visualize counterexamples. Uppaal represents models as timed automata, and Uppaal formalism supports model checking networked timed automata using temporal logics. This modeling paradigm allows the execution of requirements as temporal logic queries to check the satisfaction of relevant safety properties exhaustively. We next describe the timed automata formalism used by Uppaal.

5.2.1. Mathematical representation within timed automata.

Uppaal uses timed automata [29], a subset of hybrid automata, as a modeling formalism. One of the essential requirements in the design of human–machine interactions is to be able to model the time associated with the execution of operations or rules. A timed automata is a finite automata extended with a finite set of real-valued clocks. Clock or other relevant variable values used in guards on the transitions within the automata. Based on the results of the guard evaluation, a transition may be enabled or disabled. Variables can be reset and implemented as invariants at a state. Modeling timed systems using a timed-automata approach is symbolic rather than explicit. It allows for the consideration of a finite subset of the infinite state space on-demand (i.e., using an equivalence relation that depends on the safety property and the timed automata), which is referred to as the region automata. There also exists a variety of tools to input and analyze timed automata and extensions, including the model checker Uppaal and Kronos [30].

- **Timed Automata (TA)**
  A timed automata is a tuple (L, l_0, C, A, E, I), where: L is a set of locations; $l_0 \in L$ is the initial location; C is the set of clocks; A is a set of actions, co-actions, and unobservable internal actions; $E \subseteq L \times A \times B(C) \times 2^C \times L$ is a set of edges between locations with an

action, a guard and a set of clocks to be reset; and $I : L \rightarrow B(C)$ assigns invariants to locations.
We define a clock valuation as a function $u : C \rightarrow \mathbb{R}_{\geq 0}$ from the set of clocks to the non-negative reals. Let $\mathbb{R}^C$ be the set of all clock valuations. Let $u_0(x) = 0$ for all $x \in C$. If we consider guards and invariants as the sets of clock valuations (with a slight relaxation of formalism), we can say $u \in I(l)$ means u satisfies $I(l)$.

- **Timed Automata Semantics**
Let (L, l_0, C, A, E, I) be a timed automata TA. The semantics of the TA is defined as a labeled transition system $\langle S, s_0, \rightarrow \rangle$, where $S \subseteq L \times \mathbb{R}^C$ is the set of states, $s_0 = (l_0, u_0)$ is the initial state, and $\rightarrow \subseteq S \times \{\mathbb{R}_{\geq 0} \cup A\} \times S$ is the transition relation such that:

  1. $(l, u) \rightarrow d\ (l, u + d)$ if $\forall\ d' : 0 \leq d' \leq d \Rightarrow u + d' \in I(l)$
  2. $(l, u) \rightarrow a\ (l', u')$ if $\exists\ e = (l, a, g, r, l') \in E$ such that $u \in g$, $u = [r \mapsto 0]\ u$ and $u' \in I(l)$

  where for $d \in \mathbb{R}_{\geq 0}$, $u + d$ maps each clock x in C to the value $u(s) + d$, and $[r \mapsto 0]u$ denotes the clock valuation which maps each clock in r to 0 and agrees with u over $C \setminus r$.
  Note that a guard g of a TA is a simple condition on the clocks that enable the transition (or, edge e) from one location to another; the enabled transition is not taken unless the corresponding action a occurs. Similarly, the set of reset clocks r for the edge e specifies the clocks whose values are set to zero when the transition on edge executes. Thus, a timed automata is a finite directed graph annotated with resets of and conditions over, non-negative real-valued clocks. Timed automata can then be composed into a network of timed automata over a common set of clocks and actions, consisting of n timed automata $TA_i = (L_i, l_{i0}, C, A, E_i, I_i)$, $1 \leq i \leq n$. This enables us to check reachability, safety, and liveness properties, which are expressed in temporal logic expressions, over this network of timed automata. An execution of the TA, denoted by $exec(TA)$ is the sequence of consecutive transitions, while the set of execution traces of the TA is denoted by $traces(TA)$.

5.3. Query language for verification

The process of verification in Uppaal operates with a specific type of query language that is used to specify a set of properties that need to be examined. The query language is a subset of Computation Tree Logic (CTL) called Timed CTL (TCTL) [31]. The syntax of the Timed Computation Tree Logic is expressed as follows:
$\Phi ::= a \mid g \mid \Phi_1 \wedge \Phi_2 \mid \neg\ \Phi \mid E\ (\Phi_1 \cup^J \Phi_2) \mid A\ (\Phi_1 \cup^J \Phi_2)$, where

- ϕ is the property or proposition
- a is an atomic action.
- g is a clock constraint.
- **E** means "for some paths".
- **A** means "for all paths".
- J is an interval whose bounds are natural number.
- state $s_i \vDash \mathrm{E}(\Phi_1 \cup^J \Phi_2)$ "for some path" s_i, s_{i+1}...
  $(\exists\ \mathrm{k} \geq i, \mathrm{k\text{-}i} \in \mathrm{J})\ ((s_k \vDash \mathrm{q})\ (\forall\ \mathrm{j}, \mathrm{i} \leq \mathrm{j} < \mathrm{k})(s_j \vDash \mathrm{p}))$
- state $s_i \vDash \mathrm{A}\ (\Phi_1 \cup^J \Phi_2)$ "for every path" s_i, s_{i+1}...
  $(\exists\ \mathrm{k} \geq \mathrm{i}, \mathrm{k\text{-}i} \in \mathrm{J})\ ((s_k \vDash \mathrm{q})\ (\forall\ \mathrm{j}, \mathrm{i} \leq \mathrm{j} < \mathrm{k})(s_j \vDash \mathrm{p}))$

TCTL is similar to CTL in having temporal connectives that are expressed as pairs of symbols. Such that, the first element of the pair represents one of the path quantifiers that is either **A** or **E** whereas the second element of the pair is one of the state quantifiers that is one of the following:

- **G** means "all states in a path".
- **F** means "some state in a path".

The different combinations of path formulae and state formulae accepted by Uppaal are: **AG** invariantly `A[]`, **EG** potentially always `E[]`, **AF** eventually `A<>`, **EF** possibly `E<>`.

6. Experiment

To conduct the experiments the first step involved the experimental setup, which involved the modeling of the selected security behaviors in timed automata 6.1. Once the models were developed, the next step involved conducting the experiments (6.2) to label good and bad security behaviors.

6.1. Experimental setup: Modeling of selected security behaviors

The design of our models were guided by the capability to formally check the existence of good and bad user security behaviors (as measured by instruments taken from academic literature) in TCTL. Sections 6.1.1 to 6.1.4 will present a further illustration of why and how we chose the most critical matters within different security settings that are parts of device securement, password generation, proactive awareness, and updating. Using behavioral measurement scales in existing literature, we set up a selection of examples separating good decisions from poor ones.

In this work, we perform the data collection manually to show the concept, but the details of instrumentation (or data collection implementation) will vary based upon organizations and their *Zero Trust* solutions, devices, policies, and procedures. There are also different proposed solutions for implementing *Zero Trust* policy decisions, and any data collection or measurement of user security behavior will need to interact with the specific *Zero Trust* solution. *Zero Trust* solutions notionally include a policy enforcement point and a policy engine. Typically data needed for a security decision is collected by the policy enforcement point and sent to the policy engine for review. Organizations could add data collection agents to include end-user security behaviors according to their specific implementations and policies. Our method allows organizations that embrace a *Zero Trust* philosophy to generate context specific security policies that can be automatically verified for correctness and completion. Future work will be required to add user specific security behavioral data to existing data that is fed to the policy engine. *Zero Trust* environments are typically envisioned to have a policy engine analyzing network, application, device, and data access policies and determining a decision in an access context. Some examples include using network or host security data including MAC addresses identifiers, Internet Protocol (IP) addresses identifiers, security access tokens, session tokens, process event data, packet inspection data, running process data, threat intelligence data, device specific data such as installed application inventories, configuration settings, security relevant settings, and much more.

6.1.1. Device securement

- **Device Protection**
  We chose device protection as the first criterion for device securement. It means whether or not users protect their devices by passwords, PIN codes, fingerprints, or patterns. Even though locking devices of all kinds is a simple security task, it is sometimes undervalued by end-users. In the Pew Research Center survey, conducted in 2017, 28% of American mobile phone users reported that they do not use PIN codes or any other security feature to access their smartphones [32]. This matter of security is a lot more critical in the work environment. Because for instance, if some employees are working outside the workplace using portable devices (e.g., laptops, tablets, or smartphones) as their primary work computer, they could leave the device unprotected in some places as in a hotel room or a car. By doing so, if their device is stolen, it is already unlocked, and the company's data would be in the hand of an unauthorized individual. Most organizations require this control for devices holding their data, but it may be hard to monitor and enforce across all ecosystems and devices.

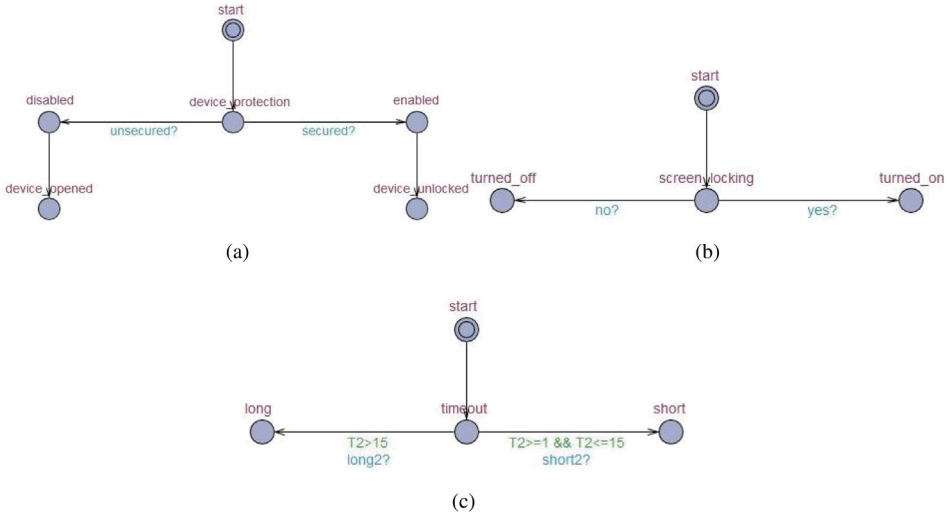

Fig. 3. Device securement state-transition graphs.

Table 2
Device securement properties.

No.	Knowledge base property
1	E<> (device_protection.enabled)
2	E<> (device_protection.disabled)
3	E<> (screen_locking.turned_on)
4	E<> (screen_locking.turned_off)
5	E<> (timeout.short)
6	E<> (timeout.long)

- **Screen-Locking**
  Password-protected screen saver feature is about setting up the device to lock off automatically after some time of inactivity. Some end-users find it a little troublesome thing to do when they have to consistently login again every moment the timeout is exceeded. Others have a low perception of the threat; they believe nothing would go wrong since they are around their portable devices almost all the time, especially smartphones [33]. Some others do set a password-protected screen saver, but they adjust the default timeout time (i.e., often 15 min) to a much longer time [34]. We examine this side of device securement because leaving the device without a password-protected screen saver would allow malicious individuals (e.g., insider threats) to access data or perform some tasks they are not entitled to see or do [35]. Insider threat is one of the most difficult security issues; malicious insiders can put the organization at a greater risk than outsiders because they are more familiar with security infrastructure, practices, and vulnerabilities. They can more easily avoid detection and remain hidden for a long period of time.
  In Table 2, we can see how we translate the criteria mentioned above into TCTL formulate whereas Fig. 3 depicts device protection, screen-locking, and screen-locking timeout state-transition graphs, respectively.

6.1.2. *Password generation*

- **Password Age**
  Passwords are a perennial problem in cybersecurity. Much advice is given, and policies are enforced, but still the problem of weak passwords is constantly growing. Setting a maximum password age is one of the traditional techniques for maintaining proper password hygiene. It requires users to change their passwords in a periodic manner, typically between 30 and 90 days [36]. Some might argue that scheduled changes make the passwords harder for users to remember, and thus more likely to be stored insecurely. Users are adjusting and developing coping mechanisms to overcome this burden by making some alterations to their current passwords. We included this rule to our set of specifications to show an example of the criteria that can be included in the knowledge base.
- **Password Length**
  People want to protect their personal information, but they are not willing to pay a little more effort for it. The biggest example of this is that despite the continuous security warnings that alert users about the use of weak passwords and the serious consequences that result from such behavior, they still use short and easy-to-remember passwords. According to the Psychology of the Password Report, 47% of people who participate in the study declared that they prefer to choose easy passwords because they are afraid of forgetting them [37]. Martin et al. (2012) illustrate that guessing a password through a brute-force attack is most likely unsuccessful against long and complicated passwords that contain capital and small letters, digits, and symbols. They mention that cracking a complicated password with lengths of 4, 8, or 16 characters would take approximately 81 s, 210 years, and 1.4 quintillion years, respectively [38]. According to Maddox and Moschetto (2019), when users assign a password for some account, they should maintain an adequate length, avoid well-known character substitutions, and use a variety of letters, numbers, and special characters [39]. We thought that the length of a password is necessary to look at to identify those whose accounts are vulnerable to password attacks.
- **Password Re-usability**
  With so many accounts to handle and keep track of, it can be tempting for users to use one password across them all and bring themselves some peace of mind; users might live to regret it. Re-using the same password for multiple sites carries more risk than writing down separate passwords on a piece of paper. If individuals decided to assign one password for their different accounts, they would allow attackers to compromise other accounts that use the same password [40]. The struggle with password protection is that the majority of end-users acknowledge the risk and the consequences of password re-use; yet, 35% of them ignore this knowledge in favor of remembering their easy and familiar passwords [37]. We chose to cover this aspect of password generation because, in the worst-case scenario, the resulting risk is not limited to the end-users but extends to the workplaces and coworkers if they re-use the work password for some other personal account.

Table 3
Password generation properties.

No.	Knowledge base property
1	E<> (password_age.non_expired)
2	E<> (password_age.expired)
3	E<> (password_length.long)
4	E<> (password_length.short)
5	E<> (password_reusability.unused)
6	E<> (password_reusability.used)

Table 3 lists the equivalent expressions of password age, length, and re-usability in Uppaal specification language.

6.1.3. Proactive awareness

- **Spot Signs of Threat**
  There is no doubt that the interest in cybersecurity is growing and expanding, making people more educated and cautious about online information sharing, fake e-commerce sites, scams, and security threats. There is always a "but" in this imperfect world because there is a significant number of people who lack digital security awareness and education, which in turn affect their ability to recognize threats even if there were apparent signs. End-users would not be able to protect themselves from identity theft if they were unable to spot a sign of spyware on their device, recognize a social engineering attempt, identify phishing or spoofing email, or any other elusive activities. According to Verizon's Data Breach Investigations Report (DBIR) (2019), the phishing attack was one of the leading causes of data breaches. It was a contributing factor in 32% of confirmed data exposures, and 78% of cyber-espionage incidents [41]. On account of this, we stressed the importance of spotting early warning signs by investigating the users' ability to recognize and avoid phishing scams before it is too late.
- **Report Threat**
  In "If You See Something, Say Something®" national campaign, which raises public awareness of the indicators of terrorism-related crime, we are encouraged to report the authorities if something does not seem quite right to keep ourselves and our communities safe [42]. It is exactly the case in organizations' environments; reporting possible security incidents can save valuable crucial time in the early stages of breach detection [43]. If employees know about cyberattack types and how they look and occur, they are more likely to notice unauthorized changes that have taken place on their systems. They probably would be more confident and willing to reach out to the IT department. Lack of cybersecurity awareness, training, and vigilance, and miscommunication between employees and the IT team would make the former hesitate and question themselves whether they have caught something real or not. The idea of including this area of interest to our research is to model whether users are truly not educated enough, or they are just too reckless to report such urgent matter.
- **Policy Compliance**
  Identifying, determining, and handling risks to the confidentiality, integrity, and availability of an organizations' assets is a top-notch priority. Security policies and procedures are part of the hierarchy of any organizations' management control to maintain the security of sensitive data, the most critical asset, from the complex and ever-evolving threat landscape. One of the biggest concerns for any organization is how to protect data from its employees [44]. Security policies and guidelines are put in place to draw a line for employees between what is acceptable and unacceptable to do when they interact with the information system. The problem lies in the employees' non-compliance [45]. According to Mutlaq et al. (2016), non-compliant behavior can be categorized as: first, deviant behavior that is driven by intentional or planned desire to harm the organization entity. Second, negligent behavior that is intended to go against security policy but with no malicious intent to cause harm. Third, ignorant behavior that steams from unawareness due to a lack of cybersecurity knowledge and training. After identifying these different interpretations of security policy violations, we modeled end-users' adherence to security policy, as it aids the process of identifying the right policy actions to impose against a specific user.
  Table 4 shows the linear-time properties that have been generated to investigate the knowledge base aspect of proactive awareness.

Table 4
Proactive awareness properties.

No.	Knowledge base property
1	E<> (spot_signs.yes)
2	E<> (spot_signs.no)
3	E<> (report_threat.yes)
4	E<> (report_threat.no)
5	E<> (security_policy.complied)
6	E<> (security_policy.violated)

6.1.4. Updating

- **Updating Mechanism**
  Security-related software updates are one of the single most important security protection tools that end-users should pay more attention to and ensure that they are being installed on a regular periodic schedule. There are several options for the operating system and application updates that end-users can configure, such as how updates are downloaded and installed (i.e., automatically or manually) on their devices. Most modern software systems are set to download and install security and other essential updates automatically without the need for humans to make decisions. In contrast, the manual update allows users to gain better control over their devices by choosing which update to install and when. There is no certain position where we can say that one mechanism is better and safer than the other because it all depends on users' behaviors regarding this matter. It may come to mind that the automatic mechanism appears as a more responsible selection. Still, the manual update is responsible as well in case end-users ensure that their systems are up-to-date as soon as a new update is available. We chose including users' preferences of the updating mechanism to observe the different behaviors.
- **Time to Update**
  Even though some software updates and patches are released to address security bugs that have been discovered in previously installed software, some end-users avoid or delay the installation because they consider that these incremental updates are just useless technical additions [46]. Herein lies the risk for those users who download and install updates manually after they were available a while ago. We discern from this that delaying the installations of the latest updates and patches creates windows of opportunities for malicious individuals to exploit open security vulnerabilities on the users' devices [47]. Thus we investigated further in characterizing and modeling the negligent behavior of users against timely updates.
- **Time To Reboot**
  Software developers have unremittingly endeavored to improve security by excluding the user role from the software update cycle. They found that user intervention remains a must because some updates require a device reboot to allow changes to take effect [48]. There are operating systems, such as Microsoft Windows, developed to alert users if a reboot is required after updates are installed. In this case, the system shows up a notification pop-up that a reboot will occur within a while (i.e., usually 10 min).

Table 5
Updating properties.

No.	Knowledge base property
1	E<> (updating_mechanism.automatically)
2	E<> (updating_mechanism.manually)
3	E<> (time_to_update.short)
4	E<> (time_to_update.long)
5	E<> (time_to_reboot.right_away)
6	E<> (time_to_reboot.after_awhile)

Users are given the option either to reboot the device immediately or to postpone for an additional specific time. If users chose to postpone, the warning dialog would appear again with the same options [48]. Users delay the rebooting task because they might have some pressing matters that keep them from rebooting for a couple of hours. The decision to postpone rebooting more than once could negatively impact the security of the device because, for the computer system, the update installation is not completed. We observed this aspect of updating to draw the attention of negligent and unaware users to the importance of immediate reboot if required.

In Table 5, we present the formal specifications of updating that are expressed in terms of timed temporal logic TCTL.

6.2. Experiments: Classifying security behavior

In this section, we model several test cases with different security behaviors, as Finite-State Automata (FSA). For each separate test case, we generate a set of user-specific linear-time properties. We then classify the behavior as good or bad based on the results of the reachability analysis. Once the behavior is classified we generate the relevant security permissions such as, Strict, Moderate and Least restrictive.

6.2.1. Automated analysis

We seek to analyze users' behaviors in order to make a careful analysis and draw out their poor security decisions that have a significant impact on the security system. To ultimately achieve our goal, we designed six test cases that cover as much as possible of different security behaviors exhibited by users in real-world scenarios that revolve around device securement, password generation, proactive awareness, and updating. For each test case, we represent the user's behavior as a state-transition graph using the Uppaal tool,[1] manually generate user-specific linear-time properties, apply reachability analysis, identify good and bad security behaviors and generate user-specific policy. In this section, we demonstrate one example of these test cases as an illustration of our formal method-based approach.

6.2.2. Representing test cases

In order to have reliable test cases, we had to make several assumptions and predictions of how some users might behave and make security-related decisions. In one test case, we created a scenario with a user named Tom, who works as a Data Entry Specialist at a network marketing company. Tom is assigned a laptop to perform duties directly related to the business of the company and to allow him to work remotely and outside of regular working hours. On this basis, the company requires him to be responsible and take reasonable precautions to protect and maintain the laptop and its content. For this research, we are focusing on capturing Tom's security behavior rather than his system role. Fig. 4 represents the state-transition graph of Tom's security-related behavior. All the other models can be found at https://github.com/sbhattacharyya/USPGZeroTrust.

[1] Integrated tool environment for modeling, validation, and verification of Finite-State Automata (FSA) [24].

6.2.3. Generating user-specific security properties

In the previous section, we have successfully managed to model the user's security-related behavior through the modeling graphs of Finite-State Automata (FSA). We made Tom exhibit different good and poor security behaviors regarding device securement, password generation, proactive awareness, and updating. In order to automatically analyze Tom's behavior, we needed to create linear-time properties manually, which are generated specifically for Tom. In Fig. 5, we generated the properties we needed to check based on properties identified previously specified in Section 4. We are generating these properties in order to distinguish the aspects where Tom has failed to apply proper security practices.

6.2.4. Reachability analysis

Reachability analysis with *model checking* is a verification procedure for models that are designed based on the state-transition concept. According to Kong et al. (2015), reachability analysis is a technique used in a state-transition system in order to find out the type and number of states which can be accessed through a particular system model [49]. Reachability analysis allows formal analysis for validation, verification, and checking performance metrics, explained as follows:

- **Validation:** A simulation of the process is shown where the model should reflect what it indented to represent.
- **Verification:** A checking process is carried out to ensure that the specifications meet the model that is built.
- **Performance:** A set of predictions about the key performance indicators is made.

Among these procedures, we are checking the satisfiability of our specifications (i.e., properties) in any, some, or all states of user's behavior model. According to Eleftherakis et al. (2001), "A model checker takes a model and a property as inputs and outputs either a claim that the property is true or a counterexample falsifying the property." [50]. In this research, we favored reachability analysis over other methods such as graph matching approach because it provides the opportunity to intensely and automatically check all possible paths to check the satisfaction of security properties while being computationally less expensive than graph matching.

For the set of combined linear-time security properties shown in Fig. 5, we applied reachability analysis using Uppaal to see which properties were satisfied and which were not. The results of this procedure, allow us to examine Tom's behavior and identify his security weaknesses. We identified through two examples of reachability analysis results in Uppaal, where Tom enabled manual updating and did not install the new updates within the first day of their release.

Uppaal executes reachability analysis and checks whether a state is reachable either with Breadth-First Search (BFS) or Depth-First Search (DFS) algorithms for traversing graphs, respectively. We chose Breadth-First Search (BFS) to check the satisfaction of our reachability properties (i.e., security properties) in users' state-transition graphs because it allows traversing a graph with ease.

6.3. Policy generation

After the execution of the experiments we were able to check the satisfiability of the properties and then associate labels with user specific security behaviors. According to those labels we were able to generate and define a set of policies to be imposed on specific users based on the satisfaction of poor or good security behavior found from model checking. The type of policies to be enforced depends on the users' decisions that represent their adherence to the rules set, which we established for each security aspect highlighted in this research: Device Securement (DS), Password Generation (PG), Proactive Awareness (PA), and Updating (U). We assign each security aspect one of three standardized policy types that we drafted as follows:

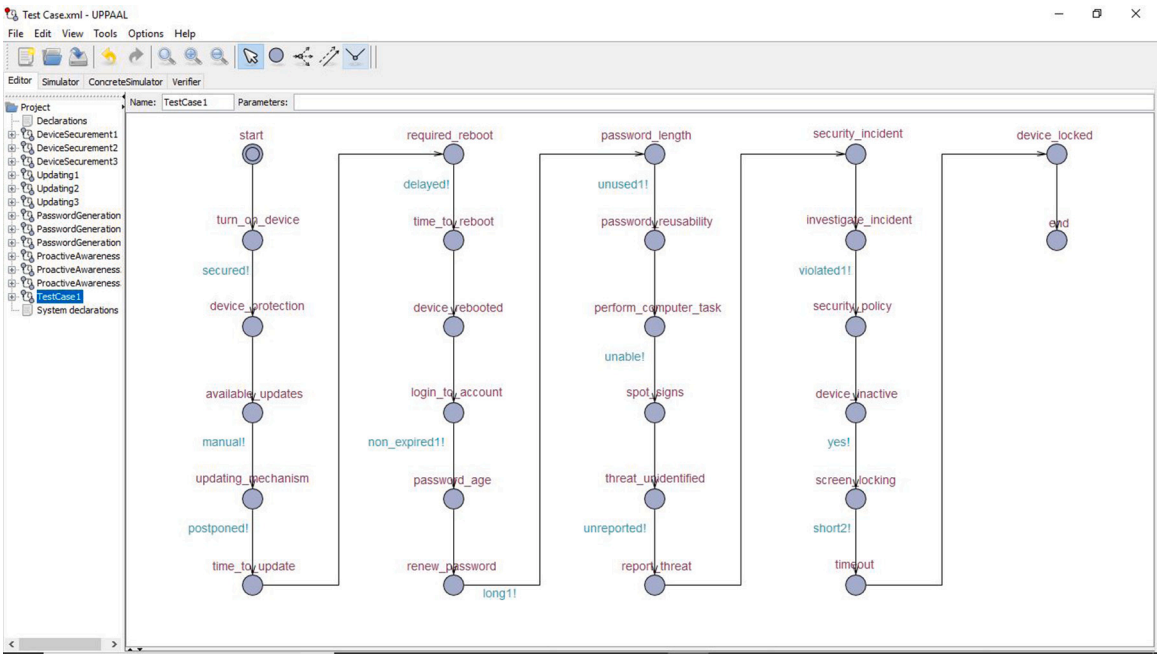

Fig. 4. Tom's state-transition graph.

No.	Security Linear-Time Property
1	E<> (Tom.device_protection and device_protection.enabled)
2	E<> (Tom.device_protection and device_protection.disabled)
3	E<> (Tom.screen_locking and screen_locking.turned_on)
4	E<> (Tom.screen_locking and screen_locking.turned_off)
5	E<> (Tom.timeout and timeout.short)
6	E<> (Tom.timeout and timeout.long)
7	E<> (Tom.updating_mechanism and updating_mechanism.manually)
8	E<> (Tom.updating_mechanism and updating_mechanism.automatically)
9	E<> (Tom.time_to_update and time_to_update.long)
10	E<> (Tom.time_to_update and time_to_update.short)
11	E<> (Tom.time_to_reboot and time_to_reboot.after_awhile)
12	E<> (Tom.time_to_reboot and time_to_reboot.right_away)
13	E<> (Tom.password_age and password_age.non_expired)
14	E<> (Tom.password_age and password_age.expired)
15	E<> (Tom.password_length and password_length.long)
16	E<> (Tom.password_length and password_length.short)
17	E<> (Tom.password_reusability and password_reusability.unused)
18	E<> (Tom.password_reusability and password_reusability.used)
19	E<> (Tom.spot_signs and spot_signs.no)
20	E<> (Tom.spot_signs and spot_signs.yes)
21	E<> (Tom.report_threat and report_threat.no)
22	E<> (Tom.report_threat and report_threat.yes)
23	E<> (Tom.security_policy and security_policy.violated)
24	E<> (Tom.security_policy and security_policy.complied)

Fig. 5. Security properties verified.

1. **Strict Policy** (s): Applied when the user is exhibiting severe disregard for the appropriate security measures.
2. **Moderate Policy** (m): Applied when the user is exhibiting negligence in following the appropriate security measures.
3. **Least Restrictive Policy** (r): Applied when the user is exhibiting good security behavior with full respect to the appropriate security measures.

The assignment of a policy to a user is as described in the policy generation Algorithm 1. The input to the algorithm are: 1. model of all the known security behaviors, as TA, 2. representation of user security behaviors ($U_i Sb_k$) as a trace in TA, where U_i is user i and sb_k is the security behavior k exhibited by user i, formulation of security properties in TCTL $SProp_j$, it is the security property that is being verified and the satisfaction output, for a user exhibiting all the behaviors stored in a list Sat_Out_i, it stores the security behavior and the behavior label satisfied by the user i. The behavior label can be good (G_k), bad (B_k) or a mixture of good and bad (GB_k). Based on the satisfaction outcome the policy ($P_{type_{ik}}$) is selected to be least restrictive, moderate or strict policy for user i for security behavior k. In the algorithm we consider the number of users to be n, the number of security behaviors to be p and number of security properties to be m.

In our example, a policy is based on four security behaviors, as discussed in Section 4. The four security behaviors are device securement, password generation, proactive awareness, and updating. The representation of a general security policy based on the described algorithm is as given below:

$Sb = \{DS, PB, PA, UP\}$ if $Sat\_Out_i = \{DS(G), PG(B), PA(G/B), UP(G)\}$ $Policy_i = \{r_i, s_i, m_i, r_i\}$

As per the formal description of the general policy, we can determine the appropriate policy that is suitable for the six test cases tested are as follows:

Algorithm 1 **INPUT**: Results of Model checking performed on Timed Automata representation of User Security Behavior (TA, $\{\{U_1\ Sb_1, ..., U_1\ Sb_k ... U_1\ Sbp\}, \{U_i\ Sb_1, ..., U_i\ Sb_k ... U_i\ Sbp\}, \{U_n\ Sb_1, ..., U_n\ Sb_k\ U_n\ Sbp\}\}$ $\{SProp_1, ..., SProp_j, ... SProp_m\}$, $\{Sat\_Out_1 ... Sat\_Out_i ... Sat\_Out_n\}\}$

OUTPUT: User Specific Security Policy $((User_1, (Sb_1, P_{type_{11}}), (Sb_2, P_{type_{12}}), (Sb_p, P_{type_{1p}}), (User_i, (Sb_1, P_{type_{i1}}), (Sb_2, P_{type_{i2}}), (Sb_p, P_{type_{ip}}), (User_n, (Sb_1, P_{type_{n1}}), (Sb_2, P_{type_{n2}}), (Sb_p, P_{type_{np}}))$

for all $i \in \{1, \ldots, n\}$ **do**
  for all $k \in \{1, \ldots, p\}$ **do**
    SELECT $U_i Sb_k$; select security behavior trace for user i
    for all $j \in \{1, \ldots, m\}$ **do**
      SELECT $SProp_j$; select security property to verify
      PERFORM MODEL CHECKING ON $(U_i Sb_k)$ WITH TCTL QUERY $(SProp_j)$
      STORE SATISFACTION OUTPUT FOR $User_i$ in a list $Sat\_Out_i = \{(Sb_k, Beh\_Label_k)\}$; where Beh_Label_k *is* $G_k(Good)$ *or* B_k (Bad) *or* GB_k *equal number of good and bad security behavior*
    end for
  end for
end for
for all $i \in \{1, \ldots, n\}$ **do**
  for all $k \in \{1, \ldots, p\}$ **do**
    From Sat_Out_i Extract behavior Sb_k and Beh_Label_k
    if $\{Sb_k : Beh\_Label_k == G_k\}$ **then**
      $P_{type_{ik}} = r_i$; Least restrictive policy
    else if $\{Sb_k : Beh\_Label_i == B_k\}$ **then**
      $P_{type_{ik}} = s_i$; Strict security policy
    else
      $P_{type_{ik}} = m_i$; Moderate security policy
    end if
    STORE policy $Policy_i = \{P_{type_{ik}}\}$
  end for
  PRINT $Policy_i = \{P_{type_{i1}}, ..., P_{type_{ik}}, ..., P_{type_{ip}}\}$
end for

1. $Policy_{Tom}$ = {DS(G), PG(G), PA(B), U(B)} = $\langle r_{Tom}, r_{Tom}, s_{Tom}, m_{Tom} \rangle$
2. $Policy_{Sara}$ = {DS(B), PG(B), PA(G/B), U(B)} = $\langle s_{Sara}, s_{Sara}, m_{Sara}, s_{Sara} \rangle$
3. $Policy_{Zoe}$ = {DS(G), PG(G/B), PA(NA), U(NA)} = $\langle r_{zoe}, m_{zoe}, m_{zoe}, m_{zoe} \rangle$
4. $Policy_{Bella}$ = {DS(B), PG(B), PA(G), U(G)} = $\langle s_{Bella}, s_{Bella}, r_{Bella}, r_{Bella} \rangle$
5. $Policy_{John}$ = {DS(G), PG(G), PA(G), U(G)} = $\langle r_{John}, r_{John}, r_{John}, r_{John} \rangle$
6. $Policy_{Zac}$ = {DS(G), PG(NA), PA(B), U(NA)} = $\langle r_{Zac}, m_{Zac}, s_{Zac}, m_{Zac} \rangle$

So the policy can be read as follows: For Tom the policy engine should implement least restrictive policy (r) wherever, device securement and password generation are required, as he has shown good behavior in implementing these security behaviors. Whereas, a strict policy (s) needs to be implemented for proactive awareness, such as by sending more alerts or warnings as, Tom has shown weak behavior in spotting security threats. Finally, moderate policy (m) should be implemented in regards to updates, such as when to send alerts or when to implement specific actions that need to be taken if updates are not installed within a timeline.

7. Results

Using our FMUSPG framework we were able to select relevant user security behaviors from existing literature. We were then able to decompose the security behaviors exhibited by users at different levels of abstraction to represent the knowledge about the potential behaviors. This abstracted representation of security behaviors allowed us to create the required knowledge base as finite state automata and it also allows for scalability, as while checking the satisfiability you are checking one level at a time instead of including all the levels at once.

Once modeled, it enabled automated reasoning to check the satisfiability of good or bad security behaviors exhibited by users. As a result, using our framework user specific security related policies can be generated. As the policy generation was algorithmic the policy generation can be an automated process for zero trust environment to generate policies with changing user behavior.

We verified 90 properties for six test cases that were generated. The properties were executed on a 64 bit Mac, each property proved within 6–18 s. This higher level reasoning can lead to the generation of parameters to monitor for individual users to capture the user specific behaviors.

8. Conclusion

We were able to provide a solution that supports the concept of *Zero Trust* by eliminating the trust, that all users act responsibly. Most importantly, we achieved success in answering the research question "How to automatically identify users security practices and then generate security policy after observing and analyzing security behaviors, especially security-related decisions, exhibited by end-users in an environment with *Zero Trust* assumptions?" In our approach, Finite-State Automata supported modeling user security behavior. It allowed showing how a user could transition from safe to unsafe state based on making some specific decisions. TCTL language was used to generate linear-time properties, and thus, with reachability analysis we could check the satisfaction of the security behavior. After observing security behavior and analyzing security behavior, the appropriate policy was assigned to address security gaps caused by specific user.

Our approach demonstrated if we can categorize the behavior of users and capture relevant information regarding the behavior, it enables automated reasoning to then identify weaknesses in a users behavior to generate specific policies. Future work involves developing surveys to evaluate if there are more selection predictors. Another extension is to apply the method to accessing network based services, based on the analysis discussed in this research.

CRediT authorship contribution statement

Arwa AlQadheeb: Acquisition of data, Analysis and/or interpretation of data, Writing – original draft. **Siddhartha Bhattacharyya:** Conception and design of study, Analysis and/or interpretation of data, Writing – original draft, Revising the manuscript critically for important intellectual content. **Samuel Perl:** Conception and design of study, Writing – original draft, Revising the manuscript critically for important intellectual content.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

All authors approved the version of the manuscript to be published.

References

[1] Kindervag J, Ferrara E, Hollandand R, Shey H. Developing a framework to improve critical infrastructure cybersecurity. Tech. rep., Forrester Research, Inc; 2013.

[2] Gratian M, Bandi S, Cukier M, Dykstra J, Ginther A. Correlating human traits and cyber security behavior intentions. Comput Secur 2018;73:345–58. http://dx.doi.org/10.1016/j.cose.2017.11.015.

[3] West R, Mayhorn C, Hardee J, Mendel J. Social and human elements of information security: Emerging trends and countermeasures. Hershey, PA: IGI Global; 2009, p. 43–60. http://dx.doi.org/10.4018/978-1-60566-036-3.ch004.

[4] Security I. Cost of a data breach report. Tech. rep., Ponemon Institute; 2019.

[5] Egelman S, Peer E. Predicting privacy and security attitudes. ACM SIGCAS Comput Soc 2015;45(1):22–8. http://dx.doi.org/10.1145/2738210.2738215.

[6] Canali D, Bilge L, Balzarotti D. On the effectiveness of risk prediction based on users browsing behavior. In: ASIA CCS '14 proceedings of the 9th ACM symposium on information, computer and security. New York, NY: ACM; 2014, p. 171–82. http://dx.doi.org/10.1145/2590296.2590347.

[7] Mitnick KD, Simon WL, Wozniak S. The art of deception: Controlling the human element of security. Hoboken, NJ: Wiley; 2002.

[8] Curzon P, Rukšėnas R, Blandford A. An approach to formal verification of human–computer interaction. Form Asp Comput 2007;19(4):513–50. http://dx.doi.org/10.1007/s00165-007-0035-6.

[9] A D, M. H.

[10] Bolton M, Bass E, Siminiceanu R. Using formal verification to evaluate human-automation interaction: A review. Syst, Man, Cybern: Syst, IEEE Trans 2013;43:488–503. http://dx.doi.org/10.1109/TSMCA.2012.2210406.

[11] Houser A. Mental models for cybersecurity: A formal methods approach. 2018.

[12] Egelman S, Peer E. Scaling the security wall: Developing a security behavior intentions scale (SeBIS). In: CHI '15 proceedings of the 33rd annual ACM conference on human factors in computing systems. New York, NY: ACM; 2015, p. 2873–82. http://dx.doi.org/10.1002/andp.19053221004.

[13] Chen B, et al. A security awareness and protection system for 5G smart healthcare based on zero-trust architecture. IEEE Internet Things J 2020;403–16.

[14] Mandal S, Khan DA, Jain S. Cloud-based zero trust access control policy: An approach to support work-from-home driven by COVID-19 pandemic. New Gener Comput 2021;39.3:599–622.

[15] Eidle D, et al. Autonomic security for zero trust networks. IEEE 8th Annual UEMCON; 2017.

[16] Dean E, et al. Toward a zero trust architecture implementation in a university environment, Vol. 6.4. The Cyber Defense Review; 2021, p. 37–48.

[17] West R. The psychology of security. Psychol Secur: Why Do Good Users Make Bad Decis? 2008;51(4):34–40. http://dx.doi.org/10.1145/1330311.1330320.

[18] Herley C. So long, and no thanks for the externalities: The rational rejection of security advice by users. In: NSPW '09 proceedings of the 2009 workshop on new security paradigms workshop. New York, NY: ACM; 2009, p. 133–44. http://dx.doi.org/10.1145/1719030.1719050.

[19] Halevi T, Memon N, Lewis J, Kumaraguru P, Arora S, Dagar N, et al. Cultural and psychological factors in cyber-security. In: IiWAS '16 proceedings of the 18th international conference on information integration and web-based applications and services. New York, NY: ACM; 2016, p. 318–24. http://dx.doi.org/10.1145/3011141.3011165.

[20] Baier C, Katoen JP. Principles of model checking. Cambridge, MA: MIT Press; 2008.

[21] Security I. Cost of a data breach report. Tech. rep., Ponemon Institute; 2019.

[22] P. CP. RTL hardware design using VHDL: Coding for efficiency, portability, and scalability. Hoboken, NJ: Wiley; 2006, p. 313–71. http://dx.doi.org/10.1002/0471786411.ch10.

[23] Cimatti A, Clarke E, E. G, F. G, M. P, M. R, et al. NuSMV 2: An OpenSource tool for symbolic model checking. In: CAV '02 Proceedings of the 14th international conference on computer aided verification. Berlin, Heidelberg: Springer; 2002, p. 359–64. http://dx.doi.org/10.1007/3-540-45657-0_29.

[24] Uppaal. Uppaal website. 2010, http://www.uppaal.org.

[25] Owre S, Rajan S, Rushby JM, Shankar N, Srivas M. PVS: Combining specification, proof checking, and model checking. In: 1996 Proceedings of computer aided verification: 8th international conference. Berlin, Heidelberg: Springer; 1996, p. 411–4. http://dx.doi.org/10.1007/3-540-61474-5_91.

[26] Moura LD, Bjørner N. Z3: An efficient SMT solver. In: Proceedings of the theory and practice of software, 14th international conference on tools and algorithms for the construction and analysis of systems. TACAS'08/ETAPS'08, Berlin: Heidelberg: Springer-Verlag; 2008, p. 337–40, http://dl.acm.org/citation.cfm?id=1792734.1792766.

[27] Bengtsson J, Larsen K, Larsson F, Pettersson P, Yi W. Uppaal: A tool suite for automatic verification of real-time systems. Theoret Comput Sci 1996.

[28] Larsen KG, Pettersson P, Yi W. Model-checking for real-time systems. In: Proc. of fundamentals of computation theory. Lecture notes in computer science, (965):1995, p. 62–88.

[29] Alur R, Dill DL. A theory of timed automata. Theoret Comput Sci 1999;126:183–235.

[30] Bozga M, Daws C, Maler O, Olivero A, Tripakis S, Yovine S. KRONOS: A model-checking tool for real-time systems. In: Proceedings of the 10th international conference on computer aided verification (CAV'98), Vol. 1998. Berlin: Heidelberg: Springer-Verlag; 1998, p. 546–50.

[31] Behrmann G, David A, Larsen KG. A tutorial on uppaal 4.0. 2006.

[32] Olmstead K, Smith A. Americans and cybersecurity. Tech. rep., Paw Research Center; 2017.

[33] Albayram Y, Khan MMH, Jensen T, Nguyen N. "...Better to use a lock screen than to worry about saving a few seconds of time": Effect of fear appeal in the context of smartphone locking behavior". In: Proceedings of the thirteenth symposium on usable privacy and security. Berkeley, CA: USENIX; 2017, p. 49–63.

[34] Support M. How to change the logon screen saver in windows. 2018.

[35] Cappelli D, Moore A, Shimealland TJ, Trzeciak R. Common sense guide to prevention and detection of insider threats. 2006.

[36] Barrett D, Hausman KK, Weiss M. CompTIA Security+ SY0-401 Exam Cram. Hoboken, NJ: Pearson IT Certification; 2015, p. 422–37.

[37] LastPass. LastPass psychology of the password. Tech. rep., LastPass; 2016.

[38] Martin S, Tokutomi M. Password cracking. 2012.

[39] Maddox I, Moschetto K. Modern password security for users: User-focused recommendations for creating and storing passwords. 2019.

[40] Ives B, Walsh KR, Schneider H. The domino effect of password reuse. Human-Comput Etiquette 2004;47(4):75–8. http://dx.doi.org/10.1145/975817.975820.

[41] Verizon. Data breache investigations report. Tech. rep., Verizon; 2019.

[42] of Homeland Security UD. If you see something, say something. 2015.

[43] Easen N. Speed is key in tackling data breach fallout. Raconteur: Cybersecur 2019.

[44] Alotaibi M, Furnell S, Clarke N. Information security policies: A review of challenges and influencing factors. In: 11th International conference for internet technology and secured transactions. New York, NY: IEEE; 2016, http://dx.doi.org/10.1109/ICITST.2016.7856729.

[45] Pahnila S, Siponen M, Mahmood A. Employees' behavior towards IS security policy compliance. In: 2007 40th Annual Hawaii international conference on system sciences. New York, NY: IEEE; 2007, http://dx.doi.org/10.1109/HICSS.2007.206.

[46] Vaniea KE, Rader E, Wash R. Betrayed by updates: how negative experiences affect future security. In: CHI '14 proceedings of the SIGCHI conference on human factors in computing systems. New York, NY: ACM; 2014, p. 2671–4. http://dx.doi.org/10.1145/2556288.2557275.

[47] Sarabi A, Zhu Z, Xiao C, Liu M, Dumitras T. Patch me if you can: A study on the effects of individual user behavior on the end-host vulnerability state. In: 18th International conference on passive and active network measurement. Cham, Switzerland: Springer; 2017, p. 113–25. http://dx.doi.org/10.1007/978-3-319-54328-4_9.

[48] Wash R, Rader E, Vaniea K, Rizor M. Out of the loop: How automated software updates cause unintended security consequences. In: 10th Symposium on usable privacy and security. Berkeley, CA: USENIX; 2014.

[49] Kong S, Gao S, Chen W, Clarke E. dReach: δ-reachability analysis for hybrid systems. In: 21st International conference on tools and algorithms for the construction and analysis of systems. Berlin/Heidelberg: Springer; 2015, p. 200–5. http://dx.doi.org/10.1007/978-3-662-46681-0_15.

[50] Eleftherakis G, Kefalas P. Advances in signal processing and computer technologies. World Scientific and Engineering Society Press; 2001, p. 321–6.