Towards Run-Time Verification of Adaptive Security for IoT in eHealth

Arild B. Torjusen
Norwegian Computing Center
P.O.Box 114, Blindern
NO-0314 Oslo, Norway
+47 22 85 25 54
Arild.Torjusen@nr.no

Habtamu Abie
Norwegian Computing Center
P.O.Box 114, Blindern
NO-0314 Oslo, Norway
+47 22 85 25 95
Habtamu.Abie@nr.no

Denis Trcek
University of Ljubljana
Tržaška c. 25
1000 Ljubljana, Slovenia
+386 1 241 85 00
Denis.Trcek@fri.uni-lj.si

Åsmund Skomedal
Norwegian Computing Center
P.O.Box 114, Blindern
NO-0314 Oslo, Norway
+47 22 85 26 06
Asmund.Skomedal@nr.no

Ebenezer Paintsil
Point Transaction Systems AS
Østre Aker vei 24
Pb 73 Økern, N-0508 Oslo
+47 815 02 200
Ebenezer.Paintsil@point.no

ABSTRACT
This paper presents the integration of run-time verification enablers into the ASSET adaptive security framework. Run-time adaptive behavior that deviates from the normal mode of operation of the system represents a major threat to the sustainability of critical eHealth services. Therefore developing verification methods for guaranteeing the achievement of the required self-adaptive runtime security properties is one of the major challenges facing the entire security research field. This paper integrates runtime verification enablers in the feedback adaptation loop of the ASSET adaptive security framework to guarantee the achievement of self-adaptive security and privacy properties in the eHealth settings. The roles of the enablers are (i) to make machine processable formal models of the system state and context available at run-time; (ii) to make requirements, which define the objectives of verification, available at run-time as formal specifications; and (iii) to enable dynamic context monitoring and adaptation. The paper also presents the results of the high level instantiation of the run-time verification framework in Color Petri Net (CPN) and the validation of the model.

Categories and Subject Descriptors
D.4.6 [Security and Protection]: Verification, Access controls, Authentication, and Information flow controls

General Terms
Algorithms, Security, Theory, Verification

Keywords

1. INTRODUCTION
A major threat to the sustainability of critical eHealth services is the run-time adaptive behavior that deviates from the normal mode of operation of the system. Therefore developing verification methods for guaranteeing the achievement of the required self-adaptive runtime security properties is one of the major challenges facing the entire security research field. Adaptive security is quite a challenging task on its own. Although ideas about adaptive security are many years old (and there exist some efforts that mimic biological systems), the full-blown implementations are yet to be seen. Because of this we have decided to follow a different path, which is a formal methods based (not to mention that another reason is that ASSET project [27] addresses also formal verification of self-adaptive systems). The ASSET project develops risk-based adaptive security methods and mechanisms for smart Internet of Things (IoT) in eHealth.

Formal methods provide the means to rigorously specify and reason about the behavior of adaptive systems and have been applied during both system development and runtime to provide guarantees about the required properties of self-adaptive systems. In self-adaptive systems (SAS) the focal point for verification moves from design-time to run-time. While for the non-adaptive parts of a SAS, traditional and well-established verification methods may be applied at design-time, self-adaptive properties and objectives must be verified at run-time to cope with changing environmental conditions and the self-adaption and evolvement of the system itself. Formal methods provide the means for rigid specification, reasoning and automatic tool supported run-time verification. Applying light weight formal methods during execution is seen as a viable way forward [14]. Developing certifiable verification and validation (V&V) methods and tools is critical for the success of autonomous, autonomic, smart, self-adaptive and self-managing systems [8].

This paper presents the integration of runtime verification enablers in the feedback adaptation loop of the ASSET adaptive security framework to guarantee the achievement of self-adaptive security properties in the eHealth settings. These enablers make machine processable formal models of the system state and context available at run-time, make requirements that define the objectives of
verification available at run-time as formal specifications, and enable dynamic context monitoring and adaptation.

For a consistent process for the integration of these enablers in the ASSET framework this paper also presents the high level instantiation of the run-time verification framework in Color Petri Net (CPN) and the validation of the model. CPN provides a formal semantics for the modelling of the ASSET feedback control loops and their interaction with the run-time verification enablers and allows the analysis of the dynamics of the adaptations both at design time and at run-time.

The rest of the paper is organized as follows. Section 2 presents the background of the run-time verification of self-adaptive security. Section 3 describes the main models of the ASSET risk-based adaptive framework that are relevant to this study. Section 4 presents the integration of run-time enablers into the ASSET framework. Section 5 describes the CPN instantiation of the integrated ASSET framework. Finally Section 6 presents the conclusions and future work.

2. BACKGROUND

Automated reasoning and verification play pivotal role in modelling the interactions between a security system and the potential attackers. Based on such models the security of the target systems can be evaluated using rigorous, mathematically-sound techniques, and privacy and security level can be adjusted according to practical concerns and dynamic contexts.

For the verification activities in SAS standard model checking, using a variety of modelling and property languages, probabilistic model checking, using Discrete Time Markov Chains, and a variant of the latter, viz. quantitative verification [15] have been applied. The combination of high complexity due to the adaptive setting and constraints on execution time and memory often renders model checking of SAS at run-time unachievable. While improvement may be achieved by, e.g., enhanced algorithms, state space-reduction and incremental analysis, analyzing real-scale SAS is a challenge. An alternative to standard model checking is parametric probabilistic model checking [28] where the analysis process is split into two phases: (i) at design time a computational complex parametric analysis is done, and (ii) at run time a much simpler analysis on the actual values substituted for parameters is done. An alternative divide and conquer approach is to specify the SAS components' external observable behavior as rely/guarantee specifications, and use compositional reasoning.

There is no commonly accepted best practice for runtime V&V methods for adaptive systems. Four foundational questions that need to be addressed (research drivers) for any approach to V&V for adaptive systems are:

- Characterization of the Viability Zone, i.e., the set of states where the system's requirement and adaptation goals (desired properties) are satisfied. This is highly context dependent; hence dynamic context monitoring is required.

- Validation domain: What are (1) the adaption goals (properties of the target system), and (2) adaption properties (properties of the adaption mechanism) – (1) could typically include quality attributes while (2) would include adaption properties such as stability, accuracy, settling time, overshoot, robustness etc. A framework for evaluating self-adaptive systems have been proposed in [36] which also includes a set of adaption properties based on a survey of SAS system papers. The relevant context, the mapping between the two kinds of properties, and where and how they are observed must also be identified. In addition to identification of properties, well-defined metrics is needed to enable runtime V&V of SAS. Some examples can be found in [36] but more research is required. It remains an important challenge to investigate how formal techniques may be used to assess relevant properties at run-time. In traditional systems simulation of environment behavior can be used for V&V. In adaptive systems runtime-monitoring is necessary since the SAS system will change in response to context changes at run-time. To understand the dependency of properties on runtime-changes is an important challenge.

- Where in the adaptation cycle should V&V tasks be performed? This regards separation of concerns between the target system and the adaptation mechanism. V&V tasks should be incorporated as a part of the adaptation mechanism; hence additional components dedicated for V&V should be added to the common context monitoring elements. A challenge is to avoid interference with the target system, which should not be adversely affected to an unacceptable degree by testing. A qualitative model of runtime testability is presented in [11] where the concepts of test sensitivity and test isolation characterize resp. the degree of which testing may affect the system and the means that may be employed to reduce interference.

- When to do V&V – e.g., at design time, configuration time or runtime? On one hand formal methods applied at runtime may become too computationally expensive since they have to tackle high complexity due to adaptivity, while on the other hand at runtime context dependent variables are bound, which reduces complexity and the verification space. The answer to this question is highly dependent on which properties are targeted for verification.

A recent survey [10] shows an increasing attention for formal methods in SAS but attention to formal methods at run-time remains sparse. Reasoning and model checking are the most common applications, targeting functional correctness and different quality characteristics. Security has received considerably less attention in existing work [10]. Markov models, fuzzy sets and state machines have been used for intrusion detection systems [7], [6], [3], but not at run-time.

A formal reference model (FORMS) for specifying self-adaptive software is presented in [14] and it provides a precise structure for describing and reasoning about the primary architectural characteristics of self-adaptive systems. The reference model is flexible enough to be able to describe three rather different adaption perspectives: reflective computation, distributed coordination and MAPE-K (the latter being the most relevant in ASSET).

Model checking [19] can be used to guide the adaptation process. Adaptation goals or goal policies can be specified formally and, provided that a runtime-model of the system is available, model checking can be used to evaluate candidates for adaption. Model checking can also be used to assess the correctness and consistency of system objectives. An approach using the model checker SPIN can be found in [29], but scalability is the main challenge and the examples are far from full-scale systems.

A more recent alternative to model checking is quantitative verification [15], which seems promising for run-time verification
of the correctness, performance and reliability properties of systems
with stochastic behavior [13]. System and domain are modelled
using discrete-time Markov chains, requirements are modelled
using probabilistic computation tree logic, and verification is done
with a probabilistic model checker. This approach complements
earlier work based on model checking, which under the label
runtime certification [16] targets qualitative correctness properties
such as (e.g., safety and reachability).

Quantitative verification has been realized using the probabilistic
model checker PRISM and the general-purpose autonomic
computing framework GPAC and applied to domains such as
Dynamic power management. Adaptive allocation of data-center
resources and Dynamic QoS management in service-based systems
[18].

Under the heading runtime verification there are several
approaches where execution traces are recorded and analyzed at
time to verify correctness requirements. These may be
expressed in temporal logics, state machines, regular expressions,
rule systems and action-based contract languages.

Danny Weyns [17] proposes an integrated approach to validate
required software quality in self-adaptive systems by combining
three activities: (1) model checking design, (2) model-based testing
during development, and (3) runtime analysis supporting
adaptations for particular quality goals. A prerequisite for all is a
good understanding of properties particular to self-adaptive
systems, such as: stability of adaptation behavior, failsafe updates,
progress during updates, adaptation integrity, mismatch, and
interference freedom. Getting a better understanding of these
properties and how they can be verified, and the realization of
runtime monitoring is pointed out as areas that need further
research.

Of lightweight formal methods, model finding has been used to
derive models for specifications [18]. From a formal specification
of a computer system, model-finding techniques are employed to
synthesize a model of the system that satisfies the specification.
The synthesized model corresponds to a system configuration that
fulfills the system goals given its current state, and is used to derive
a configuration procedure capable of reaching this target
configuration without violating any system constraints. The formal
specification framework Alloy [31] and the model finder Alloy
Analyzer was used in the mentioned work. A challenge of this
approach is limited scalability.

Adaptive Verification (AdaptiV) [21] is a tool chain and
methodology for verifying adaptive systems which consists of an
interesting combination of techniques: 1) stability analysis to
to partition the system model into stable and unstable (adaptive)
component models, thus reducing the state space that needs
verification; 2) state space reduction techniques that will be applied
to counter state space explosion when model checking the adaptive
components, 3) high performance computing (HPC) simulations to
explore component behavior for the reduced state space of unstable
components, 4) compositional verification techniques to enable
reasoning of the whole system based on component verification;
and 5) operational monitors to detect and take action to correct
undesired unstable behavior of the system during operation. In the
AdaptiV methodology only the last technique will be employed at
run-time, however the other techniques are of interest when
developing runtime methods for verification of SAS.

A general multi-level model for SAS is proposed in [25]. The
lower level describing the actual behavior of the system modelled
as a state machine and the upper level describing the dynamically
changing environmental constraints on the system as a second-
order state machine whose states have associated formulas over
observable variables of the lower level. This gives a formal
computational characterization of self-adaptive systems that forms
a sound basis for further analysis. The authors point out interesting
extensions for their work. These extensions will be investigated in
the ASSET architecture: A higher-level algebraic language for
specifying multi-level systems may be able to handle more complex
and larger models of adaptive systems; by embedding quantitative
aspects into the models, a more fine-grained view on the probable
development of adaptation paths may be achieved, thus improving
prediction; and finally by extending the knowledge between the
two levels of the model, runtime model checking techniques may
be used to enable the system to behave in an anticipatory way, i.e.,
making it possible to adjust present behavior in order to address
future faults.

One major challenge in adaptive systems is to provide guarantees
about the required runtime quality properties. Formal methods
provide the means to rigorously specify and reason about systems
and have been applied during both system development and
runtime to provide guarantees about the required properties of self-
adaptive systems [23], [24], [26], [14]. The formal specification,
assessment, and verification of the ASSET framework thus involve
verifying that the description of the security and privacy models
ensures the correctness of security and privacy solutions.

3. ADAPTIVE SECURITY FRAMEWORK
FOR IOT IN E-HEALTH

Adaptive security here refers to a security solution that learns, and
adapts to changing environment dynamically and anticipates
unknown threats without sacrificing too much of the efficiency,
flexibility, reliability and security of the IoT system. It involves
gathering contextual information both from within the system and
from the environment, measuring security level, analyzing the
collected information, and responding to changes. Adaptation is
achieved (i) by adjusting internal working parameters such as
encryption schemes, security protocols, security policies, security
algorithms, different authentication and authorization mechanisms,
changing the QoS (Quality of Service) available to applications,
and automating reconfiguration of the protection mechanisms,
and/or (ii) by making dynamic changes in the structure of the
security system.

Adaptation forms include parametrical adaptation, structural
adaptation, goal adaptation, and/or combination of these. The
analysis part of such an adaptive approach requires flexible
learning and decision making processes that help set priorities and
make the best decision when both the qualitative and the
quantitative aspects of a decision need to be considered [27]. In
this section we present ASSET’s high level risk-based adaptive
framework that meets these challenges.

The framework closes the adaptive feedback control loop of
management of security and privacy risks, dynamically taking into
account the necessary context information to ensure efficiency over
time. It applies the Monitor-Analyze-Adapt (plan, execute and
learn) methodology by using feedback mechanisms. The
framework learns at run-time by (i) combining adaptive risk-based
decision model, adaptive security and privacy models, and
actuators to make effective adaptive reaction, and (ii) integrating
different metrics for validation and verification, adaptive risk
assessment, and predictive analytics models for estimation and prediction of security and privacy risks and impacts.

The framework consists of (i) the adaptive risk management model, (ii) the adaptive monitoring model, (iii) the analytics and predictive models, (iv) the adaptive decision-making models, and (v) the evaluation and validation models. The following sections describe the high level key features of these models. The ensuing sections briefly describe the main models that are relevant to this study.

3.1 Adaptive Risk Management
Adaptive risk management (ARM) here refers to a risk management model, which is capable to learn, adapt, prevent, identify and respond to known and unknown threats in real-time. The key function of this model is the development of risk-based adaptive security methods and mechanisms for smart IoTs that estimate and predict risk damages and future benefits by integrating adaptive monitoring, analytics and predictive models, adaptive decision-making models, and evaluation and validation models in a continuous cycle, thus enabling the security methods and mechanisms to adapt their security decisions upon those estimates and predictions.

To meet these challenges the ARM model takes the following five required measures into account [27]: (i) Identify - ability to predict problems, (ii) analyze - ability to predict impact, (iii) plan - ability to implement planned actions, (iv) track - ability to maintain management focus on risk mitigation actions, and (v) control - ability to reduce risk exposure. These are achieved through the coordination of the other models which are described in the ensuing sections.

3.2 Adaptive security monitoring (Monitor)
The Monitor adapts the reference architecture of a general monitoring framework in [34] for obtaining automated technical evidence for the purposes of continuous operational security monitoring. It uses a continuous cycle of monitoring of the information about context and status of the smart IoTs which is exploited at runtime in the adaptation process. The requirements to monitoring the context need to adapt to a constantly changing context which is important to optimize the effectiveness of verification. Run-time verification of output from feedback control loops enables verification of adaptation plans before (or concurrent with) instrumenting them and include relevant mechanisms to keep track of the validation aspects.

3.3 Analytics and predictive models (Analyze)
The Analyzer (analytics and predictive models) analyzes the monitored information from the adaptive monitoring model using game theory and context awareness to estimate and predict security and privacy risks and future benefits dynamically. Game theory was chosen because it can model the dynamic behavior of stakeholders with conflicting interests including the strategies of real world adversaries. The model also uses to further improve the accuracy of estimation and prediction mechanisms by applying optimized machine learning and rule-based algorithms. This will increase the ability to precisely predict and measure the risk of damages and future benefits and adapt security decisions upon those predictions. Moreover, the models will improve the light-weight abilities of smart things by improving their context-awareness and self-abilities. This includes optimizing algorithms for different IoT processing capabilities to detect in real-time unknown security and privacy threats, respond to them, and adapt to the environment and changing degree of security and privacy breaches.

3.4 Adaptive security models (Adapt)
The Adapt model adapts to the dynamism of Things, their interactions, and the environment, and to the varying degrees of risk that the IoT eHealth system will be compromised. It does this by dynamically determining whether changes and adaptation should be made or not, and, if to be made, to select the ‘best’ adaptive security model for a given situation, and to apply the identified changes and adaptation by ensuring the highest likelihood of achieving the greatest benefit for the smallest risk.

However, to avoid inconsistencies the adaptation requires a sound reasoning about the correctness of the system. The basic idea is to use formal techniques in a way where minimal risk is achieved by appropriate system configuration. One such approach is that combines goal-modeling techniques with constraint programming to identify the variants that are best suited for a particular environmental context [11]. With this approach the system uses architectural adaptation that is conceptualized as dynamic software product line (DSPL). Each configuration (depending on the context) represents one possible variant of the DSPL. The variability can be structural (this concerns the particular configuration of components), and it can be environmental (it concerns the states that may be adopted by the environment at a runtime).

Now as shown in [1] the specification of a DSPL can be reduced to a formal representation, more precisely a constraint satisfaction problem. More precisely, there exists a tool Variability Modeling Language VML aligned with DSPL. The VML model represents a constrained optimization problem and Maude [32] is used for prototyping, simulation and verification.

4. FORMAL VERIFICATION OF SELF-ADAPTIVE SECURITY BEHAVIOUR
Run-time adaptive behaviour that deviates from the normal mode of operation of the system represents a major threat to the sustainability of critical infrastructure services. Verification methods for guaranteeing the achievement of self-adaptive security and privacy properties are necessary.

A common feature of state of the art approaches to runtime verification of self-adaptive systems is to integrate the verification activities in the adaptation process by making the activities explicit in the feedback loop [24], [12], [20], [22].

This section integrates run-time verification enablers in the ASSET feedback adaption loop working alongside the adaption mechanisms. Four enablers models at run-time, requirements at run-time, dynamic context monitoring, and runtime verification component are developed and integrated in every phase of ASSET feedback the control loop to provide effective support to materialize verification assurances for self-adaptation [4], [5]. Figure 1 depicts this integration and the ensuing section describes these enablers.

Run-time verification in ASSET builds on the approach proposed in [24] of integrating the V&V activities in the adaptation process and making the activities explicit in the feedback loop. In this framework the Runtime V&V component verify outputs from the
4.3 Dynamic context monitoring
Dynamic context monitoring - Since the context (e.g. threat scenarios and even monitoring requirements) is constantly changing at run-time, dynamic context monitoring needs to adapt to context change thereby enabling run-time verification with relevant monitoring mechanisms that keep track of aspects to validate. Based on current state of the art such an enabler will be developed and integrated into the “Monitor” phase of the ASSET feedback control loop.

4.4 Runtime verification component
Runtime verification component – Runtime verifier will be developed and integrated into the “Analyze & Learn” phase of the ASSET feedback control loop to verify outputs from the “Analyzer & Learner” Component enabling verification of an adaptation plan before or after instrumenting it.

5. HIGH LEVEL INSTANTIATION
This section shows how the ASSET runtime verification framework in Figure 1 can be instantiated by means of colored petri nets (CPN). CPN provides means for specifying, visualizing, simulating and analyzing the static framework before implementation in order to reduce implementation errors [35].

5.1 CPN instantiation
CPNs’ models consist of places, transitions (events), input and output arcs. We represent the places by ellipses, transitions by rectangles, input/output arcs by directed arcs. A place may hold a collection of tokens and may represent system conditions. A CPN token is a variable with data type and a value. We refer to the data type as color set and the values as token colors. The set of tokens on all the places at a given moment represents the system state or marking. The transition represents the events or actions that can cause a system to change state. An arc serves as data input and output for a transition. It enables a transition to remove one or more tokens from an input place to an output place. When this happens, we say that the transition is fired.

Figure 2: Top Level CPN Model

One concrete implementation of an autonomic system prototype with ability to self-test can be found in [12] here the adaptation control loop (MAPE-K) is extended with test managers alongside the existing adaption managers, two validation strategies are used, safe adaptation with validation (ensuring that the integrity of the system is maintained during adaptation) [20], and replication with validation (changes are validated on a replication of the system before being implemented). Also similar to the approach in [24] is MAPE-T [22], a feedback loop for supplementing testing strategies with run-time capabilities based on the monitoring, analysis, planning, and execution (MAPE-K [5]) architecture for adaptive systems. The paper identifies some challenges for adaptively testing a SAS at run time.

4.1 Models at run-time
The Models at run-time require the machine processable/formal models or abstractions of the system state and context must be available at run time. A variety of models and formalisms have been proposed in the literature. Based on current state of the art a system abstraction applicable for adaptive security management in a critical infrastructure environment will be developed and integrated into both “Analyze & Learn” and “Adapt (Decide & Act)” phases of the ASSET feedback control loop.

4.2 Requirements at run-time
Requirements at run-time – Requirements including adaptation properties define the objectives of verification and must be available at run-time as machine-readable/formal specifications. Verification techniques need to cope with the dynamic nature of the adaptive system. In particular to support incremental verification the ability to trace changes to requirements is needed. Based on current state of the art formalism for capturing relevant requirements for adaptive security in critical infrastructures will be developed and integrated in the “Analyze & Learn” phase of the ASSET feedback control loop.

Figure 1. ASSET run-time verification framework

Figure 2. Top Level CPN Model

4.3 Dynamic context monitoring
Dynamic context monitoring - Since the context (e.g. threat scenarios and even monitoring requirements) is constantly changing at run-time, dynamic context monitoring needs to adapt to context change thereby enabling run-time verification with relevant monitoring mechanisms that keep track of aspects to validate. Based on current state of the art such an enabler will be developed and integrated into the “Monitor” phase of the ASSET feedback control loop.

4.4 Runtime verification component
Runtime verification component – Runtime verifier will be developed and integrated into the “Analyze & Learn” phase of the ASSET feedback control loop to verify outputs from the “Analyzer & Learner” Component enabling verification of an adaptation plan before or after instrumenting it.

5. HIGH LEVEL INSTANTIATION
This section shows how the ASSET runtime verification framework in Figure 1 can be instantiated by means of colored petri nets (CPN). CPN provides means for specifying, visualizing, simulating and analyzing the static framework before implementation in order to reduce implementation errors [35].

5.1 CPN instantiation
CPNs’ models consist of places, transitions (events), input and output arcs. We represent the places by ellipses, transitions by rectangles, input/output arcs by directed arcs. A place may hold a collection of tokens and may represent system conditions. A CPN token is a variable with data type and a value. We refer to the data type as color set and the values as token colors. The set of tokens on all the places at a given moment represents the system state or marking. The transition represents the events or actions that can cause a system to change state. An arc serves as data input and output for a transition. It enables a transition to remove one or more tokens from an input place to an output place. When this happens, we say that the transition is fired.

Figure 2: Top Level CPN Model

One concrete implementation of an autonomic system prototype with ability to self-test can be found in [12] here the adaptation control loop (MAPE-K) is extended with test managers alongside the existing adaption managers, two validation strategies are used, safe adaptation with validation (ensuring that the integrity of the system is maintained during adaptation) [20], and replication with validation (changes are validated on a replication of the system before being implemented). Also similar to the approach in [24] is MAPE-T [22], a feedback loop for supplementing testing strategies with run-time capabilities based on the monitoring, analysis, planning, and execution (MAPE-K [5]) architecture for adaptive systems. The paper identifies some challenges for adaptively testing a SAS at run time.

4.1 Models at run-time
The Models at run-time require the machine processable/formal models or abstractions of the system state and context must be available at run time. A variety of models and formalisms have been proposed in the literature. Based on current state of the art a system abstraction applicable for adaptive security management in a critical infrastructure environment will be developed and integrated into both “Analyze & Learn” and “Adapt (Decide & Act)” phases of the ASSET feedback control loop.

4.2 Requirements at run-time
Requirements at run-time – Requirements including adaptation properties define the objectives of verification and must be available at run-time as machine-readable/formal specifications. Verification techniques need to cope with the dynamic nature of the adaptive system. In particular to support incremental verification the ability to trace changes to requirements is needed. Based on current state of the art formalism for capturing relevant requirements for adaptive security in critical infrastructures will be developed and integrated in the “Analyze & Learn” phase of the ASSET feedback control loop.
These monitors and runtime verifier are responsible for validating or verifying their respective decisions at runtime. The runtime enablers are integrated into the model. It is possible to model the enablers outside the model.

The IOTEnvironment corresponds to the SmartIOT in Figure 1. The IOTEnvironment consists of things, their identifications, contexts etc. The threat that emerges from the IOTEnvironment flows to the Monitor, a strategy is generated based on the threats, vulnerabilities and the resources that are at risk. The strategy flows to the Analyzer which computes the impact and sends the result to the Adaptor to adapt if the risk impact is not acceptable. The loop continues to the IOTEnvironment where the entire process may be repeated. In [30] a context-aware Markov game theoretic model for risk impact assessment to measurably evaluate and validate the runtime adaptivity of the ASSET security solutions. Some of the models mentioned hereinafter such as adversary, game, impact, vulnerability, patient, and threat and their interactions are shown in Figure 2 in [30].

![Figure 3: IOTEnvironment Model](image)

Figure 3 represents the IOTEnvironment of the model. It consists of the adversary transition which observes or examines the activities of the environment and generates an appropriate threat. The Patient place produces personal data such as AGE in the IOTEnvironment. The environment may be vulnerable in a given context. The context and the vulnerability places represent the Context and Vulnerability respectively in the IOTEnvironment. The threat generated by the adversary is sent to the Monitor substitution transition in the adaptive loop.

![Figure 4: The Monitor Submodel](image)

Figure 4 is the Monitor of the adaptive loop. The Monitor monitors the IOTEnvironment and the threat generated by the adversary. Based on this information a strategy is determined by the Monitor which is examined by the runtime DCVMonitor before it passes on to the Analyzer in the adaptive loop. The runtime monitor function (monitor1()) examines the strategy for any unexpected outcome and normalizes the outcome before it reaches the Adaptor transition. The share database DB is the knowledge base of the entire loop. The DBs are connected by fusion sets.

![Figure 5: Analyzer Submodel](image)

Figure 5 represents the Analyzer in the adaptive loop. Here we assume that the analyzer will use game theory for the decision making. The Game transition will accept the strategy as an input and compute the risk impact. The result is validated by the RM Runtime transition before it gets to the Adaptor submodel.

![Figure 6: Adaptor Submodel](image)

Figure 6 represents the Adaptor submodel. It decides if the strategy should be adapted depending on the risk impact. The VM Runtime transition validates the decision before the response is sent.

Data types:
- \( \text{colset \text{PATIENT}= \text{with \text{AGE}}, \text{WEIGHT};} \)
- \( \text{colset \text{VULNERABILITY}= \text{with \text{VUL1}}, \text{VUL2};} \)
- \( \text{colset \text{CONTEXT}= \text{with \text{CXT1}}, \text{CXT2};} \)
- \( \text{colset \text{STRATEGY}= \text{with \text{STR1}}, \text{STR2}, \text{STR3}, \text{NA};} \)
- \( \text{colset \text{THREAT}= \text{with \text{TH1}}, \text{TH2};} \)
- \( \text{colset \text{IMPACT}= \text{with \text{LOW}}, \text{MEDIUM}, \text{HIGH}, \text{UNKNOWN};} \)
- \( \text{colset DB= union \text{cast1:STRATEGY + cast2:IMPACT};} \)

Functions:
- \( \text{fun monitor1(str:STRATEGY)=if str=NA then str=STR3 else str=str;} \)
- \( \text{fun monitor2(imp:IMPACT)=if imp=UNKNOWN then imp=HIGH else imp=imp;} \)
- \( \text{fun verify(pact:IMPACT)=if pact=LOW then 1"OK" else 1"Adapt";} \)
- \( \text{fun adapt()=true;} \)

### 5.2 Validation of the Model

We generate the state space graph of the model using the CPNTools. The statistics show that model generated a state space of 41 nodes and 40 arcs. Similarly the strongly connected components graph (SCC) also generated the 41 nodes and 40 arcs. This means there is no infinite occurrence sequences or cycle in the model. Hence the model may terminate because we decided to initialize the model to a finite set of initial markings. Infinite loop can easily be modeled by modifying the arcs or the initial markings in the IOTEnvironment and Monitor submodel.
6. CONCLUSION AND FUTURE WORK

This paper presents the development of run-time verification of self-adaptive security behavior for guaranteeing the achievement of self-adaptive security properties. It integrates runtime verification enablers in the feedback adaptation loop of the ASSET adaptive security framework to validate and verify the achievement of self-adaptive security properties in the eHealth settings. These runtime enablers allow the run-time verification of the outputs from feedback control loops for the validation of adaptation plans before (or concurrently with) instrumenting them, and include relevant mechanisms to keep track of the validation aspects. The paper also presents the high level instantiation of the run-time verification framework in Color Petri Net (CPN) and the validation of the model.

In our future work, we plan to implement context-aware Markov game theoretic estimation and prediction mechanisms in CPN by addressing the timing and delay between the various components and demonstrate its effectiveness in IoT eHealth environment through series of simulations. We also plan to improve the accuracy of the estimation and prediction mechanisms by applying optimized machine learning algorithms for IoT components that are constrained by energy, communications, and computation capabilities.

7. ACKNOWLEDGMENTS

The work presented here has been carried out in a research project ASSET - Adaptive Security for Smart Internet of Things in eHealth (2012-2015) funded by the Research Council of Norway.

8. REFERENCES


