Intelligence Graphs for Threat Intelligence and Security Policy Validation of Cyber Systems

Vassil Vassilev[1], Viktor Sowinski-Mydlarz[1], Pawel Gasiorowski[1], Karim Ouazzane[2] and Anthony Phipps[2]

[1] London Metropolitan University – Cyber Security Research Centre, London, UK
[2] London Metropolitan University – School of Computing and Digital Media, London, UK
v.vassilev,w.sowinskimydlarz,p.gasiorowski,k.ouazzane,tsaphip1@londonmet.ac.uk

Abstract. While the recent advances in Data Science and Machine Learning attract lots of attention in Cyber Security because of their promise for effective security analytics, Vulnerability Analysis, Risk Assessment and Security Policy Validation remain slightly aside. This is mainly due to the relatively slow progress in the theoretical formulation and the technological foundation of the cyber security concepts such as logical vulnerability, threats and risks. In this article we are proposing a framework for logical analysis, threat intelligence and validation of security policies in cyber systems. It is based on multi-level model, consisting of ontology of situations and actions under security threats, security policies governing the security-related activities, and graph of the transactions. The framework is validated using a set of scenarios describing the most common security threats in digital banking and a prototype of an event-driven engine for navigation through the intelligence graphs has been implemented. Although the framework was developed specifically for application in digital banking, the authors believe that it has much wider applicability to security policy analysis, threat intelligence and security by design of cyber systems for financial, commercial and business operations.

Keywords: Knowledge Graphs, Ontologies, Threat Intelligence, Security Policies, Security Analytics

1 Logical Vulnerability, Threats and Risks in Cyber Security

Most organizations, including the banks, typically use fragmented security procedures, which do not cover all possible security threats and attack vectors. Where they pay attention they concentrate mainly on direct technical threats and simple fraud as opposed to sophisticated social engineering fraud, which takes advantage of the loopholes in the security policies. The security policies are typically not shared between the branches and the headquarters and are not integrated into one coherent system. The organizations lack standardized, unified and established methodology to analyze them. Therefore, banks are at high risk from security breaches, fraud and other malicious activities. They are vulnerable and their security policy is ineffective [1].

In a second place, the growing use of cloud services and IoT devices is an industry trend resulting in threats specific to these technologies. Small and medium-sized enterprises without resources to maintain their operations depend on third parties for technical maintenance, servicing and provision. However, the security risks are also high because external services increase the communications and introduce complex access control. The banks are reluctant to employ the cloud technology also due to the increased risks for protection of the financial data [2].

The problem with logical vulnerability largely stems from the fragmentation, high distribution and difficulty in integration of the policy rules due to the lack of standards and tools for automation [3,4]. Top players in cyber security (CISCO, Symantec, Palo Alto, Juniper) deliver predominantly physical level tools. Logical vulnerability analysis is in an early stage of development [5,6,7]. The risks associated with the new technologies complicate the picture even further. Employing Internet-of-Things increases the vulnerability, but although it has been analysed in some critical applications, such as medical and transport systems, it is far from solved in general. Risk assessment is even more difficult despite the recent push in industry for automation [8-11], it is subjective and requires contracting consultants, which charge at a very high rate. Tools for development, testing, analysis and control of the security policies are very rare. They are needed not only for large corporations, financial institutions and service providers but also for SMEs which outsource services offshore and to third parties. Such software would have a great impact for improvement of their security policies and reducing their security risks. The small number of available tools are mainly product of significant investments from large corporate companies [12,13]. They serve the purpose to a various degree, but practically suffer from the lack of solid theoretical foundation and limited potential for use outside of the designated domain.

In this article, we are presenting a new framework for performing analysis of the logical vulnerability, for threat modelling and validation of the security policies in cyber systems. It is based on a four-layer model which, unlike most of the tools supporting threat intelligence [14,15] is strictly formal [16]. The lowest layer of the framework, the *ontological layer* models situations, threats and activities, formalized in description logic using the standard languages of the Semantic Web - RDF/RFS and OWL. The second layer, the *heuristic layer* contains security policies, formalized in clausal logic using the standard rule language of Semantic Web – SWRL. The third layer, the *workflow layer* formalizes the transactions under threats by navigating through the nodes of a directed graph, formed by the situations and governed by the security policies. The top *process layer* performs various analytics for assessing the vulnerabilities, estimating the risks and various security analytics. Although the framework was developed specifically for validation of security policies in banking, we believe that it has much wider applicability in financial, commercial and business systems.

2 Ontologies, Knowledge Graphs and Process Workflows

Ontological engineering is an outcome of the research program for Semantic Web [17]. Core of the software architecture of an ontology-enabled system is the *domain ontology*, which fulfills two different roles: it defines the terminology in the problem domain and simultaneously provides the base for intelligent system analysis, design, operation and control. Standard modeling language for ontologies is OWL [18]. It has strict formal semantics given by Description Logic. The *expert knowledge* in an ontology-enabled intelligent system can be modelled within the paradigm of Semantic Web using rules in SWRL [19]. The main problem faced in this approach is that if the ontology requires explicit representation of both synchronous and asynchronous activities, the logical level becomes too complex to be practically useful due to the infamous *frame problem* in AI. A typical case is the area of cyber security, where the unauthorized intrusions, information leaks, frauds and damages are achieved through malicious ac-

tivities, which are asynchronous, while the protection is achieved by applying preventative or correcting countermeasures, which are synchronous. Because of this, the semantic technologies have been used mainly for formal specification and standardization of the security ontology [20]. An excellent overview of the security ontologies is given in [21]. We avoid these problems by using carefully constructed ontological theory of situations and actions [16] which allows to make the formal approach more practical.

Our framework is multi-layered and operates on four levels: a) the *ontological level*, on which the ontology is modelled using RDF/RDFS/OWL, b) the *heuristic level*, on which the security policies are modelled as rules in SWRL, c) the *workflow level,* on which the analytics according to the underlying logics form directed graphs, and d) *process level,* on which the analytics are executed against the representation and/or external data (analytics on demand). The workflow level is the base for introducing a large number of important security concepts, such as *accessibility*, *vulnerability*, *risks* and *mitigation*, as well as the algorithms for *analyzing* them in a formal manner. This way, it bridges the knowledge-based approach to AI, based on symbolic logic, and the data analytics approach, based on machine learning. As a result, we come to the richer concept of *intelligence graphs* as a vehicle for orchestrating the security analytics. Unlike the approach of knowledge graphs [22], which only combines data analytics with expert heuristics, our approach integrates the modeling and the data analytics adding the possibility to make logical inferences, based on the results of data analytics, and to apply machine learning to the logical inferences themselves.

3 Ontology of Transactions under Security Threats

3. 1 Logical Foundations of the Ontological Modeling

We formalized the security ontology within a logical theory of situations and actions with terminological vocabulary of classes and relationships in standard DL (or concepts and properties in OWL) as shown in Table 1 [16].

Table 1. Predefined terminology on ontological level

Term	OWL	DL	Meaning
`Situation`	concept	unary predicate	Static reference to the world in time
`Item`	concept	unary predicate	Qualitative description of situations, events, threats and items or quantitative valuation
`Threat`	concept	unary predicate	Malicious entity which appears in situations and may lead to transitions
`Event`	concept	unary predicate	Asynchronous activity which happens in situations and may lead to transitions
`Action`	role	binary predicate	Synchronous transition between situations
`occurs-in`	role	binary predicate	Events happening in situations
`tampers-with`	role	binary predicate	Threats interfering with situations
`present-at`	role	binary predicate	Attribution of items to situations
`appears-in`	role	binary predicate	Attribution of items to events
`controlled-by`	role	binary predicate	Attribution of items to threats
`has/ value`	role	binary predicate	Pairing two items in associative link or quantitative valuation of items
`follows`	role	binary predicate	Pairing two situations in temporal order
`causes`	role	binary predicate	Pairing two events in causal dependence

In our theory the static descriptions of the world, the *situations*, are modelled in an object-oriented manner using a hierarchy of classes. The security *treats* form another hierarchy. We distinguish the asynchronous activities (*events*) and synchronous activities (*actions*) by using two different logical constructs – the events, which happen in situations without changing them are modelled as classes, while the actions, which change them are modelled as relations. The parameters of the actions are not represented explicitly but are bound contextually by the situation parameters. Our approach solves the infamous *frame problem* about what needs to be changed and what needs to be preserved during transitions: while the actions bind their input parameters in the current situation they affect the environment only through their output parameters.

Principle of preservation: *Any description of the situations within the domain of the action in terms of input parameters remains unchanged in the situations within its range. This retains the items describing the situations after execution exactly as before.*

Principle of propagation: *Any descriptions of the situations within the domain of the actions which involve their output parameters should be deleted from the situations within their range before the transition is completed. This updates the situations along the execution path.*

These principles are valid because the situations are logical terms themselves, so they can be described using items which serve the role of event and action parameters. Computationally, this reduces the complexity of the logical inference needed to calculate the necessary changes because it allows to use templates in the symbolic representation and to perform indexing of both facts and rules based solely on their structure.

3. 2 Situations, events, threats and items

The main classes of our ontology are organized in four separate taxonomies:

Items: represent the material and conceptual entities of importance (`Account`, `Credentials`, `ATM`, `VoiceApp`, etc.). They provide information related to other entities within the domain of interest and form a well-established taxonomy of classes, not necessarily related to security, but included also some classes related to the security policies, such as *channel*, *connection*, *session*, *credentials*, *profiles*, *permissions*, *threats*, etc.

Situations: model partial state of the world from security point of view (i.e., `s_User_Logged`, `s_Account_Locked`, etc.). From security point of view some of the situations will be vulnerable, while some others will be safe, but this classification depends on both the ontology and the security policies and since the vulnerability as a concept cannot be defined on ontological level, we cannot distinguish the situations from vulnerability point of view.

Events: model the activities which cannot be predicted and controlled (i.e., `e_Credentials_Accepted`, `e_Card_Refused`, etc.). The events by their nature are asynchronous, they can happen in any situation but they do not change the situations – they can only initiate the changes, which actually happen only after executing the corresponding actions.

Threats: model the security threats which may interfere with the normal execution of the transactions (i.e., `t_ManInTheMiddle`). We have identified and modelled 39 different threats of importance in banking domain and their taxonomy is given in Fig. 1 as an example.

All classes in our ontology are modelled in OWL as concepts. The modeling is a preliminary phase of the work in our framework, which is completely independent from its subsequent use for simulation and analysis. It is an interactive process which can be fully automated using ontological editors such as Protégé.

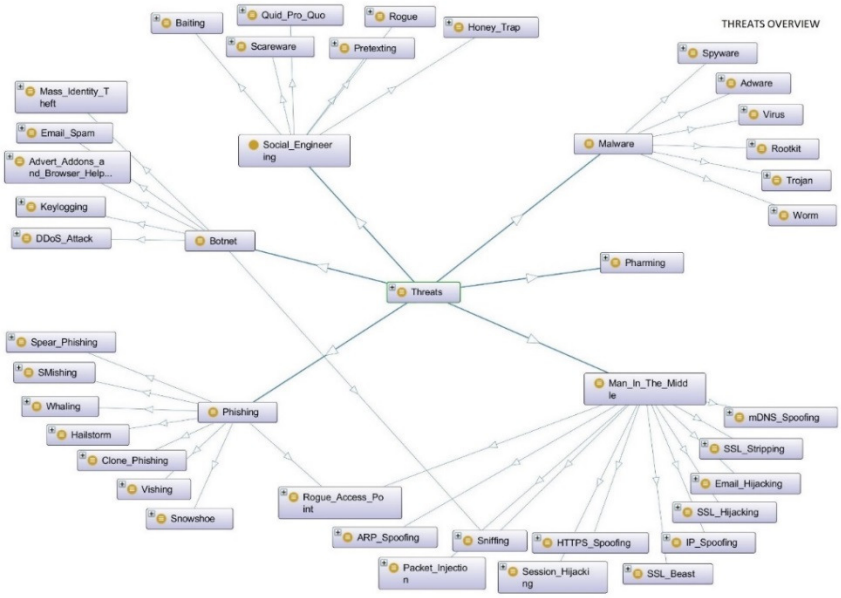

Fig. 1 Taxonomy of Threats in Cyber Security

3. 3 Actions

The actions are synchronous activities which lead to changes of situations. Formally they specify relations between the situations and are modelled as properties of the OWL classes. Since they can be executed under different initiatives – the user, the system or the threats, it is possible to classify them. We have user-initiated actions (they do not need countermeasures), actions triggered by system errors or external interventions (controlled by the system via countermeasures) and actions, which potentially lead the system to a situation which is unwanted (out of control). Based on this understanding, the actions in the ontology have been classified into three separate hierarchies:

Normal: Actions, which are planned as parts of the user journeys in accordance with the endorsed security policies (example: `a_Login`).

Abnormal: Actions, which are result of malfunctioning, unauthorized intrusions, or malicious activities of potential security threats; such actions can lead the system to a vulnerable state (example: `a_Cancel`).

Correcting: Actions, undertaken after abnormal actions or in response to unwanted event which can put the system back on track.

This classification will play an important role in the analysis - the abnormal actions will increase the vulnerability and the risks for completing the user journeys, while the correcting actions will reduce them in accordance with the security policies.

3. 4 Parametrization

Our approach to modeling the actions as relations between situations brings an interesting possibility: the parameters can be defined in terms of properties of the situations they transform. We consider action parameters to be those characteristics of the situa-

tions (items) in which the actions apply, which are common for all of them. This proposes a natural distinction between input and output parameters – the input parameters are the items which describe the situations in which we execute the actions, while the output parameters are the items which describe the situations after execution. For example, the action `a_Login`, which represents the transformation from a state in which we are not logged in to a state in which we are logged, has as an input parameter the item `credentials` needed for authentication, while its output parameter is `session` which can be used for further execution of the operations included in the transaction. The same applies to other dynamic entities, like events and threats - their parameters are the items which characterize all situations in which the events happen (or the threats occur, respectively). Unlike the actions, however, which can have a side effect through their output parameters, the events and threats have only input parameters, determined contextually by the situations in which they happen or occur.

4 Heuristic Level and Security Policies

The ontological level provides reference objects needed for the logical vulnerability analysis, risk assessment and neutralization of potential treats. The heuristic level will model the security policies governing the execution of the transactions under threat.

4.1 Security Policies as Heuristics

The security policies in our framework form the heuristic level. They refer to the classes and properties as defined on ontological level, but their syntax and semantics are not given by the underlying DL logic. For representing the security policy rules we have adopted SWRL, a rule language which binds the concepts and properties of OWL ontology in antecedents/consequents pairs in a similar way to the infamous Horn-clause Predicate Logic (HPL). The policies can be formulation interactively within the same editor which specifies the OWL ontology (in our case, **Protégé**), which greatly simplifies the process of modeling. An example of a rule in this format is the following:

```
s_Account_Active (?aa) ∧ e_Card_Declined (?cd) ∧
        a_Cancel (?aa,?tc) -> s_Transaction_Cancelled(?tc)
```

The intended meaning of the rule is that if an account is active in a given situation, but the system declines the card the policy prescribes execution of an action which cancels the operation and after executing this action the system will be in a new situation in which the transaction is cancelled.

4.2 Types of Heuristics

The heuristic rules in our framework can be classified into different types depending on the structural patterns of the antecedent and the consequents. The antecedents combine conditions on situations, in which the rules are applicable, on events, which can fire them whenever happen and on threats, which need to be neutralized. The consequents combine static expressions, valid within the same situation as the antecedents, and dynamic expressions, involving situations resulting from the execution of actions. In accordance with this we can distinguish the following types of heuristic rules:

Description Rules: allow inferring additional information about the current based on purely static descriptions.
Detection Rules: allow detection of the presence of threats based on observations.
Identification Rules: used for recognition of known threats.
Classification Rules: used for classification of unknown threats into known categories.

Prediction Rules: used to analyze the potential effect of the actions executed under the influence of the threats.
Correction Rules: recommend actions in response to events and/or detected threats.

The rule classification plays an important role on workflow level of the framework, where it guides the navigation through the intelligence graphs.

4.3 Examples of Heuristic Rules

Let's consider an example of purely logical predictive analytics. The starting situation is "user not logged in". In this situation the following rule applies:

```
s_Not_Logged_In(?nli) ∧ e_Logging_In(?l) ∧ a_Login(?nli, ?li) ->
        s_Logged_In(?li)
```

When the event "logging in" happens in this situation, it triggers "login action" which results in a situation "user logged in". Considering that this situation is characterized by property "session" we can derive the new session as an effect. In the next leg of the journey we can proceed with crediting the account using the following rule

```
e_Crediting_Account(?ac) ∧ s_Pay_In_Cash(?pic) ∧
        a_PayIntoAccount(?pic, ?bic) -> s_Balance_In_Credit(?bic)
```

Now the triggering event is "crediting account" and the initial situation is "paying in by cash" with parameter "account". The action which follows is "paying into account". The account can be paid in via the property "amount" of the initial situation and the new situation is "balance in credit". However, in some situations there is a possibility of "declining transaction".

```
s_Account_In_Overdraft(?ao) ∧ e_Card_Refused(?td) ∧
        a_Decline_Transaction(?ao,?cr)->s_Transaction_Cancelled(?cr)
```

The initial situation is "account in overdraft" which has property "overdraft fee". The abnormal situation "transaction cancelled" can happen as a result of obstructive events, in this case executing action "decline transaction". The irregular event which fires the rule is "card refused". The cause for the event is probably that the overdraft is not payed.

The heuristic rules allow us to specify the security policies by adding threats in the conditions of the rules and prescribing corrective actions. The conditions on situations, events and threats in the rules refer to their attributes within the ontology, while the action parameters will be determined by the properties of their domains and ranges.

5 Workflow Level and Intelligence Graphs

5.1 Transaction Flow as a Graph

On workflow level the framework works as a graph traversal. The intelligence graph is built using situations, events and threats as nodes and actions and rioles as edges. The traversal is performed within the simulation loop as a navigation from an initial towards the final situation. Along the path happen asynchronous events, which do not change the situations, and synchronous actions, which change them. As an illustration Fig. 2 shows the intelligence graph of one particular scenario, **Logging under threat**.

5.2 Framework Validation

The framework described above has been validated by modeling a number of scenarios for executing banking transactions under security threats. Table 2 contains brief description of the most typical scenarios used for validation of the framework.

Table 1 Scenarios for Validation of Transactions under Threats

Scenario	Starting Situation	Threat(s)	Threat Present In	Final Situation	Potential Deadlock
Update Antivirus with Spyware	S0_Antivirus_Not_ Updated	Spyware Baiting	S2_Infected_Attachment S23_Infected_Software	S5_Normal_State	
Login with Spyware	S0_Browser_Started	Spyware	S0_Browser_Started	S5_User_Logged_In	S6_Login_Refused S45_Credentials_Stolen
Money Transfer with Spyware	S0_Browser_Started	Spyware	S13_Infected_With_Malware	S10_User_Logged_Out	S17_Site_Maintenance S18_Disconnected …
Balance with DDoS	S0_User_Logged_In	DDoS Attack	S0_User_Logged_In	S4_User_Logged_Out	S10_Operation_Cancelled
View Balance with Quid Pro Quo	S0_Browser_Started	Quid Pro Quo	S12_IT_Support_Imitated _By_Hacker	S8_User_Logged_Out	S25_Site_Maintenance S16_Disconnected S22_Invalid_Credentials
Vishing & SMishing	S0_Normal_State	Vishing Smishing	S0_Normal_State	S5_Payment_To_Criminals	S8_Transaction_Cancelled
Withdrawal with Session Hijacking	S0_Card_Inserted	Session Hijacking	S2_User_Authenticated	S5_Card_Removed	
Sending Email Spam	S0_Machine_Overtaken _By_Hacker	Email Spam	S0_Machine_Overtaken _By_Hacker	S6_Spam_Received	S2_System_Monitored
Email Spam Received	S0_Spam_Received	Email Spam	S0_Spam_Received	S9_Machine_Overtak _By_Hacker	S5_Vital_Data_Extorted
Cross Channel with Pretexting	S0_IT_Support_ Imitated_By_Hacker	Pretexting	S0_IT_Support_Imitated _By_Hacker	S5_Payment_To_Criminals	S8_Transaction_Canc S9_Account_Locked S10_Credentials_Stol
Scareware & Rogue	S0_Normal_State	Scareware, Rogue	S1_Infection_Simulated	S6_Machine_Overtaken _By_Hacker	
ATM Infected	S0_ATM_OS_Not_Updat	ATM Infected	S0_ATM_OS_Not_Updated	S10_Pament_To_Criminals	S9_Transaction_Cancelled

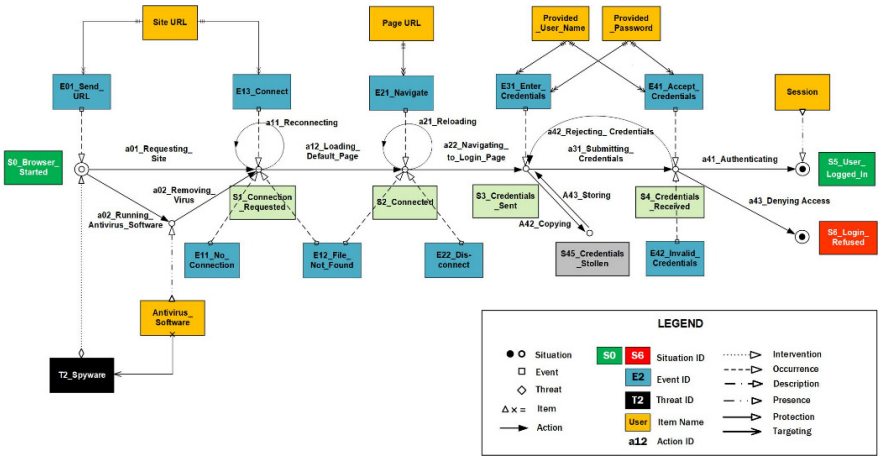

Fig. 2 Intelligence Graph for Transaction under Threat

6 Process Level, Implementation and Future Work

The general software architecture of the prototype implementation is shown on Fig. 3. The core of the system is written on C++ and operates through the **DASHBOARD**, which coordinates the work in all modes (Fig. 4). The **SIMULATOR** executes the simulation loop, which builds dynamically the graph from an initial situation, while two other components - **DATA ANALYZER** and **LOGICAL ANALIZER** - run machine learning and rule-based inference algorithms on demand. Currently the logical analysis executes Java programs locally, while the data analysis runs Python scripts within Docker containers on a private Kubernetes cloud remotely.

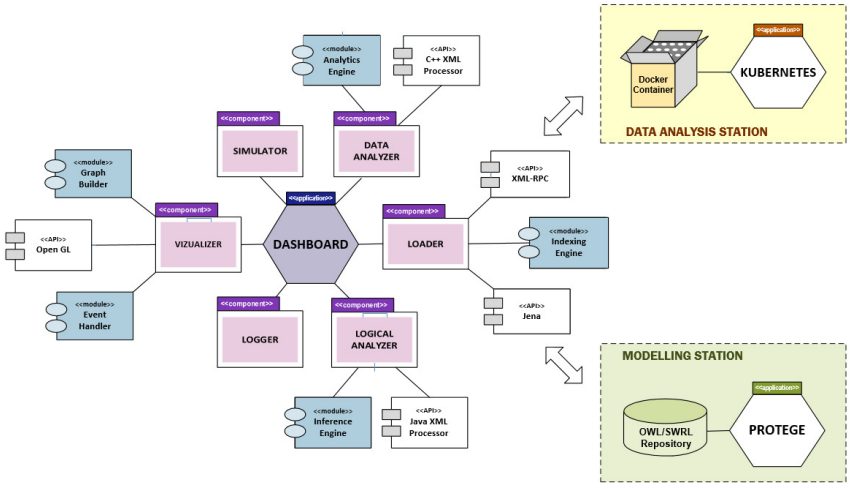

Fig. 3 Software Architecture of the Implementation

Initially we used DROOLS to process the policies encoded in SWRL rules [23], but the performance showed decline with the size of the repository. We implemented our own rule-based inference engine in Java, which makes use of the index structure created offline by running the indexing engine and it greatly improved the performance.

The other major components – **VIZUALIZER**, **LOADER** and **LOGGER** – provide interfaces for interaction with the user and the external repository, and for storing the graph traversal. The **VIZUALIZER** can be used also as a rule editor as well.

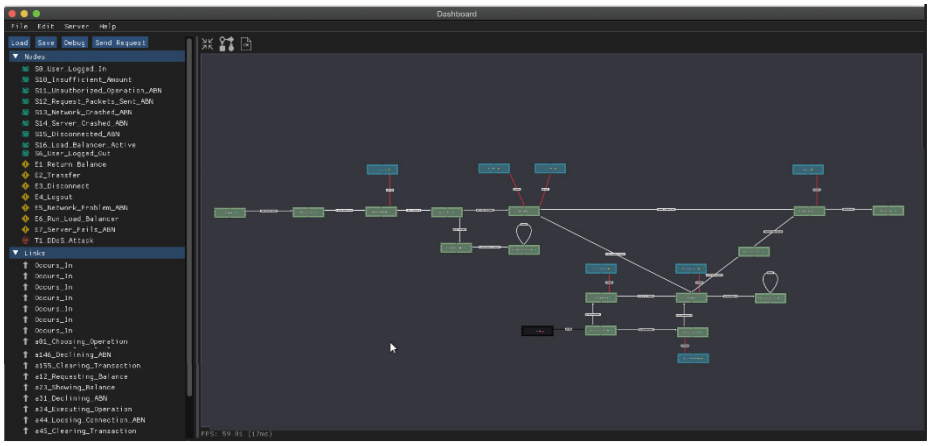

Fig. 4 Visual dashboard for simulation of the transactions under threats

The framework can be used in several modes: *modeling*, *indexing*, *validation*, *simulation* and *execution*. In simulation mode the system runs through scenario of operation under security threats, governed by the policy rules. The simulation begins from an initial situation and continues by emulating synchronous and asynchronous activities in a loop until it reaches a terminal situation. The events are triggered by external factors. The alternative actions are found by determining the applicable rules in each situation. Throughout the simulation the **LOGGER** produces a rich log of situations, events which occur, threats which are detected and actions executed upon firing applicable rules. Below is an abridged output of such a scenario which executes operation for crediting of an account after successful authentication:

```
Current situation is 's_Pay_In_Cash'
Action applicable to 's_Pay_In_Cash' is 'a_Cash_To_Be_Paid_In'
Rule applicable to 's_Pay_In_Cash' is rule 2
     resulting_situation s_Balance_In_Credit
Rule applicable to 's_Pay_In_Cash' is rule 6
     resulting_situation s_Balance_In_Credit
Event applicable to 's_Pay_In_Cash' is : e_Crediting_Account
Event applicable to 's_Pay_In_Cash' is : e_Logging_In_Unsuccessful
Rule applicable to event 'e_Crediting_Account' is rule 2
     resulting_situation s_Balance_In_Credit
Rule applicable to event 'e_Crediting_Account' is rule 6
```

The simulation can be used both for validating the policy rules as well as for obtaining insight into the threats behavior and the effect they may have on the operations.

Currently we are working on engines for analysis of *network traffic*, *clickstream data*, *event logs* and *transaction records*. Later on we are planning to implement separate engines for all major cyber defense tasks - prevention, detection, prediction, countermeasures, as well as using learning from experience for analyzing the multiple logs from execution of the same scenario.

Acknowledgements

The work on the framework has been carried out at the Cyber Security Research Centre of London Metropolitan University. It was initiated in collaboration with Lloyds Bank and won a grant from UK DCMS. The examples in the paper are solely for the purpose of illustration and do not use any internal data from the bank. Any concepts, ideas and opinions formulated by the authors in this article are not associated with the current security practices of Lloyds Banking Group.

References

[1] Nearly, J.: 75% of banks were unprepared for cyber attacks in 2018 (2019), https://www.teiss.co.uk/threats/banks-cyber-threat-2018/, last accessed 2019/10/27.

[2] Marous, J.: Technology Giants Pose Major Threat to Banking Industry. *The Financial Brand* (2019), last accessed 2019/10/27.

[3] Acunetix: Logical and Technical Vulnerabilities – What they are and how can they be detected? (2019), https://www.acunetix.com, last accessed: 2019/10/27.

[4] Netsparker: Understanding the Differences Between Technical and Logical Web Application Vulnerabilities (2019), https://www.netsparker.com/blog/web-security/logical-vs-technical-web-application-vulnerabilities/, last accessed: 2019/10/27.

[5] Intruder Systems, A proactive vulnerability scanner, for your external infrastructure, 2019, https://intruder.io, last accessed: 2019/06/30.

[6] Greenbone Networks, OpenVAS - Open Vulnerability Assessment System, 2019, http://www.openvas.org/, last accessed: 2019/07/01.

[7] Rapid7, Nexpose. Your on-prem vulnerability scanner, 2019, https://www.rapid7.com, last accessed: 2019/07/01.

[8] InfoSight, Network & Cyber Security Services, 2016, https://www.infosightinc.com/solutions/it-security-services/network-security.php, last accessed: 2019/06/29.

[9] Kenna Security, 2018, https://www.kennasecurity.com, last accessed: 2019/06/29.

[10] Coalfire, Cyber Risk Services, https://www.coalfire.com, last accessed 2019/04/26.

[11] Vigilant Software: vsRisk Cloud – Cyber risk assessments made simple (2019), https://www.vigilantsoftware.co.uk/topic/vs-risk, last accessed: 2019/10/27.

[12] ABB: System 800xA Cyber Security - Maximizing cyber security in process automation, https://new.abb.com/control-systems; last accessed: 2019/10/27.

[13] Google: CSP Evaluator, https://csp-evaluator.withgoogle.com/, last accessed: 2019/10/27.

[14] Threatmodeler: The Evolution of Threat Modeling (2016), https://threatmodeler.com/evolution-of-threat-modeling/, last accessed: 2019/10/27.

[15] Blokdyk, G.: Threat modelling, 2nd ed., 5STARCooks (2018); ISBN: 0655196072.

[16] Bataityte, K., Vassilev, V. and Gill, O.: Ontological Considerations of Situations and Actions in Description Logic as a Basis for Vulnerability Analysis (2019), to appear.

[17] Allemang, D., Hendler, J., Semantic Web for the Working Ontologist, MK (2011).

[18] McGuinness, D. and Van Harmelen, F. (eds.): OWL Web Ontology Language (2004), https://www.w3.org/OWL/, last accessed 2019/04/23.

[19] Horrocks, I., Patel-Schneider, P. et al. (eds.): SWRL - A Semantic Web Rule Language (2004), https://www.w3.org/Submission/SWRL/; last accessed 2019/04/23.

[20] Herzog, A., Shahmehri, N. and Duma, C.: An Ontology of Information Security. Int. J. Inf. Security and Privacy 1/4 (2007), pp. 1-23.

[21] Souag, A., Salinesi, C. and Wattiau, I.: Ontologies for Security Requirements, In: Proc. Int. Conf. on Advanced Information Systems Engineering CAISE2010 (2010), pp. 61-69.

[22] Iannacone, M., Bohn, S., Nakamura, G. et. al.: Developing an Ontology for Cyber Security Knowledge Graphs. Proc. ACM CISR'15 (2015), pp. 12:1-12:4.

[23] Red Hat, Inc., Drools (overview), https://www.drools.org/; last accessed 2019/03/11.