Phishing Knowledge based User Modelling in Software Design

Linfeng Li[1], Timo Nummenmaa[2], Eleni Berki[2], Marko Helenius[3]

[1] Beijing Institute of Petrochemical Technology, Beijing, China, 48222692@qq.com
[2]University of Tampere, Tampere, Finland, {firstname.lastname}@uta.fi
[3] Tampere University of Technology, Tampere, Finland, marko.t.helenius@tut.fi

Abstract. Due to the limitations of anti-phishing software and limitations in creating such software, we propose the usage of metamodelling frameworks and software tools for implementing software systems where phishing prevention is already designed as a part of the system itself. An expressive computational, verifiable and validatable metamodel is created that captures user behaviour. Next it is shown through examples that the metamodel follows and describes reported phishing scams accurately. The model is then used to create specification in an executable formal specification tool. The formal specification, which can be executed to observe user behaviour, can be used as a building block in the specification of a larger software system, resulting in an inherently phishing-resilient software system design in the form of a formal specification.

Keywords: Phishing, Metamodelling, Formal Methods, Software Design

1 Introduction

Though a variety of anti-phishing technologies have been used against online identity theft (commonly known as phishing), phishers (online identity thieves) had never been discouraged. On the contrary, the phishing attacks become more advanced and more sophisticating [1], utilizing knowledge and expertise from socio-technical and cognitive domains. Technical anti-phishing solutions e.g. static phishing preventions, based on a pre-defined white list or black list [1,2,3] and dynamic anti-phishing appli-

cations follow reactive or preventive maintenance principles [4,5,6] that are not adequate for users' protection. These prevention methods are not adaptable enough to meet users' personal anti-phishing/spam demands and software designers' requirements [1]. To discover the user requirements for anti-phishing software, many user-centered research studies have been carried out, by Zhang et al. [7], Wu et al.[8], Jakobsson and Ratkiewicz [9], Li [1] and other, considering usability and security together. These studies did not, however, result in a reliable, abstract and general model, practical for the designers of anti-phishing software. In this paper, we approach the problem by not creating anti-phishing software, but by creating a metamodel of phishing. This metamodel can help to design software systems using a formal specification system. In so doing, phishing avoidance is already considered in the design of the software. Thus our research question is: *How could user behavior in phishing context be computationally metamodelled so that the resulted metamodel can be used as a basis for a formal design model of a software system?*

Traditional metamodelling frameworks and tools [10,11] cannot be useful here, because the nature of this research requires to model and simplify the various users' requirements and SW needs, instead of constructing metamodels based on the collected requirements. Hence, in this research work, the authors apply an enriched version of the finite state machine (FSM) model, which is a straightforward and exact computational modelling methodology to describe system processes. The rest of the paper is organized as follows: Related research approaches and studies in anti-phishing and phishing modelling are briefly reviewed. Next, two real phishing scams are presented. An FSM model of user behavior is presented and exemplified through the real phishing cases. The FSM model is then used as the basis for the model of the user in a specification created in the DisCo [12] formal specification language, showing how the FSM can be utilized in software system design. Concluding, the authors summarise the strengths and future potential of this computational metamodelling of user behaviour from the software quality point of view.

2 Research Rationale and Related Work

There are no formal models or metamodels for understanding phishing activities and phishers' and corresponding victims' behaviours online; such model's construction is not a simple task. From the software quality point of view, such a model should be adequately expressive and rich in modeling human activity in detail on the

one hand and sufficiently generic and abstract on the other hand. The latter are needed in order to communicate and test specialization and/or generalization details when needed by and for the various interested groups in the software design of anti-phishing technologies. Moreover, the correctness and consistency of the phishing actions will certainly influence the quality of the resulted anti-phishing prevention software. In any case, the phishing situations are hard for software designers to model, understand and grasp in details unless a modeling solution fits to their cognitive needs. Hence, it is imperative to search for a more abstract, formal point of view and a higher level of modeling for all software development stakeholder groups and especially for those of software users and software designers. Our assumption, which we aim to prove though our ongoing research, is that developers wishing to create phishing-proof software, should employ formal metamodelling techniques to construct dynamic models (like the dynamic nature of phishing) that assist in design.

The following models and notations employ formality and other quality criteria. They are related to but also very different approaches from our work.

Chandrasekaran et al. [13] conducted a pilot project to detect phishing emails based on a set of characteristics of phishing emails. The results showed that the authors' methodology performed with 95% accuracy rate and low false positive rate [13]. However, phishing scams launched from other communication channels, e.g. online social media are still not adequately addressed in the use of this methodology.

Shahriar and Zulkernine [14] introduced a method to model and examine phishing web pages by using FSM theory. By testing the behaviour of given web pages' response, the method brought to them the way to identify phishing pages. However, phishers can easily circumvent it by using embedded objects and malformed web pages, continuing playing games with different roles and versatile game rules.

Akhawe and other [15] presented a formal model of web security based on an abstraction of the web platform and used this model to analyze the security of several sample web mechanisms and applications., Nonetheless, this model is still lacking in considering human factors. For example, how well users understand web communications and the risks of misusing phishing preventions are still questionable.

Kumaraguru et al. [16] attempted to model online trust in their research on trust and security with six components. The aim of the research was to create tools and training modules to help online users make correct decisions about trust.

Dong et al. [17] elaborated on why users form false perceptions and fail to discover mismatches between authentic and phishing web services. They mentioned four rea-

sons: insufficient information, misinterpretation, inaccurate/incomplete expectation and perception ability drop and the users' carelessness when taking the next planned action. Despite of adequate modeling of the user-phishing interactions, this modeling approach is still too general to assist in the design of efficient phishing prevention.

3 Example Reported Phishing Attack and User Behaviors in the Phishing Attack

Most research projects define phishing attacks as cybercrime. This type of crime aims at deceiving victims through fraud web pages in order to steal their personal information. Jacobs describes phishing as a marriage between technology and social engineering [9]. In his definition, the extent of phishing is (i) widened and (ii) not Internet-restricted. Thus more scenarios could be included, where phishers take advantage of other means than the Internet, web browsers or emails communication channels, e.g. surface mail. In order to build a comprehensive and reliable model, the authors collected phishing reports based on the analysis of abused by phishers communication channels and tools. Next, we expose two real-life examples from our data collection.

3.1 Reported phishing attack examples

Attack1: Man-in-the-middle attack

In September 2011 two banks in Finland, Osuuspankki and Nordea experienced a man-in-the-middle attack [18] in which there was no need to install malware to victims' computers. In this attack victims received an email that contained a link to a malicious, but real-looking web site. There was no certificate and the web site address was false, resembling though the original address. If credentials, including a one-time password, were given to the fraudulent site the victim was presented with a fake message, which instructed to wait for two minutes. During that time the server had time to use the credentials and login to the real service for preparing the money transfer. Before this could be accomplished a final one-time verification code was needed. That was asked after the delay and the money could be, thereafter, directly transferred [19].

Attack2: A sophisticated Trojan horse program

There were three banks in Finland attacked: Nordea, Sampo Bank and OP-Pohjola in January 2012. Losses were reported at least from OP-Pohjola. In Nordea e-Bank

money transfers were cancelled because verifications were sent to the victims by SMS about suspicious money transfers [19].

In this case, a sophisticated Trojan horse program (later on called as “Trojan”) was installed on the users’ computers. The Trojan was created by a Zeus toolkit program [20]. There are several possibilities on how a user might have gotten the Trojan installed on his computer:

1. Email that contains the Trojan or link to the Trojan.
2. A malicious or non-malicious source (e.g. web site, memory stick or other media) from which the user installed the Trojan without knowing its true nature.
3. A malicious or non-malicious source (e.g. web site, memory stick or other media) that the user used and vulnerability in the user’s computer was used for infection.
4. Other vulnerabilities that left the user’s computer open to Trojan injection.

It should be noticed that possibilities 1, 3 and 4 have two basic possibilities: either a user does something actively that causes infection or the infection occurs automatically. However, active doing is more probable in possibilities 1 and 3 than in 4.

After the Trojan is installed it activates when a user logs into her/his e-bank. This time a user just needs to pay a bill and at that exact moment the Trojan intercepts the session by changing the bank account and the sum from the transfer. Because a user gives all the credentials authorizing money transfer, the Trojan hiding in the computer has access to the credential information and can change the transfer accordingly. What makes this even more difficult to observe is that the Trojan is able to hijack the browser session and show incorrect account balance.

The authors dare to ask the question: Are the anti-phishing technologies unreasonably ineffective? Why did not we become more secure from all the security technologies that have been deployed against phishing attacks? The authors are also tempted to answer that in terms of metacognitive design of secure software, the right computational metamodels had never been completely realised. As computers and the Internet pervaded all areas of social life, communication and production, computationally and cognitively effective metamodels are increasingly required.

4 Computational Modelling of User Behavior

A finite state machine or finite automaton is a formal, computational model to describe the states in a process and the transition functions among the states. In formal

definition, a FSM consists of 5-tuple elements (Q, Σ, δ, q_0, F), which accordingly represent the state set (Q), transition alphabet of inputs (Σ), transition function (δ), the start state (q_0), and the set of final (accept) states (F) [10, 11, 21].

The Labelled Transition System Analyser (LTSA) [22] is a verification tool for concurrent systems. It is scriptable and able to generate FSMs based on the scripts. It is also possible to automatically analyze the model and detect errors and deadlocks. In this section, we present our LTSA script, and introduce our model complying with the formal definition of FSM theory.

According to the examples of phishing attacks presented earlier, we have designed the LTSA script in Table 1. In the FSM notation, the model starts from the composing initial state, which means end users' behaviours begin to take place. That is, the user behaviour is to unfold or process according to the state-transition (processing) functions defined in the FSM when a phishing or non-phishing message is composed. When a user receives a message, the user starts processing the message, which means a user needs to either learn from the message or take further actions upon receiving the message.

Table 1. LTSA script to model user behavior in phishing context

```
COMPOSING =
    (messageReceived->INFORMATION),
INFORMATION =
    (messageRead->PROCESSING),
PROCESSING=
    (learnFromMessage ->ACQUIRING
    |takeActionUponMessage -> DECISIONMAKING),
ACQUIRING =
    (understandMessage->COMPOSING
    |misunderstandMessage->COMPOSING),
KNOWLEDGE =
    (knowledgeMatchinformation.normalInfo
     ->BELIEVE
     |knowledgeNotmatchinformation.detectedFraud
     ->NOTBELIEVE
     |knowledgeNotmatchinformation.checkingMoreinfo
     ->INFORMATION
     |knowledgeNotmatchinformation.misled
     ->DECISIONMAKING),
MISKNOWLEDGE =
    (misknowledgeMatchinformation.undetectedFraud
     ->BELIEVE
     |misknowledgeNotmatchinformation.obstinacy
     ->NOTBELIEVE
     |misknowledgeNotmatchinformation.
     misknowledgeCorrected->DECISIONMAKING),
DECISIONMAKING =
     (informedDecision->KNOWLEDGE
     |uninformedDecision->MISKNOWLEDGE
     |nothingDecided->ACQUIRING),
BELIEVE =
    (decisionMade->COMPOSING),
NOTBELIEVE =
    (decisionMade->COMPOSING).
```

When the user learns from the message, the information in the message could be correctly understood or misunderstood. Correct understandings result in the user's advanced knowledge, while any misunderstandings finally end up with misknowledge in mind. Both knowledge and misknowledge affect the user's decisions later on. If, in the message, the user needs to take further actions upon the message, s/he will make an informed decision, uninformed decision, or nothing will be decided. As soon as a user makes a final decision on the given message or processes/learns the

information in the received message, the FSM is reset to the initial state to wait for the next piece of composed message either from innocent senders or phishers.

Informed decision represents that the user applies the knowledge to check against the information mentioned in the received message. When the user makes an uninformed decision, misknowledge is used; and 'nothing decided' means there is no knowledge or misknowledge in the mind to take action upon the information in the received message. In the nothing-decided case, the user has to look for more related information from other sources in order to make a proper decision later on.

When knowledge is applied, it does not necessarily mean that the user made a correct final decision. For example, the phishing information in the received message could be so persuasive that the user is misled by the phishing content to make another decision, even though there is an explicit conflict between the user's knowledge and the phishing information. At this point, it is highly likely that the user will be misled and deceived by the phishing information. When the user is determined on the knowledge, the phishing information can be detected, and no fraud can succeed.

When misknowledge is taken into use, the phishing scam may succeed, especially when the phishing information in the received message matches the mis-knowledge in the user's mind. In the FSM model, this transition is called *misknowledgMatchinformation.undetectedFraud*, and this transition goes to the state named BELIEVE; that is the user believes in the phishing information. When a piece of innocent information showing in the received message mismatches the user's mis-knowledge, the user could either be persistent to stick on the misknowledge or be sensible enough to actively correct the misknowledge and re-make the decision. The transition alphabet is listed in the Table 2 and the state-transition functions and inputs among the states are listed in the Table 3.

Table 2. States of the FSM

States	Explanation	States	Explanation
q_0	A message is composed	q_5	Corresponding uninformed decision is made
q_1	A message is received	q_6	Not believing the given information
q_2	Processing information	q_7	Believing the given information
q_3	Making decision	q_8	Corresponding informed decision is made
q_4	Acquiring information		

Table 3. The state transition and the inputs of the FSM

States transitions	Input	States transitions	Input
$->q_0$	A message is composed	$q_8->q_3$	knowledgeNotmatchinformation.misled
$q_0->q_1$	ReceivedMessage	$q_5->q_6$	misknowledgeNotmatchinformation.obstinacy
$q_1->q_2$	messageRead	$q_5->q_7$	misknowledgeMatchinformation.undetectedFraud

q_2->q_3	takeActionUponMessage	q_6->q_0	decisionMade
q_2->q_4	learnFromMessage	q_7->q_0	decisionMade
q_3->q_4	nothingDecided	q_8->q_7	knowledgeMatchinformation.normalInfo
q_3->q_5	uninformedDecision	q_8->q_6	knowledgeNotmatchinformation.detectedFraud
q_3->q_8	informedDecision	q_8->q_1	knowledgeNotmatchinformation.checkingMoreinfo
q_4->q_0	understandMessage, misunderstandMessage	q_5->q_3	misknowledgeNotmatchinformation .misknowledgeCorrected

In order to verify the FSM model that we elaborated earlier, we explain the phishing example in the FSM. Taking advantage of the animation feature in LTSA, we were able to produce a sequence of interactions in each phishing context. Examples of these traces are presented next.

Attack1: When a victim receives an email containing a link to a malicious, but real-looking web site, and decides to give credentials to the phishing web site (Verification 1-1):

messageReceived ->messageRead ->takeActionUponMessage->
uninformedDecision-> misknowledgeMatchinformation.undetectedFraud
->decisionMade

In this case, the misknowledge is that the given link in the email is mis-understood as an authentic and trustworthy web site. Therefore, the victim decides to hand out the credentials to the web site.

When a victim receives an email containing a link to a malicious real-looking web site, the victim may suspect the authenticity of the visited web site (Verification 1-2):

messageReceived -> messageRead ->takeActionUponMessage->informedDecision
->knowledgeNotmatchinformation.checkingMoreinfo

In this case, the victim does not use the same misknowledge as the one above. Instead, the victim checks more information to verify the authenticity of the displayed web page. If verifying the authenticity of the web page, the possible victim's behavior could be e.g. acquiring new information from various sources.

Attack 2: The four possibilities of users' behaviors to get Trojan horse program installed could be abstracted into two types, installation with users' awareness and permission and installation with no users' awareness. The two types of installation decisions could be described as below:

1. Installation with users' awareness and permission (Verification 2-1):

   messageReceived -> messageRead -> takeActionUponMessage
   ->uninformedDecision ->misknowledgeMatchinformation.undetectedFraud
   -> decisionMade

2. Installation with no users' awareness (Verification 2-2):

   messageReceived -> messageRead -> takeActionUponMessage

-> nothingDecided -> misunderstandMessage

After the Trojan horse program is installed, the following user behaviors are committed under the mis-understanding that the communication traffic is secure and no one else has the access to the credentials or monitor the traffics. In this case, the victim's behavior could be explained like this (Verification 2-3):

messageReceived -> messageRead -> takeActionUponMessage
->uninformedDecision ->misknowledgeMatchinformation.undetectedFraud
->decisionMade

5 Modeling Software: Usage of the FSM Model

The presented FSM model depicts the behavior of a user in a phishing situation. This kind of knowledge can be utilized in various ways, as long as it is understood. The FSM model developed with LTSA is aimed to be easy to understand and use because of its simplicity, focus on state transitions instead of states and the possibility for animating those transitions to help in understanding the model.

To demonstrate how the FSM model can be used in software development, the authors created an executable specification of dynamic users in a dynamic environment in the DisCo language. DisCo is an action based executable formal specification software package for modelling reactive and distributed systems [23]. It has been updated to include support for probabilistic modelling [24,25,26]. Models created with the system can be instantiated with different system states and can be animated in a graphical animator. The goal of specifications created with the system is that they can later be implemented as software systems.

While DisCo is a modelling system itself (as LTSA is), it is also different from LTSA because DisCo models stay manageable in larger sizes than LTSA models. DisCo models also facilitate larger system views and not just state transitions. On the other hand, DisCo does not support similar automatic analysis of models as LTSA does. Because of the way the LTSA model has been written in the LTSA language, that is to focus on state transitions and not actual states, the specification can easily be converted into an action based DisCo specification. In fact, the conversion could even, at least partially, be done automatically. The converted information can be used as part of a larger specification.

Models created with DisCo can support multiple independent objects that can all have states of their own. Thus, it is possible to model larger multi-user phishing at-

tacks and other scenarios that resemble actual real world scenarios. DisCo can be used to analyze how objects following the state transitions given in an LTSA model act in those scenarios.

The model depicts a world view where several persons and information sources exist. Each person acts based on the user behavior model in our FSM, but their decisions at the points where multiple choices are possible, are influenced by both a knowledge level variable and a misknowledge variable that are updated when the persons learn correct or incorrect information. Each information source has a different value for how influential it is, which results in users earning different knowledge and misknowledge depending on the information sources they access.

The specification is implemented by specifying classes and actions. The classes of the specification are presented in Table 4 and the actions are presented in Table 5. The classes are instantiated to be objects in the specification system. The actions are largely based on transitions in the LTSA model and are what alters the state of the system. Each action has a guard that determines whether the action is enabled and can be executed. The guard is a boolean expression that must evaluate true for the action to be enabled. Because DisCo is an action-based system, objects take part in actions but the action chooses its participants through the guard expression.

Table 4. DisCo specification classes

Class	Explanation
InformationSource	A source of information for a person.
Person	A person who can be the target of phishing.

Table 5. DisCo Specification actions

Action	Explanation	Action	Explanation
receiveMessage	A source of information for a person.	takeAction	A person starts to take action.
learn	A person starts learning.	canNotDecide	A person cannot make a decision.
understand	A person understands and its knowledge value is increased.	Misunderstand	A person understands and its misknowledge value is increased.
makeUninformed Decision	A person starts making an uninformed decision.	makeInformed Decision	A person starts making an informed decision.
checkMoreInfo	A person starts reading more information.	Correct-Misknowledge	A person starts taking action and its misknowledge is lowered.
mislead	A person is misled and starts taking action.	detectFraudDecision	A person detects a fraud decision.
normalInfo-Decision	A person makes a decision.	undetect-edFraudDecision	A person makes a decision but cannot detect fraud.
obstinacyDecision	Makes a stubborn decision.	updateSkip-Values	Update values that dictate the probability of a person understanding, misunderstanding, and making informed and uninformed decisions

An example action can be seen in Table 6. The class *Person* (Table 7) is a representation of a person, who can access information that might include phishing mes-

sages. Person has a state variable, which is based on the states in the LTSA model. The state variable is used to control the state of each individual Person instance. The person has a variable for mis-knowledge and knowledge, so it is possible to control how knowledgeable that Person object is. These values are changed through certain actions such as 'mis-understand'. *Person* also has skip variables which enable instances of *Person* to decline taking part in actions. For example, an instance of *Person* may decline to take part in 'understand' if that instance's misknowledge is much higher than knowledge. The *informationSource* class (Table 8), is a generic information source that a person may access. It only has one variable, *informationValue*, which means how influential the specific source of information is.

To test the DisCo model, a creation was instantiated. The creation is a view of the state of a system in DisCo and is instantiated based on the specified classes. In this case, two instances of *Person* were instantiated (Person_1 and Person_2) along with two instances of *informationSource* (InformationSource_1 and InformationSource_2). Person_1 started with knowledge and misknowledge values of 60, while Person_2 started with both values at 30. The *informationValue* of InformationSource_1 is 20, while the *informationValue* of InformationSource_2 is 10.

Table 6. receiveMessage action

```
action receiveMessage (p : Person; isource: Infor-
mationSource) is
when (p.updateSkips = false and p.personState'idle)
do
        p.personState->reading()||
        p.currentIs:=isource; end;
```

Table 7. Person class

```
class Person is  personState :
(idle,reading,read,learning,taking_action,indecisive,
making_informed_decision,making_uninformed_decision);
skipPossibilitymisunderstand : integer;
skipPossibilityunderstand : integer;
skipPossibilitymakeUninformedDecision : integer;
skipPossibilitymakeInformedDecision : integer;
skipPossibilitycanNotDecide : integer;
updateSkips : boolean; knowledge : integer;
misknowledge : integer;
currentIs : reference InformationSource;  end;
```

Table 8. InformationSource class

```
class InformationSource is
informationValue : integer;end;
```

There were 100 execution steps, resulting in a state where the knowledge of Person_1 was 130 while mis-knowledge was 90. The knowledge of Person_2 was 150 and misknowledge 10. In this execution, Person_2 had become more knowledgeable and shed off some misknowledge, while Person_1 still had a lot of misknowledge, even while learning new knowledge.

Table 9 shows a trace of the first 20 actions that were executed. Traces of executed actions can be used to analyze behavior within a scenario. Table 9 divides each action into the name of the action and the participant objects that took part in the action to give a clear picture of what has happened in each step. Among the things we can dis-

cover from this trace, is when a Person is part of an undesirable action and why that happened.

Table 9. Execution trace of the first 20 actions executed

Step	Action	Participants	Step	Action	Participants
1	receiveMessage	Person_2, InformationSource_1	11	makeInformedDecision	Person_2
2	takeAction	Person_2	12	detectFraudDecision	Person_2
3	makeUninformedDecision	Person_2	13	receiveMessage	Person_2, InformationSource_2
4	undetectedFraudDecision	Person_2	14	takeAction	Person_2
5	receiveMessage	Person_1, InformationSource_2	15	makeUninformedDecision	Person_2
6	takeAction	Person_1	16	receiveMessage	Person_1, InformationSource_1
7	makeUninformedDecision	Person_1	17	obstinacyDecision	Person_2
8	receiveMessage	Person_2, InformationSource_2	18	receiveMessage	Person_2, InformationSource_1
9	undetectedFraudDecision	Person_1	19	takeAction	Person_1
10	takeAction	Person_2	20	takeAction	Person_2

We can see that in step 4, Person_2 was a participant in the *undetectedFraudDecision* action which is considered the worst case scenario in which a phishing attempt is successful. Person_2 became a part of that action by first receiving a message from InformationSource_1, then by taking action and making an uninformed decision. Person_1 also takes part in the *undetectedFraudDecision* action in step 9.

As DisCo is built to support incremental development of specifications through a layer system, the presented model can easily be used as a building block of a software system. A specification created in this way would enable the designer to verify that the software design takes into account how a user behaves in phishing situations. Also, as a DisCo specification can include outside influences, phishers could be modeled separately as classes in a specification to gain a better understanding of possible phishing situations regarding the system in design.

6 Conclusions and Future Research

In contrast to specific reactive anti-phishing solutions, the method proposed in this paper focuses on the analysis and modelling of user behaviour to facilitate the proactive design of phishing-resistant systems. First, user behaviour in a phishing context is modelled with FSM notation. The FSM-based model provides a general and abstract view to describe user behavioural patterns in phishing context. This model can be a helpful guide to evaluate the design quality of anti-phishing and phishing-resistant systems at early design stages. The model can be verified, both with automatic meth-

ods, and through real cases testing with it. The FSM model can be used as a basis for a specification of a full software system using a tool such as DisCo. That model can take into account important security aspects found in the abstract FSM model. Our method is a series of transitions, first transitioning from the FSM model, to a DisCo implementation, then to a more refined DisCo implementation of a software system through a layer system, and finally to a software implementation. The refined DisCo implementation and software implementation are left as future research.

Rich and multipurposed, testable, computational and easily implementable metamodels that are expressible and extendable (yet, by no means complete) can assist: i) software designers and developers to realize the dynamic design of security auditing systems by considering a great amount of static and dynamic situations; ii) towards an improved understanding and empowerment with knowledge of the digitally competent person to protect his/her own personal data and take appropriate security decisions; iii) software sponsors to decide on cost-effectiveness by checking and testing for security in early design steps. This research concentrates on improving security through preventive, and not corrective or adaptive, systems maintenance. For this reason cost-effectiveness is a quality property that can also be achieved with our suggested phishing prevention solutions.

Although the advantages of this metacognitive and computational FSM-based metamodel demonstrate that the model is an efficient analytical tool with great design and implementation potential, it still requires certain efforts on metamodelling. Arguably, limitations may hinder the usage of the enriched FSM metamodel, but this necessitates the investment of efforts and resources on model-driven testing and corresponding automated tools at the early stages of the anti-phishing software lifecycle.

Naturally, general considerations that will be tackled in future research are: i) the scalability of the approach, since phishing typically affects large groups of people. DisCo simulations can simulate groups of people, but it really should not be important to model a million individuals in an abstract simulation; the focus is elsewhere. ii) The FSM described in the paper consists of very general transitions. Nonetheless the knowledge we hope to obtain from analysing its usage is that we can see that the transitions conform to the reality and then we can integrate those transitions to a realistic SW design. We can, then, analyse that design which includes the transitions. iii) In the case of phishing, and due to its dynamic (not static) nature, unavoidably people become more or less informed over time; there should always be ways to distinguish

and model between the user making an informed or an uninformed decision. However, this is, by and large, a socio-cognitive issue.

References

1. Li, L.: A Contingency Framework to Assure the User-Centered Quality and to Support the Design of Anti-Phishing Software, PhD. Dissertation, University of Tampere (2013).
2. SpamAssassin: The Apache SpamAssassin Project, http://spamassassin.apache.org/, retrieved on 12th Aug. 2015.
3. Sun, B., Wen, Q., Liang, X.: A DNS Based Anti-phishing Approach. In: 2010 Second International Conference on Networks Security Wireless Communications and Trusted Computing (NSWCTC), pp. 262-265 (2010).
4. Atighetchi, M., Pal, P.: Attribute-Based Prevention of Phishing Attacks. In: Eighth IEEE International Symposium on Network Computing and Applications, pp. 266-269 (2009).
5. GMAIL: How is spam handled, http://support.google.com/mail/bin/answer.py?hl=en&answer=78759, retrieved on 11th Dec., 2014
6. SpamAssassin: DNS Blocklists, http://wiki.apache.org/spamassassin/DnsBlocklists, retrieved on 2nd May 2015.
7. Zhang, Y., Egelman, S., Cranor, L., Hong, J.: Phinding Phish: Evaluating Anti-Phishing Tools. In: the 14th Annual Network & Distributed System Security Symposium, San Diego, CA, USA (2007).
8. Wu, M., Miller, R. C., Garfinkel, S. L.: Do Security Toolbars Actually Prevent Phishing Attacks? In Proceedings of Conference on Human Factors in Computing Systems (2006).
9. Jakobsson, M., Ratkiewicz, J.: Designing Ethical Phishing Experiments: A study of (ROT13) rOnl auction query features. In: the 15th annual World Wide Web Conference, pp. 513-522 (2006).
10. Berki, E.: Establishing a Scientific Discipline for Capturing the Entropy of Systems Process Models. CDM-FILTERS. A Computational and Dynamic Metamodel as a Flexible and Integrated Language for Testing, Expression and Re-engineering of Systems. Ph.D. Thesis. Faculty of Science, Computing and Engineering. University of North London (2001).

11. Eilenberg, S.: Automata, Languages and Machines. Academic Press. (1974).
12. DisCo: DisCo Home Page, http://disco.cs.tut.fi/, retrieved on 22nd Dec., 2014.
13. Chandrasekaran, M., Narayanan, K., Upadhyaya, S.: Phishing email detection based on structural properties, In: the NYS Cyber Security Conference (2006).
14. Shahriar, H., Zulkernine, M.: PhishTester: Automatic Testing of Phishing Atacks, In: 2010 Fourth international Conference on Secure Software Integration and Reliability Improvement, pp. 198-207 (2010).
15. Akhawe, D., Barth, A., Lam, P. E., Mitchell, J., Song D.: Towards a Formal Foundation of Web Security. In: 23rd IEEE Computer Security Foundations Symposium, pp. 290-304 (2010).
16. Kumaraguru, P., Acquisti, A., Cranor, L.: Trust modeling for online transactions: A phishing scenario. In: Privacy Security Trust, Ontario, Canada (2006).
17. Dong, X., Clark, J. A., Jacob, J.: Modelling user-phishing interaction. In: Human-System Interaction, Kraków, Poland (2008).
18. F-Secure: Man-in-the-Middle Attacks on Multiple Finnish Banks, http://www.f-secure.com/weblog/archives/00002235.html, retrieved on 30.1.2015.
19. YLE: Finnish bank accounts breached, http://www.yle.fi/uutiset/news/2012/01/finnish_bank_accounts_breached_3191383.html, retrieved on 10.4. 2015.
20. Wyke, J.: What is Zeus, technical paper, http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/what-is-zeus.aspx, retrieved on 30.4.2015.
21. Sipser, M.: Introduction to the Theory of Computation, PWS Publishing Company (1997).
22. LTSA: Labelled Transition System Analyser, http://www.doc.ic.ac.uk/ltsa/, retrieved on 10th November, 2015.
23. Aaltonen, T., Katara, M., Pitkänen, R.: DisCo toolset - the new generation. Journal of Universal Computer Science, 7, 1, 3–18 (2001).
24. Nummenmaa, T.: A method for modeling probabilistic object behaviour for simulations of formal specifications. In: NWPT 2008, Abstracts, Tallinn, Estonia, (2008).
25. Nummenmaa, T.: Adding probabilistic modeling to executable formal DisCo specifications with applications in strategy modeling in multiplayer game design. Master's thesis, University of Tampere (2008).
26. Nummenmaa T.: Executable formal specifications in game development: Design, validation and evolution. PhD thesis, University of Tampere (2013)..