Aiding In The Creation of Social Engineering Attack Prevention Tools

Saleem Manjoo
University of Cape Town
manjoosaleem@gmail.com

ABSTRACT

With the improvements made to the technological elements of security systems, the human element has become the weakest, and thus, more often targeted by attackers in what are referred to as social engineering attacks. These attacks involve the influencing of people to divulge sensitive information. This has created the need for social engineering prevention and training tools to be developed. As a part of the SEPTT project, back-end services were required to be created to aid in the development of such tools. A database was designed to store an attack detection model. A web-API was created to provide public access to the database. Finally a visualization tool was created with the aim of visualizing the attack detection model and provide a means to make changes to it. The first of the two requirements were successfully developed but the last visualization tool fell short of its requirement to allow for the functionality to make changes to the model.

1. INTRODUCTION

Information security is a fast growing discipline with the protection of personal and sensitive information being of vital importance to governments and organizations who have a vested interest in securing such information [9]. As the technological element to security systems improve, they become increasingly difficult to exploit. Thus, the target has shifted from the technological element to the human element which can be often be considered the more vulnerable of the two in the system [1]. Attacks on information security systems which target the human element are referred to as social engineering attacks.

While there are various definitions of social engineering, it can be defined as the various techniques that are used for the purpose of obtaining information by the exploitation of human vulnerabilities in order to bypass security systems [7]. Or, more simply put, it can be seen as the art of influencing people to divulge sensitive information.

Since social engineering deals with the exploitation of human vulnerabilities, we can see that there is a strong psychological aspect to it. There are various psychological vulnerabilities which humans possess that can be exploited by social engineers with the aim at influencing an individual's emotional and cognitive state to make them more susceptible to divulging sensitive information [1]. While this paper will not go into detail about the various psychological vulnerabilities, it is important to note that it is due to these vulnerabilities that the human element in security systems can be deemed the weakest and easiest to exploit.

Given that the human element of information security systems is considered to be the weak element and is thus the focus of attacks, this creates a need for the development of tools which can aid in the detection and prevention of such attacks.

This paper will first briefly provide a simplified view of how social engineering attacks are carried out as well as the research done in the detection and prevention of such attacks. It will then discuss the work done in the Social Engineering Prevention Training Tool (SEPTT) project to create such tools based on the Social Engineering Attack Detection Model(SEADM) [8], a model created to aid users detect whether a request for sensitive information is legitimate or a social engineering attack. The main focus of the paper will be on the development of services to aid in the development of future social engineering attack prevention tools.

1.1 Social Engineering Attacks

The Social Engineering Attack Framework [9] depicted in figure 1 describes the planning and flow of a full scale social engineering attack. It can be seen as the methodology used by social engineers when panning and performing an attack.

We can see 6 core phases depicted in this framework:

1. Attack Formulation: The goal and the target of the specific attack is identified.
2. Information Gathering: All sources of information on both the goal and the target are identified.
3. Preparation: All information gathered is combined and an attack vector is developed. All elements of the ontological model can be identified in this phase.
4. Develop Relationship: The attacker establishes communication with the target and a trust relationship is built.
5. Exploit Relationship: The relationship between the target and the social engineer is exploited and the tar-

get is elicited to perform the request or action that the social engineer desires.

6. Debrief: This phase tests whether the goal has been satisfied. If it is, the attack is a success. If not, the attack can return to the preparation phase where a new attack vector can be developed for another attempt.

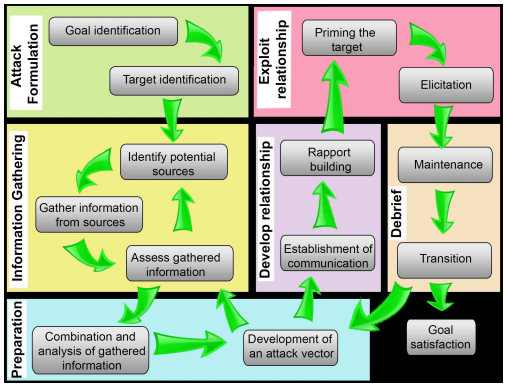

Figure 1: Mouton et al. [10] Improved Attack Framework

A social engineering attack consists of a social engineer (the attacker), a target, a goal, a medium, and the various psychological techniques that are used influence the target to divulge sensitive information [10].

Given the social engineering attack framework, we can see that the success of a social engineering attack hinges on the exploitation of the relationship between the attacker and the target. In the following section, the detection and prevention of social engineering attacks during the interaction between the attacker and the target will be discussed.

1.2 Detection of Possible Attacks

The success or failure of a social engineering attack is determined by the outcome of the interaction between the attacker and the target. The interaction would commonly involve the attacker requesting sensitive information which the target then has to determine whether or not it is safe to comply with the request. Several different models have been developed for the purpose of the detection and prevention of social engineering attack during this form of interaction. Examples of these models include:

1. The use of a feedback neural network to detect whether a request is a social engineering attack or not [11].

2. The Social Engineering Defense Architecture (SEDA)[4]. A common social engineering attack would involve a social engineer pretending to be an authoritative figure over the target. SEDA proposes using a voice signature authentication system to determine whether a requester is who they claim to be over a telephonic interaction with the target.

3. The use of natural language processing to detect social engineering attacks [12]. This uses a software system to detect textual software engineering attacks, such as phishing emails.

4. The Social Engineering Attack Detection Model version 2 (SEADMv2) [8]. An improved version of the SEADM[1]. This contains a set of binary states through which a user traverses through by answering questions in order to determine whether or not a request is a social engineering attack.

The SEPTT Project will focus on the SEADMv2, developed by Mouton [9], a supervisor of the project. Thus the tools discussed later in the paper will all be based on the SEADMv2.

As depicted in figure 2, a user who has received a request begins at the start state and traverses through the model by answering the questions until an end state is reached which determines whether the user should carry out the request or not.

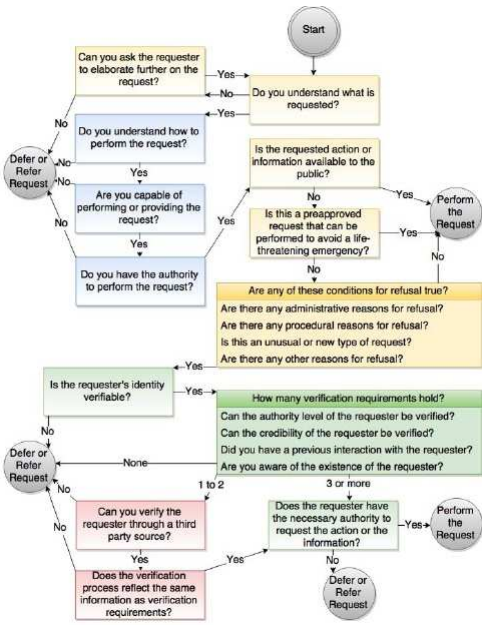

Figure 2: Mouton et al. [8] Social Engineering Attack Detection Model version 2

The questions in the SEADMv2 each belong to a state which defines what type of question it is. The yellow state deals with the request itself. The blue state deals with whether or not the user understands the request. The green state deals with the requester and any information that can be determined about the authenticity of the requester. Lastly, the red state deals with whether the requester can be verified by a third party.

1.3 Tools for the Prevention of Attacks

With the human element of an information security system highlighted by attackers as the weak spot, and thus targeted more often, the need for the development of tools to detect and prevent possible social engineering attacks has arisen.

The SEPTT project aims both to create such tools, based on the SEADMv2, as well as services to aid in the creation of tools in the future. As part of the project, two applications were developed: a website based application, and a mobile application. Both of these applications were created

for the detection of a social engineering attack by traversing through the SEADMv2 described above. Using the applications, users have to assess their situation by answering the questions in SEADMv2 until they reach an end state where they have to defer the request in the case that it is determined that it is a possible social engineering attack or they perform the request in the case that it is determined that it is a valid request. The project also involves the testing of these applications to determine whether they were effective as tools to detect and prevent social engineering attacks.

This paper will focus on the creation of the services to aid in the development of social engineering prevention and training tools. These services will also be centered around the SEADM and will provide a database which stores the SEADM, as well as a web-API to publicly access the database remotely. A web-based visualization and editing tool will also be provided if any further iterations are needed to be made to the SEADM.

2. SERVICES FOR TOOLS TO DETECT AND PREVENT SOCIAL ENGINEERING ATTACKS

This section will outline the design requirements for the services to aid in the development of social engineering prevention and training tools. These services are made up of 3 elements:

1. The designing and implementation of a database to store the SEADM.
2. A web-based API to give public access to to the model.
3. A web-based tool to visualize the SEADM as well as to allow for updates to be made to it.

The functions and requirements for each of the above elements will be discussed.

2.1 Database

The purpose of the database is to store the model and be used by applications to access its contents. The requirements for the design of the database were as follows:

- Store the questions of the SEADM
- Store the transitions between questions and states of the SEADM
- store the different states of the SEADM
- Cater for non-binary questions
- Designed to be easy to edit the SEADM in any way (changing transitions, adding or removing of questions / states, etc)

2.2 Web API

The purpose of the web-API is to give public access to the database storing the model and providing the relevant functions to make any changes to the database. The requirements of the web-API are as follows:

- Allow for the retrieval of all fields of all the tables in the database.
- Allow for editing of any of the fields in all of the tables in the database.
- Allow for the deletion of any field in the tables in the database.
- Allow for all the above actions to be carried out remotely by an application by HTTP request.
- Return information requested from the database by an application in JSON format so that it could be used by any application regardless of platform.
- Designed to be well-structured and easily maintainable in the case that changes are made to the structure of the database.

2.3 Visualization Tool

The purpose of the visualization tool is to allow for the dynamic visualization of the model. This would be the drawing of the model in the form of a flow diagram (similarly to that of the diagram of the SEADMv2 depicted in figure 2) as well as to allow for the changes to made to the model. The requirements for the visualization tool were as follows:

- Make use of the web-based API outlined above to access the model.
- Display all the questions in the model.
- Display the transitions of between the questions in the model.
- Allow updates and edits to be made to the model and storage of the updated model through the use of the web-based API.

3. DESIGN AND IMPLEMENTATION

This section will discuss the design and the implementation of the 3 requirements outlined in the previous section. The choices of technologies and tools used to create them will also be discussed.

3.1 Database

The purpose of the database was to the store the SEADM in a manner which it can be traversed given only what was stored, thus storing the transitions between the questions and the states. This would allow for applications with access to the database being able to traverse the model in the intended way.

Designing a database that could both store the information of the SEADM, as well as its functionality in terms of the transitions between questions and states and accommodate for any changes to its structure proved to be a very difficult challenge with no simple solution. While early designs met some of the criteria required, issued persisted which would either compromise the maintainability of the database or the functionality of the SEADM. The final design was able to store the functionality of the SEADM as well as accommodate for structural changes to the SEADM, with the sacrifice of simplicity.

3.1.1 Early Design

The early database design was very simplistic, containing only 2 tables, but could not accommodate for all of the features of the SEADM. The design of the database is depicted in the figure below.

As the name implies, the Question table stored the questions of the SEADM as well as the transitions depending on the user's answer. The 'yes' and 'no' fields stored the transitions ID to the next question if that answer was chosen.

The State table stored the name of the state as well as mapped to the state field in the question table, allowing for future states to be created if needed.

Figure 3: Early design of the databse to store the SEADM

While this design would work for most questions in the SEADM, since it allowed for only 2 possible answers, it could not accommodate for one of the questions which was not a binary state, and had 3 possible answers. An early consideration was to add a third field to accommodate that stand-out question, but that solution would mean that there would be a null third option field for most questions. It would also not accommodate for future questions which could possibly have more than three options.

Another problem with this early design was that it took for granted that questions were answered simply as yes or no, as future iterations of the SEADM could have different possible answers to the questions.

3.1.2 Final Design

Upon finding that designing a database to store the SEADM with all of its features was a very difficult task, the final design was created by project supervisor Francois Mouton. The final design of the database is depicted in the figure below.

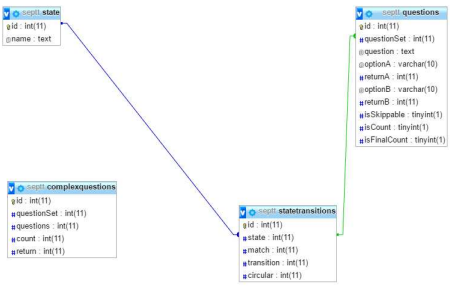

Figure 4: Final design of the database to store the SEADM

This more complex design contained 4 tables, but preserved the features of the SEADM as well as allowed for any form of future question to be accommodated as well as changes to the structure of the SEADM.

The Questions table contained the text of the question as well as the text of the transitions leaving it. The State table contained the names of the state and mapped to the state in the questions table as the early design. The newly added State Transitions table stored the transitions between states and questions and the Complex Questions table stored the information required to accommodate questions that had more than 2 possible outcomes.

3.1.3 Traversal Algorithm

Given the final design depicted in the previous section, applications could traverse the SEADM as intended with the following algorithm:

1. Present the user with the first question in the Questions Table and give them options A and B.

2. Get the QuestionSet value (referring to the question's state) for the current question.

3. Get the return value for the option the user selected (if A was selected, get the value from the Return A field, likewise get the value from the return B field if B was selected).

4. In the State Transitions table, find the entry where the State field is equal to the QuestionSet value of the current question (gotten in 2) and where the Match field is equal to the return value (gotten in 3).

5. In the field in the State Transitions table where the above conditions were met, get the Transition value. This value indicates which state the next question in the SEADM belongs to.

6. If the Transition value indicates that the next state is different to that of the current question, find the first entry in the Questions table where the QuestionSet value is equal to the Transition value. This will be the next question to present the user.

7. If the Transition value indicates that the state of the next question is the same as the current question, present the user with the question that immediately follows the current question in the database if the Circular value in the State Transitions table is equal to -1. if the Circular value is not equal to -1 then present the user with the question that immediately precedes the current question in the Questions table.

3.2 Web-API

In order to provide public access to the database containing the SEADM, a web-API was developed. It allows for applications to access and edit the database through the use of HTTP requests. The web-API returns the requested information from the database in JSON format to be used by applications. Applications can use the web-API for purposes such as the visualization of the SEADM or the creation of social engineering attack prevention tools.

3.2.1 Technologies Used

Flask, a Python based micro-framework, was used for the development of the Web API. The choice of Flask was influenced by its ease of use and its robust core that includes the basic functionality that is essential for all web applications and web-APIs, such as the support of relational databases, and the URL routing of functions. The URL routing built into Flask allowed for a different URL to be assigned to each function which allowed for the different types of transactions to the database to be called by a different URL. Flask also supports many third-party extensions to handle transactions with relational databases [3].

SQLAlchemy was the third-party extension that was chosen for the web-API. SQLAlchemy provided an easy means to access both local and remotely hosted databases. SQLAclhemy is an object-relational mapping (ORM) tool which maps entries in database tables to Python Objects to allow for easy access and manipulation and thus required no actual SQL to be written in the development of the web-API, simplifying development and maintenance [2].

3.2.2 Structure

The structure of the web-API was based on the Model-View-Controller (MVC) design pattern that is commonly used in software development. The MVC Pattern seperates user interfaces and code dealing with user interactions from the underlying data represented by the interface. This separation allows for code to be well-structured, organized, and makes the job of maintenance easier [5].

The model in the web-API consists of separate files for each of the tables in the database. The files each contain the code for where SQLAlchemy connects with the database, and maps the table's fields to python objects.

The controller, like the model, consists of separate files for each of the tables in the database. The purpose of the controller is to interact with the model and to perform the transactions to the database that is requested from the view. The controller files contain separate functions for each of the different transactions that can be made to their respective tables.

Since the web-API does not have a user-interface, there is no traditional view. Instead, a single script that handles the hosting of the web-API and incoming user-requests is treated as the view.

3.2.3 Workflow

Thus given the technologies chosen and the structure of the web-API detailed above, an example of a transaction where all the fields of a table was requested would look as follows:

1. The URL of the HTTP request to return all fields of table runs the function that it is assigned to in the view.

2. The function in the view calls the function in the specified table's controller that is responsible for returning all the fields in that table.

3. The function in the controller then queries the model for the all the fields in the table to which the model returns as a list of python objects. The controller converts the python objects to a list that can be converted into the JSON format and returns the list to the view.

4. The view converts the list into a JSON object and returns it to the application that made the request.

Requests that update, add, or delete entries to the database behave in a similar way, except they return a message if the request was carried out successfully instead of returning a JSON object.

3.3 Visualization Tool

The purpose of the visualization tool was to provide a graphic visualization of the SEADM, similar to the one depicted above, as well as to provide the functionality to make changes to the SEADM. However due to limitations found in the visualization library chosen to create the tool, not all features were able to have been developed. While the tool was able to visualize the SEADM in the manner that was intended, this came at the cost of removing the functionality of editing the SEADM. Due to how late in development the limitations were found, the overhead in terms of time that come with starting from scratch with another library meant that even the basic functionality would not have been developed before the deadlines.

3.3.1 Technologies Used

The Flask micro-framework was again used, although in a limited capacity. For the purposes of the visualization tool it was used to host the tool on a webpage and handle URL routing as well as get and update the tables on the database using calls to the web-API.

Upon investigating options of frameworks/libraries to handle the drawing of the SEADM, Vis.js, a dynamic browser based visualization based library was chosen. Vis.js offers several different types of visualizations, one of which was that of a network diagram, which seemed most appropriate for the purposes of this tool.

3.3.2 Limitations Found in the Vis.js

The major limitation found in Vis.js was that given a complex data set with too many nodes and edges (questions and transitions in the case of the SEADM), the tool would behave erratically and unpredictably. The visualization would at times be drawn over the length of 3 screen sizes, or with nodes and edges overlapping each other to the extent that the visualization was very difficult if not impossible to follow.

The solution to this problem was to reduce the number of nodes and edges by combining some of the questions into one node (similar to how some of the questions in the diagram of the SEADM depicted above). However since the database was not designed to cater for this sort of grouping, hard-coding this grouping into the visualization tool was required to produce an output that could be understood and followed by a user.

This hard-coding of the groupings however meant that any changes to the SEADM would offset the groupings severely with some questions being incorrectly grouped with others. Therefore the features that changed the structure of the SEADM by adding or removing questions or transitions were removed. However the feature to edit the text of a question was still retained as that did not involve the changing of the structure of the SEADM.

In summary, a neat visualization solution, one which could be understood and followed by a user, could not be found without the use of hard-coding elements to be grouped to-

gether. This then led to the inability to change the structure of the SEADM. In the end the solution that was neat, yet partially hard-coded was chosen.

To alleviate the issues caused by the limitations found in the Vis.js library, the possible visualization libraries could have been considered before the development of the database, thus allowing any found limitations to be compensated for in the design of the database.

An alternate solution would be to create a new custom visualization library that ensures that all the needs of the given visualization model is met.

3.3.3 Final State of the Visualization Tool

The final state of the visualization tool included the ability to draw the SEADM in a manner that could be followed and understood by user, but without the ability to make structural changes to the SEADM itself. An image of the visualization is depicted in the figure below:

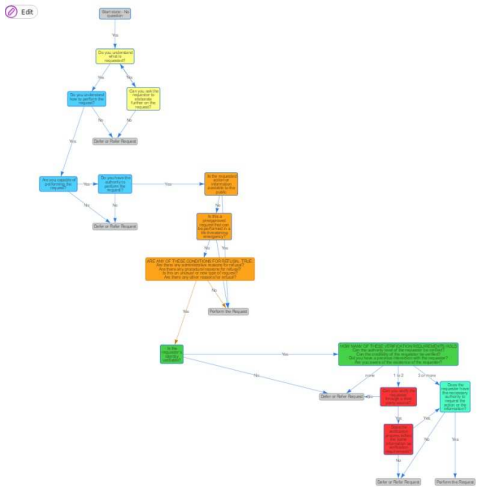

Figure 5: The visualization of the SEADM created by the visualization tool

4. SOFTWARE ENGINEERING PRACTICES CARRIED OUT

The software engineering methodology practiced throughout the development of the 3 elements of the project was similar to that of the Rapid Application Development (RAD), a methodology which favoured the rapid development of prototypes over excessive planning [6].

This form of methodology was chosen given the time constraints of the project. This would allow for working prototypes of the software to be available at any given stage of development, thus ensuring that at least some working functionality was completed by deadlines, with the more important functionality given a higher priority to be developed first.

4.1 In the Development of the Web-API

During the development of the web-API, working prototypes of different aspects were developed. For example, in the initial phase, the functionality of the API regarding the returning of information from the database in JSON format after receiving URL requests was completed without any actual connection to the database as the data from the database was mocked into the application. Only later in the development was an actual connectivity to the database established.

Early prototypes of the web-API involved monolithic code (all the code for an application contained in a singular file) and did not have any regard for structure. The MVC structure was only later implemented after early prototypes successfully performed all the required functionality.

4.2 In the Development of the Visualization Tool

Similarly to the development of the Web-API, early prototypes of the visualization tool had no actual connectivity to the database and instead relied on mock data. This was to first establish a familiarity with working with the visualization library to and to try to ensure that it was capable of the visualization of the model.

5. CONCLUSIONS

As part of the SEPTT project, 3 major element were developed in order to provide services to support the development of social engineering prevention and training tools. While the database and Web-API service were successfully developed, the Visualization tool fell short of the requirements.

5.1 Reflection on the Development of the Visualization Tool

The failure of the Visualization tool to meet the requirements set out was due to several factors. One of which was that the development of the tool was left to last, as it relied on both the database and the web-API to be complete. Given the time-constraints involved, the development had to rely on the functionality of third party visualization libraries, namely VIS.js. While initially VIS.js looked very promising, its limitations only became apparent once it was too late to start over from scratch, forcing the removal of key features from the visualization tool.

In order to have created a visualization tool that met the requirements, one of two aspects needed to have been done differently. Firstly, a visualization library could have been picked and experimented with before the design of the database. This would have allowed for any limitations with the visualization library to have been found early on and for the design of the database to cater to these limitations. Alternatively, a new custom visualization library could have been developed with the specific needs of the project in mind to ensure that there are no limitations that will hinder development of the visualization tool.

5.2 Future Work

A possible avenue for improvement on what's been made would be to allow for the backend web-services to store and provide access to multiple different versions of the SEADM. This would allow for the development of versions of the SEADM that tailored to particular situations or companies.

In the broader scope of the prevention of social engineering attacks, tools based on other means of detecting and preventing such attacks other than the SEADM could be considered. Their effectiveness in preventing attacks as well as educating users about social engineering could be compared with effectiveness of the SEADM.

References

[1] Monique Bezuidenhout, Francois Mouton, and Hein S Venter. "Social engineering attack detection model: SEADM". In: *2010 Information Security for South Africa*. IEEE. 2010, pp. 1–8.

[2] Rick Copeland. *Essential sqlalchemy*. " O'Reilly Media, Inc.", 2008.

[3] Miguel Grinberg. *Flask Web Development: Developing Web Applications with Python*. " O'Reilly Media, Inc.", 2014.

[4] Michael Hoeschele and Marcus Rogers. "Detecting social engineering". In: *IFIP International Conference on Digital Forensics*. Springer. 2005, pp. 67–77.

[5] Avraham Leff and James T Rayfield. "Web-application development using the model/view/controller design pattern". In: *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International*. IEEE. 2001, pp. 118–127.

[6] Hugh Mackay et al. "Reconfiguring the user: Using rapid application development". In: *Social studies of science* 30.5 (2000), pp. 737–757.

[7] Kevin D Mitnick and William L Simon. *The art of deception: Controlling the human element of security*. John Wiley & Sons, 2011.

[8] Francois Mouton, Louise Leenen, and HS Venter. "Social Engineering Attack Detection Model: SEADMv2". In: *2015 International Conference on Cyberworlds (CW)*. IEEE. 2015, pp. 216–223.

[9] Francois Mouton, Louise Leenen, and HS Venter. "Social engineering attack examples, templates and scenarios". In: *Computers & Security* 59 (2016), pp. 186–209.

[10] Francois Mouton et al. "Towards an ontological model defining the social engineering domain". In: *IFIP International Conference on Human Choice and Computers*. Springer. 2014, pp. 266–279.

[11] Hanan Sandouka, Andrea J Cullen, and Ian Mann. "Social Engineering Detection using Neural Networks". In: *CyberWorlds, 2009. CW'09. International Conference on*. IEEE. 2009, pp. 273–278.

[12] Yuki Sawa et al. "Detection of Social Engineering Attacks Through Natural Language Processing of Conversations". In: *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*. IEEE. 2016, pp. 262–265.