Detecting Open Banking API Security Threats Using Bayesian Attack Graphs

1st Dawood Behbehani
School of Mathematics, Computer Science & Engineering
City, University of London
London, United Kingdom
dawood.behbehani@city.ac.uk

2nd Nikos Komninos
School of Mathematics, Computer Science & Engineering
City, University of London
London, United Kingdom
nikos.komninos.1@city.ac.uk

3rd Khalid Al-Begain
Kuwait College of Science & Technology Kuwait
Kuwait
k.albegain@kcst.edu.kw

4th Muttukrishnan Rajarajan
School of Mathematics, Computer Science & Engineering
City, University of London
London, United Kingdom
r.muttukrishnan@city.ac.uk

***Abstract*—Particularly amid Covid-19, enterprises' digital transformation has rapidly accelerated, making cybersecurity an even bigger challenge. Financial institutions adopt FinTech technologies to advance their service and achieve an enhanced customer experience that creates a competitive edge in the market. FinTech products utilise open banking API services to allow communication between a financial institution and a FinTech provider. However, such an integration introduces significant security concerns. Therefore, financial firms must ensure that a robust API service to protect the bank's infrastructure and its customers' information. To address this concern, we propose a Framework for Open Banking API security that utilises STRIDE model to identify security threats in FinTech integration via Open Banking API and Bayesian Attack Graphs to automate predictions of the most exploitable attack paths.**

***Index Terms*—Open Banking API, STRIDE, Risk Assessment, Threat Assessment.**

I. Introduction

'FinTech', a term derived from the financial and technology sectors, is a new phenomenon that seeks to enhance financial transactions for customers. Providing an enhanced customer experience creates a competitive edge in the market. Financial services firms, for example, are continuously adopting and embarking upon new and innovative technologies that aim to provide flexibility and advance operations and customers' experiences. As a result, FinTech has grown exponentially to encompass the advancement of various services, such as digital onboarding, money transfers, advanced expenditure analysis and investment management.

A. Open Banking API

An open banking API can enable open banking services to share financial data services with third-party developed applications. The objective is to enhance customers' experiences with financial services by transforming traditional banking services into digital banking services. However, such an integration introduces significant security concerns regarding the financial data that are passed on to the FinTech application. Therefore, financial firms must ensure that a robust API service and a vigorous FinTech application onboarding process are in place. Therefore, financial institutions must provide robust API services to third-party applications to ensure confidentiality, integrity and availability of online services, as such matters can adversely impact the financial institution's reputation.

B. Contributions

The main contribution of this paper is as following:

- We performed attack predictions generated from the STRIDE threat model using Bayesian attack graphs (BAGs). Based on the threats identified by the STRIDE model, the BAGs automate predictions of the most exploitable attack paths.
- We use the [1] approach to find the access probability of the cyclic graph. We use the [2] approach to find relevant countermeasure plans. The combination of these two techniques is novel. [1] approach is novel because it introduces a computationally efficient way to find access probabilities for cyclic BAGs. Their approach has been found to reduce the time to find access probabilities for cyclic BAGs by a factor of 105 compared to the conventional variable elimination method for graphs with approximately 25 nodes. Although novel, Stan's (2019) approach does not include cyclic BAGs and works only for acyclic BAGs. Our approach to tackling cyclic BAGs uses [2] path-finding countermeasure assessment plan alongside [1] combinational logic approach. We used the A* algorithm to start with the largest possible countermeasure set. However, whenever any countermeasure set is to be checked, our algorithm uses the combinational logic approach to set the access probabilities of the countermeasure set in question to zero and obtain a steady-state access probability. This in turn is used to

calculate the risk to the final node (such as database server access by a hacker).
- We also discuss the possible problems of using the A* algorithm, in which the initial conditions may affect the final countermeasure plan. These issues have not been previously discussed in a network architecture context.

C. Paper organisation

The remainder of this paper is organised as follows: Section 2 demonstrates related work and justifies the reasons for this research; our proposed framework is presented in Section 3; the case study is highlighted in Section 4 to demonstrate our model; Section 5 illustrates the threat assessment and presents and discusses the proposed model; and our work is concluded in Section 6.

II. Related Work

Attack graph is ideal to graphically predict the attacker's potential attack path. Adequate prediction of attacker's path results in selecting efficient mitigation controls to minimize the risk to unacceptable level. Introducing Bayesian network into attack graph enables an useful probabilistic reasoning model. Bayesian Attack Graph (BAG) is a model based on Bayesian Network, a directed acyclic graph comprised of nodes, edges and probabilities. The nodes exemplify random variables, and edges between nodes signify the relationships among variables. Every node has different probability distributions [3]. In [4], the authors claim that BN provides a useful mechanism in the risk analysis domain for its ability to model probabilistic data. The model can use new data to update existing probabilities of events deviating from normal operations and predict prior probabilities in numerous, ways such as the statistical analysis of posterior data. [3], the authors proposed a dynamic risk assessment model using BAGs that enable the quantification of the changes in network infringement at various levels. The model adopts the cause–consequences relationships of various network statuses and considers the likelihoods of exploiting such relationships. In a dynamic environment, the probability is changeable; therefore, a BAG can be used to compute the posterior probabilities to evaluate the new condition. Similar model was also used in this literature [5], whereby the authors proposed a risk assessment framework for wireless sensor networks in cloud that utilises attack graph model to quantify attack vectors. This is achieved b first reviewing the impact of attacks and predict reasonable time frames that forecast the degradation of WSN security posture. [6] also proposed a dynamic risk assessment model using BAG that enables the quantification of the changes of network infringement at various levels. The model adopts the cause-consequences relationships various network status and considers the likelihoods of exploiting such relationships. In a dynamic environment, the probability is changeable, therefore, BAG can be used to compute the posterior probabilities to evaluate the new condition.

[1] proposes a novel approach because they introduce a computationally efficient way to find access probabilities for cyclic BAGs. Their approach has been found to reduce time to find access probabilities for cyclic BAGs by a factor of 105 compared to the conventional Variable elimination method for graphs with approximately 25 nodes. The iterative approach involves implementing BAGs like combinational logic circuits. A BAG is initialized with a randomized access probability (except the leaf nodes whose probabilities remain constant always). The access probability of each node is modified according to the access probability of their parent nodes at each step until the access probability of all nodes stays constant for future iterations. This steady state probability is considered as the access probability of the BAG. Furthermore, [2] countermeasure assessment approach is interesting. It involves finding the maximal set of all countermeasures that can be applied to any BAG (generally countermeasures are applied to leaf nodes) within a relevant budget. A path finding algorithm (A* algorithm) is then initiated which takes this maximal set of countermeasures as the initial starting point and operates an iterative algorithm to find the possible countermeasure plan that within a budget. For each countermeasure set, the access probability of the node is set to zero (that corresponds to the respective countermeasure). Different countermeasure sets are analysed (the order of selecting the next countermeasure set is based on a heuristic they have developed which we have incorporated as well) until a set is obtained that falls within the required budget. Although novel, [2] approach does not include cyclic BAGs and only works for acyclic BAGs.

III. Proposed Framework

The detailed process of the proposed methodology, STRIDE threat modeling using BAG is shown in Figure 1.

1) ***Development of the STRIDE model****:* In this work, we use the STRIDE model, which utilises a components catalogue developed with the main assets of a network architecture for a small enterprise network (specifically, a banking network) with user access to core banking services via an API gateway and middleware. The middleware layer can access data from the database server using privileged requests and obtain a corresponding response. The network architecture is shown in Figure 2. The threat identification process is then executed via a rule-based threat database to effectively outline the security threats associated with the model.

2) ***Vulnerability calculation****:* We perform a vulnerability calculation using the Common Weakness Scoring System (CWSS) of the network to identify Common Vulnerability Scoring System (CVSS) vulnerabilities that an attacker can use to gain access to specific servers. Some examples of the vulnerabilities detected are as follows:

- An adversary may gain unauthorised access to a web API due to poor access control checks.

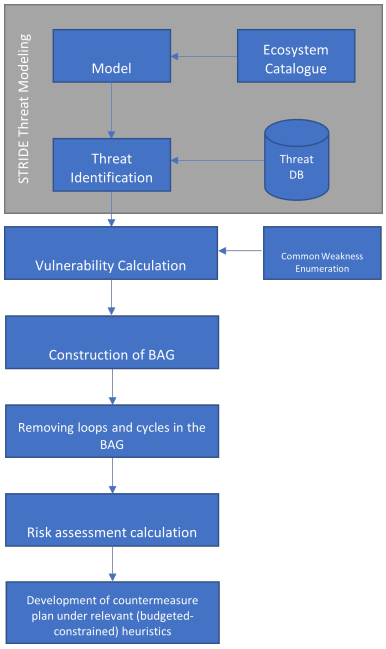

Fig. 1. Proposed framework

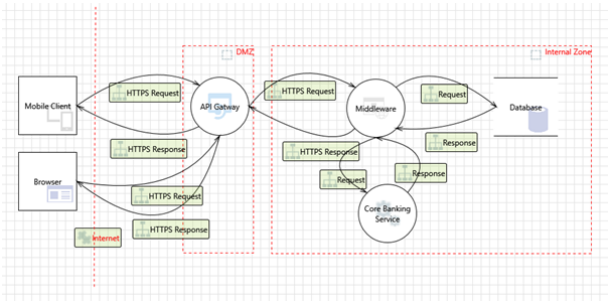

Fig. 2. Network architecture

- An adversary may gain unauthorised access to a database due to a lack of network access protection.

Each vulnerability is associated with its exploitability score via CVSS scores. These scores can be used to obtain the probability of successful attack execution.

3) ***Constructing Bayesian attack graphs (BAG)****:* A BAG represents causal relationships between important network nodes. Using an effective treatment of nodes and edges, the BAG can represent the security landscape and vulnerabilities that may lead to the exploitation of important events or compromise important servers. The exploitation of vulnerabilities gives an attacker a higher-order status on the network, which can then be used to exploit the system further. A chain of nodes in the network connected in this way represents an attack path or route. The nodes are a representation of the possible attack states (which are a combination of the host's current state, the attacker's access level and the assault's effects), while edges indicate actions that produce a change of state.

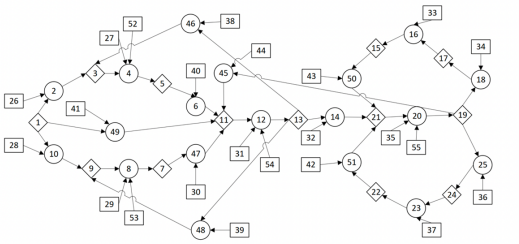

Fig. 3. BAG of the network architecture shown in Figure 2

There are three node types: derivations, primitive facts and derived facts. Derivation nodes are rules that enable primitive facts to be true (circles); primitive facts describe the current state of the system (visualised as rectangles) and may include its security vulnerabilities. Derived facts are a consequence of applying rules to the facts (visualised as diamonds). Edges connect a derivation with a primitive or derived fact.

In this work, the BAG will be simulated as a graph using Python.

4) ***Removing loops and cycles in the BAG****:* A feature of BAGs is the presence of loops and cycles. They represent the different ways in which a critical node (for example, a DMZ or database server) can be accessed. However, a disadvantage of these cycles is that while computing the BAG, iterations will never finish. To prevent this, an algorithm to compute the marginal probabilities that the attacker will access any node is needed. Two methods have been suggested in the literature: variable elimination methods [7] and combinatorial circuits [1]. Both techniques have their pros and cons, and both work well when the number of nodes in the BAG is below 25. For a larger number of nodes, combinatorial circuits are recommended. The BAG we constructed has 44 nodes (Figure 4), so we will apply combinatorial circuit methods for resolving circles i.e. primitive facts.

5) ***Risk assessment and calculation****:* Before finding a cost-effective countermeasure plan for network security, risk assessment must be carried out to estimate the current state of the system and understand the options for enhancing it. For example, a risk mitigation strategy could be to update a firewall rule and disable communication between two subnets (assuming the firewall exists).

The next step is to compute the risk using equations derived from the BAG in the pre-processing phase. The ease with which any attack path can be taken is used to determine the risk. The risk introduced by the attacker's efforts towards their goal, as well as countermeasures that can impede these actions, are represented mathematically in a risk equation. The OR and AND gates are treated as such (refer to the combinatorial logic approach described above), and vulnerabilities are converted to their respective CVSS scores to develop risk profiles for different nodes.

*6) **Development of a countermeasure plan under heuristics:*** Prioritising and implementing proper security countermeasures to decrease a system's risk is part of the treatment phase of any network security system. Because it is impractical to eliminate all risks, we aim to reduce them to an acceptable level within an allocated budget. In the end, we hope to generate a list of countermeasures that hold higher priority and the residual risk to the system when each countermeasure is deployed.

The countermeasure selection problem is a path-finding problem that uses some heuristic (a budget constraint in this work). To find the path from a given initial node to the goal node with the smallest possible cost (budget), two algorithms have been suggested in the literature: the A* algorithm and the Hierarchical Path-finding A* (HPA*) algorithm [8]. HPA* outcompetes A* when multiple starting nodes lead to the destination (attack node). Any countermeasure plan will start with a single node, which comprises all possible countermeasures. The final node is the relevant countermeasure. This means the starting node is defined, but the end node is unknown. For this reason, the HPA* approach is not feasible for our case and we use the A* algorithm.

We will start with a BAG in which the maximal number of countermeasures is implemented and slowly remove certain countermeasures to find the relative risk difference along with money saved. This statistic will be performed for different budget limits to give us the ideal countermeasure implementation.

A. Model description and discussion

1) Formulation of the model: A BAG is developed as an attack graph (V, E) with a set of nodes V = v1, v2 .... vn and edges given by the set E $\subset$ V $\times$ V, which connect the different nodes of the attack graph. The graph can consist of cycles and loops that make probability estimation challenging. A cycle is represented by a set of nodes (v1, v2 ... , vn) such that (vi, vi+1) $\in$ E for all i and vn = v1. Without any cycles, the BAG is a directed acyclic graph. A loop is a set of nodes (v1, v2 ... , vn) such that vn = v1 for any i, and whether (vi, vi+1) $\in$ E or (vi+1, vi) $\in$ E.

Each node is associated with an access probability p(v), which determines the probability with which any given node can be exploited during an attack. The access probability of a BAG is a function of its parent nodes and depends on the relationship between the parent nodes, which can be an OR or AND relationship (or a combination thereof). The access probability can be formulated using the following risk equations:

$$P(v) = \begin{cases} p(v), & if\ v\ \ V_l \\ p(v)\prod_{v' \mathrm{pa(v)}} P(v'), & if\ v\ \ V_a \\ p(v)(1-\prod_{v' \mathrm{pa(v)}} (1-P(v')), & if\ v\ \ V_o \end{cases}$$

where pa (v) is the set of parent nodes, Vl is a leaf node, Va is the set of parent nodes that have an AND relationship and Vo is the set of parent nodes that have an OR relationship.

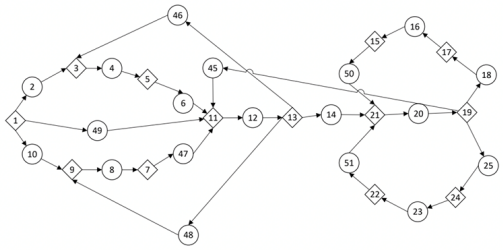

Fig. 4. Simplified view of the BAG without the leaf nodes

Note that the probability values of leaf nodes are calculated by their CVSS scores if the node represents a vulnerability. The probability is defined as:

$$P(v) = \frac{C}{F} xAVxACxPRxUI$$

where C denotes the exploitation factor and F denotes the upper limit of the exploitation score. AV, AC, PR and UI represent the static metrics: the access vector, the attack vector metric, the required permissions for intervention metric and the user interaction metric, respectively.

The other type of leaf node represents network communication between different servers or clients (for example, mobile client to API or middleware to database server).

The network architecture used in this work is shown in Figure 2. Figure 3 shows the corresponding developed BAG. A description of the nodes is presented in Table **??**. Figure 4 shows a simplified view of the BAG, from which the excess leaf nodes have been removed.

2) Mathematical description of the model: We hereby deploy the algorithm developed by [1] called "combinational circuits with probabilistic inputs" to obtain the steady state of probabilities of the different BAG nodes.

The algorithm works as follows: An attack graph (V, E) with a set of nodes v1, v2 .... vn and edges given by the set E is converted from a BAG to an augmented attack graph (V, E) with nodes V = v1, v2, ... vn, ... v1, v2, ... vn. The modified augmented graph is developed by adding a node v for each node in V such that the edges are E = E U (v1,v1),...,(vn,vn). The method works on an iterative procedure in which an initial probability, Pini(v), is defined for all the nodes in the BAG. Note that the access probability of all access nodes remains constant and equal to the initial access probability.

These added nodes play the role of tackling the local probabilities at node v. In this iterative approach, the value of node v at the (k + 1)th iteration is given by

v(k + 1) = g(pa(v)(k),v),

where the function g(a,b) is the AND or the OR gate and pa(v)(k) are the parents of node v at the kth iteration.

The behaviour of the attack graph is now written as an iterative numerical equation,

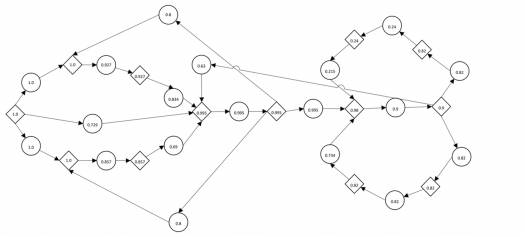

Fig. 5. Steady state probability values of the BAG

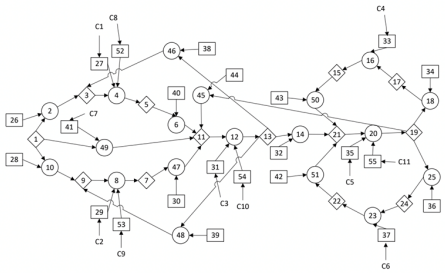

Fig. 6. BAG with the countermeasures

v1(k + 1) = g(pa(v1)(k),v1)
v2(k + 1) = g(pa(v2)(k),v2)
.
.
.
vn(k + 1) = g(pa(vn)(k),vn).
The above set of equations is solved until we obtain a condition where vi(k + 1) = vi(k) for all the nodes of the graph. This is the steady-state solution.

We assign initial random probabilities to the BAG nodes (for the derivation facts and the derived facts nodes), except for the leaf nodes (primitive nodes for which the probability remains constant over time). The algorithm discussed in [1] iteratively updates the probability of the BAG nodes until a steady-state value is reached. This is shown in Figure 7, where the probabilities of nodes are updated with iterations until a steady state is achieved. The initial and steady-state values of the nodes are shown in Table **??**. Figure 6 shows the simplified BAG with the steady-state probabilities of the nodes.

3) Countermeasure selection: Prioritising and implementing the proper security countermeasures to decrease the system's risk is part of the treatment phase of any network security system. Because it is impractical to eliminate all risks, we aim to reduce risk to an acceptable level within an allocated budget.

Before finding a cost-effective countermeasure plan for network security, a risk assessment was performed to estimate the current state of the system and to understand the options for enhancing it. The vulnerability assessment was generated based on the STRIDE model output, which highlights the different vulnerability points along the network, the type of vulnerability and possible mitigation actions. For example, a risk mitigation strategy could be to update a firewall rule and disable communication between two subnets (assuming the firewall exists).

Countermeasures operate on the lead nodes and make the access probability of the lead nodes equal to 0 (and thus resistant to exploitation). This will effectively reduce risk propagation within the BAG. The details of the selected countermeasures are listed in Table I. Table II further highlights the countermeasures, their SDL phases, their respective locations in the BAG and their implementation costs. Figure 7 shows the locations of the countermeasures on the leaf nodes.

4) Countermeasure selection problem: The countermeasure selection problem is a path-finding problem using some heuristic. This problem was inspired by [2], who employed the A* algorithm to find a relevant set of countermeasures. The problem statement that the graph finding problem solves is as follows:

We define a maximum budget that has been allocated to reduce risk in the system. A path-finding algorithm starts by considering all the possible countermeasures in the system and estimates the total budget required to implement them and the risk reduction they cause. This set C = C1, C2 ...., Cn, where Ci is the ith countermeasure, represents the starting point (starting node) of the path-finding algorithm. The A* algorithm works by removing nodes from the original set C and comparing whether the new set C fits the budget. The removal of nodes and node access in the next step is determined by a heuristic. A priority queue that stores the different paths traversed by this heuristic, sorted in ascending order, is defined.

The heuristic is defined as f(x) = g(x) + h(x), where g(x) is the risk in the system when all countermeasures in x are deployed, and h(x) is an estimate of the risk added by removing countermeasures from x to comply with budget limitations. For a detailed description of the heuristic, readers are directed to [2].

Whenever a new set of countermeasures is tried, the probability of accessing the nodes they affect turns to zero. Now, the combinational circuit algorithm, as discussed in the previous section, is deployed to identify the new risk in the system. The measure of risk in the system is defined by the access probability of Node 17, that is, the risk of the attacker gaining access to the database server.

The algorithm will stop when a given set of countermeasures reaches the budget limit.

IV. Open Banking API Use Case

STRIDE threat modelling will be applied to the Open Banking API use case based on an AWS well-architected model. In a typical open banking ecosystem, a consumer would access third-party apps after acquiring consent to access consumer data or authorisation to make payments. AWS posits that a well-architected Open Banking API offers an OAuth 2.0 authorization standard [9]. The solution is scalable,

Countermeasure id	Description
C1	Ensure that standard authentication techniques are used to secure Web APIs.
C2	Ensure that standard authentication techniques are used to secure Web APIs.
C3	Encrypt sections of Web API's configuration files that contain sensitive data.
C4	Add digital signature to critical database securable.
C5	Ensure SQL server connection encryption and certificate validation
C6	Consider using a standard authentication mechanism to authenticate to Web Application.
C7	Implement proper authorization mechanism in ASP.NET Web API.
C8	Ensure that auditing and logging is enforced on Web API.
C9	Ensure that sensitive data relevant to Web API is not stored in browser's storage.
C10	Ensure that model validation is done on Web API methods.
C11	Disable XSLT scripting for all transforms using untrusted style sheets.

TABLE I
DESCRIPTION OF COUNTERMEASURES USED

Id	SDL phase	Location	Cost of implementation
C1	Design	Between mobile client and API	100
C2	Design	Between browser and API	200
C3	Implementation	Between API and Middleware	1000
C4	Implementation	Between database server and Middleware	300
C5	Implementation	Between middleware and database server	700
C6	Design	Between core banking services and middleware	500
C7	Implementation	Between attacker and API access	400
C8	Implementation	Between mobile client and API	200
C9	Implementation	Between browser and API	300
C10	Implementation	Between API and Middleware	800
C11	Implementation	Between middleware and database server	800

TABLE II
DESCRIPTION OF THE COUNTERMEASURES AND THE COST OF DEPLOYING EACH COUNTERMEASURE

elastic, instant (or near-instant in terms of providing access to data), and tamper resistant (when it comes to its logging and auditing capabilities). Typically, in an open banking system, the consumer accesses the third-party application approved by the financial institution and provides their consent for the application to access their data or to perform payment transactions [**?**]. A high-level view of the model is shown in Fig. 2.

Third-party applications are connected to the API web service using an OAuth 2.0 token. The API web service is connected to the API, which is also known as middleware. The middleware offers microservices and integration with other core banking services or satellite systems, such as an accounting system, payment gateway, and loan origination system. Therefore, the ecosystem consists of three zones, as shown in 2: third-party apps, DMZ, and internal systems.

The third-party apps are developed and tested by FinTech companies to offer added-value services to customers. Typically, their business model is based on monthly or yearly subscriptions.

The API gateway provides a unified entry point for internal systems or satellite systems within the organization. The API gateway also offers other critical attributes, such as controlled user access and other security measures that are applied as a security policy [10]. Typically, API gateways operate like a proxy; they listen to incoming requests that are then routed to relevant applications for a service.

The internal systems contain middleware that integrates with core banking services, such as the core banking system, payment gateway, and card switch system.

V. RESULTS AND DISCUSSION

As discussed above, the A* algorithm is initiated by supplying a complete list of countermeasures as the initial node. In our first instance of analysing the data, it was observed that the sequence with which nodes are arranged in the initial countermeasure list does affect the risk of the BAG to some extent. The reason is that the A* algorithm uses a priority queue and might not be able to traverse the entire space, thereby leaving certain practical and feasible solutions behind. We expect that for a relatively small countermeasure set, as discussed in [2], the A* algorithm will always generate the best plan, but as the countermeasure space increases, this may not be true (as we have seen). We show two implementations of the A* algorithm, one in which we submit the initial countermeasure plan as a forward (f) plan C1, C2, ... C11 and another in which we submit the initial countermeasure plan as a reverse (r) plan C11, C10, ... C1.

(Also mentioned in Framework) It was observed that the algorithm predicts different optimal countermeasure plans depending on the initial starting conditions. We ran the A* algorithm with two starting conditions: one in which the maximal set is submitted as C1, C2,...., Cn and the other in which we submitted the maximal countermeasure set as Cn, Cn-1,... C1. We expect that as the number of possible countermeasures increases, such instances will be more likely,

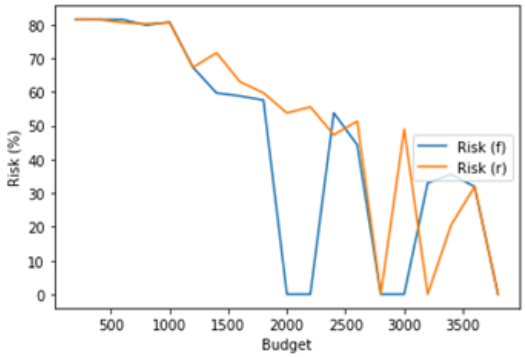

Fig. 7. Risk in the network as a function of budget for 2 ways of initiating the countermeasure nodes

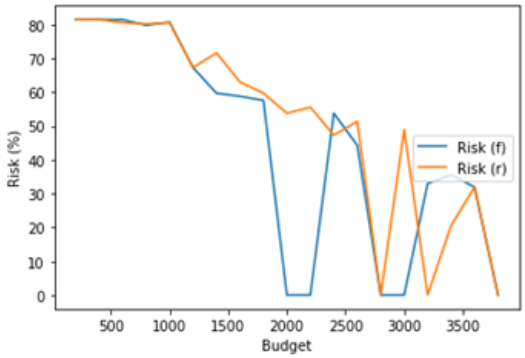

Fig. 8. Time required in calculating the risk of database server access as a function of the maximum budget allocated for 2 ways of initiating the countermeasure nodes

and a user would be advised to try different randomised starting sets of the maximal countermeasure plan to arrive at the optimal strategy.

Figure 7 shows the risk of accessing the database server under both initial conditions. It was observed that the forward (f) plan can find an optimum countermeasure set that reduces the risk to the BAG to zero at a lower budget compared to the reverse (r) plan. Another point is that although plan (f) predicts good countermeasures for $2000 and $2200, when the budget is increased to $2400, it predicts a different countermeasure plan with non-zero risk. This is because of the way we chose our heuristic, f(x). Under different maximum budgets, the heuristic can modify our priority queues, thereby resulting in suboptimal predictions of countermeasure plans.

Figure 8 compares the time required to find the optimal countermeasure plan with the forward (f) and reverse (r) plans. The time needed to run each plan is similar, but that is expected since both employ A* and follow the same heuristic logic.

Note that for each A* iteration, we had to deploy [1] combinational logic algorithm to find the steady-state access probability of the BAG nodes. As such, we expect our algorithm to be slower than [2] algorithm, which did not have cycles in the BAG, making it a directed acyclic graph for which risk can be calculated using an analytical function. However, we always have to perform the iterative combinational logic approach to calculate the risk, which is still far better than the variable elimination method. As the budget increases, the A* algorithm generates a faster output because it must remove fewer countermeasures to generate the optimal countermeasure plan.

VI. Conclusion

Identifying security threats in the early stages of software development helps developers to understand the threat landscape. Every industry and every organisation within an industry has a different risk profile. Therefore, understanding threats is essential to help determine the security controls required to minimize risks. This work presents a new framework for identifying threats towards open banking API services using a catalogue of threats and Bayesian Attack Graph. In this paper, we performed attack predictions generated from the STRIDE threat model using Bayesian attack graphs (BAGs). Based on the threats identified by the STRIDE model, the BAGs automate predictions of the most exploitable attack paths. We adopted [1] approach to find the access probability of the cyclic graph. We use the [2] approach to find relevant countermeasure plans. The combination of these two techniques is novel. We used the A* algorithm to start with the largest possible countermeasure set. However, whenever any countermeasure set is to be checked, our algorithm uses the combinational logic approach to set the access probabilities of the countermeasure set in question to zero and obtain a steady-state access probability. This in turn is used to calculate the risk to the final node (such as database server access by a hacker). We also discussed the potential problems of using the A* algorithm, in which the initial conditions may affect the final countermeasure plan. Future research will combine the STRIDE methodology with tactics, techniques and procedures (TTPs) to ensure the identification of more detailed security threats that can then be utilized to develop security incident response procedures.

References

[1] Isaac Matthews, John Mace, Sadegh Soudjani, and Aad van Moorsel. Cyclic Bayesian Attack Graphs: A Systematic Computational Approach. *Proceedings - 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2020*, pages 129–136, 5 2020.

[2] Orly Stan, Ron Bitton, Michal Ezrets, Moran Dadon, Masaki Inokuchi, Yoshinobu Ohta, Tomohiko Yagyu, Yuval Elovici, and Asaf Shabtai. Heuristic Approach for Countermeasure Selection Using Attack Graphs. *Proceedings - IEEE Computer Security Foundations Symposium*, 2021-June, 2021.

[3] Bijay B, Priscilla George, V. R. Renjith, and Anish Job Kurian. Application of dynamic risk analysis in offshore drilling processes. *Journal of Loss Prevention in the Process Industries*, 68:104326, 11 2020.

[4] Roberto Bubbico, Shenae Lee, Daniel Moscati, and Nicola Paltrinieri. Dynamic assessment of safety barriers preventing escalation in offshore Oil&Gas. *Safety Science*, 121:319–330, 1 2020.

[5] Amartya Sen and Sanjay Madria. Risk assessment in a sensor cloud framework using attack graphs. *IEEE Transactions on Services Computing*, 10(6):942–955, 11 2017.
[6] Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. Dynamic security risk management using Bayesian attack graphs. *IEEE Transactions on Dependable and Secure Computing*, 9(1):61–74, 2012.
[7] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.
[8] Daniel Foead, Alifio Ghifari, Marchel Budi Kusuma, Novita Hanafiah, and Eric Gunawan. A systematic literature review of a* pathfinding. *Procedia Computer Science*, 179:507–514, 2021.
[9] Open Banking - Financial Services Industry Lens.
[10] Prabath Siriwardena. Edge Security with an API Gateway. *Advanced API Security*, pages 103–127, 2020.