Computing Optimal Attack Strategies Using Unconstrained Influence Diagrams

Viliam Lisý and Radek Píbil

Agent Technology Center, Department of Computer Science and Engineering
Faculty of Electrical Engineering, Czech Technical University in Prague
{viliam.lisy,radek.pibil}@agents.fel.cvut.cz
http://agents.fel.cvut.cz

Abstract. Attack graphs are a formalism for capturing the most important ways to compromise a system. They are used for evaluating risks and designing appropriate countermeasures. Analysis of attack graphs sometimes requires computing the optimal attack strategy that minimizes the expected cost of the attacker in case of stochastically failing actions. We point out several results in AI literature that are highly relevant to this problem, but remain unnoticed by security literature. We note the problem has been shown to be NP-hard and we present how the problem can be reduced to the problem of solving an unconstrained influence diagram (UID). We use an existing UID solver to assess the scalability of the approach, showing that it can be used to optimally solve attack graphs with up to 20 attack actions.

Key words: dependency attack graph, optimal attack strategy, unconstrained influence diagram

1 Introduction

Attack graphs (AG) are a popular tool for analyzing and improving security of computer networks, but they can be used also in any domain, where attacks consist of multiple interdependent attack actions. Attack graphs capture all the most important sequences of actions that may lead to compromising a system, and they can contain additional information, such as the cost of individual actions and the probability that the actions will be successfully executed. Attack graphs can be used to evaluate risks and design appropriate countermeasures. In analysis of attack graphs, it is often of interest to identify the optimal strategy of the attacker (i.e., the best sequence of actions) and its expected cost. For example, the expected cost of the attack indicates if a rational attacker would attack the system at all [1] and for penetration testing, it is important to know the expected time required to break into the system [2]. A problem equivalent to computing the optimal attack strategy, with a different domain motivation, has been proven to be NP-hard in [3], and we did not find any network security paper citing this paper. On the contrary, Sarraute et al. [2] present an allegedly optimal algorithm for a variant of this problem, which has been previously shown incorrect in general in [3].

In this paper, we show that the problem of finding the optimal attack strategy for an attack graph with stochastically failing costly actions can be transformed to the problem of solving an Unconstrained Influence Diagram (UID)[4]. UID is a generic graphical model for decision making in uncertainty. Based on a set of random variables, decision variables, utility functions and their dependencies, it allows computing the optimal order of decisions, observations, and the expected utility of the problem. Creating the link between the optimal attacks in attack graphs and solving UIDs can be beneficial for research in both the AI and security domains. For security research, this link allows using the existing UID solvers for fast prototyping of attack graph analysis tools and the theory and algorithms developed for UIDs can help in designing more efficient specific algorithms for attack graphs. On the other hand, connection with attack graphs can give the UID research an additional practical application with specific requirements and tradeoffs that can guide their future research.

The following two sections briefly introduce attack graphs (Section 2) and unconstrained influence diagrams (Section 3). Section 4 presents the transformation of AG to UID followed by scalability experiments in Section 5, showing that the existing solvers are sufficient to solve small, but realistic problems.

2 Attack Graphs

Several different types of attack graphs have been proposed in literature, but we focus on the most commonly used *dependency attack graphs*. These AND-OR graphs include two types of nodes, which represent attack actions (AND nodes, ellipses) and privileges (OR nodes, diamonds) that serve as their preconditions and post-conditions. An action can be performed only if the attacker obtains all the privileges that are preconditions of the action. It is represented by arrows from the action to the privileges in the attack graph. The privileges can be either a result of some other actions, which is denoted by an arrow to that action, or they can be a priori available to the attacker. The latter are termed facts and denoted by rectangles. Single or a very small number of nodes in the attack graph are selected as the top level goal of the attacker.

Dependency attack graph for a computer network can be automatically generated[5, 6] based on reports from network scanning tools, such as Nessus[1] or OpenVAS[2]. These tools scan the network (or individual hosts) for open ports, installed software versions and similar characteristics, and based on large databases of known vulnerabilities identify the once that are present in the network. The vulnerability databases also include additional structured information usable to assess the probability of success and the expected cost of exploiting individual vulnerabilities.

An example dependency attack graph for a simple network is presented in Figure 1. The attacker's goal is to access a database represented by the top node. This can be done either by a remote exploit execution on the database

[1] http://www.nessus.org [2]http://www.openvas.org

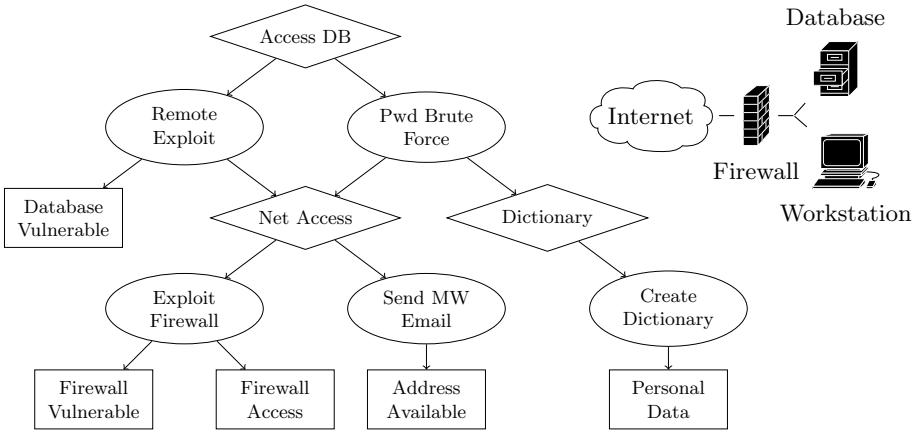


Fig. 1. An example dependency attack graph representing the possible ways to gain an unauthorized access to the Database.

server or by brute force attack on the password. Both these actions require direct network access to the database that the attacker does not have. However, it can be obtained either by breaking into the firewall or by sending the user at the workstation an email with malware (MW), i.e., a Trojan horse. In order to perform the brute force attack, the attacker needs a dictionary that might include the database password. This can be created based on the personal data of the administrator, publically available online.

2.1 Removing Directed Cycles

In general, attack graphs may contain directed cycles: If an attacker can modify an arbitrary file on a machine, she/he can also execute code as administrator by modifying a file often executed with these privileges; if he/she can execute code as an administrator, she/he can run a program to modify an arbitrary file. The cycles make the algorithms working with attack graphs more complicated. Alternatively, the cycles can be removed without changing semantics of the attack graph or adding any assumptions. Wang et al. [7] divide the cycles present in attack graphs to three different types. In the first, none of the privileges in the cycle can be achieved and the whole cycle can be removed from the attack graph. In the second type, an edge in the cycle points to a privilege that had to be satisfied in order to enter the cycle. This edge can be removed. In the third type, Wang et al. [7] observe that the cycle cannot be easily removed, but Homer at al. [8] present a method that substitutes these cycle by all unique acyclic paths that traverse the cycle in the attack graph and few additional nodes that merge them. This method creates an equivalent attack graph without the cycle. We did not find any thorough theoretical examination of the effect of complicated nested cycles on the size of the attack graph after applying this method, but we encountered no complex cycles in our experiments.

2.2 Computing the Optimal Attack Strategy

Computing the optimal (i.e., cost minimizing) attack strategy and its expected cost can be very useful. Besides the earlier examples [1, 2], the optimal attack strategy is necessary to compute measures, such as the probability that a specific action will be used in an (successful) attack, usable to guide deployment of additional sensors. The possibility of action failure makes finding the optimal sequence of attack action nontrivial. If the attacker has more independent options to achieve a privilege, he/she prefers to try the less costly and more probable options first. If she/he has to gain multiple privileges to execute an action, he prefers to try to obtain the less probable first to fail before incurring additional costs. Moreover, shared preconditions and effects make the success and cost of different options interdependent. The fact that computing the optimal attack strategy is a computationally complex task has been discussed also in [1]. The authors propose an algorithm to compute an upper bound on the attacker's utility instead of the exact value, which can be sufficient for certain applications.

Greiner et al. [3] analyze several variants of the problem of "probabilistic AND-OR tree resolution" in generic AND-OR graphs, some of which are equivalent to finding the optimal attack strategy. However, there is no connection to network security in this paper and their results seem to be unknown in network security literature. They show that if the graph is a directed acyclic graph (DAG), the problem of computing the optimal attack strategy is NP-hard. In the case of attack trees, which have to assume independence that does not hold in practice, the complexity is unknown for general case and they provide a dynamic programming algorithm exponential in the size of the tree. Furthermore, Greiner et al. [3] show that the algorithm for solving AND-OR trees later reinvented by Sarraute et al. [2] (and presented as optimal for any depth) is correct only for AND-OR trees of depth 2 and it can have an arbitrary error on larger trees.

3 Unconstrained Influence Diagrams

Unconstrained influence diagrams (UID)[4] are a graphical computational model developed for decision making with uncertainty, based on probability inference. In comparison to classical influence diagrams, the unconstrained version allows defining only a partial ordering on possible decisions. An UID is a directed acyclic graph (DAG) with four types of nodes:

- A *decision* node represents a variable that can be assigned one of a discrete set of values by the user. It is drawn as a rectangle and the edges pointing to a decision node determine the variables that must have their values assigned before the user can make the decision.
- An *observable chance* node represents a discrete random variable that can be observed by the user. They are drawn as double circles and the edges pointing to a chance variable represent causal influence in the same way as in Bayesian networks. The observations can be performed after all decisions that can influence them are set.

- A *non-observable chance* node is similar to an observable chance node, but the user can never observe the value assigned to the random variable.
- A *utility* node represents the utility gained by the user (i.e., the decision maker). They are drawn as diamonds and the adjacent edges represent functional dependence.

The graph structure of UID is supplemented by a substantial amount of further data. For each chance node, it requires the conditional probability of each value assignment, given all combinations of the values in the variables, from which an edge points to the chance node. For each utility node, it requires the utility gained by the user for each combination of the variables the utility node depends on. For the decision node, it requires utility for each decision. The utilities in the decision nodes could be represented by a utility node with an edge only from the decision node, but as almost all decisions induce some costs, having utility directly in the decision nodes simplifies the graphs.

The solution of an UID is a strategy for the user determining when to make which decision and observation in order to maximize the expected utility. The order of the decisions and the selected values generally depend on the observations made by the user; hence, the strategy is in fact a contingency plan (i.e., a decision tree) defining an optimal decision for each possible sequence of observations. The size of the explicit representation of this strategy can be exponential in the number of observations in the worst case; hence, their compact representations in form of a DAG are studied [4]. Moreover, approximate and anytime (e.g., [9]) algorithms are available for solving UIDs.

One of the advantages of using a generic computational model is the availability of existing solvers. There are several solvers capable of solving UIDs. We use a solver called Guido described in [10]. It computes the optimal strategy based on the compact DAG representation. The worst case complexity of its computation is exponential in both the size of the input UID [11] and the size of the domains of the variables. However, it can still be used to solve smaller problems. We evaluate the scalability of this tool on the UIDs based on attack graphs in Section 5, but we expect that substantially larger problems can be (approximately) solved using more advanced solvers.

4 Translating Attack Graphs to UIDs

The key problem in both solving an attack graph and an UID is the ordering of actions (decisions) that influence the expected utility of the attacker (user). The problem of solving an UID is more general. Computing the optimal strategy in an attack graph can be reduced to solving an UID. The first step is removing the directed cycles, as it is described in Section 2.1. We further assume the attack graph does not contain directed cycles.

All variables in the UID for solving an attack graph are binary; the values are either true or false. The structure of the UID very closely copies the structure of the attack graph. Similarly to the probabilistic calculation presented in [8], we

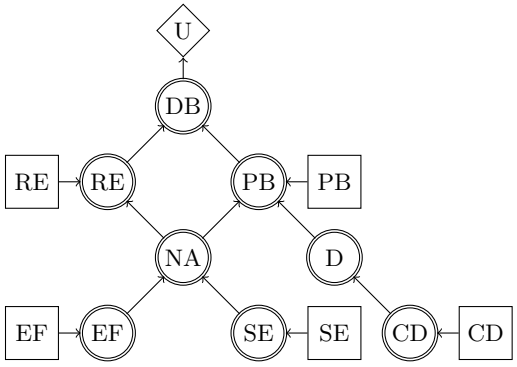
can remove all the fact nodes form the attack graph. All of them are certainly true and do not incur any costs or ordering constraints.

Any of the action nodes (i.e., AND nodes) in the attack graph can be translated to two nodes in the UID, one decision node and one observation node. The decision node determines if the attacker executes the action or not. The chance node represents whether the execution is successful. If the decision or any of the preconditions is *false*, the probability of success is zero. If the decision and all the preconditions are *true*, the probability of success is given by the probability of success of the action in the attack graph. The cost of the action is reflected in the utility in the decision node. If the attacker decides not to execute the action, the immediate utility is zero. If the decision is to execute the action, the cost (negative utility) is paid regardless of the success of the action.

Any privilege node (i.e., OR node) can be represented by an observable chance node. The conditional probabilities in the node represent the boolean OR function. If any of the variables it depends on is *true*, the chance node will be *true* with probability 1, otherwise it is *false* with probability 1.

There is one utility node for each top level goal in the constructed UID, connected with single edge from the variable representing the goal. If the goal is satisfied (the value of the variable is *true*), the utility is the reward of the attacker for a successful attack. If the reward is not known, it can be set to a sufficiently large number and the expected cost can be calculated at the end by subtracting the probability of successful attack multiplied by this reward.

4.1 Example



DB	u(DB)
T	10^6
F	0

[RE]	u([RE])
T	$-c$(RE)
F	0

RE	PB	p(DB|RE,PB)
T	T	1
T	F	1
F	T	1
F	F	0

[RE]	NA	p(RE|[RE],NA)
T	T	p(RE)
T	F	0
F	T	0
F	F	0

Fig. 2. Unconstraint influence diagram representing the optimal attack problem in the example attack graph from Figure 1 and some of the required data tables.

The example in Figure 2 presents the UID constructed for the attack graph in Figure 1 and a few data tables. The labels in the nodes abbreviate the full

labels from Figure 1 and we distinguish the decision and chance variables by "[]" around the decision variables in the tables. The top left table belongs to node U and says that if the database is accessed, the reward is high (10^6), but if it is not accessed there is no reward for the attacker. The variant of the top right table is present in each decision node. In this case, it says that for executing the remote exploit action, the attacker pays $c(RE)$ and if he/she decides not to use the action, no cost is incurred. The other two tables represent a sample OR and AND node conditional probability tables and the latter includes also the probability of success of an attack action $p(RE)$.

5 Scalability Experiments

We have implemented automated conversion from the output of MulVAL attack graph generator [6] to an UID and solved the UID using the Guido solver [10]. We have generated the attack graphs for one small real-world network and a few meaningful network topologies. The real-world network is composed of one router, two desktop computers and one android mobile phone. All four machines are directly accessible form the Internet. The network was scanned by the Nessus scanner, which found 1, 1, 2 and 0 vulnerabilities on the machines and the top level goal was to execute code on the router. MulVAL generated an attack graph with 12 AND nodes, 8 OR nodes and 11 facts. One edge had to be removed to break a cycle. The optimal attack strategy for this graph has been computed in approximately 3 seconds.

Further attack graphs have been synthetically generated from three network topologies, which are inspired by real world networks and allow gradually increasing the problem size. The *Local* network is the network from Figure 1 without any vulnerability on the firewall and with 1 to 6 vulnerabilities on both the workstation and the target DB server. The *Local+* network is similar (Figure 3, left), but it includes one public web server in front of the firewall, which has one remote vulnerability and can access the database. We were adding 1 to 6 workstations with one remote client vulnerability on each. The *Chain* network is the other network in Figure 3 with one vulnerability per machine, with 0 to 5 copies of the middle segment and nodes reachable only along direct lines. The probabilities of success of attack actions were randomly selected between 0.5 and 1, and their costs were random integers between 0 and 10. The results

Fig. 3. Sample network topologies used for experimentation: *Local+* on the left and *Chain* on the right.

Size	Local		Local+		Chain	
	AND	OR	AND	OR	AND	OR
1	5	5	8	6	8	7
2	7	5	12	8	12	11
3	9	5	16	10	16	15
4	11	5	20	12	20	19
5	13	5	24	14	24	23
6	15	5	28	16	28	27

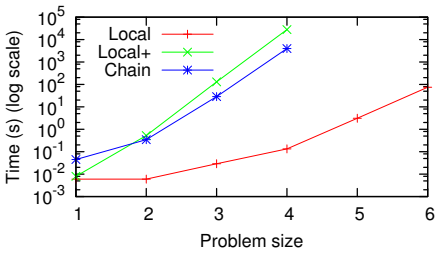


Fig. 4. The number of different kinds of nodes in the attack graphs generated for the sample networks and the time required to solve the corresponding UIDs.

are summarized in Figure 4. The table presents the number of AND and OR nodes in the generated attack graphs for individual problem sizes and the graph presents the computational time needed to compute the optimal solution. The simple solver we used is able to solve attack graphs with up to 20 attack actions in order of hours on a single core of Intel i7 3.2GHz with 20GB RAM.

6 Conclusion

In this paper we make a link between two closely related lines of research that has been developing independently: attack graph analysis in computer network security and probabilistic graphical models in artificial intelligence. We point out that this link allows reusing existing theoretic and algorithmic results from AI for better understanding of the problem of computing the optimal attack strategy in an attack graph. Many variants of this problem have been shown to be NP-hard and algorithms have been developed for solving the hard as well as the simpler variants. In particular, we present how the most general variant of the problem can be reduced to the problem of solving an unconstrained influence diagram and existing software package can be used to solve it. Our experimental evaluation shows that this approach scales to attack graphs with up to 20 actions with the basic solver we used. This size is sufficient only for very small or strongly abstracted real-world networks, but we believe the existing and future advanced UID solution methods and specialized solvers inspired by the UID algorithms may scale to larger problems.

Acknowledgements. The authors would like to thank Xinming Ou and Su Zhang for sharing their MulVAL input generator. This research was supported by the Office of Naval Research Global (grant no. N62909-12-1-7019), the Grant Agency of the Czech Technical University in Prague (grant no. OHK3-060/12), and the Czech Science Foundation (grant no. P202/12/2054).

References

1. Buldas, A., Stepanenko, R.: Upper bounds for adversaries utility in attack trees. In Grossklags, J., Walrand, J., eds.: Decision and Game Theory for Security. Volume 7638 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 98–117
2. Sarraute, C., Richarte, G., Lucángeli Obes, J.: An algorithm to find optimal attack paths in nondeterministic scenarios. In: Proceedings of the 4th ACM workshop on Security and artificial intelligence. AISec '11, New York, NY, USA, ACM (2011) 71–80
3. Greiner, R., Hayward, R., Jankowska, M., Molloy, M.: Finding optimal satisficing strategies for and-or trees. Artificial Intelligence **170**(1) (2006) 19–58
4. Jensen, F.V., Vomlelová, M.: Unconstrained influence diagrams. In: Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence. UAI'02, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2002) 234–241
5. Ingols, K., Lippmann, R., Piwowarski, K.: Practical attack graph generation for network defense. In: Proceedings of the 22nd Annual Computer Security Applications Conference. ACSAC '06, Washington, DC, USA, IEEE Computer Society (2006) 121–130
6. Ou, X., Boyer, W.F., McQueen, M.A.: A scalable approach to attack graph generation. In: Proceedings of the 13th ACM conference on Computer and communications security. CCS '06, New York, NY, USA, ACM (2006) 336–345
7. Wang, L., Islam, T., Long, T., Singhal, A., Jajodia, S.: An attack graph-based probabilistic security metric. In Atluri, V., ed.: Data and Applications Security XXII. Volume 5094 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2008) 283–296
8. Homer, J., Ou, X., Schmidt, D.: A sound and practical approach to quantifying security risk in enterprise networks. Technical report, Kansas State University, Computing and Information Sciences Department (2009)
9. Luque, M., Nielsen, T.D., Jensen, F.V.: An anytime algorithm for evaluating unconstrained influence diagrams. In: Proc. 4th European Workshop on Probabilistic Graphical Models. (2008) 177–184
10. Isa, J., Lisy, V., Reitermanova, Z., Sykora, O.: Unconstrained influence diagram solver: Guido. In: 19th IEEE International Conference on Tools with Artificial Intelligence, 2007. ICTAI 2007. Volume 1., IEEE Computer Society (2007) 24–27
11. Iša, J., Reitermanová, Z., Sýkora, O.: On the complexity of general solution dags. In: Proceedings of the 2009 International Conference on Machine Learning and Applications. ICMLA '09, Washington, DC, USA, IEEE Computer Society (2009) 673–678