Computers & Security 126 (2023) 103081

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Attack graph analysis: An explanatory guide

Kengo Zenitani

Tokyo Institute of Technology, 2-12-2, Ookayama, Meguro-ku, Tokyo, 152-8552, Japan

ARTICLE INFO

Article history:
Received 11 July 2021
Revised 16 September 2022
Accepted 24 December 2022
Available online 28 December 2022

Keywords:
Attack graph
Exploit dependency graph
Cycle handling
Network security metrics
Network hardening
Bayesian attack graph

ABSTRACT

Attack graph analysis is a model-based approach for network-security analysis. It analyzes a directed graph called an attack graph. Usually, each node in it corresponds to a malicious event caused by attackers, and the edges correspond to the causal relations between events. We can obtain an attack graph from the network topology, its configuration, and the distribution of vulnerabilities. An attack graph gives us various information relevant to network security. Also, there are several relevant algorithms to find desirable security controls applicable to the network. Over twenty years of research have made much progress in this field. However, it comprises a breadth of definitions and discussions, and it is difficult for people new to this field to comprehend the key ideas. This article aims to briefly introduce this method to prospective researchers by summarizing their progress by selecting and reviewing foundational studies. We elaborate on the essential concepts, such as exploit dependency, AND/OR graph, monotonicity, and cycle handling.

1. Introduction

Our digitized economy depends on computer networks composed of thousands of millions of machines across the world. We need a robust, reliable approach to understand the risks tied to this complex, vast infrastructure. Attack graph analysis is one of the attempts to computerize those risk analyses.

Attack graph analysis is a model-based approach for network-security analysis. It analyzes a directed graph called an attack graph. Usually, each node in an attack graph corresponds to a malicious event caused by attackers, and the edges correspond to the causal relations between events. We can obtain an attack graph from the network topology, its configuration, and the distribution of vulnerabilities. It is a merged set of attack paths, each of which is a sequence of steps an attacker could take to compromise the concerned network. Various researchers have proposed network security metrics based on attack graphs. Also, several algorithms exist searching within them for the optimal combination of security controls. This broad applicability spanning from risk analysis to risk control is a unique strength of attack graph analysis.

Phillips and Swiler (1998) proposed the first comprehensive framework of attack graph analysis. Much progress has been made since then. It includes the compact representation of attack graphs and monotonicity assumption, the two types of node combinations, hierarchical visualization technique, a mapping to propositional logic, approximated optimization algorithm for security controls, and more. The research history emphasizes several issues, too, such as scalability, the difficulty in metrics design, and handling of cyclic structures. This field still requires further effort.

Behind this progress, there have been various definitions of attack graphs. At an early age, it was defined as finite automata, called *state enumeration graphs* (Jha et al., 2002; Phillips and Swiler, 1998; Swiler et al., 2001). Later, the definition as cyclic directed AND/OR graphs, called *exploit dependency graphs* (Ammann et al., 2002; Jajodia et al., 2005; Ou et al., 2006), was explored. Now, several researchers generalize it as a special case of Bayesian networks called *Bayesian attack graphs* (Frigault and Wang, 2008; Muñoz-González et al., 2019; 2017). One can find even more different definitions. Each of them is still in use by many researchers, and this situation makes it difficult for people new to this field to comprehend past achievements.

Many surveys focus on attack graphs. To exemplify a few, Hong et al. (2017) provide a broad survey of graphical security models, including various forms of attack graphs. The survey by Kaynar (2016) focuses on the attack graph generation process and representative definitions. Zeng et al. (2019) mainly summarize the application studies of existing well-established theories, such as Markov models, to attack graph analysis. Unfortunately, none provides an in-depth description of the foundational concepts commonly used in attack graph analysis and is suitable for beginners.

This article aims to summarize the foundational concepts in attack graph analysis for prospective researchers. We will focus on foundational studies and capture their major contributions. Most

E-mail address: zenitani.k.aa@m.titech.ac.jp

https://doi.org/10.1016/j.cose.2022.103081
0167-4048/

of those selected studies satisfy at least one of the following criteria: (1) relatively well-cited and (2) focusing on exploit dependency graphs since it is the most well-studied form in this field. We also discuss Bayesian attack graphs because they generalize exploit dependency graphs with substantial progress in their expressiveness. The crucial concepts are elaborated, and we will discuss the dependencies among them. It is not our scope to encompass the past and recent studies as much as possible. The emphasis is on the brief, concrete understanding of the essential ideas. This approach provides a sound basis for further research.

This article summarizes five topics; foundation, visualization, network security metrics, network hardening, and Bayesian attack-graph. Section 2 introduces the foundational concepts of attack graph analysis. It also provides the definition of relevant terms and an efficient, concise algorithm for attack-graph generation. Section 3 refers to several visualization techniques for an attack graph. In general, an attack graph contains too many nodes and edges and is difficult to comprehend, and visualization techniques help. We will see attack-graph based network security metrics in Section 4. It is an indicator summarizing the level of risk implied by an attack graph. Theoretical issues in practical metrics design will be pointed out. Section 5 is dedicated to network hardening, i.e., combinatorial optimization of security controls applicable to the concerned network. While combinatorial optimization is known to be computationally expensive, an approximated optimization algorithm computationally efficient and sufficiently close to optimal is known. Section 6 examines the Bayesian attack graph. It can leverage Bayesian networks' powerful expressiveness to handle the complex dependencies among exploits and observed facts such as detected intrusions. Finally, we will sum up our discussion.

2. Foundation

In this section, we summarize the concept of attack graph analysis and the definition of attack graphs. This part provides the foundational understanding of attack graph analysis and its key components.

2.1. Conceptual framework

Phillips and Swiler (1998) proposed the first comprehensive framework of attack graph analysis. Their framework enumerates the following inputs for attack graph generation:

- Network configuration: A list of hosts and the connection structure among them. Software configuration on each host is also included.
- Vulnerability distribution: A list of pairs of a vulnerability and a host. Each pair denotes that the host has the paired vulnerability.
- Threat: Entity, such as cybercriminals and malicious insiders, exploits vulnerabilities distributed over the network.
- Attack template: Descriptions of the way the threat exploits vulnerabilities to accomplish adversarial results.

This set of information enables us to simulate the threats' behavior. For instance, we can imagine that an attacker (Threat) connects to a gateway server with a known vulnerability (Network configuration and Vulnerability distribution) and then exploits the vulnerability to penetrate further (Attack template). Similarly, if we know that the gateway connects to another vulnerable server, it is expected for the attacker to compromise the server as a next step. Appropriately formalized input enables algorithms to automate these inferences. The repetition of step-by-step automated inference forms a chain of exploits. An attack graph is a graph structure given by merging those chains of exploits. It means that we can extract each chain of exploits from an attack graph. We call a chain of exploits extracted from an attack graph an attack path. We will define these terms more precisely later.

Phillips and Swiler (1998) referred to fault-tree analysis as a comparison. A fault tree also gives us the logically inducible attack patterns. However, they pointed out that "fault trees don't model cycles (such as an attacker starting at one machine, hopping to two others, returning to the original host, and starting in another direction at a higher privilege level)." That is, attack graphs are intended to allow such cyclic structures.

Once an attack graph is generated, it tells a lot about the security of the concerned network (See Fig. 2.1). For example, the number of attack paths on the graph shows the availability of attack options for the attacker. If the minimum length of attack paths increases, it also raises the bottom line of the attacker's required effort. An exploit being shared between many attack paths might correspond to an effective target for security patching. An attack graph is a rich source of these practical implications. As a whole, the generation and analysis of an attack graph have the potential for computerized assistance to cybersecurity management.

2.2. Attack graph representation

The attack graph generation algorithm needs to be computationally efficient. In its early history, attack graph generation algorithms got computationally exploded easily. The leading cause resided in the form of attack graph representation.

Suppose we have N computers on the concerned network, each computer has M vulnerabilities, and each vulnerability has a state expressing whether it is exploited or not. Under this setting, the number of possible states of the whole network is 2^{MN}. *State enumeration graph* is a representation form of attack graphs in which each node corresponds to a possible state in that 2^{MN} options, and each edge represents an exploit, i.e., an attacker's action to cause a state transition. The size of a state enumeration graph is intractable even for a small-sized network with tens of machines. The earliest attempts for attack graph analysis, such as Swiler et al. (2001) and Ritchey and Ammann (2000), adopted this form and got stacked up in the scalability problem.

The scalability problem was later solved by *exploit-dependency graph*, another representation form of attack graphs. An exploit-dependency graph is a directed graph composed of two types of nodes; *exploit nodes* and *condition nodes*. Edges run from condition nodes to exploits nodes or from exploit nodes to condition nodes. There are no edges directly connecting exploits to exploits or conditions to conditions, i.e., an exploit-dependency graph is a bipartite graph. Semantically, a condition node denotes a fact such as a specific host is reachable from the Internet, the host has a vulnerability, and an attacker compromises the host. Here, let us name each condition node as $c_{\text{reachable}}$, $c_{\text{vulnerable}}$, and $c_{\text{compromised}}$, respectively. An exploit node represents the causal relationship among these conditions. Suppose we have an exploit node e, and three edges $c_{\text{reachable}} \to e$, $c_{\text{vulnerable}} \to e$, and $e \to c_{\text{compromised}}$. Then, the small subgraph in Fig. 2.2 implies that the two conditions $c_{\text{reachable}}$ and $c_{\text{vulnerable}}$ are the required *preconditions* for the exploit e to realize the *postcondition* $c_{\text{compromised}}$. A postcondition could be a precondition for another exploit. We can interpret an exploit node as a description of AND combination of its preconditions, and a condition node corresponds to an OR combination of its parent exploits. In this way, an exploit-dependency graph represents the dependency relationships among conditions relevant to the concerned network's security (See Fig. 2.3).

The concept of exploit dependency is introduced by Ammann et al. (2002), and Jajodia et al. (2005) organized it as the use of two types of nodes; condition and exploit nodes. The following definition is an adapted version of Wang et al. (2006b) and is a widely accepted form of attack graphs by various researchers:

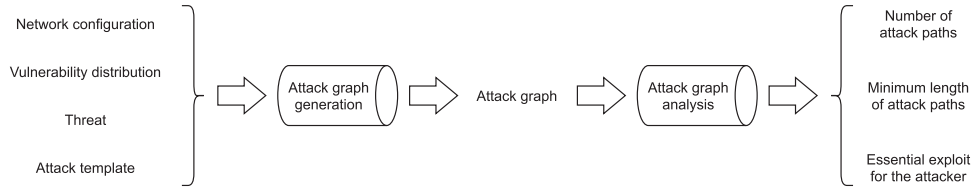


Fig. 2.1. The process of attack graph analysis.

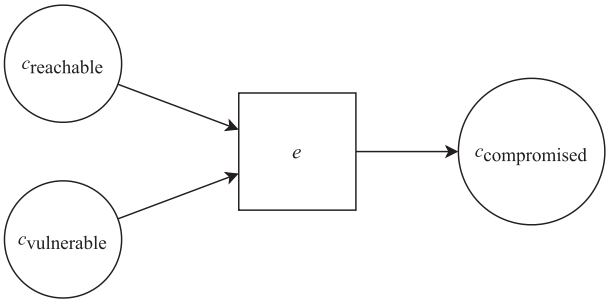


Fig. 2.2. A minimal example of attack graph.

Definition 1 (Attack graph)**.** Given a set of *exploits* E, a set of *security conditions* C where $E \cap C = \emptyset$, a *require*-relation $R_r \subseteq C \times E$, and an *imply*-relation $R_i \subseteq E \times C$, an *attack graph* G is the connected, directed graph $G = (E \cup C, R_r \cup R_i)$, where $E \cup C$ is the node and $R_r \cup R_i$ is the edge set. For each attack graph $G = (E \cup C, R_r \cup R_i)$, *initial conditions* refer to the subset $C_i \subseteq C$ composed of every condition $c \in C$ satisfying $\nexists e \in E$ such that $(e, c) \in R_i$, i.e., the set of all the leaves in the graph, whereas *intermediate conditions* refer to the subset $C \setminus C_i$.

Take a look at Fig. 2.3. In this attack graph, condition nodes are depicted by circles, and exploit nodes are in boxes. The conditions $\{c_7, c_8, c_9\}$ are the intermediate conditions, and others are the initial conditions. Also, the graph contains cyclic structure comprises e_3, c_9, e_4, and c_8. This composition shows the resemblance to and the difference from fault trees.

The major difference between exploit-dependency graphs and state-enumeration graphs is the meaning of nodes. In state-enumeration graphs, each node represents a network state, which is a combination of states of all the hosts in the network. In contrast, a node in exploit-dependency graphs holds far more limited information, such as one host connects another and whether a vulnerability is exploited or not. This difference makes exploit-dependency graphs substantially smaller than state-enumeration graphs. In general, the number of nodes in an exploit-dependency graph is $O(N^2 + NM)$ compared to $O(2^{MN})$ in a state-enumeration graph, where N is the number of machines, and M is the number of vulnerabilities each machine has.

2.3. Attack template representation

The definition of attack graphs lacks the detail of how we can describe the conditions and exploits in a practical manner. Ou et al. (2005) proposed to use Datalog (Ceri et al., 1989) for this purpose, which is a restricted but simple logic programming language.

A Datalog program is a set of *facts* and *rules*. Both facts and rules are composed of *terms*. A term is a *predicate* symbol followed by a list of *arguments*. An argument is either a *constant* or a *variable*. In this section, constants are denoted by lowercased names and variables by capitalized names. For example, "`connectsTo(gw, H)`" is a term with predicate "`connectsTo`" with a constant "`gw`" and a variable "H". This term represents a situation that a host named "`gw`" connects to some host referenced by "H".

A *fact* is a term without variables. For example, "`connectsTo(gw, db)`" is a fact. On the other hand, "`connectsTo(gw, H)`" is not a fact since it contains a variable "H". The term without arguments, such as "`attackerExists()`", is also a fact. We can use facts as the representation of conditions in an attack graph. See the following examples:

```
connectsTo(inet, gw).
hasVulnerability(gw, v1).
compromised(gw).
```

Each fact corresponds to $c_{\text{reachable}}$, $c_{\text{vulnerable}}$, and $c_{\text{compromised}}$ in Fig. 2.2, respectively.

A *rule* is a pair of *head* and *body*. A *head* is a term. A *body* is a list of terms. Both are subject to several conditions. For example,

```
compromised(H):- compromised(X), connectsTo(X, H),
hasVulnerability(H, V).
```

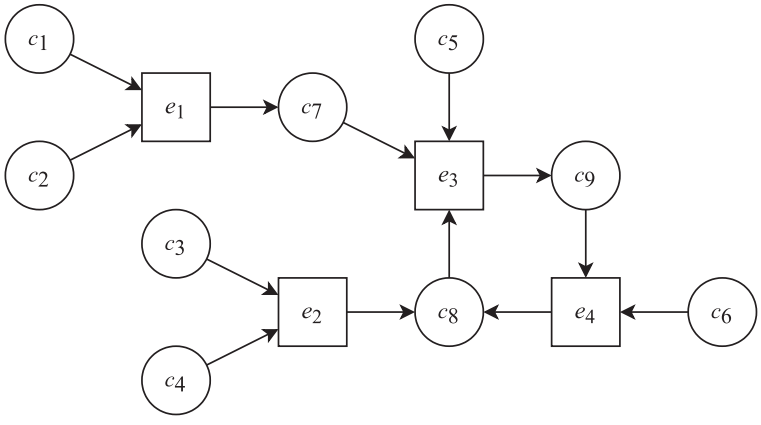


Fig. 2.3. Another small example of attack graph.

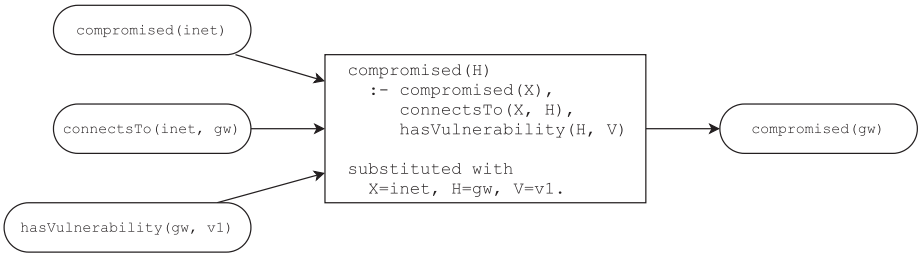


Fig. 2.4. An example of minimal attack graph with Datalog.

is a rule with the head "`compromised(H)`" and the body "`compromised(X), connectsTo(X, H), hasVulnerability(H, V)`". A rule is a conjunctive function of facts, i.e., the fact implied by the head is inferred when every term in the body is presumed or inferred by the program. For the exemplified rule above, it means that we can say that "H" is compromised when all the three conditions, "X" is compromised, "X" connects to "H", and "H" has some vulnerability "V", are satisfied. In other words, the body implies a requirement for a constants substitution for variables "X", "H", and "V" which matches the facts defined in or inferred by the program. To understand what it means, suppose we have a Datalog program containing the following facts:

```
compromised(inet).
connectsTo(inet, gw).
connectsTo(inet, www).
hasVulnerability(gw, v1).
```

For these facts, the substitution, $\texttt{X} = \texttt{inet}$, $\texttt{H} = \texttt{gw}$, and $\texttt{V} = \texttt{v1}$ induces the body "`compromised(inet), connectsTo(inet, gw), hasVulnerable(gw, v1)`". Then, the corresponding head "`compromised(gw)`" is inferred successfully. Similarly, another substitution, $\texttt{X} = \texttt{inet}$, $\texttt{H} = \texttt{www}$, and $\texttt{V} = \texttt{v2}$, induces the body "`compromised(inet), connectsTo(inet, www), hasVulnerability(www, v2)`". However, the term "`hasVulnerability(www, v2)`" is not in the presumed facts. Therefore, this substitution fails, and the corresponding head "`compromised(www)`" cannot be inferred. We call a rule supplied with a successful substitution an *instance*. We can represent attack templates as rules, and exploits as instances in Datalog. Figure 2.4 depicts a Datalog leveraged form of the minimal attack graph in Fig. 2.2.

As Fig. 2.4 shows, Datalog provides rich expressiveness in the attack graph nodes and the attack template representation. Ou et al. (2005) discuss various attack templates written in Datalog. The example in Fig. 2.4 also explains why the size of an exploit-dependency graph has the complexity of $O\left(N^2 + NM\right)$. In the Datalog representation, network connectivities appear as the conditions describing point-to-point connection, such as "`connectsTo(inet, www)`". The number of these conditions has the $O\left(N^2\right)$ complexity at worst. Similarly, the vulnerability distribution appears as conditions like "`hasVulnerability(gw, v1)`". This sort of condition has the $O(NM)$ complexity. As the result, the spatial complexity is assured to be polynomial.

Meanwhile, Ou et al. (2006) give the following definition of attack graphs, which is also popular among the researchers in this field:

Definition 2 (Logical attack graph (adapted from Ou et al., 2006)). $\left(\left(C_r \cup C_p \cup C_d\right), R\right)$ is a logical attack graph, where C_r, C_p and C_d are three sets of disjoint nodes in the graph, and $R \subset \left(C_r \times \left(C_p \cup C_d\right)\right) \cup \left(C_d \times C_r\right)$. C_r, C_p, C_d are the sets of derivation nodes, primitive fact nodes, and derived fact nodes, respectively.

This definition is essentially equivalent to Definitio 1. The set of conditions C in Definitio 1 corresponds to $C_p \cup C_d$. The set of exploits E is to C_r, $C_d \times C_r$ is to R_r, and $C_r \times \left(C_p \cup C_d\right)$ is to R_i, respectively. From the viewpoint of Datalog, C_p is the set of predefined facts in a Datalog program, C_r is the set of instances, and C_d is the inferred facts appearing in the heads of instances in C_r.

2.4. Attack graph generation

Ou et al. (2006) developed a software named MulVAL, a Datalog execution runtime specially designed for the attack graph generation. It is freely available on the Internet and used by many researchers. Here, however, let us see a more simplified algorithm.

The process to generate legitimate instances from a rule and the set of facts is called *unification*. In the context of attack graph generation, the execution of a Datalog program is the repetition of unification. Successful unifications generate new instances, and the heads of them contain new facts. Therefore, repeated unifications grow the set of facts. This process terminates at some point when the facts set get saturated, i.e., it will stop when all the logically inducible facts are identified. After the saturation, the set of predefined facts and the set of facts generated by unification compose an attack graph. Algorithm 1 implements this flow.

Algorithm 1 An algorithm for attack graph generation

Require:
R: the set of rules, equivalent to the attack templates.
F: the set of facts, equivalent to the initial conditions.
Ensure:
C: the set of facts, equivalent to the inferred conditions.
E: the set of instances, equivalent to the inferred exploits.
R_r: the set of require relations.
R_i: the set of imply relations.

$C' \leftarrow F$
$E' \leftarrow \emptyset, R_r \leftarrow \emptyset, R_i \leftarrow \emptyset$
repeat
  $C \leftarrow C'$
  $E \leftarrow E'$
  for $r \in R$ **do**
    for $I \in$ unification (r, C) **do**
      $C' \leftarrow C' \cup \{\text{head}\ (I)\}$
      $E' \leftarrow E' \cup \{I\}$
      $R_r \leftarrow R_r \cup \{(c, I) \mid c \in \text{body}\ (I)\}$
      $R_i \leftarrow R_i \cup \{(I, \text{head}\ (I))\}$
    end for
  end for
until $E = E'$

Algorithm 1 has significant redundancy. In theory, the unification process examines every pair in $r \times 2^C$ although just a few of them are legitimate. However, new instances to be generated always depends on the facts generated just before it. There is a

Datalog execution strategy called *semi-naive evaluation*, which is based on this observation and is computationally efficient. An example program with less than one hundred code lines written in Python is provided as a supplement to this article. The semi-naive evaluation-based algorithm scales almost linearly in the resultant attack graph's size if we can assume that each unification cost is $O(1)$.

Algorithm 1 has an important implication. It grows the set of facts C monotonically, and the set never decreases its size. It means that any condition once achieved by the attacker keeps forever. For example, if the attacker has compromised the gateway, the attack-graph-based analysis supposes the gateway is kept compromised then and after. This assumption of *monotonicity* is introduced by Ammann et al. (2002).

One more thing to note is that Datalog programs cannot accept negation in general. We cannot write a rule such that "if the attacker NOT yet compromised the gateway". Because the rules containing negation make the inference result depend on the order of unification. There are several proposed approaches for the Datalog with negation (See Ceri et al., 1989 for detail).

2.5. Attack paths extraction

An *attack path* is a connected subgraph of an attack graph satisfying the following conditions:

1. It ends at one condition node (called a *goal condition*).
2. Every condition node in it has only one incoming edge at most.
3. Every exploit node in it has all the incoming edges shown in the original attack graph.

An attack path represents a sequence of steps that an attacker must take to achieve the goal condition. As the second condition above implies, the sequence shown by an attack path is without freedom of choice. Therefore, it corresponds to a definite sequence of attack steps.

The extraction of attack paths gives us useful information. The number of attack paths in an attack graph indicates the degree of freedom available to the attacker. The minimum length of attack paths implies the least required effort for the attacker to compromise the network. An attack graph is a compact, combined form of these attack paths. Algorithm 2 enumerates all the attack paths in the given attack graph.

Figure 2.5 shows two attack paths leading to c_9 extracted from the attack graph in Fig. 2.3.

As this example shows, an attack path is not necessarily a chain of nodes. It is a directed acyclic graph. Besides, in general, the number of attack paths in an attack graph could be extremely larger than the size of the attack graph. Thus, Algorithm 2 is not a generally feasible tool to evaluate the security of the concerned network. For network security evaluation, we will discuss more sophisticated ways in Section 4.

Algorithm 2 An algorithm for attack path extraction

Require:
C_g: the set of goal conditions.
R_i: the set of imply relations.
R_r: the set of require relations.
Ensure:
the set of attack paths, each path in which is represented as a set of exploits.

```
function ENUMERATE-ATTACK-PATHS(C_g, R_i, R_r)
    E ← {{e | (e, c) ∈ R_i, c = g} | g ∈ C_g} \ {∅}
    if E = ∅ then
        return {∅}
    else
        R_i' ← {(e, c) | (e, c) ∈ R_i, c ∉ C_g}
        E' ← ∅
        for (e_1, e_2, ...) ∈ ∏_{E∈E} E do
            E' ← {e_1, e_2, ...}
            C_g' ← {c | (c, e) ∈ R_r, e ∈ E'}
            for E'' ∈ ENUMERATE-ATTACK-PATHS(C_g', R_i', R_r) do
                E' ← E' ∪ {E' ∪ E''}
            end for
        end for
        return E'
    end if
end function
```

2.6. Discussions

The explanation above simplifies various aspects to focus on the key concepts. However, the omitted details contain various issues relevant to the practical application of attack graphs. Also, there are different forms of attack graphs other than exploit dependency graphs suitable for specific application scenarios. The following subsections discuss them.

2.6.1. Computational complexity

In general, the computational complexity of the state-of-the-art algorithm for attack graph generation is linear in the resultant attack graph's size. MulVAL is one of the most efficient tools which accomplish that performance. However, an attack graph's size depends on various factors, including the number of hosts, network topology, and attack templates. It is difficult to estimate the size of an attack graph precisely, thus the computational complexity too. For example, if the network has a ring structure, then the number of hosts and the edges are equal. In contrast, if the network forms a complete graph, then the number edges are approximately one-half of the square of the number of hosts. Such a difference in the network size also affects the size of the attack graph to be generated. Nevertheless, in the literature, the computational complexity is usually shown as a function of the number of hosts on a network.

Let us exemplify a few. In Ammann et al. (2002), the authors evaluate the computational complexity of the algorithm to be $O(A^2E)$, where A is the number of attributes (facts) and E is the number of exploits that might be performed. However, this complexity is evaluated to be $O(N^6)$ in Ou et al. (2006), where N is the number of hosts on a network. Because the attribute in Ammann' s algorithm is supposed to express the connectivity among hosts, and if we assume that the network topology is a complete graph, then the total number of attributes is $O(N^2)$. Additionally, in a complete graph, each node can be compromised from every other node. Thus the number of exploits is $O(N^2)$. In short, this evaluation is based on a worst-case scenario. Compared to that, the worst-case computational complexity of MulVAL is evaluated to be $O(N^2)$ in Ou et al. (2006). This estimation comes from the unification characteristics and the maximum number of variables appearing in the rules. Furthermore, the form of attack templates is different from Ammann' s. In another more recent study, Kaynar and Sivrikaya (2016) propose a distributed algorithm for attack graph generation. Their algorithm's computational complexity is evaluated to be $O(N^2/\log(N))$, where N is the number of hosts. It is not the worst-case complexity, and the algorithm processes an intentionally limited form of attack templates. It is inappropriate to compare these different estimations directly. As this paragraph illustrates, we need to clarify each complexity estimation's premises when comparing the algorithms.

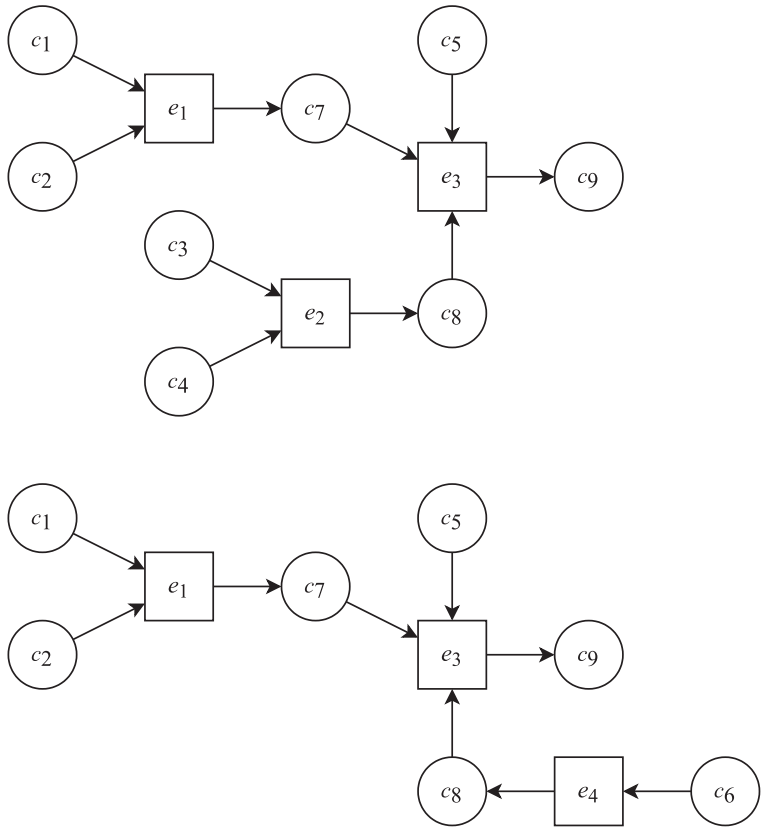


Fig. 2.5. The extracted attack paths.

Broad surveys in the field of attack graph analysis are given in Lippmann and Ingols (2005), Singhal and Ou (2011), Hong et al. (2017). The survey in Lippmann and Ingols (2005) refers to studies before 2005, while important progress was made since then. The description in Hong et al. (2017); Singhal and Ou (2011) refers to more recent efforts. While these surveys are insufficient in considering computational complexities, they are a valuable source of information to examine the progress of attack graph analysis studies, including its performance improvement.

2.6.2. Cyclic structures

The attack graph in Fig. 2.3 contains a cyclic structure—the loop composed of e_3, c_9, e_4, and c_8. This does not mean that the attacker infinitely realizes that cycle. It is a natural consequence of merging overlapping attack paths in Fig. 2.5. Cycles are inherent in attack graphs and could arise even in a network with non-cyclic topology. However, those cycles cause difficulty in network security evaluation and optimization. We will discuss this topic in Sections 4 and 5.

2.6.3. Attack template preparation

The attack graph generation process depends on the set of attack templates defined as inference rules. Each attack template represents the knowledge about how an attacker exploits vulnerabilities. Therefore, we have to maintain the set of those attack templates up-to-date to make our analysis realistic.

The primary source of vulnerability information is National Vulnerability Database (NVD). Roschke et al. (2009) use a rule-based approach to extract information from the textual description in NVD. Later, Aksu et al. (2018) examine machine-learning-based approaches to generate attack templates. They report that the machine-learning method outperforms the simple rule-based approach. Although this research field is still immature, it seems promising to leverage the recent epochal progress in artificial intelligence research.

2.6.4. Reliability of inference

Sommestad and Sandström (2015) have verified the attacks infered by MulVAL. They have performed penetration tests by experts on the real network containing 199 machines and compared the result with generated attack graphs. The prediction accuracy of the attack graph analysis is reported to be poor. “Of the 119 machines MulVAL predicted as impossible to compromise, 34 (29 per cent) were successfully compromised as root. In 3 of these 34 cases, MulVAL reported the machine as reachable but failed to describe a path that included the exploited vulnerability.” The reasons for this mis-analysis are identified as (1) the lack of attack templates for several vulnerabilities and (2) the weakness in the default rules given in Ou (2005). As this report tells, the reliability or credibility of the attacks inferred by attack graphs depends on the facts and rules we prepare.

2.6.5. Attack expressiveness

As mentioned just before, the coverage of inferable attacks depends on the variety of attack templates in use. That is the expressible variety of attacks by attack templates matters. In Datalog, rules represent attack templates, and predicates are their skeleton. Carefully designed predicates enable not only complex attack inference but also impact analysis.

Bacic et al. (2006) expand the default rules given in Ou (2005) to model the network infrastructure, including subnets and routers. Furthermore, their effort includes the modeling of risk aspects such as confidentiality, integrity, and availability. Froh and Henderson (2009) advance that effort to accommodate the dependency between services and their infrastructure.

Table 1
Three major forms of attack graphs.

Type	Space complexity	Feature
State-enumeration graph	$O(2^{MN})$	- A kind of finite automata. - Strong relevance to Markov chains. - Poor scalability.
Exploit-dependency graph	$O(N^2 + MN)$	- A kind of AND/OR graphs. - Strong relevance to Datalog. - Good scalability.
Host-based attack graph	$O(N)$	- A kind of network model. - Detail is specific to each study. - In general, highly scalable.

Stan et al. (2020) propose a suite of Datalog predicates and rules based on the OSI reference model, i.e., their suite can infer the attack happens in the lower layer, such as the data-link layer. Their study also contains a detailed evaluation report of their rules on a physically constructed testbed. Inokuchi et al. (2019) discuss a process model for designing practical sets of predicates, rules and facts.

From the theoretical point of view, Saha (2008) points out that, in some cases, access control modeling requires negation, and it is processible with stratified Datalog. The study contains an efficient incremental algorithm for attack graph update, too. Johnson et al. (2018) propose a domain-specific language named Meta Attack Language (MAL). At a glance, MAL is very different from Datalog. For example, it can define class hierarchies, the negation of initial conditions, and relation types among nodes. However, most of it can be interpreted as a syntax sugar to stratified Datalog.

All the above indicates the powerful expressiveness of Datalog and the importance of the exploit-dependency graph since its AND/OR combinations of nodes are the reflection of the description structure in Datalog.

2.6.6. Other forms of attack graphs

While state enumeration graphs are intractable due to their space complexity, there are many useful theoretical tools for them. They are a kind of finite automata. If we assign probabilities to edges, it becomes Markov chains. In more complex scenarios, we can use various theories like Hidden Markov Model (HMM), Markov Decision Process (MDP), Partially Observable Markov Decision Process (POMDP), and so on. Jha et al. (2002) have clarified the fundamental properties of state enumeration graphs. Historically, this form is also called full attack graph or complete attack graph.

Another popular definition is host-based attack graph. In this form, usually, a node corresponds to a host, a graph describes the network topology, and a node is regarded as a set of its vulnerabilities. Its generation scales well, but the expressiveness is limited; it ignores the dependencies among exploits and vulnerabilities within a host. The studies by Ammann et al. (2005) and Ingols et al. (2006) are the well-cited ones based on this form.

The three variations, state-enumeration graph, exploit-dependency graph, and host-based attack graph, comprise major categories of attack graph forms. Table 1 summarizes the characteristics of these major forms of attack graphs, where N denotes the number of machines in the network and M is the number of vulnerabilities per machine.

One can even find a hybrid of these forms. For example, the Hierarchical Attack Representation Model (HARM) proposed by Hong and Kim (2016), Ge et al. (2017) is a hybrid of host-based attack graph and exploit dependency graph. In short, HARM models the inside of each host as an attack tree and treats the network topology as a general directed graph. This mixture enables scalability in applications and moderately limited but still rich expressiveness.

Each definition has its strengths and weaknesses. The prominent characteristics of exploit dependency graphs are (1) the powerful expressiveness explained just before and (2) the practical scalability compared with state enumeration graphs. However, as we will see in the following sections, it has several obstacles in handling security metrics and optimization. The optimal choice of definitions depends on its use case. This article sticks to the exploit-dependency graph because it is the most well-studied form in the literature.

3. Visualization

Figure 3.1 shows an example of “small” attack graph. How can we grasp the implication about the security risk from Fig. 3.1? It is not uncommon in practice to see an attack graph composed of thousands of nodes. It is impractical for humans to extract the risk implication directly from an attack graph in raw representation.

Several researchers proposed specially tailored visualization techniques for attack graphs. They provide a compact form of attack graph representation and help us comprehend. We take a look at them in this section.

3.1. Information reduction

Wei Li and Vaughn (2006) examine a condition by which we can aggregate a subgraph in an attack graph into a node and restore the original subgraph from an aggregated node. For example, if two exploits share the same preconditions and postconditions, we can merge them into one exploit with new labeling indicating the original two exploits without losing information. This approach makes the detailed structure of an attack graph compact. However, the original attack graph is assumed to be represented in a form called E-graph in which condition nodes are omitted and only exploit nodes are shown.

Noel and Jajodia (2004) propose an intuitive, useful hierarchical aggregation technique. Their approach defines several rules to aggregate condition and exploit nodes in an attack graph. The following are the major aggregation rules:

- Machine: Sets of conditions are aggregated into a machine abstraction if they are all conditions on that machine.
- Exploit set: Sets of exploits are aggregated into those with the same pair of (possibly identical) attacker/victim machines. That is, only exploits between the same pair of machines are aggregated.
- Protection domain: The protection domain abstraction represents a set of machines that have unrestricted access to one another' s vulnerabilities, so that the attack graph is fully connected among them (if one treats exploit sets between a pair of machines as graph edges).

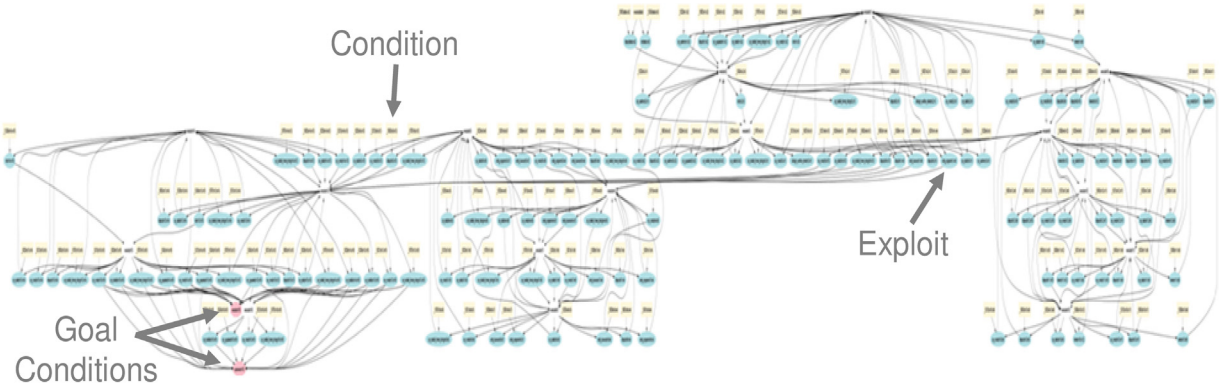


Fig. 3.1. Non-aggregated attack graph (quoted from Noel and Jajodia (2004)).

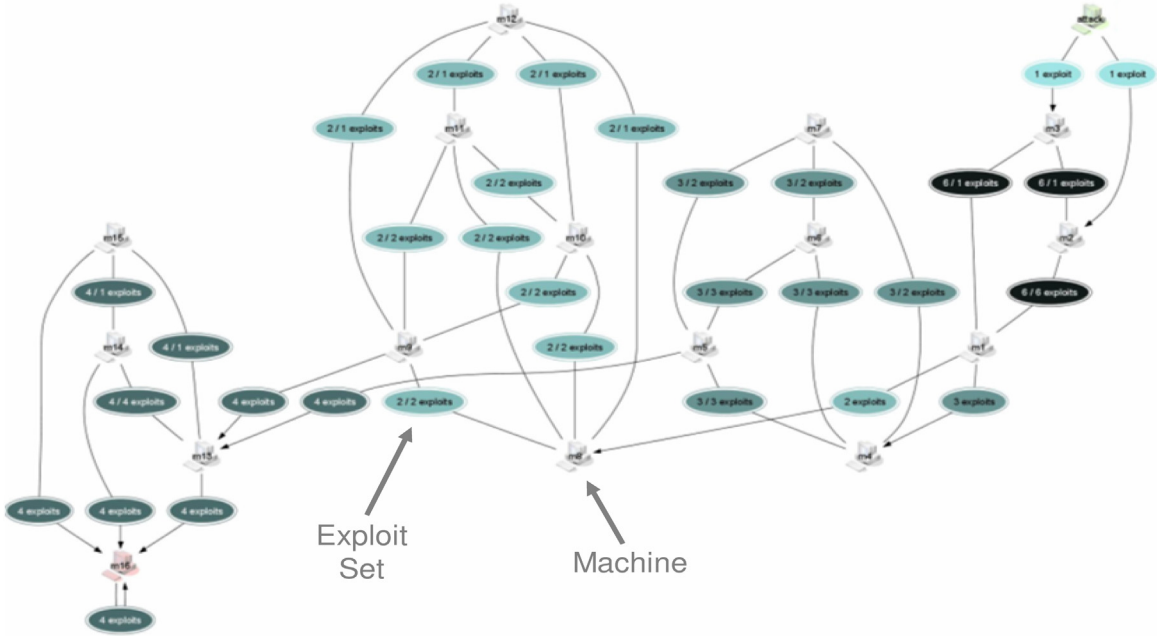


Fig. 3.2. Aggregation to machines and exploit sets (quoted from Noel and Jajodia (2004)).

Figures 3.2 and 3.3 show the example of aggregated forms of Fig. 3.1. The technique dramatically reduces the number of nodes in an attack graph and helps us understand the whole picture.

The aggregation in Figs. 3.2 and 3.3 leverages that every exploit is bound to some machine or subnet in the concerned network. Thus, we can merge the set of exploits bound to the same region into one node in the aggregated attack graph. We can point to the most concerning part of an attack graph and dig into its detail with this aggregated form.

Similarly, Homer et al. (2008b) propose to depict the graph as a network of connected machines and not to show the condition/exploit nodes. The edges in the graph represent abstracted attack paths between machines. Furthermore, "useless" attack paths are omitted that do not provide the privilege escalation valuable to the attacker. Lastly, exploit nodes shared among multiple attack paths are explicitly shown in the graph. Because such exploit nodes are good candidates for applying countermeasures.

3.2. Aspect oriented view

Not every piece of information provided by attack graphs is relevant to the administrator's concern. Irrelevant information could be noisy for comprehension. Visualization methods focusing on specific aspects of attack graphs are useful in such cases.

Noel and Jajodia (2005) propose to use the adjacency matrix of an attack graph to compute the reachability among vertices. The key idea is that the adjacency matrix's multiplication produces another matrix in which the non-zero numbers imply the reachability between vertices on the corresponding graph. Their algorithm computes the transitive closure of the matrix and generates a bitmap image to visualize the connectivity among vertices.

Chu et al. (2010) use treemaps to show the targeted area and the depth of attacks where attack graph nodes are categorized into predefined areas corresponding to subnets. The size and color of rectangles in the treemap is leveraged to express the compromised privilege and the value of compromised assets. This visualization emphasizes the severity of the presumed attack on essential assets.

3.3. Visual syntax

There is no formal standardization for the attack graph representation. Should we depict exploit nodes by squares or by circles? Does the difference in depiction affect the perception by network administrators? Lallie et al. (2020) focus on this topic and clarify the undesirable breadth of variety in visual syntax for attack graph

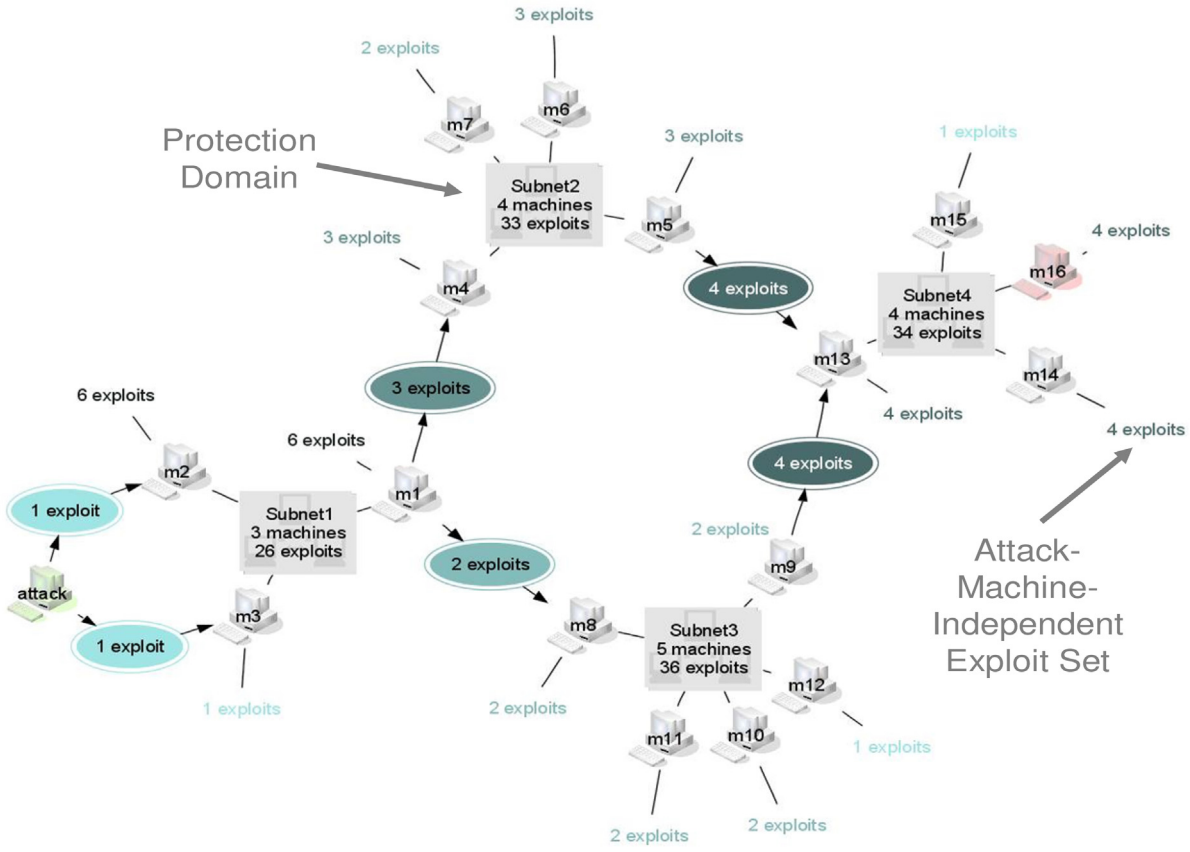


Fig. 3.3. Attack graph aggregated to protection domains (quoted from Noel and Jajodia (2004)).

representation. The need for visual syntax standardization and further research for it is discussed. This survey is also an exhaustive analysis of the past attack graph studies to date. Their previous study (Lallie et al., 2018) examines the perceptual difference between attack graphs and attack trees (fault trees).

4. Network security metrics

An attack graph is too detailed for decision-makers who need to know the whole picture of their IT infrastructure's risk. We can use network security metrics instead. Attack-graph-based network security metrics are concise, numerical indicators expressing the various aspects of the risk landscape.

Here we categorize and explain the attack-graph-based network security metrics. Firstly, we distinguish whether the metric is a probabilistic indicator or not. Secondly, the applicability of the metrics to cyclic attack graphs is concerned. The product of these two categorizations leads to the following four subsections.

Note that the metrics explained below are chosen to provide a comprehensive view of attack-graph-based security metrics design issues and approaches. Even though some of them are specific to a particular form of attack graphs, the idea behind them applies to other cases.

4.1. Non-probabilistic metrics for cyclic graphs

In the most general settings, attack graphs are supposed to be cyclic, and there is no statistical data. In such cases, we cannot calculate probabilities. The metrics reveal qualitative aspects of network security.

The *Network Compromise Percentage (NCP)* in Lippmann et al. (2006) indicates the percentage of network assets an attacker can compromise. This metric shows the range of possible compromise. The *Weakest Adversary (WA)* metric by Pamula et al. (2006) gauges the minimum set of initial conditions required for the attacker to compromise the network. This metric has been introduced as a metric for state-enumeration graphs. We omit the detail here.

Gonda et al. (2017) focus on graph centrality, a measure indicating the relative importance of a node in a graph. For example, the number of incoming and outgoing edges of a node is called a degree centrality. If the centrality gets greater, then the dependence of attack paths going through the node increases. Their study examines several types of graph centralities and compares their relevance to shortest attack paths. Also, their study proposes to convert an exploit dependency graph into a planning graph, which is assured to be acyclic.

Noel and Jajodia (2014) propose a suite of metrics applicable to general attack graphs. Their metrics are categorized into four families composed of eleven metrics in Fig. 4.1. We pick up some of the metrics and dig into the detail.

Let us clarify the implicit assumption for these metrics. All metrics are designed to be within the scale of $[0, 10]$. The larger value implies less security. Also, the network is supposed to be the connected graph of strongly connected components that every host can connect to each other within it. We call each strongly connected component a protection domain (See Section 3.1). A protection domain usually corresponds to a subnet in the network.

Victimization family metrics are the simple aggregation of the count and CVSS scores of the vulnerabilities in an attack graph. For example, the *Exploitability* metric is the average value of the

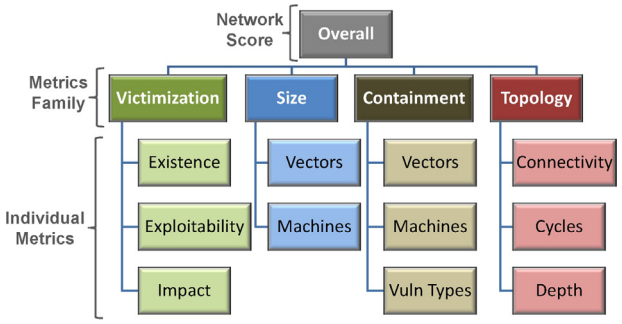


Fig. 4.1. Attack graph metrics families (quoted from Noel and Jajodia (2014)).

CVSS Exploitability score (the relative ease of exploitation), averaged over all vulnerabilities over all hosts. The metric $m_{\text{exploitability}}$ is defined as

$$m_{\text{exploitability}} = \frac{\sum_i^{|U|} \text{CVSS}_{\text{Exploitability}}(u_i)}{|U|},$$

where $\text{CVSS}_{\text{Exploitability}}(u_i)$ denotes the CVSS Exploitability score for vulnerability u_i, and U denotes the set of all vulnerabilies in the attack graph.

The *Size* family metrics measure risk in terms of the attack graph size since the larger graph implies the more ways available to the attacker. The *Attack Vectors* metric in this family is the number of exploits relative to the total possibilities for the network. The exploits are categorized into two groups: implicit and explicit. An explicit exploit is a step of attack performed between different protection domains, and an implicit exploit is a step performed within a protection domain. The metric $m_{\text{attackVectors}}$ is defined as follows:

$$m_{\text{attackVectors}} = 10\sqrt{\frac{v_a}{v_p}},$$

where v_a and v_p are defined as follows:

$$v_a = \sum_i^d (m_i - 1) \sum_j^{m_i} v^{i,j}_{\text{implicit}} + \sum_{i,j}^d v^{i,j}_{\text{explicit}},$$

$$v_p = (m-1)\sum_i^m s_i.$$

In this definition,

- d: the number of protection domains,
- m: the number of machines in the network,
- m_i: the number of machines contained in the ith protection domain,
- $v^{i,j}_{\text{implicit}}$: the number of implicit exploits in the jth machine on the ith protection domain,
- $v^{i,j}_{\text{explicit}}$: the number of explicit exploits across ith and jth protection domains,
- s_i: the number of exploits in the ith machine in the network.

The value v_a means the number of attack vectors, and v_p means the potential number of attack vectors. Note that this definition is based on the assumption that each implicit exploit can be invoked with every other machine in the same protection domain. This assumption introduces the term "$(m_i - 1)$" in the definition. Similarly, in the worst case, every exploit can be invoked by access from every other machine. This is the reason why the term "$(m-1)$" appears on the right-hand side of v_p.

The *Containment* family measures the degree to which the attack graph contains attacks across network protection domains. The *Vulnerability Types* metric is the number of unique vulnerability types referred to by explicit exploits, relative to the total number of vulnerability types across the entire attack graph. This metric implies that multiple instances of the same vulnerability type are less costly to mitigate. The metric $m_{\text{vulnerabilityTypes}}$ is defined as follows:

$$m_{\text{vulnerabilityTypes}} = 10\frac{t_a}{t_a + t_w},$$

where t_a denotes the number of unique vulnerability types referred by explicit exploits, and t_w denotes the number of unique vulnerability types referred by implicit exploits.

The last family of metrics is the *Topology* family. It captures the graph-theoretic property of an attack graph reduced to the protection domain level. The key intuition behind this is that the network services should be kept separate as possible. Suppose we have two vulnerable services on a network. Even in such a situation, the whole network can be secure if there is no access route between the two. The *Cycles* metric in this family is the number of strongly connected components in the domain-level attack graph, relative to the best (most secure) cases possible. In the Cycles metric, strongly connected components are connected by directed edges and might have one or more directed routes in between. The metric m_{cycles} is defined as follows:

$$m_{\text{cycles}} = 10\left(1 - \frac{w_{\text{strong}} - 1}{d - 1}\right),$$

where w_{strong} denotes the number of strongly connected components. Note that the number of domains d corresponds to the best-case, for it corresponds to the situation where no strongly connected domains.

The use of these metrics requires careful consideration. Each metric captures a specific aspect of network security. The improvement in the metrics does not necessarily mean the improvement of security. For example, software patching can reduce the Exploitability metric, while it does not guarantee to eliminate attack paths compromising some critical assets.

4.2. Non-probabilistic metrics for acyclic graphs

The metrics in the previous subsection ignore the attack detail. Suppose we have two attack graphs with the same graph structure. The difference exists only in the CVSS scores of the vulnerabilities contained in them. Then, most of the metrics show the same scores. One possible way to overcome this weakness is to enumerate attack paths and then examine those details. It means that we will see an attack graph as a combined set of attack paths, i.e., acyclic subgraphs.

4.2.1. Combination of major metrics and assistive metrics

Idika and Bhargava (2012) propose to use multiple metrics derived from attack paths. They define two categories of metrics: Decision metrics and Assistive metrics. Most of the metrics are calculated from the extracted attack paths. Let us see the detail of their approach.

As preparation, let G be the attack graph concerned, n be the number of attack paths in G, each p_k for $k = 1, 2, \ldots, n$ be the attack path extracted from G, and the function $l : P \to \mathbb{N}$ be the length of the given attack path.

The *Decision* metrics are composed of five metrics: Shortest Path, Number of Paths, Normalized Mean of Path Lengths, Network Compromise Percentage, and Weakest Adversary metric.

The *Shortest Path (SP), Number of Paths (NP), Normalized Mean of Path Lengths (NMPL)* metrics are defined as follows:

$$SP(G) = \min\{l(p_1), l(p_2), \ldots, l(p_n)\},$$
$$NP(G) = n, \text{ and}$$

$$NMPL(G) = \frac{\sum_i l(p_i)}{NP(G)^2}.$$

The definition of NMPL comes from the observation that a simple averaged length of paths fails to consider the number of paths. Suppose we have two attack graphs, each of which has the same averaged length of paths while the number of paths is considerably different. It is inappropriate to think that the risk of both is the same. The additional division by $NP(G)$ metric is intended to reflect the difference.

The calculation and comparison of Decision metrics tell us some aspects of the security of the network. But, how should we determine which of G_1 and G_2 is more secure when both of $SP(G_1) \le SP(G_2)$ and $NP(G_1) \le NP(G_2)$ hold? In that case, Assistive metrics solve those conflicts.

The *Assistive* metrics are composed of four metrics: *Mean of Path Lengths (MPL), Standard Deviation of Path Lengths (SDPL), Mode of Path Lengths (MoPL), and Median of Path Lengths (MePL)* metric. These are the standard set of well-known statistics of the path lengths.

In conjunction with the Decision and the Assistive metrics, the algorithm by Idika and Bhargava (2012) defines a decision flow to determine which of G_1 and G_2 is better or worse. The final comparison result is either "all are equal", "strictly dominated", "majority dominated", or "incomparable". Their study reports that 99.9% of the experimental cases are concluded to be other than "incomparable". Within the result, 99% of the cases are concluded only with the comparison of Decision metrics. Only 0.9% of cases resort to Assistive metrics. This approach implies that the insufficiency of a suite of metrics can be complemented with the composite use of additional metrics such as Assistive metrics.

4.2.2. Composition of metrics for acyclic subgraphs

An attack path can also be a subgraph of another attack path. In general, an attack path is a composition of its sub-paths. This fact brings a question of how we can define a consistent metric for the composite attack path if we have metrics defined for its sub-paths. Wang et al. (2007a,b) propose the following three principles for security metrics:

1. A longer path leading to the attack goal means better security.
2. A metric should never yield a value that is greater than the smallest attacking effort required for reaching the attack goal.
3. Multiple attack paths together are less secure than any of the paths alone.

Starting from the natural, convincing principles above, let us define a metric named *attack resistance* of node x in an attack graph $G = (E \cup C, R_r \cup R_i)$; remind that E denotes the set of exploit nodes, and C denotes the set of condition nodes. An attack resistance represents the difficulty or effort for the attacker to realize the condition/exploit of the given node. Thus, the larger the resistance, the more secure the network is. Let us introduce several functions as preparation. The "parent" function is the mapping from a node in an attack graph to the set of parent nodes and is defined as follows:

$$\text{parent}(x) = \begin{cases} \{c \mid (c,x) \in R_r\} & (x \in E) \\ \{e \mid (e,x) \in R_i\} & (x \in C) \end{cases}.$$

Let $\mathbb{D}$ be a *domain* with total ordering, such as $\mathbb{N}$ and $\mathbb{R}$. The *choice operator* $\oplus$ is an arbitrary function $\oplus : \mathbb{D} \times \mathbb{D} \to \mathbb{D}$ satisfying that $r_1 \oplus r_2 \le \min\{r_1, r_2\}$ for any $r_1, r_2 \in \mathbb{D}$, which denotes the composition of paths at a condition node. Its characteristic also corresponds to the second principle above. The *bind operator* $\otimes$ is an arbitrary function $\otimes : \mathbb{D} \times \mathbb{D} \to \mathbb{D}$ satisfying that $\max\{r_1, r_2\} \le r_1 \otimes r_2$, which denotes the composition of paths at an exploit node. It corresponds to the third principle. The *individual resistance* of node x is a function $\rho : (E \cup C) \times \mathbb{D} \to \mathbb{D}$ satisfying that $r \le \rho(x, r)$ for any combination of $x \in E \cup C$ and $r \in \mathbb{D}$, corresponding to the first principle. With these sub-functions, now we define the attack resistance $R : (E \cup C) \to \mathbb{D}$ as follows:

$$R(x) = \begin{cases} \rho\left(x, \bigoplus_{e \in \text{parent}(x)} R(e)\right) = \rho\left(x, R(e_1) \oplus \cdots \oplus R\left(e_{|\text{parent}(x)|}\right)\right) & (x \in C) \\ \rho\left(x, \bigotimes_{c \in \text{parent}(x)} R(c)\right) = \rho\left(x, R(c_1) \otimes \cdots \otimes \left(c_{|\text{parent}(x)|}\right)\right) & (x \in E) \end{cases}.$$

(This definition is adapted from that in Wang et al., 2007a; Wang et al., 2007b).

The above definition of attack resistance has some freedom in defining sub-functions, such as choice and bind operator and the individual resistance. It suffices for each sub-function to follow the constraint introduced above. As long as the constraints are observed, the derived attack resistance is guaranteed to follow the three principles. For example, the two operators, $r_1 \oplus r_2 = \min\{r_1, r_2\}$ and $r_1 \otimes r_2 = \max\{r_1, r_2\}$, are compliant with the constraints. In another formulation, $r_1 \oplus r_2 = \frac{r_1 r_2}{r_1 + r_2}$ and $r_1 \otimes r_2 = r_1 + r_2$ are also legitimate definition, and is analogous to the resistance in electric circuits. The definition discussed above is rather a framework for designing the specific metric compliant with the natural principles.

4.3. Probabilistic metrics for cyclic graphs

We can apply the probabilistic graphical model, such as Markov chain and Bayesian network, to attack graphs and calculate the various probability assigned to the nodes. The Markov chain model applies to cyclic graphs. However, the existence of AND combination of preconditions inhibits the simple application of the theory. In this subsection, we discuss this issue and a limited solution for it.

4.3.1. State rank for state-enumeration graph

Mehta et al. (2006) propose a metric inspired by the PageRank algorithm by Google. Their metric defines the rank of states in a state enumration graph. A state enumeration graph G is a 3-tuple (S, τ, s_0), where S is the set of states, $\tau \subseteq S \times S$ is the set of transitions, and $s_0 \in S$ is the initial state. A probabilistic attack graph G' is a 3-tuple $(S, P, \boldsymbol{s})$, where $|S| = n$ for some $n \in \mathbb{N}$, P is a state transition matrix of size $n \times n$ in Markov chain, and $\boldsymbol{s} = (s_1, s_2, \ldots, s_n)^\top \in [0,1]^n$ is the initial state vector. Note that the content of P and $\boldsymbol{s}$ reflects that of τ and s_0. For the probabilistic attack graph, we can calculate the vector $\boldsymbol{r}^{(m)}$ as follows, which denotes the set of probabilities of each state to be activated after the m-steps of transitions:

$$\boldsymbol{r}^{(m)} = \sum_{n=0}^{m} P^n \boldsymbol{s}.$$

This equation derives from the property of Markov chain. In addition, we assume that the number of steps that the attacker takes to perform attacks follows a geometric distribution with the "failure" probability η on each trial. With these settings, we get the following formula which computes the reachability probability vector $\boldsymbol{r} = (r_1, \ldots, r_n)^\top \in [0,1]^n$ as follows:

$$\boldsymbol{r} = \frac{1-\eta}{\eta} \sum_{m=1}^{\infty} \eta^m \sum_{n=0}^{m} P^n \boldsymbol{s}.$$

Each number in $\boldsymbol{r}$ denotes the probability, i.e., the rank, of the corresponding state to be reached by the attacker.

The formulation above is straightforward, and the Markov chain model applies without problems. However, state enumeration graphs are intractable for its state explosion. We need a method applicable to exploit-dependency based attack graphs.

4.3.2. Asset rank for exploit-dependency graph

In exploit dependency graphs, an exploit requires a set of preconditions to be realized simultaneously. The Markov chain cannot handle such a simultaneous activation of states. We need a twist to handle those AND combinations of preconditions.

Sawilla and Ou (2008) propose a probability-like metric named AssetRank, which can handle exploit dependency graphs. AssetRank is a vector in $[0, 1]^n$. Each of the values represents the preference for an attack graph node by the attacker. The higher value means that the attacker is eager to achieve the corresponding exploit or condition. In a broad sense, we can regard each value in an AssetRank as a proxy indicator of the reachability probability by the attacker to the corresponding node.

The key intuition behind AssetRank calculation is the use of multi-agent simulation. Suppose we have an attack graph with a goal condition, and the attacker tries to get that condition. We put an agent on this goal initially. This agent can clone and distribute itself to the adjacent nodes. The cloned, distributed agents have the same ability. We assume agents' distribution propagate from postconditions to its source exploits and from exploits to its preconditions. Thus, the agents spread over the graph as time goes on. Along with the process, some of the agents die out at each node at each step. We call the death rate of agents per step as damping factors. We use the damping factors assigned to nodes to express the difference between AND and OR combinations of conditions. Also, at each step, we feed a fixed number of agents continuously at the goal condition. After the passage of a sufficiently long time, each node is supposed to hold a stable number of agents. When the count of agents at a node is high, it implies that the node is a better place for the agents to stay, i.e., preferable. The numbers of agents at each node normalized to be within $[0, 1]$ compose the AssetRank. See Sawilla and Ou (2008) to confirm the detail of the definition.

AssetRank is similar to the state rank defined in the previous subsection in the sense that both approaches regard an attack graph as a kind of state transition model. The computation of AssetRank is based on the concept of multi-agent simulation, while the latter uses the Markov chain model. Nevertheless, the resultant ranks have similar semantics. The higher the rank, the attacker is likely to achieve the corresponding condition or exploit. Though we cannot use AssetRank as the actual probabilities, we can still compare the probabilistic degree of nodes to be compromised.

4.4. Probabilistic metrics for acyclic graphs

If the concerned attack graph is acyclic, we can apply a more straightforward approach to calculate the probability assigned to each node.

Let G be an attack graph $G = (E \cup C, R_r \cup R_i)$, and "parent" be the function defined in Section 4.2.2. Besides, let us define a function $p : (E \cup C) \to [0, 1]$, which assign individual scores to each node in G. With these functions, we define the probability P of the event implied by an exploit e or a condition c in G as follows (Be cautious in distinguishing the use of P and p, and $P(e)$ means P(event e occurs)):

$$P(e) = p(e) \prod_{c \in \text{parent}(e)} P(c), \text{ and}$$

$$P(c) = \begin{cases} p(c) & (\text{parent}(c) = \emptyset) \\ p(c)\left\{1 - \prod_{e \in \text{parent}(c)} (1 - P(e))\right\} & (\text{parent}(c) \neq \emptyset) \end{cases}.$$

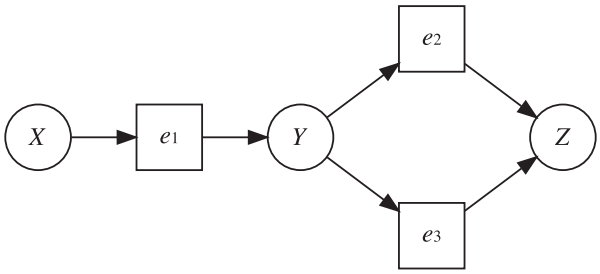


Fig. 4.2. An example of non-independent scenarios.

This definition follows Wang et al. (2008) and is intuitive and natural. Essentially, this form of probability follows the framework in Wang et al. (2007b).

Unfortunately, the probability calculations based on the above definition do not apply to cyclic attack graphs, as the cyclic structure leads to an infinite loop. Another important caveat is that it is implicitly assumed that the events represented by nodes are independent of each other.

In Fig. 4.2, suppose every individual score, such as $p(X)$ and $p(e_2)$, equals to 0.5. Then, the probability $P(Z)$ is calculated as follows:

$$P(Y) = p(Y)P(e_1) = p(Y)p(e_1)P(X) = 0.5^3,$$

$$P(e_2) = p(e_2)P(Y) = 0.5^4,$$
$$P(e_3) = p(e_3)P(Y) = 0.5^4,$$

$$P(Z) = p(Z)\{1 - (1 - P(e_2))(1 - P(e_3))\} \simeq 0.0605.$$

However, two attack paths reach Z via e_2 and e_3, and they are identical until halfway through. When one of the two occurs, the other is also likely to occur. The correct probability for Z, considering that they are not independent, is as follows:

$$P(Z) = p(Z)\{1 - (1 - p(e_2))(1 - p(e_3))\}P(Y) \simeq 0.0469.$$

As can be seen from this example, the event probabilities will be different depending on whether we assume each event's independence. The assumption of event independence implies that an attacker will only try one attack path at a time. In practice, the attacker will try numerous attack paths. We would say that the assumption of independence is an oversimplification.

Ou and Singhal (2012), Homer et al. (2013) propose an algorithm that correctly computes the probabilities of the events, assuming that they are not independent. This algorithm is very similar to that of Bayesian networks. We do not get into the details of their approach. Instead, we will discuss approaches based on Bayesian networks in Section 6.

4.5. Discussions

The following subsections discuss several topics important in practice and further applications not mentioned above.

4.5.1. Computational complexity

For the non-probabilistic metrics, the metric by Idika and Bhargava (2012) is intuitive but potentially very costly. This is because their metrics enumerate attack paths implicitly. The enumeration of attack paths can cause a computational explosion. In comparison, Noel and Jajodia (2014) design the metrics carefully to keep

the computational complexity within a minor range. However, it is not a simple task to use the metrics to understand the true nature of risks.

At first glance, the computational complexity of probabilistic metrics in this article seems relatively tractable; the computational complexity of StateRank and AssetRank is dominated by the sum of products computation performed for matrices, which is $O(N^3)$ for the size $N \times N$ of the matrices. Note that the size N is equal to the number of nodes in the attack graph. The metric proposed by Wang et al. (2008) can be calculated even more efficiently by dynamic programming. The required computation scales linearly. Though, this metric assumes independence between events, which is an oversimplification of reality. At this point, the metric by Homer et al. (2008b, 2013) helps. Unfortunately, it requires an exponential amount of computation in the worst case. After all, calculating accurate probabilistic metrics requires considerable computational effort.

4.5.2. Cycle handling

Wang et al. (2008) apply to acyclic attack graphs. Although we did omit the detail, their method includes an algorithm to handle cycles in an attack graph. Similarly, several other studies, such as Noel et al. (2010) and Homer et al. (2008b, 2013), have proposed approaches that deal with cycles.

Noel et al. (2010) propose to "resolve any attack graph cycles through their distance from the initial conditions". More precisely, according to Noel et al. (2003), "preference is given to dependency edges that are closer to the initial conditions." The implicit assumption behind this approach is that an attacker will only use the shortest attack paths. This assumption seems unrealistic. At the very least, reducing the number of attack paths affects metrics such as Idika and Bhargava (2012).

Wang et al. (2008) propose an algorithm to eliminate cycles. When calculating the probability of reaching an exploit node e, consider an attack graph that removes the implicit edges going out from the exploit node e. We denote this subgraph as $A(G, e)$ where G is the whole attack graph before the edge removal. The subgraph $A(G, e)$ contains no cycles going through the exploit node e. We apply a probability calculation algorithm to this graph. The problem with this method is that $A(G, e)$ may still contain cycles that do not pass through the exploit node e. Therefore, we need to apply the cycle removal algorithm recursively. Consequently, depending on the order of these eliminations, the probabilities change because the obtained cycle-free graphs can vary. The algorithm by Wang et al. (2008) uses breadth-first order, but its justification is not mentioned.

In the method proposed by Homer et al. (2008b, 2013), roughly speaking, when a cycle is encountered during the probability calculation, the algorithm enumerates partial attack paths overlapping with the cycle. The individual paths enumerated are cycle-free, so probability calculations apply. The problem is that the number of extractable attack paths can explode.

Although the handling of cycles is not often discussed these days, it is one of the essential issues in attack graph analysis. In particular, the probabilistic interpretation of an attack graph requires decomposing cycles into attack paths up to some degree, which leads to a computational explosion.

4.5.3. Reliability of metrics

Non-probabilistic metrics are primarily qualitative; we can compare two metrics and argue which one is better, but we cannot say it is twice as safe just because it has twice the value. This means that we can choose the best of several options while it is impossible to calculate their cost-benefits. If the metrics are defined on a partially-ordered set, we cannot even choose the best option.

The probability value is a quantitative measure, thus, in theory, a preferrable metric. The problem with using probabilistic metrics is that statistical data for input is generally unavailable. A probability derived from unreliable inputs cannot be reliable. However, it is possible to understand the risk trend even though the exact probability is not known. For example, Noel et al. (2010) propose a risk assessment method using Monte-Carlo simulation with varying parameters.

Problems with the reliability of risk assessments are not limited to attack graphs and derived metrics; Verendel (2009) has conducted a critical survey of security risk assessment studies that advocate quantitative assessments. The survey points out that the quantitative assessment of security risk is immature. While research efforts to overcome this challenge are indispensable, it is first and foremost important to understand the limitations of theories when we use them.

4.5.4. Further readings

There are many noteworthy security metrics based on attack graph analysis that are not covered in detail in this article. Let us refer to some of them here.

A common weakness of attack graph analysis is that it requires every vulnerability to be known. The k-zero day safety proposed by Wang et al. (2010) compensates for this weakness. In this approach, we calculate how many unknown vulnerabilities are required to make the attack successful if it fails with known vulnerabilities only. When the minimum number of vulnerabilities that need to be added is k, we say that the network is k-zero day safe. The required computation for k will be NP-hard in the worst case. However, it is possible to determine in polynomial time whether k is greater than an arbitrarily selected k'. Also, Wang et al. (2014a) propose an improved algorithm with significantly reduced computational complexity under some assumptions, such as that all the assumed unknown vulnerabilities are unique.

Wang et al. (2014b) focus on the network diversity metrics indicating the breadth of varieties of resources in a network. The idea behind this is that the more divergent network is more difficult to compromise. Later, Zhang et al. (2016) have refined that idea to incorporate the dependency among exploits to handle the case where one successful exploit makes other exploits easier.

Alert correlation is a method to identify the occurrence of anomalous events by analyzing various network logs. We can regard it as a kind of dynamic security metric. Noel et al. (2004) propose a method to infer ongoing attacks as a chain of exploits from the logs by applying the information contained in the attack graph. This method is further improved by Wang et al. (2006a) to overcome several weaknesses, such as the intolerance of slow attacks. Roschke et al. (2011) propose applying attack-graph-based alert correlation to network security forensics. Husák et al. (2019) give a survey about alert correlation, including attack graph applied methods.

Cao et al. (2018) propose to expand the scope of security measurement to cover the business impact analysis. Their study leverages the expressiveness of MulVAL/Datalog to describe the dependency between services and its network infrastructure, i.e., the goal conditions represent the business impacts. They discuss pruning the generated attack graphs to simplify the relevant risk metrics calculation.

Most of the metrics referenced in this section are described in detail in the book by Wang et al. (2017). While not necessarily limited to attack graphs, see Kordy et al. (2014), Ramos et al. (2017), and Pendleton et al. (2017) for a broader survey of security metrics, including recent research developments.

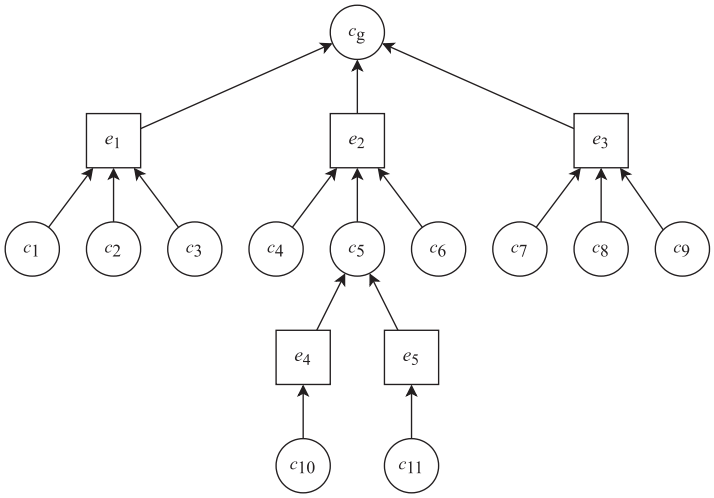


Fig. 5.1. Another example of an attack graph.

5. Network hardening

Network hardening is the problem of searching for the optimal set of security controls. Attack-graph-based network hardening primarily searches for a subset of the initial conditions that correspond to a minimal set of prerequisites for the chosen goal conditions. In an attack graph, the initial conditions represent vulnerabilities and configuration defects. The security controls correspond to the removal of initial conditions. In turn, the minimal set of initial conditions required for the goals implies the minimal set of sufficient security controls. In this section, we will see various approaches for network hardening with attack graphs.

5.1. Minimal-set search

We are interested in a subset of the initial conditions composed of only those essential conditions to the attack concerned. While, ideally, the minimum set should be present, it is not always met. Suppose we have an attack graph depicted in Fig. 5.1.

The logical proposition corresponding to this graph is as follows:

$$c_g = (c_1 \wedge c_2 \wedge c_3) \vee (c_4 \wedge (c_{10} \vee c_{11}) \wedge c_6) \vee (c_7 \wedge c_8 \wedge c_9).$$

To make the goal condition c_g false, we need to all the exploits e_1, e_2, and e_3 to be false. Therefore, the number of possible combinations of initial conditions which are set to be false to make c_g false is $\left(2^3-1\right)^3 = 343$. For example, any of $\{c_1, c_{10}, c_{11}, c_8\}$, $\{c_1, c_2, c_6, c_8\}$, and $\{c_1, c_6, c_8\}$ can make c_g false. As this example shows, one combination can be a subset of another combination. In general, the solution set of a network hardening problem is a partially ordered set of sets. Thus, there is not necessarily a minimum element. Let us call these legitimate combinations of initial conditions *solutions* for the network hardening problem. A solution is said to be *minimal* when no solution is a subset of that solution. The minimum solution is always minimal, but the reverse is not true, e.g., $\{c_1, c_{10}, c_{11}, c_8\}$ is minimal but not minimum. It is the set of minimal solutions that we focus on as the solution set for network hardening problems.

Given an attack graph and a goal condition, Ammann et al. (2002) propose an algorithm to find a minimal solution that prevents the goal condition from being fulfilled. Algorithm 3 shows the detail.

Note that this algorithm is only applicable for acyclic attack graphs. Besides, the algorithm's output is not uniquely determined since it chooses any one of the condition nodes in line 5. There are usually multiple minimal solutions, and this algorithm chooses one of them. An algorithm enumerating all the minimal solutions does not work. It can easily lead to a computational explosion.

Algorithm 3 An algorithm to find a minimal solution (adapted from Ammann et al. (2002))

Require:
C_g: the non-empty set of goal conditions.
R_i: the set of imply relations.
R_r: the set of require relations.
Ensure:
the set of initial conditions required for C_g.

```
1:  function FIND-MINIMAL-SOLUTION(C_g, R_i, R_r)
2:      C'_g ← ∅
3:      for e ∈ {e | (e, c) ∈ R_i, c ∈ C_g} do
4:          C ← {c | (c, e') ∈ R_r, e' = e}
5:          C'_g ← C'_g ∪ {Take an arbitrary element from C}
6:      end for
7:      if C'_g = ∅ then
8:          return C_g
9:      else
10:         C_i ← {c | c ∈ C_g, (e, c) ∉ R_i}
11:         return C_i ∪ FIND-MINIMAL-SOLUTION(C'_g, R_i, R_r)
12:     end if
13: end function
```

5.2. Least-cost-set search

While every minimal solution can prevent the goal condition from being achieved, their desirabilities are different since the actual cost assigned to each security control is different. It would be more beneficial to find the cost-minimum minimal solution. We call the minimal solution with the lowest implementation cost as the *least-cost solution*.

Let us clearly define the problem to find the least-cost solutions. Suppose we have an attack graph $G = (E \cup C, R_r \cup R_i)$ and its initial conditions $C_i \subseteq C$. For this attack graph, let w be a function $C_i \to \mathbb{R}_+$, which maps an initial condition to its invalidation cost. For any set of initial conditions S, let us define the total cost of S as follows:

$$\text{cost}(S) = \sum_{c \in S} w(c).$$

Besides, let $c_g \in C$ be the goal condition, and $\mathcal{S}(c_g)$ be the set of all solutions, each in which invalidates c_g. With these terms, we can define the least-cost solution L as follows:

$$L = \underset{S \in \mathcal{S}(c_g)}{\arg\min} \operatorname{cost}(S).$$

Note that L cannot necessarily be unique under this definition.

Noel et al. (2003) propose an algorithm for finding a least-cost solution. The algorithm transforms an acyclic attack graph into a corresponding logical proposition and converts the expression into Conjunctive Normal Form (CNF). Each maxterm contained in it is the solution to the network hardening problem. A maxterm is a disjunction of initial conditions. After calculating the cost of each maxterm, we can choose the maxterm with the least cost. The problem is that the process of converting an attack graph to a CNF requires exponential computational complexity in the worst case since the number of maxterms can grow exponentially.

Wang et al. (2006b) propose an improved version of the above approach. This improved approach can handle multiple goal conditions and can also apply to cyclic attack graphs. Algorithm 4 shows the detail of this approach.

The points of this algorithm are as follows. First, the derived logical proposition L is expressed in DNF rather than CNF. This difference comes from the conjunction of the negation of goal conditions used as the starting point. The initial conditions contained in each conjunction in the obtained DNF are the solution to the network hardening problem. We can choose the least-cost solution by calculating and comparing each cost. Secondly, the "procedure avoids running into cycles by only expanding the search towards those vertices not reachable from the current vertex (line 13), and it also avoids introducing unsatisfiable logic loops into the final result (line 10). The procedure handles exploits in a similar way (line 18 through line 29)." (Wang et al. (2006b), the line numbers are recapped). This treatment makes this algorithm applicable to cyclic attack graphs. Still, the algorithm also has the potential computational explosion; the number of conjunctions in the DNF can increase exponentially for the number of initial conditions.

Algorithm 4 A procedure for deriving solutions to the network hardening problem. (adapted from Wang et al. (2006b))

Require:
$(E \cup C, R_r \cup R_i)$: An attack graph.
$C_g \subseteq C$: The goal conditions.
Ensure:
A solution L to the goal $\bigwedge_{c \in C_g} \neg c$.

1: $L \leftarrow \bigwedge_{c \in C_g} \neg c$ ▷ The initial goal
2: **let** Q be a queue initialized with C_g ▷ A queue used for searching in the graph
3: **for** $e \in E$ **do** $Pre(e) \leftarrow \{e\}$ ▷ The predecessor list
4: **for** $c \in C$ **do** $Pre(c) \leftarrow \{c\}$ ▷ The predecessor list

5: **while** $Q \neq \emptyset$ **do**

6: **for** each condition c dequeued from Q **do**
7: $S_e \leftarrow \{e_k \mid (e_k, c) \in R_i\}$ ▷ $|S_e| = n$
8: $T \leftarrow (e_1 \vee e_2 \vee \cdots \vee e_n)$ ▷ Temporary variable
9: **for** $e_i \in S_e \cap Pre(c)$ **do**
10: **replace** e_i with *FALSE* in T
11: **end for**
12: **replace** c with T in L
13: **for** $e_i \in S_e \setminus Pre(c)$ **do**
14: **enqueue** e_i in Q
15: $Pre(e_i) \leftarrow Pre(e_i) \cup Pre(c)$
16: **end for**
17: **end for**

18: **for** each exploit e dequeued from Q **do**
19: $S_c \leftarrow \{c_k \mid (c_k, e) \in R_r\}$ ▷ $|S_c| = m$
20: $T \leftarrow (c_1 \wedge c_2 \wedge \cdots \wedge c_m)$ ▷ Temporary variable
21: **for** $c_i \in S_c \cap Pre(e)$ **do**
22: **replace** c_i with *FALSE* in T
23: **end for**
24: **replace** e with T in L
25: **for** $c_i \in S_c \setminus Pre(e)$ **do**
26: **enqueue** c_i in Q
27: $Pre(c_i) \leftarrow Pre(c_i) \cup Pre(e)$
28: **end for**
29: **end for**

30: **end while**

31: **transform** L into DNF

5.3. Allowable-set search

Suppose we have two initial conditions denoting (1) a web server has a vulnerability available to the attacker, and (2) the web server is running. We cannot invalidate the latter condition while keeping the first condition. That is, some initial conditions cannot be invalidated independently and can only be invalidated in combination with some other initial conditions. A generalization of this idea is the allowable set described below.

An *allowable action* is any subset of initial conditions of an attack graph such that all the conditions in it can only be jointly disabled. A *strategy* is a set of allowable actions such that the attacker cannot achieve the goal condition when every condition in those actions is disabled. In a more realistic network hardening, our interest is in finding the best strategy.

Albanese et al. (2012) propose an algorithm to find an least-cost strategy. However, the algorithm's worst-case computational complexity is enormous. They estimate it to be $O\left(n^{n^d}\right)$, where d is the maximum distance (number of edges) between initial and goal conditions, and n is the maximum in-degree of nodes in the attack graph. While the concept of allowable set is useful, this algorithm is impractical because of this computational burden.

5.4. Approximated-set search

As the discussions above clarify, reducing computational complexity is a critical issue in network hardening. The root cause of the computational explosion is finding the minimum solution or the least-cost solution. In principle, the number of solutions to the network hardening problem is exponential in the size of the network. Therefore, to achieve a drastic reduction in computational complexity, we must first give up on obtaining the best solution. Albanese et al. (2012) propose the following Algorithm 5 for the approximate minimum solution, which is exactly in line with this observation. Note that, in this algorithm, $\mathcal{S}$ refers to the set of all strategies, C_i denotes the initial conditions, and $\mathcal{A}$ is the set of all allowable actions.

The algorithm is based on a forward search and identifies strategies to block the attacks that depart from the initial conditions and reach a single goal condition. When multiple-goal conditions are needed, just add a dummy goal condition and a dummy exploit node that aggregate the goal conditions.

The major difference of this algorithm from the prior ones is the reduction of the search scope. The $\sigma(q)$ represents the set of strategies blocking all the paths to node q. The function "TopK" in line 15 limits the size of this set to k, in order of least cost. "TopK"

Algorithm 5 A forward-search algorithm for the least-cost strategy (adapted from Albanese et al. (2012))

Require:
$(E \cup C, R_r \cup R_i)$: An attack graph.
k: The optimization parameter.
Ensure:
$\sigma : C \cup E \to 2^{\mathcal{S}}$: a mapping.
$minCost : C \cup E \to \mathbb{R}^+$: a mapping.

1: $Q \leftarrow \text{TopologicalSort}\,(C \cup E)$
2: **while** $Q \neq \emptyset$ **do**
3: $q \leftarrow$ a node dequeued from Q
4: **if** $q \in C_i$ **then**
5: $\sigma(q) \leftarrow \{\{A\} \mid A \in \mathcal{A}, q \in A\}$
6: **else if** $q \in E$ **then**
7: $\sigma(q) \leftarrow \bigcup_{(c,q)\in R_r} \sigma(c)$
8: **else if** $q \in C \setminus C_i$ **then**
9: $\sigma' \leftarrow \{\emptyset\}$
10: **for** $(e, q) \in R_i$ **do**
11: $\sigma' \leftarrow \left\{S \cup S_e \mid S \in \sigma', S_e \in \sigma(e)\right\}$
12: **end for**
13: $\sigma(q) \leftarrow \sigma'$
14: **end if**
15: $\sigma(q) \leftarrow \text{TopK}\,(\sigma(q), k)$
16: $minCost(q) \leftarrow \min_{S\in\sigma(q)} \text{cost}\,(S)$
17: **end while**

implicitly refers to the minimum costs of the strategies calculated in line 16. The topological sort ensures the readiness of those minimum costs before the first application of "TopK". The scope trimming by "TopK" suppresses the computational complexity explosion effectively.

The least cost strategy obtained by this algorithm is only an approximation. However, Albanese et al. (2012) report no significant deviation from the exact optimal strategy in numerical experiments. This algorithm shows excellent scalability without causing a computational explosion.

5.5. Discussions

Network hardening is computationally very expensive. It is still a tough but unignorable problem that we should solve. The following sees the breadth of issues and other attempts.

5.5.1. Computational complexity

Ammann et al. (2002)'s algorithm for finding a single minimal solution has polynomial complexity. The algorithm by Wang et al. (2006b) to find the least-cost solution and the backward-search algorithm by Albanese et al. (2012) to consider the allowable set have exponential complexity in the worst case. No estimates are given for the algorithm by Albanese et al. (2012) to find an approximate solution, but the numerical experiments' results show good scalability.

Is there room for significant improvement in the algorithm for exact solutions? At least, it is a tough problem. Remind that Wang et al.'s algorithm transforms the attack graph into a logical proposition in DNF and searches the least-cost disjunction. Li (2004) coined such a problem as the MinCostSAT problem. It is a search problem for the true/false assignment to propositional variables that makes the concerned proposition true while considering the assignment's cost, given the individual costs of variables. This problem is known to be NP-hard. Although the worst-case computational explosion is inevitable, we can leverage the general-purpose MinCostSAT solver. Homer et al. (2008a) approach this problem along with this observation.

5.5.2. Cycle handling

In the algorithms presented in this section, only Wang et al. (2006b) explicitly incorporates the considerations for cyclic attack graphs. The other algorithms apply to acyclic attack graphs. Albanese et al. (2012) state that the method by Wang et al. (2006b) can be leveraged to handle cycles, though the detail of such an integration is not given. As discussed in Section 4.5.2, the conversion to acyclic graphs makes the resultant graph depend on the order of cycle elimination. The use of topological sort in the approximation algorithm by Albanese et al. (2012) poses a similar problem. The result of topological sort is not necessarily unique. The reasonable and stable approach for cycle handling is still an important research issue.

5.5.3. Multi-objective network hardening

The algorithms discussed above are for qualitative, single-objective optimization. To date, there are few studies about multi-objective optimization algorithms for quantitative security metrics over unrestricted exploit-dependency graphs. The closest is the work by Stan et al. (2021) using A* solver to approximate the Pareto front of Cost/Benefit optimization, but for acyclic graphs. Poolsappasit et al. (2012) use a genetic algorithm (NSGA-II) for acyclic graphs. Dewri et al. (2007) and Dewri et al. (2012) use NSGA-II for attack trees rather than attack graphs. Ali and Khan (2013) apply the particle swarm optimization technique to attack trees. Almohri et al. (2016) apply Linear Programming techniques to acyclic graphs. Any of these algorithms is an approximation algorithm, and there is no guarantee for its optimality. Computational complexity is often problematic, too. The research in this field seems still immature.

5.5.4. Game theoretic approach

The game-theoretic approach is useful to discuss network hardening problems, too. In the approach, we regard network hardening as a game between an attacker and a defender. The attacker tries to compromise the network while the defender blocks it. An attack graph provides constraints that determine the actions available to the attacker and the defender. We can formulate this situation as a game by supplying utility and cost functions for the two.

Letchford and Vorobeychik (2013) examine this approach and enumerate the important factors such as uncertainty assumption and computational complexity. Durkota et al. (2019) discuss the game alike where the defender's action is the placement of honeypots to deceive the attacker. Their study also elaborates on the detail of this approach. Nguyen et al. (2018) propose a sampling-based heuristic algorithm to solve another attack-graph game. Miehling et al. (2018) elaborate on a very similar but different approach based on POMDP. They use Monte-Carlo sampling and the dynamical data from IDS to estimate the actual state of the network while searching for the optimal defense action online. Hu et al. (2017) apply a Q-learning approach to a similar problem based on Bayesian attack graphs with an unknown utility function.

The problem of the game-theoretic approach is computational complexity. In general, the game's tree is too huge to search for the globally optimal solution. Therefore, it is almost inevitable to use some heuristics. The studies referred to above propose such heuristics and evaluate their performance.

There is another study worth noting here. The experiment by Durkota et al. (2016) compares the optimal defense strategy induced from attack graphs by their algorithm and the strategies

worked out by human respondents. The result shows that the optimal strategy is robust against the worst-case attacks, while it is inferior to the man-made strategies when applied to man-made attack strategies. This result implies an unexplored field.

6. Bayesian attack graph

A Bayesian attack graph is an attack graph defined as a kind of Bayesian network. It is a generalized form of probabilistic attack graphs, which treats the exploit-dependency graph as a particular case. Moreover, it considers the dependencies among events, discussed in Section 4.4, during the attack probability calculation. It is also capable of updating the probability under some observed attack information. We can leverage the rich results in Bayesian network research to enhance attack graph analysis. Although it is not possible to handle cyclic attack graphs due to their acyclic definition, it is the most comprehensive framework to date for discussing attack graphs. This section elaborates on its characteristics.

6.1. Brief introduction to Bayesian network

Firstly, we define several terms relevant to Bayesian network. In this section, the capital variables such as $X_1, \ldots, X_n$ denote Bernoulli variables. Then, a *Bayesian network* is a joint distribution of $X_1, \ldots, X_n$ defined as

$$P(X_1, \ldots, X_n) = \prod_k P(X_k \mid \text{parents}\,(X_k))$$

where (1) "parents (X_k)" denotes the subset of $\{X_1, \ldots, X_n\}$ satisfying

$$P(X_k \mid X_1, \ldots, X_{k-1}, X_{k+1}, \ldots, X_n) = P(X_k \mid \text{parents}\,(X_k)),$$

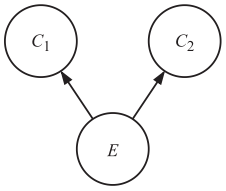
and (2) the dircted graph composed of $X_1, \ldots, X_n$ and their dependencies is a DAG, i.e., the set of all ascendants of any X_k does not contain X_k. A leaf X_k without parents is supposed to have a *prior distribution* $P(X_k)$. On the other hand, every node X_k with parents has a *conditional probability distribution* $P(X_k \mid \text{parents}\,(X_k))$. The conditional probability distribution is often expressed in a *conditional probability table (CPT)*.

The accurate computation of various probabilities for a Bayesian network, along with the definition, is called *exact inference*. The computation at the expense of accuracy while suppressing the computational complexity is called *approximate inference*. The exact inference is known to be an NP-hard problem (Dagum and Luby, 1993). Therefore, it is inevitable to rely on approximate inferences to process large-scale networks.

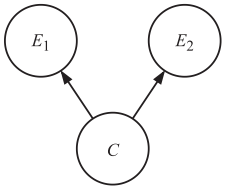
Belief propagation is a well-known inference algorithm proposed by Pearl (1988). It is an exact inference algorithm applicable to Bayesian networks with polytree (also called singly-connected) structure. At the same time, it is also empirically known as an approximate inference algorithm applicable to arbitrary Bayesian networks (Murphy et al., 1999). The computational complexity of belief propagation is almost linear, although belief propagation over non-polytree networks produces approximated inference results. The textbook by Darwiche (2009) provides a concise but detailed explanation of various methods.

6.2. Bayesian attack graph

A node other than initial conditions appearing in an attack graph represents either AND or OR combinations of other nodes connecting to itself. In a Bayesian network, a conditional probability represents the combination among nodes. If we can use CPT to represent the AND and OR combination, an attack graph can be interpreted as a Bayesian network. Frigault and Wang (2008) pointed out that it is indeed possible. Figure 6.1 shows the CPT representations of those bindings. Note that the edge direction is different from that of an attack graph.



C_1	C_2	P(E = T | C_1, C_2)	P(E = F | C_1, C_2)
T	T	1	0
T	F	0	1
F	T	0	1
F	F	0	1



E_1	E_2	P(C = T | E_1, E_2)	P(C = F | E_1, E_2)
T	T	1	0
T	F	1	0
F	T	1	0
F	F	0	1

Fig. 6.1. AND and OR combinations by CPTs.

In the case for E, an exploit node, the CPT on the right says that the variable E becomes true if and only if its two parents C_1 and C_2 are true. Similarly, In the case for C, a condition node, the CPT says that the variable C becomes false if and only if its two parents E_1 and E_2 are false. This behavior is what we exactly need to express an attack graph as a Bayesian network. In this way, we can handle an attack graph as a specific case of Bayesian networks. Furthermore, we can use various probabilities such as 0.3 or 0.7 in the exploit node's CPT. Such a probability assignment generalizes the concept of attack graphs. Based on this observation, from now on, we will call a Bayesian network interpretable as a generalized acyclic attack graph a *Bayesian attack graph*.

Bayesian attack graphs have a variety of benefits. Firstly, the inference considers the dependence among attack paths. Secondly, CPT's flexibility allows us to represent complicated exploits that are difficult to represent in a qualitative attack graph. We will elaborate on the use of this flexibility in the next subsection. The third, but not the last, benefit is that we can estimate the relevant event's probability based on the observation of other variables' values. For example, when one host is found compromised, we can estimate the probability of its neighbors' compromise. A method for estimating the current state of a network based on observed evidence is called *dynamic analysis*. Peng Xie et al. (2010) propose a modeling method for Bayesian attack graphs to estimate the attack probability more accurately based on the information detected by IDS.

6.3. Advanced exploit-dependency modeling

Suppose we have a host with three vulnerabilities named V_1, V_2, and V_3, respectively. If the attacker exploits V_1 or V_2, he can log in to the host. The login state is required to exploit V_3. Figure 6.2 is a Bayesian attack graph that illustrates this situation.

In this Bayesian attack graph, to achieve the goal condition C_g, the attacker must achieve at least two exploits, E_1 and E_3, or E_2 and E_3. Here, let us assume that it is easier to achieve E_3 after E_2 than E_3 after E_1. In that case, how can we express the difference in difficulty? Attack graphs cannot handle cases like this. In contrast, Bayesian attack graphs can naturally represent it using the CPT, shown in Table 2.

The above discussion's essence is that a successful exploit has side effects that affect the success probability of other exploits. Wang et al. (2008) pointed out that Bayesian attack graphs could handle such dependence considerations between exploits.

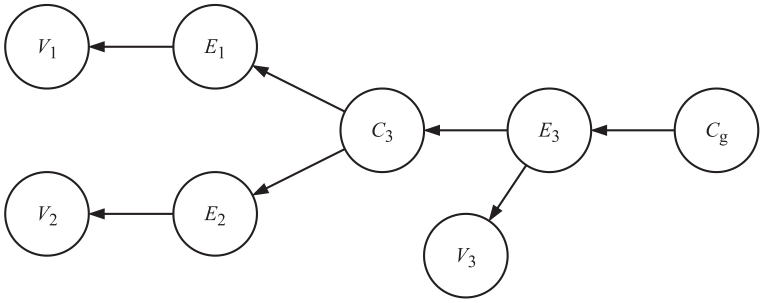


Fig. 6.2. An example of dependent exploits.

Table 2
An example CPT with varying exploitability.

V_3	C_3	E_2	$P(E_3 = T \mid V_3, C_3, E_2)$
T	T	T	0.61
T	T	F	0.55
T	F	T	0.12
T	F	F	0
F	T	T	0
F	T	F	0
F	F	T	0
F	F	F	0

Cheng et al. (2012) propose an algorithm to automatically determine the dependencies between exploits by focusing on individual base metrics extracted from CVSS scores. Cheng et al. (2017) propose the refined algorithm that enables this probability adjustment process within the framework of Bayesian attack graphs by adding some nodes and edges to the original Bayesian attack graph.

The powerful expressiveness of the Bayesian attack graph is also helpful in dealing with network security's complex nature. Although not covered in this article, there exist other thought-provoking studies. Frigault et al. (2008) propose a security assessment method combining Bayesian attack graphs and CVSS temporal metrics. Sun et al. (2016) propose a method for detecting zero-day attacks by constructing a kind of Bayesian attack graph from system logs.

6.4. Loopy belief propagation and noisy OR/AND

Belief propagation applied to general Bayesian networks as an approximate inference algorithm is called *loopy belief propagation*. While the loopy belief propagation works efficiently, the computational load relevant to huge CPTs inhibits efficiency. Since CPTs allow us to set probabilities for each possible combination of the parent variables' values, we must articulate the exponential number of cases. It implies the combinatorial explosion and intractable memory consumption. In contrast, using the conditional probability determined by a specific rule makes it possible to simplify the calculation. A typical example is the use of Noisy OR/AND, which is shown below.

Noisy OR is a model that determines the conditional probability with N parent variables by N configuration parameters $c_1, \ldots, c_N$. Noisy AND is another model similar to Noisy OR with $2N$ configuration parameters $c_1, \ldots, c_N$ and $s_1, \ldots, s_N$ for N parent variables. Each of them is defined as follows:

$$P_{\mathrm{OR}}(X = \mathrm{T} \mid \mathbf{u}) = 1 - \prod_{k \in I_{\mathrm{T}}(\mathbf{u})} (1 - c_k),$$

$$P_{\mathrm{AND}}(X = \mathrm{T} \mid \mathbf{u}) = \prod_{k \in I_{\mathrm{T}}(\mathbf{u})} c_k \prod_{k \in I_{\mathrm{F}}(\mathbf{u})} s_k,$$

where $\mathbf{u}$ denotes the set of observed values, such as $(U_1 = \mathrm{T}, U_2 = \mathrm{T}, U_3 = \mathrm{F})$, for the parents variables of X, and c_k and s_k are the configuration parameters. The term $I_{\mathrm{T}}(\mathbf{u})$ and $I_{\mathrm{F}}(\mathbf{u})$ are the set of indices corresponding to the observed values in the parents. For example, if $\mathbf{u} = (U_1 = \mathrm{T}, U_2 = \mathrm{T}, U_3 = \mathrm{F})$, then $I_{\mathrm{T}}(\mathbf{u}) = \{1, 2\}$ and $I_{\mathrm{F}}(\mathbf{u}) = \{3\}$. Roughly speaking, the configuration parameters c_k in Noisy OR express the contribution of the parent facts to X. If one c_k has a value which equals to one, then the observation of $U_k = \mathrm{T}$ assures that $X = \mathrm{T}$. If every c_k is less than one, X will not be guaranteed to be T even when every $U_k = \mathrm{T}$. The alike applies to Noisy AND.

Pearl (1988) discusses the use of Noisy OR/AND in the belief propagation. It simplifies the equations for the message passing. The computation is quite efficient since the transformed expression contains only $O(N)$ operations, where N is the number of parent variables.

Muñoz-González et al. (2019) report the performance of exact inference using Noisy OR/AND. Furthermore, Muñoz-González et al. (2017) investigate the performance with the combination of loopy belief propagation and (Leaky) Noisy OR/AND. Compared to the brief introduction in this article, their report examines the detail about the treatment of Bayesian networks. There is already a considerable amount of studies for Bayesian networks. The two studies are good starting points to understand those details from the viewpoint of attack graph analysis.

6.5. Discussions

Unfortunately, Bayesian-attack-graph-based practical methods seem immature due to the excessive computational burden. The following is a list of topics informative for further research.

6.5.1. CPT and canonical models

The Noisy OR/AND introduced in Section 6.4 is a kind of *canonical models*. Although we introduced Noisy OR/AND to reduce the computational complexity, the canonical model is usually preferred for its ease of modeling. Suppose that a variable in a Bayesian network has N parent variables. In this case, the CPT for this variable has 2^N rows, and the number of rows increases exponentially as N increases. It is not easy to determine the appropriate probabilities for each row. The canonical model provides a solution to this challenge since it determines the CPT with a limited number of parameters, such as N. Diez and Druzdzel (2006) provide a concise description of canonical models.

6.5.2. Reliability of inference

Attack graph analysis is the automation of logical reasoning based on input data, and the reliability of the analysis depends on the reliability of the input data. The situation is no different in Bayesian attack graphs. Rather, it is more challenging to ensure the reliability of the input data. Qualitative attack graph analysis only requires clarifying the pre/postconditions of the exploit. For

Bayesian attack graphs, we need to add probabilistic evaluations of exploits.

In the literature, the CVSS score is widely used for probability assignment. The caveat is that it is essentially qualitative and does not solve the issue. For example, suppose we have two probability values A and B, based on the CVSS score, and they satisfy that $A = 2B$. If A and B are reliable as probabilities, we can estimate the events corresponding to A as occurring twice as often as those of B. However, it is groundless.

Generally speaking, we cannot take the various probabilities obtained from a Bayesian attack graph at face value. In practice, we need additional careful treatment for the results.

6.5.3. Network hardening

Most of the methods in Section 5 do not apply to Bayesian attack graphs. While these methods assume that the effect of security controls reflects in binary, boolean values, Bayesian attack graphs require them to be probabilities.

One way to solve the network hardening problem for Bayesian attack graphs is to apply a generic solver. For example, Poolsappasit et al. (2012) propose applying the well-known genetic algorithm NSGA-II to Bayesian attack graphs. There are general-purpose algorithms for multi-objective optimization, such as genetic algorithms and swarm optimization. However, all of these algorithms are iterative. In the iterations, the tentative solution must be evaluated repeatedly. This means that we have to infer the probabilities on the Bayesian attack graph many times, and the whole computational complexity is enormous. Furthermore, general-purpose algorithms tend to be slow to converge, which makes this approach infeasible. This field of research needs further development.

6.5.4. Uncertain graph

Finally, we would like to refer to the studies by Nguyen et al. (2017), Nguyen and Nicol (2020). Their studies are based on uncertain graphs; an uncertain graph is a probabilistic distribution of deterministic graphs. This framework defines a cyclic attack graph as a probabilistic distribution of acyclic attack graphs. When we want to calculate the reachability to a node, in theory, we extract every possible acyclic graph and check the path's existence on each graph. Then, determine the reachability as the weighted sum according to each acyclic graph's existence probability. The two studies elaborate on the Monte-Carlo sampling strategy for uncertain graphs. Though it seems not scalable so far, it is a very different, challenging approach. The uncertain graph provides a formalism that can generalize the exploit dependency graphs with probability in a way other than Bayesian networks.

7. Conclusion

In this article, we summarized the major contributions in the field of attack graph analysis. An attack graph is a kind of cyclic AND/OR graphs. The semi-naive evaluation of a Datalog program generates an attack graph efficiently. While an attack graph comprises thousands of nodes, intelligible visualization techniques such as hierarchical representation are available. Decision-makers can also use various network security metrics based on attack graphs. These metrics clarify the probabilistic or qualitative evaluation of the risk posture of the concerned network. Furthermore, several network hardening algorithms can search for optimal security controls to prevent undesirable events. From the theoretical point of view, the Bayesian attack graph seems promising since it can generalize the concept of attack graphs with probabilistic information. The remaining key issues in this field include handling cycles, AND combinations of conditions, and the intractable computational complexity. Further research is required to make this approach more practical.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data Availability

No data was used for the research described in the article.

Acknowledgments

The author wishes to acknowledge Dr. Keisuke Tanaka, Professor in the Department of Mathematical and Computing Science, School of Computing, Tokyo Institute of Technology, for reviewing and providing constructive advice on the drafts of this article.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.cose.2022.103081.

References

Aksu, M.U., Bicakci, K., Dilek, M.H., Ozbayoglu, A.M., ıslam Tatli, E., 2018. Automated generation of attack graphs using NVD. In: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy. ACM, New York, NY, USA, pp. 135–142. doi:10.1145/3176258.3176339.

Albanese, M., Jajodia, S., Noel, S., 2012. Time-efficient and cost-effective network hardening using attack graphs. In: IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012). IEEE, pp. 1–12. doi:10.1109/DSN.2012.6263942.

Ali, H., Khan, F.A., 2013. Attributed multi-objective comprehensive learning particle swarm optimization for optimal security of networks. Appl. Soft Comput. 13 (9), 3903–3921. doi:10.1016/j.asoc.2013.04.015.

Almohri, H.M., Watson, L.T., Yao, D., Ou, X., 2016. Security optimization of dynamic networks with probabilistic graph modeling and linear programming. IEEE Trans. Dependable Secure Comput. 13 (4), 474–487. doi:10.1109/TDSC.2015.2411264.

Ammann, P., Pamula, J., Street, J., Ritchey, R., 2005. A host-based approach to network attack chaining analysis. In: 21st Annual Computer Security Applications Conference (ACSAC'05). IEEE, pp. 72–84. doi:10.1109/CSAC.2005.6.

Ammann, P., Wijesekera, D., Kaushik, S., 2002. Scalable, graph-based network vulnerability analysis. In: Proceedings of the 9th ACM conference on Computer and communications security - CCS '02. ACM Press, New York, New York, USA, p. 217. doi:10.1145/586110.586140.

Bacic, E., Froh, M., Henderson, G., 2006. MulVAL Extensions For Dynamic Asset Protection. Defence R&D Canada. April

Cao, C., Yuan, L.P., Singhal, A., Liu, P., Sun, X., Zhu, S., 2018. Assessing attack impact on business processes by interconnecting attack graphs and entity dependency graphs. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 10980 LNCS, pp. 330–348. doi:10.1007/978-3-319-95729-6_21.

Ceri, S., Gottlob, G., Tanca, L., 1989. What you always wanted to know about Datalog (and never dared to ask). IEEE Trans. Knowl. Data Eng. 1 (1), 146–166. doi:10.1109/69.43410.

Cheng, P., Wang, L., Jajodia, S., Singhal, A., 2012. Aggregating CVSS base scores for semantics-rich network security metrics. In: 2012 IEEE 31st Symposium on Reliable Distributed Systems. IEEE, pp. 31–40. doi:10.1109/SRDS.2012.4.

Cheng, P., Wang, L., Jajodia, S., Singhal, A., 2017. Refining CVSS-based network security metrics by examining the base scores. In: Network Security Metrics. Springer International Publishing, Cham, pp. 25–52. doi:10.1007/978-3-319-66505-4_2.

Chu, M., Ingols, K., Lippmann, R., Webster, S., Boyer, S., 2010. Visualizing attack graphs, reachability, and trust relationships with NAVIGATOR. In: Proceedings of the Seventh International Symposium on Visualization for Cyber Security - VizSec '10. ACM Press, New York, New York, USA, pp. 22–33. doi:10.1145/1850795.1850798.

Dagum, P., Luby, M., 1993. Approximating probabilistic inference in Bayesian belief networks is NP-hard. Artif. Intell. 60 (1), 141–153. doi:10.1016/0004-3702(93)90036-B.

Darwiche, A., 2009. Modeling and Reasoning with Bayesian Networks. Cambridge University Press.

Dewri, R., Poolsappasit, N., Ray, I., Whitley, D., 2007. Optimal security hardening using multi-objective optimization on attack tree models of networks. In: Proceedings of the 14th ACM Conference on Computer and Communications Security - CCS '07. ACM Press, New York, New York, USA, p. 204. doi:10.1145/1315245.1315272.

Dewri, R., Ray, I., Poolsappasit, N., Whitley, D., 2012. Optimal security hardening on attack tree models of networks: a cost-benefit analysis. Int. J. Inf. Secur. 11 (3), 167–188. doi:10.1007/s10207-012-0160-y.

Diez, F.J., Druzdzel, M.J., 2006. Canonical Probabilistic Models for Knowledge Engineering. Technical Report. UNED, Madrid, Spain. https://www.ia.uned.es/~fjdiez/papers/canonical.pdf

Durkota, K., Lisý, V., Bošanský, B., Kiekintveld, C., Pěchouček, M., 2019. Hardening networks against strategic attackers using attack graph games. Comput. Secur. 87, 101578. doi:10.1016/j.cose.2019.101578.

Durkota, K., Lisy, V., Kiekintveld, C., Bosansky, B., Pechoucek, M., 2016. Case studies of network defense with attack graph games. IEEE Intell. Syst. 31 (5), 24–30. doi:10.1109/MIS.2016.74.

Frigault, M., Wang, L., 2008. Measuring network security using Bayesian network-based attack graphs. In: 2008 32nd Annual IEEE International Computer Software and Applications Conference. IEEE, pp. 698–703. doi:10.1109/COMPSAC.2008.88.

Frigault, M., Wang, L., Singhal, A., Jajodia, S., 2008. Measuring network security using dynamic Bayesian network. In: Proceedings of the 4th ACM Workshop on Quality of Protection - QoP '08. ACM Press, New York, New York, USA, p. 23. doi:10.1145/1456362.1456368.

Froh, M., Henderson, G., 2009. MulVAL Extensions II. Defence R&D Canada-Ottawa. August

Ge, M., Hong, J.B., Guttmann, W., Kim, D.S., 2017. A framework for automating security analysis of the internet of things. J. Netw. Comput. Appl. 83 (April 2016), 12–27. doi:10.1016/j.jnca.2017.01.033.

Gonda, T., Shani, G., Puzis, R., Shapira, B., 2017. Ranking vulnerability fixes using planning graph analysishttps://home/durkota/Downloads/RankingVulnerabilityFixedUsingPlanningGraphAnalysis.pdf.

Homer, J., Ou, X., McQueen, M., 2008. From Attack Graphs to Automated Configuration Management-An Iterative Approach. Technical Report. https://people.cs.ksu.edu/~xou/publications/tr_ou_0108.pdf

Homer, J., Varikuti, A., Ou, X., McQueen, M.A., 2008. Improving attack graph visualization through data reduction and attack grouping. In: Visualization for Computer Security. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 68–79. doi:10.1007/978-3-540-85933-8_7.

Homer, J., Zhang, S., Ou, X., Schmidt, D., Du, Y., Rajagopalan, S.R., Singhal, A., 2013. Aggregating vulnerability metrics in enterprise networks using attack graphs. J. Comput. Secur. 21 (4), 561–597. doi:10.3233/JCS-130475.

Hong, J.B., Kim, D.S., 2016. Towards scalable security analysis using multi-layered security models. J. Netw. Comput. Appl. 75, 156–168. doi:10.1016/j.jnca.2016.08.024.

Hong, J.B., Kim, D.S., Chung, C.-J., Huang, D., 2017. A survey on the usability and practical applications of graphical security models. Comput. Sci. Rev. 26, 1–16. doi:10.1016/j.cosrev.2017.09.001.

Hu, Z., Zhu, M., Liu, P., 2017. Online algorithms for adaptive cyber defense on Bayesian attack graphs. In: MTD 2017 - Proceedings of the 2017 Workshop on Moving Target Defense, co-located with CCS 2017, 2017-January, pp. 99–109. doi:10.1145/3140549.3140556.

Husák, M., Komárková, J., Bou-Harb, E., Čeleda, P., 2019. Survey of attack projection, prediction, and forecasting in cyber security. IEEE Commun. Surv. Tutor. 21 (1), 640–660. doi:10.1109/COMST.2018.2871866.

Idika, N., Bhargava, B., 2012. Extending attack graph-based security metrics and aggregating their application. IEEE Trans. Dependable Secure Comput. 9 (1), 75–85. doi:10.1109/TDSC.2010.61.

Ingols, K., Lippmann, R., Piwowarski, K., 2006. Practical attack graph generation for network defense. In: 2006 22nd Annual Computer Security Applications Conference (ACSAC'06). IEEE, pp. 121–130. doi:10.1109/ACSAC.2006.39.

Inokuchi, M., Ohta, Y., Kinoshita, S., Yagyu, T., Stan, O., Bitton, R., Elovici, Y., Shabtai, A., 2019. Design procedure of knowledge base for practical attack graph generation. In: AsiaCCS 2019 - Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, pp. 594–601. doi:10.1145/3321705.3329853.

Jajodia, S., Noel, S., O'Berry, B., 2005. Topological analysis of network attack vulnerability. In: Managing Cyber Threats. Springer-Verlag, New York, pp. 247–266. doi:10.1007/0-387-24230-9_9.

Jha, S., Sheyner, O., Wing, J., 2002. Two formal analysis of attack graphs. In: Proceedings - Computer Security Foundations Workshop, 15th IEEE, pp. 49–63. doi:10.3724/SP.J.1001.2010.03584.

Johnson, P., Lagerström, R., Ekstedt, M., 2018. A meta language for threat modeling and attack simulations. In: ACM International Conference Proceeding Series doi:10.1145/3230833.3232799.

Kaynar, K., 2016. A taxonomy for attack graph generation and usage in network security. J. Inform. Secur. Appl. 29, 27–56. doi:10.1016/j.jisa.2016.02.001.

Kaynar, K., Sivrikaya, F., 2016. Distributed attack graph generation. IEEE Trans. Dependable Secure Comput. 13 (5), 519–532. doi:10.1109/TDSC.2015.2423682.

Kordy, B., Piètre-Cambacédès, L., Schweitzer, P., 2014. DAG-based attack and defense modeling: Don't miss the forest for the attack trees. Comput. Sci. Rev. 13-14, 1–38. doi:10.1016/j.cosrev.2014.07.001. 1303.7397.

Lallie, H.S., Debattista, K., Bal, J., 2018. An empirical evaluation of the effectiveness of attack graphs and fault trees in cyber-attack perception. IEEE Trans. Inf. Forensics Secur. 13 (5), 1110–1122. doi:10.1109/TIFS.2017.2771238.

Lallie, H.S., Debattista, K., Bal, J., 2020. A review of attack graph and attack tree visual syntax in cyber security. Comput. Sci. Rev. 35, 100219. doi:10.1016/j.cosrev.2019.100219.

Letchford, J., Vorobeychik, Y., 2013. Optimal interdiction of attack plans. In: 12th International Conference on Autonomous Agents and Multiagent Systems 2013, AAMAS 2013, Vol. 1, pp. 199–206.

Li, X.Y., 2004. Optimization algorithms for the minimum-cost satisfiability problem. North Carolina State University, Raleigh, North Carolina Ph.D. thesis. http://www.lib.ncsu.edu/resolver/1840.16/4594

Lippmann, R., Ingols, K., Scott, C., Piwowarski, K., Kratkiewicz, K., Artz, M., Cunningham, R., 2006. Validating and restoring defense in depth using attack graphs. In: MILCOM 2006. IEEE, pp. 1–10. doi:10.1109/MILCOM.2006.302434.

Lippmann, R.P., Ingols, K.W., 2005. An Annotated Review of Past Papers on Attack Graphs. Technical Report. https://pdfs.semanticscholar.org/a186/1778dc31079c80ca5b5d771f124f759ceaa3.pdf

Mehta, V., Bartzis, C., Zhu, H., Clarke, E., Wing, J., 2006. Ranking attack graphs. In: International Workshop on Recent Advances in Intrusion Detection, pp. 127–144. doi:10.1007/11856214_7.

Miehling, E., Rasouli, M., Teneketzis, D., 2018. A POMDP approach to the dynamic defense of large-scale cyber networks. IEEE Trans. Inf. Forensics Secur. 13 (10), 2490–2505. doi:10.1109/TIFS.2018.2819967.

Muñoz-González, L., Sgandurra, D., Barrere, M., Lupu, E.C., 2019. Exact inference techniques for the analysis of bayesian attack graphs. IEEE Trans. Dependable Secure Comput. 16 (2), 231–244. doi:10.1109/TDSC.2016.2627033.

Muñoz-González, L., Sgandurra, D., Paudice, A., Lupu, E.C., 2017. Efficient attack graph analysis through approximate inference. ACM Trans. Privacy Secur. 20 (3), 1–30. doi:10.1145/3105760.

Murphy, K.P., Weiss, Y., Jordan, M.I., 1999. Loopy belief propagation for approximate inference: an empirical study. In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence. Morgan Kaufmann Publishers Inc., pp. 467–475. https://arxiv.org/abs/1301.6725

Nguyen, H.H., Nicol, D.M., 2020. Estimating loss due to cyber-attack in the presence of uncertainty. In: Proceedings - 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2020, pp. 361–369. doi:10.1109/TrustCom50675.2020.00057.

Nguyen, H.H., Palani, K., Nicol, D.M., 2017. An approach to incorporating uncertainty in network security analysis. In: ACM International Conference Proceeding Series, Part F1271, pp. 74–84. doi:10.1145/3055305.3055308.

Nguyen, T.H., Wright, M., Wellman, M.P., Singh, S., 2018. Multistage attack graph security games: heuristic strategies, with empirical game-theoretic analysis. Secur. Commun. Netw. 2018, 1–28. doi:10.1155/2018/2864873.

Noel, S., Jajodia, S., 2004. Managing attack graph complexity through visual hierarchical aggregation. In: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security - VizSEC/DMSEC '04. ACM Press, New York, New York, USA, p. 109. doi:10.1145/1029208.1029225.

Noel, S., Jajodia, S., 2005. Understanding complex network attack graphs through clustered adjacency matrices. In: 21st Annual Computer Security Applications Conference (ACSAC'05). IEEE, pp. 160–169. doi:10.1109/CSAC.2005.58.

Noel, S., Jajodia, S., 2014. Metrics suite for network attack graph analytics. In: Proceedings of the 9th Annual Cyber and Information Security Research Conference on - CISR '14. ACM Press, New York, New York, USA, pp. 5–8. doi:10.1145/2602087.2602117.

Noel, S., Jajodia, S., O'Berry, B., Jacobs, M., 2003. Efficient minimum-cost network hardening via exploit dependency graphs. In: 19th Annual Computer Security Applications Conference, 2003. Proceedings. IEEE, pp. 86–95. doi:10.1109/CSAC.2003.1254313.

Noel, S., Jajodia, S., Wang, L., Singhal, A., 2010. Measuring security risk of networks using attack graphs. Int. J. Next Gen. Comput. 1 (1), 135–147.

Noel, S., Robertson, E., Jajodia, S., 2004. Correlating intrusion events and building attack scenarios through attack graph distances. In: 20th Annual Computer Security Applications Conference. IEEE, pp. 350–359. doi:10.1109/CSAC.2004.11.

Ou, X., 2005. A logic-programming approach to network security analysis. Princeton University, USA Ph.D. thesis.

Ou, X., Boyer, W.F., McQueen, M.A., 2006. A scalable approach to attack graph generation. In: Proceedings of the 13th ACM Conference on Computer and Communications Security - CCS '06. ACM Press, New York, New York, USA, pp. 336–345. doi:10.1145/1180405.1180446.

Ou, X., Govindavajhala, S., Appel, A.W., 2005. MulVAL: a logic-based network security analyzer. In: Proceedings of the 14th conference on USENIX Security Symposium, Vol. 14. https://www.usenix.org/conference/14th-usenix-security-symposium/mulval-logic-based-network-security-analyzer

Ou, X., Singhal, A., 2012. The need for quantifying security. In: Kansas State University Technical Report, pp. 1–3. doi:10.1007/978-1-4614-1860-3_1.

Pamula, J., Jajodia, S., Ammann, P., Swarup, V., 2006. A weakest-adversary security metric for network configuration security analysis. In: Proceedings of the 2nd ACM Workshop on Quality of Protection - QoP '06. ACM Press, New York, New York, USA, p. 31. doi:10.1145/1179494.1179502.

Pearl, J., 1988. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc..

Pendleton, M., Garcia-Lebron, R., Cho, J.-H., Xu, S., 2017. A survey on systems security metrics. ACM Comput. Surv. 49 (4), 1–35. doi:10.1145/3005714. 1601.05792.

Peng Xie, Li, J.H., Xinming Ou, Peng Liu, Levy, R., 2010. Using Bayesian networks for cyber security analysis. In: 2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN). IEEE, pp. 211–220. doi:10.1109/DSN.2010.5544924.

Phillips, C., Swiler, L.P., 1998. A graph-based system for network-vulnerability analysis. In: Proceedings of the 1998 Workshop on New Security Paradigms, pp. 71–79. doi:10.1145/310889.310919.

Poolsappasit, N., Dewri, R., Ray, I., 2012. Dynamic security risk management using Bayesian attack graphs. IEEE Trans. Dependable Secure Comput. 9 (1), 61–74. doi:10.1109/TDSC.2011.34.

Ramos, A., Lazar, M., Filho, R.H., Rodrigues, J.J.P.C., 2017. Model-based quantitative network security metrics: asurvey. IEEE Commun. Surv. Tutor. 19 (4), 2704–2734. doi:10.1109/COMST.2017.2745505.

Ritchey, R., Ammann, P., 2000. Using model checking to analyze network vulnerabilities. In: Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000. IEEE Comput. Soc, pp. 156–165. doi:10.1109/SECPRI.2000.848453.

Roschke, S., Cheng, F., Meinel, C., 2011. A new alert correlation algorithm based on attack graph. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 6694 LNCS, pp. 58–67. doi:10.1007/978-3-642-21323-6_8.

Roschke, S., Cheng, F., Schuppenies, R., Meinel, C., 2009. Towards unifying vulnerability information for attack graph construction. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 5735 LNCS, pp. 218–233. doi:10.1007/978-3-642-04474-8_18.

Saha, D., 2008. Extending logical attack graphs for efficient vulnerability analysis. In: Proceedings of the ACM Conference on Computer and Communications Security, pp. 63–73. doi:10.1145/1455770.1455780.

Sawilla, R.E., Ou, X., 2008. Identifying critical attack assets in dependency attack graphs. In: European Symposium on Research in Computer Security. Springer, pp. 18–34. doi:10.1007/978-3-540-88313-5_2.

Singhal, A., Ou, X., 2011. Security risk analysis of enterprise networks using probabilistic attack graphs.

Sommestad, T., Sandström, F., 2015. An empirical test of the accuracy of an attack graph analysis tool. Inform. Comput. Secur. 23 (5), 516–531. doi:10.1108/ICS-06-2014-0036.

Stan, O., Bitton, R., Ezrets, M., Dadon, M., Inokuchi, M., Ohta, Y., Yagyu, T., Elovici, Y., Shabtai, A., 2021. Heuristic approach for countermeasure selection using attack graphs, 1–161906.10943. 10.1109/csf51468.2021.00003

Stan, O., Bitton, R., Ezrets, M., Dadon, M., Inokuchi, M., Yoshinobu, O., Tomohiko, Y., Elovici, Y., Shabtai, A., 2020. Extending attack graphs to represent cyber-attacks in communication protocols and modern IT networks. IEEE Trans. Dependable Secure Comput. 5971 (c), 1–18. doi:10.1109/TDSC.2020.3041999. 1906.09786.

Sun, X., Dai, J., Liu, P., Singhal, A., Yen, J., 2016. Towards probabilistic identification of zero-day attack paths. In: 2016 IEEE Conference on Communications and Network Security (CNS). IEEE, pp. 64–72. doi:10.1109/CNS.2016.7860471.

Swiler, L.P., Phillips, C., Ellis, D., Chakerian, S., 2001. Computer-attack graph generation tool. In: Proceedings - DARPA Information Survivability Conference and Exposition II, DISCEX 2001, Vol. 2, pp. 307–321. doi:10.1109/DISCEX.2001.932182.

Verendel, V., 2009. Quantified security is a weak hypothesis. In: Proceedings of the 2009 Workshop on New Security Paradigms Workshop - NSPW '09. ACM Press, New York, New York, USA, p. 37. doi:10.1145/1719030.1719036.

Wang, L., Islam, T., Long, T., Singhal, A., Jajodia, S., 2008. An attack graph-based probabilistic security metric. In: IFIP Annual Conference on Data and Applications Security and Privacy, pp. 283–296. doi:10.1007/978-3-540-70567-3_22.

Wang, L., Jajodia, S., Singhal, A., 2017. Network Security Metrics. Springer International Publishing, Cham doi:10.1007/978-3-319-66505-4.

Wang, L., Jajodia, S., Singhal, A., Cheng, P., Noel, S., 2014. k-Zero day safety: a network security metric for measuring the risk of unknown vulnerabilities. IEEE Trans. Dependable Secure Comput. 11 (1), 30–44. doi:10.1109/TDSC.2013.24.

Wang, L., Jajodia, S., Singhal, A., Noel, S., 2010. k-Zero day safety: measuring the security risk of networks against unknown attacks. In: European Symposium on Research in Computer Security, pp. 573–587. doi:10.1007/978-3-642-15497-3_35.

Wang, L., Liu, A., Jajodia, S., 2006. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. Comput. Commun. 29 (15), 2917–2933. doi:10.1016/j.comcom.2006.04.001.

Wang, L., Noel, S., Jajodia, S., 2006. Minimum-cost network hardening using attack graphs. Comput. Commun. 29 (18), 3812–3824. doi:10.1016/j.comcom.2006.06.018.

Wang, L., Singhal, A., Jajodia, S., 2007. Measuring the overall security of network configurations using attack graphs. In: Proceedings - IFIP Annual Conference on Data and Applications Security and Privacy, pp. 98–112. doi:10.1007/978-3-540-73538-0_9.

Wang, L., Singhal, A., Jajodia, S., 2007. Toward measuring network security using attack graphs. In: Proceedings of the 2007 ACM Workshop on Quality of Protection - QoP '07. ACM Press, New York, New York, USA, p. 49. doi:10.1145/1314257.1314273.

Wang, L., Zhang, M., Jajodia, S., Singhal, A., Albanese, M., 2014. Modeling network diversity for evaluating the robustness of networks against zero-day attacks. In: Network Security Metrics, pp. 494–511. doi:10.1007/978-3-319-11212-1_28.

Wei Li, Vaughn, R., 2006. Cluster security research involving the modeling of network exploitations using exploitation graphs. Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06). IEEE doi:10.1109/CCGRID.2006.1630921. pp. 26–26

Zeng, J., Wu, S., Chen, Y., Zeng, R., Wu, C., 2019. Survey of attack graph analysis methods from the perspective of data and knowledge processing. Secur. Commun. Netw. 2019. doi:10.1155/2019/2031063.

Zhang, M., Wang, L., Jajodia, S., Singhal, A., Albanese, M., 2016. Network diversity: a security metric for evaluating the resilience of networks against zero-day attacks. IEEE Trans. Inf. Forensics Secur. 11 (5), 1071–1086. doi:10.1109/TIFS.2016.2516916.

Kengo Zenitani is a PhD student at the Graduate Major in Mathematical and Computing Science, Department of Mathematical and Computing Science, School of Computing, Tokyo Institute of Technology. From 2014 to 2018, he has been a project researcher at the Graduate School of Information Science and Technology, the University of Tokyo. His focus is on the mathematical support for information security management, especially on the information security risk assessment. He is interested in the mathematically formulated and computerized support for decision-making to narrow the fluctuation caused by unstable decisions in a broader sense.