International Journal of Information Security (2022) 21:37–59
https://doi.org/10.1007/s10207-020-00533-4

REGULAR CONTRIBUTION

Automatic analysis of attack graphs for risk mitigation and prioritization on large-scale and complex networks in Industry 4.0

George Stergiopoulos[1,2] · Panagiotis Dedousis[2] · Dimitris Gritzalis[2]

Published online: 27 February 2021

Abstract
Threat models and attack graphs have been used more than 20 years by enterprises and organizations for mapping the actions of potential adversaries, analyzing the effects of vulnerabilities and visualizing attack scenarios. Although efficient when describing high-level interactions in simpler enterprise networks, they fall short in modern decentralized systems, especially in microservices architectures and multi-cloud environments with increased complexity and interactions. Most current research focuses on automatically generating attach graphs for such complex environments and deals with scaling and mapping issues, while neglecting to address the overall complexity of actually analyzing and extracting useful information from these overly convoluted models. In this paper, we present a method for automatically analyzing complex attack graphs both in microservices-based and multi-cloud infrastructures. We piggyback on previous research to automatically create complex attack graphs for such enterprise networks and use it as input to relate microservices, virtual system states and cloud services (represented as graph nodes) with prioritization algorithms that use mathematical graph series and group clustering. Our tool prioritizes existing vulnerabilities, analyzes the effect of system states to the overall network and proposes which system states, vulnerabilities and configurations have the biggest overall risk to the ecosystem, while taking into consideration every potential sub-attack path and subliminal path on an attack graph. We test the efficiency of our software on two real-world use cases: one multi-cloud enterprise network and a NetFlixOSS microservices Docker architecture.

Keywords Attack graph · Attack paths · Arborescence · Clustering · Closeness · Centrality · CVE · Derivation · Risk · Dependency · Microservices · Docker

1 Introduction

Researchers and industry practitioners use threat models [1], vulnerability assessments [2, 3], and asset-oriented risk assessments [4] both in IT and OT enterprise networks to model adversary behavior and predict malicious activities [5]. These tools are often used together or in isolation. Efficient threat models reflect multiple, complex attacks and produce meaningful results to be used by security officers to secure enterprise networks and mitigate existing cybersecurity risks.

A major issue when creating threat models is the inherent complexity of interactions between company services and systems. Modern companies utilize complex system interconnections that include cloud systems, microservices and virtual services (e.g., Kubernetes/Docker in cloud environments [6]) to support everyday business. Therefore, it is important to utilize efficient tools and techniques able to automatically model and analyze such enterprise networks, risks and configurations and highlight potential risks and underlying vulnerabilities.

Modern research has mostly focused on automatic tools that model networks and detect vulnerabilities. Proposed software automatically maps potential vulnerabilities and connected systems to form attack paths, where potential malicious actions are tied to detected vulnerabilities and

✉ Dimitris Gritzalis
dgrit@aueb.gr

George Stergiopoulos
g.stergiopoulos@aegean.gr

Panagiotis Dedousis
dedousisp@aueb.gr

[1] Department of Information and Communication Systems Engineering, University of the Aegean, Mytilene, Greece

[2] INFOSEC Laboratory, Department of Informatics, Athens University of Economics and Business, Athens, Greece

systems to form attack paths. Such models put IT staff in the adversary's position and have been proven crucial for security officers during risk mitigation [5, 7].

Still, threat models and attack graphs find it difficult to interpret and adequately analyze attacks on modern Industry 4.0 systems. Modern enterprise networks have more complex connections and corporate networks combine multiple systems and service environments with seemingly disjoint configuration issues [8]. Microservices and service-oriented architectures structure corporate systems as collections of loosely coupled services over virtual systems and Docker images. Other implementations structure enterprise networks over multi-cloud environments, with dedicated tunneling between each cloud.

In this context, security officers struggle to prioritize and remediate security findings provided by the assessments and solutions mentioned above. Existing automatic threat model tools and attack graph generators have come a long way toward automatically modeling and mapping complex enterprise networks, tackling scalability and detecting causality relations between adversary actions and system states, but suffer when interpreting produced models and prioritizing targets and mitigation solutions.

1.1 Contribution

This paper extends previous work on automatic attack graph generation and modeling to provide a new mathematical approach in analyzing and understanding attack graphs. Our methodology does not focus on improving upon attack graph generation research, but it picks up from where previous studies have ended to analyze already-generated attack graphs. We provide a novel solution that automatically proposes network security solutions for risk mitigation and prioritizes security control implementation on large-scale and complex networks in Industry 4.0. Tested systems include modern Docker and cloud environments. The proof-of-concept tool detects the highest risk attack paths and offers a metric analysis of existing vulnerability effects on the overall enterprise network. To our knowledge, no similar research can automatically analyze complex attack graphs and offer prioritization solutions for risk mitigation.

The idea is to use Edmonds' algorithm, graph centrality metrics and clustering on attack graphs weighted with risk assessment calculations to provide automated prioritization of systems and detected vulnerabilities. Logical attack graphs model potential attacks over interconnected system configuration states and their derived events. We produce a spanning arborescence and groups of tightly interdepended malicious events and states. This way, we can extract data to:

1. Prioritize detected attack paths based on their overall risk to the enterprise network,
2. Pinpoint present system configurations that introduce the highest risk on the overall system, and
3. Use Clustering techniques to detect system states with attack patterns that often contain functionally related vulnerabilities.

We use this output to propose which system states, vulnerabilities and configurations have the biggest overall risk to the ecosystem while considering every potential sub-attack path and subliminal path on an attack graph. We develop a standalone software and test its efficiency on two real-world use cases: one multi-cloud enterprise network and a NetFlixOSS microservices Docker architecture.

1.2 Structure

Section 2 briefly presents research publications that are relevant to our work and compares our contributions to existing literature. Section 3 introduces the main building blocks used in our methodology. Section 4 presents the algorithmic steps of our methodology, along with input and output for each step. In Sect. 5 we discuss our experiments and present our findings to validate the methodology. Section 6 concludes our work and focuses on current limitations, the limits of our current contribution and potential future challenges.

2 Related work

Various forms of attack graphs have been proposed for analyzing and evaluating the security of industry and corporate networks [7, 9–13]. Attack graphs can be categorized in two major types. In Phillips and Swiler [12], Sheyner et al. [13] authors consider each node that represents an entire system state and edges represent state transitions caused by a propagating attack, these types of attack graphs often are identified as state enumeration attack graphs [14]. In Ammann et al. [9], Ingols et al. [10], Noel et al. [11], Noel and Jajodia [14] authors identify each node as a system condition, not as an entire system state, in some form of logical sentence, and edges as the causality relations between the system conditions. These types of attack paths are recognized as dependency attack paths [15].

A broad range of publications exists that deals with attack paths and attack graph generation [9, 11, 12, 16, 17]. Advances on the sector enabled processing of attack graphs for complex networks of thousands of machines [7, 10]. While some focus on graph scalability issues [9, 11, 18], others focus on automatically generating detailed attack scenarios to depict potential adversary routes inside enterprise networks [19].

Sheyner et al. [13] utilize a finite-state machine model checker to calculate multi-stage, multi-host attack paths on a

network in the form of a scenario graph. Still, their approach scales poorly against graph generation time and graph size [20]. Following a similar approach, authors in Phillips and Swiler [12] applied model checking with a custom search engine to conduct the analysis of attack graphs, facing the same scalability problems.

Ammann et al. [9] address the scalability problem of the model checking-based attack graph methodologies by utilizing the monotonicity characteristic, where an attacker does not need to relinquish privileges he already gained since his ability to attack does not diminish. They implemented their algorithm in the Topological Vulnerability Analysis tool and provided a tangible understanding of how individual and combined vulnerabilities impact overall network security [16]

In Musa et al. [21], authors utilizing organization vulnerability assessments effectively model and produce attack graphs to quantitatively assess and analyze the attacks performed on the computing networks. Ivanov et al. [22] present an automated system based on a comprehensive method that includes calculation of security indicators, risk assessment and selection of protective measures, based on attack graphs for assessing the security risks in the smart infrastructure and choosing the protective measures. In Al Ghazo et al. [23], authors propose a model-checking-based automated attack graph generator and visualizer in order to analyze how interdependencies among existing vulnerabilities may be exploited by an adversary to stitch together an attack that can compromise a system. In Ibrahim et al. [24], authors present the Attack Scenarios Generation and Filtration Tool (ASGFT) which automatically generates all possible attack scenarios utilizing the description of an industrial control system.

In Ou [25] authors present the MulVal framework that utilizes a different attack tree mapping based on logical statements. They named their models "logical attack graphs" and utilize Datalog (a subset of Prolog) to express system configuration information as Datalog tuples and attack techniques and OS security semantics as Datalog rules. Their model uses two kinds of nodes: derivation nodes and fact nodes. Fact nodes are further divided into primitives and derived facts, while edges in their tree represent dependencies between these logical constructs. However, their approach fell short in terms of scalability even in medium sized network [7]. Authors in Ou [7] based on the work of [25] proposed a tool that has the ability of generating complete attack graphs for networks with thousands of machines by utilizing the monotonicity characteristic to validate the polynomial time of similar attack trees that use system configuration information. More recent research on "logical attack graphs" presents several alternate improvements on MulVal framework addressing several assumptions and limitations [26].

Recent advances have enabled computing attack graphs for large and complex networks [9, 10, 26]. However, even when attack graphs can be efficiently computed, the resulting size and complexity of the graphs is still too large and complex for a human to fully comprehend [27, 28]. Even for relatively small networks, produced attack graphs are still complex and difficult to understand and analyze for network administrators and security officers.

While network administrators and security officers utilizing attack graphs will quickly understand that attackers can penetrate the network, it is extremely challenging to identify which privileges, vulnerabilities, and assets are the most important/critical to the attacker success. Network administrators and security officers require a tool, which can automatically process the immense amount of information into a simple list of priorities, proposing specific risk mitigation actions that will help them to secure the network at hand fast, making efficient use of often limited human and financial resources [29].

Our proposed method utilizes multiple algorithms to achieve the analysis of such attack graphs (both logical and automatically generated): We implement (i) a previous methodology on automatic attack graph generation and modeling [7, 19, 25, 26], (ii) the risk dependency analysis for attack paths from [30, 31], and (iii) the clustering concept from [32], utilizing centrality metrics [33, 34].

While solutions proposed in Ammann et al. [9], Ibrahim et al. [24], Ingols et al. [10], Ivanov et al. [22], Lippmann and Ingols [20], Musa et al. [21], Ramadhan et al. [26] provide automatic modelling, mapping, and analysis of complex networks through attack path generation, still they lack the ability to automatically suggest mitigation solutions and prioritization. Our solution can also automatically analyze attacks graphs but continues ahead in providing solutions for risk mitigation and prioritization, detect highest risk attack paths, and offer metric analysis of existing vulnerability effects on the overall enterprise network addressing issues and limitations network administrators and security officers are facing [29]. In addition, our implementation is capable of handling and analyzing large and complex attack graphs addressing issues of scalability [9, 10, 26].

3 Building blocks

3.1 CVE metrics for risk estimation

Common Vulnerabilities and Exposures (CVE) is a database of entries that contains information on publicly known cybersecurity vulnerabilities on vendor systems and services. According to CVE, "CVE Entries are used in numerous cybersecurity products and services from around the world, including the U.S. National Vulnerability Database

(NVD)" (Common Vulnerability and Exposures [35] (2020). The NVD is "the U.S. government repository of standards-based vulnerability management data" (National Vulnerability Database [36] (2020).

NVD entries on publicly recorded CVEs utilize the CVSS 2.0 Severity and Metrics scoring system. The Common Vulnerability Scoring System (CVSS) provides a quantitative algorithm that captures key characteristics of a vulnerability and produces numerical scores that reflect each vulnerability's Severity (i.e. impact on a system) and Exploitability (ease of use from malicious users). These metrics are in line with international and industry standards on measuring the risk of a cybersecurity attack and underlying vulnerability [4, 37]. The common reference of risk as a cybersecurity assessment metric is the following Eq. 1:

The presented method calculates the Overall Attack Graph Risk as follows:

$$\text{Risk} = \text{Likelihood*}\ \text{Severity} == (\text{Threat*}\ \text{Vulnerability}) * \text{Severity} \quad (1)$$

In our case, we model attack paths based on CVE vulnerabilities that can be exploited in each step until the attacker reaches his goal. By applying risk measurements on CVE paths, we can calculate the risk of each attack path step using the aforementioned definition, where the feasibility of a threat and vulnerability is reflected by the CVE's Exploitability Subscore and Severity of an attack is reflected by CVE's Impact Subscore. This way, each attack graph connection that depicts an attacker's use of an exploit can have a quantifiable Risk metric.

3.2 Attack graphs

Attack path mapping (APM) is a methodology that identifies the highest risk assets in a corporate network and prioritizes controls, mitigations, and remediations by "mapping and validating all routes an attacker could use to reach a target" [8]. Attack paths depict information flow on a company's interdepended assets. Together they create an attack graph. Attack graphs map potential attacks in a system based on its configuration and detected vulnerabilities. They are a concise representation of all possible attack paths through a system that ends in a state where an intruder/attacker has successfully achieved his goal [38]. They depict ways that an adversary can exploit system vulnerabilities to achieve a desired state.

NIST proposes the use of attack graphs for forensic analysis and to aid investigators and security officers in identifying attack scenarios and pinpoint necessary countermeasures for mitigation (pre-attack) and evidence acquisition (post-attack) [19]. Attack graphs identify vulnerabilities in a network and how potential attackers can exploit these.

Following the conceptual modeling of [18], our attack risk graphs utilize derivation nodes and graph edges. In the presented methodology, each node represents a potential system state; i.e., a malicious action that produces a specific system state proven able to happen, since previous logical dependencies leading up to that derived fact are true. Nodes are the result of applying interaction rules iteratively on facts (represented by edge attributes).

A directed edge illustrates the dependency of a system state (node) V_j on another V_i, i.e. $V_i \rightarrow V_j$. Edge dependencies effectively construct attack traces with information to construct a logical dependency path [7]. Each edge depicts a different derivation, so the number of edges is equal to all possible states' derivations from observed system configurations [7, 18, 19]. Edges represent logical dependencies between potential system states and contain logical requirements as attributes. These attributes reflect the preconditions for an attacker to realize a step/achieve a system state. Attributes can either be configuration primitives (an implemented system configuration state) or derived facts detected during the analysis of primitives (e.g., vulnerability CVE-2019 exists on a web server). Primitives are generally configuration information of systems, as reported by the host and network scanners (e.g., "access control list granted" that indicates that a firewall permits access to a server)

3.2.1 Attack graph reduction

A graph reduction is used on the modeled attack multigraph to produce a simple graph. A weighted directed multigraph is a graph with multiple edges with the same start and end nodes. We reduce an attack multigraph by replacing all the given edges $E_{i,j}, \{i, j\} \in V$ between nodes with the respective optimum one, thus producing a simple weighted graph G' with $V' = V$ nodes and $E' \subseteq E$ edges. In a simple weight graph, each edge connects two distinct nodes, and no two edges connect the same pair of nodes. Also, the sum of the weights of all the edges in the reduced simple weighted graph should be optimum. Depending on the problem at hand, maximum or minimum weight defines the aforementioned optimum edge or graph. Based on risk assessment consensus and the fact that edge (attacks) weights depict risk, we consider optimum the edge with the maximum weight (worst-case scenario).

In case multiple maximum weight edges between two nodes exist, the reduction process can produce multiple alternates, simple graphs with the same overall weight. The produced graph represents possible attacks with the highest risk (maximum), and as such, we choose one of them for further analyses. We utilize graph reduction to *detect and remove low-risk attacks (minimum weight edges)* between connected assets and thus producing a disjunctive attack tree(s) with the higher overall risk.

For our implementation of the proposed reduction process, as presented in Algorithm 1, we utilize a simple

iterative algorithm, which iterates over all graph edges and uniquely stores the edge with the maximum weight for each pair of connected nodes. This way, we remove low-risk attacks (minimum weight edges) between connected assets. The running time of our implementation of the suggested algorithm for graph reduction is $O(|E|^2)$, where $|E|$ the number graph edges. After finding a reduced graph with the maximum weight, we can find all the alternative ones by simply altering edges with those with the same source and target node pair, and weight from the original graph.

3.3 Attack paths and risk chains

A multi-risk dependency analysis algorithm [39, 40] is used on the graph model. In our implementation, an edge denotes a derivation, i.e.,$V_i \rightarrow V_j$; thus it inherits a risk relation that is derived from a dependence of state V_j on an accessible/available vulnerability provided by state V_i. Based on risk assessment standards [4, 37], the methodology quantifies the risk of each graph edge using the impact $I_{i,j}$, and the likelihood $L_{i,j}$ of a vulnerability being exploited. The product of these two values is defined as the dependency risk $R_{i,j}$ of system state V_j due to its dependence on state V_i. The numerical value of each edge is the level of the cascade risk between the receiver and the sender node. This risk is depicted using a risk scale [1–10] where 10 is the most severe risk.

The algorithm assesses the nth-order cascading risks or attack paths using a recursive algorithm based on [31, 39]. If $S_1 \rightarrow S_2 \rightarrow \cdots \rightarrow S_n$ is an nth-order dependency between n system states S, with weights $R_{i,i+1} = L_{i,i+1}I_{i,i+1}$ corresponding to each first-order dependency of the attack path, then *the cascading risk* $R_{1,\ldots,n}$ *exhibited by* S_n *for this state dependency path* is computed as shown in Eq. 2.

The presented method calculates the Cascading risk of a system state dependency path as follows:

$$R_{1,\ldots,n} = L_{1,\ldots,n}I_{n-1,n} = \left(\prod_{i=1}^{n-1} L_{i,i+1}\right)I_{n-1,n} \tag{2}$$

The cumulative dependency risk (Eq. 3) is the overall risk exhibited by all the system states in the sub-chains of the nth-order dependency. If $S_1 \rightarrow S_2 \rightarrow \cdots \rightarrow S_n$ is a chain of system states dependencies of length n, then the cumulative dependency risk, denoted as $CR_{1,\ldots,n}$, is defined as the overall risk produced by an nth-order dependency:

The presented method calculates the Cumulative dependency risk of an nth-order dependency as follows:

$$CR_{1,\ldots,n} = \sum_{i=1}^{n} R_{1,\ldots,i} = \sum_{i=1}^{n}\left(\prod_{j=1}^{i-1} L_{j,j+1}\right)I_{i-1,i} \tag{3}$$

Algorithm 1. Graph reduction process using a simple iterative algorithm to find maximum weight simple graph(s)

```
Procedure ReduceGraph (graph)
    Inputs:
      An attack mutli-graph : graph
    Output:
      A reduced attack graph: reducedGraphEdges

    Let reducedGraphEdges as new List
    For each e in graph.edges do
        Let c = GetBySourceAndTarget(e, reducedGraphEdges)
        If c is not nothing then
            If e.weight > c.weight then
                //Replace edge c with edge e
                reducedGraphEdges.remove(c)
                reducedGraphEdges.add(e)
            End if
        Else
            reducedGraphEdges.add(e)
        End if
    End for
    FindAlternativeGraphs(reducedGraphEdges, graph)
End Procedure

Procedure FindAlternativeGraphs( reducedGraphEdges, graph)
    Inputs:
      An attack mutli-graph : graph
      A reduced attack graph: reducedGraphEdges
    Output:
      A list of reduced attack graphs: alternativeReducedGraphs

    Let alternativeReducedGraphs as new List
    For each x in reducedGraphEdges do
        For each y in graph.edges do
            If c.source = e.source and c.target = e.target then
                If e.weight = c.weight then
                    Let newAlternative = reducedGraphEdges.copy()
                    newAlternative.remove(x)
                    newAlternative.add(y)
                    alternateReducedGraphs.add(newAlternative)
                End if
            End if
        End for
    End for
End Procedure

Procedure GetBySourceAndTarget (e, edges)
    Inputs:
      A graph edge: e
      A list of edges: edges
    Output:
      The found edge or nothing in case of no match

    For each c in edges do
        If c.source = e.source and c.target = e.target then
            return c
        Else
            return nothing
        End if
    End for
End Procedure
```

Equation 4 computes the overall dependency risk as the sum of the dependency risks of the affected nodes in the chain due to a system state realized in the source node of the dependency chain. Using the total number n of all system state sub-chains (possible attack paths) and their cumulative dependency risks, the methodology can calculate the graph's overall risk G_r as the *sum of the cumulative dependency risk for each nth-order dependency* in the graph:

The presented method calculates the Overall attack graph risk as follows:

$$G_{\mathrm{r}} = \sum_{i=1}^{n} CR_{1,\ldots,n} \tag{4}$$

3.3.1 Risk chains calculation

An attack graph analysis always needs to analyze all potential chains. To calculate these risk chains of an attack graph, we must first find all its simple non-cyclic paths. Finding all such paths in any graph is a costly process, considering that in a fully connected graph of order V, where every node connects to every other node, there are $(|V|!)$ possible paths.

As presented in Algorithm 2, in our implementation we utilized a modified DFS algorithm. The algorithm starts with any input graph node, and at each recursive call, we attempt to extend a path (save visited nodes to an array) by visiting nodes (traversing the graph) until reaching a dead end. If the visited node does not have any output edges (dead end), calculate the cumulative dependency risk and output the result for the calculated path (array) containing the visited nodes. Finally, remove the last stored node in the array and start again with the $(n-1)$th node. We do this until we reach all the dead ends or reach the first node. In our study, we should note that we were only interested in paths (chains) up to a predefined length, which is up to length 6. The running time complexity of our implementation is $O\left(|V|^3 * \log(6)\right)$ for chains of length = 6 nodes, where V is the number of nodes of the input graph.

3.4 Graph arborescences

In graph theory, the Edmonds' algorithm is an algorithm for finding a spanning arborescence of minimum weight (sometimes called an optimum branching). A graph arborescence is the directed analog of the minimum spanning tree for directed graphs. The algorithm was proposed independently first by Yoeng-Jin Chu and Tseng-Hong Liu [41] and then by Jack Edmonds [42]. It takes a graph and a selected root node as input and creates a tree with directed edges where the root node only connects once with each node in the graph (i.e., there is exactly one directed path from the root to any other node). Only the root node has no edge directed toward it. Arborescence can either be Minimum Weight Arborescence as directed spanning trees with the minimum total weight, or Maximum Weight Arborescence, connecting the root node with all other nodes opting for the maximum possible total weight [43].

In our methodology, we produce Maximum and Minimum Weight Arborescence on automatically generated attack graphs and define the attacker's end goal (attack result) as the root node. This way, we produce the maximum (or minimum) weighted spanning trees from the attacker's end goal to potential attack surfaces. This modeling approach is particularly useful for understanding complex configurations and system states, and extracting the highest (or lowest) risk attack scenarios and corresponding system states.

Minimum weight arborescence algorithms are often used to find approximate solutions for complex problems such as the bottleneck Traveling Salesman [44], and network flow/reliability optimizations [45, 46].

Algorithm 2. Graph analysis through risk chain/path calculation

```
Procedure StartGraphAnalysis(graph, depth)
    Inputs:
      An attack graph : graph
      Depth of Analysis: depth
    Output:
      The attack paths/chains and their computed cumulative risks

    Let path = [ ]
    Let path_index = 0
    For each n in graph.nodes do
        Set n.visited = false
    End for
    Let startNode = random(graph.nodes) // Select a random node from the graph
    FindPaths( startNode , graph, depth, path, path_index)

End Procedure

Procedure FindPaths (n, graph, depth, path, path_index)
    Inputs:
      A graph node: n
      Attack graph: graph
      Depth of Analysis: depth
      Dependency Path array: path
      Path array index: path_index
    Output:
      The attack paths and their expected cumulative risks

    Let path_index += 1
    Let path[path_index] = n
    Let endFlag = false // End flag is used to check the dead end of the current path
    n.visited= true // We start from the input node and we consider it visited
    Let adjacentNodeList = FindAdjacentNodes(n, graph) // Find all adjacent nodes of the input node
    For each u in adjacentNodeList do
        If  u.visited = false then
            Let endFlag= true
            FindPaths (u, graph, depth, path, path_index)
        End if
    End for
    If  endFlag = false then
        If path.lentgh() > 1 and path.lentgh() <= depth then
            // Calculate and output the cumulative dependency of the n-order path
            CalculateCumulativeRisk(path)
        End if
    End if
    n.visited = false
    path_index - = 1
End Procedure
```

A spanning arborescence is a directed graph (digraph) in which, for a node V_i' called the root and any other node V, there is exactly one directed path from V_i' to V_i. An arborescence T of a weighted directed graph G is thus the directed-graph G' form of a rooted tree such that (i) T contains every node V of graph G, and (ii) T does not contain any cycle. A cycle is a graph path in which the first node corresponds to the last. A spanning arborescence of minimum weight can be perceived as the directed equivalent of the minimum spanning tree (MST) problem [47].

The problem of finding an optimum arborescence is trickier than its undirected version since for any cut $C = (Q, W)$,where $\{x, y\} \in \mathrm{E} | x \in Q, y \in W | Q \subseteq V, W \subseteq V$,of V, of a graph $G = (\mathrm{V}, \mathrm{E})$, if there is a least-cost edge

{x', y'}, x', y' ∈ V crossing that cut, that edge may not belong to all optimum arborescence's of G, hence the cut property does not apply. A minimum weight arborescence of a weighted directed graph can be found by algorithms such as those described in Bock [48], Chu and Liu [41], Edmonds [42].

The running time of Edmonds branching algorithm is $O(|V||E|)$, where $|E|$ is the number edges and $|V|$ the number of nodes [47, 49]. In our implementation the branching algorithm is based on the work of [50, 51] utilizing a Fibonacci heap [52] resulting $O(|E| + |V|log|V|)$ in running time [47, 49], and is constructed by design to find both maximum and minimum weight arborescence of an input graph.

We apply our implementation of Edmonds algorithm on the reduced weighted simple graph, produced from the reduction process, producing a single-node tree. The resulting graph $G' = (V, E')$ is directed and non-circular, since it is based on Edmond's algorithm, where $E' \subseteq E$.

3.5 Closeness centrality clustering

We use Clustering to build groups of system states reached by attack paths. Grouped states have related attack patterns and often contain functionally related vulnerabilities, such as remote code executions (RCEs) for a specific system, or vulnerabilities that are commonly exploited by previous steps. Such group clusters can be a powerful tool for vulnerability prioritization and understanding of influence on an overall network.

Centrality metrics are used in network models and quantify the influence of nodes and their relative importance within a graph [53–55]. Nodes with high centrality values have increased influence on other graph nodes and are, thus, good candidates for implementing risk mitigation controls [33, 34]. A *network asset group or cluster* can be defined as a subset of nodes [56].

We use the centrality metrics technique on graphs and their arborescence to *quantify the importance of each system state (i.e. a node), within the context of a cyber-attack scenario* (i.e. an attack path). Affected system states (nodes) with high centrality values are able to pinpoint vulnerabilities with the highest impact in an enterprise network. Different centrality metrics capture different aspects of network topology. In this methodology, we tested and use the Closeness centrality metric for attack state analysis and clustering. *Closeness centrality* calculates the average shortest path between node x and any other node in the graph [57]. Closeness centrality captures the average distance between every pair of nodes in a graph and assumes that nodes only affect nodes with whom they are directly connected through graph edges.

By experimenting with all potential centrality metrics, we found that Closeness provides the most realistic results when quantifying vulnerability influences. Degree centrality is not relevant since it is useful in multipath graphs where some nodes have numerous incoming or outgoing connections (more than four). Attack graphs rarely include such connections. Also, Betweenness centrality metrics offer similar results with Closeness. In contrast, Eigenvector centralities quantify the importance of neighboring nodes, which is irrelevant to attack graphs. During attacks, adjacent system states have no direct relation to current states besides their direct dependency. These direct dependencies are modeled in our model through edge connections.

3.5.1 Cluster formation

The Closeness centrality metric with midpoint on extreme values works best when identifying high influence nodes and clusters with a satisfying number of attack states and derivations [32]. Higher Closeness values are better candidates for system state (i.e., node) cluster generators. As presented in Algorithm 3, we use the midpoint of the calculated closeness centrality values from all nodes as a decision boundary. Nodes with greater influence than or equal to the midpoint are considered high-risk and are candidates for cluster generators. Instead, nodes with less influence are marked as low-risk. Cluster generators are then used over partitioning methods as key points to divide the population or system states into groups with similarities. For each pair of nodes in the set of high-risk nodes, we identify a single acyclic path that connects them, and we remove all its edges, thus splitting the graph and creating clusters. Possible orphan assets (single asset clusters) are assigned to the nearest cluster, based on the initial graph topology. The running time of our implementation is $O(|V| + |E|)$, where $|E|$ the number edges and $|V|$ the number of nodes of the input graph.

Algorithm 3. Graph clustering utilizing closeness centrality to identify high influence nodes and clusters with a satisfying number of attack states and derivations

```
Procedure Clustrering (graph)
    Inputs:
        A reduced graph : graph
    Output:
        A clustered attack graph: clusteredgraph

    Let maxCentrality = 0
    Let minCentrality = 0
    Let centralityMidpoint = 0
    For each node in graph.nodes do
        Let node.centrality = ClosenessCentrality(node , graph) //Calculates the closeness centrality metric
        If maxCentrality < node.centrality then
            maxCentrality = node.centrality
        End if
        If minCentrality > node.centrality then
            minCentrality = node.centrality
        End if
    End for
    //Calculate decision boundary
    centralityMidpoint = (maxCentrality + minCentrality) / 2
    Let highInfluenceNodes as New List
    Let lowInfluenceNodes as New List
    For each node in graph.nodes do
        If node.centrality >= centralityMidpoint then
            highInfluenceNodes.add(node)
        Else
            lowInfluenceNodes.add(node)
        End if
    End for
    Let clusteredGraph = graph // copy the original graph
    For each x in highInfluenceNodes do
        For each y in highInfluenceNodes do
            //Return a set of edges denoting a path in the specified graph, starting from node x and ending at node y
            Let path = getGraphPath(x, y, graph)
            For each edge in path do
                clusteredGraph.edges.remove(edge)
            End for
        End for
    End for
    //Find and assign to clusters orphan nodes
    For each n in clusteredGraph.nodes do
        Let adjacentNodeList = FindAdjacentNodes(n, clusteredGraph) // Find all adjacent nodes of the input node
        If adjacentNodeList is empty then
            // Find all adjacent nodes of the input node in original graph
            Let adjacentNodes = FindAdjacentNodes(n, graph)
            //Get an edge from the original graph that connects the current node with an adjacent
            Let edge = getEdge(n, adjacentNodes, graph)
            //Assign node to a cluster by restoring a removed edge
            clusteredGraph.edges.add(edge)
        End if
    End for

End Procedure
```

4 Methodology

Presented approach utilizes numerous techniques to achieve its goals. Each step of the presented methodology utilizes a distinct set of algorithms, where each one provides some insight on an IoT network under analysis and outputs information to be used as input by a following step. This process uses four fundamental building blocks:

Step 1 Attack graph modelling All potential attack paths that exist and can be exploited by adversaries are mapped onto a graph. In our experiments, we use the tools, enterprise networks and research models from previous research to

Table 1 Input/output data for each step of the methodology

	Input	Output
Attack graph modelling	Attack multigraph	Reduced attack graph (JSON)
Risk chains calculation	Reduced attack graph (JSON)	Weighted attack paths and rankings (CSV)
Spanning arborescence creation	Reduced attack graph (JSON) Attacker end-goals (e.g. "steal data from database")	Maximum Graph Arborescence Minimum Graph Arborescence
System states clustering	Maximum Graph Arborescence Minimum Graph Arborescence	Groups of nodes, (state clusters) Influence of attack steps on overall enterprise network

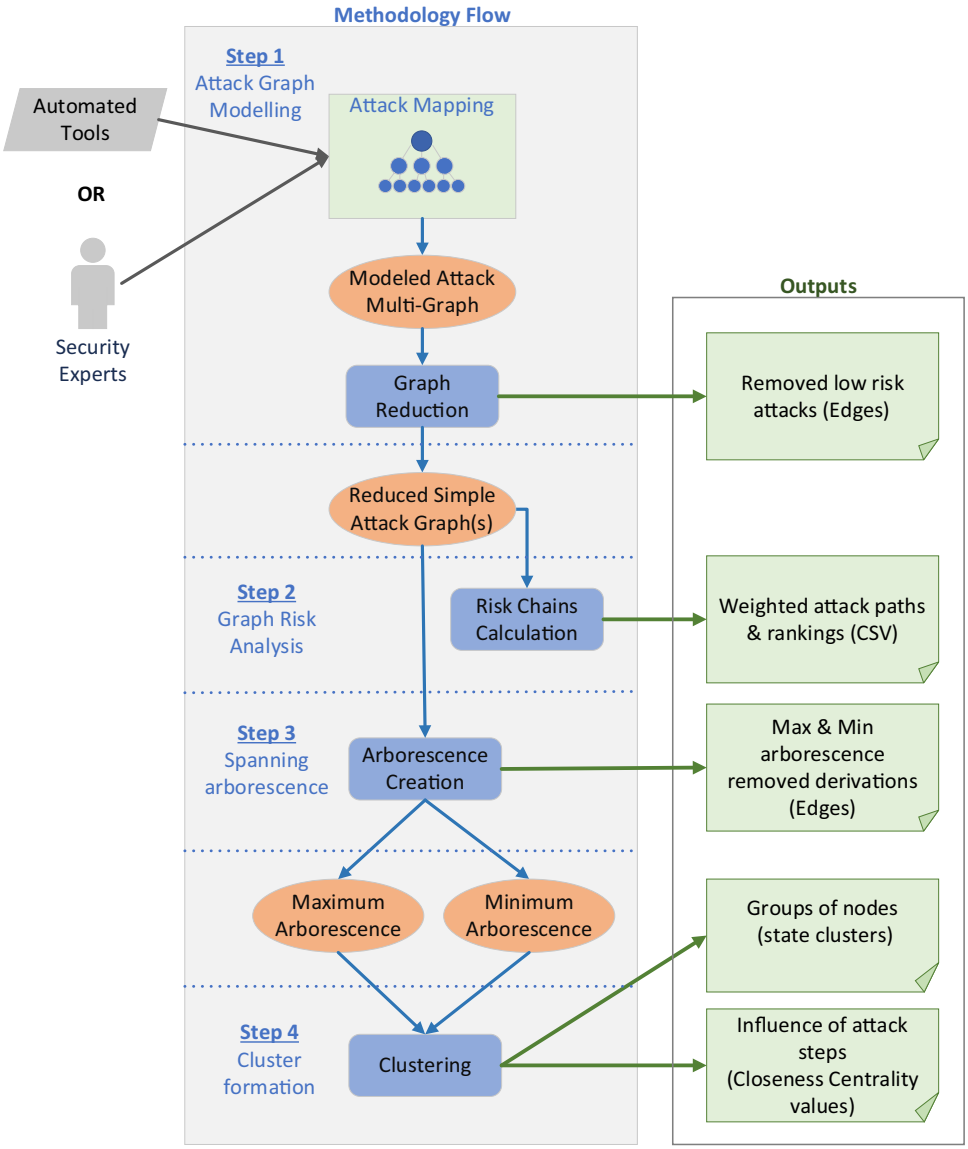

Fig. 1 Graphical representation of the overall methodology flow

automatically generate attack graphs and run our algorithm on their output [18, 19].

Step 2 Risk chains calculation We calculate the above reduced attack graph and then we compute all n-order attack paths as chains of nodes. This step outputs the cumulative dependency risk of each attack path and calculates the overall risk of all potential attack scenarios that exist (i.e. the entire network/graph risk). This step also sorts attack paths per risk and prioritizes them based on their influence on the overall enterprise network.

Step 3 Spanning arborescence creation In this step, input the reduced attack graph and create arborescence an arborescence for each one of the attackers' end-goal. For example, if attack paths map scenarios toward two different attacker end states (e.g. steal data from database server and access files in admin server), then two different arborescence will be created. This step outputs removed edges and arborescence paths, from attack surfaces to end goals. Max arborescence full attack paths introduce the highest risk on the overall system, while Min arborescence removed derivations (edges) must be considered for mitigation or removal from system preferences.

Step 4 Clustering of system states and output For each arborescence, the algorithm pre-computes the centrality metric values for each attack state (node) and creates clusters and rankings of system states.

The input and output for each step of the algorithm are summarized in Table 1. The overall methodology flow and the dependencies between its different building blocks are depicted in Fig. 1.

5 Evaluation

5.1 Tool implementation

The framework was developed as a client–server web application. Front end is implemented utilizing technologies such as HTML and JavaScript while the backend server and algorithm components are developed in Java Spring using the MySQL database.

5.2 Use case 1: multi-cloud enterprise network

To validate our methodology, we use a multi-cloud network topology consisting of two cloud infrastructures connected to the Internet through an external firewall.

5.2.1 Use case architecture

The first cloud server hosts three virtual machines, Mail server, Web server, and DNS server connected to a virtual switch. The second cloud server consists of two networks: public and private. The public network hosts two VMs; the first one hosts an SQL server; the second one hosts a NAT gateway server. The private network hosts one Admin server and three VMs (called VMs Group). Also, outside users can access the Web Server, and employees can access the SQL server through workstations inside the same LAN. Figure 2 depicts the enterprise network diagram for the use case 1 ecosystem.

Each server reflects an actual vendor-specific system and is vulnerable to a set of real-world CVE vulnerabilities. The impact of exploiting each vulnerability has been extracted from CVE (Common Vulnerability and Exposures [35] (2020). The overall attack graph is depicted in Tables 1 and 2. Figure 2 shows the attack graph, generated based on the vulnerabilities exist on the services in the second scenario. The attacker's goal is to compromise one of the VMs in VMs group in the private network, and/or compromise the database in the public network, by obtaining root access. As seen, the attacker may traverse different ways in this attack graph.

5.2.2 Tool analysis

Tables 2 and 3 present the system states (node with their IDs) that can be reached by attack path scenarios, along with the relevant derivation steps (edges). Presented edges are only worst-case scenario edges, as results from reducing the original multigraph from [18] to produce the reduced worst-case attack graph. Figure 3 provides a visual representation of the generated reduced graph (step 1).

Table 4 depicts the worst-case attack paths detected by our tool, ranked according to their cumulative risk from the used CVE vulnerabilities.

Figure 3 presents the attack graph as generated from the use case testbed and relevant vulnerabilities, while Fig. 4 depicts the produced arborescence from running the methodology on the reduced worst-case attack graph.

5.2.3 Results and prioritization

Clustering results point out that there exist system states and functionally related vulnerabilities with common attack steps that can be mitigated all-in-once by applying controls to their related attack origin states. Specifically, accessing the *VMGroups LICQ using user access privileges* (CVE-2001-0439) seems to have the highest overall influence in all possible attack scenarios, closely followed by the attack state where adversaries have root access to the *VMGroups Active Template Library* (CVE-2008-0015) (see Table 5). Prioritizing these two vulnerabilities will have the highest cumulative impact on all possible attack paths.

System states most frequently detected in highest risk attack paths involve. By combining the clustering and

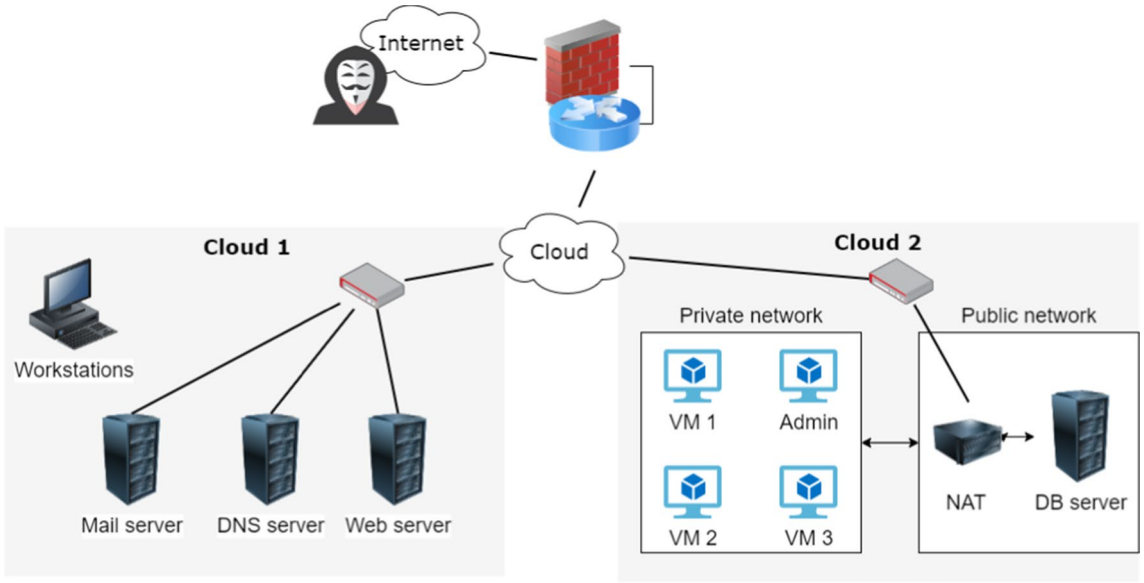

Fig. 2 Tool graphical representation of examined Netflix attack graph

arborescence results with the weighted risk ranking of attack paths, we see that that the highest risk states for the overall network that can be reached by attackers are *(i) VmGroups—LICQ (User access privilege) and (ii) WebServer (User access privilege)*. Related vulnerabilities (CVE-2001-0439 and CVE-2009-1535) should be heavily prioritized during remediation prioritization and risk mitigation.

Table 2 System states/nodes and their ID

System State	Node ID
Start	S1
AdminServer (Root access privilege)	S2
dbServer (execCode[user])	S3
dbServer (netAccess[tcp,1434])	S4
MailServer—ACLs (Root access privilege)	S5
MailServer—SMTP (Root access privilege)	S6
Nat Server—OpenSSH1 (User access privilege)	S7
NAT Server—OpenSSH2 (Root access privilege)	S8
Root access to VMs	S9
VmGroups—Active Template Library (Root access privilege)	S10
VmGroups—C Library (Root access privilege)	S11
VmGroups—LICQ (User access privilege)	S12
WebServer (Execute code)	S13
WebServer (netAccess[tcp,80])	S14
WebServer (User access privilege)	S15
WorkStation (execCode[user])	S16
WorkStation (accessMaliciousInput[secretary,'IE'])	S17
End	S18

Vulnerability *OpenSSH1 access on the NAT server* has the least influence and only in the lower risk (min arborescence) attack scenarios. This, coupled by the fact that this vulnerability is detected in both fastest attack paths (with least steps from start to finish), means that this vulnerability is key in order to perform the *easiest* attacks possible (albeit not the most influential on the enterprise network) (see Table 6).

5.3 Use case 2: Netflix OSS microservice system

The second testbed we used is a combination of containers provided by Netflix. The NetlixOSS high-level architecture of the testbed used in Use case 2 experiments is depicted in Fig. 5.

5.3.1 Use case architecture

The testbed is a public repository that realizes the spring cloud ecosystem. Figure 5 shows the different components of the system and Table 7 lists the main container parts and relevant repositories.

We used the tool from [18] to generate the NetflixOSS attack graph needed to feed it in our system. As presented in Ibrahim et al. [18], the Netflix OSS graph has linear attack dependencies, and each node connects to a small set of nodes. Containers have limited connections and keep outgoing dependencies to a minimum. In this example, each attack path can reach its end goal through multiple intermediate steps. Also, no directed edges (attack step) lead from system states with higher privileges to states with lower privileges. Also, no duplication of nodes exists (attacks that require

Table 3 Attack graph derivations and their risk

Edge ID	Vulnerability	Risk	CVE reference
E1	Browsing a malicious website	8	–
E2	Direct network access	0	–
E3	GNU C library loader flow	7	CVE-2010-3847
E4	GNU C library loader flow	7	CVE-2010-3847
E5	Heap Corruption in OpenSSH	10	CVE-2003-0693
E6	Heap Corruption in OpenSSH	10	CVE-2003-0693
E7	Improper Cookies Handler in OpenSSH	6	CVE-2007-4752
E8	LICQ buffer overflow	8	CVE-2001-0439
E9	MS SMV service stack buffer overflow	9	CVE-2008-4050
E10	MS SMV service stack buffer overflow	9	CVE-2008-4050
E11	MS video activex stack buffer overflow	9	CVE-2008-0015
E12	MS video activex stack buffer overflow	9	CVE-2008-0015
E13	Multi-hop access	8	CVE-2009-1918
E14	Multi-hop access	6	CVE-2018-7841
E15	Remote code execution in SMTP	10	CVE-2004-0840
E16	Remote exploit for a client program	8	CVE-2009-1918
E17	Remote exploit of a server program	4.8	CVE-2008-5416
E18	Remote exploit of a server program	6	CVE-2018-7841
E19	Remote exploit of db server	6	CVE-2008-5416
E20	Root access of VM Groups	7	CVE-2010-3847
E21	Root access of VM Groups	9	CVE-2008-0015
E22	Root access of VMs	9	CVE-2008-0015
E23	Root access of Admin Server	9	CVE-2008-0015
E24	Squid port scan	8	CVE-2001-1030
E25	WebDav Authentication Bypass	8	CVE-2009-1535

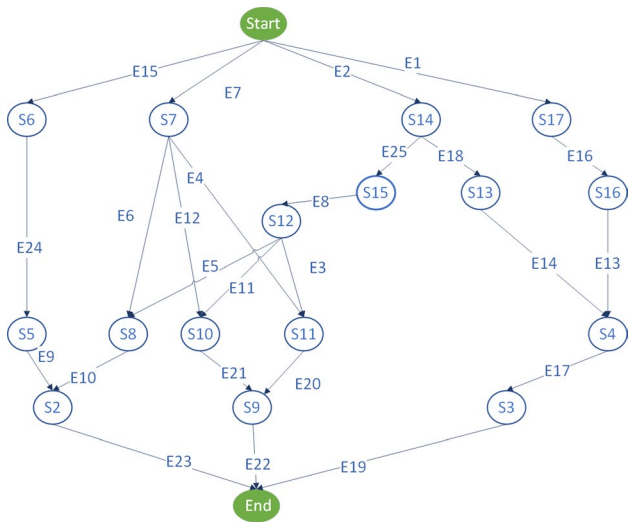

Fig. 3 A graphical representation of examined attack graph

Table 4 Top 6 derivatives dependency paths output from the risk analysis step (descending)

Paths	Cumulative dependency risk
S14–S15–S12–S10–S9	42
S14–S15–S12–S8–S2	42
S14–S15–S12–S11–S9	37.8
S6–S5–S2	34.5
S7–S10–S9	33.4
S7–S8–S2	33.4

The bold notation indicates which nodes are the most influential concerning the cumulative results calculated and presented in each table

Italic values indicate the highest ranked chains with the top risk grade detected

actions in the same asset/service), which is in line with the monotonicity property [18].

This specific use case utilizes microservices. Based on the presented architecture, each component builds as a set of services, and each service runs its processes and communicates through APIs. Each microservice may run in a virtual machine (hardware and OS visualization) or a container (only OS virtualization). Either way, the attacker's goal is to compromise one of the available servers (virtual machines or containers) by obtaining root access (admin privilege).

Netflix's original attack graph is a multigraph, and as such, a pair of nodes can be connected to more than one edge, resulting in an overly complex graph. On the original Netflix attack graph, each different edge between a pair of connected nodes corresponds to a mapped CVE with different weight/risk. Each node only connects through exactly one edge with the maximum potential weight/risk on the reduced attack graph. Table 8 depicts the potential system states (nodes) that can be reached from all different scenarios and attack paths of the reduced graph we created using the highest-risk CVE between each pair of connected nodes.

5.3.2 Tool analysis

The attack graph is generated by [18], and modeled by our tool. The overall graph and its graph edges are not presented in detail due to size restriction. Figure 6 depicts the graphical representations of the maximum (left) and minimum (right) weight arborescence attack graph for NetFlixOSS graph (entire preliminary input graph omitted due to size). For information on attack step derivations, preconditions and underlying CVE vulnerabilities present in NetFlixOSS components, please refer to "Appendix 1: "NetflixOSS CVE and edge derivations" at the end of this document.

Attack paths that exist on the graph have an order of equal or less than 6 (Table 9). The list depicted below sorts the top

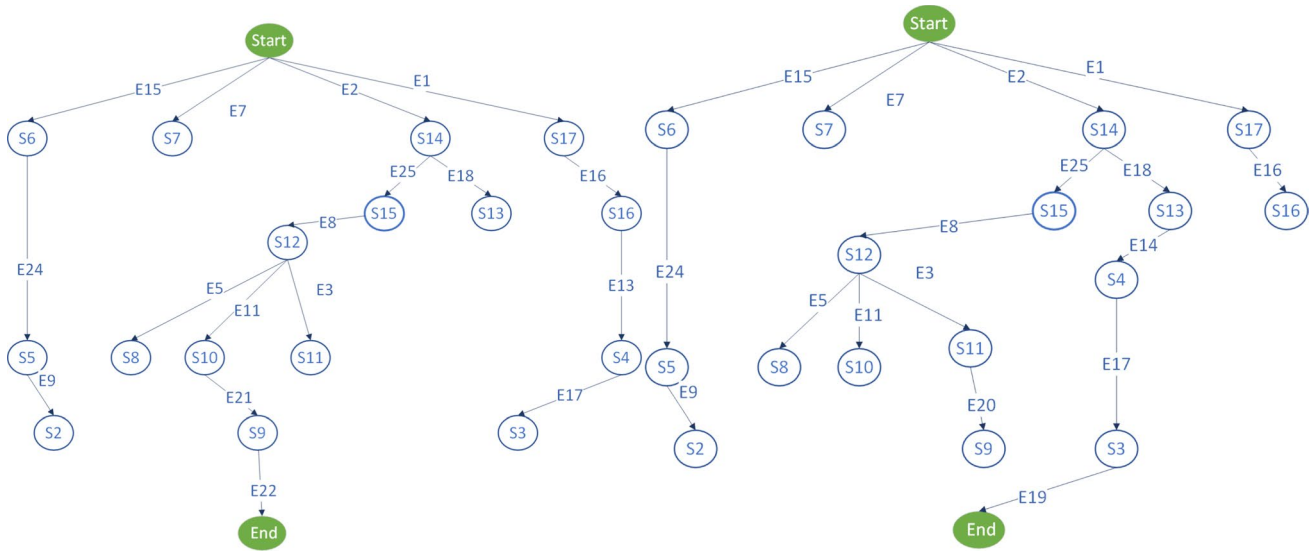

Fig. 4 Graphical representations of the max (left) and min (right) weight arborescence attack graphs

Table 5 Top 4 nodes/system states closeness centrality values for max and min weight arborescence (descending)

Max weight arborescence Top nodes		Min weight arborescence Top nodes	
Attack state—system	Closeness metric	Attack state—system	Closeness metric
VmGroups—LICQ (User access privilege)	0.071428571	Mail Server—SMTP (Root access privilege)	0.090909091
VmGroups—Active Template Library (Root access privilege)	0.058823529	WorkStation (access Malicious Input [secretary, 'IE'])	0.076923077
WebServer (User access privilege)	0.058823529	Mail Server—ACLs (Root access privilege)	0.071428571
WorkStation (access Malicious Input [secretary, 'IE'])	0.055555556	Nat Server-OpenSSH1 (User access privilege)	0.066666667

highest risk system states dependency paths according to the total cumulative risk. Top paths have all highest possible cumulative risk due to all having CVEs ranked at maximum risk (10.0).

Table 6 Key system vulnerabilities with highest overall impact and fastest attacks in all attack scenarios

Key vulnerabilities with highest impact overall	Key vulnerabilities for fastest attacks
VmGroups—LICQ (User access privilege) (CVE-2001-0439)	*OpenSSH1 access on the NAT server* (CVE-2003-0693)
WebServer (User access privilege) (CVE-2009-1535)	

5.3.3 Results and prioritization

Accessing the "*Eureka*" and "*Service a*" microservices seems to have the highest overall influence in all possible attack scenarios on the NetFlixOSS, closely followed by the attack state where adversaries have user access to the *Zuul microservice environment* (Table 10). Prioritizing vulnerabilities that affect these system states will have the highest cumulative impact on all possible attack paths.

System states most frequently detected in highest risk attack paths involve *turbine (User privilege) (S9)* and *rabbitmq (User privilege) (S19)*. Still, by combining the clustering and arborescence results with the weighted risk ranking of attack paths, we see that that these two may be included in all most high-risk attacks, but are not key attack steps when considering all possible attacks on NetFlixOSS.

Clustering relations show that both max and min arborescence have the same highest influential nodes, which means that these system states are indeed the highest influential states for all attack, and specifically: (i) *Eureka (ADMIN),*

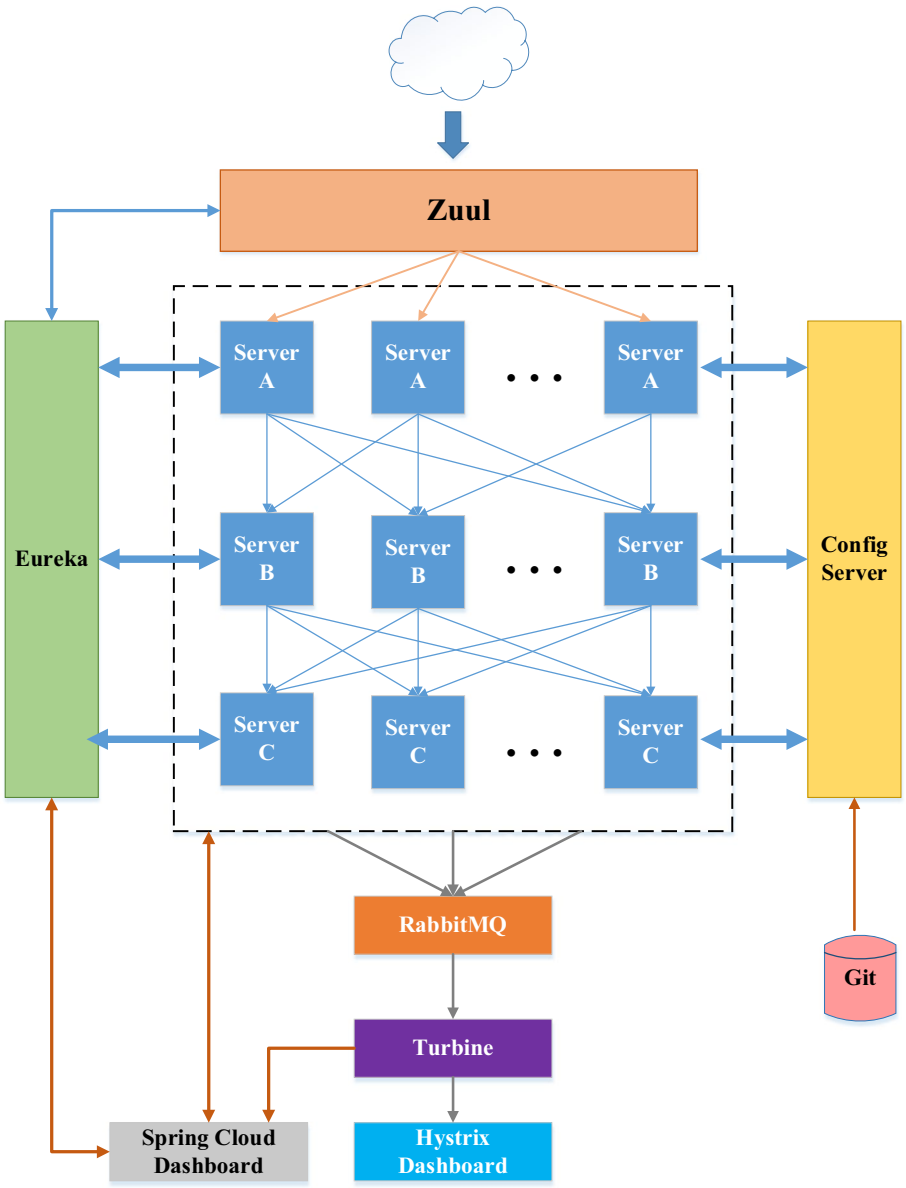

Fig. 5 NetflixOSS high-level architecture

Table 7 Netflix OSS microservices testbed components

Component	Repository
Eureka	https://github.com/Oreste-Luci/netflix-oss-example/tree/master/eureka-server
Config Service	https://github.com/Oreste-Luci/netflix-oss-example/tree/master/config-service
Turbine	https://github.com/Oreste-Luci/netflix-oss-example/tree/master/turbine
Hystrix Dashboard	https://github.com/Oreste-Luci/netflix-oss-example/tree/master/hystrix-dashboard
Microservice C: [Server micro-service for service B]	https://github.com/Oreste-Luci/netflix-oss-example/tree/master/service_c
Microservice B: [Server micro-service for service A]	https://github.com/Oreste-Luci/netflix-oss-example/tree/master/service_b
Microservice A: [Client micro-service]	https://github.com/Oreste-Luci/netflix-oss-example/tree/master/service_a
Zuul	https://github.com/Oreste-Luci/netflix-oss-example/tree/master/zuul
Spring Cloud Dashboard	https://github.com/Oreste-Luci/netflix-oss-example/tree/master/spring-cloud-dashboard
RabbitMq [histryx stats aggregator]	http://www.rabbitmq.com/
Docker Compose	https://github.com/Oreste-Luci/netflix-oss-example/tree/master/docker-compose

Table 8 System states/facts and node IDs' association

System State/Fact	Node ID	System State/Fact	Node ID
Config service (Admin privilege)	S1	Service b (Admin privilege)	S12
Config service (User privilege)	S2	Service b (User privilege)	S13
Eureka (Admin privilege)	S3	Service c (Admin privilege)	S14
Eureka (User privilege)	S4	Service c (User privilege)	S15
Hystrix dashboard (Admin privilege)	S5	Spring cloud dashboard (Admin privilege)	S16
Hystrix dashboard (User privilege)	S6	Spring cloud dashboard (User privilege)	S17
outside (Admin privilege)	S7	Turbine (Admin privilege)	S18
rabbitmq (Admin privilege)	S8	Turbine (User privilege)	S19
rabbitmq (User privilege)	S9	Zuul (Admin privilege)	S20
Service a (Admin privilege)	S10	Zuul (User privilege)	S21
Service a (User privilege)	S11	Outside (Admin privilege)	S22

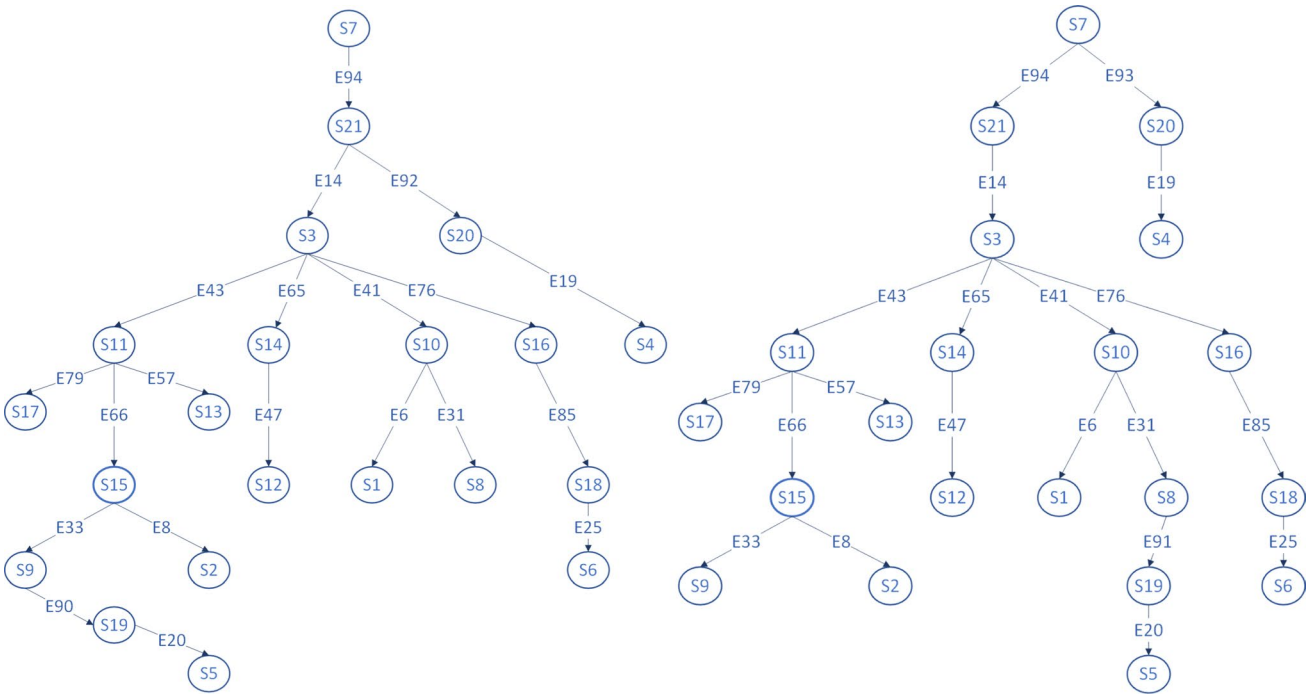

Fig. 6 Graphical representations of the max (left) and min (right) weight arborescence attack graph for NetFlixOSS

(ii) *Service a (User privilege),* (iii) *Zuul (User privilege), and* (iv) *Service a (ADMIN*). Their related vulnerabilities (CVE-2005-2541, CVE-2017-7376, CVE-2017-1000116, and CVE-2016-2108) should be heavily prioritized during remediation prioritization and risk mitigation.

Above analysis implies that some vulnerabilities are more important when we are trying to secure specific end services (Turbine (User privilege) (CVE-2011-2895), Rabbitmq (User privilege) (CVE-2016-2108)), while others are more important when trying to generally secure the overall network from as many attacks possible (see Table 11).

Table 9 Top 5 derivatives dependency paths output from the risk analysis step

Netflix Top 5 worst-risk attack paths
S9–S8–S19–S18–S6–S5
S13–S12–**S9**–S19–**S18**–S5
S17–S16–S12–**S9**–**S18**–S6
S11–S10–**S9**–S8–**S18**–S5
S11–S10–**S9**–S19–**S18**–S5

The bold notation indicates which nodes are the most influential concerning the cumulative results calculated and presented in each table

Table 10 Top 7 nodes/system states closeness centrality values for max and min weight arborescence (descending)

Max weight arborescence Top nodes		Min weight arborescence Top nodes	
Top state—system	Closeness metric	Top state—system	Closeness metric
Eureka (*Admin privilege*)	0.022727273	Eureka (*Admin privilege*)	0.022727
Service a (*User privilege*)	0.020408163	Service a (*User privilege*)	0.018868
Zuul (*User privilege*)	0.01754386	Service a (*Admin privilege*)	0.018182
Service a (*Admin privilege*)	0.016949153	Zuul (*User privilege*)	0.017544
Spring cloud dashboard (*Admin privilege*)	0.016949153	Spring cloud dashboard (*Admin privilege*)	0.016949
Service c (*User privilege*)	0.016666667	Service c (*Admin privilege*)	0.016393
Service c (*Admin privilege*)	0.016393443	Eureka (*Admin privilege*)	0.022727

Table 11 Key system vulnerabilities with highest overall impact and highest influence in all attack scenarios

Key vulnerabilities involved in highest impact attacks	Key vulnerabilities with the highest influence in all attack scenarios
Turbine (User privilege) (CVE-2011-2895)	*Eureka (Admin privilege)*—(CVE-2005-2541)
Rabbitmq (User privilege) (CVE-2016-2108)	*Service a (User privilege)*—(CVE-2017-7376)
	Zuul (User privilege)—(CVE-2017-1000116)
	Service a (Admin privilege) (CVE-2016-2108)

5.4 Efficiency and scalability

Analyzing a graph based on the proposed methodology and algorithms is a complex and computationally intensive issue that raises concerns about efficiency and scalability. The overall complexity from a single run is equal with the worst-case algorithm complexity. Among all algorithms used in our approach, the worst performance is attributed to risk chain calculation. Risk chain calculation requires us to detect all ($|V|!$) simple paths in an attack graph of order V. Finding all possible paths is a NP-Hard problem, since there is an exponential number of simple paths. The running time complexity of the algorithm is $O\left(|V|^3 * \log(6)\right)$ for chains of length = 6 nodes, where V is the number of nodes.

To cope with such a computationally intensive problem, we selected the Neo4J graph database as the main building block of our tool implementation. Graph databases are storage systems optimized for graph calculations and provide index-free adjacency. They model data more effectively than relational databases, especially when relationships between elements are the driving force for data model design [58, 59]. In a graph database, every node only needs to know the nodes to which it is connected (i.e., its edges). This allows a graph database system to use graph theory to analyze the edges of a graph effectively. Graph databases scale naturally to large data sets and/or to data sets with frequently-changing or on-the-fly schema [59].

We used the Neo4J graph database (Neo4j 2020) to implement the attack graph analysis tool due to its scalability, efficiency and implemented functionality on analyzing graph models with multiple attributes. According to research, Neo4J outperforms other systems and alternative libraries in (a) load time for millions of elements as well as in (b) time required to compute the total paths and detect the shortest path of an examined graph [60, 61, 62, 34, 58, 63]. In addition, Neo4j greatly outperforms relational databases such as MySQL in traversal tests like the one needed for our risk chain calculation. Also, Neo4J was shown to achieve the best performance among other popular graph databases in most benchmarks for graphs with over 20 M nodes and an approximate mean node degree equal to 10 [64].

Table 12 shows part of the output results of a comparative analysis presented in Kolomičenko et al. [64] demonstrating low execution times for extremely large graphs, thus proving the efficiency and scalability of graph theory algorithms in Neo4j. These numbers are at least 3 times bigger, in order of magnitude, that what is needed for our experiments (see below).

Neo4j uses property graph models, where nodes can have numerous labels as attributes and nodes and relationships can hold arbitrary properties (key-value pairs) [62, 58]. Neo4j facilitates the modeling of dependent attack path states in weighted, directed graphs and can work efficiently for up to millions of nodes without significant delays in everyday PC systems.

The tool was developed and tested on an Intel Core-i7 with 16 GB of RAM and an SSD. The number of nodes and edges greatly affects the execution time of any algorithm that analyzes graph models. Even though attack graphs and attack multigraphs differ on how the model system states and steps, we still believe that comparing scalability performance is feasible, since all graphs utilize similar concepts.

Table 12 Neo4j execution times for pertinent graph theory algorithms (graph nodes have an approximate mean node degree equal to 10)

Graph Size (nodes)	Time (ms)	
	BFS algorithm	Dijkstra's algorithm
10,000	~5	~100
50,00,000	~20	~500
10,000,000	~70	~1300
20,000,000	~110	~2900

Table 13 Methodology steps execution times

Methodology steps		Execution time (s)
Use case 1	Graph reduction	~6
	Risk chains calculation	~12
	Arborescence creation	~4
	Cluster formation	~6
	Overall execution	~28
NetFlixOSS	Graph reduction	~9
	Risk chains calculation	~22
	Arborescence creation	~6
	Cluster formation	~7
	Overall execution	~45

Table 13 shows the execution times for both use cases, and the corresponding times for each methodology step.

6 Conclusions

This paper presented a method for automatic analysis of generated attacks using a set of different mathematical models and algorithms. Experiments included two real-world microservices environments that realize complex, modern enterprise networks with seemingly disjoint configurations and dependencies.

To our knowledge, current literature on attack graphs mostly focuses on the automatic generation and scalability issues, rather in analyzing complex information within them and trying to extract useful information for security control prioritization and proper risk mitigation.

The proposed framework utilizes graph modeling algorithms such as mathematical series analysis, clustering and optimization to extract information about the effect of vulnerabilities and attack steps on enterprise networks. We conceptualize attack scenarios and extract the impact of each step from all possible attacks on the system. To this end, we have extended previous automatic generation methodologies with two features: Prioritizing detected vulnerabilities and analyze the effect of system states to the overall network for proposing which system states, vulnerabilities, and configurations have the biggest overall risk to the ecosystem. Our approach takes into consideration every potential sub-attack path and subliminal path on an attack graph.

Preliminary tests on actual microservices infrastructures and multi-cloud environments show that the presented approach looks efficient and trustworthy for attack graphs of several thousands of interconnections.

Finally, in order to validate the analysis and prioritization results we compared the output results with the corresponding practices and procedures that can actually be applied to the infrastructure and relevant networks, as they emerged from the manual analysis by the industrial cyber security experts. The comparison verified the accuracy of the analysis and the results for the two demonstrated Use Cases.

6.1 Restrictions

The presented approach should meet a number of requirements to be effective. First, information dissemination between asset owners, security officers and consultants be of utmost importance.

A white-box approach is needed to map attack paths, which requires considerable employee time. Also, it takes for granted that a risk assessment or a business impact assessment exists, from which to draw information on asset risk and potential consequences from security incidents. Without prior work, this approach will not have the necessary input to produce any results.

Finally, the attack path analyzer provides a comprehensive set of attack paths and proposes possible minimizations, but does not output an exhaustive set of every possible combination of attacks and potential mitigation schemas. Instead, it focuses only on worst-case scenarios that exploit the worst CVE available between every service or microservice connection and dependency. In other words, detections of high-risk attack paths and high-risk connection removal proposals are always true positives in terms of high risk. Still, these findings are not necessarily the result of exhaustive state analysis.

Compliance with ethical standards

Conflict of interest None of the authors have received any research grants. None of the authors have received a speaker honorarium from any company. All authors declare that none of them has any conflict of interest.

Appendix 1: NetflixOSS CVE and edge derivations

Edge ID	Vulnerability	Risk	Source Node	Target Node	Edge ID	Vulnerability	Risk	Source Node	Target Node
E1	CVE-2005-2541	10	S15	S1	E48	CVE-2017-1000116	10	S13	S12
E2	CVE-2005-2541	10	S14	S1	E49	CVE-2017-1000116	10	S11	S12
E3	CVE-2005-2541	10	S13	S1	E50	CVE-2017-1000116	10	S10	S12
E4	CVE-2005-2541	10	S12	S1	E51	CVE-2017-1000116	10	S4	S12
E5	CVE-2005-2541	10	S11	S1	E52	CVE-2017-1000116	10	S3	S12
E6	CVE-2005-2541	10	S10	S1	E53	CVE-2017-7376	10	S17	S13
E7	CVE-2005-2541	10	S2	S1	E54	CVE-2017-7376	10	S16	S13
E8	CVE-2017-7376	10	S15	S2	E55	CVE-2017-7376	10	S15	S13
E9	CVE-2017-7376	10	S14	S2	E56	CVE-2017-7376	10	S14	S13
E10	CVE-2017-7376	10	S13	S2	E57	CVE-2017-7376	10	S11	S13
E11	CVE-2017-7376	10	S12	S2	E58	CVE-2017-7376	10	S10	S13
E12	CVE-2017-7376	10	S11	S2	E59	CVE-2017-7376	10	S4	S13
E13	CVE-2017-7376	10	S10	S2	E60	CVE-2017-7376	10	S3	S13
E14	CVE-2005-2541	10	S21	S3	E61	CVE-2017-13090	9.3	S15	S14
E15	CVE-2005-2541	10	S20	S3	E62	CVE-2017-13090	9.3	S11	S14
E16	CVE-2005-2541	10	S11	S3	E63	CVE-2017-13090	9.3	S10	S14
E17	CVE-2005-2541	10	S4	S3	E64	CVE-2017-13090	9.3	S4	S14
E18	CVE-2017-7376	10	S21	S4	E65	CVE-2017-13090	9.3	S3	S14
E19	CVE-2017-7376	10	S20	S4	E66	CVE-2009-2347	9.3	S11	S15
E20	CVE-2017-16997	9.3	S19	S5	E67	CVE-2009-2347	9.3	S10	S15
E21	CVE-2017-16997	9.3	S18	S5	E68	CVE-2009-2347	9.3	S4	S15
E22	CVE-2017-16997	9.3	S6	S5	E69	CVE-2009-2347	9.3	S3	S15
E23	CVE-2017-16997	9.3	S6	S5	E70	CVE-2017-1000116	10	S17	S16
E24	CVE-2019-3855	9.3	S19	S6	E71	CVE-2017-1000116	10	S15	S16
E25	CVE-2019-3855	9.3	S18	S6	E72	CVE-2017-1000116	10	S14	S16
E26	CVE-2016-2842	10	S15	S8	E73	CVE-2017-1000116	10	S11	S16
E27	CVE-2016-2842	10	S14	S8	E74	CVE-2017-1000116	10	S10	S16
E28	CVE-2016-2842	10	S13	S8	E75	CVE-2017-1000116	10	S4	S16
E29	CVE-2016-2842	10	S12	S8	E76	CVE-2017-1000116	10	S3	S16
E30	CVE-2016-2842	10	S11	S8	E77	CVE-2017-7376	10	S15	S17
E31	CVE-2016-2842	10	S10	S8	E78	CVE-2017-7376	10	S14	S17
E32	CVE-2016-2842	10	S9	S8	E79	CVE-2017-7376	10	S11	S17
E33	CVE-2016-2108	10	S15	S9	E80	CVE-2017-7376	10	S10	S17
E34	CVE-2016-2108	10	S14	S9	E81	CVE-2017-7376	10	S4	S17
E35	CVE-2016-2108	10	S13	S9	E82	CVE-2017-7376	10	S3	S17
E36	CVE-2016-2108	10	S12	S9	E83	CVE-2017-13090	9.3	S19	S18
E37	CVE-2016-2108	10	S11	S9	E84	CVE-2017-13090	9.3	S17	S18

Edge ID	Vulnerability	Risk	Source Node	Target Node	Edge ID	Vulnerability	Risk	Source Node	Target Node
E38	CVE-2016-2108	10	S10	S9	E85	CVE-2017-13090	9.3	S16	S18
E39	CVE-2017-1000116	10	S11	S10	E86	CVE-2017-13090	9.3	S9	S18
E40	CVE-2017-1000116	10	S4	S10	E87	CVE-2017-13090	9.3	S8	S18
E41	CVE-2017-1000116	10	S3	S10	E88	CVE-2011-2895	9.3	S17	S19
E42	CVE-2017-7376	10	S4	S11	E89	CVE-2011-2895	9.3	S16	S19
E43	CVE-2017-7376	10	S3	S11	E90	CVE-2017-7376	10	S9	S19
E44	CVE-2017-1000116	10	S17	S12	E91	CVE-2011-2895	9.3	S8	S19
E45	CVE-2017-1000116	10	S16	S12	E92	CVE-2017-1000116	10	S21	S20
E46	CVE-2017-1000116	10	S15	S12	E93	CVE-2017-13090	9.3	S22	S20
E47	CVE-2017-1000116	10	S14	S12	E94	CVE-2017-7376	10	S22	S21

References

1. Kordy, B., Pièvre-Cambacédès, L., Schweitzer, P.: DAG-based attack and defense modeling: don't miss the forest for the attack trees. Comput. Sci. Rev. **13–14**, 1–38 (2014). https://doi.org/10.1016/j.cosrev.2014.07.001
2. Acunetix: (2008) http://www.acunetix.com/vulnerability-scanner/
3. Deraison, R.: Nessus (1999). https://www.tenable.com/products/nessus
4. BS ISO/IEC 27001: Information technology–security techniques–information security management systems–requirements (2013)
5. Cerotti, D., Raiteri, D.C., Dondossola, G., Egidi, L., Franceschinis, G., Portinale, L., Terruggia, R.: A Bayesian network approach for the interpretation of cyber attacks to power systems. In: ITASEC (2019)
6. Sanders, S., Border, C.: Private cloud deployment with docker and kubernetes. J. Comput. Sci. Coll. **33**, 58–59 (2018)
7. Ou, X., Boyer, W., McQueen, M.: A scalable approach to attack graph generation. In: Proceedings of the 13th ACM Conference on Computer and Communications Security—CCS'06. pp. 336–345. ACM Press, USA (2006)
8. Whitcombe, M.: What is attack graph mapping (2020) https://www.f-secure.com/en/consulting/our-thinking/what-is-attack-path-mapping
9. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. In: Proceedings of the 9th ACM conference on Computer and communications security—CCS'02, p. 217. ACM Press, Washington, DC, USA (2002)
10. Ingols, K., Lippmann, R., Piwowarski, K.: Practical attack graph generation for network defense. In: 2006 22nd Annual Computer Security Applications Conference (ACSAC'06). pp. 121–130. IEEE, USA (2006)
11. Noel, S., Jajodia, S., O'Berry, B., Jacobs, M.: Efficient minimum-cost network hardening via exploit dependency graphs. In: Proceedings of the 19th Annual Computer Security Applications Conference. p. 86. IEEE Computer Society, USA (2003)
12. Phillips, C., Swiler, L.P.: A graph-based system for network-vulnerability analysis. In: Proceedings of the 1998 Workshop on New security paradigms—NSPW'98. pp. 71–79. ACM Press, USA (1998)
13. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.: Automated generation and analysis of attack graphs. In: Proceedings 2002 IEEE Symposium on Security and Privacy. pp. 273–284. IEEE Comput. Soc, USA (2002)
14. Noel, S., Jajodia, S.: 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. p. 109. ACM Press, USA (2004)
15. Sawilla, R., Ou, X.: Identifying Critical Attack Assets in Dependency Attack Graphs. In: Computer Security—ESORICS 2008. pp. 18–34. Springe (2008)
16. Jajodia, S., Noel, S., O'Berry, B.: Topological analysis of network attack vulnerability. In: Managing Cyber Threats. pp. 247–266. Springer-Verlag, New York (2005)
17. Tidwell, T., Larson, R., Fitch, K., Hale, J.: Modeling internet attacks. In: Proceedings of the 2001 IEEE Workshop on Information Assurance and security. United States Military Academy, USA (2001)
18. Ibrahim A, Bozhinoski S, Pretschner A (2019) Attack graph generation for microservice architecture. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. pp. 1235–1242. ACM, Cyprus (2019)
19. Liu, C., Singhal, A., Wijesekera, D.: Mapping evidence graphs to attack graphs. In: 2012 IEEE International Workshop on Information Forensics and Security (WIFS). pp. 121–126 (2012)
20. Lippmann, R., Ingols, K.: An Annotated review of past papers on attack graphs. Presented at the (2005)
21. Musa, T., Yeo, K., Azam, S., Shanmugam, B., Karim, A., Boer, F., Nur, F., Faisal, F.: Analysis of complex networks for security issues using attack graph. In: 2019 International Conference on Computer Communication and Informatics (ICCCI). pp. 1–6. IEEE, India (2019)
22. Ivanov, D., Kalinin, M., Krundyshev, V., Orel, E.: Automatic security management of smart infrastructures using attack graph and risk analysis. In: 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4). pp. 295–300. IEEE, United Kingdom (2020)

23. Al Ghazo, A., Ibrahim, M., Ren, H., Kumar, R.: A2G2V: automatic attack graph generation and visualization and its applications to computer and SCADA networks. IEEE Trans. Syst. Man Cybern. Syst. **50**, 3488–3498 (2020). https://doi.org/10.1109/TSMC.2019.2915940
24. Ibrahim, M., Alsheikh, A., Al-Hindawi, Q.: Automatic attack graph generation for industrial controlled systems. In: Recent Developments on Industrial Control Systems Resilience. pp. 99–116. Springer International Publishing, Cham (2020)
25. Ou, X., Govindavajhala, S.: Mulval: A logic-based network security analyzer. In: In 14th USENIX Security Symposium. pp. 113–128 (2005)
26. Ramadhan, M., Gondokaryono, Y., Arman, A.: Network Security Risk Analysis using Improved MulVAL Bayesian Attack Graphs. IJEEI **7**, 735–753 (2015). https://doi.org/10.15676/ijeei.2015.7.4.15
27. Noel, S., Jacobs, M., Pramod, K. Jajodia, S.: Multiple coordinated views for network attack graphs. In: IEEE Workshop on Visualization for Computer Security, 2005. (VizSEC 05). pp. 99–106 (2005)
28. Williams L, Lippmann R, Ingols K (2008) An Interactive Attack Graph Cascade and Reachability Display. In: VizSEC 2007: Proceedings of the Workshop on Visualization for Computer Security. pp. 221–236. Springer (2008)
29. Dewri, R., Poolsappasit, N., Ray, I., Whitley, D.: 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. p. 204. ACM Press, USA (2007)
30. Homer, J.: A sound and practical approach to quantifying security risk in enterprise networks. In: CiteSeerX (2009)
31. Stergiopoulos, G., Kotzanikolaou, P., Theocharidou, M., Lykou, G., Gritzalis, D.: Time-based critical infrastructure dependency analysis for large-scale and cross-sectoral failures. Int. J. Crit. Infrastruct. Prot. **12**, 46–60 (2016). https://doi.org/10.1016/j.ijcip.2015.12.002
32. Stergiopoulos, G., Dedousis, P., Gritzalis, D.: Automatic network restructuring and risk mitigation through business process asset dependency analysis. Comput. Secur. **96**, 101869 (2020). https://doi.org/10.1016/j.cose.2020.101869
33. Oldham, S., Fulcher, B., Parkes, L., Arnatkevičiūtė, A., Suo, C., Fornito, A.: Consistency and differences between centrality measures across distinct classes of networks. PLoS ONE. **14**, e0220061 (2019). https://doi.org/10.1371/journal.pone.0220061
34. Stergiopoulos, G., Kotzanikolaou, P., Theocharidou, M., Gritzalis, D.: Risk mitigation strategies for critical infrastructures based on graph centrality analysis. Int. J. Crit. Infrastruct. Prot. **10**, 34–44 (2015). https://doi.org/10.1016/j.ijcip.2015.05.003
35. Common Vulnerability and Exposures (MITRE) (2020). https://cve.mitre.org/cve/
36. National Vulnerability Database (NIST) (2020). https://nvd.nist.gov/
37. NIST SP 800-30: Guide for conducting risk assessments. National Institute of Standards and Technology, USA (2012)
38. Jha, S., Sheyner, O., Wing, J.: Two formal analyses of attack graphs. In: Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15. pp. 49–63. IEEE, Canada (2002)
39. Kotzanikolaou, P., Theoharidou, M., Gritzalis, D.: Assessing n-order dependencies between critical infrastructures. IJCIS. (2013). https://doi.org/10.1504/IJCIS.2013.051606
40. Kotzanikolaou, P., Theoharidou, M., Gritzalis, D.: Interdependencies between critical infrastructures: analyzing the risk of cascading effects. In: Critical Information Infrastructure Security. pp. 104–115. Springer (2013)(b)
41. Chu, Y.J., Liu, T.H.: On the shortest arborescence of a directed graph. Sci. Sinica **14**, 1396–1400 (1965)
42. Edmonds, J.: Optimum branchings. J. Res. Natl. Bur. Stan. Sect. B. Math. Math. Phys. **71B**, 233 (1967). https://doi.org/10.6028/jres.071B.032
43. Guignard, M., Rosenwein, M.: An application of lagrangean decomposition to the resource-constrained minimum weighted arborescence problem. Networks **20**, 345–359 (1990). https://doi.org/10.1002/net.3230200306
44. Carpaneto, G., Martello, S., Toth, P.: An algorithm for the bottleneck traveling salesman problem. Oper. Res. **32**, 380–389 (1984). https://doi.org/10.1287/opre.32.2.380
45. Coscia, M.: Using arborescences to estimate hierarchicalness in directed complex networks. PLoS ONE **13**, e0190825 (2018). https://doi.org/10.1371/journal.pone.0190825
46. Glover, F.: Flows in arborescences. Manage. Sci. **17**, 568–586 (1971). https://doi.org/10.1287/mnsc.17.9.568
47. Korte, B., Vygen, J.: Spanning trees and arborescences. In: Combinatorial Optimization. pp. 131–155. Springer (2012)
48. Bock, F.: An algorithm to construct a minimum directed spanning tree in a directed network. Dev. Oper. Res. 29–44 (1971)
49. Jungnickel, D.: Spanning trees. In: Graphs, networks and algorithms. pp. 99–123. Springer, Berlin (2013)
50. Camerini, P., Fratta, L., Maffioli, F.: A note on finding optimum branchings. Networks **9**, 309–312 (1979). https://doi.org/10.1002/net.3230090403
51. Gabow, H., Galil, Z., Spencer, T., Tarjan, R.: Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica **6**, 109–122 (1986). https://doi.org/10.1007/BF02579168
52. Fredman, M., Tarjan, R.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM **34**, 596–615 (1987). https://doi.org/10.1145/28869.28874
53. Dwivedi, A., Yu, X., Sokolowski, P.: Analyzing power network vulnerability with maximum flow-based centrality approach. In: 2010 8th IEEE International Conference on Industrial Informatics. pp. 336–341. IEEE, Japan (2010)
54. Kiesling, S., Klünder, J., Fischer, D., Schneider, K., Fischbach, K.: Applying social network analysis and centrality measures to improve information flow analysis. In: Product-Focused Software Process Improvement. pp. 379–386. Springer International Publishing, Cham (2016)
55. Maccari, L., Nguyen, Q., Lo Cigno, R.: On the computation of centrality metrics for network security in mesh networks. In: 2016 IEEE Global Communications Conference (GLOBECOM). pp. 1–6. IEEE, USA (2016)
56. Zegura, E., Calvert, K., Donahoo, M.: A quantitative comparison of graph-based models for Internet topology. IEEE/ACM Trans. Netw. **5**, 770–783 (1997)
57. Bavelas, A.: Communication patterns in task-oriented groups. J. Acoust. Soc. Am. **22**, 725–730 (1950). https://doi.org/10.1121/1.1906679
58. Shao, B., Wang, H., Xiao, Y.: Managing and mining large graphs: systems and implementations. In: Proceedings of the 2012 International Conference on Management of Data—SIGMOD'12. p. 589. ACM Press, USA (2012)
59. Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., Wilkins, D.: A comparison of a graph database and a relational database: a data provenance perspective. In: Proceedings of the 48th Annual Southeast Regional Conference on—ACM SE'10. p. 1. ACM Press, USA (2010)
60. Allen, D., Hodler, A., Hunger, M., Knobloch, M., Lyon, W., Needham, M., Voigt, H.: Understanding trolls with efficient analytics of large graphs in Neo4j. BTW (2019). https://doi.org/10.18420/BTW2019-23
61. Geepalla, E., Asharif, S.: Analysis of Physical Access Control System for Understanding Users Behavior and Anomaly Detection

Using Neo4j. In: Proceedings of the 6th International Conference on Engineering and MIS 2020. pp. 1–6. ACM, Kazakhstan (2020)
62. Jouili, S., Vansteenberghe, V.: An empirical comparison of graph databases. In: 2013 International Conference on Social Computing. pp. 708–715. IEEE, USA (2013)
63. Ugurel, S., Krovetz, R., Giles, C.: What's the code? Automatic classification of source code archives. In: Proceedings of the eighth ACM SIGKDD International Conference on Knowledge discovery and Data Mining—KDD'02. p. 632. ACM Press, Canada (2002)
64. Kolomičenko, V., Svoboda, M., & Mlýnková, I. H.: Experimental comparison of graph databases. In: Proceedings of International Conference on Information Integration and Web-Based Applications & Services—IIWAS'13. pp. 115–124. (2013). https://doi.org/10.1145/2539150.2539155