Computers & Security 123 (2022) 102938

Contents lists available at ScienceDirect

Computers & Security

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

Attack Dynamics: An Automatic Attack Graph Generation Framework Based on System Topology, CAPEC, CWE, and CVE Databases

Ferda Özdemir Sönmez[a,*], Chris Hankin[a], Pasquale Malacaria[b]

[a] Institute for Security Science and Technology Imperial College, South Kensington Campus, London, SW7 2AZ, England, UK
[b] School of Electronic Engineering and Computer Science Queen Mary University of London, Mile End Road, London, E1 4NS, England, UK

ARTICLE INFO

Article history:
Received 31 May 2022
Revised 26 September 2022
Accepted 27 September 2022
Available online 1 October 2022

Keywords:
Computer Security
Risk Analysis
Network Security
Attack Graph
CAPEC
CWE
CVE
Visualization

ABSTRACT

Through a built-in security analysis feature based on metadata, this article provides a novel framework that starts with a scenario input and produces a collection of visualizations based on Common Attack Pattern Enumeration and Classification (CAPEC) and Common Weakness Enumeration (CWE) Standards. It immediately links enterprise mitigations from MITRE ATT&CK framework to the security flaws it discovered. It's also integrated with a third-party optimization tool targeted at cutting security costs for businesses, which it can perform in real-time or later using JSON output in the preferred format, depending on the execution mode. All of these stages are conducted without human intervention. Adaptive metadata with a variety of rules for capturing different sorts of known or prospective attack types allows for the production of attack graphs. It can be used as a quick and practical what-if analysis tool to detect potential intrusions for a variety of network configuration setups and assigned access privileges. As a threat modeler, it is suitable for both novice and expert users. Due to the easy input scheme and human-readable outputs, it can also be utilized as an educational tool.

1. Introduction

Automatic analysis of security threats is required for organisations because manual analysis is time-consuming and prone to errors and omissions. Automatic attack path analysis enables rapid analysis of single or multiple targets on small and large networks. Several attempts have been made in the past to develop an automatic attack graph generation tool. These tools' inputs, outputs, and features overlap to some extent with the Attack Dynamics tool. Nonetheless, this research topic still has gaps and usability isses. This paper presents a novel tool, Attack Dynamics, with the primary goal of automatic attach graph generation, in order to provide advancements in this area. The presented tool seeks to find attack paths for arbitrary network definitions based on system topology, associated vendor products, and known vulnerabilities. The creation of a topology as an input is a major challenge for automatic attack graph generation. Scanners like Nessus Rogers (2011) do not provide enough information to build a detailed topology graph. To improve the output, other data types, such as firewall rules data, must be used. The study does not cover the automatic generation of topology input. However, because the input structure is similar to that of existing attack graph generators, existing topology generator tools used by these attack graph generators, such as TVA Jajodia et al. (2005) can be used with modifications. Vulnerability scanner tools, such as those used in the MULVAL example, which generates input from the Nessus Rogers (2011) file, are another type of software that would be useful for creating input. Topology generator software dedicated solely to this target, such as Auvik Auvik (2021) can also be used to generate the topology input. Manually creating topology graphs based on knowledge of network configurations and infrastructure elements that run on the network is also possible. The topology structure is the least variable aspect of the scenario input. While creating the topology input manually, other volatile parts, such as vulnerabilities can be generated using appropriate scanner tools. The majority of attack graph generator tools only function as reachability analysis tools Kaynar and Sivrikaya (2015), with no additional analysis. In general, tools focusing on reachability, whether with or without the inclusion of vulnerabilities, do not provide enough information on how the attacker may attack or use those vulnerabilities to conduct an attack, leaving the user with a set of vulnerabilities assigned to different parts of the topology entered. Even an experienced security analyst would struggle to understand how these vulnerabilities may affect the system for multiple attack sources and multiple targets in the absence of automatic threat

* Corresponding author.
E-mail address: ferdaozdemir@gmail.com (F. Özdemir Sönmez).

https://doi.org/10.1016/j.cose.2022.102938

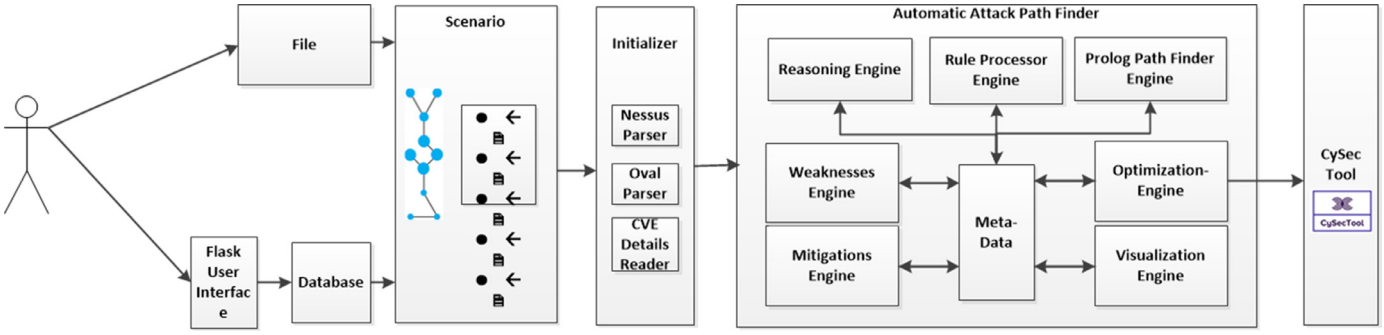


Fig. 1. Attack Dynamics, automatic attack graph generation framework overview

analysis. The proposed system formulates the attack as a function of reachability and attack definition, Eq. 1 (all equations are listed in Table A.2), where reachability is determined by system topology, assigned vulnerabilities, known vendor products, and the attackers, Eq. 2.

As part of reachability analysis, the attackers' positions in relation to target nodes are discovered using the effects of vulnerabilities assigned directly or via vendor products. At this point, the access rights at each connection point are also checked. Reachability analysis does not yield a boolean yes-no value, but rather a broader set of information such as the attackers' positions and distances, gained access rights over the nodes, and the nodes' most recent status in relation to these analysis results.

1.1. Summary of system parts and their responsibilities

CAPEC MITRE (2021a) database and three engines are used to find attack paths: a reasoning engine, a Prolog pathfinder, and a rule processor. While reasoning engine and Prolog engine deals with finding start and end nodes and paths connecting them, the rule processor is in charge of dealing with all of the unique requirements for detecting failure conditions. Low-level controls such as reachability, authentication controls, and network connectivity to/from the network and the Internet are examples of these. Failure of specific checks are among the pattern-based rules which also include determining the presence or absence of certain attributes or states caused by vulnerabilities at specific points along the path, such as the start, end, or middle nodes. Fig. 1 shows the high-level overview of the system components. After determining the attack path, the system detects the weaknesses associated with this attack and related mitigations from the metadata. Following the identification of vulnerabilities, the system generates an output in the form of a JSON file that can be used as an optimization input for the CySecTool Khouzani et al. (2019), which can select the best set of controls given budget constraints for direct and indirect costs. The optimization Engine generates this JSON input file.

Finally, a set of visualizations, namely a visualization showing CAPEC attack paths for all attack patterns, visualizations showing specific attack types for all attacks to enable detailed examination for each, and a visualization showing the weaknesses in the form of CWE MITRE (2021b) associations are created automatically by the tool. Each attack is enumerated by an ordered number at the end point of the attack in the first visualization type. As part of its reporting efforts, the tool also lists the details of each identified attack path in text form. During a system evaluation, users can use this list and visualisation to discuss potential attacks and the outcomes.

Aside from the visualization outputs, the tool has the capability of integrating with a security cost optimization software, CySecTool, either in real-time or in a delayed manner through the use of generated JSON files in the specified format for CySecTool.

1.2. Research questions and contributions

The authors sought answers to a set of questions that are identified gradually throughout the study. The first question was "which is the best source to generate automatic attack graphs with multiple attack types?". After an examination of various databases, the authors decided that the nature of the CAPEC database would best suit this purpose. Later, the research question "what information can be captured from CAPEC and be used in a reusable manner?". As an answer to this question the design of metadata structure along with the identified node types and attribute types is created. Finally, the authors wanted to know "how efficient and accurate would this new approach deliver attack graphs?". The case studying technique is used which is commonly used for the validation of these types of studies. Various scenarios are created and variations of them are examined through a series of meetings part of which are included in the paper along with numerical results.

The proposed study's major contribution over previous studies is its entirely novel approach to find attack paths which enable identifying attack types, in contrast to the majority of existing attack graph tools, that act as a reachability graph generator, and using the CAPEC database for the first time in its unique way (unlike existing tools which use it as external reference) to achieve this goal. The second contribution is the ability to enable scenario inputs close to real-world topological structures by having multiple node types and a large set of node properties captured through detailed analysis of attack descriptions. The third contribution is the advanced visualization system, which allows visual perception of attacks over a topology graph in a human-readable manner. Another contribution is the provision of CWE associations for identified attacks and the automatic assignment of mitigations via metadata. The ability to optimize the security costs for the selected attack graph is the final contribution.

The remainder of the paper is as follows. The related work section is found in section two. In section three the proposed system is explained including the details of the formation of the scenarios, visual representation of scenarios, and the basis of automatic attach path finding for the proposed system. Later, in the results section, a base scenario, modifications and their effects on the outputs are revealed. Following that, there are the discussion and conclusion sections.

2. Related work

Although attack graphs are known to be useful for various security planning and protection purposes, in the literature, there have been only a very limited number of studies which offer novel ways

for automatic creation of attack graphs Singha and Patgiri (2021). Use of attack graphs mostly depended on using manual ways. Philips and Swiller Phillips and Swiler (1998) is one of the earlier attempts to create semi-automatized attack graphs based on given configuration files consisting of node types, machine classes, network, other configuration, etc. and attack profiles showing prerequisites in the form of node-link connections. When the attack profile definitions have this type of restricted structure containing potential topology sub-parts, handling multi-phase attacks, or same attacks with different order of nodes would have issues. The definition of attack profiles should be independent of topology definitions.

Attack graph generators try to identify specific attack paths using attack profiles. Ou et al. Ou et al. (2006) uses logic programming to identify attacks in a scalable manner based on defined logical rules. The problem with this approach is having low level and technology dependent techniques such as trojan insertion or nfs shell as a part of logical rules. Having these low-level, manually defined, technology dependent attacker behaviours, or rules in the attack profiles results in inclusion of limited number of actions in the attack graphs. This also makes it difficult to define new rules in the system. Given the abundance of potential techniques and tools, trying to write rules accordingly creates serious feasibility and usability problems. The proposed system does not have such definitions. The attacker behaviors are based on dynamically calculated node properties depending on vulnerabilities and other identified attack types. So, instead of limited definitions depending on a single or a few techniques, or rules that include a limited set of vulnerabilities, the proposed system would take into account of other identified attack types. These include but not limited to Code Inclusion, Command Injection, and Code Injection which may be the result of user-defined or calculated (based on vulnerabilities and tools) node properties, consistent in level of details and more rigorous in handling more situations.

In order to be useful, automatically generated attack graphs must have certain properties Lallie et al. (2017). Depending on the standards is one of these properties that would aid in the communication of inputs and the results. Initially, network vulnerability analysis tools relied solely on the network structure and the location of hosts and servers within it Shahriari and Jalili (2007). This category also includes the input of known exploits in some cases. NetKuang Zerkle and Levitt (1996) and Ritchey and Ammann Ritchey and Ammann (2000) are a few examples of these initial studies. Later, the majority of attack graph generation studies, such as MULVAL Ou et al. (2005) used data from databases like Common Vulnerabilities and Exposures (CVE) MITRE (2016) and Common Weakness Enumeration (CWE) MITRE (2021b). These integrations took place at various stages of graph generation, such as taking this information as an input or output as part of the final graph, or both. CVE vulnerabilities were included in attack graph generation by using either an automated network/host vulnerability scanner or manually assigning known vulnerabilities to some edges/nodes.

The proposed system also depends on these popular databases. Besides the system also relies on Common Attack Pattern Enumeration and Classification (CAPEC) MITRE (2021a) definitions which allow a clear description of attack dynamics for various attacker types, whether insiders or intruders from outside the network, given a topology input. Unlike the previous approaches, the system does not include CVE ID's in the form of rule definitions. Writing rules based on CVE IDs is not feasable, considering the number of CVEs, similar to writing rules depending on attacker techniques. The proposed tool uses the potential effects of identified CVE values, thus, abstracting multiple CVE IDs and their consequences via various sets of attributes which is calculated dynamically by the system. Abstracting multiple vulnerabilities for a single node was a future work pointed out by Ammann et al. Ammann et al. (2002)

The use of vulnerability scanner tools in conjunction with attack graph generation tools is also provided by the proposed system. The proposed system captures CVE vulnerabilities from scanner outputs, similar to earlier attack graph generators such as Ou et al. Ou et al. (2005) or Ingols et al. Ingols et al. (2006). Unlike the earlier studies, the new system captures real-time details for these vulnerabilities online to have the low-level vulnerability properties along with other data. One of the main differences between the proposed tool and previous studies is that it does not only deal with vulnerabilities/exploits because some attacks may be associated with other node properties such as the system's purpose. Attack types may be caused by other node properties regardless of identified vulnerabilities or other identified attack types.

Topology input is frequently required for attack graph generation tools. Topology data can be generated automatically by agent-based systems such as Auvik Auvik (2021) Some of the attack graph generator tools, such as TVA Jajodia et al. (2005), include some semi-automated generation of the topology data as a part of the attack graph generation. This study excludes topology creation from its scope based on the assumption that relying solely on network scanner outputs will not produce adequate topology inputs. The approaches that rely on network scanners, such as the one used in TVA, are not mature enough to provide a topology structure that is close to reality, as Nessus scan results are based on a limited set of data. Other data sources, such as firewall rules data, are also required, as stated in TVA Jajodia et al. (2005), to provide more realistic outputs. The contributions of attack graph related studies can be grouped into multiple categories three of which are novel automatic path finding algorithms or approaches, novel visualization schemes and novel numerical calculations on the attack graphs. The studies in the second group, in general does not deal on the attack path finding but focus on the visualization part. The studies which are in the last group, in general do not reveal novel ways for attack path detection. The majority of the attack graph generation studies act as a reachability diagram builder regarding the first group, attack path finding Ou et al. (2005), Ingols et al. (2006). The studies rely on similar algorithms or data and in the end provide a set of connected nodes with no detailed information about the nature and type of attacks. The studies that are in the third group address the possibility of an attacker gaining access to specific nodes on the system, whether the attackers are part of the nodes or not. These are mostly used as access control checkers rather than attack graph generators. The studies rarely mention specific attack types with exceptions. Some of the studies that are in the third group include examinations of specific attack types, such as Vulnerability Take Grant (VTG) Shahriari and Jalili (2007). One of the approaches to find out the potential attacks for a topology is to use general purpose model-checking software. In this approach, the exploits are defined using a formal definition either in the form of equations Ibrahim et al. (2019)Somesh and Wing (2001) or specific language definitions such as in Templeton and Levitt (2001). Some of the studies in this category are more primitive in nature, requiring the development of topology-specific rules and exploit definitions Ritchey and Ammann (2000) Somesh and Wing (2001) which make them infeasible to be used for even smallest structures, others work on more generic definitions Ou et al. (2005). Various third-party model checker tools are used to to build attack graphs. NuSMV Cimatti et al. (2002) is an opensource model checker for symbolic model checking that is used by Sheyner et al. Sheyner et al. (2002) for building attack graphs. Ghazo et al. Al Ghazo et al. (2018) uses Jkind Gacek et al. (2018) model checker in their Automatic Attack Graph Generator and Visualizer (A2G2V) tool. A2G2V requires the input to have a specific format, in Architecture Analysis and Design Language (AADL) Feiler et al. (2006).

While, creation of input data is relatively easier compared to using AADL format, the main idea is similar. Without going into details, node types defined in the system correspond to the components and component implementation types. Important packages and properties are stored as node properties in the system while AADL has specific classes for those. In A2G2V, AGREE (Assume Guarantee Reasoning Environment) Păsăreanu et al. (1999) is used to generate the vulnerabilities for the system. AGREE depends on assumptions on components' environment. Other probabilistic approaches include Cui et al. Cui et al. (2019) which relies on Common Vulnerability Scoring System (CVSS) First (2022) values provided by the CVE databases and Bayesian theory. Wang et al. Wang et al. (2018) is another study which focuses on calculating risk levels using CVSS values. The contributions of these studies mostly are in the group of studies which does numerical calculations on the attack graphs.

The proposed system differs regarding the definition and input of vulnerabilities. First of all, it does not include assumptions or probabilities related to the CVE associations but it takes exact inputs. Second, the proposed system does not rely on third party formats or model checkers but gets vulnerability data in the form of CVE lists or vendor products or in relevant file formats such as Nessus file or Oval file. Storing data in a originally inhouse created structure and having attack graph generator modules optimized to handle that specific format yield to optimized processing time in the proposed system. Use of available CVSS values can be exploited together with Bayesian techniques along with distance of attack sources to targets in future versions of the proposed tool because the data used in the study is currently read online in real time or already is a part of the data structure but not used in the prototype implementation. As a part of this effort, in the proposed system users may select a threshold risk value which may yield eliminating paths below the selected level.

The attack graph generators handle attacks in a variety of ways. Some of the attack graph generator tools handle each and every attack in isolation. These primarily serve as vulnerability scanner tools for a single server or host. The majority of attack graph generators, such as Topological Vulnerability Analysis Tool (TVA) Jajodia et al. (2005), Mulval Ou et al. (2005) allow for the detection and presentation of attacks in combination. The proposed tool falls into the second category as well. Depending on the attack type and the type of targeted node, it necessitates the presence of other attack types in the same or related nodes (parent, child, etc.). Some attack graph generator tools, unlike the proposed tool, require predefined targets Ou et al. (2005), Wang et al. (2018). This necessitates additional preliminary work and may result in omission or ignorance due to reliance on human acumen. Similarly, some attack graph generators allow for the inclusion of multiple intruders, while others only allow for a single intruder Kaynar and Sivrikaya (2015). ADTool Kordy et al. (2013) is, without having predefined node types, an attack tree drawing tool that works on logical operators. Although, the tree allows multiple initial points, it allows the definition of a single target in the attack tree. The attack graph generators' methods for determining attack paths differ as well. Most attack graph generators display a single path, the most likely path (mostly based on Bayesian calculations), or the shortest path Swiler et al. (2001). ADTool Kordy et al. (2013) does not show the path(s), it is not an attack graph generator tool, but as a result of logical analysis, it provides Boolean success value and numeric probability value for the target along with other predefined domain analyses types. It also allows definition of other logical queries for the provided attack tree, but not having a relation to known databases of attack types or vulnerabilities, it does not make a real security analysis. It acts as a logical querying tool for attack trees. The proposed tool is designed to have all of the missing features of these tools, allowing for a multi-agent, multi-target, multi-attack type attack graph generator without the need to pre-define the targets or sources. It also displays all possible paths based on the topology connections, not just the shortest path. Hence the name "Attack Dynamics" was coined. Given a topology based on attack natures, the system targets to detect all possible attacker actions without overcomplicating the visualization outputs through reasoning. In terms of the architecture, the distributed attack graph generator Kaynar and Sivrikaya (2015) by Kaynar and Sivrikaya is an original attempt. This is based on the modified versions of the well-known network traversal algorithms such as depth-first algorithm. To deal with very large networks, the authors divide the original topology into multiple parts and use a distributed processing approach. Current implementation runs on a single server, can be replicated on multiple servers and be accessed via the web as long as the scenario is divided into multiple parts. The types of intruder actions are limited for the majority of the existing attack grap generators which depend on a limited set of CVE dependent rules. The design by Sheyner and Wing Sheyner and Wing (2003) is also in this category. This is in contrast to the large number of CVE definitions which is the source of the difficulty itself. The majority of the studies use vulnerabilities as themselves in various ways as static rules in their algorithms for reachability analysis efforts. However, since these are mostly conceptual studies with no use of real list of large set of CVE items such as Wang et al. Wang et al. (2018), they do not reveal any information regarding potential difficulties on using these vulnerabilities in a real-world environment and the capability of coverage for potential vulnerabilities using their designs. The proposed system also takes the vulnerabilities in the form of CVE identifier code(s) or as the name(s) of a known vendor product(s). Later, instead of using the codes directly in the rules it takes the attributes and depends on those attributes on rule definition and processing.

The majority of the earlier attack graph generator approaches show nodes as attack actions and edges as change of state, or nodes as states and edges as actions. In either way, since the states depend on a numerous number of attributes and vulnerabilities, combinations of them yield with a tremendous number of derived nodes/edges. Having large and very complex attack graph results creates new research topics such as aggregating attack graphs and decreasing their complexity Noel and Jajodia (2004). These large graphs show how vulnerabilities and other attributes are related to each step of the attack but they do not clearly show how these are related to topology items. MULVAL Ou et al. (2005) and TVA Jajodia et al. (2005) are in this category. They may generate a state diagram-like output that depicts the stages of the attack, Ritchey and Ammann (2000). The obligation of showing attribute, vulnerability combinations in the form of new nodes/edges generally arises due to using third party model checker and reasoning tools. The proposed system, on the other hand, provides attack graphs that show the attacker's step-by-step progress over the given topology with no additional nodes. Several attempts are made in the literature to enable machine readable attack graph results. Use of OWL (Web Ontology language) is one of them, Lee et al. (2019). The resulting attack graph in the proposed tool is stored both in graphviz dot format and also as a serializable, machine readable object format which allows automatic integration with other tools, such as the cost optimization tool which was conducted in the prototype implementation.

3. Proposed system

Although attack graph generation topic has been studied extensively, the current tools have problems mainly in terms of usability and accuracy. The usability problems caused by primitive visualization designs that do not match with the real world and that require recall or calculation instead of recognition (e.g. existence

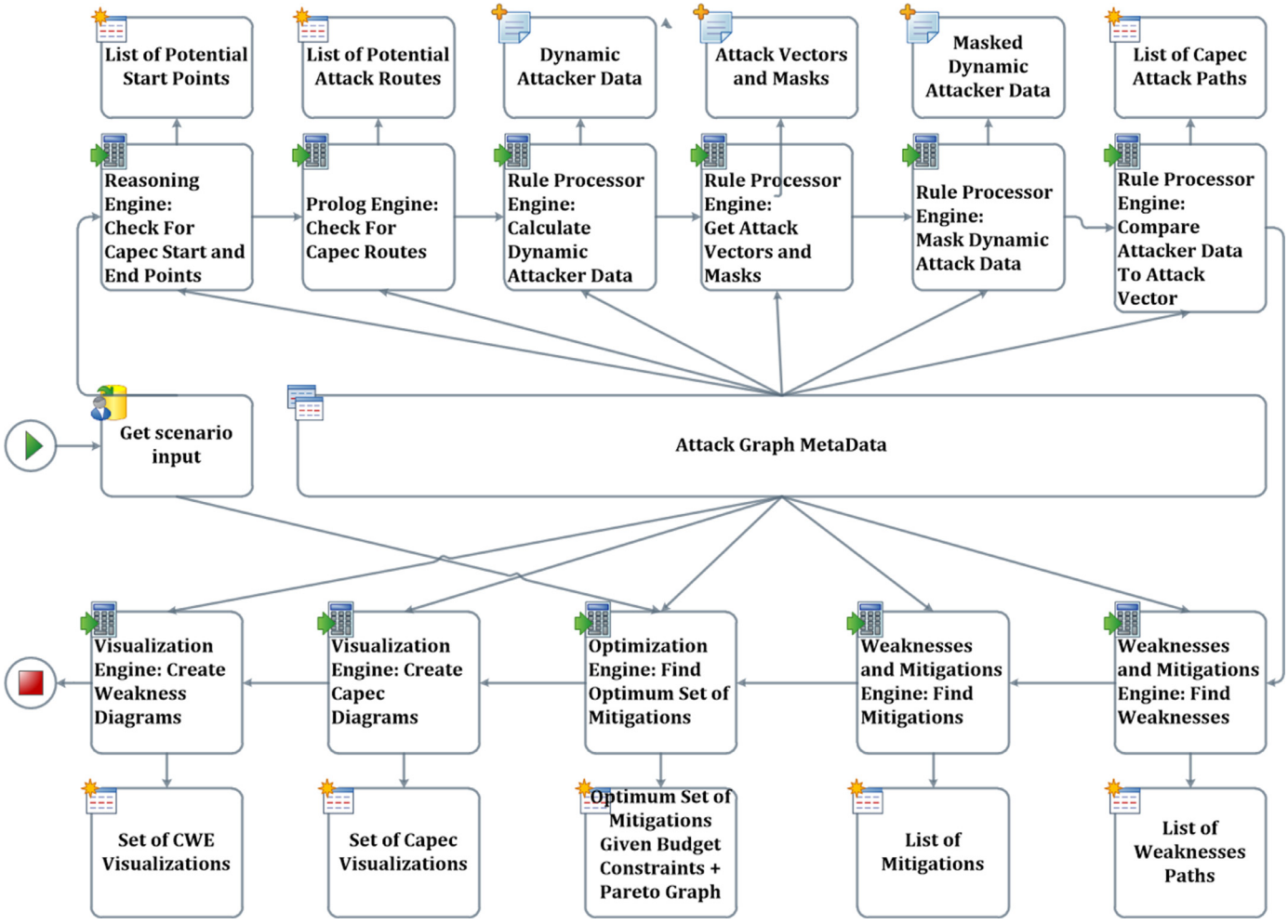


Fig. 2. Order of actions of by various system layer

of boolean or integer values in the attack graphs to be interpreted by the users). Other factors related to the usability is the flexibility and efficiency related. Accuracy problems on the other hand mainly due to two main reasons. The first reason is depending on third party metadata or third party rule engines and the second reason is primitive metadata mainly attaching threats/attack types to the assets/node types causing a high level of false positives.

The proposed system is an Automatic Attack Graph Generation Framework Tool called Attack Dynamics. This system aims to solve the mentioned usability and accuracy problems by having its own advanced metadata structure and having multiple optimized, integrated parts Reasoning Engine, Rule Processor Engine, a Prolog Path Finder Engine, Optimization Engine, and a Visualization Engine, Fig. 1, suiting to that metadata. These engines are employed in the order depicted in Fig. 2 from top to bottom. The proposed system creates an aesthetic and minimalist design unlike earlier tools which create unnecessary nodes or edges, and the system presents enough information through the use of special node types and other visual elements with a high level of status visibility relying the Nielsen's usability heuristics principals Nielsen (2005).

Metadata is the foundation of the proposed system and all of the engines depend on it. The Automatic Attack Finder Tool can be used to evaluate possible security risks towards an existing system(s) acting as a threat modeler tool. It may be used during the network topology design to provide safer topology structures through comparison. It can also be used as an educational tool in the field of cyber security. This system uses *AttackGraphMetaData*, Eq. 3, together with user input, a *scenario*, Eq. 4, in the form of a network structure and related data, details of both will be explained gradually in this paper.

The proposed system has one main target, *FindAttackPatterns* which consists of multiple sub-targets, operations. The user may have multiple scenario sets and run the tool against each one in turn. The following section describes the definition of a *Scenario* as well as the specifics of how to create one. A *Scenario* is a combination of multiple inputs. Nodes, Eq. 5, edges, Eq. 6, vulnerabilities, Eq. 4, vendor products Eq. 4 form a *Scenario*.

3.1. Scenario formation

This system's scenario is a collection of information that includes a system topology for the selected organization, system top-level details, detected vulnerabilities, built-in protection configuration, and known vendor products. A dynamic set of costs and effectiveness values for a predefined set of mitigations is also included in the scenario. The latter information is required as part of the scenario because it is also dependent on the organization chosen. For this last group, the user has the option of using the default values. There are no behavioral model definitions for the intruders in the scenario definition. Scenario parts have relations to both user inputs and metadata parts. The scenario is made up of several parts that can be entered sequentially. The first step is to con-

struct an arbitrary network structure using nodes and edges. The nodes should be chosen from a predefined set of 44 node types, and the edges should be directional (arcs). A system administrator who is knowledgeable on the topology for a specific organization, infrastructure elements, and the important software running on the client and server hosts may easily provide the necessary input. Several versions of the same scenario can be created easily and some assumptions such as knowing the purpose of a critical software, thus assigning "Critical Operations" to that node, knowing the content of the database, thus assigning "Critical Data" to that node, would be very straightforward and sufficient for the system. Detailed in-depth knowledge and corresponding user input such as port/service level accessibility from a tool output are not necessary. If the targeted organization is large, each department can fill out a non-technical form that has been specially prepared to make entries about their department in the scenario, for which purposes the devices used for the purpose are used, and to determine the points that must be specifically identified and included in the scenario. If the project has not yet been realized, system engineers and software project managers should get involved at the project stage to identify the end-points, then network engineers should get involved and add the network sections, the points that express the connection of these sections to the organization's main network.

3.1.1. Scenario formation principles

To produce neat, human-readable, and correct outputs, the correct set of nodes and edges, as well as the total number of nodes and edges in a scenario, must be chosen. As a result, the use of several principles is recommended during the scenario creation process. The first principle is to eliminate repetitions of similar nodes on the scenario based on structure, content, and properties similarity. The content is linked to the node's own properties, such as its purpose and connected nodes. Thus, two servers in the same network with the same type of server software are similar, but if one has a connection to a third node and the other does not, they are not. Similarly, even if they appear to be similar, if one has a node attribute and the other does not, they are not. The second principle is to include possible connections to primary access points. The system is in charge of allocating secondary access points for future interactions. In other words, if the internal user uses software type X which will be shown as a separate node (node X) in the scenario, associating the internal user with node X is sufficient. If node X has other connections and reads data from node Y, and there is a risk that some attacks will spread to that node, the system is responsible for locating those links. The third principle is to include only important or focused nodes, which means that if there is a machine in the network with no criticality or, more commonly, if there is software in a part of the system with no anticipated risks, such nodes should not be included in the graphs. This principle can also be used when gradually examining large topologies, so that different parts of the network or different end points can be examined in detail while other parts are eliminated. In scenario development, the focus of the risk analysis is critical. If a department is the focus point, for example, the unique configurations associated with that can be taken along with associated information related to the other departments that have close interactions with the focused department and the rest can be excluded. Similarly, if we are focusing on a specific type of software, such nodes may be added while other types of software are excluded. For example, if we are investigating the risks of an intranet application, there is no need to include the web server serving other users if they are not integrated. The focus points can also be restricted based on the type of human user. Because the scenario is based on the real-world configuration of the network, hardware, and software structures, while a security analyst can define the focus point of the risk analysis, a system administrator with system knowledge can provide the corresponding information. The following is a non-exhaustive list of sample questions that can be used during the scenario formation process to include/exclude some nodes and edges.

- Is it in the scope of the current risk analysis?
- Is there a similar host/server in the scenario with the same setup and position?
- What are the critical applications in the division/part of the organization, implying integrations with local or remote systems, critical operations or data, and etc.?
- What are the functions of the client and server machines, and who are the typical users?
- What are the top-level relationships, not between computing devices, but between departments? For example, accounting software serving the finance department, ERP software serving all staff from all departments.
- What are the current protection systems and where they stand?

In the end, the network inputted will include items such as hardware devices, security systems, server and client hosts, operating systems, and other software installed on those machines. The hardware types for clients and servers differ due to potential attacks for both types. Aside from the conventional hardware systems, the hardware may include portable memory devices, and surveillance systems. There is also a wide range of software types, such as web applications, authentication modules, desktop applications, and so on, owing to the connectivity properties of these various software architecture types. This information corresponds to an infrastructure set up for an organization. This step is called "Form the Network Structure". This step allows differentiating different parts of a network, such as a DMZ section or protected sections through the use of various node types. Besides the security protection systems that can be incorporated into the system, the user of the tool may also mark physical protection for specific nodes or node groups. The segmentation of the network can also be indicated through the use of specific nodes again. This way, not only the actions of the outsiders can be evaluated, but the actions of malicious Insider users accessing different sections of the network and the consequences can be observed in this system. The user must enter node attributes in order for the system to generate comprehensive results covering more attack types. These node attributes are inputted in the second phase of the scenario formation task in the form of assigning specific attributes to the nodes. "is design development machine", "uses external libraries", and "has critical operations" are a few examples. These are intended to highlight properties that may be associated with the attack types later on. The set of attributes is formed after a thorough examination of the mechanisms of security attacks and their prerequisites. "Set the Node Attributes" is the name of this step. The user does not need to enter application/vendor specific information at this level. The Automatic Attack Finder Engine is expected to find some attack routes to some nodes based on the node attributes and network structure.

If the user has more in-depth knowledge of the nodes or wishes to test the system structure for more low-level attack types (meta attacks that can be associated with assigned vendor vulnerabilities), he or she may assign a set of vendor products, software, hardware, and networking devices to the nodes. "Windows 10" can, for example, be assigned to a "OS" (Operating System) node, or "VMWare Vsphere" to a server node. These vendor product assignments can be made by assigning the product name or the vendor name plus the product name. Automatic Attack Finder would be able to use vendor product-specific low-level vulnerability information as a result of these assignments. This is the "Assign

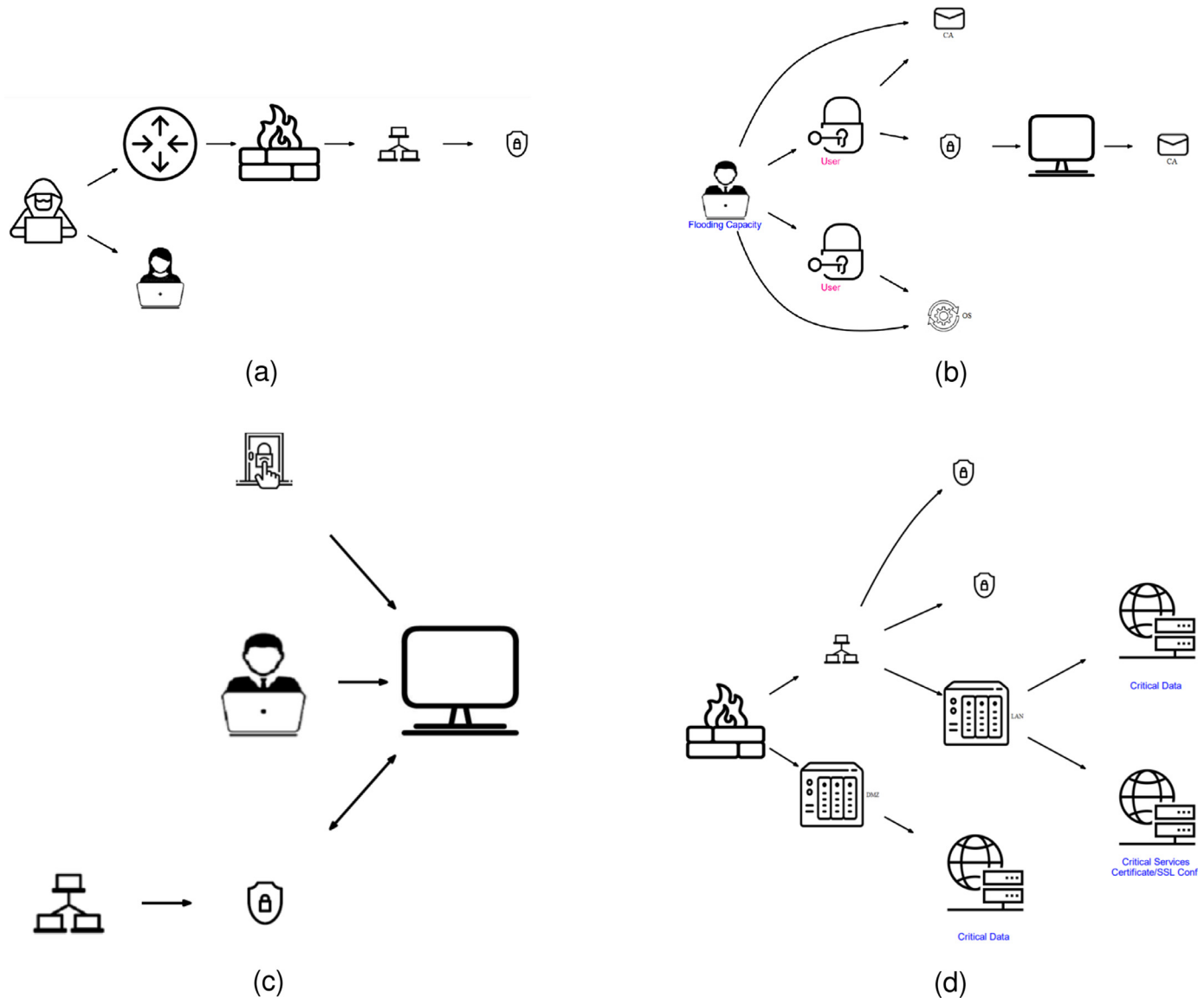


Fig. 3. Demonstrative samples of scenario visual representations

Vendor Products" step. These assignments may be made only to certain critical nodes, such as a web server that receives Internet requests, rather than to all node types. If the system's user requires more specificity, the nodes can be assigned some known vulnerabilities. For these assignments, the known CVE-IDs, such as "CVE-2017-12062", should be used. The outputs of the Automatic Vulnerability Scanner tools, in the form of a Nessus results file or an Oval file, can then be integrated into the system inputs in this manner. This is the "Assign Vulnerabilities" step.

3.2. Visual representation of scenarios

In this part, various visual aspects of scenario representations will be depicted through figures. These figures are part of a larger scenario. Fig. 3 (a) shows a typical structure having an enterprise router and a firewall that allows the transfer of data inwards or outwards for the enterprise local area network (LAN) in a controlled manner. In this figure, there are two actor icons, the one on the left represents an outsider, Hacker (as called in the system for simplicity), and the one on the right represents an Insider user. Finally, this figure has an icon marking a part of the network which is separated from the LAN, a sub-network either virtual or physical. Having this icon allows preventing access from some parts of the topology from others and is very useful for marking, for example, different departments or services in a larger enterprise that have controlled network access from other parts. Fig. 3 (b) depicts an Insider user interacting with a client operating system (OS) and a mail client application. There are multiple paths for both authenticated and authorised access, as well as non-authenticated access. Displaying both ways allows you to distinguish attacks based on the behaviours of an authorised and unauthorised user. Fig. 3 (c) shows one of the security systems, "Mechanical Protection", which can be attached to specific hosts to indicate physical access difficulty (physical protection) for these devices. Because the graph does not provide physical locations/separations for the nodes, rather network and logical positions are shown in the topology, this type of marking is required. Other available node types that contribute to system security include "Firewall", "Anti-Virus Spam Software", and "Intrusion Detection Software". Fig. 3 (d) depicts two distinct parts of a network, the DMZ section and the LAN section, as well as the server machines that reside in those areas. Those servers house three web server applications in total. Some of

the nodes in this figure series have textual descriptions like "Critical Data" and "Certificate/SSL Conf". This text is intended to assist the user in understanding why attacks/weaknesses occur for some nodes/edges but not others.

3.3. Used databases

The system encapsulates Common Attack Pattern Enumeration and Classification (CAPEC) MITRE (2021a) attack definitions in the form of metadata. This metadata is used to find paths from potential starting points to end points. Later, it associates the paths with the Common Weaknesses Enumeration (CWE) MITRE (2021b) weaknesses, and mitigations are attached to those weaknesses. Finally, the tool provides a set of visualizations showing how the attack paths and weaknesses are related to the given network. In this system, an attack is defined as a path that begins at one node and ends at another node that is associated with a CAPEC Id. This CAPEC was chosen because it focuses not only on specific vulnerabilities or weaknesses, but also on the attack pattern itself. CAPEC provides an easily discernible category of attack definitions for known attacks. The definitions of CAPEC attacks identify potential attacker types, attack mechanisms, potential targets, prerequisites. Meta Attack Patterns, Standard Attack Patterns, and Detailed Attack Patterns are all available in CAPEC. Techniques or methodologies are the focus of the Standard Attack Patterns category. The Meta Attack Patterns category contains abstract definitions that do not include specific methodologies or techniques. It aims to provide a high-level understanding of the nature of the attack. The Detailed Attack Patterns category contains low-level details that make use of specific low-level techniques, tools, and strategies. It also contains low-level vulnerability associations for specific attack types. All of the CAPEC attack definitions have parent-child relationships with one another, forming a hierarchical tree structure of attack patterns. The attack definition tends to be higher in the attack patterns tree as the level of abstraction increases, while the attack pattern is most likely in one of the children (leaf) nodes as the level of details increases.

Since the number of potential attacks is very high, a scope had to be defined for the study. The attack pattern definitions in the CAPEC database are classified as stable, draft, and usable. The tool aims to cover all of the known stable Meta Attack Patterns (as of August 2021 there are 31 stable Meta Attack Patterns) which can be expressed in the form of metadata when it is completed. This metadata includes information related to attacks nature, attacker types, strategies, techniques, tool requirements, etc. which was derived from a thorough examination of CAPEC patterns. A few well-known Standard Stable Attack Patterns (there are 33 stable Standard Attack Patterns as of August 2021) are also encapsulated in the system.

The Detailed Attack Patterns are not available for four reasons. The first and most important reason is that these types of attacks are product and/or vulnerability dependent, and thus require corresponding products and/or vulnerabilities. Having these low-level definitions in the metadata has several drawbacks. The first drawback is the tremendously increasing size of the metadata which would negatively impact the tool's efficiency. The presence of product or vulnerability level definitions in the metadata, on the other hand, makes the tool more prone to errors. Because the nature of the products changes on a regular basis, the metadata used would be extremely difficult to maintain. The third and fourth drawbacks are not related to the metadata's nature and content. Regardless of the level of details stored in the metadata, including the Detailed Attack Patterns would cause a momentous increase in the number of attack patterns investigated in the system which would decrease the overall performance of the tool significantly. Furthermore, having a huge number of attack definitions will end up in the automatic creation of attack graph visualizations that are incomprehensible due to too many visual items. In an abstract manner, the Meta Attack Pattern definitions already cover the Detailed Attack Pattern definitions. When the system detects a Metadata Attack, the user can always check the CAPEC Database to see the low-level attack types using the attacks hierarchy and focus on the ones that are relevant to the system topology.

To identify the flaws, the Common Weakness Enumeration (CWE) Standard is used. Because these CWE definitions are pattern and topology dependent, the same set of attack patterns will result in a different set of weaknesses as the topology changes. In comparison to the Capec Meta-Data set, the CWE set contains a large number of elements. The top-25 CWE list is used as a starting point, with other common weaknesses added. The number of flaws visible in the graphs will increase as the number of implementations increases. Similarly, for identified weaknesses, the mitigations engine recommends predefined mitigations from MITRE ATT&CK's Strom et al. (2018) enterprise mitigation list.

3.4. Automatic attack path finding

3.4.1. Reasoning, rule processor, and prolog engine layers

One of the critical modules, the reasoning layer commences the system's execution, Fig. 2. Given an arbitrary network structure and generic reasoning metadata, Reasoning Engine finds potential starting and ending points for various attack types. Reasoning, Rule Processor, and Prolog Engines use the *NodesMetaData*, Eq. 7, and *AttackTypesMetaData*, Eq. 8 which has sub-parts that will be explained gradually. The goal of this layer is to incorporate known cyber security knowledge into the system in order to answer the questions including "How does an attack happen?" "Is it coming from the outside or the inside?, and "Is it aimed at what nodes, local, adjacent, or remote?", and "What are the potential target points, and what node types are there?". In a system, some nodes appear to have a higher criticality in terms of potential attack, but this does not imply that they will have any or all attack types. Because each attack type has its own set of mechanisms and objectives, this layer aims to capture potential start and end points for each attack type.

GenericPathsMetaData Eq. 10 is the first part of the metadata used by Reasoning Engine. *GenericPathsMetaData* basically consisted of an hierarchical set of definitions including simple definitions containing one or more pairs combinations of them.

- $hacker\_to\_authentication\_authorization = [(hacker\_nt, aa\_nt)]$
- $insider\_to\_lan\_machines = [(insider\_nt, computing\_device\_server\_host\_in\_lan\_nt), (insider\_nt, computing\_device\_client\_host\_in\_lan\_nt)]$
- $to\_web\_server = hacker\_to\_web\_server \cup insider\_to\_web\_server$

depending on node types and node type groups. These are human readable definitions, which maybe provided to the end users for editing in the future versions. This may result in their involvement to update the metadata definitions for their specific organizations. During the creation of these definitions, CAPEC attack mechanisms are examined in depth and 44 node types are defined. Later, these mechanisms are mapped to CAPEC attack types forming *AttacksGenericPathsMetaData*, Eq. 9. Here, the nature of the attack types are kept in mind, so that the resulting diagram would be both sound and also exhaustive. Thus, the largest set of definitions which would yield sound results are assigned for each CAPEC attack type. These definitions do not act as rules including pre-post conditions for attacker actions but just draw a set of potential high level directions for each attack type. Some sample assignments are shown below.

- For CAPEC 416-Manipulate Human Behaviour: $genericPathsMetaData = hacker\_to\_insider$

- For CAPEC 594-Traffic Injection: $genericPathsMetaData = hacker\_to\_server\_machines \cup insider\_to\_server\_machines$.

Assumptions depending on the CAPEC descriptions are made at this point. Here, the first assumption is the manipulation of human behaviour is towards insider nodes. This manipulation may yield further attack types but this is not included in the definition of Manipulate Human Behaviour. The reachability analysis and the effects of one attack to another are not handled through these definitions but in the Rule Processor Engine part of the system. The second reasonable assumption is that traffic injection is performed on server machines but not on client nodes. These assumptions cause both sound but also succinct results eliminating some illogical paths but not all of them. Reasoning engine uses these generic definitions to find potential starting and ending points on the given topology.

Later, the Prolog Engine is used to find the paths connecting the starting and ending points. This yields a set of potential paths, each of which is associated with a different attack type described in the CAPEC database. Following the discovery of potential paths, the Rule Processor Engine is launched, which is based on the graph structure, node attributes, assigned vendor products, and vulnerabilities. This layer serves as a secondary reasoning layer that delves deeper.

GenericRulesPackageMetaData Eq. 11 consists of generic rules in the form of a combination of *genericNodeStatus* (Eq. 12), *genericAttackerStatus* (Eq. 13), and *genericAttackerPosition* (Eq. 14) where *AttackerNodeAttributes* is a subset of node attributes, *NodeAttributes* related to attacker capabilities, *AttackerNodeType* is a vector having zeros and ones for different attacker types, *GenericAttackerPosition* is a vector pointing out various attacker distances as local, remote, or adjacent, finally *GenericPathControl* is a vector containing attributes related to the path such as existence of protection systems. *GenericRulesPackageMetaData* is associated to CAPEC attack types in the system to form the *AttacksGenericRulesPackageMetaData* Eq. 15.

AttacksGenericRulesPackageMetaData holds may look complicated. It consists of human readable rules in the form of fixed length vectors containing a combination of scalar items and finite length lists in the form of 1 and 0's (yes/no's) having attack type a, attacker type b, target node type c, the lists of middle nodes to be existed or not to be existed, d and e, minimum effective attacker positions (local, remote, or adjacent), f, not allowed attacker positions (due to unlogicalness), g, the attributes that attacker should/should not have, h and i, and the attributes that target node should/should not have, j, and k. This may appear to be a lengthy list of items to work on for each attack type, but only a few of the attributes, middle node types, and so on will have non-default values for each CAPEC type. This is the translation of CAPEC database in a form of vectors. For each CAPEC meta attack type, several concatenated vector sets are stored in the system, each is a synopsis of an attack type independent of the, attackers, potential targets, given topology and vulnerabilities. Each synopsis has a mask vector so that only the related attributes are masked on the dynamically created data for comparison.

The final list of initial attacks is generated following the execution of the Rule Processor Engine, as shown in Fig. 2. This list does not include the paths that were eliminated because the rules did not apply. The Rule Processor Engine first checks the reachability based on not static but dynamic gained accesses and it validates specific attack-type rules to verify potential existence of an attack. These rules process the node attributes for the starting node, end node, and nodes in between, the network's topology and the positions of the starting and end nodes in relation to that topology, thus structure, detected vulnerabilities, and detected other attack types.

Rule Processor Engine simply checks *NodeStatus*, *AttackerStatus*, and *AttackerPosition* against generic *AttacksGeneric RulesPackageMetaData* where *NodeStatus*, *AttackerStatus*, and *AttackerPosition* are calculated dynamically ending up the *AttackData*, Eq. 19.

3.4.2. Weaknesses, mitigations, and optimizations layers

Once the attack paths have been identified, the system invokes the weaknesses engine. The Weaknesses Engine makes use of metadata to link each attack type and node type to a specific set of weaknesses. This association also contains information about where the weakness occurs, such as before the node, after the node, or on a specific type during a path, such as between a firewall and a router, or between authentication and access control and the operating system. The new information collected by this layer is attached to the previous list of potential paths, Fig. 2. The system is primarily comprised of the CWE top 25 weakness set as well as some other significant networking and hardware flaws discovered during the study. Addition of other CWE types is possible as long as corresponding metadata is included to the system. For illustrative purposes, a small set of CWE was selected but the software is easily scalable to handle a larger set of CWE items.

The metadata related to CWE weaknesses has two parts. In the first part, we have a dictionary of CWE topology associations in four groups: "Pre Target Node", "Post Start Node", "Post End Node", and "Specific Edge Type" which mark potential associations of CWE to topology parts. For example, "CWE-345: Insufficient Verification of Data Authenticity" can be associated to nodes that store data such as Database, Web Server, File Server independent of attack types as Pre Target Node -CWE association. If the weakness can be associated to topology parts, for example, pre/post a specific node type such as Operating System, Access Control, or Firewall nodes, or specific edges (edges directly connecting specific node types) then corresponding CWE definition would exist in the dictionary. The first group of information is mainly gathered from CWE definitions and known associations of them to some specific node types such as firewall, operating system, etc. In the second part, we store known associations of CAPEC attack types to CWE. The second group of information is gathered from available CAPEC-CWE associations from the MITRE ATT&CK web site. So, for example we have associated "CAPEC-148 Content Spoofing" to "CWE - 345" independent of the topology. Our algorithms use topology and identified CAPEC paths together with the CWE metadata to find the final correct list of CWE's. All parts of metadata including the CWE parts are entirely generic and do not hold any topology specific information.

Once a CAPEC attack is identified by the Rule Processor Engine, *WeaknessesMetaData*, Eq. 20, is used to find out relevant CWEWeaknesses set using both topology information and CAPEC path, Eq. 21.

MitigationsMetaData Eq. 22, consists of two parts. The static part uses existing information from MITRE web site which associates weaknesses to mitigations. The second part holds the dynamic information related to mitigations. Given a *Scenario* and *AttackGraphMetaData*, dynamic part of the *MitigationsMetaData* is updated, Eq. 23, so that organization specific mitigation constraints are handled by the system.

The mitigations engine's working mechanism is more straightforward. The metadata stores the set of mitigations associated with each identified system weakness. MITRE ATT&CK's Enterprise Mitigations set is used to create the set of mitigations. The optimization engine employs the system-stored mitigations' predefined costs and effectiveness. The system's user interface allows the user to update the system's default information to suit the specific organization, Eq. 23. The optimization engine generates a JSON file containing all of the detected weaknesses and mitigations, as well

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
16551	CVE-2017-12071	918			2017-09-08	2019-10-09	4.0	None	Remote	Low	???	Partial	None	None
Server-side request forgery (SSRF) vulnerability in file_upload.php in Synology Photo Station before 6.7.4-3433 and 6.3-2968 allows remote authenticated users to download arbitrary local files via the url parameter.														
16552	CVE-2017-12068	79		XSS	2017-08-01	2017-08-10	4.3	None	Remote	Medium	Not required	None	Partial	None
The Event List plugin 0.7.9 for WordPress has XSS in the slug array parameter to wp-admin/admin.php in an el_admin_categories delete_bulk action.														
16553	CVE-2017-12062	79		Exec Code XSS	2017-08-01	2017-08-15	4.3	None	Remote	Medium	Not required	None	Partial	None
An XSS issue was discovered in manage_user_page.php in MantisBT 2.x before 2.5.2. The 'filter' field is not sanitized before being rendered in the Manage User page, allowing remote attackers to execute arbitrary JavaScript code if CSP is disabled.														
16554	CVE-2017-12061	79		XSS	2017-08-01	2017-08-15	4.3	None	Remote	Medium	Not required	None	Partial	None
An XSS issue was discovered in admin/install.php in MantisBT before 1.3.12 and 2.x before 2.5.2. Some variables under user control in the MantisBT installation script are not properly sanitized before being output, allowing remote attackers to inject arbitrary JavaScript code, as demonstrated by the $f_database, $f_db_username, and $f_admin_username variables. This is mitigated by the fact that the admin/ folder should be deleted after installation, and also prevented by CSP.														

Fig. 4. Vulnerability attributes

as other optimization constants such as allowed budgets for indirect and direct costs. This information can then be used to generate Python case study code, which can be run in real time to determine the optimal set of controls for the chosen arbitrary network given budget constraints, Fig. 2. Optimization can be performed at a later time by using a JSON file as an input to the third-party CySecTool, which is in charge of the optimization task.

3.4.3. Visualization layer

The visualization engine relies on Graphviz libraries and predefined icon set for each node type. These icons help to identify infrastructure elements easily. The majority of the icons consist of only figures, only a few icons that look similar to each other are differentiated using text in the icon such as electronic mail server application and mail client application. There are four different types of visual elements on the graphs. The nodes, marked with specific icons, the edges, marked with colors, text on the nodes, and the text on the edges. The text on the nodes makes a short summary of node attributes, vulnerabilities, vulnerability attributes, and access levels. The edge text has CAPEC attack names, and a sequentially increasing attack number only on the final edge before the target for attack graphs, on weaknesses graphs this text has the CWE weakness category Id and name. Fig. 2 depicts three different types of graphs generated automatically by the system: a graph displaying CAPEC attack types, a graph displaying CWE, and individual graphs for each CAPEC for detailed examination. Samples of visualizations are available in the Results section and the project's Github site[1].

The authors believe that it is necessary to explain what is not displayed at this point. Rather than explicitly marking these steps on the graphs with additional text, nodes, or edges, common potential actions such as local access, network access, bypassing a firewall, and bypassing an authentication system are represented by specific node types. If these types of actions related to each edge for each attack type had been explicitly shown, the graphs would have become overly complicated. CAPEC attack names are used to depict other common attack-related activities such as command injection, file inclusion, code inclusion, and command execution. "Command Execution" on a Database node, for example, refers to the execution of a SQL command, whereas "Command Execution" on an OS node refers to the execution of an OS command. Malware file injection is depicted as the attack type "File Inclusion", whereas Malware Code Injection is depicted as the attack type "Code Inclusion". As a result, in their current form, the graphs speak for themselves by providing the attack types, attacker positions, node types, node properties, vulnerabilities detected, vulnerability properties causative of actions, and access rights without the need for additional decision nodes to depict these steps in various ways. The proposed visualization system creates outputs which are superior to existing automatic attack graph generators due to having its own reasoning and rule processor engines and not having the extra nodes created by third party model checkers.

3.4.4. Node attributes and their effects on the system

The nodes, as mentioned in the Scenario Formation Section, may have one or more attributes. The types of node attributes are determined by examining CAPEC attack definitions. The goal is to find properties for all node types, including actors, that produce specific results in attack path searches, either positive or negative. So far, thirty node attribute types, excluding vulnerability attributes, have been identified in the system. While the majority of these attributes are expected to be assigned statically, some are calculated dynamically by the system during engine initialization or execution. The types of attributes that nodes can have are not same; that is, the attributes that are meaningful for each node type differ. "Has flooding capacity", for example, has meaning for an actor. or "Has critical operations" is applicable to software. If the user assigns an attribute which is meaningless for a node type does not cause an error.

3.4.5. Vulnerability types and their effects on the system

The rule processor is a critical component of the system. Due to the large number of vendor products (5898 as of the third quarter of 2021 in the CVE database) and vulnerabilities (157758 as of the same quarter), the rules rely not on the products or vulnerabilities directly but also on the vulnerability attributes as one of its inputs. Fig. 4 depicts the vulnerability attributes available from the CVE Database. Each vulnerability can have one or more vulnerability attributes assigned to it. In this figure, for example, the second vulnerability is classified as XSS, while the third is classified as both XSS and Exec Code. DOS, Code Execution, Overflow, Memory Corruption, SQL Injection, XSS, Directory Traversal, HTTP Response Splitting, Bypass Something, Gain Information, Gain Privileges, CSRF, and File Inclusion are all included in the list. Other information besides vulnerability attributes includes access level, complexity, authentication, confidentiality, integrity, and availability. The type of access can be Local, Remote, or Adjacent. The pointed access type determines how the vulnerability property becomes advantageous to the attacker during an attack. As a result, the system stores and processes three sets of these vulnerability attributes.

3.4.6. Assigned access levels, access controls, A/A node, and their effects to the system

Despite the fact that authentication and access rights configuration are two separate activities, a single node type is created for both. Because, in most cases, access rights are granted once the user has successfully completed the authentication process. To

[1] Github page is https://github.com/ferdasonmez/AttackDynamics.

evaluate the effects of using them in various parts of the topology, the Authentication and Access Control (A/A control) node type can be added to the scenarios. As part of a scenario, access levels can be assigned to A/A control nodes, actor nodes, and other software, hardware, and networking nodes. The default access level for A/A controls is User (the other option is Admin), the default for actor nodes is None (the other options are User and Admin), and the default for infrastructure elements is None (the other alternatives are User, Admin, and Other). As a result of the vulnerabilities applied to the nodes, access levels for the nodes may be gained. In that case, the nodes' accessibility changes. Some attack types necessitate the attacker being an authenticated user with the necessary rights. As a result, the system can evaluate the possibility of passing the A/A controls based on the A/A control access level and user access level. Being able to assign independent A/A nodes with different access levels such as user and root simultaneously for a node would also enable observation with comparison. The presence of vulnerabilities may result in lowered access levels as gained access for some nodes. For example, a target node may be assigned a "User" access level, but the presence of a vulnerability may change its level to "None", or the node may be assigned a "Admin" access level, but the presence of a vulnerability may change its accessibility level to "User" or "Other". Because this access control validation occurs after path finding, future controls such as open ports and services can be added without modifying the program structure. Some attributes, such as "has critical remote services", are currently assigned manually, and are used by Rule Processor. These attributes can be assigned automatically again in the future by using a scanner like Nessus that has service detection plugins for services like FTP, Telnet, RemoteExec, RemoteLogin, and so on.

3.4.7. Attacker types, positions and their effects to the system

In general, the system may include regions such as the Internet, a local area network, a demilitarized zone of the firewall, an outside section of the firewall, and virtual or physical subnetworks. The system allows us to designate a portion of the network as a trusted zone, such as the CEO computer, so that we know this portion of the network cannot launch an attack. The adjacency concept is used to separate the network via the subnetwork icon. When we use this icon multiple times to form a closed structure that no actor can reach, we call it a protected region. In general, attacks do not end in protected areas unless a specific vulnerability causes them to do so. Based on the actor's position and the end point, the distance can be local, adjacent, or remote. This concept of distance differs from the roles associated with node types, Insider, Hacker, and Stakeholder. Despite being on the outside, the Internet, a hacker, malicious person, or group may be local to an unprotected DMZ node with direct access from the Internet and no network or firewall protection. An Insider who stays in a completely different part of a network, on the other hand, may act as an Adjacent or Remote user. This information is used by both the Reasoning layer when determining potential starting and ending points and also by Rule Processor layer when checking the existence of specific attacks.

The vulnerabilities are valid or usable for the attackers from a distance (local, remote, adjacent), so the rule processor takes the attacker's position with respect to the efficacy of the vulnerabilities for that distance. An Insider user cannot attack a protected area of the network in the majority of attacks unless a vulnerability exists that allows access from adjacent networks. In this context, protected refers to a section of the network that the Insider cannot access. On the contrary, for some attack types, such as those classified as brute force attacks, being in a remote location does not preclude attackers from conducting an attack against a target. As a result, the networking positions of the actors and the allowances that emerge as a result of vulnerabilities are realized differently in different attack types.

3.5. Results

Several scenarios and how they differ from one another will be explained in this section, along with corresponding differences in the resulting diagrams. What-If Analysis is the process of altering the input values to see how they affect the outcome. The proposed system can be used to perform what-if analysis. The sample set of scenarios shown in this section is intended to demonstrate the effects of inputs on output in order to establish the tool's what-if analysis usage along with demonstration of tool's features. To comprehend the proposed system, it is necessary to depict why some attacks are chosen over others. Due to space constraints, the paper only includes a few small scenarios (a small base scenario and its variations) to demonstrate how the system works. Alternative scenarios with different node types and attack types is available on the project's Github site[2]. The readers should know that these graphics are created by the tool which is still in its prototype stage without the completion of implementation for all attack types and lacking detailed production tests.

3.5.1. Base scenario

The first scenario, also known as the base scenario, is similar to the scenario depicted in the second figure of the MULVAL paper Ou et al. (2005), which describes one of the most well-known attack graphing tools. In this scenario, there are three zones: the Internet, the Demilitarized Zone (DMZ), and the Internal Zone. In this model, instead of two firewalls (assume one firewall with three network interfaces), one is used to create a DMZ section. Having two firewalls has no effect as long as the elements remain in the same DMZ section. There is an intruder, or potential Hacker, using a machine to connect to this network via the Internet. There is a web server in the DMZ section. There is a workstation with a critical file in the intranet section. There is also an internal server running a file server application.

Fig. 5 (a) depicts the potential attacks chosen for this setup. It contains nodes, edges, and bottom textual notes for nodes highlighting node attributes/or detected vulnerability types. Clearly, the server machine in the DMZ is vulnerable to an Active OS Fingerprinting attack. Because the Insider user has direct access to the system, there is also the possibility of Content Spoofing to the File Server content. Design Alteration and Development Alteration are not selected among the already implemented attack types because there is no workstation designated as a design/development machine. There is no flooding because the user lacks flooding capacity, that is, the setup and resources required to carry out a flooding attack, and there is no machine marked with a Denial of Service (DOS) vulnerability type, one of the vulnerability attributes associated with CVE vulnerabilities. Command Injection can be done towards several node types such as Operating System nodes, Database nodes or it may be achieved through the use of a specific device Portable Memory Device. The presence of these nodes does not guarantee the attack, several other conditions must be met. For example, in order to perform a Command Injection in an OS, the system searches for an authenticated path to that node.

Code Injection is even more restricted. Among other things, it necessitates the absence of an Intrusion Detection and Prevention System (IDS) through the detected path, as well as the absence of an Anti-Virus/Anti-Spam node connected to the end node. In addition to other structural controls, the Code Inclusion attack rule looks for the presence of a File Inclusion vulnerability on the end

[2] Github page is https://github.com/ferdasonmez/AttackDynamics.

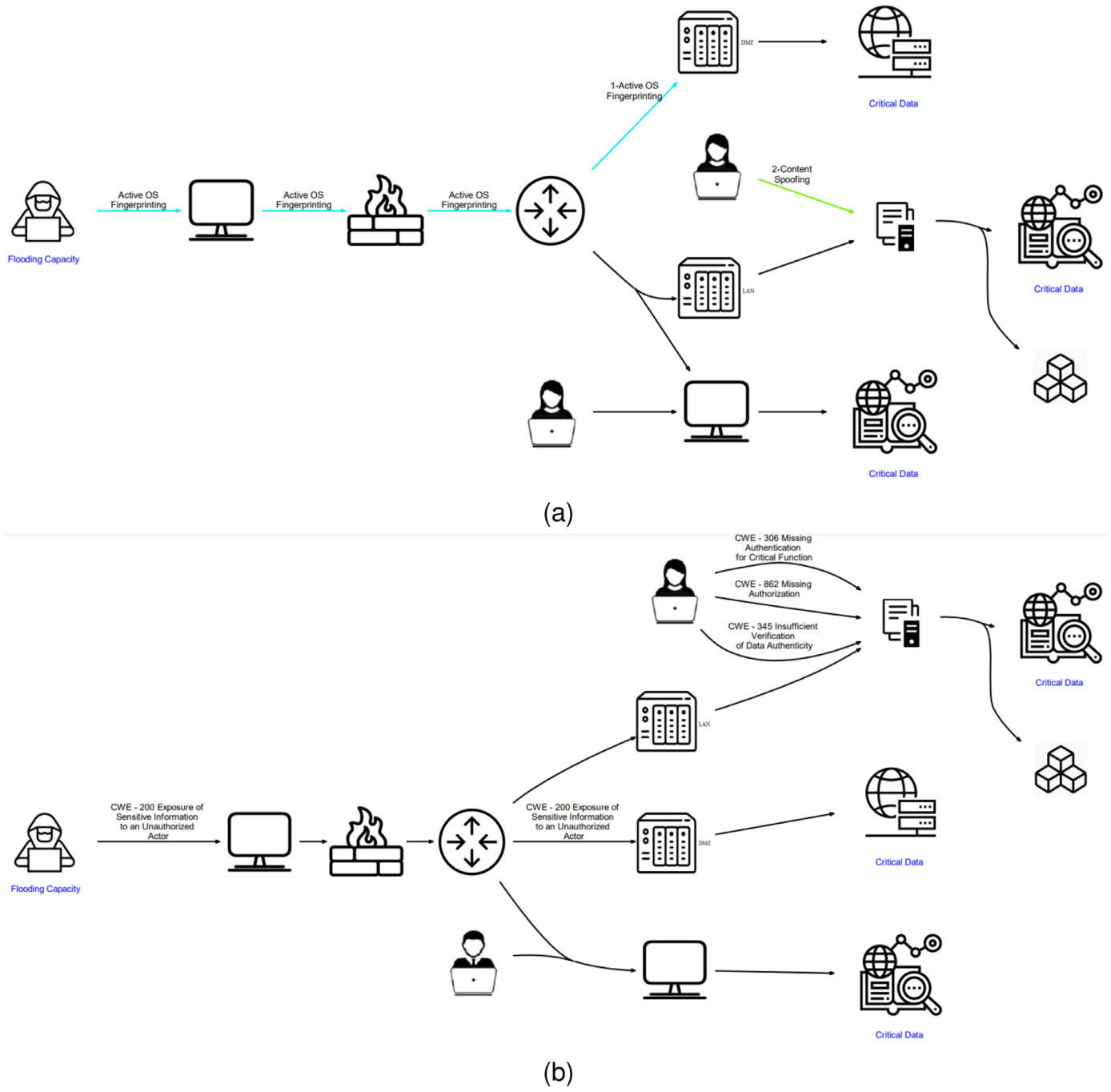


Fig. 5. Sample scenario 1- base scenario

node. None access is enabled by default on all infrastructure nodes, software, hardware, and the network. If there was an Authentication/Access control node in the system, it would have its own set of consequences. In this base scenario, four CWE weaknesses have been associated to the topology, Fig. 5 (b), CWE - 200 Exposure of Sensitive Information to an Unauthorized Actor, CWE - 862 Missing Authorization, CWE - 345 Insufficient Verification of Data Authenticity, and CWE - 306 Missing Authentication for Critical Function.

3.5.2. Effects of node properties

Some effects would be produced by assigning some node properties to the base scenario nodes. The web server is marked as Has SSL Configuration in this modified scenario, the Hacker node is marked as Has Flooding Capacity, and the workstation that the Insider user is accessing is marked as Developer/Designer Machine, indicating that it is part of a development/design environment. The User access level is also granted to this Insider user. These modifications resulted in a modified attack graph Fig. 6 (a). A new attack type, potential Communication Channel Manipulation due to SSL Configuration in the web server, can be seen in this figure. Because of the newly added node attribute to the workstation and the authenticated access by the Insider user, the Design Alteration and Development Alteration attacks exist. CWE - 701 Weaknesses Introduced During Design, and CWE - 702 Weaknesses Introduced During Design are two weaknesses associated with the workstation after the modifications. CWE - 201 Insertion of Sensitive

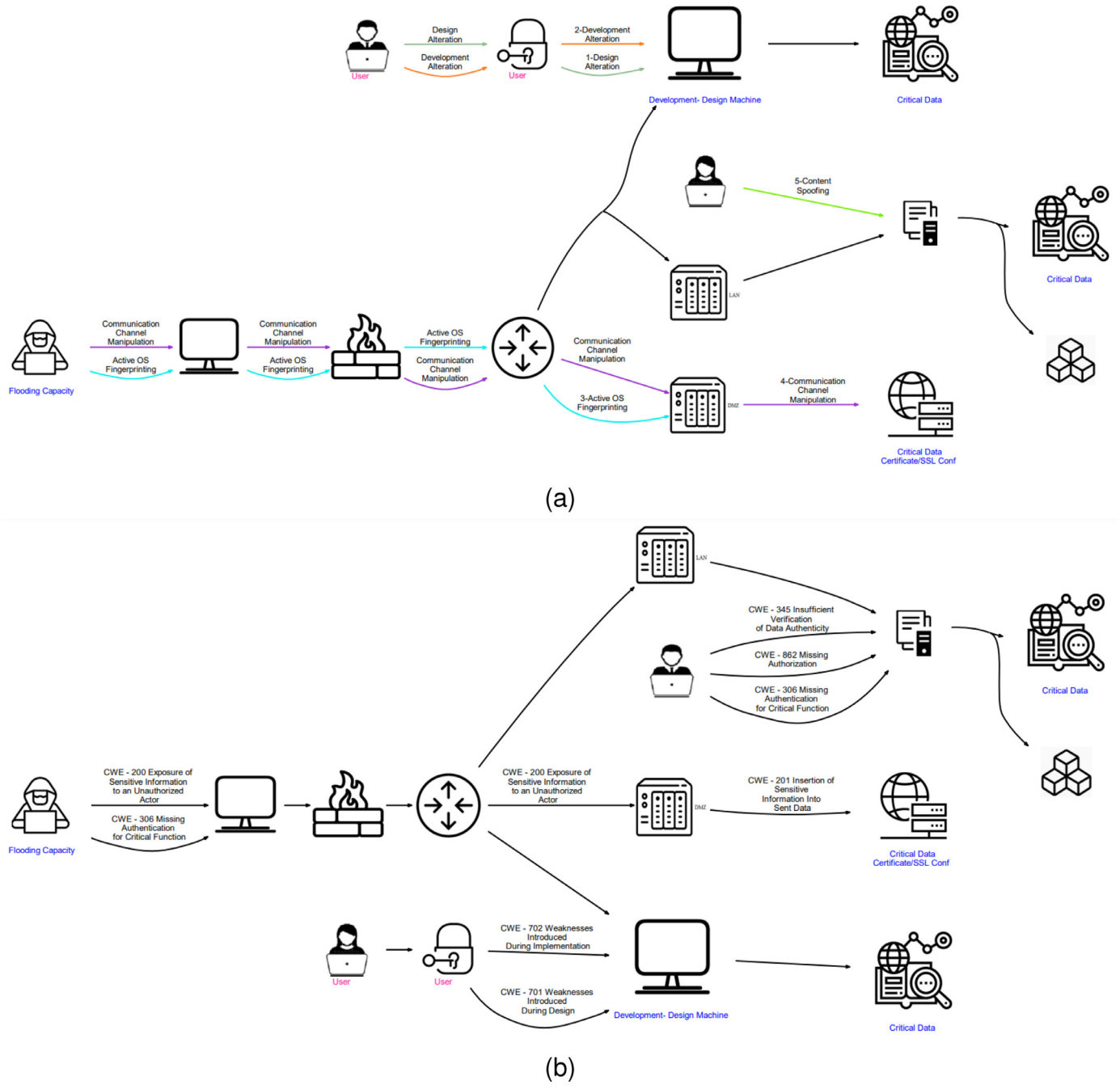


Fig. 6. Scenario 2- After node modifications

Information Into Sent Data and CWE - 306 Missing Authentication for Critical Function are related Communication Channel Manipulation attack through the use of incorrect SSL certificates or similar, Fig. 6 (b).

3.5.3. *Effects of access control*

What happens if we add a stricter access control level for the workstation. In order to draw such a scenario, one modification should be made on the system. The A/A node connected to the Insider user and the workstation has an assigned level "Admin" this time. In this modified scenario, a second A/A control is added in front of the other Insider user with the default, "User" access level. The results after these modifications are shown in Fig. 7 (a). It can be seen that since the first Insider user does not have the authentication and access rights, then the system does not detect the associated attack types towards the workstation. The addition of a second A/A system has a completely different effect on the system. Access is not an issue in this case because the A/A has the User level and the Insider actor has the User access level. Because Content Spoofing is not expected to come from an authenticated user with the necessary rights, it is not represented in this graph. The system automatically removes CWE weaknesses associated with attack types that do not exist in this version, Fig. 7 (b). Mitigation assignments are also dynamic and vary depending on the scenario.

3.5.4. *Effects of vulnerabilities*

A further change is made to the scenario at this point. The web server has been assigned a vulnerability, "CVE-2002-0392"

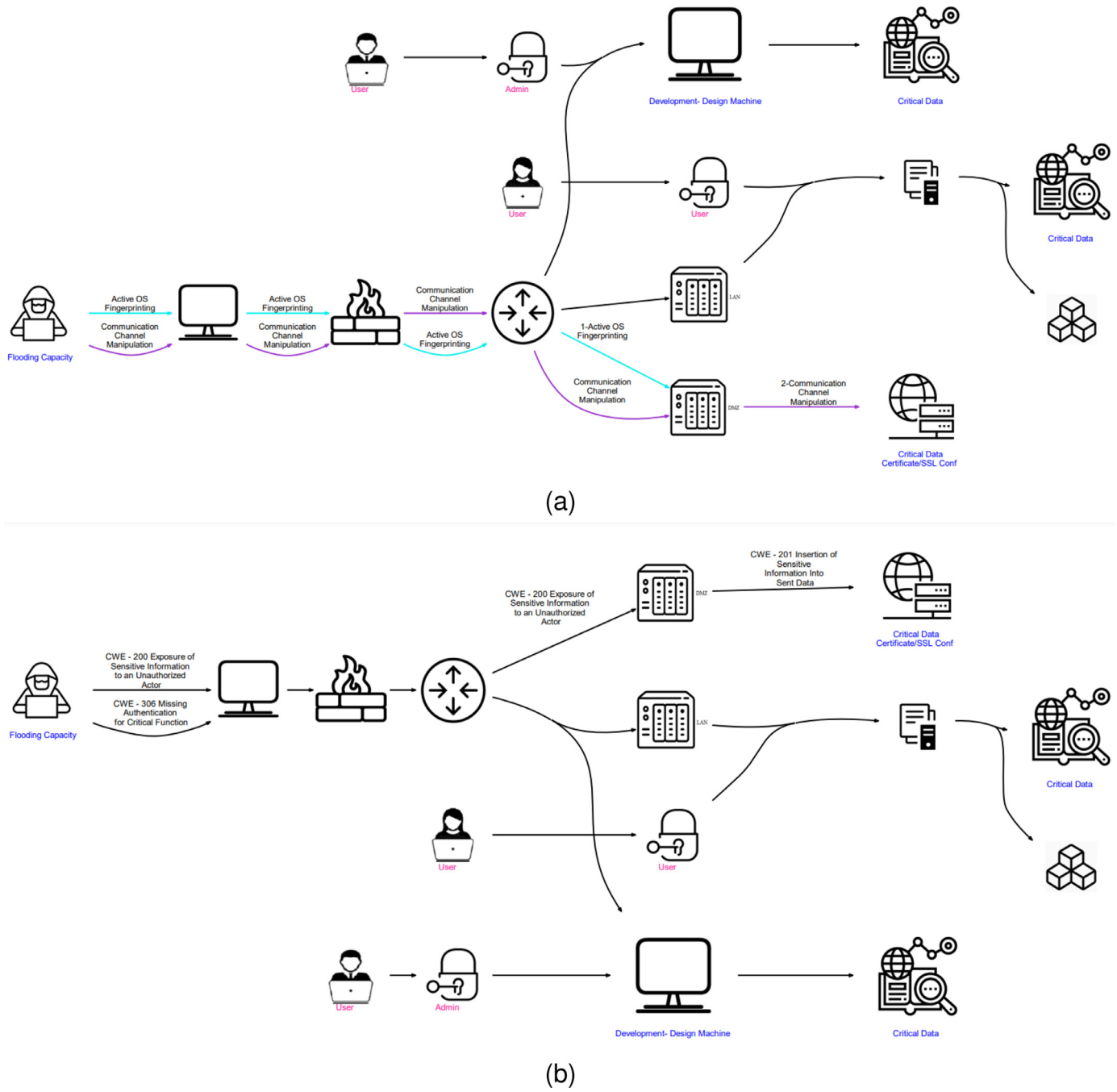


Fig. 7. Scenario 3- After Access Control Modifications

(the same CVE was employed in MULVAL). As a result of this, the web server is assigned two vulnerability attributes: DOS and Code Execution. Fig. 8 (a), Flooding, is a new attack type seen in this new Figure set. Remember that in the second scenario, even though the Hacker has the Flooding Capability attribute, the potential attack did not occur due to a lack of DOS attack marker in the server because, as previously stated, this system considers both intentions and system structure. Having the Flooding attack in the graph resulted with two new weaknesses CWE - 770 Allocation of Resources Without Limits or Throttling due to input data bombardment towards the web server and CWE - 404 Improper Resource Shutdown or Release due to primitive filtering configuration of Firewall which allowed the flooding to occur potentially in the system, Fig. 8 (b).

3.5.5. *Effects of vendor products*

The scenario is modified to include one vendor product, Rapid4 by Rapidflows, as a known vendor product on the Web Server. At this point, the scenario is further modified by marking the web server node as Uses External Libraries. As a result of this, the web server is assigned two new vulnerability attributes, File Inclusion and Directory Traversal. This resulted in a more colourful attack graph with three new attack types not present in previous graphs: Resource Injection, Code Inclusion, and Local Execution of Code.

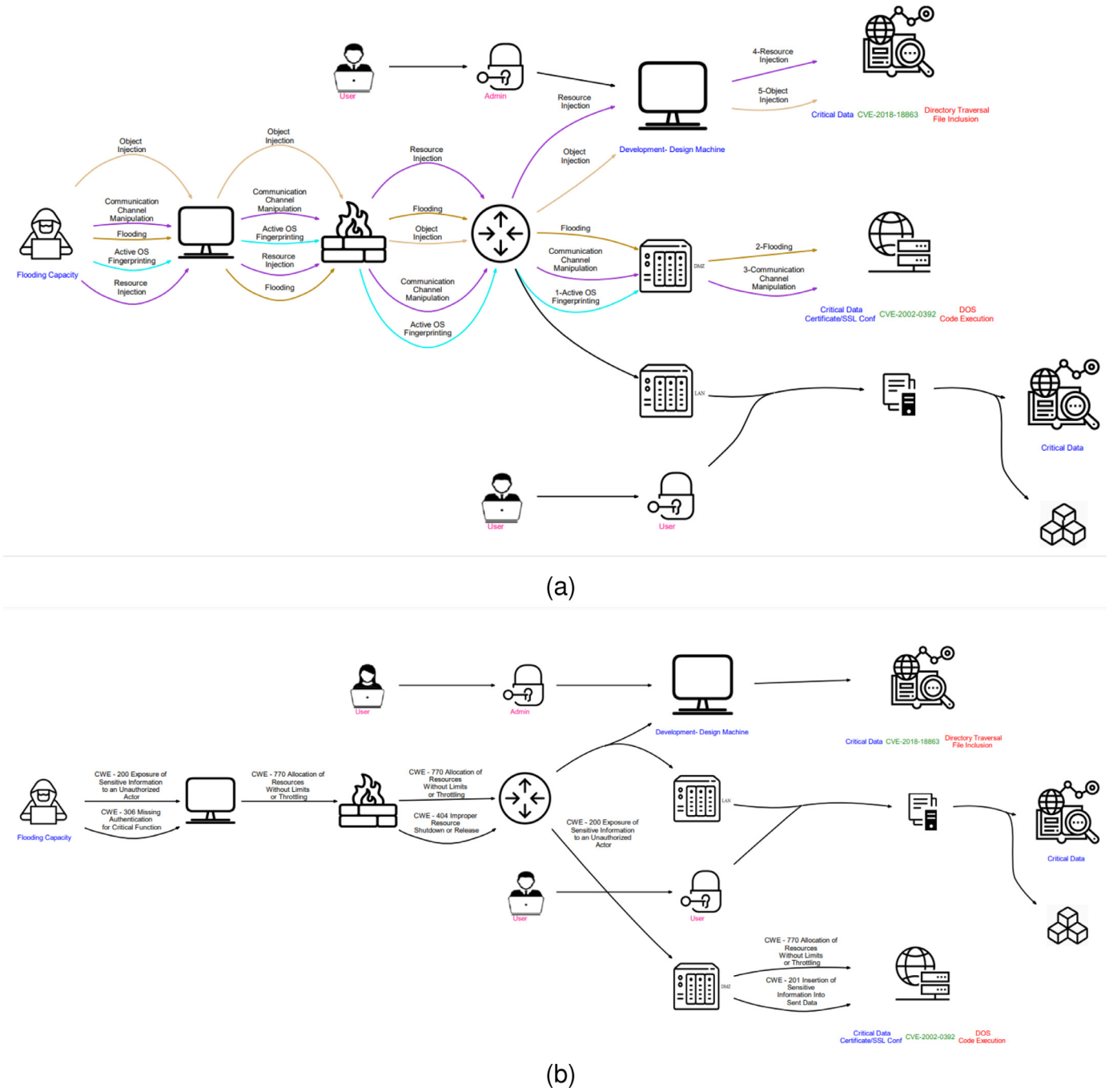


Fig. 8. Scenario 4- After vulnerability modifications

These are related to each other in some way. The graph, for example, would not have the Code Execution attack if the Code Inclusion was not present. The effects of attacks to each other are defined in attack type level. Thus, in each attack type, existence of related attacks (as a prerequisite is checked). So, far the authors didn't encounter any pairs which are prerequisites to each other. The existence of other attack types are checked during dynamic node status generation and not by executing the rules for each attack type in specific order which is less efficient. As the development progresses (covering all the meta-data CAPEC attack types), the order or attack rules can be arranged for quicker results. Fig. 9 (a) and (b) has the attack and weaknesses graphs for this scenario configuration. M0804-Human User Authentication, M1035-Limit Access to Resource Over Network, M1037-Filter Network Traffic, M1016-Vulnerability Scanning, M1017-User Training, M1047-Audit, and M1041-Encrypt Sensitive Information are the mitigations chosen (from MITRE Enterprise Mitigations List) for this attack graph, which contains many different types of attacks.

3.5.6. *Effects of security protection systems*

In this last scenario modification, first the firewall is configured to restrict access from outside of the router, by intruder to the Web Server application. Later, a specific node for an intrusion detection software (IDS) is added to the topology in a position between Router and Firewall, mainly aiming to detect attacks from outside.

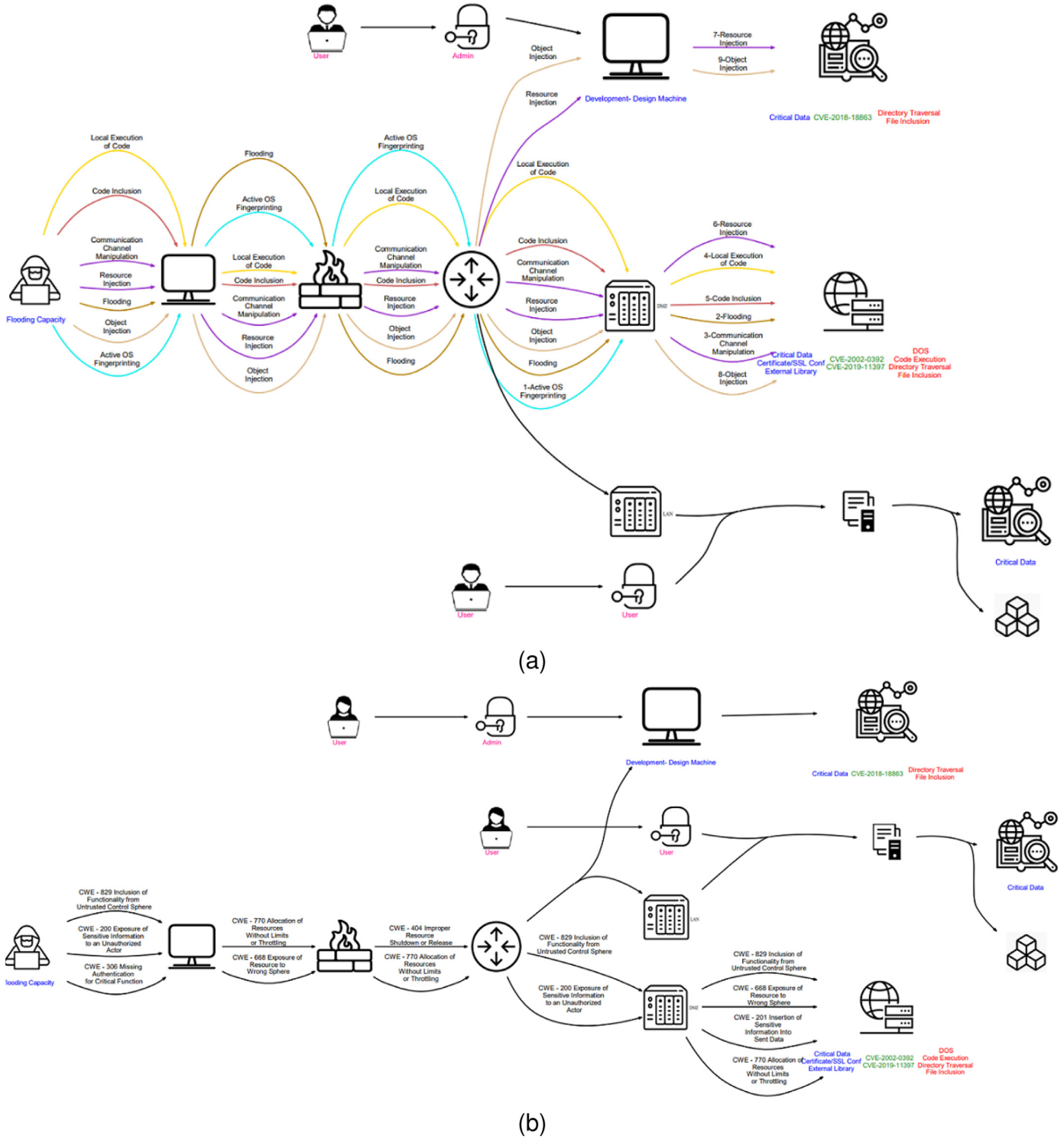
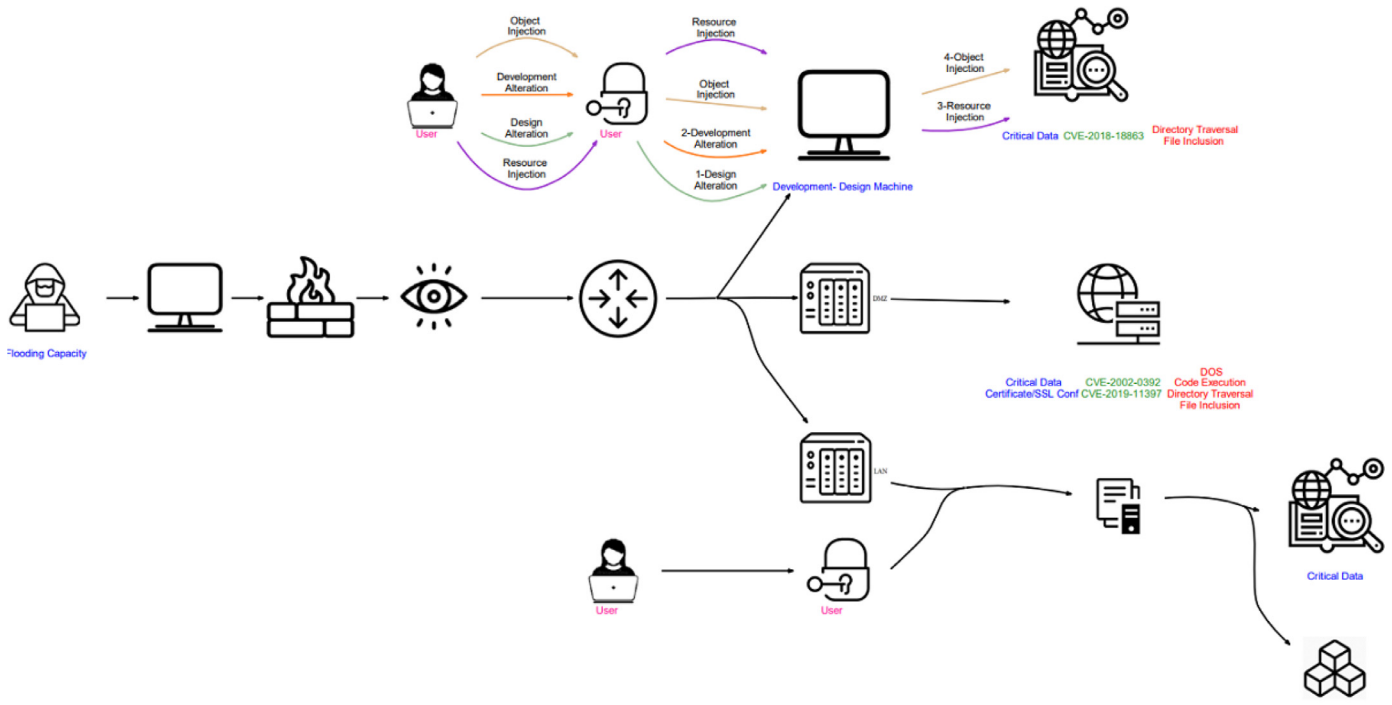
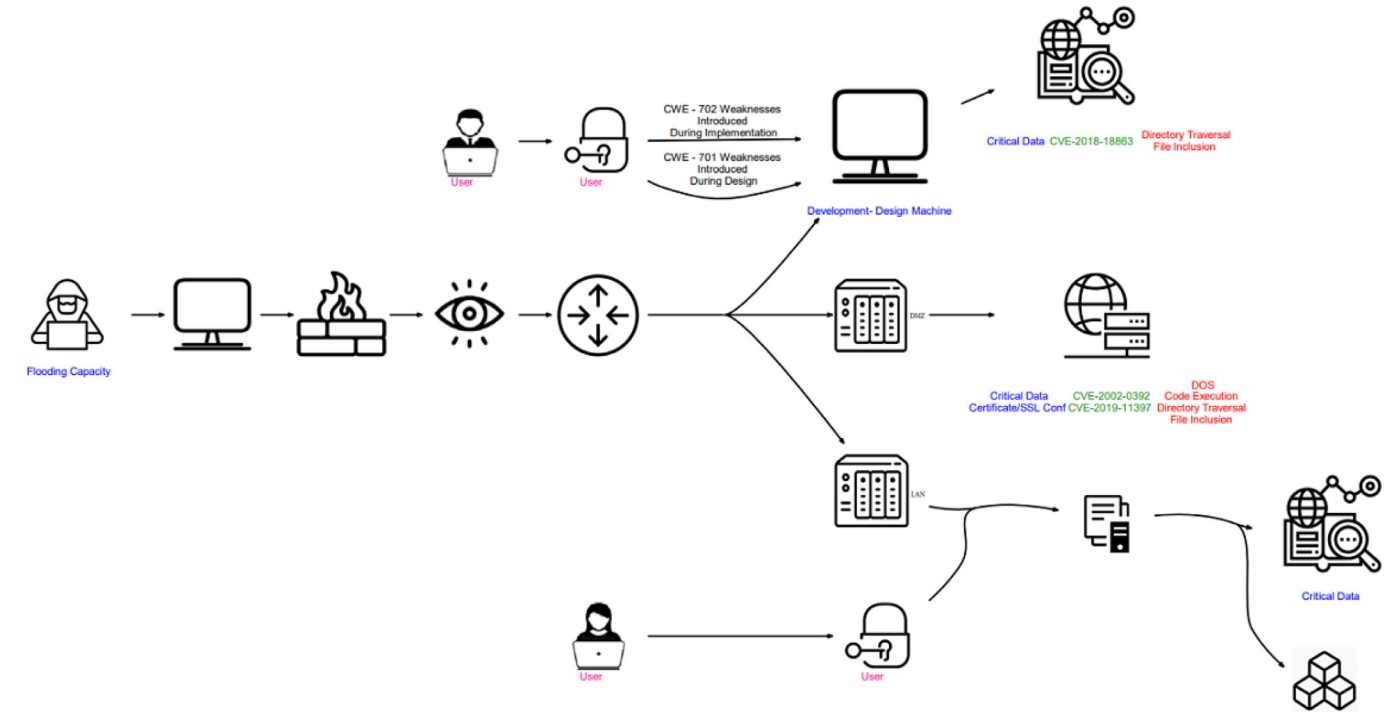


Fig. 9. Scenario 5- After vendor product modifications bringing new vulnerabilities

Besides these modifications, to show the effects of IDS, the required access right for the developer machine is reduced from Admin to User. This resulted in the diagrams shown in Fig. 10 (a) and (b). A few of the attacks from outside towards the Web Server were prevented due to Firewall configuration and some others due to having IDS. IDS also prevented the attacks towards the Critical Data node from outside again. The selected security protections systems did not have an effect on the attacks from the Insider. If we had a second IDS agent in front of the Development/Design machine then the resulting pictures would have been totally different. The list of mitigations changed due to the positions and weaknesses of the identified weaknesses resulting in the selection of M1013-Application Developer Guidance, M1022-Restrict File and Directory Permissions, M1054-Software Configuration, M1016-Vulnerability Scanning, and M1026-Privileged Account Management.



(a)



(b)

Fig. 10. Scenario 6- After security protection mechanisms setup

4. Discussion

This research provided an entirely novel approach for locating attack pathways using CAPEC descriptions. This method has several advantages. Table A.3 contains a sorted list of tool properties as well as brief descriptions of how to implement not-yet-implemented features. A large number of node types and node attributes obtained from a thorough investigation of CAPEC definitions is one improvement which are used to input real-world scenarios inputs, human-readable visualizations, and fine-tuning of the attack paths.

So far, the tool has been tested using a variety of scenarios, either from published papers (such as NIST NIST (2021a) resources) or scenarios created by the authors to test various tool capabilities.

Table 1
Execution durations for different scenarios and scopes.

	Initialization	Find Attack Paths	Draw Attack Graphs	Create Optimization Input	Draw Weakness Diagram	End
Simple Scenario 16 CAPEC Definitions, 16 nodes	0.31	0.73	2.71	2.72	3.67	6.77
Simple Scenario 1 CAPEC Definition Content Spoofing, 16 nodes	0.26	0.27	1.14	1.14	2.01	7.42
Simple Scenario 1 CAPEC Definition Code Injection, 16 nodes	0.24	0.25	1.05	1.0	1.99	5.07
Complex Scenario 16 CAPEC Definitions, 29 Nodes	0.31	4.92	6.58	6.61	7.82	13.90
Complex Scenario 1 CAPEC Definition Content Spoofing, 29 Nodes	0.24	0.27	2.21	2.21	3.64	9.17
Complex Scenario 1 CAPEC Definition Code Injection, 29 Nodes	0.25	0.29	2.08	2.08	2.88	7.45

* All units are in seconds.

Having specific node types and node attributes allows for a more in-depth investigation of various issues.

The majority of the earlier tools allow manual vulnerability assignments with some exceptions which include parsers for parsing scanner result files such as OVAL or Nessus results. The proposed system can use the vulnerability input as lists or through Nessus and Oval parsers in an integrated manner with no human interaction. Assigning vendor products directly to the tool as a new feature simplifies user input and eliminates the need for using a scanner by dealing with known and recorded vulnerabilities.

This tool does not require the input of exploit conditions, which was a bottleneck for the majority of the previous tools. Currently, the metadata is not editable, however, the authors are considering changing the present rule engine so that users may see and amend the rules in future releases. This is possible primarily due to the use of the Python programming language, and the CAPEC-based rule system, which results in a small number of attack synopsis vectors that can be easily managed and modified not only by developers but also by end-users.

One of the difficulties in finding attack paths is dealing with complex graph structures. For attack graphs showing only access-grant relations without detailed security analysis,in general, there are four main states for nodes: initial, accessed, granted, and final or possessed state. When the system is limited to this set of states, without the identification of specific attack types the possibility of cyclic relations and loops increases. There may be two main types of problems related to cycling relations. To begin with, the path traversal operation may be an infinite task. The second issue is that the status of one node can affect the status of another and vice versa simultaneously. The presence of a reasoning engine aids in overcoming the first issue. With the assistance of the reasoning engine, the search does not become a random path search in a complex graph ending in an infinite task. During the tests, pairs of two sets of new edges were introduced to create two pairs of loops (inside the same part of the network and between different parts of the network) in both scenarios mentioned in Table 1. These modifications didn't result in infinite search or a significant increase in the execution times for both of them. The second problem is mainly handled due to having specific attack types, not just the identified vulnerabilities, or limited node states. Identification of specific attack types creates newly defined states for the nodes associated with the attack types. Knowing the relations of these new states, instead of four basic states provides a clearer perspective, decreases the amount of processing difficulty, and results in less complicated diagrams. As a future task, more detailed tests on other complex topology structures will be performed after the rules metadata for all attack types (31 stable Meta Attack Patterns) are created.

Having the new states, one or more attack types can still be a prerequisite for one or more attack types. For example, Code Inclusion and Code Injection are prerequisites for Code Execution, and Code Execution on some nodes is one of the prerequisites for Communication Channel Manipulation. These types of relations, which employ states generated by attack types, are less complicated than basic state relations.

During the design and implementation of this tool, earlier tools were examined to some extent. However, this examination is primarily limited to the examination of published papers, with some exceptions for which the examination of open source code (if available) or execution of the software with the available, distributed sample scenarios were made. Unlike some tools which focus only on visualization, the tool under consideration has its own decision-making/reasoning engines. This has both benefits and drawbacks. In terms of efficiency, tools that only provide visualization would outperform tools that perform detailed reasoning and security analysis. Because the proposed software deals with complex scenarios, any tool that replicates the same node (host) a number of times (large or small) with no correspondence to the real world is expected to perform better with or without reasoning. However, relying on the powerful the Prolog engine allows the proposed software to respond in a reasonable amount of time to even the most complex scenarios.

A comparison of the proposed tool's efficiency to that of the available tools is impossible for a variety of reasons. One of the reasons is the presence of unrelated tasks in each system. The required amount of pre and post tasks such as initialization, weakness and mitigation associations, optimization file creation, and optimization make a direct comparison difficult, especially since the offered product is designed as an end-to-end security product rather than a tool that does reachability analysis or produces only one type of graphics. Another factor is the lack of implementation or numerical performance values for real-world scenarios (in most cases, only random graphs with a single node type exist) or any scenarios at all for many. The proposed system's encapsulation of multiple attack types rather than single access, grant type of vulnerability checks as in the majority of previous studies is the final reason. Each attack type has a unique set of rules, which lengthens the execution time. As previously stated, the authors did not have the opportunity to observe the execution and/or code for the majority of the tools in the literature because some of the earlier studies are only conceptual, some do not reveal an implementation or code, and others provide limited usage due to limitations/difficulties such as creating input for a specific tool.

To provide some insight about the tool's performance, Table I shows the clock time at the end of each phase (elapsed time from start and its breakdown). for two different scenarios. The simple scenario has the same setup described in the case study, the complex scenario has a different network structure with more layers and nodes. The tool's efficiency decreases as the number of CAPEC definitions incorporated into the system grows. In the case of very large topology definitions, the user can investigate only a subset of attack patterns or all of them sequentially. The final column in this table, "End", includes the execution time for optimizing the chosen mitigations given the cost constraint. The optimization time includes drawing a Pareto diagram for the given budget interval

by gradually increasing the allowed budget and executing the optimization function on each round. Thus, while the size of the network and the number and placement of the selected mitigations all have an impact, the duration of the optimization execution phase is primarily determined by the size of the allowed budget interval.

As previously stated, the automatic generation of the topology is not within the scope of this study. Even those that claim to do so, cannot do it perfectly. This effort would include the use of similar tools to this study, such as the Nessus Vulnerability Scanner tool, but it should also require other data input, such as network traffic data and firewall rules data. As a result, authors decided to treat it as a separate study, with the emphasis on the generation of topology rather than security analysis or attack path finding.

The majority of existing studies are based on user-defined exploit information. In addition to the difficulty of entering those exploit conditions, the variety of exploit conditions makes it difficult to use. Therefore the attack graphs do not cover the vast majority of security issues. Using the more abstract CAPEC definitions improves the situation significantly. Unlike many other existing studies, the system reads the attributes of the identified CVE vulnerabilities online, and the Rule Processor engine uses these attributes to make decisions. As a result, while covering a very large set of CVEs (all), depending on the CAPECs and vulnerability attributes, the effort to design and implement attack synopsis vectors becomes very reasonable.

In general, attack graphs have two layers: the topology layer and the attack visualisation layer. Attack graph generators, which act as reasoning tools that check the connectivity of nodes from a start point to an end point, act as a chain builder among a group of nodes by analysing accessibility from one point to another one by one. This creates a layer of connected nodes made up of infrastructure elements, rules, and vulnerability assignments. With these extra nodes in the attack visualisation layer, the resulting visualisations become too complex and non-relatable to the original topology input. The proposed tool has two layers as well: the topology layer and the attack visualisation layer. Unlike the chain builder, model checker approaches with additional nodes, and edges, sometimes resulting with million or nodes and edges for a 10 node topology input, the proposed approach does not include additional nodes but only the identified attack paths on top of the original topology input. The proposed system demonstrates attacks using only the original topology nodes.

Through the usage of set operations, graph simplification is made to avoid repeating the same type of attack types between same node pairs. The use of grouped and coloured human-readable labels for both attacker and targeted nodes, as well as nodes in between, as well as coloured lines and node types, allows casualty information to be obtained without having to delve into the boolean or numeric values exhibited in the previous graphsAl Ghazo et al. (2018).

The tool automatically generates a single graph for identified attacks plus multiple graphs that show only specific CAPEC types. Aside from that, it has two additional layers that are not commonly found, the CWE Weaknesses and Mitigations layers. The CWE weaknesses are displayed in dedicated graph, and the selected mitigations are automatically listed as part of pdf outputs, resulting in an end-to-end security product approach rather than a simple attack paths visualizer.

In terms of mitigation selections, the system currently offers mitigations from the MITRE ATT&CK framework's enterprise mitigations list. This could be extended in the future by creating metadata for other security control lists, such as security control families from the NIST Risk Management Framework NIST NIST (2021b) or security controls from the Center for Internet Security (CIS) CIS (2021). Following this section, there is a subsection discussing how the existing commercial tools using using CAPEC database differs from the proposed tool based on usage and purpose.

4.0.7. Other related tools

The CAPEC database has previously been used for a variety of purposes and/or in a variety of ways by various tools MITRE (2022). It is primarily used as an external library or for categorization, which suits its primary functionality. The authors found no existing tool that exploits or uses CAPEC in the same way or for the same purpose as the proposed tool. As part of the dynamic application penetration test tool, AttackForge AttackForge (2022), for example, users are allowed to manually create attack chains and associate them with MITRE ATT&CK patterns. Cairis Cairis (2022) is a security requirements management tool in which security requirements along with known threats are stored. It acts like a structured repository. It allows input of user defined keywords for various parts of the data stored. It uses these keywords to search security databases such as CAPEC and CWE. IBM started a categorization of their known attact types database based on CAPEC which was previously categorized based on CVE. Similar to previous studies, they take it as an external knowledgebase. However, the authors believe that their switch to CAPEC is an evidence of the usability of CAPEC for identifying the attacks. CybOnt SBSI (2022) is an ontology development tool. Although the demo link for the tool does not appear to work, it seems to demonstrate the CAPEC hierarchy in the form of an ontology, which is completely unrelated to the proposed tool's purpose. IriusRisk IriusRisk (2022) is a threat modeler, but not an attack graph generator. Nonetheless, since we claim that our tool can also be used as a threat modeler, it has a closer aim compared to other tools. IriusRisk IriusRisk (2022) uses CAPEC not to find attack paths, but as an external library associated with many asset types/components in the system. As a result, if the user selects a specific asset, he may see a CAPEC attack as a part of the textual threat modeling report. However, as the authors experimented on the trial version they couldn't encounter any CAPEC attack pattern, because many of the available components do not have any associated pattern. The tool isn't about attack paths, thus, including similar assets/components in the model would result in the repetition of the threats in the final list. This is a completely different approach to threat modeling. Rapid7 Rapid7 (2022) is a tool for dynamic application security testing (DAST). To identify weaknesses dynamically, the DAST tools rely on making http requests on web pages which is a totally different approach. Rapid7 refers to some known vulnerabilities in its outputs and uses CAPEC as an external reference. Synopsys Seeker Synopsys (2022) is an interactive application security testing tool. It analyzes code parts to find vulnerabilities in an interactive manner. It is not an attack graph generator tool. Potentially it uses CAPEC as a reference source. It is not revealed how it uses CAPEC in any academic source or tool web site. Similar to Synopsys Seeker, it is not revealed how Riskaware Riskaware (2021) uses CAPEC. The authors can see from the tool's video demo that it includes MITRE ATT&CK patterns, but they couldn't find any sign of CAPEC patterns through revealed demo. Open source Verdict tool Ge-High-Assurance (2022) claims to use CAPEC for categorization rather than CVE. It uses an external reasoning engine. It is not clear how it uses CAPEC. However, examining the open source code the authors could only found CAPEC as a filename provided to the third party reasoning tool.

Vfeed VFe (2022) is an intelligence feed generator continuously updated and used by third party security tools in the form of JSON file. Although CAPEC does not aim to categorize mitigations, the documentation says the tool uses CAPEC to enumerate mitigations and the details of the enumeration is not revealed. Threat Modeler Thr (2022) claims to use CAPEC along with other databases as a part of threat library. In this tool, similar to Irius-

Risk, threats are associated to system components and the list of threats and security requirements increases as the user adds new components to the visual model. It is not an attack graph generator. Pytm Foundation (2022) is an open source project which aims to conduct threat modeling based on its embedded JSON based threat library. CAPEC links are included as external references in the JSON file. Finally, VirSec Virsec (2022) is a dynamic web application security analyzer claims to use CAPEC attack patterns. There may be other tools that use CAPEC in various ways. For example, the authors didn't have access to the demo version of Praetorian tool pra (2022) and couldn't find any related academic publication. Nevertheless, although there may be other threat modelers which use CAPEC either as external resources or closely attached to system components, they do not use this library for attack path finding, and, in general, they reveal the threats in a list form associated to system components. The way how CAPEC is used for "automatic attack graph generation" is novel and unique to the proposed study.

5. Conclusion

A novel automatic attack graphing tool is presented in this paper. Starting with topology and related data input and ending with a set of visualizations, a list of attacks, a JSON file to be used in an optimization system that aims to optimise the set of controls in a limited budget, and the optimization results, an end-to-end system is implemented using Python and Prolog languages. The tool has been put through its paces in both complex and simple scenarios. The tool's usage has been demonstrated through the use of a set of scenarios drawn from published and synthesized papers. For arbitrary network definitions, the proposed system effectively finds attack paths, assigned weaknesses, and mitigations. The most significant contribution of the proposed tool is that it is the first attack graph generator tool to use CAPEC attack definitions rather than generic attack behaviors. The tool's human-readable visualization schemes are also one of its main contributions. These visualizations enable it to be used for educational purposes as well as during the planning and security evaluation of network and infrastructure setups, including the associated software and hardware nodes. The tool is simple to integrate with automatic vulnerability scanners that provide standard information by using standard vulnerability definitions. The relatively short execution times allow it to be used sequentially multiple times for complex scenarios as a part of a what-if analysis for the organizations.

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.

CRediT authorship contribution statement

Ferda Özdemir Sönmez: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Software, Project administration, Resources, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. **Chris Hankin:** Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Software, Project administration, Resources, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. **Pasquale Malacaria:** Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Software, Project administration, Resources, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing.

Acknowledgement

This work was supported by The Engineering and Physical Sciences Research Council *EPSRC* (Grant Number: EP/R002983/1 - ISAA_P67936)

Appendix

Table A1
Formulas

Table 2: Formulas

$$attack = f_1(reachability, attack\_definition) \quad (1)$$

$$reachability = f_2(topology, vulnerabilities, vendor\_products, system\_configuration) \quad (2)$$

$$\begin{aligned} &AttackGraphMetaData = \\ &NodesMetaData+ \\ &AttackTypesMetaData+ \\ &WeaknessesMetaData+ \\ &MitigationsMetaData \end{aligned} \quad (3)$$

$$\begin{aligned} &Scenario = Nodes\&Edges\&Vulnerabilities \\ &\&VendorProducts\&MitigationsDynamicMetaData \\ &\\ &where \\ &\\ &Nodes \in \mathbb{P}\, Node \\ &Edges \in \mathbb{P}\, Edge \\ &Vulnerabilities \in \mathbb{P}\, Vulnerability \\ &Vulnerability \mapsto Node \\ &VendorProducts \in \mathbb{P}\, VendorProduct \\ &VendorProduct \mapsto Node \\ &NodeNames = \{x : Node.NodeName \mid Node \in Nodes\} \end{aligned} \quad (4)$$

$$\begin{aligned} &Node = NodeName\&NodeTypeName\&NodeAttributes \\ &where \\ &NodeTypes = NodesMetaData.NodeTypes \\ &NodeTypeName \in NodeTypes \end{aligned} \quad (5)$$

$$\begin{aligned} &Edge = start \mapsto end \\ &\\ &where \\ &start \in NodeNames \\ &end \in NodeNames \\ &start! = end \end{aligned} \quad (6)$$

$$\begin{aligned} &NodesMetaData = \\ &NodeTypes\&NodeAttributes\&NodeIconswhere \\ &NodeTypes = \{Hacker, Insider, .., NetworkDeviceRouter, .., \\ &MailServerApp, ..Memory, ..CriticalDataInDataStore, ..\} \\ &NodeAttributes = \{.., HasAuthenticationSystem, \\ &HasConfigurationData, .., HasLocalUserInputDeviceAccess, .., \\ &HasClientSideScript, ..\} \\ &NodeIcons = \{icon_1, .., icon_N\} \end{aligned} \quad (7)$$

$$AttackTypesMetaData = AttacksGenericPathsMetaData+ \\ AttacksGenericRulesPackageMetaData \quad (8)$$

$$AttacksGenericPathsMetaData = \\ \{AttackType \mapsto GenericPathsMetaData\} \quad (9)$$

where

$$\begin{aligned} &GenericPathsMetaData = \\ &\mathbb{P}(NodeType\_A \mapsto NodeType\_B) \\ &where\ NodeType\_A\ and\ NodeType\_B \in NodeTypes. \end{aligned} \quad (10)$$

$$\begin{aligned} &GenericRulesPackageMetaData = \{ \\ &genericNodeStatus \in \mathbb{P}\, GenericNodeStatus \parallel \\ &genericAttackerStatus \in \mathbb{P}\, GenericAttackerStatus \parallel \\ &genericAttackerPosition \in \mathbb{P}\, GenericAttackerPosition\} \\ &\mapsto boolean \end{aligned} \quad (11)$$

$$GenericNodeStatus = \vec{NodeType} \| \vec{AccessRight} \| \\ \vec{GenericPathControl} \| \vec{NodeAttributes} \quad (12)$$

$$GenericAttackerStatus = \vec{AttackerNodeType} \| \\ \vec{AccessRight} \| \vec{GenericPathControl} \| \\ \vec{AttackerNodeAttributes} \quad (13)$$

$$\begin{aligned} &AttackerNodeAttributes \subseteq NodeAttributes \\ &AttackerNodeType = \langle hacker, \vec{stakeholder}, insider \rangle \\ &GenericAttackerPosition = \langle local, \vec{remote}, adjacent \rangle, \\ &GenericPathControl = \langle has\_aa, ..., has\_\vec{protection}1, ..has\_protectionN \rangle \end{aligned} \quad (14)$$

$$AttacksGenericRulesPackageMetaData = \\ \{AttackType \mapsto GenericRulesPackageMetaData\} \quad (15)$$

$$NodeStatus = f_3(Node, Other\_Nodes, \\ Edges, Vulnerabilities, VendorProducts, DetectedAttacks) \quad (16)$$

$$AttackerStatus = f_4(Attacker\_Node, \\ Other\_Nodes, Edges, \\ Vulnerabilities, VendorProducts, DetectedAttacks) \quad (17)$$

$$AttackerPosition = f_5(Attacker\_Node, Target\_Node, \\ Other\_Nodes, Edges) \quad (18)$$

$$AttackData = f_6(Node\_Status, Attacker\_Status, \\ Attacker\_Position, AttacksGenericRulesPackageMetaData) \quad (19)$$

$$WeaknessesMetaData = \\ WeaknessToTopologyDictionary \cup \\ CapecToWeaknessDictionary \quad (20)$$

$$WeaknessSet = \\ f_7(Attack_Data, WeaknessesMetaData, Topology) \quad (21)$$

$$MitigationsMetaData = \\ WeaknessToMitigationDictionary \cup \\ MitigationsDynamicMetaData \quad (22)$$

$$AttackGraphMetaData.MitigationsMetaData. \\ MitigationsDynamicMetaData' = \\ MitigationsDynamicMetaData \quad (23)$$

Table A2
Summary of tool features

Issue	Answer	Explanation
Input Related		
Requires structural model input (topology)	Yes	
Encapsulates security protection systems	Yes	
Merge of multiple scenarios	Yes	Not implemented yet but it is possible and easy to implement as long as same node names are repeated in multiple scenarios
Divide a very large scenario into multiple pieces	Possible	No implementation has been made to automatically divide the scenarios into multiple pieces, but having specific node-types for LAN and physical and virtual sub-parts of the LAN, as well as routers and switches would allow easy division of a large scenario into multiple pieces.
Divided scenario can benefit information from other parts	Possible	While shrinking nodes of the eliminated network parts the actor nodes can be kept in place, connected to the nearest network-sub-portion nodes keeping their distance status (local, adjacent, or remote) and types.
Enables real-life scenario definitions	Yes	Node types and node attributes allow definition of detailed real-life scenarios.
Requires behavioral model input	No	The tool evaluates all possible actions once the structure is provided for the system exhaustively. No need to define attacker actions.
Requires topology specific rule definitions or exploits	No	
Provides a user input interface	Yes	A web based interface exists. Scenarios can also be provided as files.
Generation of topology graph using Nessus	No	This is left out of scope.
Limited to a specific domain	No	
Objective and Functionality Related		
Use in real time	Partially	As long as topology definition is provided before, the system can process real-time vulnerability scanning results on multiple hosts
Built in access type rules	Yes	
Built-in attack type rules	Yes	
Built in security analysis?	Yes	No security expertise is needed threat analysis is builtin.
Security Counter Measures Provided By the Model	Yes	
Real time update of attack graphs	No	Current system does not include interactive graphs but this may be included in the future through change of selected libraries.
Effects of agents to other agents	Yes	
Effects of attack types to other attack types	Yes	
Considers attack exploits in isolation	Yes	
Considers attack exploits in combination	Yes	
Need to define target	No	The system evaluates all potential targets exhaustively.
	No	Unlike tools which find paths through backward execution of commands, this tool does not require target identification.
Shows multiple attack to single target	Yes	Multiple potential paths initiated by same or different attackers is shown on the graphs.
Shortest path calculation	No	No numeric calculation is made, still, the distance and the number of steps taken is clearly demonstrated on the graphs.
Attack Types Related		
Deals with only a single or a few attack types	No	
Single stage attacks	Yes	
Multi stage attacks	Yes	
Single target	Yes	
Multiple target	Yes	
Single Agent/Source	Yes	
Multiple Agent/Source	Yes	
Agent to agent attacks	Yes	
Fine tuning for specific attack types	Yes	Specific node types and node attributes allow fine tuning for specific attack types.
Visualization Related		
Provides multiple node types to increase readability of graphs	Yes	
Visualization of attack paths	Yes	
Visualization of weaknesses related to attacks	Yes	
Show paths with identified attack types	Yes	
Visual representation of attacks over a network topology	Yes	
Shows multiple attacks simultaneously		
Shows just reachability graph	No	
Show additional nodes or edges to mark state transitions	No	Until visualization phase attack paths are finalized.
Show additional nodes or edges to mark effects of attributes and vulnerabilities	No	Colored labels are used to mark these effects.
Show boolean or numeric values to be interpreted by the users	No	Only human readable graphics are generated.
Integration with Third Party Tools and Data Related		
Oval Integration	Yes	
Nessus Integration	Yes	
CVE Integration	Yes	
Calculation of attack success via CVE values	No	
Calculation of attack success via CWE values	Possible	
Security Control Optimization Given Budget Constraints	Yes	

References

Al Ghazo, A.T., Ibrahim, M., Ren, H., Kumar, R., 2018. A2g2v: Automated attack graph generator and visualizer. In: Proceedings of the 1st ACM MobiHoc Workshop on Mobile IoT Sensing, Security, and Privacy, pp. 1–6.

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, pp. 217–224.

AttackForge, 2022. Attackforge&reg - penetration testing workflow management, productivity & collaboration tools. http://attackforge.com/.

Auvik, 2021. Auvik- true network visibility, control, and confidence. https://www.auvik.com/.

Cairis, 2022. An open source platform for building security and usability into your software. https://cairis.org/.

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A., 2002. Nusmv 2: An opensource tool for symbolic model checking. In: International conference on computer aided verification. Springer, pp. 359–364.

CIS, 2021. Center for internet security. https://www.cisecurity.org/.

Cui, Y., Li, J., Zhao, W., Luan, C., 2019. Research on network security quantitative model based on probabilistic attack graph. In: ITM Web of Conferences, Vol. 24. EDP Sciences, p. 02003.

Feiler, P.H., Gluch, D.P., Hudak, J.J., 2006. The architecture analysis & design language (AADL): An introduction. Technical Report. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst.

First, 2022. Common vulnerability scoring system. https://www.first.org/cvss/.

Foundation, O., 2022. Owasp pytm. https://owasp.org/www-project-pytm/.

Gacek, A., Backes, J., Whalen, M., Wagner, L., Ghassabani, E., 2018. The jk ind model checker. In: International Conference on Computer Aided Verification. Springer, pp. 20–27.

Ge-High-Assurance, 2022. Ge-high-assurance/verdict: Darpa's cyber assured systems engineering (case) project named verification evidence and resilient design in anticipation of cybersecurity threats (verdict). https://github.com/ge-high-assurance/VERDICT.

Ibrahim, M., Al-Hindawi, Q., Elhafiz, R., Alsheikh, A., Alquq, O., 2019. Attack graph implementation and visualization for cyber physical systems. Processes 8 (1), 12.

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.

IriusRisk, 2022. The automated threat modeling platform. https://www.iriusrisk.com/.

Jajodia, S., Noel, S., O'berry, B., 2005. Topological analysis of network attack vulnerability. In: Managing cyber threats. Springer, pp. 247–266.

Kaynar, K., Sivrikaya, F., 2015. Distributed attack graph generation. IEEE Trans. Dependable Secure Comput. 13 (5), 519–532.

Khouzani, M., Liu, Z., Malacaria, P., 2019. Scalable min-max multi-objective cyber-security optimisation over probabilistic attack graphs. Eur. J. Oper. Res. 278 (3), 894–903.

Kordy, B., Kordy, P., Mauw, S., Schweitzer, P., 2013. Adtool: security analysis with attack–defense trees. In: International conference on quantitative evaluation of systems. Springer, pp. 173–176.

Lallie, H.S., Debattista, K., Bal, J., 2017. 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.

Lee, J., Moon, D., Kim, I., Lee, Y., 2019. A semantic approach to improving machine readability of a large-scale attack graph. J. Supercomput. 75 (6), 3028–3045.

MITRE, 2016. Common vulnerabilities and exposures. https://cve.mitre.org.

MITRE, 2021a. Capec common attack pattern enumeration and classification. https://capec.mitre.org/.

MITRE, 2021b. Cwe common weakness enumeration. https://cwe.mitre.org/.

MITRE, 2022. Common attack pattern enumeration and classification. https://capec.mitre.org/community/usage.html.

Nielsen, J., 2005. Ten usability heuristics.

NIST, 2021a. Nist national institute of standards and technology. https://www.nist.gov/.

NIST, 2021b. Nist risk management framework. https://csrc.nist.gov/projects/risk-management.

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, pp. 109–118.

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, pp. 336–345.

Ou, X., Govindavajhala, S., Appel, A.W., et al., 2005. Mulval: A logic-based network security analyzer. In: USENIX security symposium, Vol. 8. Baltimore, MD, pp. 113–128.

Product & application security. 2022. https://www.praetorian.com/product-security.

Păsăreanu, C.S., Dwyer, M.B., Huth, M., 1999. Assume-guarantee model checking of software: A comparative case study. In: International SPIN Workshop on Model Checking of Software. Springer, pp. 168–183.

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.

Rapid7, 2022. Cybersecurity & compliance solutions & services. https://www.rapid7.com/.

Riskaware, 2021. Cyberaware. https://www.riskaware.co.uk/predict.

Ritchey, R.W., Ammann, P., 2000. Using model checking to analyze network vulnerabilities. In: Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000. IEEE, pp. 156–165.

Rogers, R., 2011. Nessus network auditing. Elsevier.

SBSI, 2022. Silver bullet solutions, inc.http://www.silverbulletinc.com/demos2.htm.

Shahriari, H.R., Jalili, R., 2007. Vulnerability take grant (vtg): An efficient approach to analyze network vulnerabilities. Comput. Secur. 26 (5), 349–360.

Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M., 2002. Automated generation and analysis of attack graphs. In: Proceedings 2002 IEEE Symposium on Security and Privacy. IEEE, pp. 273–284.

Sheyner, O., Wing, J., 2003. Tools for generating and analyzing attack graphs. In: International symposium on formal methods for components and objects. Springer, pp. 344–371.

Singha, M. F., Patgiri, R., 2021. A comprehensive investigation on attack graphs.

Somesh, J., Wing, J.M., 2001. Survivability analysis of networked systems. In: Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001. IEEE, pp. 307–317.

Strom, B.E., Applebaum, A., Miller, D.P., Nickels, K.C., Pennington, A.G., Thomas, C.B., 2018. Mitre att&ck: Design and philosophy. Tech. report.

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'01, Vol. 2. IEEE, pp. 307–321.

Synopsys, 2022. Eda tools, semiconductor ip and application security solutions. https://www.synopsys.com/.

Templeton, S.J., Levitt, K., 2001. A requires/provides model for computer attacks. In: Proceedings of the 2000 workshop on New security paradigms, pp. 31–38.

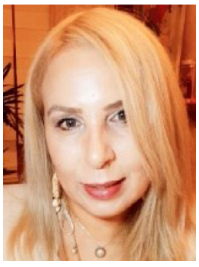
Threat modeler. https://threatmodeler.com/. 2022.

Virsec, 2022. Deterministic protection technology for software applications. https://virsec.com/.

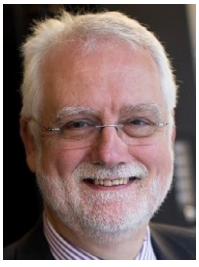
Vulnerability intel as a service. http://vfeed.io/. 2022.

Wang, H., Chen, Z., Zhao, J., Di, X., Liu, D., 2018. A vulnerability assessment method in industrial internet of things based on attack graph and maximum flow. IEEE Access 6, 8599–8609.

Zerkle, D., Levitt, K.N., 1996. Netkuang-a multi-host configuration vulnerability checker. USENIX Security Symposium.



Dr. Ferda Özdemir Sönmez (Member, IEEE) received the B.Sc. degree in electrical and electronics engineering in 1997, the M.Sc. and the Ph. D degrees in information systems in 2012 and 2014, respectively, all from Middle East Technical University (METU), Ankara, Turkey. After receiving her B.Sc. degree, she worked in the private sector for 18 years as software specialist, software development consultant, project manager and IT manager. She is holding the PMP degree since 2009. Projects she worked on were mainly in the areas of e-government and telecommunications. Currently, she is a post-doctoral researcher at the Institute for Security Science and Technology, Imperial College, London, UK. Her research interests include attack graphs, security visualization, security requirements engineering, and blockchain security



Professor Chris Hankin joined Imperial College London in 1984 and was promoted to Professor of Computing Science in 1995. He served as Dean for Engineering (2000-2003), Pro Rector for Research (2004-2006) and Deputy Principal for Engineering (2006-2008). He was Director and then Co-Director of the Institute for Security Science and Technology from 2010 until 2019. His research is in cyber security, data analytics and theoretical computer science. He leads multidisciplinary projects focused on providing better decision support to defend against cyber-attacks for both enterprise systems and industrial control systems. He has published about 130 publications including 3 textbooks and 7 edited volumes.

Professor Pasquale Malacaria is at School of Electronic Engineering and Computer Science at Queen Mary University of London. His research focuses on quantitative analysis of security with recent applications to quantifying confidentiality leaks in Linux Kernel functions and, sponsored by Google, developing a plug-in for NASA model checker Java PathFinder to quantify confidentiality leaks in Java bytecode. In 1996 he was awarded an EPSRC Advanced Fellowship to further his work on game semantics. He worked with Chris Hankin at Imperial College on applications of game semantics to program analysis and security, collaboration which brought the publication of several algorithms for program analysis and security based on abstractions of game semantics. He is an associate editor for the ACM Computing Surveys and is currently serving as program committee member for several international conferences and workshops. He is an academic visitor with the Institute for Security Science and Technology at Imperial. Malacaria has held several EPSRC grants, and currently is principal investigator for the Queen Mary parts of Games and Abstraction and Compositional Security Analysis for Binaries. The former is part of the GCHQ/EPSRC Research Institute in Science of Cyber Security and the latter is part of the GCHQ/EPSRC Research Institute in Automated program Analysis and Verification.