Topology-aware Prioritized Patching for EV Charging Infrastructure Vulnerabilities

Roland Plaka[1][a], Mikael Asplund[2][b] and Simin Nadjm-Tehrani[3][c]
[1,2,3]*Department of Computer and Information Science, Linköping University, Sweden*
{*roland.plaka, mikael.asplund, simin.nadjm-tehrani*}*@liu.se*

Keywords: Vulnerability, Prioritization, Topology, Exploitable Path

Abstract: Modern critical infrastructures are becoming increasingly complex and exposed to cyber-attacks. As with any digitalized system, these systems suffer from vulnerabilities that threaten overall system security. As a result, eliminating vulnerabilities is imperative for security analysts to counteract potential future attacks. However, vulnerability management is time-consuming and expensive because it requires testing, verification, and validation for the patches. Therefore, there is a need to prioritize which vulnerabilities to fix first in an efficient manner. This paper introduces a patching strategy by identifying the attack path that poses the most severe system risk and the patches with the highest potential to mitigate this risk. The risk assessment is based on novel metrics incorporating dynamic exploitability, impact scores, and the network topology. The method is evaluated on a case study based on electric vehicle charging infrastructures. We collect information on vulnerabilities, exploits, and available patches for this domain and instantiate a realistic network model with relevant components, some of which contain vulnerabilities. Our results show that the proposed method outperforms baseline methods to reduce overall system risk.

1 Introduction

As the usage of electric vehicles continues to grow, there is a clear need for greater interoperability with the energy infrastructure and an increasing reliance on digital technologies. The digitalization of services in electric vehicle charging infrastructure (EVCI) has its advantages and disadvantages. On the one hand, it improves these services, but on the other hand, it creates new opportunities for malicious actors to attack these systems.

Some possible attack vectors are (i) Manipulation of demand (Mad) attacks and (ii) False Data Injection (FDI) attacks, mentioned by [Kern et al., 2024], which can pose threats to grid operation. Another study from [Vailoces et al., 2023] highlights various vulnerabilities in electric vehicle supply equipment (EVSE), whose consequences could lead to physical damage to the charging stations or user privacy issues. [Alcaraz et al., 2023] study a scenario with public electric vehicle charging integrated into a microgrid-based control and identify vulnerabilities and threats using a method combining STRIDE and DREAD.

An in-depth security assessment conducted by [Sarieddine et al., 2024] analyzes the Open Charging Point Protocol (OCPP) backend implementation and reports various vulnerabilities, which allow an adversary to hijack legitimate charging station connections with the OCPP backend, replacing them with attacker-controlled charging stations, and enabling a set of covert attack scenarios with a potential goal of disrupting charging processes or launching further attacks on connected systems, causing instability of the charging network infrastructure. The implementation of EVCI is evolving, and as the number of vulnerabilities grows, it is important to prioritize the most critical ones. Even if a patch exists for a known vulnerability it is not always possible to immmediately apply it. The latest research by Kenna Security and Cyentia tracked exposed vulnerabilities at hundreds of companies and found that the monthly median remediation rate was only 15.5 percent. A quarter of companies remediated less than 6.6 percent of their open vulnerabilities monthly. These findings show that vulnerability remediation is complex and requires more efficient efforts specially in ecosystems dominated by many actors and many small or medium sized enterprises.

[a] https://orcid.org/0009-0000-5847-6317
[b] https://orcid.org/0000-0003-1916-3398
[c] https://orcid.org/0000-0002-1485-0802

As the system's complexity increases, testing every possible threat scenario becomes harder. Consequently, the likelihood of hidden vulnerabilities within the dependencies (such as libraries and frameworks) between hardware and software components rises, potentially opening new attack vectors. The huge number of vulnerabilities is a challenge for system security analysis, so there is a need for a smarter method to prioritize vulnerabilities.

This paper presents a topology-aware prioritized patching method. Our approach is valuable when dealing with complex topologies and accessible vulnerability information for the used technologies. In the context of risk management, vulnerability prioritization is a critical sub-process. Network topology, in this sense, describes the interdependencies and communication paths between hardware and software assets.

Vulnerability prioritization is essential because it distinguishes the risk that different vulnerabilities pose to the system. As a result, practitioners working on vulnerability management can allocate resources effectively to patch the vulnerabilities and reduce the overall risk. The main contributions in this paper are as follows:

1. A method called Dynamic Patch Management (DPM) for vulnerability instance prioritization to reduce overall system risk.
2. An empirical study on vulnerabilities, exploits, and patches in EV charging.
3. Evaluation of the proposed method using a prototype implementation and scenario based on the collected data.

The paper is structured as follows. In Section 2, we present our proposed method by describing the main steps in detail and a dedicated subsection where we introduce the prerequisites. Section 3 describes the approach for the empirical study and provides insights about the collected data on vulnerabilities, exploits, and vendor patches. Section 4 shows the evaluation of our method on a use case, including the results in comparisons with existing methods. Section 5 discusses related work in the target domain, and section 7 concludes the paper.

2 Dynamic Patch Management

This section introduces our proposed approach, Dynamic Patch Management (DPM), which ranks specific vulnerabilities based on their risk including potential impact on a system and performs patches for the most critical ones. Figure 1 shows an overview of the DPM method. We assume as input a model of the system that includes the network topology as well as information about vulnerabilities. Moreover, we rely on standard vulnerability information such as vulnerability impact scores, as well as information about exploitability scores for the relevant vulnerabilities. With this information available, the DPM metod is composed of three steps, as follows.

1. **Risk calculation**: The first step of the method is to calculate the base risk level for the system. We propose a new metric that combines vulnerability impact with dynamic exposure data.
2. **Attack simulation**: In this step, we apply our proposed attacker algorithm to find all the shortest attack paths in the system starting from predefined entry points.
3. **Prioritized patching**: The final step prioritizes mitigating vulnerabilities based on attack paths and the prioritization algorithm's output. We select the one that introduces the highest risk to the system among the generated attack paths.

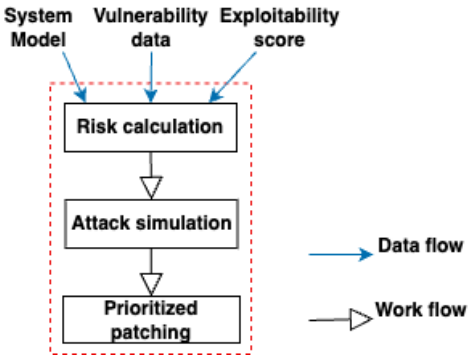

Figure 1: DPM method

We now proceed to provide more details about the necessary system model as well as the other inputs we assume, followed by a subsection for each step in the DPM method.

2.1 Prerequisites

System model: In our method, the topology of a system and its set of known vulnerabilities is defined using a graph. The *system S* is a tuple of *hosts H*, *links L*, *vulnerabilities V* and mapping *c* as follows :

$$S = \langle H, L, V, c \rangle$$

- $\boldsymbol{H} = \{h_1, h_2, \ldots, h_n\}$: is the set of hosts in the system
- $\boldsymbol{L}$: is the set of logical links connecting the hosts; a link is a pair $l = \langle h_i, h_k \rangle, h_i, h_k \in H$ and $i \neq k$

- $\boldsymbol{V} = \{v_1, v_2, \ldots, v_m\}$: is the set of vulnerabilities in the system
- $\mathbf{c}(h, v)$: denotes that a host $h \in H$ contains the vulnerability $v \in V$.

We define the following predicates over **S**:

- *hasInitialAccess*(h): denotes that a host $h \in H$ is initially accessible by an adversary.
- *hasExploit*(v): denotes that a vulnerability $v \in V$ has an available exploit
- *isEntryPoint*(h): denotes that a host $h \in H$ is a potential entry point for an attacker. This is true when *hasInitialAccess*(h) = True and $c(h, v) =$ True for some vulnerability $v \in V$ such that *hasExploit*(v) = True. This means that each entry point must have at least one vulnerability with an available exploit.

Vulnerability data: We assume access to Common Vulnerabilities and Exposure (CVE) data. Each CVE describes a specific security flaw in the system's assets, such as software or hardware components, that attackers could exploit. The impact score represents the potential impact a vulnerability could have if exploited. To account for the impact of a vulnerability exploitation, we consider threats to confidentiality, integrity, and availability properties using three levels: High, Low, and None. We borrow the notion of confidentiality, integrity, and availability impact from [FIRST, 2019]. CVSS defines the impacts both qualitatively as *high*, *low*, and *none*, but also numerically.

Exploitability score: While the CVE information also contains information about exploitability, there are several problems with this metric, including the static nature of scoring, which assigns the severity of a CVE once and does not change over time. There is need for a dynamic metric that reflects the real-world exploitability of CVEs and adapts as new exploits emerge and systems evolve within a changing threat landscape.

As an alternative metric, the *EPPS* (Exploit Prediction Scoring System) [Jacobs et al., 2023] estimates the likelihood of any exploitation attempt for a given vulnerability in the next 30 days. EPSS scores range from 0 to 1, with higher values indicating a greater likelihood of exploitation. By incorporating EPSS in our calculations, we can prioritize vulnerabilities that have a high impact and are more likely to be exploited by attackers. We generate exploitability scores from the open source Tesorion tool developed by T-CERT [1].

[1] https://github.com/Tesorion/vulnerability-explorer

This user-friendly tool helps incident response teams identify vulnerabilities in a particular application and enhance it with data to calculate the likelihood of exploitation using EPSS.

2.2 Risk calculation

The first step of the DPM process is to establish the baseline system risk level. This number is calculated using the notions of *frequency* and *vulnerabilityRisk* as follows.

The frequency of a vulnerability $v \in V$ equals the number of hosts h for which $c(h, v)$ holds, divided by the total number of hosts in the system as shown in Equation 1:

$$Frequency(v) = \frac{1}{|H|} \sum_{h \in H} c(h, v) \tag{1}$$

vulnerabilityRisk: denotes the risk associated with a single vulnerability and is calculated as in Equation 2:

$$\text{vulnerabilityRisk}(v) = \text{EPSS}(v) \times \text{Impact}(v) \tag{2}$$

Quantified values of impact used in the calculation can for example. be chosen as follows:

- *High Impact*: A significant breach of any security property. The numerical value for this level is 0.56. This is the value used in CVSS v3.0.
- *Medium Impact*: A minor compromise of any security property. The numerical value for this level is 0.22. This is the value used in CVSS v3.0.
- *No Impact*: No considerable impact on security properties. The numerical value for this is 0.

Finally, the ***systemRisk*** is a quantitative value from 0 to 1 representing how much a given system is exposed to attacks based on the frequency of vulnerabilities and their associated risks. The *systemRisk* is calculated as in Algorithm 1 (line 2-7).

The estimation of the total system risk is obtained by summing up the contributions from all vulnerabilities. We simplify calculation of the risk by considering exposure of hosts and assuming that all hosts are impacted by equal amounts due to known vulnerabilities. Adding weights to various hosts to signify higher impact is also possible and could be a straight forward extension of the approach.

2.3 Attack simulation

In this subsection, we introduce a potential strategy that an attacker could use to traverse the target system. We illustrate this strategy by presenting the attacker algorithm in more detail. Note that the attacker

Algorithm 1: calculateSystemRisk

Require: Set of hosts H, Set of vulnerabilities V, function denoting vulnerable hosts c
Ensure: systemRisk *sysRisk*

1: $sysRisk \leftarrow 0$
2: **for all** vulnerabilities $v \in V$ **do**
3: $\quad frequency \leftarrow \frac{1}{|H|} \sum_{h \in H} c(h, v)$
4: $\quad epss \leftarrow EPSS(v)$
5: $\quad impact \leftarrow impactScores(v)$
6: $\quad vulnerabilityRisk \leftarrow epss \times impact$
7: $\quad sysRisk \leftarrow$
$\quad sysRisk + (frequency \times vulnerabilityRisk)$
8: **end for**
9: **return** *sysRisk*

algorithm is used to identify the most critical vulnerabilities and should not be seen as a generic model of how attackers behave.

We assume a rational or destructive attacker that initially only has access to the entry points. The attacker's goal is to compromise as many hosts as possible. The constraint is that the attacker reaching a patched host or a host with no vulnerabilities is blocked from proceeding. To model an attacker in our method, we utilize a breadth-first-search (BFS) strategy. BFS is an algorithm for traversing or searching a tree.

It starts at the root node (which, in our case, corresponds to an entry point) and explores all neighbor nodes at the present depth level before moving on to nodes at the next depth level. In this context, BFS visits all reachable hosts from any entry point in the network graph while recognizing cycles in the graph to reach termination. The attacker's purpose is to find and attack all possible vulnerable hosts. This approach explores all paths systematically, ensuring that all shortest attack paths are considered.

The algorithm has three phases:

- Queue Initialization: The algorithm starts with a queue initialized with some entry point. Each element in the queue will be a tuple containing the current host and the path taken to reach it.
- Exploration of Neighbors: The algorithm dequeues an element and explores its neighbors. For each neighbor, if it hasn't been visited (and is not patched), it gets added to the queue, and the path is extended.
- Completion: The process continues until all nodes reachable from the entry point have been explored. The result is a list of possible paths an attacker could take starting from the initial entry point.

We model an attacker's strategy to traverse the network. As illustrated in Attacker Simulation Algorithm 2, the attacker explores the network from a specified entry point e, and prioritizing hosts with high connectivity and exploitability. The algorithm initializes three variables: R (reached hosts), Q (queue) and P (path). R denotes a set to track visited hosts, starting with an entry point *e (line 2)*. Q contains a queue of pairs, where each pair consists of a host to visit, together with the path leading to the host from the entry point (line 3). P is a set that stores all distinct paths explored by the attacker (line 4).

The main loop (line 5) processes elements from Q until the queue is empty. For each iteration, the algorithm pops a path from Q, and for each neighbor of the current host, evaluates unvisited neighbors based on the number of connections and the EPSS value. These properties are appended to the list of candidates C (line 6-14).

The sort(C) function, called on line 15, sorts the candidates first by the number of connections (descending), as the attacker prefers hosts with more connections for lateral movement. Since the attacker performs reconnaissance and discovery to understand the target environment and they are able to identify vulnerabilities.

Once they have sufficient information, they can evaluate and decide which part of the system to attack. Then if two hosts have the same number of connections, the attacker compares their average EPSS scores. A higher EPSS indicates a greater likelihood of vulnerability exploitation. If the EPSS scores are equal, the attacker selects randomly.

The algorithm begins by selecting a candidate from the available options (line 16) . If a candidate is found, (line 17), it is added to the set R, which records the hosts that have been reached, preventing the attacker from revisiting previously explored hosts (line 18). The selected host is then appended to the path sequence, and a new path is created by extending the previous one with this host. The updated path is pushed back into the queue for further exploration (line 19).

The attacker continues exploring and extending paths until there are no more valid candidates or all reachable hosts have been explored. If no candidates are found, the current path is terminated and added to the list of paths. Once all paths have been explored, the algorithm returns the set of distinct paths, line 20, representing the different routes the attacker could take through the system.

Algorithm 2: Attacker Algorithm

Input: S System model, e Entry Point, V Vulnerabilities
Output: P Set of paths

Initialize:
$R \leftarrow \{e\}$; // Reached hosts
$Q \leftarrow [(e,[e])]$; // Queue of hosts to visit, with path from entry point
$P \leftarrow \varnothing$;
// Set of completed attack paths
while $Q \neq []$ **do**
  $(h,p) \leftarrow Q.pop()$;
  // Extract the current host and the path taken to reach it
  $C \leftarrow []$; // Candidate hosts to visit next
  for *each* $n \in S.neighbors(h)$ **do**
    if $n \notin R$ **then**
      $conn \leftarrow$ getConnections(n,S);
      $EPSS \leftarrow$ getEpss(V,n);
      C.append$((n,conn,EPSS))$;
      // Add candidate hosts with their connection and EPSS data
  if $C = []$ **then**
    P.add(p);
    // Add the current path to the set of paths if no new candidates
  sort(C);
  // Sort the candidate hosts based on chosen criteria
  while $C \neq []$ **do**
    $next \leftarrow C$.pop();
    // Choose the next candidate host
    R.add$(next)$;
    // Mark the host as reached
    Q.append$((next, p+[next]))$;
    // Append the next host to the path and continue exploring
return P;
// Return the set of all paths explored by the attacker

2.4 Prioritized Patching

In this subsection, we explain what the proposed defender algorithm does. The defender algorithm performs patching to minimize system risk in a given topology vulnerable to attacks.

The algorithm takes as input the system model

$$S = \langle H, L, V, c \rangle$$

described in Section 2.1 with a base risk calculated in Algorithm 1, a set of paths, and a set of patches. It creates a dictionary named pathRisk(o) to store the system risk value for each path after hypothetical patching. It then iterates over each attack path in P . Within each path, the algorithm iterates over hosts that the path traverses.

For each host, the algorithm examines the vulnerabilities it possesses. The c' function evaluates whether a vulnerability should be patched by evaluating its impact on the current path's risk. If a patch is applied and it successfully reduces the risk, c' returns 0, indicating the vulnerability has been patched. Otherwise, if no patch is available for the vulnerability, c' returns 1, indicating that the risk from the vulnerability persists on the path (line 1,2).

After simulating the patching process for all vulnerabilities along the path, the algorithm calculates the total system risk by calling the function calculateSystemRisk(H,V,c') in Algorithm 1 with the updated vulnerabilities (line 3). This function computes risk based on the hosts, vulnerabilities, and the alternative mapping c', which depends on the vulnerabilities patched. The calculated risk for that path is then stored in pathRisk(o).

After calculating risk values for all generated attacker paths, the algorithm identifies the path with the highest residual risk. This is done by finding the minimum risk value in pathRisk(o) (line 4). Then, the algorithms return the best path to patch, patches, and the updated state of the system (line 5).

As shown in Algorithm 3, the defender algorithm focuses on efficiently patching vulnerabilities to mitigate potential attacks. Note that the algorithm does not consider different patches having different costs to apply.

Algorithm 3: Defender Algorithm with Patching

Input: The system model $S = \langle H, L, V, c \rangle$, the set of paths P, the set of patches F

1 **for** $p \in P$ **do**
2 $c'(h,v) \leftarrow \begin{cases} 0 & \text{if } h \in p \text{ and } F \text{ patches } v, \\ c(h,v) & \text{otherwise} \end{cases}$
3 $pathRisk(o) \leftarrow$ calculateSystemRisk(H,V,c')
4 $chosenPath \leftarrow \arg\min_{p \in P} pathRisk(o)$
5 Apply patches to $chosenPath$

3 Empirical Study

This section outlines the methodology used in our empirical study, focusing on the collection and analysis of data related to vulnerabilities, exploits, and vendor patches. The datasets serve as the primary inputs for our method, enabling its application and assessment. We provide insights into the types of vulnerabilities identified, corresponding exploits, and available patches, which are integral to our approach.

While several works have examined various aspects of Electric Vehicle Charging Infrastructure (EVCI) security, this study is the first to conduct an empirical analysis of vulnerabilities, exploits, and patches specific to EVCI assets. By mapping known CVEs, their exploitability, and potential impacts, we identify the most threatened components and offer insights into how attackers might target EVCI systems.

This section is critical for the rest of the paper as it provides the foundational data necessary for our method, which is directly applied in the subsequent analysis. Our findings are based on an architecture similar to that described by [Alcaraz et al., 2023], modeling an electric vehicle charging infrastructure within a microgrid, ensuring the relevance of our study to real-world EVCI systems.

3.1 Vulnerability collection

We start by collecting EVCI-relevant data from various sources such as search engines and vulnerability databases using keywords like "OCPP," "charging stations," "CSMS," "EVSE," hardware/software components, and product names. After reviewing all the possible hits generated from our keywords, we removed the duplicates, resulting in 46 vulnerabilities related to EVCI with assigned CVE IDs. The severity scores of the selected vulnerabilities range from 4.3 to 9.8.

The fact that we only found 46 EVCI-related vulnerabilities shows that they are underreported. Table 1 shows the EVCI components ordered by the number of reported vulnerabilities related to each component. The analysis of this table shows that charging stations (CSs), charging station management systems (CSMSs), and charging station controllers (CSCs) appear to be the most frequently targeted assets. They commonly suffer from firmware or software vulnerabilities, making them the primary entry points to the system architecture.

Table 1: Distribution of vulnerabilities in EVCI infrastructure

Asset Type	Known CVEs
Charging station(CS)	21
Charging station controller(CSC)	6
Automation server(AS)	1
Electric Vehicle Supply Equipment (EVSE)	1
Charging station management system (CSMS)	6
Industrial network switches(SW)	5
Load Management System (LMS)	2
Power meter(PM)	1
Electric Vehicle(EV)	1
Industrial Gateway(GW)	1
Remote telemetry system(RTS)	1

3.2 Exploit collection

As discussed earlier, our primary focus is on the exploitability of the collected vulnerabilities. Hence, we investigate and find the available exploits in the wild that attackers can use to hack the EVCI infrastructure.

[Yoon et al., 2023] proposed a framework to detect exploits in the wild by collecting exploit data from sources like Exploit-DB, Github, and CISA, which provides Known Exploited Vulnerability (KEV) information. Building on their work, we extend this list of sources by including additional exploit databases such as Packetstorm, Coalition, Sploitus, and 0day.today. By querying these databases—comprising both the original sources from [Yoon et al., 2023] and the additional ones we identified — we detect 11 available exploits for the collected CVEs using CVE IDs and the affected products as search entries.

Table 2 shows the identified exploits aligned with their exploitability scores. From these two tables, we can see that for 46 vulnerabilities, there are 10 available exploits, which means 1 in 4 detected vulnerabilities can be immediately exploited with the help of online databases to damage the EVCI infrastructure.

Table 2: Released exploits in EVCI infrastructure

CVE	Asset	Exploit	CVSS	EPSS	Source
2020-8006	CS	Buf. Overflow	9.8	0.1	PS
2018-12634	CSC	Cred. Disclosure	9.8	0.95	Sploitus
2023-28343	LMS	Comm. Inject.	9.8	0.95	EDB
2022-22808	LMS	CSRF	8.8	0.11	CESS
2016-5809	PM	CSRF	8.8	0.22	EDB
2021-34591	CSC	Priv. Esc.	7.8	0.04	CESS
2023-49955	CSMS	DoS	7.5	0.05	CESS
2021-22708	CS	Auth. Bypass	7.2	0.1	PS
2016-2278	AS	Multiple Vuln.	7.2	0.02	0day
2022-0878	CST	DoS	6.5	0.04	CESS
2020-15912	EV	Relay Attack	6.5	0.1	PS

3.3 Patch collection

The patching process requires continuous monitoring of the CVE fixes, especially those released by ven-

dors. Keeping track of the recent patches is time-consuming because one must monitor vendor websites, GitHub releases, or other communication channels. These channels are not fixed and can vary from vendor to vendor.

In our empirical study, we did not find many available patches published by the vendors. This shows that not all vendors share their patches on their websites; possibly because they use subscription-based sharing with their clients via e-mails or other private channels. We detect the following patches from one vendor only on their website [Electric, 2025].

The vendor provides clear instructions on performing the patching process and proposes mitigations. More details about various advisories can be found in a related article [SecurityWeek, 2025].

Table 3 gives an overview of the collected patches for a few CVEs, the impacted device, and their impact score.

Table 3: Known CVEs with released patches

Vulnerabilities	Hosts	Impact Score
CVE-2021-22820	CS	0.56
CVE-2021-22730	CS	0.56
CVE-2021-22727	CS	0.56
CVE-2021-22707	CS	0.56
CVE-2021-22725	CS	0.22
CVE-2022-22808	LMS	0.22
CVE-2021-22726	CS	0.22
CVE-2021-22708	CS	0.22
CVE-2021-22818	CS	0.22
CVE-2021-22774	CS	0.22
CVE-2022-22807	LMS	0.22
CVE-2021-22773	CS	0.1
CVE-2021-22721	CS	0.1

4 Evaluation

In this section, we evaluate the DPM method and compare its performance with 2 baseline approaches. We aim to assess the effectiveness of our patching strategy in reducing system risk. Specifically, we focus on how well DPM minimizes the total risk in the system after patching vulnerabilities along the attacker's most exploitable path.

4.1 Evaluation methodology

In our evaluation method, we develop a graph-based mesh topology to express the relationships between the EVCI infrastructure components, as shown in Figure 2. We model the system utilizing the NetworkX library[2]. NetworkX is a Python package mainly used for the creation and analysis of the structure, dynamics, and functions of complex networks. Our topology consists of nodes and edges where nodes (i.e., hosts) refer to the EVCI infrastructure assets. Edges represent the links between the hosts in our model. Every host has its name and additional properties. Utilising the empirical study knowledge, we assume that attackers can start exploiting from some of the entry points, e.g., "CSC," "CS", "CSMS", "LMS", and "SW", as they the components with the most vulnerabilities.

We compare DPM with two baseline methods. The first baseline for comparison is the earlier one described by [Petho et al., 2021], which is the closest method we can compare DPM with. In their work, the authors assign weights to the links in the graph to represent the presence of security measures. For links connecting any type of component that does not have any protective measures in place, the authors assign weight 1 to the link. For the links connecting one graph component with a gateway and with no security measure in place, they assign weight 2. To ensure a fair comparison, we have adopted the same weighing scheme for the links in our graph, maintaining consistency with their approach.

To compare DPM with a second baseline, we apply Dijkstra's algorithm as the attacker algorithm, seeking the shortest attack path on a simpler model. In this case, each link is assigned a uniform weight of 1, and the attacker selects paths with the fewest hops between hosts, regardless of security measures. After finding the shortest paths, we apply our defender algorithm to patch the vulnerabilities on the paths and calculate the overall system risk reduction.

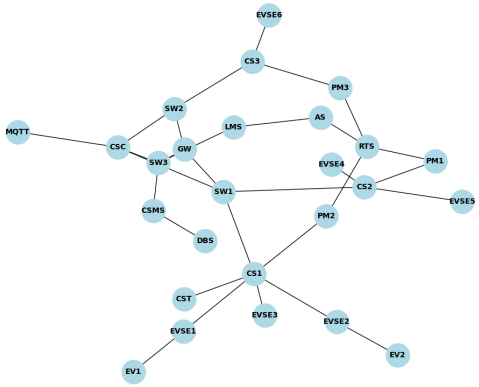

Figure 2: EV charging infrastructure

[2]https://networkx.org/

4.2 Evaluation metric

The success of the compared methods is measured by the relative risk reduction metric in the system risk after patching. The system risk is quantified using an initial score, which represents the vulnerabilities present in the system before any patches are applied. The formula for relative risk reduction is given by Equation (3) as follows:

$$\text{Risk Reduction } (\%) = \left(1 - \frac{R_{\text{post}}}{R_{\text{pre}}}\right) \times 100 \qquad (3)$$

where R_{pre} represents the system risk before patching, and R_{post} represents the system risk after patching. This gives a value between 0% and 100%, where 100% represents complete risk reduction (i.e., no remaining risk), and 0% indicates no risk reduction.

4.3 Results

In this section we present the results of the evaluation. Table 4 shows the risk reduction when applying one of the three evaluated methods starting from a specific entry point. The entry points are all the nodes for which we found published exploits (Table 2). DPM and the other two methods generate different paths that are not guaranteed to be the same length or cover the same amount of vulnerabilities. We take into account the riskiest path for DPM and compare only the riskiest path per algorithm and not a cumulative metric of overall paths.

The results in Table 4 are divided into two categories based on how DPM compares to the other two methods: DPM shows a significant advantage in the first category over the other methods. For entry point (AS), DPM achieves 79.35%, outperforming Petho et al. and Dijkstra. In the second category, DPM outperforms the other methods by a small margin. For entry point (CSMS), DPM achieves 67.97%, compared to 62.76% of Petho et al. and Dijkstra 48.16%. However, in one case, DPM's performance is worse than that of the other two methods. Only for entry point (CS2) we see a small advantage when applying the alternative methods. DPM achieves 59.85%, slightly lower than Petho et al. and Dijkstra (62.36% each).

In Figure 3, we present the attacker path starting from entry point CS3, highlighting the riskiest attack path and the hosts in that path.

We now proceed to investigate how the risk reduction increases with the number of patches increasing. We select 3 cases where DPM outperforms the baseline methods with a significant risk reduction, one where DPM is slightly better and one where is slightly worse.

Entry	DPM	Petho et al.	Dijkstra
CS3	**80.00%**	74.16%	74.16%
AS	**79.35%**	46.71%	46.69%
PM1	**74.38%**	59.80%	59.78%
LMS	**65.98%**	46.58%	46.77%
PM2	**60.86%**	46.77%	46.77%
CSMS	**67.97%**	62.76%	48.16%
EV2	**67.97%**	62.36%	62.36%
CS1	**66.34%**	60.68%	60.68%
CST	**66.34%**	60.68%	60.68%
CSC	**65.84%**	64.52%	64.52%
PM3	**65.67%**	60.15%	60.15%
CS2	59.85%	**62.36%**	**62.36%**

Table 4: Evaluation of DPM, Petho et al., and Dijkstra for different entry points

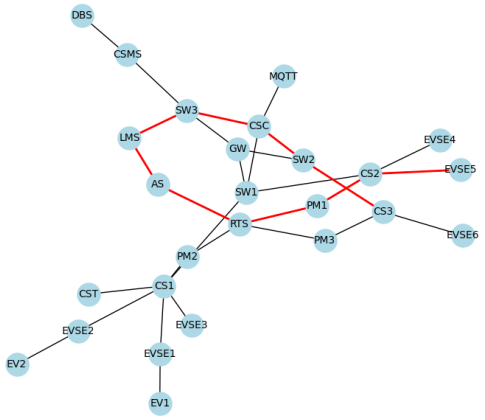

Figure 3: An example highlighting the riskiest attack path starting from entry point CS3.

Figure 4 presents the relative risk reduction for entry point (AS) across three methods: DPM, Petho et al., and Dijkstra. As the number of patches increases, DPM demonstrates the most significant improvement in risk reduction, reaching a peak of 79.35%. Petho et al. and Dijkstra show a more gradual increase, with their maximum risk reduction at 46.71% and 46.69%. DPM outperforms the other two methods because it applies patches incrementally, aiming to maximize risk reduction with every step, resulting in a higher number of patches applied, compared to the other two methods, which identify fewer vulnerabilities to patch along the identified riskiest path.

Figure 5 illustrates the results for entry point EV2. DPM achieves the highest reduction, peaking at 67.9% after 29 patches. Petho et al. reaches 62.3% after 26 patches, and Dijkstra also reaches 62.3%, but

with lower effectiveness than DPM and Petho et al. Overall, DPM slightly outperforms the other methods.

The findings for entry point CS2 are shown in Figure 6. DPM shows a significant increase in risk reduction, peaking at approximately 59.8% after 16 patches. Petho et al. reach 62.3% after 23 patches, while Dijkstra also approaches 62.3%, but with a slightly slower rate of increase.

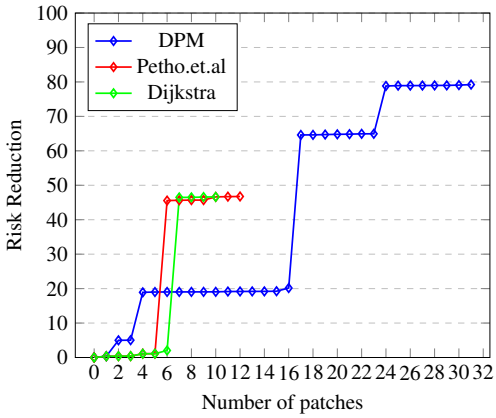

Figure 4: Relative Risk Reduction for entry point AS

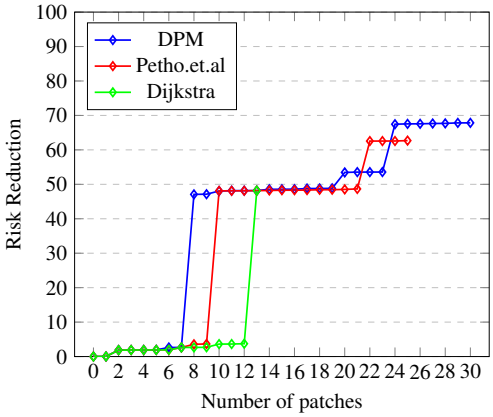

Figure 5: Relative Risk Reduction for entry point CSMS

5 Related Work

In this section, we examine previous works on vulnerability prioritization.

5.1 Topology-aware prioritization

[Petho et al., 2021] analyzed vulnerability levels in in-vehicle communication networks using undirected graph representations. Edges represent connections between components, with protection levels modeled as resistance-like parameters. Security is evaluated by summing the shortest paths between nodes, determined via a generalized Dijkstra's algorithm. Unlike DPM, their approach initializes the starting node with zero weight and iteratively updates cumulative edge weights (protection costs) to identify minimal-cost paths. The study offers a structured method for quantifying network security based on component protection and interconnections. However, while their analysis includes various topologies, EVCI networks are considerably larger and involve more entities and interdependencies compared to single-vehicle network sizes they demonstrated on.

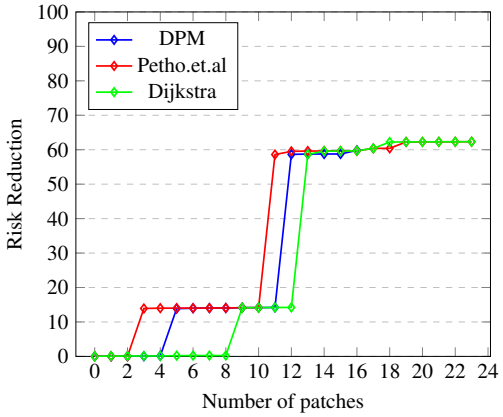

Figure 6: Relative Risk Reduction for entry point CS2

[Alperin et al., 2019] presented a data-driven approach for a vulnerability assessment system that relies on the feature extraction from the information provided in the description of both CVEs and exploits. Their approach customizes vulnerability prioritization by considering the likelihood that specific threats will target a network, using historical threat intelligence to make these predictions. This approach to path prioritisation can in principle be combined with our topology-aware approach and the selection of the instance to patch first. However, if new exploits appear, it is not clear how/when their re-evaluation of the overall threat risk should be initiated.

IoT-PEN [Yadav et al., 2020], a penetration testing framework for IoT systems that follows a client-server architecture in point, star, and mesh topology, was introduced to discover the possible ways an attacker can breach the target system using graphs. IoT-PEN focuses on detecting multi-host and multi-stage attacks and recommending patch priorities on CVE severity-based critical vulnerabilities. In contrast, our approach focuses on minimizing system risk by evaluating attacker paths, prioritizing patches based on exploitability. While both approaches aim to enhance security, our method addresses vulnerabilities in the

most critical attack paths, based on their exploitability.

5.2 Non-graph based prioritization

[Farris et al., 2018] proposed a vulnerability management strategy named VULCON based on performance metrics such as time-to-vulnerability remediation and total vulnerability exposure. Their optimized approach reduces long-term risk on the network, even if it doesn't fix as many vulnerabilities overall, focusing on the most dangerous ones rather than addressing the highest number. Similarly, [Jung et al., 2022] present a context-aware vulnerability prioritization model that calculates temporal-enabled vulnerability scores of CVEs and visualizes them. However, their work generally requires expert knowledge and time to validate the detected vulnerabilities. [Wu et al., 2022] propose an OS-aware vulnerability prioritization approach that employs differential severity analysis, utilizing techniques such as static program analysis and natural language processing to assess the severity of vulnerabilities for specific Linux and Android systems. Lastly, PatchRank [Yadav and Paul, 2019] considers SCADA system interdependencies and ranks nodes' vulnerabilities using CVSS scores and their potential impact.

Compared to all the above works, we use a topology-aware approach combined with attack-path selection algorithms to simulate attacker behavior. Additionally, we leverage EPSS to rank vulnerabilities based on exploitability, providing a dynamic and up-to-date view. Our method addresses vulnerabilities in distributed systems and demonstrates its effectiveness in EV charging networks, where a single subsystem vulnerability can have cascading effects on connected systems.

[Ma et al., 2024] introduce a new tool named VulNet, which provides priority ranking for the dependencies in software libraries and associated vulnerabilities and overcomes other platforms such as Maven Repository (MVN) and Open Source Insights (OSI). They rank the vulnerabilities based on severity and dependencies based on dependency depth. However, their research is limited only to software library dependencies and not other systems, and they use the CVSS metric score to assign severity. Moreover, they do not discuss any attack-path selection approach.

5.3 Graph-based prioritization

In their work, [Olswang et al., 2022] main contributions are prioritizing vulnerability patches and attack graph visualization to assist in decision-making. Their primary focus is estimating a node's importance within a graph. Their method is based on the number of attacks that pass through a specific vulnerability on a specific device. In contrast, DPM focuses on the number of vulnerabilities in each host and the likelihood of exploiting that particular vulnerability.

[Stergiopoulos et al., 2022] present a method for automatically analyzing complex attack graphs in multi-cloud infrastructures. Their proposed tool prioritizes existing vulnerabilities, explores the effect of system states on the overall network, and suggests which system states, vulnerabilities, and configurations have the most significant overall risk to the ecosystem. They use Edmond's algorithm to create a directed tree from a root node that connects to every node in the graph, modeling potential attack paths. The root represents the attacker's end goal, which differs from DPM that uses the BFS algorithm, and in our case, the roots are entry points. We want to see the riskiest path the attacker can take from various nodes. Moreover, we utilize the EPSS score, which differs from their work, which uses CVSS.

[Pirani et al., 2022] developed an attack graph model for a network's vulnerability and topology that minimizes attack success likelihood with minimum cost. They study connectivity measures of several vehicle platoon topologies and reveal how these measures affect the ability of distributed algorithms to reject communication disturbances, detect cyberattacks, and be resilient against them. However, the authors focus solely on network topologies but do not discuss vulnerabilities. They do not address specific weaknesses in the software, hardware, or protocols that attackers could exploit, which limits their study's application in practical cybersecurity scenarios.

5.4 Attack-path prioritization

[Stellios et al., 2021] propose a target-oriented and source-driven methodology to assess the attack paths against critical assets. By extending CVSS metrics of CVEs, they use vulnerability vectors to assess attack-path scenarios, whereas we use EPSS, which represents updated information regarding the exploitability of the CVEs. [Yang et al., 2023] propose a method for the risk assessment of IoT hosts based on attack graphs. Their approach involves generating attack graphs and quantifying both the atomic attack probability and vulnerability impact value based on multiple vulnerability attributes. The criticality of a host, referred to as its "asset value," is determined through a method, which combines expert scoring with the host's role in the network topology. However, they do not focus on the vulnerability prioritization process.

Table 5: Comparison of Various Vulnerability Assessment Approaches Based on Specific Criteria

Authors	Topology-aware	Distributed System	Dynamic	Attack Path Selection
[Petho et al., 2021]	Yes	Yes (In-vehicle)	No	Yes
[Rencelj Ling and Ekstedt, 2023]	No	Yes (ICS)	No	No
[Yang et al., 2023]	Yes	Yes (IoT)	No	Yes
[Wu et al., 2022]	No	No	No	No
[Yoon et al., 2023]	No	Yes (ICS)	Yes	No
[Jacobs et al., 2023]	No	No	Yes	No
[Farris et al., 2018]	No	No	No	No
[Jung et al., 2022]	No	No	No	No
[Stergiopoulos et al., 2022]	Yes	Yes (Large-scale networks)	No	Yes
[Yadav et al., 2020]	Yes	Yes (IoT)	No	Yes
[Olswang et al., 2022]	Yes	Yes (Enterprise)	No	Yes
[Yadav and Paul, 2019]	Yes	Yes (SCADA)	No	No
[Stellios et al., 2021]	Yes	Yes (IoT)	No	Yes
[Ma et al., 2024]	No	Yes (Maven and OSI libraries)	No	No
Our work (DPM)	Yes	Yes (EV charging)	Yes	Yes

5.5 Summary

We summarise the related work in Table5, in which we consider four selection criteria for vulnerability prioritization: topology-aware, distributed systems, dynamic, and attack-path. The column "Topology-aware" indicates whether other works consider the network topology when prioritizing vulnerabilities. The column "Distributed System" indicates whether the analysis considers a networked system with interconnected nodes, where vulnerabilities may impact multiple components across the system, or if it assumes an isolated, standalone environment. The underlying reason is that the EV charging infrastructure is distributed, where information is processed in several systems. The column "Dynamic" indicates whether other works employ methods or metrics that dynamically adapt to CVE exploitability, accounting for new exploits and evolving threat landscapes, thereby providing a real-time and context-aware analysis of vulnerabilities. Next, on column "Attack Path Selection" we review whether the related works prioritize vulnerabilities based on possible attack-paths. This work is the only one that meet all the criteria in the analysis.

6 Conclusions

This paper introduces a topology-aware dynamic patch management method to enhance the cybersecurity of Electric Vehicle Charging Infrastructure. Unlike traditional approaches, our method leverages changes in vulnerability exploitability for efficient risk reduction. It improves risk mitigation by identifying the most vulnerable attack paths and prioritizing vulnerabilities based on exploitability and impact. Our empirical study on charging infrastructure vulnerabilities revealed 46 vulnerabilities that need prompt patching to avoid risks to energy infrastructures.

The results show that our method significantly reduces risk across various access locations, outperforming other methods, including closely related algorithms. Its design maximizes risk reduction, especially when prioritization is crucial and patches are limited.

Future work could explore how DPM can be made more efficient for reapplication to a new domain. We can also consider investigating and developing a measurement to quantify the costs of patching different vulnerabilities.

REFERENCES

Alcaraz, C., Cumplido, J., and Trivino, A. (2023). Ocpp in the spotlight: threats and countermeasures for electric vehicle charging infrastructures 4.0. *International Journal of Information Security*.

Alperin, K., Wollaber, A., Ross, D., Trepagnier, P., and Leonard, L. (2019). Risk prioritization by leveraging latent vulnerability features in a contested environment. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*.

Electric, S. (2025). Cybersecurity security notifications. *Schneider Electric Cybersecurity*.

Farris, K. A., Shah, A., Cybenko, G., Ganesan, R., and Jajodia, S. (2018). Vulcon: A system for

vulnerability prioritization, mitigation, and management. *ACM Transactions on Privacy and Security (TOPS)*.

FIRST (2019). *CVSS v3.0 Specification Document*. Forum of Incident Response and Security Teams (FIRST). Accessed: 2024-11-18.

Jacobs, J., Romanosky, S., Suciu, O., Edwards, B., and Sarabi, A. (2023). Enhancing vulnerability prioritization: Data-driven exploit predictions with community-driven insights. In *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*.

Jung, B., Li, Y., and Bechor, T. (2022). Cavp: A context-aware vulnerability prioritization model. *Computers & Security*.

Kern, D., Krauß, C., and Hollick, M. (2024). Attack analysis and detection for the combined electric vehicle charging and power grid domains. In *Proceedings of the 19th International Conference on Availability, Reliability and Security*.

Ma, Z., Mondal, S., Chen, T.-H., Zhang, H., and Hassan, A. E. (2024). Vulnet: Towards improving vulnerability management in the maven ecosystem. *Empirical Software Engineering*.

Olswang, A., Gonda, T., Puzis, R., Shani, G., Shapira, B., and Tractinsky, N. (2022). Prioritizing vulnerability patches in large networks. *Expert Systems with Applications*.

Petho, Z., Khan, I., and Torok, Á. (2021). Analysis of security vulnerability levels of in-vehicle network topologies applying graph representations. *Journal of Electronic Testing*.

Pirani, M., Baldi, S., and Johansson, K. H. (2022). Impact of network topology on the resilience of vehicle platoons. *IEEE Transactions on Intelligent Transportation Systems*.

Rencelj Ling, E. and Ekstedt, M. (2023). Estimating time-to-compromise for industrial control system attack techniques through vulnerability data. *SN Computer Science*.

Sarieddine, K., Sayed, M. A., Torabi, S., Attallah, R., Jafarigiv, D., Assi, C., and Debbabi, M. (2024). Uncovering covert attacks on ev charging infrastructure: How ocpp backend vulnerabilities could compromise your system. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*.

SecurityWeek (2025). Ics patch tuesday advisories published by siemens, schneider electric, aveva, cisa. *SecurityWeek*.

Stellios, I., Kotzanikolaou, P., and Grigoriadis, C. (2021). Assessing iot enabled cyber-physical attack paths against critical systems. *Computers & Security*.

Stergiopoulos, G., Dedousis, P., and Gritzalis, D. (2022). Automatic analysis of attack graphs for risk mitigation and prioritization on large-scale and complex networks in industry 4.0. *International Journal of Information Security*.

Vailoces, G., Keith, A., Almehmadi, A., and El-Khatib, K. (2023). Securing the electric vehicle charging infrastructure: An in-depth analysis of vulnerabilities and countermeasures. In *Proceedings of the Int'l ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications*.

Wu, Q., Xiao, Y., Liao, X., and Lu, K. (2022). {OS-Aware} vulnerability prioritization via differential severity analysis. In *31st USENIX Security Symposium (USENIX Security 22)*.

Yadav, G. and Paul, K. (2019). Patchrank: Ordering updates for scada systems. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*.

Yadav, G., Paul, K., Allakany, A., and Okamura, K. (2020). Iot-pen: An e2e penetration testing framework for iot. *Journal of Information Processing*.

Yang, H., Yuan, H., and Zhang, L. (2023). Risk assessment method of iot host based on attack graph. *Mobile Networks and Applications*.

Yoon, S.-S., Kim, D.-Y., Kim, G.-G., and Euom, I.-C. (2023). Vulnerability assessment framework based on in-the-wild exploitability for prioritizing patch application in control system. In *International Conference on Information Security Applications*.