TOWARDS AUTOMATED CONTEXT-AWARE VULNERABILITY RISK MANAGEMENT

Vida Ahmadi Mehri

Blekinge Institute of Technology
Doctoral Dissertation Series No. 2023:07

Department of Computer Science

Towards Automated Context-aware Vulnerability Risk Management

Vida Ahmadi Mehri

Blekinge Institute of Technology Doctoral Dissertation Series
No 2023:07

Towards Automated Context-aware Vulnerability Risk Management

Vida Ahmadi Mehri

Doctoral Dissertation in Computer Science

Department of Computer Science
Blekinge Institute of Technology
SWEDEN

Department of Computer Science
Publisher: Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden
Printed by Media-Tryck, Lund, Sweden, 2023
ISBN: 978-91-7295-459-5
ISSN: 1653-2090
urn:nbn:se:bth-24468

“The more I learn, the more I realize how much I don’t know.”

Albert Einstein

Abstract

The information security landscape continually evolves with increasing publicly known vulnerabilities (e.g., 25064 new vulnerabilities in 2022). Vulnerabilities play a prominent role in all types of security related attacks, including ransomware and data breaches. Vulnerability Risk Management (VRM) is an essential cyber defense mechanism to eliminate or reduce attack surfaces in information technology. VRM is a continuous procedure of identification, classification, evaluation, and remediation of vulnerabilities. The traditional VRM procedure is time-consuming as classification, evaluation, and remediation require skills and knowledge of specific computer systems, software, network, and security policies. Activities requiring human input slow down the VRM process, increasing the risk of exploiting a vulnerability.

The thesis introduces the Automated Context-aware Vulnerability Risk Management (ACVRM) methodology to improve VRM procedures by automating the entire VRM cycle and reducing the procedure time and experts' intervention. ACVRM focuses on the challenging stages (i.e., classification, evaluation, and remediation) of VRM to support security experts in promptly prioritizing and patching the vulnerabilities.

ACVRM concept is designed and implemented in a test environment for proof of concept. The efficiency of patch prioritization by ACVRM compared against a commercial vulnerability management tool (i.e., Rudder). ACVRM prioritized the vulnerability based on the patch score (i.e., the numeric representation of the vulnerability characteristic and the risk), the historical data, and dependencies. The experiments indicate that ACVRM could rank the vulnerabilities in the organization's context by weighting the criteria used in patch score calculation. The automated patch deployment is implemented with three use cases to investigate the impact of learning from historical events and dependencies on the success rate of the patch and human intervention. Our finding shows that ACVRM reduced the need for human actions, increased the ratio of successfully patched vulnerabilities, and decreased the cycle time of VRM process.

to my dear family and brave Women of Iran
Woman Life Freedom

Preface

This thesis consists of an introductory part (Chapters 1-6), one peer-reviewed journal article, and three peer-reviewed conference papers (Chapters 7-10). I have been the main contributor to all the publications in the thesis. The studies in all papers have been developed and designed under the supervisors' guidance and co-authored with supervisors. The formatting of included papers has been changed to maintain a consistent style throughout the thesis, but the content is unchanged.

Included Papers

Paper I Ahmadi Mehri, V., Arlos, P., and Casalicchio, E. "Normalization of Severity Rating for Automated Context-aware Vulnerability Risk Management", 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C), Washington, DC, USA, 2020, pp. 200-205, ©2020 IEEE. doi: 10.1109/ACSOS-C51401.2020.00056.

Paper II Ahmadi Mehri, V., Arlos, P., and Casalicchio, E. "Normalization Framework for Vulnerability Risk Management in Cloud", 2021 8th International Conference on Future Internet of Things and Cloud (FiCloud), Rome, Italy, 2021, pp. 99-106, ©2021 IEEE. doi: 10.1109/FiCloud49777.2021.00022.

Paper III Ahmadi Mehri, V., Arlos, P., and Casalicchio, E. "Automated Context-Aware Vulnerability Risk Management for Patch Prioritization", Electronics 2022, 11, 3580. https://doi.org/10.3390/electronics11213580

Paper IV Ahmadi Mehri, V., Arlos, P., and Casalicchio, E. "Automated Patch Management: An Empirical Evaluation Study", Accepted in 2023 IEEE International Conference on Cyber Security and Resilience (CSR), Venice, Italy. ©2023 IEEE.

Related Papers

Paper V Tutschku, K. Ahmadi Mehri, V., Carlsson, A., Chivukula, K. V., Christenson, J. "On Resource Description Capabilities of on-board Tools for Resource Management in Cloud Networking and NFV Infrastructures". 2016 IEEE International Conference on Communications Workshops (ICC), Kuala Lumpur, Malaysia, 2016, pp. 442-447, doi: 10.1109/ICCW.2016.7503827.

Paper VI Tutschku, K.Ahmadi Mehri, V., Carlsson, A. "Towards Multilayer Resource Management in Cloud Networking and NFV Infrastructures". In *12th Swedish National Computer Networking Workshop (SNCNW)*, 2016.

Paper VII Ahmadi Mehri, V., Tutschku, K. "Flexible Privacy and High Trust in the Next Generation Internet: The Use Case of Cloud-based Marketplace for AI". In *13th Swedish National Computer Networking Workshop(SNCNW)*, 2017.

Paper VIII Ahmadi Mehri, V., Ilie, D., Tutschku, K. "Privacy and DRM Requirements for Collaborative Development of AI Application". In *13th International Conference on Availability, Reliability and Security, ARES 2018: Workshop On Interdisciplinary Privacy and Trust.*, 2018. https://doi-org.miman.bib.bth.se/10.1145/3230833.3233268

Paper IX Ahmadi Mehri, V., Tutschku, K. "Privacy and Trust in Cloud-Based Marketplaces for AI and Data Resources". In *11th IFIP International Conference on Trust Management(TM)*, Springer International Publishing, 2017.

Paper X Ahmadi Mehri, V., Ilie, D., Tutschku, K. "Towards Privacy Requirements for Collaborative Development of AI Applications". In *14th Swedish National Computer Networking Workshop (SNCNW)*, 2018.

Paper XI KOYYADA, S., Deshmukh, D., Badampudi, D.,Ahmadi Mehri, V., Usman, M. "Towards automated open source assessment–An empirical study.". In *arXiv preprint*, 2022.

Acknowledgements

I want to thank the people who have supported me in my research education and without whom this thesis would not have been possible. First and foremost, I would like to express my deepest and most sincere gratitude to my main supervisor Dr. Emiliano Casalicchio for his invaluable guidance, encouragement, patience, and continuous support throughout my research adventure.
I would like to extend my sincere gratitude to my second supervisor Dr. Patrik Arlos for providing full-time support, sharing deep experiences, inspiring and encouraging me during my research education.
I would like to thank all my friends and colleagues in the department of Computer Science who have helped me through discussions and sharing their knowledge.

I thank *Rudder.io* for allowing me to test their tool in my research. I would like to acknowledge *Celura* for their Cloud infrastructure access.

I would like to thank Denim Deshmukh and Sai Pranav Koyyada for their help in collecting the vulnerability Data from RedHat Security Advisory.

I am thankful to my wonderful family for their never-ending support and motivation. Special thanks to my loving husband Mohammad, who never gave up on me and has been my best friend and the source of inspiration. I am grateful to my children, Sina and Sarina, who always forgive my absence in favor of my job.

Stockholm, May 2023
Vida Ahmadi Mehri

Contents

List of Abbreviation

A	Availability
AC	Attack Complexity
ACVRM	Automated Context-aware Vulnerability Risk Management
AUS	Application/Unit/Service
AV	Attack Vector
C	Confidentiality
CIS	Center of Internet Security
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DSA	Debian Security Advisories
I	Integrity
NVD	Natinal Vulnerability Database
OC	Organization Context
PoC	Proof of Concept
PS	Patch Score
RHSA	RedHat Security Advisory
SC	Severity Score
USN	Ubuntu Security Notice
VD	Vulnerability Database
VMM	Vulnerability Management Mode
VRM	Vulnerability Risk Management

1 Introduction

Over the last few years, digital transformation and remote work accelerated, exposing more information to the public. Protecting data from unauthorized access becomes essential in Information Technology (IT) as traditional network perimeters and firewalls are insufficient. In addition, an increasing number of sophisticated attack tools evolved, which support bad actors for financial gain. Cybercrime causes significant economic losses and reputational damage in IT each year [1]. Hence, information security must advance its technique and method to deal with growing cyber-attacks.

The information security landscape changed quickly in recent years due to the increased number of publicly known vulnerabilities (i.e., 40% from 2019 to 2022), the COVID-19 pandemic (i.e., Since 2019), and Russia's invasion of Ukraine (i.e., since February 2022). Attackers rapidly exploit unpatched vulnerabilities (i.e., weaknesses in an IT system) to steal sensitive data or disrupt the operation of the targeted system [1, 2]. Recently, a global ransomware attack launched by exploiting VMware ESXi servers vulnerability CVE-2021-21974 and targeted thousands of servers in Italy, France, Finland, the United States, and Canada as reported by Reuters[1]. The COVID-19 pandemic increased the speed of digital transformation and changed the IT work model from on-site to remote. The remote work model expands the attack surface (i.e., the entry point of a system or an environment that the attacker can try to enter) by increasing the number of public-facing IT (i.e., intended for access by the general public) services and infrastructures [2]. Russian state actors launched a massive destructive cyberattack against the Ukrainian government, technology, and financial sectors hours before the missiles were launched [2]. Exploiting vulnerabilities in public-facing applications such as VPN (e.g., CVE-2020-4006, CVE-2019-19781, CVE-2019-11510, CVE-2019-

[1] https://www.reuters.com/world/europe/italy-sounds-alarm-large-scale-computer-hacking-attack-2023-02-05/

9670, and CVE-2018-13379) and Microsoft SQL Server (CVE-2021-1636) are common intrusion methods used to initiate attacks by Russian [3, 4]. Hence, Vulnerability Risk Management (VRM) is part of a defense mechanism in information security and a robust method for cybersecurity hygiene.

VRM is a cyclic procedure consisting of four tasks; 1)identification, 2) classification, 3) evaluation, and 4) remediation of the security vulnerabilities in the system and software [5]. Security scanning tools such as OpenVAS [6] and Nessus[7] can detect the vulnerabilities in a system, and patch management tools such as ManageEngine and NinjaOne could remediate the vulnerabilities. However, classification and evaluation require skill and knowledge about the system design and IT policy because, in practice, it is impossible to patch all detected vulnerabilities. A security expert might leave some vulnerabilities unpatched due to limited attack surface or the higher remediation cost than exploitation [8–17]. There is a lack of tools that can classify vulnerabilities based on the exploit implication to an organization's business and initiate efficient patching for the specific organization [18]. Therefore, security experts play a vital role in VRM to protect organizations' assets and remain compliant with local and international regulations. Some regulations include binding operational directive 19-02 [2] and Cyber essentials [3] define an explicit deadline (e.g., 15 days for critical vulnerabilities) for patching the vulnerability in public-facing services. Hence, human involvement introduces a compliance challenge as it increases the time and cost of VRM procedure[8, 12]. The patch deadline is counting from the day the remediation is released. According to [19], the mean time to remediation the vulnerability varies in different industries (e.g., 44 days in healthcare and 92 days in public administration). Since 4282 critical vulnerabilities were just published in 2022, there is a need for improvement in the VRM procedure to reduce human intervention, improve patch times, and prioritize remediation based on the organization's context.

The organization context is a set of data that defines assets, the organization intends to protect and the rules, (e.g., asset inventory lists, asset priority list, and access policy). The organization context is essential in VRM to

[2] https://www.cisa.gov/binding-operational-directive-19-02
[3] https://www.ncsc.gov.uk/cyberessentials/overview

provide efficient patch planning. For instance, a vulnerability affecting an asset in the priority list is critical for the organization and must be patched within a specific time (i.e., the time defined by the organization's information security framework).

1.1 Aim and Objectives

This study aims to answer the main research question: **How can we automate the VRM process to make it time-efficient, context-aware, and with enhanced patch prioritization decisions?** We introduced the Automated Context-aware Vulnerability Risk Management (ACVRM) framework to answer the research question.

- Customise the VRM process for a given organization by learning the organization's assets and policy
- Automate the VRM procedure by applying predefined decision criteria
- Develop learning from past VRM experience for automated error handling

The thesis was implemented in a public cloud to investigate the time and cost savings in the VRM procedure. The cost saving is based on reducing human intervention and VRM process time.

1.2 Contribution

Figure 1.1 maps the contribution of each paper into the VRM stages. The summary of the contribution in the papers briefly describe as follows:

1.2.1 Papers outline

- **Paper I** reviewed the literature on VRM and existing vulnerability management tools. The study discovered the lack of tools and methodology to cover all VRM stages and the lack of consistency in scoring the vulnerabilities in different vulnerability databases. The first ACVRM workflow is proposed to maintain all stages of VRM and evaluate the vulnerability in the context of an organization. It also introduced using

multiple Vulnerability Databases (VD) in the identification stage to widen the knowledge of published vulnerabilities.

- **Paper II** introduced and implemented a normalization framework to normalize obtained severity scores from multiple VDs. It discovered the role of VDs in VRM and highlighted the need for using multiple VDs to generalize the severity score of each vulnerability. The risk and impact of exploiting vulnerabilities are determined to depend on the organization's assets, security requirements, and security policies. Thus, the framework allowed organizations to select the vulnerability management mode. Hence, the study improved the identification and classification stages in VRM.

- **Paper III** designed and implemented a PoC of ACVRM to classify and evaluate vulnerabilities in an organization's context. The literature review and security experts interview are conducted to determine the criteria for selecting the order in patch prioritization. The defined criteria could be weighted based on the organizational context. The patch score (i.e., the score is calculated by weighting the selected criteria in the organization's context for each vulnerability)is a metric to rank the vulnerability in an evaluation stage. The patch prioritization derived from ACVRM was compared with the Rudder vulnerability management tool (CVE-plugin) and showed the efficiency of the proposed solution. This paper improved classification and evaluation stages in VRM by reducing evaluation time and experts' intervention.

- **Paper IV** designed and implemented a PoC of the remediation stage of VRM. It also improved the evaluation stage by adding a feedback loop (i.e., the output of the patch deployment) to the ACVRM process. The benefit of a feedback loop is discovered to improve the success rate of patches based on the lesson learned from historical events. This paper enhanced the evaluation and remediation stages of VRM by decreasing the patch failure and experts' intervention in the process.

1.2.2 Thesis Outline

The remainder of this thesis is structured as follows:

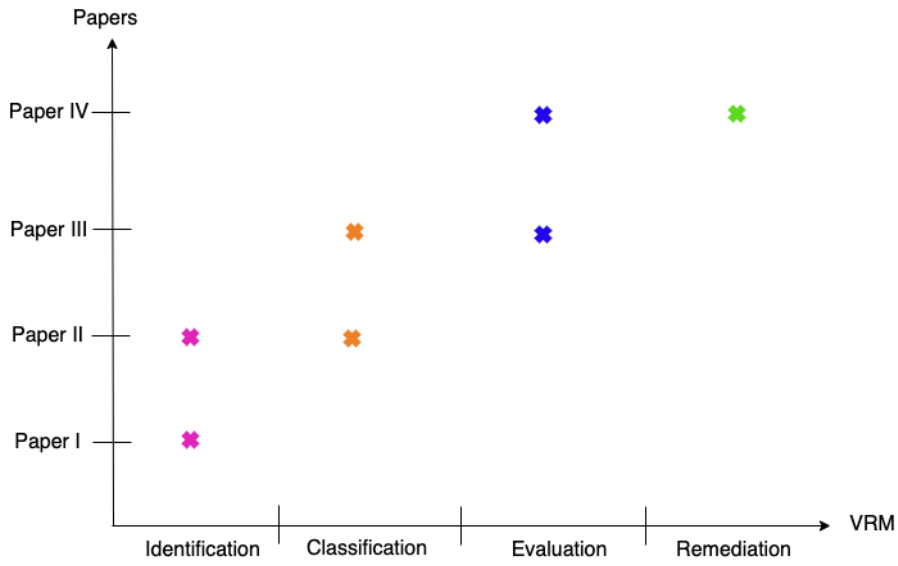

Figure 1.1: *Maps included papers to VRM steps*

- **Chapter 2: Background** provides necessary background knowledge to this study, including Vulnerability Risk Management (VRM), Common Vulnerabilities and Exposures (CVE), Common Vulnerability Scoring System (CVSS), Vulnerability databases, and automation. This section also presents the related work to address the challenge in VRM.
- **Chapter 3: Scientific Approach** presents the research methodology and research questions used in this thesis. In addition, it describes the validation of the results of the thesis.
- **Chapter 4: Automated Context-aware Vulnerability Risk Management** presents our method, ACVRM, and its design methodology.
- **Chapter 5: Summary of Papers** outlines the main contribution to the studies included in the thesis. It also highlights the contribution to the research community by answering the research questions.
- **Chapter 6: Conclusion and future work** concludes the thesis by summarizing contributions and discussing possible future directions.

- **Chapters 7-10** are Papers I-IV included in the thesis.

2

Background

A security vulnerability (hereafter vulnerability) is a weakness, flaw, or error found within a computer system (e.g., Operating System (OS), software, or hardware) that has the potential to be leveraged by threat actors, a person or group of people that intentionally causes harm by exploiting vulnerabilities in the cyber sphere, to compromise a computer system. Typical vulnerabilities are misconfiguration, broken authentication, cross-site scripting, remote code execution, or SQL injection. The result of a successfully exploited vulnerability might cause Denial of Service (DoS), buffer overflow, malware injection, or data breach [20].

Security experts should proactively monitor their environments to identify and mitigate vulnerabilities before threat actors. Security controls and configuration monitoring could remediate misconfiguration, such as the default admin password. However, OS, software, and hardware vulnerabilities require patches released from the vendors. Security experts must also regularly monitor relevant vendors' security advisory websites to be informed about the latest identified vulnerabilities and remediation solutions.

Some vendors (e.g., RedHat, Microsoft, and Google) communicate critical vulnerabilities identified in their products with their VIP customers before the information becomes publicly available on their security advisory websites or VDs. But this is not the case for all vendors. In addition, each vendor might have a different method of ranking the vulnerabilities that resulted in different severity scores for the same vulnerability (e.g., CVE-2020-8130 has a severity score "8.1" in NVD [21], "Medium" in USN [22], "9" in DSA [23], and "Moderate" in RHSA [24]).

With over 25064 new vulnerabilities reported in 2022, it is challenging to remediate all publicly known vulnerabilities with the experts' constraint in IT [9, 13, 15–17, 25]. Hence, security experts must assess vulnerabilities based on the risk of exploitation and their implications on the business. Most organizations assess the vulnerabilities based on the severity score [26, 27] because some regulations have an explicit deadline for patching vulnerabilities with high and critical severity scores. Vulnerability assessment (i.e., evaluation stage in VRM) is time-consuming and labor-intensive, especially when the number of critical vulnerabilities increased 60% in 2022 compared with 2019 (i.e., 4282 vulnerabilities were reported critical in 2022, while 2640 in 2019, 2720 in 2020, and 2677 in 2021) [21]. One vulnerability might be found in hundreds of systems.
There is a need for methodologies to support security experts in making better decisions on vulnerability handling (remediate or not). To help with this, vulnerability risk management standards and frameworks have been introduced in ISO 27002, ISO 62443, and NIST Cybersecurity framework [28–30]. Some vulnerability management tools were developed using the standard framework to address the mentioned challenges. Moreover, some studies develop methods to predict which vulnerability must be remediated based on the organization's resources availability [8, 12, 31, 32]. However, the conducted studies are domain specific with scoping part of the VRM procedure (e.g., identification and remediation) [33–35].

This chapter provides an overview of the concepts and technologies adopted in this work.

2.1 Vulnerability Risk Management (VRM)

VRM is a continuous and proactive process to protect computer systems, networks, and applications from cyberattacks and data breaches. It has been listed as one of the top ten critical security controls by the Center of Internet Security [36]. Furthermore, the information security standards and legislation (e.g., ISO 27002, EU Cybersecurity act [37], USA homeland security act [38]) mandate organizations to establish a VRM procedure to be compliant. The increased number of known vulnerabilities forced the computer security industry towards a risk-based approach. Only vulnerabilities that exceed the organization's acceptable risk level will be remediated in this approach.

Identifying which vulnerability should be remediated first is a challenge that requires a deep understanding of the unique combination of software, hardware, system and network configuration, and the organization's security policy. VRM consists of four stages that are presented in Figure 2.1 and described as follows:

1. **Identification**: the first step in VRM process is to identify the vulnerabilities that may exist in organizations' environments. Vulnerability scanner tools can detect open ports, vulnerable versions of software or hardware, and outdated services. Vulnerability scanners rely on a database of known vulnerabilities and provide reports based on the information in their database. Hence, they are not able to detect unknown or zero-day vulnerabilities. For example, OpenVAS depends on NVD [21] description of vulnerabilities and severity scores in its report. However, relying on one source of information (e.g., one VD) is inefficient in the identification stage based on [39] findings.

2. **Classification**: the discovered vulnerabilities should be categorized into groups to ease prioritization. The classification can be based on the nodes or group of nodes, the type of vulnerabilities, access level (i.e., access through an internal network or external network), or the criticality of the vulnerable service/software to the organization. Vulnerability scanner tools classify the vulnerabilities in their report node-based and score based. However, the external access and criticality of the service or software to the business are essential metrics in the risk assessment. The vulnerabilities in a public-facing (i.e., external access) service have a higher risk of exploitation than those only available via an internal network. Hence, the classification stage needs experts' knowledge of the system and services.

3. **Evaluation**: the classified vulnerabilities must be evaluated based on the organization's risk management strategy. Some vulnerabilities with a low risk of exploitation and limited attack surface (e.g., a vulnerability in Adobe Flash Player while it is entirely disabled from being used in web browsers and other client applications) could remain unpatched if the cost of mitigation is substantially higher than the cost of the vulnerability being exploited. Vulnerability scanners reported the severity score of the vulnerabilities based on the score in their reference VDs. However, the severity score from VDs considers an

out-of-the-box risk rating that evaluates each vulnerability's risk in a silo.
The risk evaluation depends on other factors, such as the likelihood of exploitation, attack complexity, attack vector, and the age of the vulnerability. The evaluation stage decides on a patch prioritization for each node or group of nodes in the organization. The decision might be based on the risk and availability of the resources to patch the vulnerabilities. This time-consuming and labor intense stage is the bottleneck in the VRM procedure as domain experts constrain increase the time to patch for each identified vulnerability [8, 12, 31, 32]

4. **Remediation**: is a corrective action that aims to prevent exploitation by removing or mitigating a threat's capability to exploit a specific vulnerability in the system. The patch method is the system or service dependent. Vendors provide patch instruction, but it needs to be customized for each organization based on their unique system setups and concerning the other vulnerabilities. In addition, the instructions might differ based on the operating system type, server or desktop edition, and the installation method (e.g., software source code, package repository).
The remediation process includes verification of the patch and any side effects it can cause. Patch verification could be done through the vulnerability scanner or version control. However, there is a need for knowledge of organizational context and services to verify the side effect or define test cases. For example, after patching OS on a server that hosts multiple virtual instances, the test could verify the status of each virtual instance. The organization should have a rollback plan if the side effect is detected. The rollback plan must return the system or software to a stable version before the patch, and the expert should execute troubleshooting. If patching the vulnerability cause a breakage in the system and services, the security expert should produce a mitigation plan. Therefore, the remediation stage requires expert intervention for customizing the patch instruction, creating a rollback plan, defining a verification method, troubleshooting in case of an error, and creating a potential mitigation plan.

As described above, stages 2-4 of the VRM require experts' involvement,

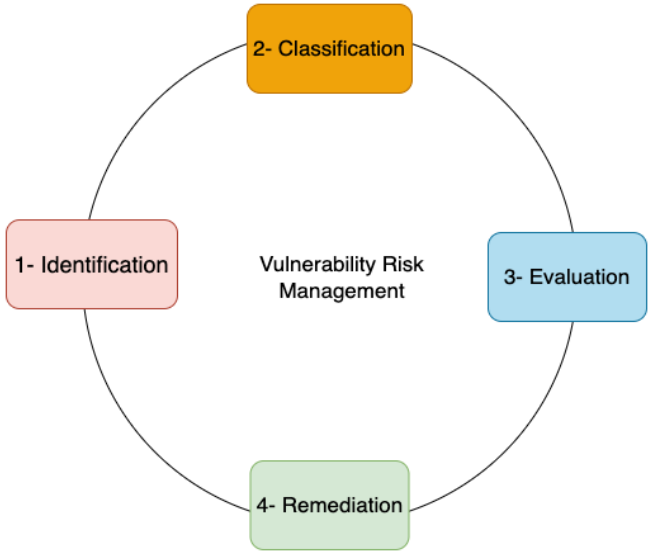

Figure 2.1: *Vulnerability Risk Management procedure*

increasing the VRM process time. Automating stages 2-4 in the VRM procedure could decrease the processing time for each vulnerability. However, efficient automated VRM requires learning organizational contexts such as service policy, inventory of assets, and access policy.

2.1.1 Patch VS. Update

Patch and update are often used interchangeably in system and software contexts. However, there is a big difference in terms of the implementation approach. Software or system updates generally include bug fixes and functional improvement. At the same time, the patch is an update to address a specific vulnerability in the system or software. Some vendors like Microsoft, Adobe, and Oracle regularly release a patch on the second Tuesday of each month (i.e., a patch Tuesday) that addresses some bugs and vulnerabilities with low and medium severity scores in their products and services during their life cycle. However, patches for critical vulnerabilities are released outside the regular patch Tuesday cycle [1]. Regular system update does not necessarily address remediation of the vulnerability and is

[1] https://techcommunity.microsoft.com/t5/windows-it-pro-blog/windows-quality-updates-primer/ba-p/2569385

not intended to eliminate the need for VRM procedure. In addition, some services or applications depend on specific versions of libraries or databases, or other software [40]. Therefore, regular system updates might not be used for a complex system (e.g., IaaS) as the cost of system or software failure is unpredictable in such a system. For instance, updating *software A* may change its functionality or behavior, which causes issues in *software B* that depend on *software A's* functionality. Hence, *software A* should not be updated until *software B* is capable of working with the new version of *software A*.

2.2 Common Vulnerabilities and Exposures

The Common Vulnerabilities and Exposures (CVE) was launched in 1999 and maintained by the US National Cybersecurity Federally Funded Research and Development Center operated by the MITRE Corporation [41]. The primary purpose of CVEs is to set a standard identification reference for publicly known vulnerabilities and exposures. Before CVE, each vendor maintained their identification system and different attributes for each vulnerability, making collaboration and information exchange challenging. CVE ensures the availability of vulnerabilities ID for anyone to use.

The CVE identifier (CVE ID) is a unique number assigned to one software, hardware, or firmware vulnerability. Security professionals use the CVE ID to track information about a particular vulnerability. CVE ID provides a reliable way for vendors, enterprises, academics, and other interested parties to exchange information about security issues. Moreover, CVE ID facilitates sharing across the security industry and helps the mitigation speed.

CVE ID consists of three parts separated with a dash (e.g., CVE-Year-Number). The first part is a CVE prefix. The second part is the year when the vulnerability was reported. A third part is a sequential number assigned by CVE Numbering Authorities (CNA). For example, CVE-2022-31705 is a vulnerability reported in 2022, and CNA assigned number 31705.

The vulnerability is commonly reported by researchers, white hat hackers, and vendors to the CNAs. Some vendors (e.g., Microsoft[2]) also offer bug bounties to encourage the community to seek out the security flaws of their products and services and report them. The CNAs will assign the CVE ID

[2] https://www.microsoft.com/en-us/msrc/bounty

to the vulnerability acknowledged by the vendor as a security flaw affecting only one codebase [42]. The details of CVE are disclosed to the public 45 days after the initial report is confirmed, regardless of the existence or availability of patches or workarounds from affected vendors. The 45 days are the deadline for the corresponding vendor to issue a patch or other fix to ensure protection once the information is made public.

CVE creates an entry for each publicly known vulnerability that includes the CVE ID, a brief description, and a reference to NVD or vendors' security advisory for more information about CVE. CVE is not a vulnerability database and does not contain any information (e.g., risk, impact, severity score, or technical information) needed for VRM. Therefore, it should only be used for reference CVE ID that links information about specific vulnerabilities in different sources.

2.3 Common Vulnerability Scoring System

The Common Vulnerability Scoring System (CVSS) is an open framework maintained by the Forum of Incident Response and Security Teams (FIRST), a US-based nonprofit with over 500 member organizations globally [43]. CVSS provides a method to capture the characteristics and severity of software, hardware, and firmware vulnerabilities to a numeric score from 0 to 10. The numerical score can then be translated into a qualitative representation (e.g., low, medium, high, and critical) to help organizations in their VRM program.

CVSS comprises three metric groups: Base, Temporal, and Environmental, each consisting of metrics, as shown in Table 2.1. The base metrics reflect the severity of a vulnerability based on its characteristics, which are consistent over time across different deployed environments. The temporal metrics adapt the severity of base metrics of a vulnerability according to the criteria that changed over time (e.g., the availability of the exploit code). The environmental metrics adjust the severity of base and temporal metrics for the specific environment (e.g., existing mitigations in the environment). CVSS assigned a value to each metric and translated them to a numeric value to calculate the severity score of each vulnerability. The numeric representation of the base metric is described in Table 9.4. The CVSS vector is a text string that represents the CVSS metrics. The CVSS vector is used

Table 2.1: *CVSS version 3.1 metrics*

Group Metric	Metric Name	Possible Value	Description
Base	Confidentiality (C)	High(H)/Low (L)/None (N)	C measures the impact on the confidentiality of asset due to a successfully exploited vulnerability
	Integrity (I)	High (H)/Low (L)/None (N)	I refers to the trustworthiness and veracity of information.
	Availability (A)	High (H)/Low (L)/None (N)	metric refers to the loss of availability of the impacted component due to successfully exploited vulnerability.
	Attack Vector (AV)	Network (N)/Adjacent (A)/Local (L)/ Physical (P)	AV reflects the access context required for vulnerability exploitation.
	Attack Complexity (AC)	Low (L)/ High (H)	AC refers to conditions that must exist to exploit the vulnerability.
	Privileges Required (PR)	High (H)/Low (L)/None (N)	PR describes the level of privileges an attacker must possess before successfully exploiting the vulnerability.
	User Interaction (UI)	None (N)/ Required (R)	UI captures the requirement for human user participation in the successful compromise of the vulnerability.
	Scope (S)	Unchanged (U)/ Changed (C)	S captures whether a vulnerability in one vulnerable component impacts resources in components beyond its security scope.
Temporal	Exploit Code Maturity (E)	Not Defined (X)/High (H)/Functional (F)/Proof-of-Concept (P)/Unproven (U)	E measures the likelihood of exploit based on the current state of exploit techniques, exploit code availability, or active exploitation.
	Remediation Level (RL)	Not Defined (X)/ Unavailable (U)/ Workaround (W)/Temporary Fix (T)/Official Fix (O)	RL refers to the maturity of the remediation for the vulnerability.
	Report Confidence (RC)	Not Defined (X)/Confirmed (C)/Reasonable (R)/Unknown (U)	measures the degree of confidence in the existence of the vulnerability and the credibility of the known technical details.
Environmental	Confidentiality Requirements (CR)	Not Defined (X)/High (H)/Medium (M)/Low (L)	CR customize the CVSS score depending on the importance of the confidentiality of affected asset to an organization.
	Integrity Requirements (IR)	Not Defined (X)/High (H)/Medium (M)/Low (L)	IR customize the CVSS score depending on the importance of the integrity of affected asset to an organization
	Availability Requirements (AR)	Not Defined (X)/High (H)/Medium (M)/Low (L)	AR customize the CVSS score depending on the importance of the availability of affected asset to an organization
	Modified Base Metrics	same values as the corresponding Base Metric	It enables the analyst to override individual Base metrics based on specific characteristics of environments.

to record or transfer CVSS metric information in a concise form. The colon is used between the abbreviated metric name and its value, and a forward slash is a delimiter between each metric. The CVSS vector string begins with the CVSS label and version (e.g., CVSS:3.1 refers to CVSS version 3.1). The metric name and value are slightly different in the major version update (i.e., The privileges Required metric is called Authentication in CVSS version 2, and its possible values are Multiple, Single, and None). For example, the CVSS vector CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:U/C:L/I:L/A:N/E:F/RL:X shows a CVSS version 3.1 is used to generate a CVSS score. It is also representing each metric with its value. For instance, an example vector above depicting a vulnerability with metrics value of *Attack Vector: Network, Attack Complexity: Low, Privileges Required: High, User Interaction: None, Scope: Unchanged, Confidentiality: Low, Integrity: Low, Availability: None, Exploit Code Maturity: Functional, Remediation Level: Not Defined*. Table 2.1 presents the metrics' name and their abbreviation, possible value and abbreviation, and a brief description of each metric.

The CVSS score reported by vendors or third parties is produced from base metrics since they do not change over time and are environment independent. However, each organization should supplement the Base Score with Temporal and Environmental Scores to produce an accurate severity score of a vulnerability for their organization's environments. Moreover, the CVSS score is not the risk score and should consider as an input into the VRM process. But to rank a vulnerability and make an informed remediation decision, other criteria should be considered, including the financial implication of the exploit, the number of vulnerable systems or services, and the criticality of the vulnerable asset for the organization's business.

2.4 Vulnerability Database

A Vulnerability Database (VD) is a platform that collects, maintains, and disseminates information about publicly known vulnerabilities. VDs gather the generated data from different sources, such as vendors, vulnerability researchers, and other interested parties for each CVE ID. It is an essential tool in VRM procedure to track vulnerabilities. VDs support the organization to address the vulnerability promptly before the exploitation and during cybersecurity incidents [44]. Organizations with a specific area of interest

usually host the VDs. For example, national VDs are general-purpose VDs that aim to support the computer security community national wide (e.g., US National Vulnerability Database (NVD) [45], Chinese National Vulnerability Database (CNNVD) [46]). Vendor VDs are subject-specific for tracking vulnerabilities that affect vendors' specific products and services (e.g., Ubuntu Security Notice (USN) [22], RedHat Security Advisory (RHSA) [24]).

National VDs create an entry for each published CVE ID and collect and analyze the relevant information about CVE ID, such as published date, patch instruction, impact metrics, technical assessment, link to other sources, and severity score. Vendor VDs create an entry for each publicly known CVE ID related to their specific scope. Vendors usually use their internal naming system in VDs and are responsible for providing the patch instructions, which might differ based on the products and services. Vendors also apply their method for severity rating and reporting in VDs.

Some VDs use the CVSS framework to rate the vulnerability's severity, and some are developing their rating system. The VD might report severity ratings in numeric or qualitative scores or both (e.g., NVD reports numeric and qualitative scores, and USN reports only qualitative scores). The severity score might differ for a CVE ID in various VDs due to different rating methods. In addition, vulnerability scanner tools use the VD's severity score in their identification report, and reference VD could not be customized for the organization. Therefore, security experts must map the severity score in the identification stage to the relevant VDs.

The severity score is one of the vital criteria in the classification and evaluation stages of VRM. It also impacts the remediation decision. Therefore, selecting proper VD as a reference is essential in the VRM procedure. The thesis investigates the role of VD in VRM procedure and shows the need to use at least one general-purpose and one subject-specific VDs in VRM procedure.

2.5 Automation

In general, automation uses technology to minimize human intervention less frequently, aiming for productivity, consistency, and efficiency in the

computing system. With a large number of new vulnerabilities (i.e., an average of 68 per day in 2022), automation is the only way to arm security experts to deal with it. VRM naturally is a subjective, manual process, time-consuming and error-prone which relies on individual knowledge and experience [47–49]. Automation in the concept of VRM could reduce the procedure time, time to patch, and expert involvement.

Some studies and tools developed automation solutions in some stages of the VRM, but they can only be used separately in each stage to support security experts[48]. Aota et al. in [50] introduced automation to classify vulnerabilities to address scalability issues in the VRM process for software vulnerabilities due to the shortage of security experts. They use a machine learning scheme to identify the vulnerability type based on the description's information. They achieved 96.92% classification accuracy with their proposed automation in the classification stage of VRM.

Dempsey et al. in [47] provided an operational approach for automating the assessment for software vulnerability management to reduce the assessment time. The capability aims to help the organization to detect vulnerable software in an IT environment faster. The study provides a framework to automate the identification stage of VRM.

Nora et al. in [49] identified the lack of standard tools and knowledge about automating the whole VRM procedure as a cultural challenge in organizations. They also determined transferring security expert knowledge into the automated process is essential for efficient VRM.

Devaux et al. in [51] proposed automation to integrate risk and vulnerability management. Their method addressed the identification stage of the VRM and offered a re-assessment of the Vulnerabilities based on the CVSS score.

ACVRM, unlike mentioned studies, proposed an automation approach for the whole VRM procedure that brings organizational context and knowledge into the automated process. ACVRM reduces the time to patch by automating the entire procedure, decreases human intervention by transferring the knowledge into an automated process, improves identification by automatically collecting vulnerability information from multiple VDs, and adapts organization context by learning security policy automatically.

3

Scientific Approach

3.1 Research Methodology

This thesis aims to answer the main research question; How can we automate the VRM process to make it time-efficient, context-aware, and with enhanced patch prioritization decisions? We approach this by Automated Context-aware Vulnerability Risk Management (ACVRM) methodology. ACVRM is designed and implemented in a test environment for proof of concept. The research domain, cybersecurity, is multidisciplinary of social science, computer science, and information technology. Therefore, the research method adapted from other scientific disciplines [52]. The four general research methods that are widely used in cybersecurity research are summarised as follows:

- **Observational research method** is a way to understand the behavior of a real cyber system. For instance, the research study on adversary behavior in a system under attack to understand the target and tactics the advisory used (e.g., intrusion study). It includes; i)*exploratory studies* that collect and analyze known system design to determine a general theory of behavior, ii)*descriptive study*, which focuses on the specific subset of the system in detail, and iii)*machine learning* that uses applied mathematical techniques to detect correlations in large volumes of data.

- **Theoretical research method** is a logical exploration of a system that investigates the condition and state of the cyber system. For instance, theoretical research could examine cyber system reactions under specific conditions. This type of research is valuable to generate theories about the system's behavior under study, while experimental

validation is impossible due to the complexity or cost. Theoretical research consists of *formal theory*, using mathematics or other logical languages to model and define the possible behavior of the system, and *simulation*, exploring the complex system to understand the theoretical model with enough confidence.

- **Experimental research method** is a type of research where hypotheses are defined for a study and experiments are carried out to obtain evidence. Experimental research could be qualitative or qualitative, depending on the hypothesis objectives. If the researchers have complete control over the experimental setups and configurations, the method is called *hypothetico-deductive*; otherwise, it is called *quasi-experiment*. The second approach required a validity threat analysis to determine potential sources of error.
- **Applied research method** is a method to process the effectiveness of research solutions. It includes designing, implementing, and testing the proposed solution to understand the system's performance. This method could be used in combination with the above-mentioned methods. For instance, *applied experimentation* is used to compare the performance of different solutions, and *applied observational study* studies the system behavior in different situations or scenarios.

Cybersecurity science is a relatively new discipline and not a well-established science domain. Most of the research in this domain is qualitative due to the lack of a method to measure the quantity of all security attributes (e.g., confidentiality, availability, integrity) [52, 53]. The thesis research method is adapted from Experimental research and applied experimentation methods in Cybersecurity science. The conducted studies used both qualitative and quantitative research whenever possible. The methodological approach of this study is described in the next section.

3.1.1 Methodological Approach

The method has been used in the thesis depicted in Figure 3.1. The methodological approach is defined based on a continuous improvement approach where the outcome of each paper is assessed and adjusted based on the obtained result. The thesis proposed ACVRM aiming to improve the existing VRM procedure. The research in each included paper started by studying

each stage of VRM to identify the current gap or challenge. The research question is formulated to define the objective of each study. Next, a technical solution is proposed to address the research question. Then, the experiment is designed to verify the efficiency of the proposed solution in each paper. The design was according to state-of-art tools and literature and also experts interview. Later, the experiment was implemented for collecting data and PoC. The result of each implementation is evaluated in comparison with the existing tools and processes. Therefore, some of the results in included papers were quantitative and some qualitative.

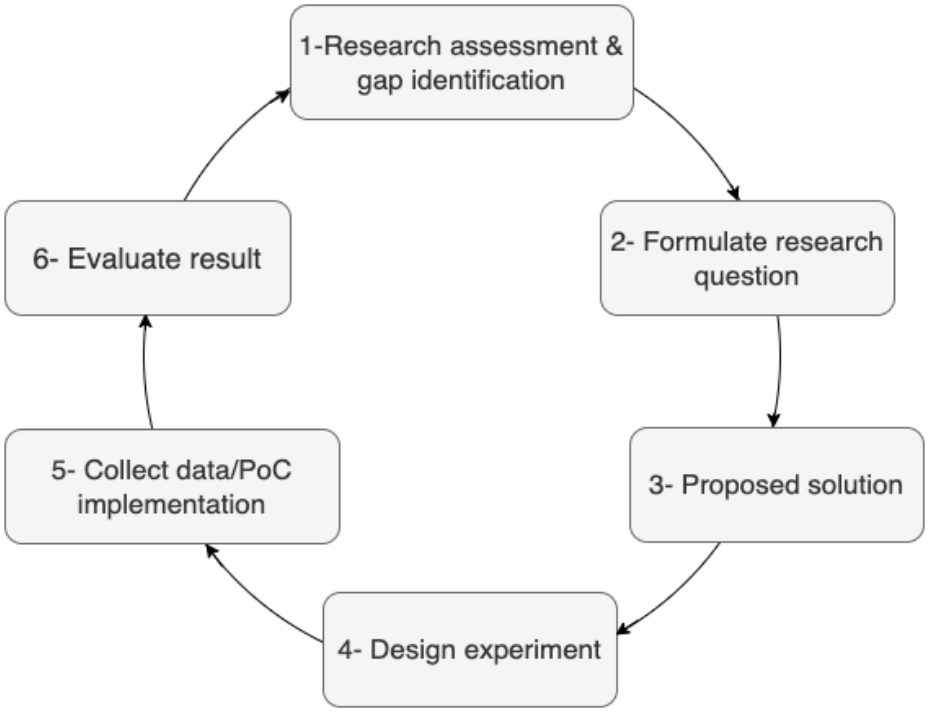

Figure 3.1: *Thesis research method and process*

3.2 Research Questions

The research work summarized in the thesis is addressed by the papers outlined in the upcoming chapters. The following research questions are derived from the main research question in 1.1.

- **Question 1:** What are the challenges in VRM? and how to improve VRM to address the challenges in the organization context?
  Answering this question helped us to identify the gap in a research topic and formulate a research problem. **Paper I**, in Chapter 7, highlighted a general view of the complex challenges that IT is facing, including the increasing number of known vulnerabilities, patch deadlines, VRM labor-intensive tasks, and the limitation of supported tools. **Paper I** also introduced the first vision of ACVRM, providing evidence that VRM could be automated based on the organization context and be classified as a self-protecting system approach.

- **Question 2:** What is the role of a vulnerability database in the VRM process? Which criteria should be considered in selecting reference VD in VRM? How to reflect on the different severity scores obtained from various VDs?
  These research questions are addressed in **Paper II**, Chapter 8, by describing the role of VD in the VRM procedure. It highlighted that the VDs are the source of providing severity scores for each CVE ID. It also investigates the impact of selected reference VD by comparing the severity score obtained from different VDs. A framework to normalize severity scores was introduced to reflect the lack of a unified way of calculating the severity score by VDs.

- **Question 3:** How can the VRM process be time-efficient, cost-effective, and organization oriented?
  Paper III, Chapter 9, answers this research question by automating the classification and evaluation stages of VRM into the process and workflow described in ACVRM Phase 2. It defines criteria for patch score calculation, a metric for automated ranking vulnerabilities. The criteria could be weighted based on the organization's context. **Paper III** adapts organizational context by learning security policy automatically. It also transfers the knowledge into an automated process to reduce the processing time. The PoC implementation is used to evaluate the time and cost efficiency of the proposed solution. **Paper III** also validates the patch prioritization of ACVRM against the prioritization obtained by Rudder, a vulnerability management tool.

- **Question 4:** How to design patch management to increase the patch

success rate and reduce expert intervention in the process?
To answer this question, **Paper IV**, Chapter 10, determines the criteria that positively impact the patch's success rate, such as dependencies and feedback. Automated patch management is designed and implemented with respect to the discovered criteria in **Paper IV** to address this research question. It also introduces initial ideas about learning from historical patch events and patch feedback loops. The learning facilitates automated error handling during the process thus reducing human intervention.

3.3 Validation Method

The validation method is adapted from [54] in this thesis. The overview of construct validity, internal validity, external validity, and reliability related to this work is summarised in this section.

Construct validity

It is referred to determine if the measurements are according to the research questions. Construct validity reflects that inappropriate definitions of the variables to be measured might impact the experimental results [54]. In **paper I-IV**, the variables and the tools have a clear definition in their field (e.g., severity score, patch priority).

Internal validity

Internal validity is vital in a quantitative study, where the measurement concerns the casual relationship. This stipulates that the dependent variables are not affected by confounding variables [54]. In this work, **paper II** collected the severity score from three VDs and compared the VDs scores. The study focuses on the score's distribution and normalization's impact. To remove the impact from other factors, the data has been collected for all CVE IDs in the study during the same period from the selected VDs.

External validity

External validity refers to the generalization of findings and to what extent the result would be applied in other cases [54]. ACVRM design is generalized for VRM procedure in any organization regardless of the domain, focusing

on the vulnerability published in CVE. However, the implementation in **Paper III** and **Paper IV** were for a test organization that utilized its services in a public cloud. **Paper IV** investigated the patch management for the vulnerabilities of the software packages installed on Ubuntu 18.04 LTS (i.e., close to its End Of Life (EOL 2023-04-30) is chosen in the experiment as it likely has more vulnerability to be patched compared with a newer version) from Ubunutu's package repository. Therefore, the obtained results are too specific for the selected use cases.

Reliability

Reliability addresses the consistency of the result, which considers the result of each test should not be biased and dependant on the specific researchers [54]. In other words, the result should be repeatable regardless of who conducts the study. **Paper I-IV** provide the details setup of the research and experiment. In the **Paper II**, Python script is used to process the data, executed multiple times to ensure the result is identical. In the **Paper III**, vulnerabilities are added manually in a controlled manner, and the scripts to calculate the patch score and patch prioritization are executed 50 times for each case. The patch deployment in **Paper IV** was a controlled experiment where the vulnerable version was installed manually on the node. The experiment for each case in **Paper IV** was repeated multiple times to ensure the reliability of the result. Moreover, the reason for describing the details system setup and processes is to address the repetition and reliability of the finding in each paper.

4

Automated Contex-aware Vulnerability Risk Management

We introduce the ACVRM method to answer the main research question highlighted in Section 1.1. ACVRM aims to improve VRM procedures by reducing execution time and cost. ACVRM focuses on automating the entire cycle of VRM to support security experts in their battle against threat actors. It leverages multiple VDs and organization inventory to identify broader concepts' vulnerabilities and severity scores. It allows security experts to use multiple criteria in risk evaluation based on the organization's risk appetite. The design methodology is domain independent and could be used to manage OS, software, and network vulnerabilities.

ACVRM design consists of three phases; Phase 1: retrieval and pre-processing, Phase 2: prioritization, and Phase 3: patch management. The phases are divided conceptually and could be used by any organization regardless of its specific domain. Figure 4.1 indicates the ACVRM phases and their corresponding VRM stages. ACVRM Phase 1 and two steps of Phase 2 map the identification stage of VRM. The filter step in ACVRM Phase 2 matches the classification stage of VRM. Evaluation, Sort, and Review prioritization of ACVRM Phase 2 are equivalent to the evaluation stage in VRM. ACVRM Phase 3 matches with the remediation stage of VRM. ACVRM phases are briefly described as follows:

- **Phase 1** is collected data from multiple VDs. Each organization could select relevant VDs as part of the identification stage of VRM. The collected data are pre-processed to build a unified data structure. The preprocess also converts the qualitative severity score from VD to the numeric score. A normalization framework was introduced to aggregate the severity score from multiple VDs and provide a single

severity score for each CVE ID.

- **Phase 2** consists of the processes for complementing the identification, classification, and evaluation stages of VRM. The processes are based on the organizational context. For instance, a vulnerability scanner report and host inventory are used to identify the vulnerability that affected the organization's environment. The organization selects Vulnerability Management Mode (VMM) (e.g., Basic, Standard, and Restrictive) and defines the process's mode. The VMM impacted the Normalized severity score calculation. The filter step created a list of the vulnerabilities affecting the organization's environment and classified them based on the node. Evaluation, Sort, and Review prioritization steps assess each vulnerability's risk in the organization's context and provide a patch prioritization list for each node. ACVRM uses a Patch Score (PS) (i.e., a mathematical approach to calculating the priority of each vulnerability from the evaluation criteria and their weight based on the organizational context) for ranking a vulnerability.
- **Phase 3** consists of patching the selected vulnerabilities. Verifying a patch and its impact on an organization's environment are essential steps in patch management. ACVRM enables the feedback from patch verification to the Review prioritization step. The feedback loop aims to build a learning process based on historical events and improve the patch prioritization decision.

The details of each ACVRM phase and their implementations are described in Chapters 7-10. The experiment results in Paper I-IV show that ACVRM reduces the VRM process time and expert intervention in a test environment. However, data on the VRM process time is not available to be used as a reference to measure the improvement offered by ACVRM quantitatively.

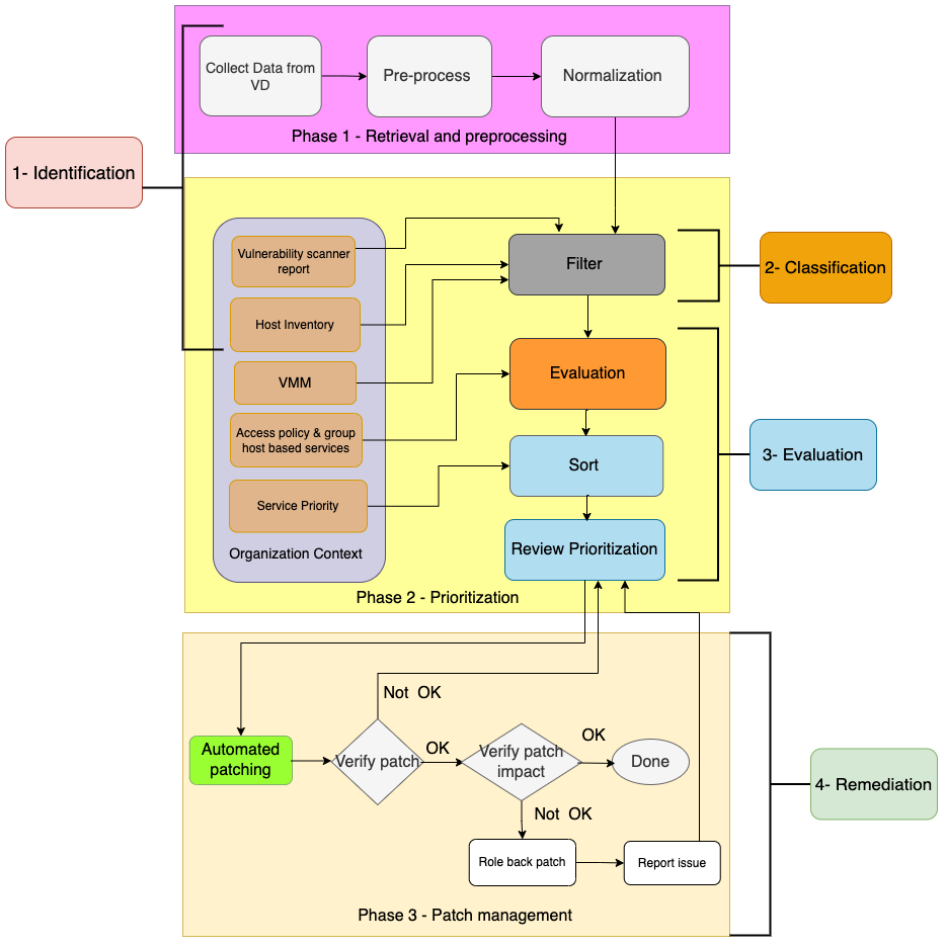

Figure 4.1: *Map ACVRM phases to VRM stages*

5

Summary of Papers

This chapter summarizes the research findings and highlights the main results of included papers in a related subsection below. The relationship between the obtained results in each paper and the overall context of the thesis is explained in this section.

5.1 Normalization of Severity Rating for Automated Context-aware Vulnerability Risk Management

In this study, Paper I, we reviewed the literature and tools to understand the challenges in VRM. We observed the lack of studies on self-vulnerabilities assessment and treatment to remediate a vulnerability based on the organization's preference. Comparing existing tools based on their capabilities to scan networks, identify existing vulnerabilities, analyze the vulnerability, evaluate, and remediate detected vulnerabilities automatically, indicated the absence of a tool that covered all stages of VRM procedure.

The first version of ACVRM was introduced to leverage organization knowledge to support security experts in prioritizing vulnerability patching. One of the identified challenges in the VRM procedure was inconsistency in reporting the severity score of each vulnerability. We observed each VD has its method to calculate severity score and is not necessarily aligned with CVSS metrics. Therefore, a CVE ID could have different severity scores in various vulnerability databases. The drawback of using a single VD in the VRM procedure is discussed in Paper I, and we proposed multiple VDs to have better coverage on known vulnerabilities [55].

A framework to normalize severity scores was introduced to aggregate severity scores from multiple VDs and obtain a single numeric severity score for each CVE ID based on the organization's preference. The normalization

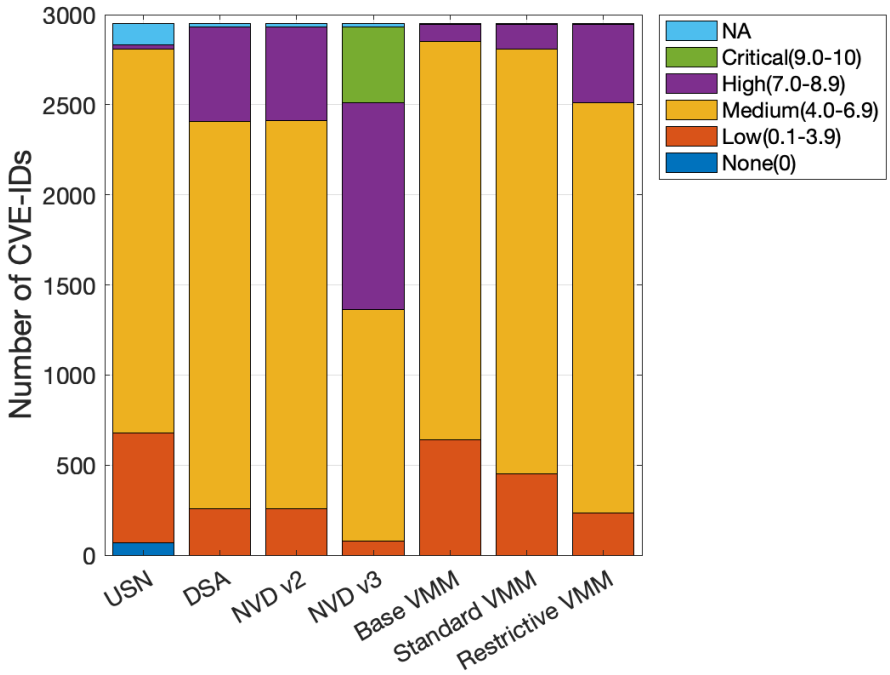

Figure 5.1: *Distribution of the severity score in selected VDs and VDNF in 2019*

framework offers three Vulnerability Management Modes (VMM); basic, standard, and restrictive to convert the qualitative severity score to a numeric value.

5.2 Normalization Framework for Vulnerability Risk Management in Cloud

The proposed VD Normalization Framework (VDNF) is improved in Paper II, where we integrate VDNF into the VRM procedure. VDNF aims to advance the identification, classification, and evaluation stages in VRM while dealing with multiple VDs. This study suggests the selection of VD based on the organization's asset inventory. For instance, organizations with RedHat, Debian, Apache, and Canonical assets should use all relevant VDs as references in the VRM procedure.

In Paper II, we analyzed the distribution of severity scores in the differ-

	Before	After Normalization				
VD	NA	NA	Low	Medium	High	Critical
USN	122	0	10	90	21	1
DSA	21	0	4	17	0	0
NVD	19	0	4	15	0	0

Table 5.1: *The change in severity score of Not Available (NA) 2019 after normalization*

ent VDs and the impact of VDNF on generalizing the severity score of the various VDs. Figure 5.1 from Paper II, presented in this section to highlight our research findings. We observed that the severity score *None* is reported only by USN, possibly because others do not report any vulnerability with a severity score of zero. We also observed that USN has a more significant number of CVE IDs without severity scores (i.e., Not Available (NA)) than other VDs in our experiment. However, the number of *NA* decreased to zero after normalization, regardless of the VMM.

In addition, Table 5.1 shows the number of *NA* severity scores in VDs before and after normalization. Despite data collected from VDs on February 4th, 2021, the VDs do not have information on some CVE IDs registered in 2019. It is due to the VD's procedure and policy to publish the severity score and remediation for CVE IDs which is not transparent. We strongly recommend selecting multiple VDs in the VRM procedure for better coverage of publicly known vulnerabilities.

Based on the result achieved from the experiment in this paper, the positive impact on VDNF is proved in the large sample of CVE IDs. In addition, the role of VD in the VRM procedure is visualized by statistics analyzing the severity score from multiple VDs.

5.3 Automated Context-aware Vulnerability Risk Management for Patch Prioritization

In Paper III, we focus on automating the classification and evaluation stages of VRM in the organization's context, ACVRM Phase 2, to reduce process time and human intervention in patch prioritization. Our approach facilitates learning the organization's policy, service priority, and assets inventory which enables the customization in the classification stage of the VRM procedure. The evaluation criteria (hereafter criteria) must be defined to automate the

evaluation stages of VRM. This paper defines criteria from the literature review and security expert interview. We proposed Patch Score (PS) as a numeric representation of the vulnerability's severity in the organization's context. The PS is calculated based on the score of selected criteria; severity, confidentiality, integrity, availability, attack vector, and attack complexity. The security experts could influence the PS by weighting the criteria. We use the PS score to rank the vulnerabilities in the evaluation stage.
In this study, we design and implement a proof of concept experiment to validate our proposed patch prioritization method in ACVRM Phase 2. The data collected by the method described in Paper I and Paper II are used to identify vulnerabilities in a test environment. The experiment is designed and implemented for four cases with different criteria weights. We installed 24 random CVE IDs in test nodes for a controlled experiment. The result of ACVRM patch prioritization compared with the one obtained from Rudder, a vulnerability management tool, to validate our experiments. We observed that the CVE IDs that could be exploited from networks with a low attack complexity gain higher priority in ACVRM cases than Rudder. We noticed the CVE ID with the same PS value ranked based on age (i.e., the age of vulnerability is calculated based on the date it has been known publicly) by ACVRM, while Rudder randomly generates their position. The result shows the ACVRM improved VRM procedure by reducing the processing time in the classification and evaluation stages with less expert intervention.

5.4 Automated Patch Management: An Empirical Evaluation Study

In this study, Papper IV, ACVRM Phase 3 is proposed to address the challenges in patch management, the last stage of VRM. It provides an overview of the patch management issues and identifies the essential criteria that ensure the successful remediation of vulnerabilities. For instance, we discovered an organization's historical patch data and practical experience are inevitable factors in increasing the success rate of a patch. Some patch failures could be avoided by applying the knowledge from past patch experiences.

This paper challenged the patch prioritization method in Paper III as only the patch score indicates the patch order. It argues the deficiency of prioritization where the dependencies and patch histories are not considered.

Three use cases of automated patch management are studied in Paper VI to investigate the impact of 1) usage of historical patch events, 2) automated review to adjust patch prioritization lists, and 3) dependencies on the success rate of the patch. In the experiment, 21 random vulnerabilities are installed in test nodes. Figure 5.2 shows the result of implementing patch management in a test environment for cases 1-3. The patch prioritization is based on the patch score in Case 1, which assumes historical data (i.e., the outcome of the patch deployment in the past) are unavailable. Seven out of twenty-one (33%) vulnerabilities were patched successfully in case 1, and ten out of twenty-one vulnerabilities (47.6%) required expert intervention due to error.

In case 2, the patch prioritization is automatically reviewed to remove the multiple cumulative patches of software or application from the list. It assumes the dependencies are not available for the vulnerability in the list. In our test nodes, the number of CVE IDs in the patch priority list decreased to 10, a reduction of 11 after review. Six out of ten (60%) vulnerabilities were successfully patched, as shown in Figure 5.2. One out of ten (10%) vulnerabilities escalate to the expert due to the error.

In case 3, the patch prioritization is automatically reviewed to remove unnecessary patches and adjust the list based on dependencies. The number of CVE IDs remains the same as Case 2, but the patch order has been changed due to identified dependencies. Eight out of ten (80%) vulnerabilities were patched successfully, as presented in Figure 5.2. None of the CVE IDs escalates to the expert as no error is reported in patch deployment. Thus, the expert involvement reaches zero in case 3.

This study enhanced the ACVRM Phase 2 by adding patch feedback, automated review, and dependencies. The patch feedback provides the output of the patch deployment in a node, which can be used for error handling. Automated review assesses the patch priority list based on predefined conditions, such as sorting and grouping all vulnerabilities for each software or application, selecting the latest patch release, checking the existing history of the patch, and inspecting the current dependency tree. The result shows that the proposed method improved the patch prioritization list leading to less expert intervention and a higher success rate in automated patch management.

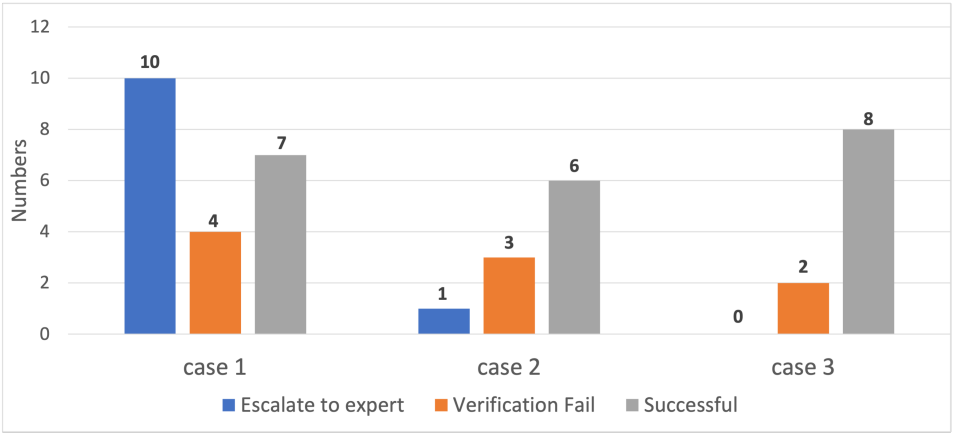

Figure 5.2: *The status of patch management for cases 1-3*

6

Conclusion and Future Work

This thesis proposed a method (ACVRM) to automate organization-oriented VRM procedures to reduce patch and expert intervention time. Decreasing the cost of VRM procedure encourages more organizations to patch the vulnerability proactively and protect their asset from cyber attacks. The main contributions of this thesis are as follows:

1. We proposed and implemented a method to normalize the severity score. The normalization framework facilitates identification and classification stages in VRM when dealing with multiple VDs. We integrated our method into VRM procedure to generalize the severity score from multiple VDs based on the organization's selected Vulnerability Management Mode (VMM). The proposed method is examined on a large set of CVE IDs for PoC. The result indicates the advantage of using multiple VDs based on the organization's assets to cover publicly known vulnerabilities accurately.

2. We learned experts' intervention increases the processing time and the cost of VRM. We proposed a method to automate the classification and evaluation stages of VRM to reduce experts' intervention. We conducted a literature study and interviewed experts to learn which criteria play a role in evaluating vulnerability by experts. We mathematically formulated finding criteria (e.g., security score, attack vector, attack complexity, confidentiality, integrity, availability, and exposure level) to achieve a numeric value (Patch Score) that the automated script could use to create patch orders. There is a possibility to add more criteria to the Patch Score formula if desired. We designed and implemented automated context-aware patch prioritization in a test environment where weighting criteria customized the patch score. The experiment was executed for four cases with differently weighted

criteria. The patch priority list obtained by our method, ACVRM, was compared with Rudder's vulnerability management tool to validate the experiment. The execution time of the classification and evaluation stages of VRM was seven minutes in our test environment without expert intervention (i.e., experts only define a weight vector before starting the process). To generalize the time efficiency of the proposed solution, further study needs to be done. Currently, there is no public information regarding the VRM processing time, and it is difficult to get volunteer organizations to allow us to publish the result.

3. Despite the impact of VRM on compliance and business implications caused by cyber-attacks, many organizations do not patch vulnerabilities proactively for various reasons, including experts' availability, cost, system downtime, and dependencies [56]. We proposed a method to automate patch management to reduce expert intervention while increasing the success rate of the patch. We designed and implemented ACVRM Phase 3, which consists of automated tests, patch deployment, verifying the impact of vulnerability patches on the system functionality, and a feedback loop of the error. The experiment performs in a test environment for three cases with different patch orders. The case studies are developed to investigate the impact of the feedback loop, historical patch data, and automatic review of dependencies. The result shows the feedback loop, dependencies, and historical data reduce the expert intervention in patch management to zero while increasing the success rate. Our method could help more organizations become compliant and secure themselves from bad actors by patching known vulnerabilities promptly.

The thesis presented the design and implementation of our proposed method. The obtained result from studies proves the efficiency of the ACVRM methodology. All experiments were performed in the public cloud and focused on the vulnerabilities publicly known in CVE. The zero-day vulnerability was beyond the scope of this study. We expect our method to be used in any domain, such as IoT, IaaS, and critical infrastructure.

In the future, we could collect vulnerability data from social media and the cybersecurity community blogs and add them to ACVRM Phase 1 to enhance the coverage of our solution. Another future direction could be investigating the time and cost efficiency of ACVRM in different domains and

compared with the recently published state-of-the-art approaches. Another possible future work could be using a machine learning algorithm, such as a decision tree in the Learning module in ACVRM Phase 2, or OpenAI to improve error handling.

References

[1] *Vulnerability and Threat Trends Report 2022*. Tech. rep. SkyBox Security, 2022.

[2] *Microsoft Digital Defense Report 2022*. `https://www.microsoft.com/en-us/security/business/microsoft-digital-defense-report-2022`. [Online; accessed 12-April-2023].

[3] *Understanding and Mitigating Russian State-Sponsored Cyber Threats to U.S. Critical Infrastructure*. `https://www.cisa.gov/news-events/cybersecurity-advisories/aa22-011a`. [Online; accessed 11-April-2023].

[4] *Ukraine: Disk-wiping Attacks Precede Russian Invasion*. `https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/ukraine-wiper-malware-russia`. [Online; accessed 11-April-2023].

[5] P. Foreman. *Vulnerability management*. Auerbach Publications, 2019. ISBN: 9781439801512.

[6] *Open Vulnerability Assessment Scanner(OpenVAS)*. `https://www.openvas.org/`. [Online; accessed 12-Apr-2023].

[7] *Nessus Vulnerability Scanner*. `https://www.tenable.com/products/nessus`. [Online; accessed 12-April-2023].

[8] G. Post and A. Kagan. "Computer security and operating system updates". In: *Information and Software Technology* 45.8 (2003), pp. 461–467.

[9] B. Marx and D. Oosthuizen. "Risk Assessment and Mitigation at the Information Technology Companies". In: *Risk Governance & Control: Financial markets and institutions* 6.02 (2016), pp. 44–51.

[10] J. Dunagan, R. Roussev, B. Daniels, A. Johnson, C. Verbowski, and Y.-M. Wang. "Towards a self-managing software patching process using black-box persistent-state manifests". In: *International Conference on Autonomic Computing, 2004. Proceedings.* IEEE. 2004, pp. 106–113.

[11] C. Tiefenau, M. Häring, K. Krombholz, and E. Von Zezschwitz. "Security, availability, and multiple information sources: Exploring update behavior of system administrators". In: *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. 2020, pp. 239–258.

[12] H. Huang, S. Baset, C. Tang, A. Gupta, K. M. Sudhan, F. Feroze, R. Garg, and S. Ravichandran. "Patch management automation for enterprise cloud". In: *2012 IEEE Network Operations and Management Symposium*. IEEE. 2012, pp. 691–705.

[13] S. Midtrapanon and G. Wills. "Linux patch management: with security assessment features". In: *4th International Conference on Internet of Things, Big Data and Security (IoTBDS)*. 2019.

[14] F. Zhang, P. Huff, K. McClanahan, and Q. Li. "A machine learning-based approach for automated vulnerability remediation analysis". In: *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2020, pp. 1–9.

[15] C.-W. Chang, D.-R. Tsai, and J.-M. Tsai. "A cross-site patch management model and architecture design for large scale heterogeneous environment". In: *Proceedings 39th Annual 2005 International Carnahan Conference on Security Technology*. IEEE. 2005, pp. 41–46.

[16] M. Procházka, D. Kouril, R. Wartel, C. Kanellopoulos, and C. Triantafyllidis. "A Race for Security: Identifying Vulnerabilities on 50 000 Hosts Faster than Attackers". In: *Proceedings of Science (PoS). International Symposium on Grid and Clouds*. 2011.

[17] J.-H. Lee, S.-G. Sohn, B.-H. Chang, and T.-M. Chung. "PKG-VUL: Security Vulnerability Evaluation and Patch Framework for Package-Based Systems". In: *ETRI journal* (2009).

[18] K. Kritikos, K. Magoutis, M. Papoutsakis, and S. Ioannidis. "A survey on vulnerability assessment tools and databases for cloud-based web applications". In: *Array* 3 (2019), p. 100011.

[19] *2022 Vulnerability statistics report*. `https://www.edgescan.com/2022-vulnerability-statistics-report-lp/`. [Online; accessed 12-April-2023].

[20] *Common Weakness Enumeration Specification*. `https://nvd.nist.gov/vuln/categories`. [Online; accessed 12-April-2023].

[21] *NIST National Vulnerability Database search*. `https://nvd.nist.gov/vuln/`. [Online; accessed 10-Jan-2023].

[22] *Ubuntu Security Notice*. `https://usn.ubuntu.com/`. [Online; accessed 12-April-2023].

[23] *Debian Security Tracker*. `https://www.debian.org/security/DSAS`. [Online; accessed 12-April-2023].

[24] *RedHat Security Advisories*. `https://access.redhat.com/security/security-updates/`. [Online; accessed 12-April-2023].

[25] R. Schwarzkopf, M. Schmidt, C. Strack, and B. Freisleben. "Checking running and dormant virtual machines for the necessity of security updates in cloud environments". In: *2011 IEEE Third International Conference on Cloud Computing Technology and Science*. IEEE. 2011, pp. 239–246.

[26] T. Duy Le, M. Ge, P. The Duy, H. Do Hoang, A. Anwar, S. W. Loke, R. Beuran, and Y. Tan. "CVSS based attack analysis using a graphical security model: Review and smart grid case study". In: *Smart Grid and Internet of Things: 4th EAI International Conference, SGIoT 2020, TaiChung, Taiwan, December 5–6, 2020, Proceedings*. Springer. 2021, pp. 116–134.

[27] A. Tripathi and U. K. Singh. "On prioritization of vulnerability categories based on CVSS scores". In: *2011 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*. IEEE. 2011, pp. 692–697.

[28] C. Fruhwirth and T. Mannisto. "Improving CVSS-based vulnerability prioritization and response with context information". In: *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE. 2009.

[29] G. Spanos, A. Sioziou, and L. Angelis. "WIVSS: a new methodology for scoring information systems vulnerabilities". In: *Proceedings of the 17th panhellenic conference on informatics*. ACM. 2013.

[30] U. Gentile and L. Serio. "Survey on international standards and best practices for patch management of complex industrial control systems: the critical infrastructure of particle accelerators case study". In: *International Journal of Critical Computer-Based Systems* 9.1-2 (2019), pp. 115–132.

[31] F. Li, L. Rogers, A. Mathur, N. Malkin, and M. Chetty. "Keepers of the machines: Examining how system administrators manage software updates for multiple machines". In: *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*. 2019, pp. 273–288.

[32] A. Shah, K. A. Farris, R. Ganesan, and S. Jajodia. "Vulnerability selection for remediation: An empirical analysis". In: *The Journal of Defense Modeling and Simulation* 19.1 (2022), pp. 13–22.

[33] M. Walkowski, M. Krakowiak, M. Jaroszewski, J. Oko, and S. Sujecki. "Automatic CVSS-based vulnerability prioritization and response with context information". In: *2021 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE. 2021.

[34] G. Yadav, P. Gauravaram, A. K. Jindal, and K. Paul. "SmartPatch: A patch prioritization framework". In: *Computers in Industry* (2022).

[35] Y. Jiang and Y. Atif. "Towards automatic discovery and assessment of vulnerability severity in cyber–physical systems". In: *Array* (2022).

[36] *CIS Controls*. `http://www.cisecurity.org/controls/`. [Online; accessed 12-April-2023].

[37] *EU Cybersecurity Act*. `https://eur-lex.europa.eu/eli/reg/2019/881/oj`. [Online; accessed 12-April-2023].

[38] *Homland Security Act 2002*. `https://www.dhs.gov/homeland-security-act-2002`. [Online; accessed 12-April-2023].

[39] D. Dey, A. Lahiri, and G. Zhang. "Optimal policies for security patch management". In: *INFORMS Journal on Computing* 27.3 (2015), pp. 462–477.

[40] A. Nappa, R. Johnson, L. Bilge, J. Caballero, and T. Dumitras. "The attack of the clones: A study of the impact of shared code on vulnerability patching". In: *2015 IEEE symposium on security and privacy*. IEEE. 2015, pp. 692–708.

[41] *Common Vulnerabilities and Exposures(CVE)*. `https://cve.mitre.org/`. [Online; accessed 12-April-2023].

[42] *CVE Numbering Authorities*. `https://www.cve.org/ProgramOrganization/CNAs`. [Online; accessed 12-April-2023].

[43] *Common Vulnerability Scoring System(CVSS)*. `https://www.first.org/cvss/`. [Online; accessed 12-April-2023].

[44] V. M. Vilches, L. U. S. Juan, B. Dieber, U. A. Carbajo, and E. Gil-Uriarte. “Introducing the robot vulnerability database (rvd)”. In: *arXiv preprint arXiv:1912.11299* (2019).

[45] H. Booth, D. Rike, and G. Witte. *The National Vulnerability Database (NVD): Overview*. en. 2013. URL: `https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=915172`.

[46] *Chinese National Vulnerability Database*. `https://www.cnvd.org.cn/`. [Online; accessed 12-April-2023].

[47] K. Dempsey, E. Takamura, P. Eavy, and G. Moore. *Automation support for security control assessments: Software vulnerability management*. Tech. rep. National Institute of Standards and Technology, 2020.

[48] H. Tian, L. Huang, Z. Zhou, and Y. Luo. “Arm up administrators: automated vulnerability management”. In: *7th International Symposium on Parallel Architectures, Algorithms and Networks, 2004. Proceedings.* IEEE. 2004, pp. 587–593.

[49] N. Tomas, J. Li, and H. Huang. “An empirical study on culture, automation, measurement, and sharing of devsecops”. In: *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. IEEE. 2019, pp. 1–8.

[50] M. Aota, H. Kanehara, M. Kubo, N. Murata, B. Sun, and T. Takahashi. “Automation of vulnerability classification from its description using machine learning”. In: *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2020, pp. 1–7.

[51] T. Devaux, T. Massip, A. Ulliac, J.-L. Simoni, and P. Varela. “Automation of Risk-Based Vulnerability Management Based on a Cyber Kill Chain Model”. In: *Proceedings of the 28th C&ESAR* (2021), p. 233.

[52] B. Bailey. *Case studies: A security science research methodology*. secau Security Research Centre, Edith Cowan University, Perth, Western Australia, 2011.

[53] T. Edgar and D. Manz. *Research methods for cyber security*. Syngress, 2017. ISBN: 9780128129302.

[54] P. Runeson and M. Höst. “Guidelines for conducting and reporting case study research in software engineering”. In: *Empirical software engineering* 14.2 (2009), p. 131.

[55] V. Ahmadi, P. Arlos, and E. Casalicchio. "Normalization of Severity Rating for Automated Context-aware Vulnerability Risk Management". In: *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. IEEE. 2020, pp. 200–205.

[56] *Costs and Consequences of Gaps in Vulnerability Response*. `https://www.servicenow.com/lpayr/ponemon-vulnerability-survey.html`. [Online; accessed 12-April-2023].

7

Normalization of Severity Rating for Automated Context-aware Vulnerability Risk Managemen

Vida Ahmadi Mehri, Patrik Arlos, Emiliano Casalicchio

Published as:

Ahmadi Mehri, V., Arlos, P., and Casalicchio, E. "Normalization of Severity Rating for Automated Context-aware Vulnerability Risk Management", 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C), Washington, DC, USA, 2020, pp. 200-205, ©2020 IEEE. doi: 10.1109/ACSOS-C51401.2020.00056.

Abstract: In the last three years, the unprecedented increase in discovered vulnerabilities ranked with critical and high severity raise new challenges in Vulnerability Risk Management (VRM). Indeed, identifying, analyzing and remediating this high rate of vulnerabilities is labour intensive, especially for enterprises dealing with complex computing infrastructures such as Infrastructure-as-a-Service providers. Hence there is a demand for new criteria to prioritize vulnerabilities remediation and new automated/autonomic approaches to VRM.

In this paper, we address the above challenge proposing an Automated Context-aware Vulnerability Risk Management (AC-VRM) methodology that aims: to reduce the labour intensive tasks of security experts; to prioritize vulnerability remediation on the basis of the organization context rather than risk severity only. The proposed solution considers multiple

vulnerabilities databases to have a great coverage on known vulnerabilities and to determine the vulnerability rank. After the description of the new VRM methodology, we focus on the problem of obtaining a single vulnerability score by normalization and fusion of ranks obtained from multiple vulnerabilities databases. Our solution is a parametric normalization that accounts for organization needs/specifications.

7.1 Introduction

VRM is a method to reduce the probability of exploitation and severity of damage in a system. Continuous VRM is ranked third of among critical security controls by the Center for Internet Security (CIS) [1]. Today, VRM is facing new complex challenges to remediate vulnerabilities, that are most critical for a specific organization context, on the proper time and sequence. The number of new vulnerabilities have increased enormously during the last three years, from 6,447 in 2016 to 17,306 unique vulnerabilities in 2019 [2]. According to the NIST National Vulnerability Database (NVD) [2], 57.17% of the new vulnerabilities reported in 2019 ranked with critical and high severity (41.82% high severity and 15.35% critical severity). Vulnerabilities with a higher probability of exploit need to be patched in a certain time-window regarding the organization context. For example, the federal agencies in U.S. have to patch the critical vulnerabilities within 15 days and the vulnerability with high severity within 30 days due to binding operational directive [1]. Hence, an autonomic VRM is vital to support decision makers by reducing labour intensive tasks and to address the large number of vulnerabilities within the specific deadlines.

Current VRM methods, used in production environments, prioritize vulnerability patching based on the severity score, thus vulnerabilities with critical and high severity are patched first. D. Dey et al. [3] compares several practical patch policies and concludes that patch policies relying on a single metric such as severity level are not optimal. Indeed, if the severity level of a vulnerability is high but the risk of exploitation is low, it can be patched in a later time. V. Katos et al. [4] reports that 8.65% of known vulnerabilities are exploitable, hence the current VRM practices should be changed to account for the exploitation probability, when prioritize patching.

[1] https://cyber.dhs.gov/bod/19-02/

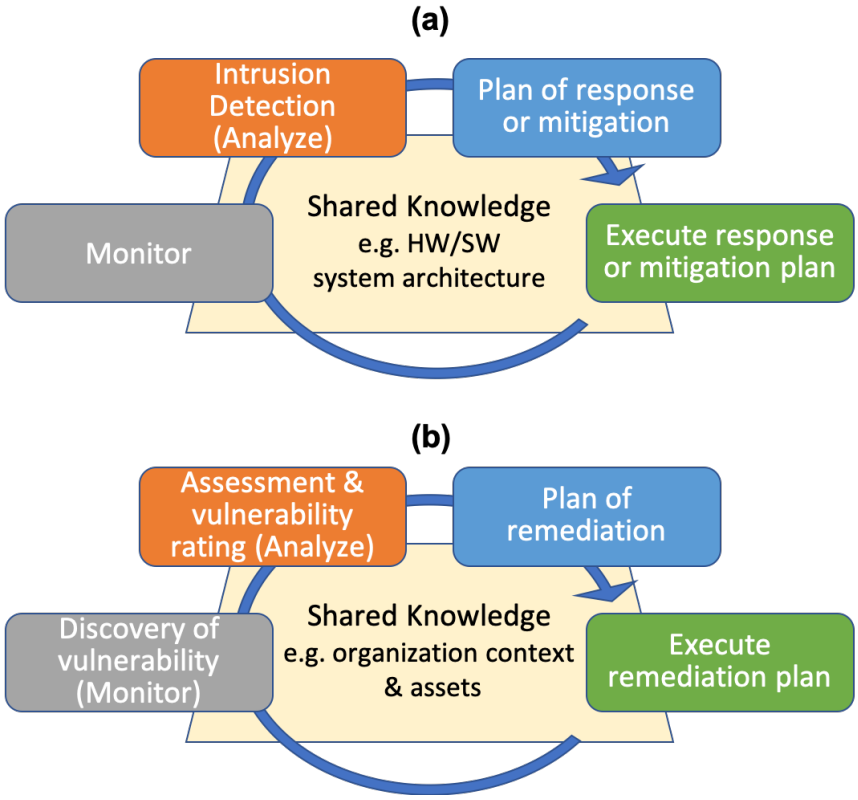

Figure 7.1: *(a) The self-response & mitigation manager; and (b) the self-vulnerability assessment & remediation manager.*

Furthermore, the risk and impact of exploit of vulnerabilities depends on the organization's assets, security requirements and security policies (the organization context hereafter). Hence the vulnerabilities should not be analyzed in an isolated manner, but within the organization context; and also the risk of exploitation should be evaluated in the context of the organization to efficiently plan and to prioritize the vulnerability patching [5].

In this study, we address the aforementioned challenges by proposing AC-VRM. AC-VRM leverages knowledge about organization context to support a security decision-maker in prioritizing vulnerability patching. AC-VRM aims: 1) to reduce the labour intensive tasks of security experts; 2) to prioritize vulnerability remediation on the basis of the organization context rather than risk severity only; 3) it considers multiple vulnerabilities databases

to have a great coverage on known vulnerabilities and to determine the vulnerability rank. AC-VRM is the first step towards the development of a self vulnerabilities assessment and remediation manager. The self assessment and remediation manager is shown in Fig. 7.1(b) and is compared with the self-protecting system manager for response and mitigation of cyber-attacks (widely investigated in literature [6]). A self-protecting system manager is a complex software that autonomously: monitor a system to collect information on its status; analyze the monitored data to detect anomalies (e.g. cyber-attacks or vulnerabilities); plan actions to bring the system back to normal operating conditions (e.g. attack response or mitigation actions, or vulnerabilities remediation actions); execute the planned actions. In this paper, we describe AC-VRM and we propose a normalization of vulnerability severity score based on the chosen vulnerability management mode by organization. That is we focus on the analysis phase in Fig. 7.1(b).

The remaining of this paper is organized as in what follows. Section 7.2 describes related scientific work and vulnerability management tools. The insight to VRM is described in Section 7.3. Our proposed solution for an AC-VRM is presented in Section 7.4. Finally, Section 10.8 concludes the paper.

7.2 Related Work

In this section: firstly, we briefly review the literature about automated VRM and self-protection. Then, we describe the widely used tools for vulnerability management and we compare their features

7.2.1 Literature review

According with the taxonomy proposed in [6] AC-VRM could be classified as a self-protecting system that: 1) offers the planning and prevention self-protection level; 2) it is independent from the system layer to protect; 3) it addresses the confidentiality, integrity, and availability goals; and it focuses on development and deployment time.

Today, most research on self protecting systems mainly proposed solutions for self response and mitigation, e.g. [7, 8]. Only a few work has focused on self-vulnerability assessment and remediation [9].

Most of the literature on automated VRM address the problem of automated penetration testing (e.g. [10] [11]) or other form of automated vulnerability discovery (e.g. [12] [13] [14] [15]). In this work we assume that the organization has its own well tested methodology to discover vulnerabilities. Moreover, in complex production environments, like a Infrastructure-as-a-Service cloud provider, it is almost impossible to apply active vulnerability discovery methods [16].

In [17] the authors propose the Cyber Risk Scoring and Mitigation (CRISM) tool to estimate cyber-attack probabilities. CRISM directly monitors and scores cyber risk based on assets at risk and continuously updated software vulnerabilities. CRISM also produces risk scores that allows organizations to optimally choose mitigation policies that can potentially reduce insurance premiums. The approach used in CRISM is similar to the context-aware assessment proposed in our AC-VRM. However, CRISM uses only one database (NVD) as a source for known vulnerabilities. Moreover, the scoring is determined on the basis of a Byesian attack graph, that in case of a IaaS cloud provider could be extremely complex to generate and analyze.

In [18] the authors propose a framework for automated risk assessment and mitigation, that accounts for the user perceived risk. That proposed framework is tailored to a smart home system and does not take into account the problem of ranking and prioritizing vulnerabilities patching, like AC-VRM does.

In [15] the authors address the problem of zero-day vulnerabilities by proposing a method for risk assessment of zero-day vulnerabilities and attack vectors. Their method builds on a zero-day attack graph, analysis of pre- and post-conditions, and on the attack complexity score, and impact score obtained from the NVD database. As mentioned in the introduction and explained later in the paper, using only a single vulnerability database is a limitation. On the contrary, the limitation of the AC-VRM approach is to not consider explicitly zero-day vulnerabilities.

In [9] the authors propose a predictive machine learning model that can identify exploitable vulnerabilities, and that allows prioritization of patching by leveraging coverage (of vulnerabilities discovered) and efficiency (i.e. patching only what is at high exploitation risk for the organization). The machine learning model is trained on the vulnerabilities extracted from the CVE [19] database, while the NVD database is used to determine severity

score. As we argue in this paper, considering only a single database to extract vulnerability score pose some risks, hence AC-VRM proposes to use different score and to normalize them on the basis of organization preferences (Vulnerability Management Mode, cf. Section IV).

7.2.2 Vulnerability Management Tools (VMT)

VMTs are used to discover vulnerabilities in network, software and hardware. Most of the VMTs are network vulnerability scanners [20–24]. However, some tools, cf. [25], rely on identification of vulnerabilities based on the system documentation. The output of VMT varies, but most generate a report. This report lists the detected vulnerabilities, and prioritizes based on their Common Vulnerability Scoring System (CVSS) score. Some of the more advanced VMTs include suggestions for mitigation, in the report [20–22].

Table 7.1 summarizes features of (some) common VMTs. That features are the following: *scan network*, i.e. the capability to scan the computer network of the organization; *identification*, that is the capability to detect existing vulnerabilities; *analysis*, i.e. the ability to rank the vulnerabilities by security score; *evaluation*, i.e. patching prioritization capacity; and *treatment*, that is the ability of patching the vulnerabilities (automatically or providing remediation plans). Supported features by tools are marked with a ✓, and the □ sign represents missing or limited support for a feature in Table 7.1. From the table it is clear, that none of the tools provide a treatment that would immunize the systems by patching them. Furthermore, from our experience there is a lack of tools that can operate in a context of an organization and cover all aforementioned features.

Table 7.1: *The vulnerability management tools and supported features*

Vulnerability Management Features					
Name	**Scan Network**	**Identification**	**Analysis**	**Evaluation**	**Treatment**
OpenVAS [20]	✓	✓	✓	□	□
Rapid7 Nexpose[22]	✓	✓	✓	✓	□
Vulnerability Manager Plus[24]	✓	✓	✓	□	□
Tripwire IP360[23]	✓	✓	✓	□	□
Qualys Vulnerability Management [21]	✓	✓	✓	□	□
Skybox Vulnerability Management [25]	□	✓	✓	✓	□

7.3 Vulnerability Risk Management (VRM)

VRM is a procedure each organization should practice to maintain proactive cyber defence. Step one is identifying the vulnerability in the organization.

The security team in an organization should regularly execute penetration testing (e.g. nmap) and vulnerability scanning (e.g. OpenVAS) on their systems to detect vulnerabilities and flaw in a system configuration, e.g. open ports and outdated software. The result of such a test is a list of detected vulnerabilities and their severity score, which is calculated by Vulnerability Database (VD). Therefore, VDs are a key components in any VRM method. The second step is analysing and ranking the detected vulnerabilities by using the security score obtained from the VD.

However, to properly and effectively analyze the detected vulnerabilities, we need knowledge about the organization's security policy, exploitation probabilities, and impact of exploit on the system to plan the appropriate response. Considering only CVSS score to prioritize the patching misleads the security decision-maker. For example, a vulnerability with a medium severity score, being actively exploited in the wild, should be marked with a higher priority. As it might pose a greater risk compared to a vulnerability with a critical score that has no known exploit [26]. Hence, the VRM process should be enhanced by adding a third step that prioritizes the patching and resolution of analyzed vulnerabilities based on the organization's context. Analyses and evaluation steps often rely on domain experts, who manually develop the protection mechanisms and define the time and order for patching. The evaluation of domain experts might leave unpatched some of the detected vulnerabilities due to the limited attack vector in the organization context or the high cost of patching compared to the exploit's cost. The aforementioned VRM process is resource intensive and slow due to the number of published and the rate that new vulnerabilities are discovered [27, 28]. Hence, the urgent demand for AC-VRM (or self vulnerability assessment and remediation controller).

The last step in VRM is patching the right vulnerabilities and verification of the procedure. As mentioned, the commonly used criteria in VRMs are the severity score of the vulnerability, CVSS described in the section 7.3.2. The VDs are repositories that record the Common Vulnerabilities and Exposures (CVE) ID and provide a CVSS score for each CVE. Some of the VDs provide the remediation or workarounds to address that vulnerabilities.

7.3.1 Vulnerability Database (VD)

VDs are repositories that contain lists of publicly known vulnerabilities. VDs are usually hosted by organizations with certain regions of interests. For example, RedHat hosts a VD for tracking vulnerabilities affecting their products. However, thanks to CVE [19], each publicly known vulnerability is assigned an Identification Number (ID), the CVE ID. VDs creates entries in their internal system corresponding to the CVE IDs, based on the VDs' naming system, severity rating, and patch instruction. Therefore, the severity score might be different for a CVE ID in different VDs. Some of the commonly used VDs are described in what follows:

i) NIST National Vulnerability Database (NVD) is one of the largest vulnerability databases that contains all published CVEs. NVD operated by NIST as a part of the US Department of Commerce. NVD provides the security-related software flaw, effected products name, impact metrics and CVSS for each CVE [2].

ii) Ubuntu Security Notice (USN) collects all CVEs that affect different releases of Ubuntu. USN has its own naming method that uses four digits after USN as main ID followed by the version (i.e. USN-4295-1). The USN database provides a patch instruction, security score and CVE ID reference or references [29].

iii) RedHat Security Advisories (RHSA) reports the vulnerabilities that affect Redhat releases. Their naming system is the database name followed the year that vulnerability was published and four digits identifying the vulnerability in the database (i.e. RHSA-2020:2404). RHSA database describes each vulnerability, the affected release or releases, patch instruction, severity score, and the reference CVE [30].

iv) Cisco Security Advisories provides information about affected Cisco products by different CVEs. Cisco names each vulnerability by their product name and type of the exploit (i.e. Cisco IOS and IOS XE Software Tcl Denial of Service Vulnerability). Cisco Security Advisories describes each vulnerability and provides patch instruction, severity score, and reference CVE [31].

Table 7.2: *Mapping CVSS score to the qualitative rating in some vulnerability databases [2, 29–31, 33]*

CVSS Score	Qualitative rating
0.0	None
0.1-3.9	Low
4.0-6.9	Medium/Moderate
7.0-8.9	High/Important
9.0-10.0	Critical

7.3.2 Severity Score

The Common Vulnerability Scoring System (CVSS) was created by FIRST.org, Inc. [32] and it aims to transfer the vulnerability characteristics to a numeric score, and help with the quantitative analysis in vulnerability management. CVSS is an open framework to communicate attributes and the severity for each vulnerability. The current version of CVSS is 3.1 and has three metrics; Base, Temporal and Environmental. The base metric consists of attributes that remains unchanged over time, while the temporal metric contains characteristics that change over time. Environmental metrics are attributes which are unique to the environment. All metrics have a score range from 0 to 10. Out of the metrics, the base metric is considered as the primary score, while the other two are optional in the scoring process.
Table 7.2 gives an example of how a CVSS score can be mapped to a qualitative rating used in some VDs. Due to the difference in scoring/ratings among VDs, the same vulnerability can obtain a different score depending on the used VD. Table 7.3 provides an example for CVE-2020-8130 in some VDs which describes in section 7.4.1 in details.

7.4 Proposed Solution

We think a vulnerability should not analyse in isolation. Therefore, organizational knowledge plays a vital role during the analysis, evaluation and treatment steps in a VRM. The VMTs in Table 7.1 provide reports that are based on the severity rating of each detected vulnerability, possible remediation method and reference to the CVE ID. Those reports provide general view of vulnerabilities, and need a domain expert to review and prioritize patching (within the organization). Hence, existing solutions lack the capability to adjust for organizational requirements and automatic patching.

The proposed AC-VRM is aware of the organizational context, and thus

provides more relevant (risk) metrics to the organization. Our solution aims to address the complete VRM method in in an automated manner to minimize the time between vulnerability detection and patching. This will be achieved by more efficient vulnerability detection, evaluation, prioritization and automatic patching.

Fig. 9.2 presents the AC-VRM workflow. The left box labeled *Generic* is related with the process of collection, organization and adaptation of information from multiple VDs. The box labeled *Organization* deals with tasks that are organizational aware, i.e. the self-vulnerability assessment and remediation part of AC-VRM. A short description of AC-VRM workflow follows:

- Collect data: this task collects and updates the information of known vulnerabilities from multiple VDs. This to have as wide knowledge base as possible.
- Pre-process: the information from each VD needs to be adapted into an internal format. Since each VD has its own naming format, we use this step to list the vulnerabilities based on the CVE ID.
- Normalization: this task normalizes the ratings obtained from the used VDs. Depending on VMM setting (basic, standard or restrictive), the normalization from a VD rating to a numeric value can be influenced.
- Vulnerability Management Mode (VMM): is used when translating a VDs qualitative rating to a numeric CVSS score. There are three levels; basic, standard and restrictive. For basic, the lower value in the range is used, standard uses the center value, while restrictive uses the upper range value cf. 7.2.
- Filter: this step is identified the vulnerabilities that effect the organization. The vulnerability will be filtered based on organizational assets.
- Evaluation: vulnerabilities that influences systems in the organization are evaluated according to the organizational knowledge and normalized score (usually done by security experts). The evaluation will generate an initial patch prioritization. AC-VRM uses four priority levels; 0, 1, 2, and 3 which correspond to: ignore vulnerability, immediate patching (within 7 days), patch within 30 days and mitigate with other action.

- Organization knowledge: is the core source of inputs for the workflow. It includes organization system management documents such as information system policies, infrastructure setup, asset inventory and installed software inventory. It could even include reports from VMTs. This information is usually obtained from databases or text files.

- Sort for organization: this task schedule the patching in an order that is as time efficient as possible. It identifies patches for each asset (i.e. infrastructure) and group them together. It could recognize patches that may influence each other, and sequence them in the least disruptive order.

- Patching prioritization: assign due date for each vulnerability in scheduled patching. For example vulnerability X with priority 1 should be patched in asset Y within 7 days. This step sets a patching deadline for each priority, which is the main criteria in automated patching. It also activates the notification alert for each vulnerability in case the patch deadline passed (e.g. the patching fails or is delayed).

- Automated patching: this task is in charge of patching the vulnerable assets in the organization. The asset under patch will be taken out of production.

- Verification: in this phase, the effected assets are verified to be immune against vulnerabilities (priority 1, and 2). This is done using a verification script or a scanner. If the patch failed to address the vulnerability, the patch priority component is notified and the asset will not be back to production.

- Update: this task in the feedback loop allows to learn the new published vulnerabilities that effects organization. Update step is used to revise the patch priority list when the vulnerability with higher impact are detected or new update of listed vulnerability arrived.

As show in Fig. 9.2, the major contribution of our solution is additional steps introduced into a VRM, i.e normalization, evaluation, patch priority and automated patching. This paper focuses on the normalization phase as an important prerequisite for comprehensive scoring of vulnerability when using multiple VDs.

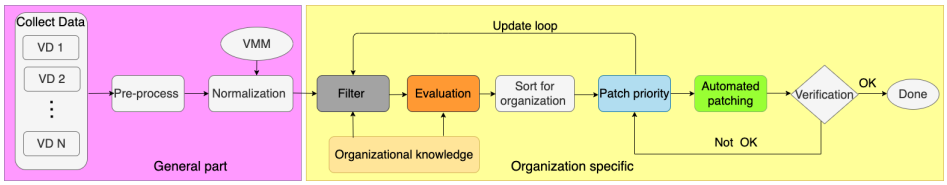

Figure 7.2: *The AC-VRM workflow. The tasks in the Organization specific part are mapped, using the same colour code, on phases of the self-vulnerability assessment and remediation controller presented in Fig. 7.1.*

7.4.1 Normalization

As we mentioned when describing VDs in Sec. 7.3.1, CVE ID is used as a reference by VDs and each VD is updated when new vulnerabilities are published. Furthermore, each VD has its own method to name vulnerabilities, describes the impact and qualitative rating.

To be proactive in protection, we need to collect the updated information from the VDs. This cause some issues, c.f. different CVSS score for the same CVE ID or different scoring method. As an example the CVE-2020-8130, see Table 7.3, scored 8.1 in NVD and medium in USN. The reason for this difference is that USN calculates CVSS qualitative rating of the vulnerability in Ubuntu releases (average of base and environment metrics) while NVD considers CVSS numerical score of the base metrics in its calculation. Therefore, we need to normalize the severity rate of vulnerabilities.

The normalization procedure is provided in Algorithm 2. The *normalizeScore* procedure gets as input the CVE_ID of a vulnerability, the set of VDs used and the value of VMM. Then, for each VD in the set it extracts the severity score for the vulnerability (lines 4 - 7) and finally it returns the average value (line 8). The severity score for a vulnerability given the VMM is computed by the *getSeverityValue* procedure. *getSeverityValue* extracts the severity score for the vulnerability, using *getSecScore*, from VD_i (line 11) and check if it is qualitative or quantitative. In the first case the score is mapped into a numeric value as defined in Table 7.2 and, depending on the value of VMM, is taken the lower bound, the center value or the upper bound of the CVSS score range (lines 12 - 17). The function CVSSmap() is responsible for the mapping and implement also the lower bound, center and upper bound methods. In case the vulnerability score is numeric, it is returned directly without any mapping operated by CVSSmap. For example

Algorithm 1 Normalization Algorithm

```
1: procedure normalizeScore(CVE_ID, VMM, VDset)
2:     /* VDset is the set of VDs considered */
3:     score = 0;
4:     for each VD_i in VDset do
5:         sv = getSeverityValue(CVE_ID, VD_i, VMM);
6:         score += sv;
7:     end for
8:     return ( score = score / VDset.numVDs );
9: end procedure
10:
11: procedure getSeverityValue(CVE_ID, VD_i, VMM)
12:     ss = getSecScore(CVE_ID, VD_i,)
13:     if isString( ss ) then
14:         switch VMM do
15:             case basic: ss = CVSSmap.lower( ss );
16:             case standard: ss = CVSSmap.center( ss );
17:             case restricted: ss = CVSSmap.upper( ss );
18:     end if
19:     return ( ss ); /* in case ss is numeric it is returned directly */
20: end procedure
```

Table 7.3: *Compare CVSS score of CVE-2020-8130 in different VDs [2, 29, 30, 34]*

Database Name	CVSS Score/Qualitative rating
NVD	8.1
USN	Medium
RHSA	Moderate
DSA	9

Table 7.4: *The output of normalization step for given vulnerability ID in basic setting*

Vulnerability ID	Normalization Score	Normalization Setting
CVE-2020-8130	6.3	Basic

high qualitative rating corresponds the interval [7.0-8.9] in CVSS score, hence the lower boundary is 7.0 as shows in the Table 7.2.

Table 7.3 shows the information retrieves for CVE-2020-8130 in different VDs and Table 7.4 represents the output of normalization step in our solution for the same vulnerability ID.

7.5 Conclusion

In this paper we presented the AC-VRM method. This augments the current state-of-the-art of VRM, by adding support for organizational knowledge that directs its behavior to generate patching priorities optimal for the organization. AC-VRM operates automatically, thus reducing the discovery/identification-remediation time. Furthermore, as to maximize AC-VRM's coverage, we propose to use input from multiple VDs that is normalized to a VD independent format, while retaining links back to the VD's and the CVE ID. During the normalization process, an organization can choose what level it want to operate via VMM parameter. This gives the method maximum flexibility and adaptability, to find solutions suitable to the organization.

References

[1] *CIS Controls*. `http://www.cisecurity.org/controls/`. [Online; accessed 5-April-2020].

[2] *NIST National Vulnerability Database*. `https://nvd.nist.gov/`. [Online; accessed 8-March-2020].

[3] D. Dey, A. Lahiri, and G. Zhang. "Optimal policies for security patch management". In: *INFORMS Journal on Computing* 27.3 (2015), pp. 462–477.

[4] V. Katos, S. Rostami, P. Bellonias, N. Davies, A. Kleszcz, S. Faily, A. Spyros, A. Papanikolaou, C. Ilioudis, and K. Rantos. *STATE OF VULNERABILITIES 2018/2019*. Tech. rep. ENISA(European Union Agency for Cybersecurity), 2020.

[5] G. L. Guzie. *Vulnerability risk assessment*. Tech. rep. ARMY RESEARCH LAB WHITE SANDS MISSILE RANGE NM, 2000.

[6] E. Yuan, N. Esfahani, and S. Malek. "A Systematic Survey of Self-Protecting Software Systems". In: *ACM Trans. Auton. Adapt. Syst.* 8.4 (2014).

[7] E. Yuan, S. Malek, B. Schmerl, D. Garlan, and J. Gennari. "Architecture-based self-protecting software systems". In: *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*. 2013, pp. 33–42.

[8] A. H. Celdrán, M. G. Pérez, F. J. G. Clemente, and G. M. Pérez. "Towards the autonomous provision of self-protection capabilities in 5G networks". In: *Journal of Ambient Intelligence and Humanized Computing* 10.12 (2019), pp. 4707–4720.

[9] J. Jacobs, S. Romanosky, I. Adjerid, and W. Baker. "Improving vulnerability remediation through better exploit prediction". In: *2019 Workshop on the Economics of Information Security*. Oxford University Press, 2019.

[10] S. Lee, J. Kim, S. Woo, C. Yoon, S. Scott-Hayward, V. Yegneswaran, P. Porras, and S. Shin. "A comprehensive security assessment framework for software-defined networks". In: *Computers & Security* 91 (2020).

[11] R. Vibhandik and A. K. Bose. "Vulnerability assessment of web applications - a testing approach". In: *2015 Forth International Conference on e-Technologies and Networks for Development (ICeND)*. IEEE. 2015, pp. 1–6.

[12] Y. Tatarinova. "Avia: Automatic vulnerability impact assessment on the target system". In: *2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP)*. IEEE. 2018, pp. 364–368.

[13] Z. Liang and R. Sekar. "Fast and automated generation of attack signatures: A basis for building self-protecting servers". In: *Proceedings of the 12th ACM conference on Computer and communications security*. 2005, pp. 213–222.

[14] S. Khan and S. Parkinson. "Review into state of the art of vulnerability assessment using artificial intelligence". In: *Guide to Vulnerability Analysis for Computer Networks and Systems*. Springer, 2018, pp. 3–32.

[15] Z. Ye, Y. Guo, and A. Ju. "Zero-day vulnerability risk assessment and attack path analysis using security metric". In: *Artificial Intelligence and Security: 5th International Conference, ICAIS 2019, New York, NY, USA, July 26–28, 2019, Proceedings, Part IV 5*. Springer. 2019, pp. 266–278.

[16] R. Negi, P. Kumar, S. Ghosh, S. K. Shukla, and A. Gahlot. "Vulnerability assessment and mitigation for industrial critical infrastructures with cyber physical test bed". In: *2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS)*. IEEE. 2019, pp. 145–152.

[17] S. Shetty, M. McShane, L. Zhang, J. P. Kesan, C. A. Kamhoua, K. Kwiat, and L. L. Njilla. "Reducing informational disadvantages to improve cyber risk management". In: *The Geneva Papers on Risk and Insurance-Issues and Practice* 43.2 (2018), pp. 224–238.

[18] P. Pandey, A. Collen, N. Nijdam, M. Anagnostopoulos, S. Katsikas, and D. Konstantas. "Towards automated threat-based risk assessment for cyber security in smarthomes". In: *Proceedings of the 18th European Conference on Cyber Warfare and Security (ECCWS 2019), Coimbra, Portugal*. 2019, pp. 4–5.

[19] *Common Vulnerabilities and Exposures(CVE)*. `https://cve.mitre.org/`. [Online; accessed 1-June-2020].

[20] *Open Vulnerability Assessment Scanner(OpenVAS)*. `https://www.openvas.org/`. [Online; accessed 12-May-2020].

[21] *Qualys Vulnerability Management*. `https://community.qualys.com/vulnerability-management/`. [Online; accessed 12-May-2020].

[22] *Rapid7 Nexpose Vulnerability scanner*. `https://www.rapid7.com/products/nexpose/`. [Online; accessed 12-May-2020].

[23] *Tripwire Vulnerability Management*. `https://www.tripwire.com/products/tripwire-ip360`. [Online; accessed 12-May-2020].

[24] *Vulnerability Manager Plus*. `https://www.manageengine.com/vulnerability-management/`. [Online; accessed 12-May-2020].

[25] *SkyBox Vulnerability Management*. `https://www.skyboxsecurity.com/vulnerability-management/`. [Online; accessed 12-May-2020].

[26] *2020 VULNERABILITY AND THREAT TRENDS*. Tech. rep. SkyBox Security, 2020.

[27] S. Khan and S. Parkinson. "Towards automated vulnerability assessment". In: *11th International Scheduling and Planning Applications woRKshop (SPARK)*. 2017, pp. 33–44.

[28] J. A. Kupsch and B. P. Miller. “Manual vs. automated vulnerability assessment: A case study”. In: *First International Workshop on Managing Insider Security Threats (MIST)*. 2009, pp. 83–97.

[29] *Ubuntu Security Notice*. `https://usn.ubuntu.com/`. [Online; accessed 8-March-2020].

[30] *RedHat Security Advisories*. `https://access.redhat.com/security/security-updates/`. [Online; accessed 8-March-2020].

[31] *Cisco Security Advisories*. `https://tools.cisco.com/security/center/mpublicationListingDetails.x?docType=CiscoSecurityAdvisory`. [Online; accessed 8-March-2020].

[32] *Common Vulnerability Scoring System(CVSS)*. `https://www.first.org/cvss/`. [Online; accessed 9-March-2020].

[33] F. Inc. “Common Vulnerability Scoring System v3.1: Specification Document”. In: (2019).

[34] *Debian Security Tracker*. `https://www.debian.org/security/DSAS`. [Online; accessed 29-April-2020].

8

Normalization Framework for Vulnerability Risk Management in Cloud

Vida Ahmadi Mehri, Patrik Arlos, Emiliano Casalicchio

Published as:

Ahmadi Mehri, V., Arlos, P., and Casalicchio, E. "Normalization Framework for Vulnerability Risk Management in Cloud", 2021 8th International Conference on Future Internet of Things and Cloud (FiCloud), Rome, Italy, 2021, pp. 99-106, ©2021 IEEE. doi: 10.1109/FiCloud49777.2021.00022.

Abstract: Vulnerability Risk Management (VRM) is a critical element in cloud security that directly impacts cloud providers' security assurance levels. Today, VRM is a challenging process because the dramatic increase of known vulnerabilities (+26% in the last five years), and because it is even more dependent on the organization's context. Moreover, the vulnerability's severity score depends on the Vulnerability Database (VD) selected as a reference in VRM. All these factors introduce a new challenge for security specialists in evaluating and patching the vulnerabilities. This study provides a framework to improve the classification and evaluation phases in vulnerability risk management while using multiple vulnerability databases as a reference. Our solution normalizes the severity score of each vulnerability based on the selected security assurance level. The results of our study highlighted the role of the vulnerability databases in patch prioritization, showing the advantage of using multiple VDs.

8.1 Introduction

Vulnerability identification, classification, and evaluation are listed as the vital criteria in the Cloud Risk Assessment models [1]. Vulnerability Risk Management (VRM) is the methodology used to identify, classify, evaluate, and remediate vulnerabilities. VRM usually involves operations that require IT security specialists intervention which is time-consuming and costly. Security specialists use vulnerability scanning tools (e.g., OpenVAS, Nexpose, and Nessus are examples of vulnerability scanners) to identify known system vulnerabilities. Vulnerability scanners rely on Vulnerability Databases (VDs), which are repositories that list the vulnerabilities, each having a unique identification number, a severity score, and patch instructions (used in the remediation phase). Because VDs are maintained by different institutions they use different severity scoring systems and could cover different sets of vulnerabilities. Hence, for security experts, is challenging to select the appropriate VD and eventually they need to rely multiple VDs. In this paper, we address the aforementioned problem. We consider vulnerabilities that are documented in the Common Vulnerabilities and Exposures (CVE) referencing system and are identified by a unique CVE-ID. Some of the VDs, such as the National Vulnerability Database (NVD), use the Common Vulnerability Scoring System (CVSS) framework to rank the severity of the CVE-ID, and others develop their own scoring system frameworks. Therefore, the same CVE-ID could have different severity score depending on what VD is used. For instance, CVE-2020-8130 has a severity score "8.1" in NVD [2], "Medium" in USN [3], "9" in DSA [4], and "Moderate" in RHSA [5]. Additionally, the time when VDs publish their ratings also differ, some are fast, some take a bit more time. The patching instructions and remediation strategy may also differ between the VDs.

Hence the variety of scoring systems available, which results in different severity scores and remediation instructions, could affect the outcome of the VRM process and, as the authors in [6–8] point out, it legitimates the following research questions:

1. Is one VD enough to use in a VRM?
2. What criteria should be considered to select the reference VD in VRM?
3. What would we gain by using multiple VDs in a VRM?

Answering the above research question is important for two reasons. Firstly, VRM is key to guarantee a desired security assurance level in cloud services as described in the European Cybersecurity Certification Scheme for Cloud Services (EUCS) [9] and in the NIST guidelines for Cloud security [10, 11]. Therefore, the security expert should be provided with guidelines on which VDs to use and if and how the VD selection impact the results of the vulnerability analysis and patch prioritization. Secondly, the number of published CVE-IDs has increased enormously during the last five years, from 6447 in 2016 to 18356 in 2020 [2], resulting in a wider differentiation among the VDs and introducing additional challenges when VRM is applied to complex system such as IaaS [12].

The main contributions of this study are outlined in what follows. Firstly, we proposed a VDs Normalization Framework (VDNF) that could be integrated in the vulnerability management process. The VDNF is a refinement and implementation of what envisioned in [13]. VDNF is intended as a tool to improve the classification and evaluation phases in VRM while dealing with multiple VDs as a source. For example, VDNF could be used by IaaS security specialist dealing with the challenge of selecting the VDs more appropriate for the specific context of their organization [8].

Secondly, we implemented our VDNF and we validated the VDNF on a large set of sampled CVE-IDs. We chose the CVE-IDs that affected Ubuntu's releases from 2017 to 2020 as Ubuntu is the most used Linux distribution in Cloud computing [14]. We collected the severity score of the sampled CVE-IDs from NVD, USN, and DSA databases to cover multiple VDs.

Finally, we analyzed the distribution of severity scores in the different VDs and we analyzed the impact of the normalization framework on generalizing the severity score of the various VDs. Results confirm that using a single VD do not allow to properly analyze vulnerabilities in a complex organization context. Indeed, VDs normalization allows to cover a larger number of CVE-IDs (e.g., USN does not consider a not negligible set of CVE-IDs that are considered in DSA and NVD). Hence, in complex contexts VRM benefits of VDs normalization.

This paper is organised as in what follows. Related works are analyzed in Section 8.2. Section 8.3 firstly describes the issues in current approaches to VRM and then it describes how the proposed solution could improve VRM. Section 8.5 presents the normalisation framework and our sampling method.

The analyses of the result of our solution are presented in Section 10.7. Finally, Section 10.8 concludes the paper.

8.2 Related works

As mentioned before, the research community have recently raised the challenges introduced by the existence of multiple VDs. For example, the authors in [6] provided a comprehensive survey on the vulnerability scanning tools and databases to support security specialists in selecting the right vulnerability scanning tools and VDs. They point out the need for orchestration in vulnerability scanning tools for obtaining the highest vulnerability coverage. However, their study is limited to the vulnerability scanning tools and VDs based using the vulnerability taxonomy defined by the Open Web Application Security Project (OWASP) [15]. Moreover their study covered only the web application vulnerability detection tools [6]. The study by Dey et al. [7] concluded that patch prioritisation based on the single metric such as severity score from one VD is not an optimal patch policy.

Furthermore, the risk and impact of exploiting vulnerabilities depend on the organization's assets, security requirements, and security policies. Thus, the exploitation risk should be evaluated in the context of the organization [8]. Therefore, security specialists need to define their asset-based criteria using multiple VDs and vulnerability scanning tools. The lack of proper classification of vulnerabilities in VDs leads to some knowledge-based VDs where vulnerabilities are divided into multiple classes based on the impact of vulnerabilities' exploit [16]. Unfortunately, such VDs only use one VD as a source of truth for their classification, which does not provide the proper coverage. The state of vulnerability in 2020 shows only 84% of the vulnerabilities are registered in CVE [17] registry with an assigned CVE-ID. However, most of those unregistered vulnerabilities are related to the software patched by the software community. The security specialist should check the software's status in their organization besides the VDs, which is a time-consuming task [18].

The authors in [19] described the time factor corresponding to vulnerability risk management and patch management. They proposed the vulnerability management center where the data collected from organization inventory and NVD database are integrated with the vulnerability scanner. Their paper aims to reduce the time of vulnerability risk management proce-

dures. However, they used one generic VD as a reference, and calculated the environmental metrics based on the CVSS v2.0 standard. Such an approach required verifying the obtained score with the subject-specific VDs, which was missing in the paper.

From our literature study emerge that although the research community recognized the limitation of considering only a single VD in VRM, no work provides a clear answer to the research questions addressed by our paper.

8.3 The role of Vulnerability Databases in Vulnerability Risk Management

VDs are the repositories of publicly known vulnerabilities usually maintained by a community or organization with a specific area of interest, i.e. the subject-specific VDs. For instance, NIST National Vulnerability Database (NVD) hosts all published CVE-IDs to support the US Department of Commerce. Canonical hosts USN, a VD for tracking the vulnerabilities that affect Ubuntu releases. Debian maintains a VD containing the Debian Security Advisories (DSAs), i.e., security vulnerability that affects a Debian package. NVD is the largest VD available, it stores CVE-IDs record since 1988, and it is used as a reference VD in most of the vulnerability scanners. NVD applies the based score of the CVSS framework to rank the CVE-IDs, while Canonical developed their framework for ranking the CVE-IDs that affected their releases. The criteria and the method for calculating a severity score by Canonical are not publicly available. However, Canonical claims that their scoring system considers the impact of the vulnerability in their environments. DSA relies on NVD scores.

VRM consists of four phases: *i*) identification; *ii*) classification and analysis; *iii*) evaluation; and *iv*) remediation. The vulnerability identification phase and the vulnerability classification and analysis phase rely on VDs; phases *iii*) and *iv*) use the results from phase *ii*). Hence the results of VRM are effected by the selected VD, as detailed in what follow.

The first phase identifies the system's vulnerability by using vulnerability scanners. Those tools report the detected vulnerabilities and a severity score. The severity score they report is calculated by the VD used by the selected tool, either by design or by configuration. That VD, may or may not use the criteria listed in the CVSS framework, to calculate the severity score.

Therefore, the VD and its scoring system play a vital role in VRM.

The second phase is the classification and analysis of detected vulnerabilities. Most security specialists use the severity score to classify the vulnerabilities (e.g., detected CVE-ID with a critical severity score). Some experts use the type of exploit (e.g., detected CVE-ID that a remote attacker could exploit). The classification of the vulnerabilities impacted the evaluation for patch prioritization in the next phase. Which raises the question, which VD's severity score should consider?

The third phase is the evaluation of the impact that the vulnerabilities have, and define a patch priority. This requires in-depth knowledge about the security policy, probability of exploitation, and impact of the exploit on the system to plan a proper response and remediation. For example, the vulnerability that can exploit by the remote attacker have a higher risk than the one required local access; hence the first get a higher priority than the second.

Remediation is the last step in VRM, which patches the detected vulnerabilities, based on their priority. The remediation phase mostly relies on the software or hardware vendor (e.g., patch intel-microcode vulnerability should install the software's minimum safe version). However, some vulnerabilities might remain unpatched in a system due to the limited attack vector.

Our VRM approach relies on the organization's knowledge, to help security decision-makers analyze and evaluate vulnerabilities within the organization's context, rather than purely relying on severity scores. Organizational knowledge refers to system management documents, such as security policy, system configuration, asset inventory, and software inventory. A context-aware VRM mitigates the challenge with different severity scores from VDs, c.f. [13].

The proposed VRM approach encourages the selection of VD based on the organization's asset inventory. For instance, the Canonical severity score should use for patch prioritization in the organization with only the Ubuntu releases in asset inventory. For the organization with various vendors, multiple VDs should use as a reference. As we mentioned in Section 10.1, some VDs provide a numerical score while the others give a qualitative score. Hence obtaining a single severity score for each CVE-ID is a new challenge in VRM. The proposed VRM approach introduce a normalization framework to support security experts and decision-makers in phases I and II.

8.4 The VDs Normalisation Framework

The Normalization framework proposes a method to obtain a single severity score for each CVE-ID by aggregating scores from multiple VDs. The framework offers three working modes, hereafter Vulnerability Management Mode (VMM): Basic, Standard, and Restrictive. The proposed VMM are inline with the three assurance levels proposed in the ENISA cybersecurity certification framework [9]. A Basic VMM is the minimum acceptable baseline for a VRM process to cover a system's vulnerability identification, and with a limited public exposure scope. A Standard VMM is suitable to serve a system with medium to a high security risk, while the Restrictive mode should be used for VRM in compliant systems (i.e., the particular system that should comply with the local or international regulatory and standards) and critical infrastructure. Hence, it should be easy to select a VMM meeting the required organizations' assurance level.

The core components of the normalization framework are described in Algorithm 2. Inputs to the normalization framework are: a list of CVE-IDs, a list of the desired VDs, and the VMM. The framework produces as output, a normalized severity score per CVE-ID.

Table 8.1: *Severity Score range in CVSS v3.x and the normalisation framework*

CVSS v3.x Score	Qualitative Rate	Basic VMM	Standard VMM	Restrictive VMM
0.0	None	0	0	0
0.1-3.9	Low	0.1	2	3.9
4.0-6.9	Medium	4.0	5.45	6.9
7.0-8.9	High	7.0	7.95	8.9
9.0-10.0	Critical	9.0	9.5	10.0

The *getSeverityValue* procedure, obtains the severity score for the identified CVE-ID and selected VD. If the severity score (*"ss"*) is qualitative, the framework converts *ss* to the numeric value based on the severity score range in Table 8.1. For example if *ss=Low* and *VMM=standard*, then *ss* is turned into the value of 2. If the severity score is numeric, it is not altered. The second procedure, *normalizeScore*, calculates the average severity score for the CVE-ID.

As each organization behind a VD operates differently, there are scenarios when some VDs might not have a score for all the published CVE-ID. Using the normalization framework, we can facilitate better coverage, as it relies on input from multiple VDs. In this way, reducing the probability of having a CVE-ID without a score.

Algorithm 2 Normalization Algorithm

```
1: procedure getSeverityValue(CVE-ID, VD_i, VMM)
2:     ss = getSecScore(CVE-ID, VD_i,)
3:     if isString( ss ) then
4:         switch VMM do
5:             case basic: ss = CVSS.basic( ss );
6:             case standard: ss = CVSS.standard( ss );
7:             case restrictive: ss = CVSS.restrictive( ss );
8:     end if
9:     return ( ss ); /* in case ss is numeric it is returned directly */
10: end procedure
11: procedure normalizeScore(CVE-ID, VMM, VDset)
12:     /* VDset is the set of VDs considered */
13:     norm_score = 0;
14:     VDCnt = 0;
15:     for each VD_i in VDset do
16:         sv = getSeverityValue(CVE-ID, VD_i, VMM);
17:         if isValue ( sv ) then
18:             norm_score += sv;
19:             VDcnt += 1;
20:         else
21:             norm_score += 0;
22:             VDcnt += 0;
23:         end if
24:     end for
25:     norm = norm_score / VDcnt;
26:     if isValue ( norm ) then
27:         return norm;
28:     else
29:         return NA;
30:     end if
31: end procedure
```

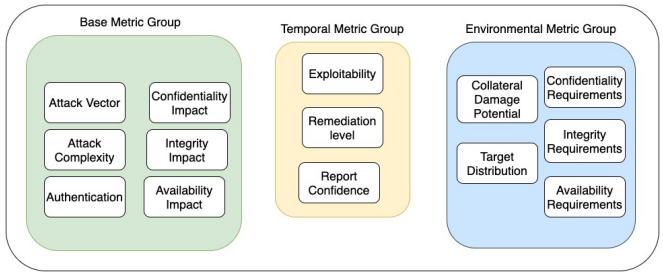

Figure 8.1: *CVSS v2.0 metrics [20]*

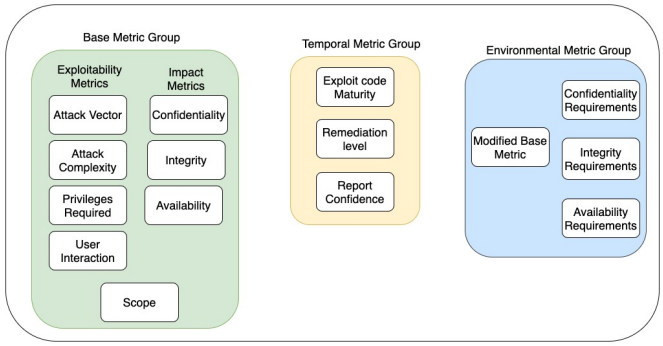

Figure 8.2: *CVSS v3.x metrics [21]*

The normalization framework uses the CVSS v3.x (i.e. version 3 and above) scoring system [21] as a reference, when converting a qualitative score to a numeric value. We use CVSS v3.x because it provides a more accurate view of the security impact on the system by expanding the basic metrics group. For instance, the authentication metrics are divided into two exploitability metrics, the *privileges required* and the *user interaction*, to provide a better evaluation on authenticity impacts, c.f. Figure 8.1 and Figure 8.2. Furthermore, a *scope* metric has been added to the basic metrics, capturing whether a vulnerability in one component affects resources managed by one or multiple security authorities. In cloud computing, this scope plays a vital role as it addresses vulnerabilities in the guest (virtual) entities that could compromise the host (Cloud infrastructure), i.e. hypervisor attacks [21].

The CVSS v2.0 and v3.x metrics are presented in Figure 8.1 and Figure 8.2. CVSS v3 provides a fine grain scale in a score by dividing the high range in CVSS v2.0 into two ranges 7.0-8.9 and 9.0-10.0. The new scale provides better visibility on vulnerabilities that have a higher risk of exploitation.

Table 8.2: *Registered CVE-IDs VS. Sample CVE-IDs*

Year	Registered CVE-ID	Sample CVE-ID
2017	14646	4043
2018	16511	3179
2019	17305	2952
2020	18355	2387

8.5 Validation Case study

To validate the proposed normalization framework, we apply the VDNF to three VDs that use different scoring mechanisms:

- National Vulnerability Database, NVD
- Debian Security Advisories, DSA
- Ubuntu Security Notices, USN

We chose NVD as it calculates a base score for CVSS v2.0 and v3.x, and records both numeric and qualitative score for each CVE-ID. We picked DSA as it reports a qualitative score which relies on NVD score but does not clarified the version of CVSS. The last, USN provides a qualitative score where the criteria and calculation method is not available. Hence, the severity scores from DSA and USN are qualitative and need to be converted to a numeric value by the normalization procedure in Algorithm 2.

To validate the framework, we have selected CVE-IDs that has affected Ubuntu's releases from 2017 to 2020. We collected the information for those CVE-IDs on February 4th, 2021 from the selected VDs. For each VD we recorded the CVE-ID, severity score, description of the vulnerability, date of publishing CVE-ID, and a reference link. The data (available in GitHub)[1] was saved in a CSV file, one for each year. Table 10.2 presents the number of the registered CVE-IDs, and the number of sample CVE-IDs, per year.

We use a Python script to process the data and implement the normalization framework, described in Algorithm 2.

[1] DVNF data repository https://github.com/vidaAhmadi/sample-CVE

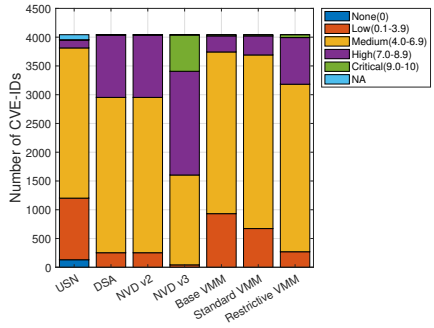

Figure 8.3: *Distribution of the severity scores in VDs and normalisation framework in 2017.*

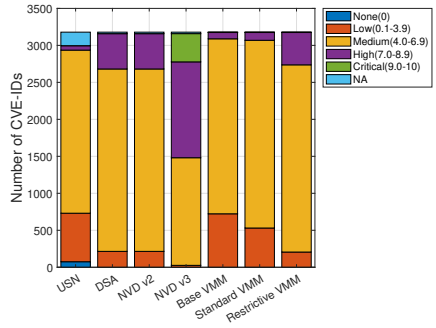

Figure 8.4: *Distribution of severity scores in VDs and normalisation framework in 2018.*

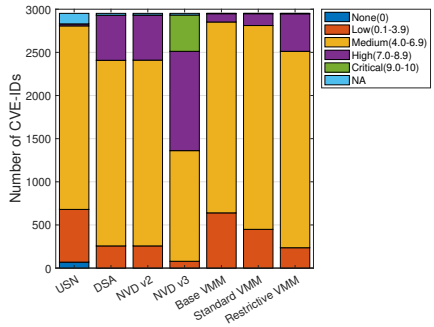

Figure 8.5: *Distribution of the severity scores in VDs and normalisation framework in 2019.*

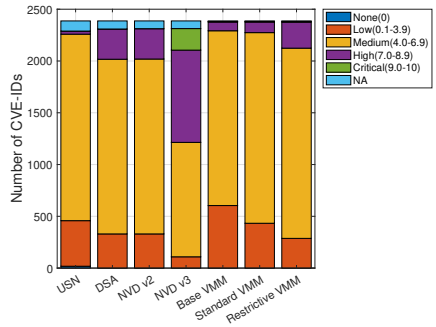

Figure 8.6: *Distribution of the severity scores in VDs and normalisation framework in 2020.*

8.6 Analysis Result

First, we analyze the distribution of severity scores for each of VDs. Then, we compare the distributions between the VDs. Finally, we analyze the impact that the normalization framework has on the distribution of severity scores.

8.6.1 Distribution of Severity Scores

8.6.1.1 USN

The severity score is reported as a priority in USN VD. Hence, severity score none is ranked negligible in USN. The majority of the CVE-IDs have

a medium score, and only three CVE-IDs ranked critical in our samples space. The NA in the Figures 8.3 to 8.6 refers to the number of CVE-ID that does not have any score. USN has the highest number of NA in our samples. Furthermore, the number of registered CVE-IDs that affected Ubuntu releases decreased by 15.2% from 2017 to 2020 as shown in Table 10.2.

8.6.1.2 DSA

Our result shows that DSA scores matches quite well with NVD v2. There is no critical score in our sample CVE-IDs, this proves that the scores are calculated with CVSS v2.0, or similar, because the critical range was introduced in CVSS v3.x. Furthermore, We did not find any evidence of independently calculated severity scores in the DSA VD or any reference to the framework used for calculating scores.

8.6.1.3 NVD

NVD uses the CVSS framework to calculate the severity score mentioned in its general documentation [2]. Most of the CVE-IDs have a severity score, from both CVSS v2.0 and v3.x, since 2016. The NVD score consists of the base metric group only, while the temporal metric and environment metric are excluded in the value. Therefore, we consider NVD as a general-purpose VD.

Figures 8.3 to 8.6 visualized the NVD has the lowest number of NA in each year comparing with the other two VDs. We also see most CVE-IDs in NVD v2 are in the medium range. Furthermore, NVD reported the least number of CVE-IDs in the low range per year in our samples.

8.6.2 Comparison of Distributions

We selected USN and DSA as they are product-specific VDs. Ubuntu and Debian are both Linux distributions, one can expect a similar severity score in those databases. Furthermore, Ubuntu is built on-top of the Debian distribution. However, Table 8.3 and Figures 8.3 to 8.6 show no similarity between severity scores in DSA and USN. Nevertheless, the severity score reported by NVD v2 is similar to the one reported by DSA. Another finding is that the range *None* only exists in USN.

[2] NVD documentation https://nvd.nist.gov/general

Table 8.3: *Severity Score distribution in VDs and the normalisation framework 2017-2020*

		VD				VMM		
Year	Score	USN	DSA	NVD CVSS v2.0	NVD CVSS v3.x	Basic	Standard	Restrictive
2017	0.0 (None)	131	0	0	0	0	0	0
	0.1-3.9 (Low)	1071	253	253	42	932	673	270
	4.0-6.9 (Medium)	2611	2700	2700	1562	2811	3018	2912
	7.0-8.9 (High)	139	1085	1085	1803	279	331	811
	9.0-10.0 (Critical)	3	0	0	631	21	21	50
	Not Available	88	5	5	5	0	0	0
2018	0.0 (None)	75	0	0	0	0	0	0
	0.1-3.9 (Low)	656	215	215	26	723	531	206
	4.0-6.9 (Medium)	2205	2466	2466	1455	2366	2538	2531
	7.0-8.9 (High)	60	478	478	1296	85	105	437
	9.0-10.0 (Critical)	0	0	0	382	3	3	3
	Not Available	183	20	20	20	2	2	2
2019	0.0 (None)	69	0	0	0	0	0	0
	0.1-3.9 (Low)	611	257	257	79	640	449	236
	4.0-6.9 (Medium)	2127	2151	2153	1282	2212	2362	2276
	7.0-8.9 (High)	23	523	523	1151	96	137	435
	9.0-10.0 (Critical)	0	0	0	421	4	4	5
	Not Available	122	21	19	19	0	0	0
2020	0.0 (None)	18	0	0	0	0	0	0
	0.1-3.9 (Low)	440	330	330	109	604	433	287
	4.0-6.9 (Medium)	1801	1687	1689	1105	1688	1841	1837
	7.0-8.9 (High)	30	292	293	891	85	103	252
	9.0-10.0 (Critical)	0	0	0	208	3	3	4
	Not Available	98	78	75	74	7	7	7

We also see that USN has the largest number of CVE-IDs in the low range, and lowest number of CVE-IDs in the High range comparing with other VDs in each year.

As mentioned before, it may be that some CVE-IDs do not have a score in a VD, this is represented as *"Not Available"* in table 8.3, and NA in Figures 8.3 to 8.6. The NA scores for 2020 are all related to recently published CVE-IDs, and it usually takes some time for the score be updated in VDs. However, USN reported the largest NA scores in 2018 for the vulnerable packages with an unknown impact on Ubuntu releases.

8.6.3 Impact of Normalization

Figures 8.3 to 8.6 shows the distribution of the severity score range in VDs and normalisation framework. The normalisation score is distributed in the four ranges Low, Medium, High, and Critical and the number of CVE-ID in the range None is zero in 2017-2020. We also identify the number of CVE-IDs without a score (cf., NA): in 2017 and 2019 it is zero after normalization regardless of the VMM, while the number on CVE-IDs with NA score are two (0.06%) and seven (0.33%) in 2018 and 2020 relatively. Hence, after normalization the number of the CVE-IDs with not available score is negligible.

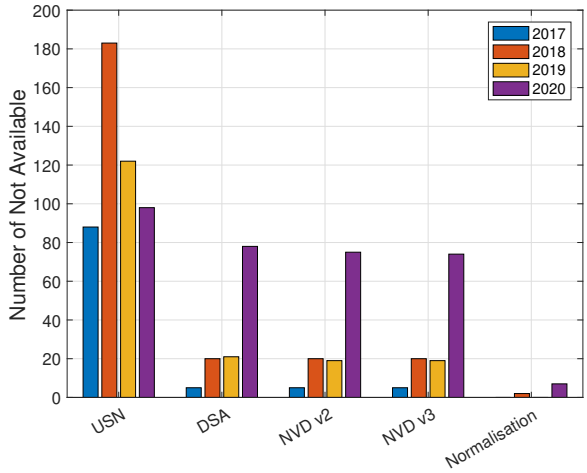

Figure 8.7: *Distribution of Not Available Score in 2017-2020*

We observe the benefits of normalization on the better coverage on CVE-IDs score. Table 8.5 presentes the number of CVE-IDs that does not have a score in each VDs and the changes on the score after applying normalization framework. We find the change in the score range for CVE-IDs with NA, does not depend on the VMM. Hence, the result in Table 8.5 is the same for Basic, Standard, and Restrictive mode.

We visualized the change in not available score in VDs and normalisation in Figure 8.7. We noticed the most of the CVE-IDs ranked NA in VDs, moved to the medium range after applying normalization framework.

To have a fair comparison of the normalization impact, we calculate the percentage of the CVE-IDs, which does not change its range after applying the normalization framework. Table 8.4 presents the percentage of the changed and unchanged severity score range in 2017-2020. We noticed that USN reported a higher percentage of unchanged severity score range in the basic VMM, while the restrictive VMM provides the higher unchanged range in NVD and DSA.

8.7 Conclusion

We proposed and implemented a normalization framework to facilitate classification and evaluation phases in VRM when dealing with multiple

Table 8.4: *The percentage of the CVE-ID with changed and unchanged severity score range after normalization 2017-2020*

Year		2017		2018		2019		2020	
VMM	VD	% unchanged	% changed	% unchanged	% changed	% unchanged	% changed	% unchanged	% changed
Basic	USN	75,2	24,8	81,5	18,5	78,4	21,6	78,2	21,8
	DSA	62,8	37,2	71,1	28,9	72	28	77,5	22,5
	NVD v2.0	62,8	37,2	71	29	72,1	27,9	77,3	22,7
Standard	USN	71	29	79,1	20,9	75,9	24,1	77,4	22,6
	DSA	66,9	33,1	73,4	26,6	74,7	25,3	78,3	21,7
	NVD v2.0	66,9	33,1	73,3	26,7	74,8	25,2	78,1	21,9
Restrictive	USN	49,2	50,8	59,5	40,5	59,5	40,5	66,3	33,7
	DSA	87,6	12,4	93,1	6,9	91,2	8,8	89,3	10,7
	NVD v2.0	87,6	12,4	93	7	91,3	8,7	89,1	10,9

Table 8.5: *The change in severity score of Not Available in VDs after normalisation 2017-2020*

Year	VD	Not Available(NA)	NA➔ NA	NA ➔ Low	NA➔ Medium	NA➔ High	NA➔ Critical
2017	USN	88	0	4	55	17	12
	DSA	5	0	2	2	1	0
	NVD	5	0	2	2	1	0
2018	USN	183	2	8	149	21	3
	DSA	20	2	0	18	0	0
	NVD	20	2	0	18	0	0
2019	USN	122	0	10	90	21	1
	DSA	21	0	4	17	0	0
	NVD	19	0	4	15	0	0
2020	USN	98	7	18	64	8	1
	DSA	78	7	19	52	0	0
	NVD	75	7	19	49	0	0

VDs. Our solution provides a numeric security score for each CVE-ID by applying a normalization algorithm. The same type of score (i.e., numeric score) from multiple VDs helps the security decision-maker to generalize the severity score. In this study, we evaluated the normalization framework's impact on a large set of sampled CVE-IDs and analyzed the distribution of severity scores.

This study shows the significant impact of the VDs' severity score on classification, evaluation, and patch prioritization in VRM. Hence, selecting one general scoped VD to use in VRM would not be sufficient for patch prioritization as environmental metrics are not considered in the score calculation. Based on our result, we recommend security expert to select at least one subject-specific and one general VD, and to use the normalized scores for decision making in the organizations's VRM.

Our study declares that we need multiple VDs as a reference in VRM to get better severity score coverage of the known CVE-IDs. Our result indicates the organization assets should be used as essential criteria for the selection of reference VDs. For example, suppose the organization used the Dell servers and Debian's operating system. In that case, they have to use Dell's and Debian's VD as reference VDs in VRM.

In the future, we will add another subject-specific VD, such as RedHat, to our sample VDs for expanding our study on the impact of the normalization framework. We also will classify, evaluate and prioritize the patch based on normalized score.

References

[1] A. Amini and N. Jamil. "A comprehensive review of existing risk assessment models in cloud computing". In: *Journal of Physics: Conference Series*. Vol. 1018. 1. IOP Publishing. 2018, p. 012004.

[2] *NIST National Vulnerability Database*. `https://nvd.nist.gov/`. [Online; accessed 30-January-2021].

[3] *Ubuntu Security Notice*. `https://usn.ubuntu.com/`. [Online; accessed 8-March-2020].

[4] *Debian Security Tracker*. `https://www.debian.org/security/#DSAS`. [Online; accessed 29-April-2020].

[5] *RedHat Security Advisories*. `https://access.redhat.com/security/security-updates/#/`. [Online; accessed 8-March-2020].

[6] K. Kritikos, K. Magoutis, M. Papoutsakis, and S. Ioannidis. "A survey on vulnerability assessment tools and databases for cloud-based web applications". In: *Array* 3 (2019), p. 100011.

[7] D. Dey, A. Lahiri, and G. Zhang. "Optimal policies for security patch management". In: *INFORMS Journal on Computing* 27.3 (2015), pp. 462–477.

[8] G. L. Guzie. *Vulnerability risk assessment*. Tech. rep. ARMY RESEARCH LAB WHITE SANDS MISSILE RANGE NM, 2000.

[9] *European Cybersecurity Certification Scheme for Cloud Services*. `https://www.enisa.europa.eu/publications/eucs-cloud-service-scheme`. [Online; accessed 11-February-2021].

[10] W. Jansen and T. Grance. *Sp 800-144. guidelines on security and privacy in public cloud computing*. 2011.

[11] M. Hogan, F. Liu, A. Sokol, and J. Tong. "Nist cloud computing standards roadmap". In: *NIST Special Publication* 35 (2011), pp. 6–11.

[12] V. Singh and S. Pandey. “Cloud Computing: Vulnerability and Threat Indications”. In: *Performance Management of Integrated Systems and its Applications in Software Engineering*. Springer, 2020, pp. 11–20.

[13] V. Ahmadi, P. Arlos, and E. Casalicchio. “Normalization of Severity Rating for Automated Context-aware Vulnerability Risk Management”. In: *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. IEEE. 2020, pp. 200–205.

[14] *Best Linux distributions for DevOps*. `https://cloudacademy.com/blog/linux-and-devops-the-most-suitable-distribution/`. [Online; accessed 23-January-2021].

[15] *Open Web Application Security Project*. `https://www.owasp.org`. [Online; accessed 07-March-2021].

[16] P. Kamongi, S. Kotikela, K. Kavi, M. Gomathisankaran, and A. Singhal. “Vulcan: Vulnerability assessment framework for cloud computing”. In: *2013 IEEE 7th International Conference on Software Security and Reliability*. IEEE. 2013, pp. 218–226.

[17] *The State of Open Source Security VULNERABILITIES*. `https://www.whitesourcesoftware.com/open-source-vulnerability-management-report/`. [Online; accessed 23-January-2021].

[18] M. J. Haber and B. Hibbert. “Vulnerability Management Architecture”. In: *Asset Attack Vectors*. Springer, 2018, pp. 213–216.

[19] M. Walkowski, M. Krakowiak, J. Oko, and S. Sujecki. “Distributed Analysis Tool for Vulnerability Prioritization in Corporate Networks”. In: *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE. 2020, pp. 1–6.

[20] *Common Vulnerability Scoring System(CVSS v2.0: User Guide)*. `https://www.first.org/cvss/v2/guide`. [Online; accessed 2-February-2021].

[21] *Common Vulnerability Scoring System(CVSS v3.0: User Guide)*. `https://www.first.org/cvss/v3.0/user-guide`. [Online; accessed 2-February-2021].

9

Automated Context-aware Vulnerability Risk Management for Patch Prioritization

Vida Ahmadi Mehri, Patrik Arlos, Emiliano Casalicchio

Published as:

Ahmadi Mehri, V., Arlos, P., and Casalicchio, E. "Automated Context-Aware Vulnerability Risk Management for Patch Prioritization", Electronics 2022, 11, 3580. https://doi.org/10.3390/electronics11213580

Abstract: The information security landscape continuously evolves by discovering new vulnerabilities daily and sophisticated exploit tools. Vulnerability Risk Management (VRM) is the most crucial cyber defense to eliminate attack surfaces in IT environments. VRM is a cyclical practice of identifying, classifying, evaluating, and remediating vulnerabilities. The evaluation stage of VRM is neither automated nor cost-effective, demanding great manual administrative efforts to prioritize the patch. Therefore, there is an urgent need to improve the VRM procedure by automating the entire VRM cycle in the context of the given organization. The authors propose automated Context-aware VRM (ACVRM) to address the above challenges. This study defines the criteria to consider in the evaluation stage of ACVRM to prioritize the patching. Moreover, patch prioritization is customized in the organization's context by allowing the organization to select the vulnerability management mode and weigh the selected criteria. Specifically, this study considers four vulnerability evaluation cases: i) evaluation criteria are weighted homogeneously; ii) attack complexity and availability are not considered important criteria; iii) the security score is the only important criteria considered, and; iv) criteria are weighted based on the organiza-

tion's risk appetite. The result verifies the proposed solution's efficiency compared with the Rudder vulnerability management tool (CVE-plugin). While Rudder produces a ranking independent from the scenario, ACVRM can sort vulnerabilities according to the organization's criteria and context. Moreover, while Rudder randomly sorts vulnerabilities with the same patch score, ACVRM sorts them according to their age, giving a higher security score to older publicly known vulnerabilities.

9.1 Introduction

Vulnerability Risk Management (VRM) is one of the critical aspects of information security. Many of today's cyberattacks exploit systems' vulnerabilities (e.g., CVE-2021-40444, CVE-2021-44228, CVE-2021-3156) [1]. Unpatched vulnerabilities caused 60% of the known data breaches, according to to [2]. Hence, VRM is a fundamental part of information security management in each organization. It consists of identifying, classifying, evaluating, and remediation of vulnerabilities.

According to [3], identification of vulnerabilities by vulnerability scanner tools (like OpenVAS [4] and Nessus [5]) is only a small part of the VRM process. Security experts use these scanners to inspect their systems regularly. In addition, security experts' knowledge of the organization is crucial to evaluating the risk of exploits and prioritizing the order of patches. The evaluation and prioritizing of remediation are the challenging parts of the VRM process. The requirement to involve security experts and the dramatic increase of known vulnerabilities in the last five years (+26%) (2016-2021) have made the VRM process time-consuming and expensive. The research question is, therefore: *How to make the VRM process time-efficient, cost-effective, and organization oriented*?

To answer the above research question, we have introduced the concept of Automated Context-aware Vulnerability Risk Management (ACVRM) [6, 7]. ACVRM facilitates the customization of the VRM process for a given organization by learning about the organization's assets and the vulnerabilities that affect the assets. ACVRM automates the VRM process by applying predefined decision criteria and related activities, thus saving time and cost.

In our previous studies [6, 7], we identified that the selection of what Vulnerability Database (VD) to use plays an essential role in the VRM

procedure and the information on the organization's assets should support the VD choice. Indeed, the vulnerability severity score comes from a VD, and there are several types of VD, from national [8] to vendor [9, 10], and even application-specific [11]. Vendors' VD, such as RedHat [9], usually list their affected releases, severity score in their environments, and patch instructions for the vulnerabilities that affected their products. In contrast, national VD's provide general information about the vulnerability and a severity score. Unfortunately, existing vulnerability scanner tools do not allow the selection of the VD to use. Moreover, all of them rely on a single VD. In addition, scanners do not know the system architecture and organization's configuration policy to identify the actual exposure of the vulnerability in the organization. Therefore, security experts should define criteria in the evaluation step in VRM to prioritize the remediation of the vulnerabilities. This research enhances our previous work [6, 7] focusing on the *VDs selection problem* and the challenge of *defining evaluation criteria for context-aware patch prioritization*. Our contribution is summarized as follows:

- We present the Prioritization phase workflow of the ACVRM framework describing the details of the Filter, Evaluation, and Patch Prioritization stages.
- We define a *Patch Score* criteria to prioritize patching that could adapt to the organization's context. The criteria build on security experts' interviews and a literature study.
- We implement a Proof of Concept (PoC) of the ACVRM framework.
- We validate the ACVRM PoC against the prioritization obtained using the Rudder tool. In the evaluation, we consider four case studies for the organizational context which impact the Patch Score. Results show the capability of ACVRM in customizing patch prioritization.

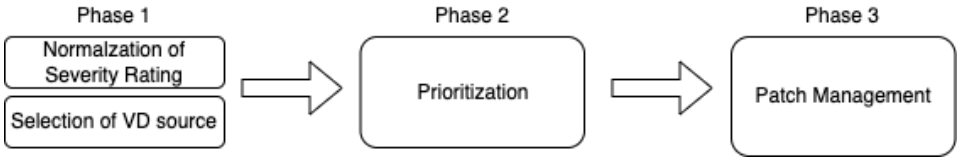

Figure 9.1: *Relationship between our current work (Phase 2) and previous work (Phase 1), and how they address the ACVRM phases*

Figure 9.1 shows the relationship between our previous work and the current work. The initial idea for ACVRM was presented in [6], and in that study, the focus was on calculating a normalized vulnerability score based on multiple vulnerabilities. This work was then augmented in [7] where we investigated the impact that the selection of VD or VD's has on the obtained score. These two works are centered around the first phase, *Retrieval and pre-processing* in the ACVRM, see Figure 9.2. This work focuses on the second phase, *Prioritization*. The third phase, *patch management* is left out for the time, as most organizations use tools to apply patches.

The paper is organized as in what follows. Section 10.2 provides background on VRM and analyzes the related literature. Section 10.4 introduces the ACVRM framework, and Section 9.4 describes the ACVRM's prioritization phase, where the paper's core contribution is. Section 9.5 presents the selection of the evaluation criteria and the definition of the patch score. Section 9.5 describes the design and implementation of a proof of concept for the prioritization phase. Experiments and results are reported in Sections 9.7 and 10.7 , respectively. Section 10.8 concludes the paper.

9.2 Related Work

Continuous VRM is in the top 10 critical security controls defined by the Center of Internet Security (CIS) [12]. VRM is one of the vital criteria to guarantee system compliance. Most information security standards (e. g., ISO 27002, PCI, SOC2) and legislation (e.g., EU Cybersecurity act [13], EU Cybersecurity Certificate (EUCS) [14], USA homeland security act [15]) include VRM as a critical control. Hence, organizations must establish a VRM process to remediate the identified vulnerabilities.

Keeping up with assessing hundreds of vulnerabilities daily is a big challenge for the security team in each organization. It is impossible to patch all detected vulnerabilities due to resources and time limitations. Therefore, most tools and security analysts prioritized the remediation based on the severity score. The severity score could be calculated using the Common Vulnerability Scoring System (CVSS). CVSS is an open framework that transfers the vulnerability characteristic to a numeric score [16]. The score obtained from CVSS is static, and the numeric value of each metric does not change over time. To overcome this problem, researchers proposed a methodology to change the numeric value of impact metrics (i.e., Confiden-

tiality, Integrity, and Availability (CIA)) in the CVSS version 2.0 in favor of improving the CVSS scoring [17]. The authors found that the violation of confidentiality is more severe than integrity and availability, hence should not weigh equally. Another approach proposed to improve patch prioritization in the VRM process is to add temporal and environmental metrics to the CVSS score [18]. Rather than adding new metrics to the CVSS or changing CVSS information over time, we propose to feed the VRM process with the Organization Context (OC) data. Indeed, in ACVRM, the organization context data complements the CVSS information in evaluating the vulnerability ranking. The OC is the set of data that defines the assets the organization intends to protect and the rules. The OC data we propose to use in ACVRM are described in Section 9.4.1 and 9.4.2.

According to the Ponemon Institute [2], 32% of the survey's participants made a remediation decision based on the CVSS score. 59% of participants in the survey disclosed that their organizations were not performing the complete VRM's life cycle. A gap in the VRM life cycle is seen as an opportunity for adversaries to leverage vulnerabilities. The 2021 Check Point Cyber Security Report [19] reveals that 80% of attacks in 2020 took advantage of the vulnerabilities reported in 2017 or earlier. Furthermore, around 50% of the participants in the Ponemon survey [2] recognized that automation is a key to responding to a vulnerability promptly. To address the above-mentioned issue, we designed ACVRM to adapt its behavior based on the organization context to prioritize the remediation.

Many studies have applied machine learning-based solutions to predict remediation decisions and classify the type of vulnerability in different domains such as power grid and software development. For example, authors in [20] built their decision tree based on data of the asset and vulnerability features for a power grid and reached 97% accuracy. However, their solution is domain-specific and requires manual verification on the small prediction portion to reduce false negatives. On the contrary, in ACVRM, we propose to improve patch prioritization over time based on the historical organizational data from the feedback loop.

Vulnerability categorization is also helpful in automating VRM and in the software development life cycle. In [21], the authors propose using multiple machine learning algorithms to classify the vulnerabilities into vulnerability categories, as understanding vulnerability types is crucial in the software

development life cycle. Similarly, a machine learning algorithm allows the classification of vulnerability types in a security patching of open-source software [22]. In our solution, we foreseen to apply, as future work, a machine learning algorithm to improve patch prioritization based on the patch verification feedback.

In [23], the authors proposed the automated CVSS-based vulnerability prioritization. Their solution uses only the vulnerability scanner report of the environment and prioritizes the patch based on the confidentiality, integrity, and availability score. The authors concluded that using a CVSS-base score is insufficient, and they should consider other metrics in a prioritization step in the future. SmartPatch is a patch prioritization method for Industrial Control Systems [24]. SmartPatch uses the network topology of the environments and the vulnerability scanner report to address patch sequencing in an interdependent and complex network. SmartPatch proposed a security metric called Residual Impact Score (RIS) by utilizing the score of the impact metrics and exploitability metrics of CVSS exported from the National VD (NVD). The authors in [25] used a mathematical approach to select the vulnerability from the scan report for remediation concerning the available experts. They used the CVSS score from NVD, the available hours of security experts, the vulnerability's age, and its persistence in the monthly scan in their approach. They concluded that the number of unpatched vulnerabilities was the lowest using multiple attributes. In [26], the authors proposed the machine learning-based method to address the inconsistency of CVSS scores across multiple VDs. They trained their algorithm with a different version of CVSS scores in NVD and validated their result with crawled vulnerability reports from SecurityFocus. Then they implemented the case study in cyber-physical systems to assess the severity of the vulnerability. The result of their case study indicated the diversity of vulnerability scores on different data sources that mislead the experts in patch prioritization. Compared to the above research work ([23, 24, 26]), ACVRM facilitates the patch prioritization for organizations independently of the domain. It utilized multiple VDs, host inventory, and scan reports to detect the existing vulnerabilities. ACVRM also customized the VRM procedure for the organization by enabling them to select the Vulnerability Management Mode (VMM) and to weigh the criteria used in patch prioritization.

Table 9.1 summarizes the comparison between our solution and the most recent state-of-the-art works on vulnerability prioritization. The comparison

Table 9.1: *Comparison of our approach to most recent related work. For acronym cf. Section 5 and the Abbreviations section*

	Reference VD	Identification approach	Evaluation criteria	Contribution
Walkowski et al. [23]	NVD	Vulnerability scan	SC, C, I, A, CDP	Proposed VRM improvement by prioritizing the patch based on the CDP value for monitored IT sources and the ratio of detected vulnerabilities to the number of monitored resources
Yadav et al. [24]	NVD	Vulnerability scan and network topology	SC, C, I, A, ER, functional and topological dependencies	Defined security metric Residual Impact Score (RIS) to prioritised the patch based on cost of defence, cost of attack, and impact of attack
Jiang et al. [26]	NVD and securityFocus	Vulnerability scan and system configuration	SC	Proposed machine learning based structure to correlate SC from multiple sources to overcome inconsistency in CVSS score
Shah et al. [25]	NVD	Vulnerability scan	SC, age, and persistence	Defined mathematical model for optimizing remediation priority with respect to evaluation criteria
Our work	NVD, DSA, RHSA, and USN	Vulnerability scan, asset inventory, and VDs data	SC, C, I, A, AV, AC, and Access level	Proposed the criteria for patch prioritization and customised the patch in the organization's context

performs along the following features: the VD used as reference (Reference VD), the vulnerability identification approach, the vulnerability evaluation criteria, and the contribution provided. The comparison highlights what follows: our solution is the only one that allows multiple VDs as input for vulnerability identification; there is a shared consensus on using multiple sources of information to identify vulnerabilities and to use multiple evaluation criteria. Concerning evaluation criteria, while the majority of the proposed solutions use the security score (SC), confidentiality (C), integrity (I), and availability (A), ACVRM also adopts the attack vector (AV) and attack complexity (AC) along with the access level (Internal or external AUS) metrics. The complexity of attacks is also addressed by other works using Exploitation Rate (ER) or Collateral Damage Potential (CDP) metrics.

9.3 Automated Context Aware Vulnerability Risk Management (ACVRM)

ACVRM aims to improve the VRM as follows: (1) it uses multiple VDs for retrieving Common Vulnerabilities and Exposures (CVE) [27] data; (2) it

automates the classification of vulnerabilities and (3) the patch prioritization process based on the organization's requirements. ACVRM is structured in three phases, as shown in Figure 9.2. Phase 1 has been addressed in our previous works [6, 7], phase 2 design and implementation is the main contribution of this paper; and phase 3 is considered to be future work.

During phase 1, ACVRM retrieves CVE data from multiple VDs. From each VD, we collect CVE-IDs, their publication date, description, severity score, affected releases, and safe version. We also store a timestamp to know when we collected or updated the data in our local database. To keep the local database updated, ACVRM periodically checks for changes in the source VD. If changes are detected, the local database is updated while keeping the old version of CVE-ID data in an archive for future reference. The pre-process stage converts the quantitative severity score of CVE-ID (if any) to an internal numeric score using the conversion algorithm described in [7]. Our internal score is based on CVSS 3.x score. Pre-process stage makes the CVE-ID data ready for the normalization stage. The main task in the Normalization stage is calculating a severity score for each CVE-ID. ACVRM offers three VMM: Basic, Standard, and Restrictive. These are in line with the three assurance levels proposed by EUCS[14]. A Basic VMM is the minimum acceptable baseline for a VRM process suggested for an organization with a limited risk of exploitation (e.g., an organization with a limited system exposed to the Internet). Standard VMM is suitable to serve an organization with medium to high-security risks. At the same time, the Restrictive VMM should be used in the compliant organization (i.e., an organization or governmental agency that should comply with local and international regulations or standards and critical infrastructure). The normalization stage calculates the normalized score for each CVE-ID by averaging the severity scores concerning VMM mode.

In phase 2, ACVRM determines the patch prioritization for the organization's needs, specified by the organization's context. Phase 2 is the core part of ACVRM and is described in detail in Section 9.4.

In the third phase, Patch management, ACVRM patches the detected vulnerabilities and verifies that the system functionalities are not compromised.

In the first stage of patch management, the automated patching component executes patch prioritization on vulnerable hosts. Then ACVRM

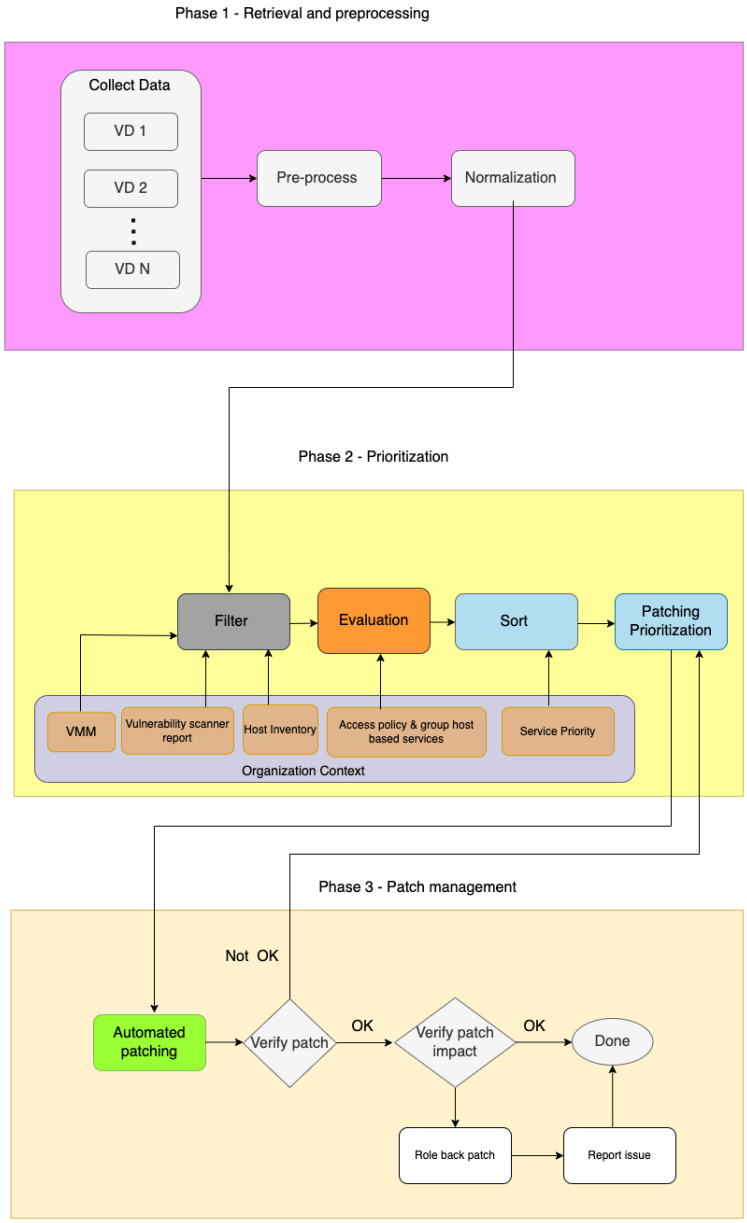

Figure 9.2: *The ACVRM phases*

verifies if the patch was successful. If the error occurs for an item in the patch prioritization list, it will jump to the next item in the list and record the one

in an error state. If an error appears due to the patch order (e.g., patching microcode vulnerability in Ubuntu requires the Kernel to be patched in advance), it will re-execute the patch at the end. For the persistent error, the report with the error state will submit to the patch prioritization stage in phase 2 for review. The final stage in phase 3, verification, consists of evaluating the impact of the patch on the Application/Unit/Service's (AUS) functionality. The functional tests refer to the series of predefined tests by experts to investigate the health of AUS. In case of unexpected behavior, the issue reports to the experts.

9.4 Prioritization

Prioritization is the second phase of ACVRM, which determines the order of CVE-IDs to be patched in each host. Figure 9.3 shows the stages in prioritization phase. In the following sections, we will briefly describe these stages.

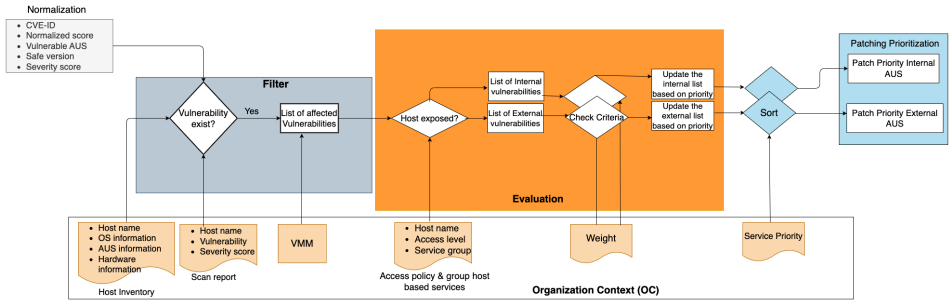

Figure 9.3: *The Phase 2 – Prioritization process*

9.4.1 Filter

The task of the Filter stage is to identify vulnerabilities that affect the organization's assets (i.e., application, software, servers). The inputs are the data from the normalization stage, the Host Inventory, and the Vulnerability Scan reports provided with the OC, cf. Figure 9.3. The output is the list of vulnerabilities affecting the organization's assets. The list includes CVE-IDs, hostname, name of vulnerable AUS, and normalized score. In more detail, the OC data used in the Filter stage are:

- Host Inventory (HI): consisting of the hosts and assets belonging to the organization. The host inventory provides the list of hosts, hardware specifications, and installed software. Examples are asset management tools (e.g., Device42, NinjaRRM, Solarwinds) or custom tools.
- Vulnerability Scanner Report (VSR): is a source used by security specialists for patch prioritization. The best security practices for the cloud, such as C5 [28] suggest monthly vulnerability scanning, which leads to thirty days of patch planning.
- Vulnerability Management Mode (VMM): defines in which mode ACVRM should operate. The organization sets a default VMM mode for the whole organization, but this can override with host-based VMM, e.g., VMM basic for host A, VMM restricted for host B, while the default is standard.

The Filter identifies the CVE-IDs that impact the organization by comparing the vulnerable software and their existence in the HI and VSR. For example, a vulnerability scanner might report faulty configurations with no CVE-ID reference (i.e., Nessus ID 153953: SSH server configured to allow weak key exchange algorithms). We might also find vulnerable software installed in the hosts in the organization's environment but not detected by the vulnerability scanner(i.e., the vulnerability in sudo before 1.9.5p2 (CVE-2021-3156) that the Nessus scanner has not discovered in our test environment). Therefore, we obtain better coverage of the potential vulnerabilities by considering both the HI and VSR. Let C represent all collected vulnerabilities; then, the filter will produce a list of vulnerabilities ($VULN$) that are affected by the VSR and/or HI

$$VULN = (C \cap C_{VSR}) \cup \Lambda(C, HI)$$

where C_{VSR} is the set of vulnerabilities contained in the VSR; and $\Lambda(C, HI)$ is a filter function that returns only the vulnerabilities that affect the hosts.

9.4.2 Evaluation

The task in the Evaluation stage is to examine the risk of each vulnerability and provide the patch prioritization. The input for the evaluation stage is the list of affected vulnerabilities from the Filter stage, the access policy and group host-based services, and the weight from the OC.

The access policy and group-based services describe conceptual information to enforce business requirements. It defines access to applications/services based on the host group, locations, and time. The access policy provides information on how accessible different AUS are, i.e., AUS exposed to the public are probably more likely to be compromised than AUS that are not. The group-based services simplify the patch process as the same patching and verification instructions will apply.

In the evaluation stage, we divide the list of affected vulnerabilities based on the access policy into external and internal groups. The external refers to the AUS being exposed to the Internet, thus having a higher risk of exploitation. On the other hand, internal indicates the AUS with limited access levels, i.e., authorized users with defined IP addresses in the Access Control List (ACL).

In this stage, we also check the criteria that impacted the patch sequence. The organization could customize the criteria by weighting them based on the impact on its business cf. 9.5.

9.4.3 Sort

The task for the Sort stage is to update the order of the CVE in each evaluated list (i.e., external list). The inputs are the output from the evaluation stage and the service policy. The service policy is optional information to influence the order in the patch list for the vital services for the organization. The services listed in service priority grant a higher position in the patch list. If the organization does not provide service priority, the sort will be based on the PS score cf. 9.5.4. The outputs are two sorted lists of vulnerabilities for internal and external AUS. Each list includes CVE-IDs, hostname, name of vulnerable AUS, PS, and priority number.

9.4.4 Patching Prioritization

The main task for patching prioritization is to adjust the patch order based on learning. This stage receives the error feedback from patch verification. It builds the knowledge to map situations to actions over time. The patch prioritization input is the sort stage output and the feedback loop from the phase 3 of ACVRM. In the first round, the patch prioritization stage provides the same output as the sort stage, as it is not yet received any feedback from phase 3.

9.5 Evaluation criteria and patch score

Finding a suitable criteria for evaluating vulnerabilities is a challenge as demonstrated by the multiple research studies on this subject [18, 23, 24, 29–38]. As we mentioned earlier in Section 10.1, the Evaluation stage depends on expert and organizational knowledge. Automating the vulnerability evaluation procedure is crucial for each organization because some vulnerabilities might remain unpatched in a system due to many vulnerabilities and the limited number of available security experts. We applied the following methods to define evaluation criteria and automate the evaluation stage to address the challenges mentioned earlier:

1. We reviewed the scientific papers on vulnerability patch prioritization to find evaluation criteria for ranking the vulnerabilities.

2. We interviewed security experts with different seniority levels in VRM to manually rank the criteria they are using to prioritize the vulnerability patch.

3. We analyzed the obtained criteria from items 1 and 2 to introduce a Patch Score(PS). PS is a mathematical approach to calculating the priority of each vulnerability from the evaluation criteria and their weight based on the organizational context.

In this section, we described our methods of finding the criteria in detail and how we can customize the PS in the organization's context.

9.5.1 Analysis of Vulnerability Evaluation Criteria in Literature

We conducted our search in google scholar because it is a comprehensive academic search engine with 389 million records [39]. The selected search string "vulnerability patch priority" was applied to identify the patch prioritization criteria in the relevant literature. The search query indicates that the string should include the title and abstract of a peer-reviewed publication. Then we excluded the papers that were not relevant to the goal of this paper based on the title and abstract. Finally, we did a full-text assessment of the fifteen selected papers.

From these fifteen papers, we identified nine criteria, reported in Table 9.2: a ✓sign means the criterion is considered in the paper. The related work

review shows that the severity score is a common criterion. In addition, ten of fifteen (66.7%) studies recognize the CVSS impact metrics, confidentiality, integrity, and availability as critical metrics in priority decisions. We also observed that eleven of fifteen (73.3%) papers identify the Exploitation Rate(similar to attack complexity in CVSS v3) as an essential criterion. The considered criteria in the related papers are described as follows:

- Severity Score (SC) is a transferring of the vulnerability characteristics to a numeric score between 0 to 10.
- Confidentiality (C) measures the impact of disclosure of the information to an unauthorized one due to a successfully exploited vulnerability.
- Integrity (I) refers to the impact of altering information by an unauthorized user on the trustworthiness of data due to a successfully exploited vulnerability.
- Availability (A) measures the impact of a successfully exploited vulnerability on the system and data accessibility.
- Age/time is a time difference between the CVE-ID published date and the current date.
- Common Configuration Enumeration (CCE) [40] is a unique identifier for system configuration issues and provides accurate configuration data across multiple tools and sources of information. CCE serves as a configuration best practice.
- Collateral Damage Potential (CDP) is an environmental metric in CVSS v.2 and refers to loss of life, physical assets, productivity, or revenue. Modified base metrics replaced CDP in CVSS v3 to reduce the impact of successfully exploiting the vulnerability by enforcing a change on the default configuration of a vulnerable component.
- Exploitation Rate (ER) provides the rate on how likely the vulnerability could be exploited. CVSS v3 addresses ER in the Attack Complexity (AC) metric, which evaluates the amount of effort to exploit the vulnerable component.
- Vulnerability Type (VT) refers to the attacker's activity as a result of successfully exploiting vulnerabilities such as Denial of Services (DoS), code execution, privilege escalation, and buffer overflow.

Table 9.2: *The evaluation criteria in related work*

Literature	Severity Score	C	I	A	Age/time	CCE	CDP	ER	VT
Al-Ayed et al. [29]	✓							✓	✓
Walkowski et al. [23]	✓	✓	✓	✓			✓		
Kamongi et al. [33]	✓	✓	✓	✓	✓			✓	
Araujo and Taylor [34]	✓					✓		✓	✓
Fruhwirth and Mannisto [18]	✓	✓	✓	✓				✓	
Patil and Modi [35]	✓	✓	✓	✓			✓	✓	
Lee et al. [36]	✓	✓	✓	✓	✓				✓
Angelini et al. [37]	✓				✓			✓	
Lin et al. [38]	✓	✓	✓	✓		✓	✓		
Li et al. [30]	✓	✓	✓	✓				✓	
Torkura et al. [31]	✓				✓				
Yadav et al. [24]	✓	✓	✓	✓				✓	
Olswang et al. [32]	✓							✓	✓
Jiang et al. [26]	✓	✓	✓	✓		✓		✓	
Shah et al. [25]	✓	✓	✓	✓	✓			✓	

9.5.2 Experts' Interview

We interviewed nine vulnerability management experts from governmental and private sectors located in USA and EU. The experts who participated in the study worked in the Information Technology domain with different levels of experience; a) three juniors who have less than two years of experience in VRM; b) three middle level who have from two to five years of experience in VRM; and c) three seniors who have more than five years of experience in VRM. We chose three different seniority levels as the response depends on knowledge and experience level [41]. The interview included two parts. In the first part, we interviewed the experts regarding the process they used to evaluate patch prioritization in their organization. In the second part, we asked the experts to rank the metrics in CVSS V3 and the accessibility level of the vulnerable AUS. The VRM experts have ranked the following criteria in the second part of the interview:

- Attack Vector (AV) is a CVSS V3 exploitability metric that refers to the context of the possibility of vulnerability exploitation(i.e., exploit vulnerability component from a network or locally)
- Attack Complexity (AC) is a CVSS V3 exploitability metric that defines the condition that must be existed in the environment to exploit the vulnerability. For example, if any security controls do not protect the vulnerable component, the attacker could successfully exploit the vulnerability with less effort.
- Privilege Requirements (PR) is a CVSS V3 exploitability metric that

describes the level of privilege an attacker must have to exploit the vulnerability successfully.

- User Interaction (UI) is a CVSS V3 exploitability metric that expresses the human intervention in the successful comprise of the vulnerable component.

- Confidentiality (C) is a CVSS V3 impact metric that measures the impact on the confidentiality of the source after a successful attack.

- Integrity (I) is a CVSS V3 impact metric that measures the impact on the integrity of the source after a successful attack.

- Availability (A) is a CVSS V3 impact metric that measures the impact on the availability of the source after successful exploitation.

- Severity Score (SC) is an output of CVSS that captures the technical characteristics of a component to a numeric score indicating the severity of the vulnerability.

- Internal AUS refers to the services that are not exposed to the public.

- External AUS refers to the services that are exposed to the public.

The interview was conducted in a virtual session on Microsoft Teams for around 60 minutes. Naturally, the number of experts in any domain is limited, which affects the number of available expert participants. Therefore, experts' participation in any study is lower than non-experts participants. Isenberg et al. [42] found the median number of expert participants in the study is nine in the survey of 113 papers.

The interviewees' ranked the criteria from one to five, where one is the lowest and five is the highest. From the interview, we calculate the statistics, including the minimum, maximum, average, and standard deviation of the expert's score in Table 9.3. In Figure 9.4 we show the individual experts' feedback. Looking at the statistics, we see that External AUS is the criterion with the highest average rating of 4.67. However, C, AV, I, A, and SC are also rated above 4.

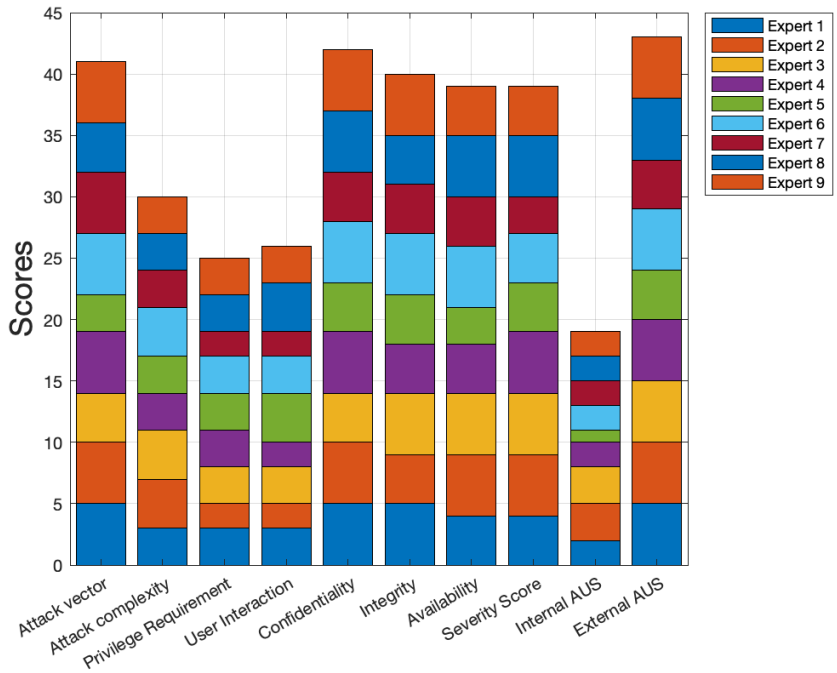

Figure 9.4: *Ranking of the criteria by security experts*

Table 9.3: *Statistics of the criteria ranked by security experts*

Statistics	AV	AC	PR	UI	C	I	A	SC	Internal AUS	External AUS
Minimum	3	3	2	2	4	4	3	3	1	4
Maximum	5	4	3	4	5	5	5	5	3	5
Average	4.56	3.33	2.78	2.89	4.67	4.44	4.33	4.33	2.11	4.78
Standard Deviation	0.68	0.47	0.42	0.74	0.47	0.50	0.67	0.67	0.57	0.42

9.5.3 Selected Criteria

We analyze the related work and expert interview results to identify criteria with an average score above 3 (e.g., above 60% of maximum scores by experts and above 60% of literature). Based on the results in Table 9.2 and Table 9.3, we chose the SC, C, I, A, AV, AC, and External AUS. The selected criteria, except External AUS, have defined metrics in the CVSS framework. Therefore, we can retrieve from the CVSS vector or the vulnerability description reported by VDs. Some VDs, such as NVD and RHSA, report the CVSS vector, and some, such as USN and DSA, use

similar keywords in the vulnerability description. In this study, we use the CVSSv3.1 vector to retrieve the score of selected criteria and calculate the patch score cf.9.5.4. The External AUS information the OC provides in access policy and group-based services.

9.5.4 Patch Score (PS)

ACVRM uses a Patch Score (PS) to determine the patch priority. PS is a scaling factor that could amplify the severity score of each CVE, and it is a function of the evaluation criteria as defined in Equation 9.1

$$PS_k = \sum_{i=1}^{n} w_i F_{k,i} + \begin{cases} 2 & \text{for AV=N and AC=L} \\ 1 & \text{for AV=N and AC=H} \\ 0 & \text{otherwise} \end{cases} \tag{9.1}$$

where: PS_k is the PS for vulnerability k; n is the number of evaluation criteria considered; w_i is a weight such that $w_i \in [0, 1]$, and $\sum_{i=1}^{n} w_i = 1$; $F_k = [F_{k,1}, ..., F_{k,n}]$ is the impact vector for vulnerability k. In this paper, we use $n = 6$ and $F_k = [SC_k, AV_k, AC_k, C_k, I_k, A_k]$. F_k can be easily expanded or reduced depending on the criteria considered.

Equation 9.1 is that it amplifies the PS for the vulnerabilities that could be exploited from network $AV_k = N$. The PS_k increases by additive factor $+2$ for the vulnerability k with the low complexity ($AC_k = L$). The PS_k raises by the additive factor $+1$ when the $AC_k = H$ is high. In all the other cases, i.e., the attack is not exposed from a network, no amplification is added.

To calculate a PS, we need to retrieve the weight vector from OC and the criteria vector from the CVSS vector in our local record. The organization could weight each criterion based on its importance and influence the PS value. The CVSS vector has been available since 2000 in NVD, and the chance of not having CVSS vector information is negligible. CVSS vector is a data string that captures the corresponding value for each CVSS metric. The CVSS vector, e.g.,

`CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:U/C:L/I:L/A:N`

starts with CVSS as a key and the version of CVSS (here:3.1) as a value. The forward slash is a delimiter between each metric. The abbreviation

Table 9.4: *CVSS v3.1 metric and the value used in ACVRM patch score[16]*

Metric Name	Metric Value	Numeric Value
Attack Vector (AV)	Network (N)	0.85
	Adjacent (A)	0.62
	Local (L)	0.55
	Physical (P)	0.2
Attack Complexity (AC)	Low (L)	0.77
	High (H)	0.44
Confidentiality (C)	None (N)	0
	Low (L)	0.22
	High (H)	0.56
Integrity (I)	None (N)	0
	Low (L)	0.22
	High (H)	0.56
Availability (A)	None (N)	0
	Low (L)	0.22
	High (H)	0.56

of each base metric is used as a key and separated from the abbreviated metric's value with a colon.

Table 9.4 presents the metrics' name and metrics' value and their abbreviation from CVSS v3.1 document[16]. The CVSS base metric groups consist of AV, AC, PR, UI, Scope (S), C, I, and A. We have excluded PR, UI, and S metrics as they have not been selected to consider in this study.

By expanding Equation 9.1 into the form used in this paper we get:

$$PS_k = w_1 SC_k + w_2 AV_k + w_3 AC_k + w_4 C_k + w_5 I_k + w_6 A_k + \begin{cases} 2 & \text{for } AV_k = N \text{ and } AC_k = L \\ 1 & \text{for } AV_k = N \text{ and } AC_k = H \\ 0 & \text{otherwise} \end{cases} \quad (9.2)$$

The PS value will be between 0 to 12. The maximum PS value could achieve when $SC = 10$ and $w1 = 1$ and AV=N and AC=L. The PS score could be zero when the highest weight is given to the metric that happens to be none.

9.6 Design and Implementation

This section describes the implementation of a PoC for the Prioritization phase (phase 2) of ACVRM, shown in Figure 9.5. It is designed as a group of functions split into four modules. Each module represents the implementation of each stage in the phase 2, and the output of each module is the input for the next one. We chose JSON as the internal data representation in this implementation since it is a supported format for most VDs and inventory tools.

Phase 1 PoC was described in [7]; hence we do not repeat it here. The output from it is a file `NF_output.json`. This contains the CVE-ID,

normalized scores for CVE, name of vulnerable AUS, safe version of AUS, and the severity score from one or multiple VDs.

The filter module matches the name of vulnerable AUS in the `NF_output.json` and the AUS information in the host inventory (`host_inventory.json`) to detect the organization's vulnerabilities. It also adds the vulnerabilities reported by the vulnerability scanner (`vunerability_scanner_report.json`) if that is not already identified in `host_inventory.json`. The organization could set a VMM in the VMM file as a default. The host-based VMM is an alternative for the organization if needed, and the VMM value should be added to the host inventory. The output of the filter stage is the `filter.json` that consists of CVE-ID, a normalized score for the CVE-ID, hosts name, and the vulnerable AUS.

The evaluation module assesses each entry (i.e., CVE-ID) in the `filter.json` based on the access policy and group host-based services to separate internal AUS and external AUS. Then, we implement the check on the selected criteria and their weight to calculate the PS for each CVE-ID. If the organization does not provide the weight vector, ACVRM will weight all criteria equally. The evaluation output is the `external_lst.json` and `internal_lst.json`, which refer to the vulnerabilities affecting internal and external AUS. Each list provides the CVE-ID, normalization score, CVSS vector, PS, host name and group (if applicable), the name of AUS, and the original severity score from VDs for each CVE-ID.

The sort module provides patch prioritization influenced by service priority. The `servicepriority.json` is a list of critical services for an organization's business. Hence, vulnerability remediation on those services should get the highest priority. If the organization does not have preferences, the patch prioritization will be based on the PS value computed from Eq. 9.1.

The output of phase 2 of ACVRM is the patch priority for each host in the inventory. The internal and external AUS are sorted separately, as the patch time might differ for each group. This output feeds into the patch management tool in the phase 3 of ACVRM.

9.7 Experimental Validation of PoC

The setup of the experimental environment to deploy and validate the ACVRM's PoC organize into four parts.

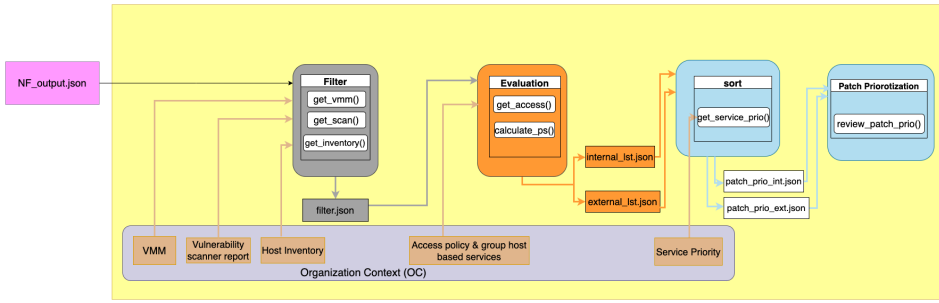

Figure 9.5: *High level software architecture of Phase 2 – Prioritization*

First, we collect the CVE-IDs data from VDs corresponding to phase 1, described in section 9.7.1. In the second stage, we create a virtual company. This company's organizational environment is characterized by a network of virtual servers deployed on a public cloud platform. This setup is shown in Figure 10.4; it consists of nine virtual servers (Ubuntu1-3, Debian1-3, CentOS1-3), one storage node (Local Storage), and one Rudder node, and one Nessus node. All nodes are connected to a switch. The servers are organized into three groups of three nodes each, where each group runs a different Linux distribution. Rudder node is a host running the *Rudder.io* manager version 6.2 [43] as an inventory tool. The Rudder manager receives the nodes' data through the installed Rudder agent on the nine virtual servers. Nessus node is a host running the Nessus [5] vulnerability scanner community edition, version 8.14.0-ubuntu110_amd64. The community edition of Nessus does not provide the CVE-ID of the detected vulnerabilities but instead reports the vulnerability with the Nessus ID. The report is generated in a limited format, such as HTML and CSV, and does not support Rest API. The report should be converted to JSON with the corresponding CVE-ID.

The nodes are created using the OS image provided by the cloud provider and then updated to the latest stable version. Table 9.5 describes the nodes specifications.

In the third stage, we deploy the Prioritization stage of ACVRM in our test environments to obtain the patch priority list for a given organization with four cases; each has different weight vectors. Finally, the fourth stage compares the output of ACVRM prioritization of each case with the `Rudder.io's` CVE plugin[44] results. CVE plugin is a VRM software developed by `Rudder.io` to identify and prioritize the vulnerabilities in

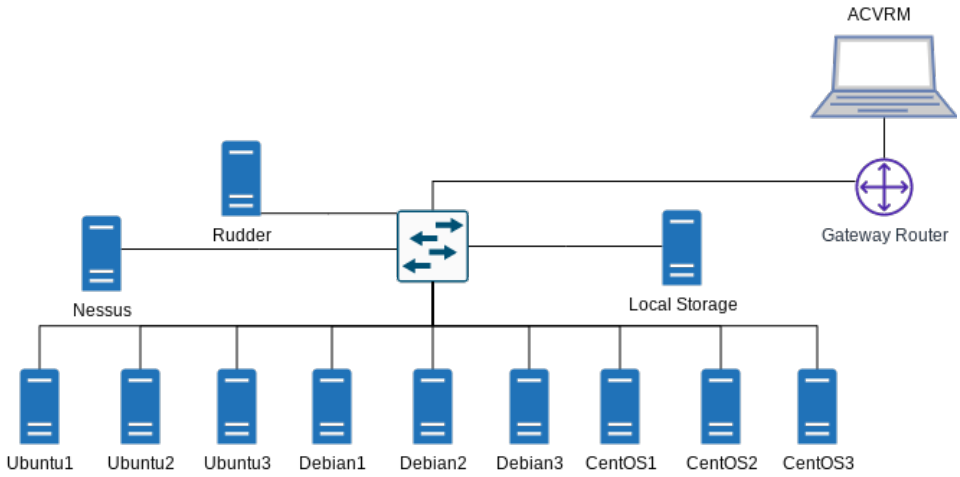

Figure 9.6: *The test environments*

installed software on each node managed by Rudder.

Table 9.5: *The specifications for the virtual servers in the test environment.*

Specification	Ubuntu	Debian	Centos	Rudder	Nessus	Storage
CPU	2			8	4	4
RAM	2GB			16GB	4GB	4GB
Storage	20GB			50GB	20GB	1TB
Distribution	18.04.4 LTS	9.1.1	8.1.1911	18.04.4 LTS	18.04.4 LTS	18.04.4 LTS
Kernel	4.15.0-158-generic	4.9.272-2	4.18.0-305.10.2.el8_4	4.15.0-158-generic	4.15.0-158-generic	4.15.0-158-generic
Nodes	3	3	3	1	1	1

9.7.1 Phase 1: Data collection and pre-processing

We collect CVE-IDs data from the four VDs described below, this is based on the existing OS in our experiment, and we add NVD as a reference:

- **RedHat Security Advisory (RHSA)** [9] is a subject-specific VD that provides the severity score based on the base and environmental metrics of CVSS v3.x. RHSA records the severity score for the CVE-IDs that effects RedHat's releases. RHSA reports the quantitative score and the severity rating based on the impact of the vulnerability in the RedHat environments.
- **Ubuntu Security Notices (USN)** [10] is a subject-specific VD that reports the CVE-IDs affected by Ubuntu's releases. USN developed its framework for calculating severity scores that are not publicly available. USN provides a qualitative severity score for each CVE-ID.

- **Debian Security Advisories (DSA)** [45] is a subject-specific VD that records the CVE-IDs affected by Debian's releases. DSA delivers a qualitative severity score, which relies on the NVD's score. It is not clear which version of CVSS applied to the DSA score.

- **National Vulnerability Database (NVD)** [8] provides a severity score based on base metric of CVSS v2.0 and v3.x framework. NVD is a generic VD and does not consider the environmental metrics in severity scores. NVD is one of the largest VD that records almost all existing CVE-IDs.

The data collected from the four VDs mentioned above are related to the CVEs affected Linux distributions from 2017 to 2021 for this study. These raw data are kept in our storage node in JSON format as a reference. The data is collected daily and archived in our local storage node, independent of other stages. This study is based on the information collected in June 2022.

9.7.2 Phase 2: Prioritization

Phase 2 implements the identification, classification, and evaluation processes of VRM. This phase automatically processed the data from phase 1 of ACVRM regarding OC. In our experiment, the Rudder node provides host inventory data, and the Nessus node generates the vulnerability scan report of the nine virtual servers. Then, we create a group of host-based services for the organization regarding the operating systems. One host from each host group is configured as an external AUS and exposed to the Internet. The rest virtual servers were set up as internal AUS. Finally, we assigned public IP addresses to the external AUS and configured an SMTP server on them. Our test organization does not prioritize services and operates on standard VMM but four different weight vectors. After preparing our test organization, we updated all nine virtual servers to patch all existing vulnerabilities via the Rudder CVE plugin. We scan with Nessus and run the CVE plugin in check mode to validate that vulnerabilities are successfully patched in servers.

We randomly selected 24 CVE-IDs (relevant to our virtual servers) and installed the vulnerable version on our nine virtual servers (Ubuntu1-3, Debian1-3, and CentOS1-3). Table 9.6 shows the selected CVE-IDs, the name of AUS, the severity score in each selected VDs in our experiment, the

CVSS vector for each CVE-ID, and the normalized score for each CVE-ID with standard VMM. We started our experiment by manually executing the Rudder and Nessus to determine the detected installed vulnerabilities. Then we run the code for phase 2 PoC. The stages in phase 2 are implemented in Python and comprise a group of functions. The functions' execution should be in order according to Figure 9.5 because the output of each stage is an input for the next one in the group. We keep the state of the virtual servers (e.g., with 24 installed vulnerabilities) unchanged and review the patch prioritization list provided by our tool for each case. We repeated the execution with standard VMM 50 times to verify that our codes gave the identical result. The execution time for phase 2 PoC was 7 minutes in our test environment. However, our focus in this experiment was on the accuracy of the patch prioritization rather than the execution time. In the future, we should investigate the relationship between execution time and the number of nodes.

Table 9.6: *The sample of vulnerabilities that affects our test environments*

CVE-ID	AUS name	SC RHSA	SC DSA	SC USN	SC NVD	CVSS vector	Normalized Score
CVE-2021-33574	glibc	5.9	high	low	9.8	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	6.4125
CVE-2021-3796	vim	7.3	medium	medium	7.3	CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:L/A:H	6.7500
CVE-2021-4192	vim	7.8	medium	medium	7.8	CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H	6.6250
CVE-2021-3778	vim	7.8	medium	medium	7.8	CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H	6.6250
CVE-2021-22555	kernel	7.8	medium	high	7.8	CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H	7.2500
CVE-2020-28374	kernel	8.1	medium	high	8.1	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N	7.4000
CVE-2021-4034	polkit	7.8	high	high	7.8	CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H	7.2500
CVE-2021-22946	curl	7.5	medium	medium	7.5	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.1000
CVE-2021-3712	openssl	7.4	medium	medium	7.4	CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:H	6.4250
CVE-2021-3449	openssl	5.9	medium	high	5.9	CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:H	6.3000
CVE-2021-41617	openssh	7.0	medium	low	7.0	CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:U/C:H/I:H/A:H	5.3625
CVE-2020-13776	systemd	6.7	medium	low	6.7	CVSS:3.1/AV:L/AC:H/PR:L/UI:R/S:U/C:H/I:H/A:H	5.2125
CVE-2021-33910	systemd	5.5	medium	high	5.5	CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:H	5.2375
CVE-2020-14308	grub2	6.4	medium	high	6.4	CVSS:3.1/AV:L/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H	6.5500
CVE-2021-30465	runc	7.5	medium	high	8.5	CVSS:3.1/AV:N/AC:H/PR:L/UI:N/S:C/C:H/I:H/A:H	7.3500
CVE-2021-20277	libldb	7.1	medium	high	7.5	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H	7.0000
CVE-2020-8831	apport	None	low	high	5.5	CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:N	5.1500
CVE-2020-8794	openSMTPD	None	high	high	9.8	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	8.5667
CVE-2021-3177	python	5.9	high	medium	9.8	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	7.2750
CVE-2021-20179	dogtag-pki	8.1	medium	high	8.1	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N	7.4000
CVE-2021-27135	xterm	9.6	high	medium	9.8	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	8.2000
CVE-2021-3156	sudo	7.8	high	high	7.8	CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H	7.8750
CVE-2020-11651	salt	9.8	high	medium	9.8	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	8.2500
CVE-2021-32760	containerd	5.5	medium	high	6.3	CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:L	6.3000

Table 9.7: *Comparing the PS value in different cases with Rudder*

CVE-ID	Rudder	SC	AV	AC	C	I	A	Add Factor	PS 1	PS 2	PS 3	PS 4
CVE-2021-33574	9.8	6.4125	0.85	0.77	0.56	0.56	0.56	2	3.6191	4.0956	8.4125	4.6874
CVE-2021-27135	9.8	8.2000	0.85	0.77	0.56	0.56	0.56	2	3.9171	4.5425	10.2000	5.3130
CVE-2021-3177	9.8	7.2750	0.85	0.77	0.56	0.56	0.56	2	3.7629	4.3113	9.2750	4.9893
CVE-2020-11651	9.8	8.2500	0.85	0.77	0.56	0.56	0.56	2	3.9254	4.5550	10.2500	5.3305
CVE-2020-8794	9.8	8.5667	0.85	0.77	0.56	0.56	0.56	2	3.9782	4.6342	10.5667	5.4413
CVE-2021-3796	8.8	6.7500	0.55	0.77	0.56	0.22	0.56	0	1.5686	2.0200	6.7500	2.6945
CVE-2021-30465	8.5	7.3500	0.85	0.44	0.56	0.56	0.56	1	2.7203	3.3300	8.3500	3.9825
CVE-2021-20179	8.1	7.4000	0.85	0.77	0.56	0.56	0	2	3.6903	4.3425	9.4000	4.9770
CVE-2020-28374	8.1	7.4000	0.85	0.77	0.56	0.56	0	0	3.6903	4.3425	9.4000	4.9770
CVE-2021-22555	7.8	7.2500	0.55	0.77	0.56	0.56	0.56	0	1.7087	2.2300	7.2500	2.9205
CVE-2021-4192	7.8	6.6250	0.55	0.77	0.56	0.56	0.56	0	1.6045	2.0738	6.6250	2.7018
CVE-2021-4034	7.8	7.2500	0.55	0.77	0.56	0.56	0.56	0	1.7087	2.2300	7.2500	2.9205
CVE-2021-3778	7.8	6.6250	0.55	0.77	0.56	0.56	0.56	0	1.6045	2.0738	6.6250	2.7018
CVE-2021-3156	7.8	7.8750	0.55	0.77	0.56	0.56	0.56	0	1.8129	2.3863	7.8750	3.1393
CVE-2021-22946	7.5	7.1000	0.85	0.77	0.56	0	0	2	3.5470	4.1275	9.1000	4.7880
CVE-2021-20277	7.5	7.0000	0.85	0.77	0	0	0.56	2	3.5303	3.9625	9.0000	4.7530
CVE-2021-3712	7.4	6.4250	0.85	0.44	0.56	0	0.56	1	2.4728	1.9588	7.4250	3.5748
CVE-2021-41617	7	5.3625	0.55	0.44	0.56	0.56	0.56	0	1.3390	1.7581	5.3625	2.2269
CVE-2020-13776	6.7	5.2125	0.55	0.44	0.56	0.56	0.56	0	1.3140	1.7206	5.2125	2.1744
CVE-2020-14308	6.4	6.5500	0.55	0.44	0.56	0.56	0.56	0	1.5370	2.0550	6.5500	2.6425
CVE-2021-32760	6.3	6.3000	0.85	0.77	0.22	0.22	0.22	2	3.4303	3.8975	8.3000	4.5290
CVE-2021-3449	5.9	6.3000	0.85	0.44	0	0	0.56	0	1.3586	1.7875	6.3000	2.4750
CVE-2021-33910	5.5	5.2375	0.55	0.77	0	0	0.56	0	1.1865	1.4469	5.2375	2.0761
CVE-2020-8831	5.5	5.1500	0.55	0.77	0	0.56	0	0	1.1719	1.5650	5.1500	2.0735

9.8 Results and discussions

This section analyzes the patch prioritization result generated by our tool for each case and Rudder CVE-Plugin. We used the PS value to prioritize the patch order. As described in Section 9.5.4, the PS value captures the important criteria in ranking the vulnerabilities. The organization could weight the criteria based on risk appetite to customize the patch prioritization. This study considers four cases with different weighting criteria to study patch prioritization with different risk appetites. Table 10.3 represents the numeric value of each criterion that involves in our PS calculation. The SC column is the normalized score for each CVE-ID where the VMM is standard. The Add Factor column in Table 10.3 represents the additive value in Equation 9.1. The PS1-4 in Table 10.3 are the PS value for the case1-4 respectively. The corresponding value of AV, AC, C, I, and A are from CVSS v3.1 and has two decimals. The result of our calculation, including SC, PS1, PS2, PS3, and PS4, are rounded with four decimals.

- Case 1: in this case, the organization weighs the criteria homogeneously in PS calculation as all six items are equally important for its business, i.e., $w_i = 1/N$, $\forall i$. The Equation 9.3 is expanded from Equation 9.2 for each CVE-ID, i.e., k.

$$PS_k = 0.1667(SC_k + AV_k + AC_k + C_k + I_k + A_k) + \begin{cases} 2 & \text{for } AV_k = N \text{ and } AC_k = L \\ 1 & \text{for } AV_k = N \text{ and } AC_k = H \\ 0 & \text{otherwise} \end{cases} \quad (9.3)$$

For example, the PS for $k = CVE - 2021 - 33574$ is calculated as:

$$PS_{CVE-2021-33574} = 0.1667(6.4125+0.85+0.77+0.56+0.56+0.56)+2$$

The PS1 column in Table 10.3 shows the result for case 1 by ACVRM for each CVE-ID that affected our test environments.

- Case 2: The organization does not consider AC and A as important criteria in this case. Hence, $w_3 = w_6 = 0$. However, organization weights the rest of criteria homogeneously in PS calculation, i.e., $w_1 = w_2 = w_4 = w_5 = w_6 = 0.25$ and $\sum_{i=1}^{6} w_i = 1$. The Equation 9.4 derived from Equation 9.2 for each CVE-ID, i.e., k in case 2.

$$PS_k = 0.25(SC_k + AV_k) + 0 * AC_k + 0.25(C_k + I_k) + 0 * A_k + \begin{cases} 2 & \text{for } AV_k = N \text{ and } AC_k = L \\ 1 & \text{for } AV_k = N \text{ and } AC_k = H \\ 0 & \text{otherwise} \end{cases} \quad (9.4)$$

For example, the PS for CVE-2021-33574 is calculated as:

$$PS_{CVE-2021-33574} = 0.25(6.4125+0.85)+0*0.77+0.25(0.56+0.56)+0*0.56+2$$

The result for case 2 by ACVRM presets in PS2 column in Table 10.3.

- Case 3: organization only considers the SC value for prioritizing the patch. Hence, the weight is distributed as $w_1 = 1$ and $w_2 = w_3 = w_4 = w_5 = w_6 = 0$. The Equation 9.8 is obtained from Equation 9.2 for each CVE-ID, i.e., k in case 3.

$$\mathrm{PS}_k = 1 * SC_k + 0 * (AV_k + AC_k + C_k + I_k + A_k) + \begin{cases} 2 & \text{for } AV_k = N \text{ and } AC_k = L \\ 1 & \text{for } AV_k = N \text{ and } AC_k = H \\ 0 & \text{otherwise} \end{cases}$$

(9.5)

For example, the PS for $k = CVE - 2021 - 33574$ is calculated as:

$$PS_{CVE-2021-33574} = 1*6.4125+0*(0.85+0.77+0.56+0.56+0.56)+2$$

Table 10.3 shows the outcome of case 3 for each CVE-ID in the PS3 column.

- Case 4: in this case organization weights all criteria based on its risk appetite. The weight distributed as $w_1 = 0.35$, $w_2 = 0.2$, $w_3 = w_4 = w_6 = 0.1$, and $w_5 = 0.15$ in PS calculation. The Equation 9.8 drives from equation 9.2 for each CVE-ID, i.e., k in case 4.

$$PS_k = 0.35 * SC_k + 0.2 * AV_k + 0.1(AC_k + C_k) + 0.15 * I_k + 0.1 * A_k + \begin{cases} 2 & \text{for } AV_k = N \text{ and } AC_k = L \\ 1 & \text{for } AV_k = N \text{ and } AC_k = H \\ 0 & \text{otherwise} \end{cases} \quad (9.6)$$

For example, the PS for $k = CVE - 2021 - 33574$ is calculated as:
$\text{PS}_{CVE-2021-33574} = 0.35 * 6.4125 + 0.2 * 0.85 + 0.1(0.77 + 0.56) + 0.15 * 0.56 + 0.1 * 0.56 + 2$
The result of PS for case 4 presented in column PS4 in Table 10.3 for the CVE-IDs affected our test environments.

9.8.1 Analysing the patch prioritization

In this section, we compare the patch prioritization offered by Rudder's CVE-plugin and four cases of ACVRM. Table 9.8 presents the patch order for different cases. Rudder CVE-plugin provided patch prioritization based on the SC from general purpose VD and NVD and does not reflect the organization's context.

We define Δ as the difference between the position in patch priority ($P_{k,*}$) between Rudder and ACVRM cases as:

$$\Delta_k = P_{\text{k,rudder}} - P_{\text{k,ACVRM}} \quad (9.7)$$

where k is CVE-ID. The Δ column after each case in Table 9.8 shows the changes in the CVE-ID position compared with the Rudder CVE-plugin. For example, the CVE-2021-33574 has a priority 1 by Rudder while it becomes priority seven in case 1 and priority eight in case 2. The Δ value with a negative sign means the position of the CVE-ID moves down (lower priority). In contrast, the positive value means the place of CVE-ID moves up (higher priority) in the priority list. The Δ is zero when the position of the CVE-ID is the same in the priority list provided by Rudder and ACVRM case. The CVE-ID with priority one will be patched first, and the CVE-ID with priority twenty-four in our list will be the last to patch.

We observed that the CVE-IDs that could be exploited from networks with a low attack complexity gain higher priority (e.g., priority 1-11) by

Table 9.8: *Patch priority list in test environments by Rudder CVE-plugin and ACVRM*

	Rudder	Case 1		Case 2		Case 3		Case 4	
Priority	CVE-ID	CVE-ID	Δ	CVE-ID	Δ	CVE-ID	Δ	CVE-ID	Δ
1	CVE-2021-33574	CVE-2020-8794	4	CVE-2020-8794	4	CVE-2020-8794	4	CVE-2020-8794	4
2	CVE-2021-27135	CVE-2020-11651	2	CVE-2020-11651	2	CVE-2020-11651	2	CVE-2020-11651	2
3	CVE-2021-3177	CVE-2021-27135	-1	CVE-2021-27135	-1	CVE-2021-27135	-1	CVE-2021-27135	-1
4	CVE-2020-11651	CVE-2021-3177	-1	CVE-2020-28374	5	CVE-2020-28374	5	CVE-2021-3177	-1
5	CVE-2020-8794	CVE-2020-28374	4	CVE-2021-20179	3	CVE-2021-20179	3	CVE-2020-28374	4
6	CVE-2021-3796	CVE-2021-20179	2	CVE-2021-3177	-3	CVE-2021-3177	-3	CVE-2021-20179	2
7	CVE-2021-30465	CVE-2021-33574	-6	CVE-2021-22946	8	CVE-2021-22946	8	CVE-2021-22946	8
8	CVE-2021-20179	CVE-2021-22946	7	CVE-2021-33574	-7	CVE-2021-20277	8	CVE-2021-20277	8
9	CVE-2020-28374	CVE-2021-20277	7	CVE-2021-20277	7	CVE-2021-33574	-8	CVE-2021-33574	-8
10	CVE-2021-22555	CVE-2021-32760	11	CVE-2021-32760	11	CVE-2021-30465	-3	CVE-2021-32760	11
11	CVE-2021-4192	CVE-2021-30465	-4	CVE-2021-30465	-4	CVE-2021-32760	10	CVE-2021-30465	-4
12	CVE-2021-4034	CVE-2021-3712	5	CVE-2021-3156	2	CVE-2021-3156	2	CVE-2021-3712	5
13	CVE-2021-3778	CVE-2021-3156	1	CVE-2021-4034	-1	CVE-2021-3712	4	CVE-2021-3156	1
14	CVE-2021-3156	CVE-2021-4034	-2	CVE-2021-22555	-4	CVE-2021-4034	-2	CVE-2021-4034	-2
15	CVE-2021-22946	CVE-2021-22555	-5	CVE-2021-3778	-2	CVE-2021-22555	-5	CVE-2021-22555	-5
16	CVE-2021-20277	CVE-2021-3778	-3	CVE-2021-4192	-5	CVE-2021-3796	-10	CVE-2021-3778	-3
17	CVE-2021-3712	CVE-2021-4192	-6	CVE-2020-14308	3	CVE-2021-3778	-4	CVE-2021-4192	-6
18	CVE-2021-41617	CVE-2021-3796	-12	CVE-2021-3796	-12	CVE-2021-4192	-7	CVE-2021-3796	-12
19	CVE-2020-13776	CVE-2020-14308	1	CVE-2021-3712	-2	CVE-2020-14308	1	CVE-2020-14308	1
20	CVE-2020-14308	CVE-2021-3449	2	CVE-2021-3449	2	CVE-2021-3449	2	CVE-2021-3449	2
21	CVE-2021-32760	CVE-2021-41617	-3	CVE-2021-41617	-3	CVE-2021-41617	-3	CVE-2021-41617	-3
22	CVE-2021-3449	CVE-2020-13776	-3	CVE-2020-13776	-3	CVE-2021-33910	1	CVE-2020-13776	-3
23	CVE-2021-33910	CVE-2021-33910	0	CVE-2020-8831	1	CVE-2020-13776	-4	CVE-2021-33910	0
24	CVE-2020-8831	CVE-2020-8831	0	CVE-2021-33910	-1	CVE-2020-8831	0	CVE-2020-8831	0

ACVRM compared with Rudder (e.g., priority 1-21). The priority position in Table 9.8 shows that only five CVE-IDs (e.g., CVE-2020-8794, CVE-2020-11651, CVE-2021-27135, CVE-2021-3449, CVE-2021-41617) got the same priority in ACVRM cases.

After reviewing the Δ value, we found that CVE-2020-8831 obtains priority twenty-four in Rudder and ACVRM cases 1, 3, and 4. Case 2 of ACVRM does not have any similar priority position compared with Rudder. We also noticed that thirteen CVE-IDs achieved a lower priority position in Case 2 of ACVRM, while the number of CVE-IDs is eleven in other cases. In addition, we visualized the change in each vulnerability position for all cases in Figure 9.7. As shown in the Figure 9.7, the position of five vulnerabilities (e.g., CVE-2021-33574, CVE-2021-27135, CVE-2020-8794, CVE-2021-4034, CVE-2020-14308) rise for all four cases where three of them (e.g., CVE-2021-33574, CVE-2021-27135, CVE-2020-14308) increased the same order. We also observed that the position of six vulnerabilities (e.g., CVE-2021-3177, CVE-2021-3156, CVE-2021-22946, CVE-2021-20277, CVE-2021-41617, CVE-2021-32760) decreased in the priority list in all four cases while two of them (e.g., CVE-2021-32760, CVE-2021-3177) dropped exact the same order.

We also noticed that Rudder listed the CVE-IDs with the same SC randomly (e.g., the priority 1 to 5 have a $SC = 9.8$, and the position is not related to the age of the CVE-IDs). However, ACVRM considered

the age of the CVE-IDs in prioritization when the PS value is equal (e.g., CVE-2020-28374 and CVE-2021-20179 in case 1 have the same PS value but the CVE-2020-28374 gains the higher priority as it has been known publicly for a longer time).

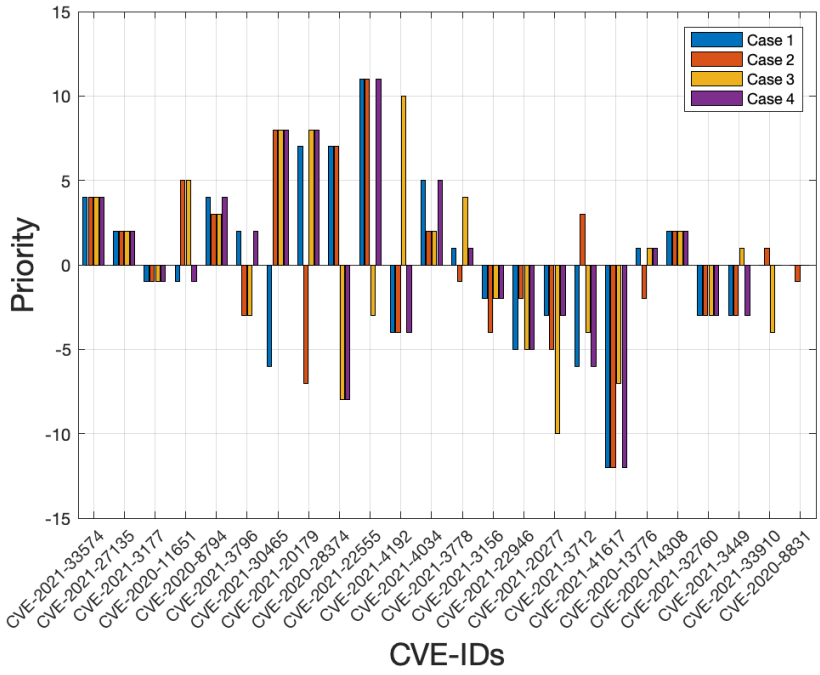

Figure 9.7: *The Δ value of patch priority orders in Rudder CVE-plugin and Case 1-4 of ACVRM*

9.9 Conclusion

The increasing number of publicly known vulnerabilities introduces the challenge in VRM as classification and evaluation phases need experts' intervention. The security experts should evaluate the vulnerability risk for the organization and define patch prioritization, which is time-consuming and resource-intensive. Therefore, we need to improve VRM to address the challenges mentioned earlier. We introduce ACVRM to automate the VRM procedure and reduce the experts' intervention. Hence, we need to learn how experts evaluate and prioritize the patch. In this study, we focus on

the classification and evaluation process of VRM in the context of a given organization. We performed an analysis in three phases as follows:

1. We conducted a literature study and experts interview to learn which criteria play a role in patch prioritization. We defined selected criteria for patch prioritization based on the result obtained in our study. We found that the security score, attack vector, attack complexity, confidentiality, integrity, and availability values and exposing the level of the AUS are essential in deciding the patch order. Therefore, we define the PS based on the selected criteria and the possibility of weighting each criterion in the organization's context.

2. We designed and implemented phase 2 of ACVRM, which consists of four modules; filter, evaluation, sort, and patch prioritization. We created the environments of the test organization in the public cloud. The experiment was executed for four cases where each case's criteria were weighted differently.

3. We verified the result of our phase 2 implementation by analyzing the outcome of each case. We also compared the patch prioritization of our tool with the Rudder CVE-plugin. Our result shows that the ACVRM could adjust the patch prioritization for each organization with less effort from security experts. The security experts only set the VMM and weight the selected criteria. Our solution also allows the security experts to add more criteria to the evaluation module if needed.

Our study showed how the organization could customize the patch priority based on its context by selecting VMM mode and weighting the criteria. We have presented the improvement in the VRM procedure by reducing evaluation time and experts' intervention. The execution time of the phase 2 was seven minutes in our test environment, including four modules(e.g., filter, evaluation, sort, and patch prioritization). However, the execution time needs to be studied further.

In the future, we want to continue the implementation of phase 3 of ACVRM and address the challenges in patch management, including automated validation of the patch deployment, verification of the side effects of patching vulnerabilities, and the possibility of a generalized verification process. Another possible future direction could be using a machine learning

algorithm to improve patch prioritization based on the patch verification feedback. Finally, we could investigate the time efficiency of our solution and compare the patch prioritization of our proposed solution with the recently published state of art approaches.

References

[1] *Top Routinely Exploited Vulnerabilities*. https://www.cisa.gov/uscert/ncas/alerts/aa22-117a. [Online; accessed 12-June-2022].

[2] *Costs and Consequences of Gaps in Vulnerability Response*. https://www.servicenow.com/lpayr/ponemon-vulnerability-survey.html. [Online; accessed 26-August-2022].

[3] *VULNERABILITY AND THREAT TRENDS Report 2021*. Tech. rep. SkyBox Security, 2021.

[4] *Open Vulnerability Assessment Scanner(OpenVAS)*. https://www.openvas.org/. [Online; accessed 18-October-2022].

[5] *Nessus Vulnerability Scanner*. https://www.tenable.com/products/nessus. [Online; accessed 18-October-2022].

[6] V. Ahmadi, P. Arlos, and E. Casalicchio. "Normalization of Severity Rating for Automated Context-aware Vulnerability Risk Management". In: *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. IEEE. 2020, pp. 200–205.

[7] V. Ahmadi, P. Arlos, and E. Casalicchio. "Normalization Framework for Vulnerability Risk Management in Cloud". In: *2021 IEEE International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE. 2021.

[8] *NIST National Vulnerability Database*. https://nvd.nist.gov/. [Online; accessed 15-October-2022].

[9] *RedHat Security Advisories*. https://access.redhat.com/security/security-updates/. [Online; accessed 10-October-2022].

[10] *Ubuntu Security Notice*. https://usn.ubuntu.com/. [Online; accessed 8-September-2022].

[11] *Apache Security Information*. https://www.apache.org/security/projects.html. [Online; accessed 16-September-2022].

[12] *CIS Controls*. http://www.cisecurity.org/controls/. [Online; accessed 5-April-2020].

[13] *EU Cybersecurity Act*. https://eur-lex.europa.eu/eli/reg/2019/881/oj. [Online; accessed 11-October-2022].

[14] *European Cybersecurity Certification Scheme for Cloud Services*. https://www.enisa.europa.eu/publications/eucs-cloud-service-scheme. [Online; accessed 11-February-2021].

[15] *Homland Security Act 2002*. https://www.dhs.gov/homeland-security-act-2002. [Online; accessed 15-October-2022].

[16] F. Inc. "Common Vulnerability Scoring System v3.1: Specification Document". In: (2019).

[17] G. Spanos, A. Sioziou, and L. Angelis. "WIVSS: a new methodology for scoring information systems vulnerabilities". In: *Proceedings of the 17th panhellenic conference on informatics*. ACM. 2013.

[18] C. Fruhwirth and T. Mannisto. "Improving CVSS-based vulnerability prioritization and response with context information". In: *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE. 2009.

[19] *Cyber Security Report 2021 by check point research*. https://www.checkpoint.com/downloads/resources/cyber-security-report-2021.pdf. [Online; accessed 16-October-2022].

[20] F. Zhang, P. Huff, K. McClanahan, and Q. Li. "A Machine Learning-based Approach for Automated Vulnerability Remediation Analysis". In: *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2020.

[21] M. Aota, H. Kanehara, M. Kubo, N. Murata, B. Sun, and T. Takahashi. "Automation of Vulnerability Classification from its Description using Machine Learning". In: *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2020.

[22] X. Wang, S. Wang, K. Sun, A. Batcheller, and S. Jajodia. "A Machine Learning Approach to Classify Security Patches into Vulnerability Types". In: *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2020.

[23] M. Walkowski, M. Krakowiak, M. Jaroszewski, J. Oko, and S. Sujecki. "Automatic CVSS-based vulnerability prioritization and response with context information". In: *2021 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE. 2021.

[24] G. Yadav, P. Gauravaram, A. K. Jindal, and K. Paul. "SmartPatch: A patch prioritization framework". In: *Computers in Industry* (2022).

[25] A. Shah, K. A. Farris, R. Ganesan, and S. Jajodia. "Vulnerability selection for remediation: An empirical analysis". In: *The Journal of Defense Modeling and Simulation* 19.1 (2022), pp. 13–22.

[26] Y. Jiang and Y. Atif. "Towards automatic discovery and assessment of vulnerability severity in cyber–physical systems". In: *Array* (2022).

[27] *Common Vulnerabilities and Exposures(CVE)*. `https://cve.mitre.org/`. [Online; accessed 18-October-2022].

[28] *Cloud Computing Compliance Criteria Catalogue (C5)*. `https://www.bsi.bund.de/EN/Topics/CloudComputing/Compliance_Criteria_Catalogue/Compliance_Criteria_Catalogue_node.html`. [Online; accessed 18-October-2022].

[29] A. Al-Ayed, S. Furnell, D. Zhao, and P. Dowland. "An automated framework for managing security vulnerabilities". In: *Information management & computer security* (2005).

[30] Z. Li, C. Tang, J. Hu, and Z. Chen. "Vulnerabilities Scoring Approach for Cloud SaaS". In: *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*. IEEE. 2015.

[31] K. A. Torkura, F. Cheng, and C. Meinel. "A proposed framework for proactive vulnerability assessments in cloud deployments". In: *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*. IEEE. 2015.

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

[33] F. Zhang and Q. Li. “Dynamic Risk-Aware Patch Scheduling”. In: *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2020.

[34] F. Araujo and T. Taylor. “Improving cybersecurity hygiene through JIT patching”. In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM. 2020.

[35] R. Patil and C. Modi. “Designing an efficient framework for vulnerability assessment and patching (VAP) in virtual environment of cloud computing”. In: *The Journal of Supercomputing* (2019).

[36] J.-H. Lee, S.-G. Sohn, B.-H. Chang, and T.-M. Chung. “PKG-VUL: Security Vulnerability Evaluation and Patch Framework for Package-Based Systems”. In: *ETRI journal* (2009).

[37] M. Angelini, G. Blasilli, T. Catarci, S. Lenti, and G. Santucci. “Vulnus: Visual vulnerability analysis for network security”. In: *IEEE transactions on visualization and computer graphics* (2018).

[38] C.-H. Lin, C.-H. Chen, and C.-S. Laih. “A study and implementation of vulnerability assessment and misconfiguration detection”. In: *2008 IEEE Asia-Pacific Services Computing Conference*. IEEE. 2008.

[39] M. Gusenbauer. “Google Scholar to overshadow them all? Comparing the sizes of 12 academic search engines and bibliographic databases”. In: *Scientometrics* (2019).

[40] *Common Configuration Enumeration(CCE)*. `https://ncp.nist.gov/cce/index`. [Online; accessed 8-September-2022].

[41] D. C. Zhang and Y. Wang. “An empirical approach to identifying subject matter experts for the development of situational judgment tests.” In: *Journal of Personnel Psychology* (2021).

[42] T. Isenberg, P. Isenberg, J. Chen, M. Sedlmair, and T. Möller. “A systematic review on the practice of evaluating visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* (2013).

[43] *Rudder*. `https://www.rudder.io/`. [Online; accessed 2-September-2022].

[44] *Rudder CVE Plugin*. `https://docs.rudder.io/reference/6.2/plugins/cve.html`. [Online; accessed 16-October-2022].

[45] *Debian Security Tracker*. https://www.debian.org/security/#DSAS. [Online; accessed 29-September-2022].

10

Automated Patch Management: An Empirical Evaluation Study

Vida Ahmadi Mehri, Patrik Arlos, Emiliano Casalicchio

Accepted to publish as:

Ahmadi Mehri, V., Arlos, P., and Casalicchio, E. "Automated Patch Management: An Empirical Evaluation Study", 2023 IEEE International Conference on Cyber Security and Resilience (CSR), Venice, Italy, 2023.

Abstract: Vulnerability patch management is one of IT organizations' most complex issues due to the increasing number of publicly known vulnerabilities and explicit patch deadlines for compliance. Patch management requires human involvement in testing, deploying, and verifying the patch and its potential side effects. Hence, there is a need to automate the patch management procedure to keep the patch deadline with a limited number of available experts. This study proposed and implemented an automated patch management procedure to address mentioned challenges. The method also includes logic to automatically handle errors that might occur in patch deployment and verification. Moreover, the authors added an automated review step before patch management to adjust the patch prioritization list if multiple cumulative patches or dependencies are detected. The result indicated that our method reduced the need for human intervention, increased the ratio of successfully patched vulnerabilities, and decreased the execution time of vulnerability risk management.

10.1 Introduction

Vulnerability Risk Management (VRM) is one of the critical aspects of information security that has been ranked in the top 10 by CIS [1]. Unpatched vulnerabilities expose organizations and individuals to cyber attacks. One well-known example is the Log4j (i.e., CVE-2021-44228, CVE-2021-45046, CVE-2021-45105, and CVE-2021-44832) which was one of the most severe threats in recent years due to the number of vulnerable systems and the ease of exploit (i.e., ten million attempts per hour [2]). VRM is a cyclic process that aims to identify, classify, evaluate, and remediate vulnerabilities and reduce an organization's attack surface. Currently, VRM is challenging due to the dramatic increase of known vulnerabilities (i.e., 40% in 2019-2022) and the explicit patch deadline enforced by regulation for public sectors. For instance, the patch deadline for federal agencies in the U.S. is 15 days for critical vulnerabilities and 30 days for vulnerabilities with high severity. For UK officials, 14 days for critical vulnerabilities[1,2]. According to the NIST National Vulnerability Database (NVD) [3], 57.69% of the new vulnerabilities reported in 2022 ranked with critical and high severity (40.61% high severity and 17.08% critical severity). Hence, automated VRM is vital to support security specialists in keeping the patch deadline. Today, we lack the tools that automatically conduct the four stages, identification, classification, evaluation, and remediation in the VRM process. In [4–6], we proposed Automated Context-aware Vulnerability Risk Management (ACVRM) to improve the VRM procedure by 1) reducing the labor-intensive tasks of security experts in patch prioritization; 2) customizing the patch prioritization for a given organization by learning about the organization's assets and the vulnerabilities that affect these assets; 3) automating VRM procedure to reduce processing time.

ACVRM consists of three phases, cf. Figure 10.1. In phase 1, described in [4, 6], we collected the publicly known vulnerabilities from multiple Vulnerability Databases (VD) and normalized the severity score for each vulnerability based on the selected vulnerability management mode by the organization. The collected data identifies existing vulnerabilities in the organization's assets. In phase 2 described in [5], we automatically scan the organization's asset inventory against the collected data to detect the

[1] https://www.ncsc.gov.uk/cyberessentials/overview
[2] https://www.cisa.gov/binding-operational-directive-19-02

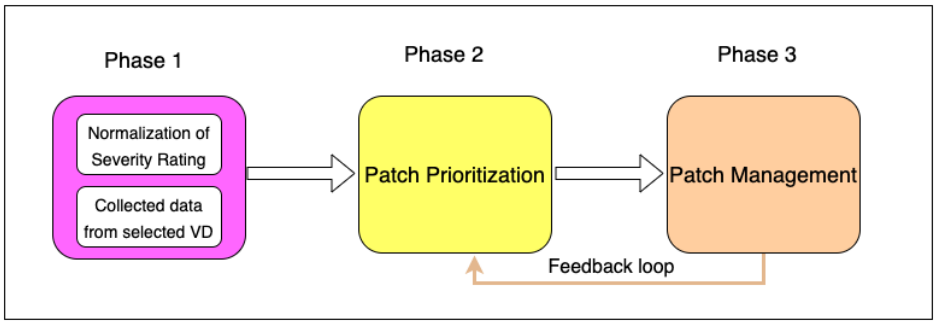

Figure 10.1: *Relationship between our current work (Phase 3 and feedback loop to phase 2) and previous work (Phase 1 & 2)*

vulnerabilities that affect the organization. We defined the criteria to be considered in patch prioritization via literature study and experts interview. The criteria weighted by security experts are based on the organization's policy and risk appetite to calculate the patch score. We use the patch score to determine the patched vulnerability's order.

This study focuses on the third phase, patch management, and the feedback loop to improve patch prioritization. Patch management is a process to mitigate the vulnerability in the organizations' assets by deploying and verifying the patch. Patch Management (PM) is one of the most complex processes in information technology, as it requires a strong understanding of the system components and the potential patch. It is also challenging due to the uncertainty of the system's reaction to the patch and the problem with the patch released by vendors (i.e., the patched version of one vulnerability introduces another vulnerability) [7]. For example, Microsoft's security patch addressed Meltdown and Spectre vulnerabilities, which are hardware vulnerabilities that affect nearly every computer processor, causing some computers to become unbootable [8].

In this paper, we design and implement phase 3 of ACVRM, PM, and show the capability of the patch feedback loop to improve prioritization for the given organization. Our solution was deployed on a test environment where we applied our method to patch software vulnerabilities.

The paper is organized as in what follows. Section 10.2 provides background on patch management and analyzes the related literature. Section 10.3 describes our contribution, and Section 10.4 introduces the ACVRM's patch management phase, where the paper's core contribution is.

Section 10.5 describes the design and implementation of a proof of concept for patch management and feedback loop. Experiments and results are reported in Sections 10.6 and 10.7, respectively. Section 10.8 concludes the paper.

10.2 Background and related work

Patch Management (PM) helps organizations keep their assets secure, reliable, and up-to-date with the required features and functionality. It is also essential for ensuring compliance with security and privacy regulations such as EU Cybersecurity act [9], EU Cybersecurity Certificate (EUCS) [10], USA homeland security act [11]. Security patches (hereafter patch) are released by hardware or software vendors to address the identified vulnerabilities in their products. PM procedure is responsible to remediate the vulnerabilities in the organization's environment. The PM generally consists of three steps [12] to address vulnerabilities in the organization:

- Patch testing tests the patch on an isolated system similar to the production to verify the impact on system/software performance or instability.
- Patch deployment is applying a patch in production environments.
- Post-deployment or patch verification is an activity to detect malfunctions or instability on the system/software post-patching vulnerability.

One of the complex issues in PM is verifying the vulnerability patches in an organization environment as each organization has its unique combination of system and configuration [12]. The patch verification answers two questions 1) Does the patch remediate the vulnerability? 2) Does the patch have side effects (i.e., does not break any other software, application, or system)? The first could be verified with vulnerability scanning or checking the version of the software or application, or system. However, the second one required a deep understanding of the architectural design of environments (e.g., system, application, software) to evaluate the impact of the patch and prepare a rollback plan [13].

Some studies [14–19] have reported the need for human expertise in PM due to the increased complexity of security patching and the limitations

of the current technologies to provide solutions covering the entire process. However, the authors in [14, 18, 20–23] highlighted the significant gap in the required skills and knowledge expertise in PM. The experts' involvement in the PM procedure increases the time to patch [14]. Therefore, keeping up with the time to patch became a challenge in IT due to the increasing number of publicly known vulnerabilities. For example, the number of vulnerabilities in 2022 is 25064 (i.e., average 68 vulnerabilities per day) where 14461 (i.e., average 40 vulnerabilities per day) of them ranked as critical and high presented in Table 10.1. Given that each vulnerability might affect N number of assets in the organization, the time required for patching each vulnerability depends on the availability of the security resources [14, 17, 24, 25]. According to [26], the mean time to remediation of the vulnerability varies in different industries (e.g., 44 days in healthcare and 92 days in public administration). An automated PM facilitates the learning of organizational context, which is an inevitable need for each organization to remain secure and compliant. Our proposed solution, ACVRM, addresses this need. ACVRM introduced the feedback loop and learned from the historical event in PM.

The authors in [16] determined that the downtime of the business critical system is a major obstacle in vulnerability patching. It also identified the struggle of the system administrators to verify the dependencies in complex applications and systems. ACVRM facilitates the creation of a dependency tree for each software and service in an organization. The dependency reflects the patch prioritization list to support the system administrator. Midtrapanon and Wills proposed automated patch management for Linux-based servers. They orchestrated the existing open-source tools to deploy the patch automatically and simultaneously for many servers. The authors claimed the set-up time was reduced and the tool was cost-efficient [18]. Our work proposed automating the entire VRM procedure for organizations independently of the platform. We improve time to patch by automated vulnerability assessment and patch prioritization in the organization's context. We also address the patch verification.

10.3 Contribution

In our previous work [5], we designed and implemented phase 2 of ACVRM, patch prioritization without a feedback loop from the patch verification task

Table 10.1: *Statistic report on the number of reported Vulnerabilities in NVD [27]*

Year	No. vulnerability	Critical	High
2019	17305	2640	7243
2020	18351 (+6%)	2720 (+3%)	7708 (+6%)
2021	20158 (+10%)	2677 (-2%)	8553 (+11%)
2022	25064 (+24%)	4282 (+60%)	10179 (+19%)

in phase 3. Moreover, the review prioritization block (cf. Figure 10.2) used only the patch score (i.e., the score is calculated by weighting the selected criteria in the context of the organization for each vulnerability) to determine the patch order. We discovered our method in [5] needs improvement because historical data and practical experience are the inevitable factors to increase the success rate of patch[28]. We also learned from the practical experience shared in the patch management community of experts[3], some patch failures could be avoided by prioritization (e.g., patching a microcode vulnerability in Ubuntu requires the kernel to be patched in advance; otherwise, the mitigation failed). In this study, therefore, we design and implement a proof of concept of phase 3 and improve patch prioritization in phase 2. We enhance the task for review prioritization in phase 2 through a feedback loop, which facilitates review and error handling capability based on the history of events. The feedback loop introduced a learning opportunity to adjust the prioritization to reduce the patch failure rate. The feedback loop helps the organization to capture the knowledge in the organization. Hence, it reduces the experts' intervention by automating the response to the known patch failure, which is one of the goals for ACVRM.

10.4 ACVRM Phase 3: Patch Management

Phase 3 and its coordination with phase 2 in handling the patches present in Figure 10.2. The stages in phase 3 are divided into two parts. First, patch prioritization applies in test environments similar to production. If the patch is successful, the second part will be initiated. The stages in the phase 3 describe briefly as follows:

- *Patch testing* is the first stage in PM, where the patch priority list will be deployed in a test environment. This stage aims to determine if a patch will cause problems for an organization's unique combination of

[3] http://patchmanagement.org/

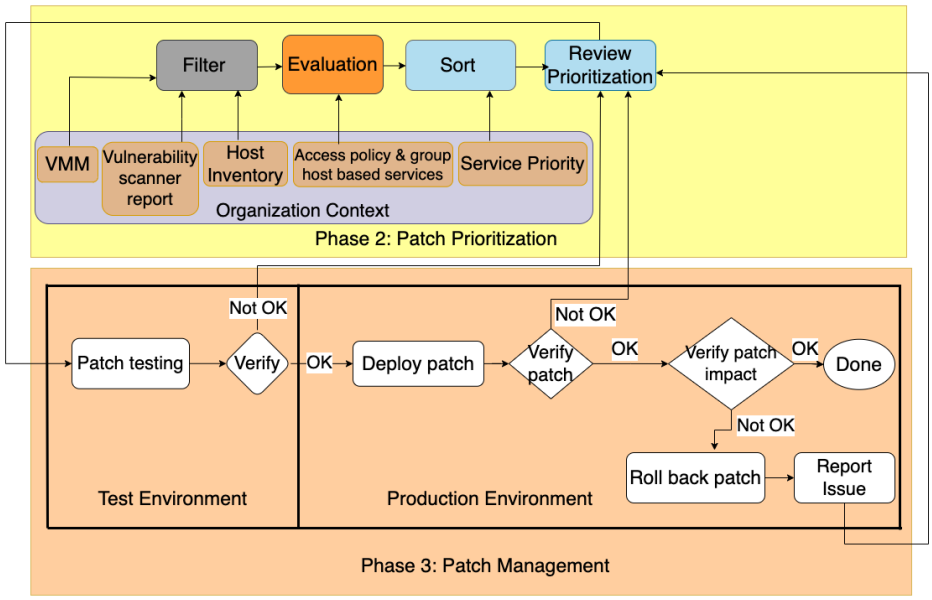

Figure 10.2: *ACVRM Phase 2 and 3*

hardware, software, and configuration settings. The test environment should be created similarly to the production environments. In this stage, the vulnerabilities in the patch prioritization list for the selected host will be patched sequentially in a test host for better visibility of potential issues.

- *Verify testing* is a stage to identify if the patches were successful in a test environment. The output of the patch testing will be captured and reviewed for success or failure. In case of an error, the output of patch execution will be sent to the Review Prioritization stage. If no error is reported by executing the patch in the test host, it verifies the remediation of the vulnerability. If the patch does not mitigate the vulnerability in a tested host, an alert will be sent to the Review Prioritization stage.

- *Verify patch impact* aims at detecting any side effect of the patch on the functionality of the services or applications in the organization. This stage consists of predefined functional tests to identify unexpected behavior of the organization systems or applications. If all tests were successful, the process is documented as successfully done.

- *Rollback patch* is a stage for returning the system to the prior state of patching. If the vulnerability patch causes an issue in the system or application functionality or behavior, the first response to address the issue is to roll back the patch. It brings the system or application to the latest working state before patch deployment.

- *Report issue* is an alert to experts for unexpected behavior. The report is sent to the Review Prioritization stage in phase 2, where the security and system experts should investigate the root cause and apply the lesson learned to the next patch prioritization. The report includes the CVE-ID, name of vulnerable software or service, hostname, error message, and time stamp.

- Success test is a step in the production environment to ensure all the vulnerabilities have been tested successfully in a test environment.

- Deploy patch is a stage similar to patch testing for patching vulnerabilities in a production environment. This stage depends on the output of the previous stage, test verification. If the patch is successful in a test environment, it will deploy automatically in production.

- Verify patch is a stage similar to the verify testing for the host in production environments. It verifies the status of the remediation in a production environment.

As we described above, all the errors or feedback loops return to the Review Prioritization stage in the phase 2, where the patch prioritization list is reviewed for the organization. All the stages could be automated for the organization, but any error needs experts' involvement. We consider the feedback loop a learning opportunity as the reported issues are recorded in a database for organizations to identify the challenges in their unique system and address them automatically. We expected the learning to improve the patch prioritization decision over time and reduce the number of errors, decreasing the patch failure rate and experts' intervention.

10.5 Design and implementation

The implementation of a PoC for the PM phase (phase 3) and Review Prioritization of phase 2, shown in Figure 10.3, is designed as a group of

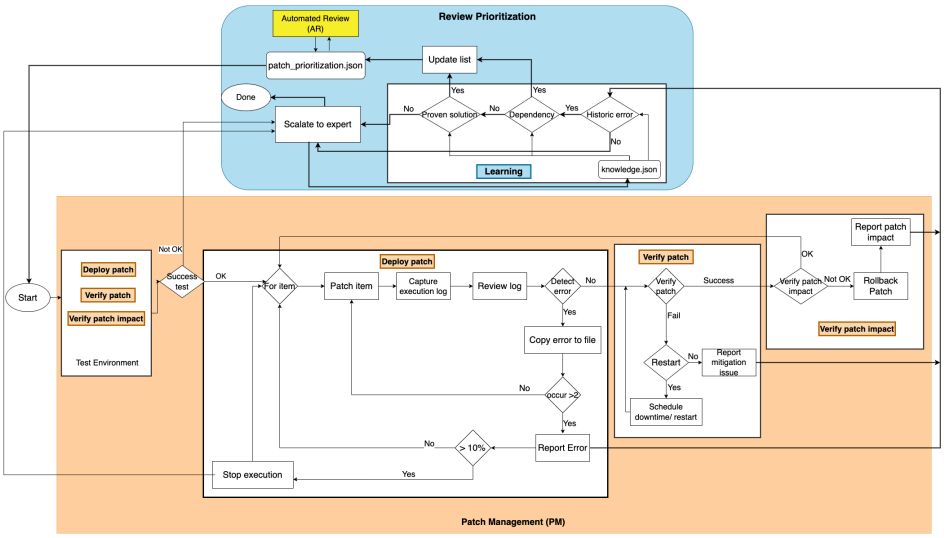

Figure 10.3: *ACVRM Phase 3 process including feedback loop to Phase 2 for test and production environments*

functions split into five modules; *Deploy patch*, *Verify patch*, *Verify patch impact*, *Learning*, and *Automated Review*. Each module represents the implementation of each stage and the output of each module is the input for the next one. Phase 2 PoC implementation was described in [5]; hence is omitted in this paper. The output of phase 2 is a patch prioritization list (`patch_prioritization.json`) for each host in the organization, and it is the input to phase 3. The PM process is responsible for testing the patch in the test environment before patching it in production. The PM process, presented in Figure 10.3, starts with executing *Deploy patch*, *Verify patch*, and *Verify patch impact* modules in a test environment. If patches were successful in a test environment, the *Deploy patch* module is initiated for a selected host in production.

The *Deploy patch* process is iterated for each item (vulnerability) in the file until the stop signal is generated. ACVRM will stop patch execution if the acceptance rate of a failure [28] (i.e., the percentage of the unsuccessful patch accepted by security experts in the organization) is exceeded. The default acceptance rate in our design is up to 10% failure of patches in `patch_prioritization.json` for each host. The rate could be customized based on the organization's desires. The deploy patch module allows the second execution of the patch for each item if the first execution encounters

an error. Sometimes, the error resolves in a second run based on the patch management community practices. This module generates an error report for each persistent error (an error occurred after two tries), and sends it to the *Learning* module, within the Review Prioritization.

The *Verify patch* module begins if no error is detected in the *Deploy patch* module. This module verifies the remediation by examining the running version of vulnerable software or using a vulnerability scanner. In our PoC, we use version control, where the running software version must be the same as the version installed by patch execution. If remediation fails, the system or service restart condition would examine. Some vulnerability patches require a system or service restart to be effected (e.g., Linux Kernel vulnerability requires a system restart because the installed version is loaded into memory when a system starts). In case of an error, report the error to the *Learning* module. If remediation is confirmed, ACVRM moves to the next stage.

The *Verify patch impact* module initiates when remediation is validated. This module verifies the patch's impact on the system or application functionality by running predefined functional tests. The test depends on the organization's unique services and applications and should be defined by the organization's experts (e.g., the impact of the patch in a node hosting the organization's website, could be verified by the status of the web server, and website response). ACVRM could automate the test and verify the expected result. If the side effect is detected, ACVRM will rollback the patch and report the impact to the *Learning* module. Otherwise, it jumps to the next item in the `patch_prioritization.json` until the last item. The *Learning* module is the enhancement in the Review prioritization stage. This module builds knowledge from the organization's past experiences and the current one to improve patch prioritization decisions. The `knowledge.json` is created from the historical events data and the data provided by a feedback loop in JSON format. Security experts could feed organizational historical data into the *Learning* module. If the historical data are not available, the module builds the knowledge based on the feedback loop only. The *Learning* module helps the organization handle known errors automatically without the expert's involvement. In our design, the learning module will check three conditions before escalating the error to the security expert. First, it inspects the existence of errors in `knowledge.json`. Second, it checks the dependencies in the error report. Finally, it searches for a proven solution in `knowledge.json`. If knowledge data matches the condition, it updates the

patch prioritization. Otherwise, it sends a report to the experts for review and response.

Moreover, we introduce *Automated Review (AR)* in our design to improve the patch prioritization list before PM cf. yellow box in Figure 10.3. We find that there is a possibility of having multiple CVE-IDs for each software or application in the patch prioritization list of the host. AR reviews and sorts the CVE-IDs in the patch list based on the name of the vulnerable software or application and the patch version. If AR finds multiple patches of the same software or application, which is cumulative, it removes the older patches from the priority list. AR also checks the dependency for each vulnerability in the list because dependency is a common reason for patch failure[15, 16, 29, 30]. The dependencies are considered prerequisites, which directly impact the outcome of PM. If the organization provides a dependency tree for each software or application, AR inspects and reflects them in the prioritization list.

10.6 Experiment

We plan to test our PM PoC by executing a controlled experiment. In the experiment, we set up a test organization by creating some nodes, injecting random vulnerabilities into nodes, verifying the vulnerabilities are detected, and a patch prioritization list is created for each node. Then, we apply the PM process to patch the detected vulnerabilities and verify the remediation and patch impact in a test organization based on the process presented in Figure 10.3. In our experiment, we assumed that the patch testing and verification were successful for a test organization. The test organization is created by a network of virtual servers deployed on a public cloud platform. Figure 10.4 shows our test organization setup; it consists of eight virtual servers (Host 1-8), one storage node (Local Storage), one Rudder node [31], and one Nessus node [32]. All nodes are connected to a switch. The servers are running Ubuntu as an Operating System (OS). Rudder node is a host running the *Rudder.io* manager version 6.2 as an inventory tool. The Rudder manager receives the nodes' data through the installed Rudder agent on the eight virtual servers. Nessus node is a host running the Nessus vulnerability scanner community edition, version 8.14.0-ubuntu110_amd64. The nodes are created using the OS image provided by the cloud provider and then updated to the latest stable version available at the testing time (December

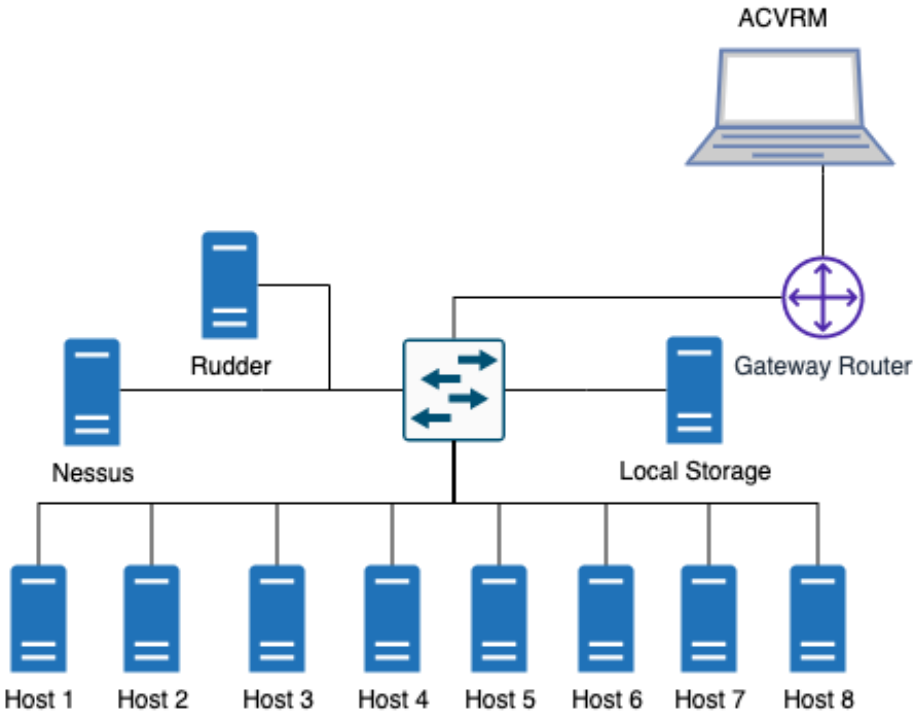

Figure 10.4: *Test environments*

2022). Each node has 1 CPU core, 1GB RAM, and OS version 18.04.4 LTS.

After initiating the test organization, we randomly select 21 CVE-IDs (relevant to our virtual servers) and install their vulnerable version on our eight virtual servers (Host 1-8). We deploy phase 1 and phase 2 of ACVRM to identify the installed vulnerability and obtain a patch prioritization for each host. In our experiment, we weight the prioritization criteria homogeneously, i.e., $w_i = 0.1667$ in Patch Score (PS) calculation. Table 10.2 shows the patch priority list for Hosts 1-8, the selected CVE-IDs, the name of the software, the Severity Score (SC), PS, and the patched version of the software. SC is a normalized score with a standard vulnerability management module [6].

We implement a PM PoC, our core contribution in this study, in our test organization according to our defined process. The aim is to investigate the impact of the patch review and feedback loop on the success of automated patching and reducing expert intervention. Hence, we define three cases:

- Case 1: PM based on the patch prioritization list in our previous work [5], where the PS determines the patch order. In this case, there is no automated review of the patch prioritization. We also consider the organization does not have any historical data from its previous patch. Moreover, we allowed the patch execution without interruption until the end of the priority list (i.e., do not check the error percentage for a test host).

- Case 2: PM based on the updated `patch_prioritization.json` by adding an AR in the process. In this case, the organization does not provide dependencies. Therefore, the update is based on removing unnecessary patches (e.g., multiple cumulative patches of software or application) from the list.

- Case 3: PM based on an improved priority list by automatically checking dependencies of the vulnerabilities. In this case, we consider the test organization has a record of dependencies, and AR will update the `patch_prioritization.json` accordingly.

For each case, we keep the state of the virtual servers (e.g., with 21 installed vulnerabilities) unchanged and review the PM output for each case. We also disabled the check percentage of error for stop execution in our experiment as we wanted to capture all errors without interruption.

10.7 Results and Discussion

In this study, we used the success rate of the patch as a metric to evaluate the proposed PM approach. We verified the patch impact in our test organization through a GET request to the website of the test organization and the state of the system (up and running) after the patch.

The result of Case 1 in Figure 10.5 shows that 48% of the patches escalate to the expert because of the errors in patch execution. We also observed that 19% of the patches failed in verification because they required a system or service restart after the patch. We noticed the success rate of the patch was 33% without a feedback loop and 52% with a feedback loop (i.e., the condition of 19% of verification failure resolved after system or service restart). Moreover, we identify the majority of the errors in Case 1 belong to software with a different version in the priority list, but the error

Table 10.2: *The sample of patch prioritization list for Host 1-4 with Ubuntu operating system*

Priority	CVE-ID	Name	SC	PS	Patched version
1	CVE-2021-3711	openssl	9.1833	4.0810	1.1.1-1ubuntu2.1∼18.04.13
2	CVE-2021-44790	apache	8.3500	3.9421	2.4.29-1ubuntu4.21
3	CVE-2022-2526	systemd	8.3500	3.9421	237-3ubuntu10.56
4	CVE-2021-39275	apache	7.7833	3.8476	2.4.29-1ubuntu4.17
5	CVE-2022-23943	apache	7.7833	3.8476	2.4.29-1ubuntu4.22
6	CVE-2022-31813	apache	7.5167	3.8031	2.4.29-1ubuntu4.24
7	CVE-2022-2068	openssl	7.3167	3.7698	1.1.1-1ubuntu2.1 18.04.19
8	CVE-2022-32221	curl	6.6833	3.6642	7.58.0-2ubuntu3.21
9	CVE-2022-22576	curl	7.2167	3.6598	7.58.0-2ubuntu3.17
10	CVE-2022-23219	glibc	6.2667	3.5948	2.27-3ubuntu1.5
11	CVE-2022-45061	python	6.8167	3.4997	3.6.9-1∼18.04ubuntu1.9
12	CVE-2020-24489	intel-microcode	8.5167	1.9198	3.20210608.0ubuntu0.18.04.1
13	CVE-2022-42896	Linux Kernel	7.8500	1.8204	4.15.0-202.213
14	CVE-2021-4034	policykit-1	7.8500	1.8087	0.105-20ubuntu0.18.04.6
15	CVE-2022-34918	Linux Kernel	7.8500	1.8087	4.15.0-191.202
16	CVE-2022-0392	vim	7.0167	1.6698	2:8.0.1453-1ubuntu1.10
17	CVE-2022-1621	vim	6.8500	1.6420	2:8.0.1453-1ubuntu1.9
18	CVE-2021-0146	intel-microcode	6.4500	1.5170	3.20220510.0ubuntu0.18.04.1
19	CVE-2021-33910	systemd	6.3167	1.3664	237-3ubuntu10.49
20	CVE-2020-24513	intel-microcode	5.8500	1.2886	3.20210608.0ubuntu0.18.04.1
21	CVE-2022-21233	intel-microcode	5.6500	1.2553	3.20220809.0ubuntu0.18.04.1

messages are not the same. The column Case 1 Result in Table 10.3 presents the patch result for Case 1.

The PM results of Case 2 in Figure 10.5 show the improvement in the success rate of the patch by 60% without the feedback loop and by 90% with a feedback loop. Also, the expert intervention decreased from 48% to 10% comparing Case 1. 30% of the patch failed in verification has been resolved by feedback loop condition as all required system or service restart. We noticed that the number of CVE-IDs in our sample list decreased from 21 to 10 with AR.

The Case 3 result in Figure 10.5 shows the 80% success rate without a feedback loop and 0% unknown error to escalate to the expert. 20% of verification failures were addressed by feedback loop as they required system restart.

Table 10.3 presents the patch result without a feedback loop to visualize the impact of the feedback loop for three cases (e.g., Case 1 Result, Case 2 Result, and Case 3 Result). The patch priority has been changed for Case 2 and Case 3 compared to Case 1. We define Δ as the difference between the position in patch priority ($P_{k,*}$) between Case 1 and other cases as:

$$\Delta_{k,j} = P_{\text{k,Case 1}} - P_{\text{k,Case j}} \quad j = 2, 3 \tag{10.1}$$

where k is CVE-ID in the list. A positive value of Δ indicates that the CVE-ID got a higher priority concerning Case 1. The negative value of Δ refers to lower priority compared with Case 1, while the zero value of Δ marks no changes in the priority. The Case 2/3 priority (Δ) column in Table 10.3 shows the priority of Case 2/3, and the Δ value in parenthesis indicates changes in the CVE-ID position compared with Case 1. For example, the CVE-2022-2526 has a priority 3 by Case 1 while it becomes a priority 1 in Case 2 and Case 3. The lack of values for the Case 2 and 3 priority (Δ) column means that the CVE-ID has been removed from the priority list. We only see the *Delta* with a positive value as the number of the CVE-IDs decreases 52% for Case 2 and Case 3. For example, Case 1 consists of four patches for apache (CVE-2021-44790, CVE-2021-39275, CEV-2022-23943, and CVE-2022-31813), which is reduced to one in Case 2 and Case 3.

We observe that Case 2 and Case 3 have the same CVE-IDs in the patch list, but the prioritization is different. For example, CVE-2022-31813 has a priority 2 in Case 2 and a priority 4 in Case 3. The differences in Case 2 and Case 3 prioritization are due to the impact of the dependencies. In Case 3, the organization provides a dependency tree of each vulnerability, and AR verifies dependencies and adjusts the priority order accordingly. In our test organization, three CVE-IDs (CVE-2022-31813, CVE-2022-2068, and CVE-2022-32221) have the `libc6` in their dependencies tree. The position of `libc6` vulnerability (CVE-2022-23219) is after those three CVE-IDs based on PS. As described in 10.5, dependencies are prerequisites. The dependence got a higher priority despite having a lower PS. Hence, the position of CVE-2022-23219 was adjusted to be patched before the software, depending on it. The result from all three case studies shows the impact of automation on reducing the expert intervention in patch vulnerabilities. We also verify the improvement in the patch prioritization by adding a feedback loop and learning from patch history.

Furthermore, we observe that AR improved efficiency in patch prioritization and PM in Case 1 and 2 because AR removes the unnecessary patch from the list and adjusts the list based on dependencies. Moreover, the verification of patch impact for all three cases was positive, and the patches did not break the system functionality.

Table 10.3: *The patch result of Case 1-3; the empty cells indicate the CVE-ID removed from the patch priority*

Priority	CVE-IDs	Name	Case 1 Result	Case 2 Priority (Δ)	Case 2 Result	Case 3 Priority (Δ)	Case 3 Result
1	CVE-2021-3711	openssl	Verification fail				
2	CVE-2021-44790	apache	Error escalate to expert				
3	CVE-2022-2526	systemd	Successful	1 (2)	Successful	1 (2)	Successful
4	CVE-2021-39275	apache	Error escalate to expert				
5	CVE-2022-23943	apache	Error escalate to expert				
6	CVE-2022-31813	apache	Error escalate to expert	2 (4)	Error escalate to expert	4 (2)	Successful
7	CVE-2022-2068	openssl	Error escalate to expert	3 (4)	Verification fail	3 (4)	Successful
8	CVE-2022-32221	curl	Successful	4 (4)	Successful	5 (3)	Successful
9	CVE-2022-22576	curl	Error escalate to expert				
10	CVE-2022-23219	libc6	Successful	5 (5)	Successful	2 (8)	Successful
11	CVE-2022-45061	python	Successful	6 (5)	Successful	6 (5)	Successful
12	CVE-2020-24489	intel-microcode	Error escalate to expert				
13	CVE-2022-42896	Linux	Verification fail	7 (6)	Verification fail	7 (6)	Verification fail
14	CVE-2021-4034	policykit-1	Successful	8 (6)	Successful	8 (6)	Successful
15	CVE-2022-34918	Linux	Verification fail				
16	CVE-2022-0392	vim	Successful				
17	CVE-2022-1621	vim	Error escalate to expert	9 (8)	Successful	9 (8)	Successful
18	CVE-2021-0146	intel-microcode	Verification fail				
19	CVE-2021-33910	systemd	Successful				
20	CVE-2020-24513	intel-microcode	Error escalate to expert				
21	CVE-2022-21233	intel-microcode	Error escalate to expert	10 (11)	Verification fail	10 (11)	Verification fail

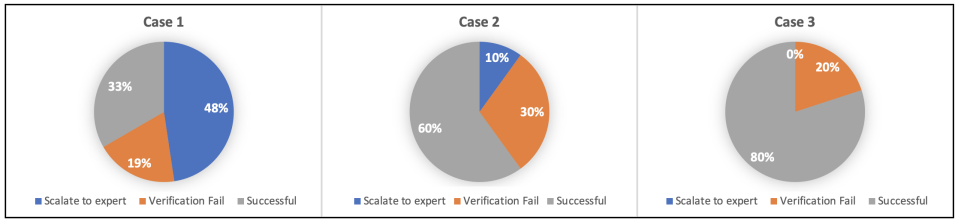

Figure 10.5: *The status of the patch management for cases 1-3*

10.8 Conclusion and future work

The increasing number of publicly known vulnerabilities introduces the challenge in VRM as it requires experts' intervention. The average number of vulnerabilities with critical and high scores was 40 per day in 2022, according to NVD[27], which forced the organizations to improve their VRM procedure to remain secure and compliant. In this study, we introduced the PM of ACVRM to automate patching, reduce experts' intervention, and improve the success rate of the patch. We performed an analysis as follows:

1. We learned the challenges in PM and the criteria that should be considered from the literature review and patch management community best practices. We determined that time to patch, expert availability, system downtime, and dependencies are important criteria in patching vulnerabilities. Therefore, we define our automated patching process to reduce expert intervention while increasing the success rate of the patch.

2. We designed and implemented phase 3 of ACVRM, which consists of testing and verifying the vulnerability patches and the impact

of patches on the system functionality. We learned that reviewing prioritization based on the history of the patch will improve the patch prioritization in the organization's context.

3. We verified the result of our proposed PM by analyzing the outcome of each case. Our result shows that the ACVRM could adjust the patch prioritization for each organization with less effort from security experts. Automated Review has been introduced to remove unnecessary patches and reduce errors due to dependencies. Our solution allows security experts to set the patch failure rate and stop the execution of the error-prone patch prioritization.

Our study shows how the organization could automate VRM based on its context. ACVRM facilitates improvement in the VRM procedure by patch prioritization in the organization's context and reducing experts' intervention. The feedback loop provides an opportunity of learning from historical data and enhances the organization's knowledge which increases the success rate of patching. In the future, we could add the learning from the patch management community, social media, and threat intelligence into the review prioritization to support better decisions on the vulnerability ranking. Hence, the success rate of patching will improve. Another possible future direction could be using a machine learning algorithm such as a decision tree in the *Learning module* to improve error handling.

References

[1] *CIS Controls*. `http://www.cisecurity.org/controls/`. [Online; accessed 10-Jan-2023].

[2] *The Log4j Vulnerability*. `https://www.wsj.com/articles/what-is-the-log4j-vulnerability-11639446180`. [Online; accessed 16-November-2022].

[3] *NIST National Vulnerability Database*. `https://nvd.nist.gov/`. [Online; accessed 10-Jan-2023].

[4] V. Ahmadi, P. Arlos, and E. Casalicchio. "Normalization of Severity Rating for Automated Context-aware Vulnerability Risk Management". In: *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. IEEE. 2020, pp. 200–205.

[5] V. Ahmadi Mehri, P. Arlos, and E. Casalicchio. "Automated Context-Aware Vulnerability Risk Management for Patch Prioritization". In: *Electronics* 11.21 (2022), p. 3580.

[6] V. Ahmadi, P. Arlos, and E. Casalicchio. "Normalization Framework for Vulnerability Risk Management in Cloud". In: *2021 IEEE International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE. 2021.

[7] S. Beattie, S. Arnold, C. Cowan, P. Wagle, C. Wright, and A. Shostack. "Timing the Application of Security Patches for Optimal Uptime." In: *LISA*. Vol. 2. 2002, pp. 233–242.

[8] *Microsoft halts AMD meltdown and spectre patches after reports of unbootable pcs*. `https://www.theverge.com/2018/1/9/16867068/microsoft-meltdown-spectre-security-updates-amd-pcs-issues`. [Online; accessed 10-Jan-2023].

[9] *EU Cybersecurity Act*. `https://eur-lex.europa.eu/eli/reg/2019/881/oj`. [Online; accessed 11-January-2023].

[10] *European Cybersecurity Certification Scheme for Cloud Services*. `https://www.enisa.europa.eu/publications/eucs-cloud-service-scheme`. [Online; accessed 11-January-2023].

[11] *Homland Security Act 2002*. `https://www.dhs.gov/homeland-security-act-2002`. [Online; accessed 11-January-2023].

[12] N. Dissanayake, A. Jayatilaka, M. Zahedi, and M. A. Babar. "Software security patch management-A systematic literature review of challenges, approaches, tools and practices". In: *Information and Software Technology* 144 (2022), p. 106771.

[13] U. Gentile and L. Serio. "Survey on international standards and best practices for patch management of complex industrial control systems: the critical infrastructure of particle accelerators case study". In: *International Journal of Critical Computer-Based Systems* 9.1-2 (2019), pp. 115–132.

[14] G. Post and A. Kagan. “Computer security and operating system updates”. In: *Information and Software Technology* 45.8 (2003), pp. 461–467.

[15] J. Dunagan, R. Roussev, B. Daniels, A. Johnson, C. Verbowski, and Y.-M. Wang. “Towards a self-managing software patching process using black-box persistent-state manifests”. In: *International Conference on Autonomic Computing, 2004. Proceedings.* IEEE. 2004, pp. 106–113.

[16] C. Tiefenau, M. Häring, K. Krombholz, and E. Von Zezschwitz. “Security, availability, and multiple information sources: Exploring update behavior of system administrators”. In: *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. 2020, pp. 239–258.

[17] H. Huang, S. Baset, C. Tang, A. Gupta, K. M. Sudhan, F. Feroze, R. Garg, and S. Ravichandran. “Patch management automation for enterprise cloud”. In: *2012 IEEE Network Operations and Management Symposium*. IEEE. 2012, pp. 691–705.

[18] S. Midtrapanon and G. Wills. “Linux patch management: with security assessment features”. In: *4th International Conference on Internet of Things, Big Data and Security (IoTBDS)*. 2019.

[19] F. Zhang, P. Huff, K. McClanahan, and Q. Li. “A machine learning-based approach for automated vulnerability remediation analysis”. In: *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2020, pp. 1–9.

[20] C.-W. Chang, D.-R. Tsai, and J.-M. Tsai. “A cross-site patch management model and architecture design for large scale heterogeneous environment”. In: *Proceedings 39th Annual 2005 International Carnahan Conference on Security Technology*. IEEE. 2005, pp. 41–46.

[21] M. Procházka, D. Kouril, R. Wartel, C. Kanellopoulos, and C. Triantafyllidis. “A Race for Security: Identifying Vulnerabilities on 50 000 Hosts Faster than Attackers”. In: *Proceedings of Science (PoS). International Symposium on Grid and Clouds*. 2011.

[22] J.-H. Lee, S.-G. Sohn, B.-H. Chang, and T.-M. Chung. “PKG-VUL: Security Vulnerability Evaluation and Patch Framework for Package-Based Systems”. In: *ETRI journal* (2009).

[23] B. Marx and D. Oosthuizen. “Risk Assessment and Mitigation at the Information Technology Companies”. In: *Risk Governance & Control: Financial markets and institutions* 6.02 (2016), pp. 44–51.

[24] F. Li, L. Rogers, A. Mathur, N. Malkin, and M. Chetty. “Keepers of the machines: Examining how system administrators manage software updates for multiple machines”. In: *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*. 2019, pp. 273–288.

[25] A. Shah, K. A. Farris, R. Ganesan, and S. Jajodia. “Vulnerability selection for remediation: An empirical analysis”. In: *The Journal of Defense Modeling and Simulation* 19.1 (2022), pp. 13–22.

[26] *2022 Vulnerability statistics report*. `https://www.edgescan.com/2022-vulnerability-statistics-report-lp/`. [Online; accessed 11-Jan-2023].

[27] *NIST National Vulnerability Database search*. `https://nvd.nist.gov/vuln/`. [Online; accessed 10-Jan-2023].

[28] Y. Kansal, D. Kumar, and P. Kapur. “Vulnerability patch modeling”. In: *International Journal of Reliability, Quality and Safety Engineering* 23.06 (2016), p. 1640013.

[29] A. Nappa, R. Johnson, L. Bilge, J. Caballero, and T. Dumitras. “The attack of the clones: A study of the impact of shared code on vulnerability patching”. In: *2015 IEEE symposium on security and privacy*. IEEE. 2015, pp. 692–708.

[30] R. Schwarzkopf, M. Schmidt, C. Strack, and B. Freisleben. “Checking running and dormant virtual machines for the necessity of security updates in cloud environments”. In: *2011 IEEE Third International Conference on Cloud Computing Technology and Science*. IEEE. 2011, pp. 239–246.

[31] *Rudder*. `https://www.rudder.io/`. [Online; accessed 14-January-2023].

[32] *Nessus Vulnerability Scanner*. `https://www.tenable.com/products/nessus`. [Online; accessed 12-January-2023].

ABSTRACT

The information security landscape continually evolves with increasing publicly known vulnerabilities (e.g., 25064 new vulnerabilities in 2022). Vulnerabilities play a prominent role in all types of security related attacks, including ransomware and data breaches. Vulnerability Risk Management (VRM) is an essential cyber defense mechanism to eliminate or reduce attack surfaces in information technology. VRM is a continuous procedure of identification, classification, evaluation, and remediation of vulnerabilities. The traditional VRM procedure is time-consuming as classification, evaluation, and remediation require skills and knowledge of specific computer systems, software, network, and security policies. Activities requiring human input slow down the VRM process, increasing the risk of exploiting a vulnerability.

The thesis introduces the Automated Context-aware Vulnerability Risk Management (ACVRM) methodology to improve VRM procedures by automating the entire VRM cycle and reducing the procedure time and experts' intervention. ACVRM focuses on the challenging stages (i.e., classification, evaluation, and remediation) of VRM to support security experts in promptly prioritizing and patching the vulnerabilities.

ACVRM concept is designed and implemented in a test environment for proof of concept. The efficiency of patch prioritization by ACVRM compared against a commercial vulnerability management tool (i.e., Rudder). ACVRM prioritized the vulnerability based on the patch score (i.e., the numeric representation of the vulnerability characteristic and the risk), the historical data, and dependencies. The experiments indicate that ACVRM could rank the vulnerabilities in the organization's context by weighting the criteria used in patch score calculation. The automated patch deployment is implemented with three use cases to investigate the impact of learning from historical events and dependencies on the success rate of the patch and human intervention. Our finding shows that ACVRM reduced the need for human actions, increased the ratio of successfully patched vulnerabilities, and decreased the cycle time of VRM process.

ISSN 1653-2090
2023:07
ISBN 978-91-7295-459-5