# Building Secure Environments for Microservices

Bradley Northern, Denis Ulybyshev
*Department of Computer Science*
*Tennessee Technological University*
Cookeville, United States of America
banorthern42, dulybyshev@tntech.edu

2022 International Workshop on Secure and Reliable Microservices and Containers (SRMC) | 978-1-6654-5539-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/SRMC57347.2022.00008

***Abstract*—Microservice-based architectures are widely used in modern software and the number of cyber attacks on software is increasing. It is essential to make microservices more reliable and resilient against cyber attacks. In this paper, we propose a methodology to detect configurations of computing systems that host microservices and containers, evaluate cyber risks of their essential components in an automatic continuous mode, and reconfigure environments aiming to make them less vulnerable against cyber attacks. Our solution supports multiple operating systems and hardware configurations. For cyber risk evaluation, our approach relies on a public database of Common Vulnerabilities and Exposures for software and hardware, as well as penetration testing and static analysis. Furthermore, the cyber risk evaluation model considers the attributes of microservices, such as their privilege level. We will show least and most vulnerable configurations for computing systems that use popular operating systems and applications.**

***Index Terms*—microservices, cyber risk assessment, cyber resiliency**

## I. Introduction

A number of software and hardware vulnerabilities keeps growing which results in cyber attacks. It is essential to protect critical infrastructures from cyber attacks on software since the consequences can be very serious. As an example, cyber attack on the water treatment facility's Industrial Control System in Florida "changed chemical levels, making the water unsafe to consume" [1]. Global cost of data breaches by 2024 is expected to be $5 trillion, according to [2]. Since microservices and containers are widely used in modern software systems, it is important to protect them and the environment they run on. In this paper, we propose a Vulnerability Analysis and Cyber Risk Assessment for Microservices (VERCASM-M) methodology to evaluate cyber risks of environments (computing systems), including software and hardware, for microservices in automated and continuous mode, aiming to harden the environment and make it less vulnerable for cyber attacks. VERCASM-M cyber risk assessment model evaluates cyber risks of separate components in the computing system and then calculates the total cyber risk score for the entire system. This model considers results of static analysis and penetration testing for software components and microservices that need to be protected. The model also considers publicly available software and hardware vulnerabilities from the Common Vulnerabilities and Exposures (CVE) database [3] for components of the system that hosts microservices and containers. The paper is a work in progress and it presents the extension of a VERCASM approach presented in [4]. Contributions of this paper are summarized below:

- Novel methodology to evaluate cyber risks of microservice environments (computing systems) in automated and continuous mode, considering the attributes of microservices and results of the penetration testing and static analysis applied to the system.
- Novel application of VERCASM approach to harden environments that host microservices and make them less vulnerable for cyber attacks.

The rest of the paper is organized as follows. Section II covers related works. The core design and implementation is presented in Section III. Afterwards in Section IV, evaluation results are shown. Section V concludes the paper.

## II. Related Work

VERCASM solution [4] allows to evaluate cyber risks for computing systems. In this work, we extend this approach to VERCASM-M that provides the following:

- Cyber risk evaluation methodology that relies on penetration testing and static analysis for microservices, in addition to using a public database of Common Vulnerabilities and Exposures for software and hardware.
- Cyber risk evaluation methodology that considers attributes of microservices, such as their privileges and permission level.

Vulners.com is "a security database containing descriptions for a large amount of software vulnerabilities in a machine-readable format" [5]. It has a search engine with Lucene-based queries to search for vulnerabilities in a specified software, Linux vulnerability scanner, and a Perimeter Scanner to scan and audit vulnerabilities. In our approach we compute cyber risk scores for each individual software and hardware component and the entire computing system. Our solution is Operating System (OS)-agnostic and evaluates cyber risk scores for software and hardware that host microservices.

Jain et al. in "Static Vulnerability Analysis of Docker® images" [6], analyze the different security tools that exist on the market to find vulnerabilities of Docker® images. Docker® unlike a Virtual Machine (VM) can be seen as a lightweight VM, without the weight of non-essential kernel modules. Like much of the single open-source software applications, Docker® images contain multiple pieces of software that can be pulled from services like Docker® Hub. Due to this, the authors raise

978-1-6654-5539-8/22/$31.00 ©2022 IEEE
DOI 10.1109/SRMC57347.2022.00008

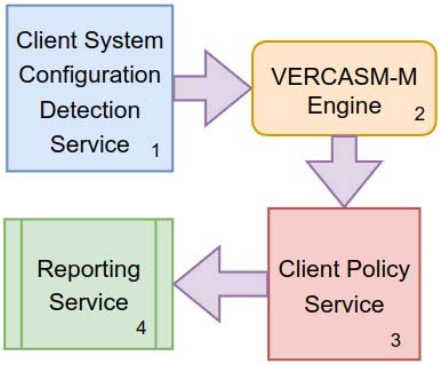


Fig. 1: VERCASM-M Architecture

important issues with Docker®. While it is free and open there is no "cryptographic proof" to say an image author is not a bot trying to publish what may seem like a useful container, but actually could be malicious software. This is why that static vulnerability analysis is needed on the Docker® images. Furthermore, vulnerability analysis on any piece of software running on a computing system or inside a container is needed. In our VERCASM-M approach, we evaluate cyber risks not only for Docker® containers, but for computing systems, in general, including software and hardware. Furthermore, VERCASM-M allows to evaluate cyber risks at a design phase for new systems.

"Tenable®", a software company, produces a software (Tenable Lumin) [7]. It collects information from a user's system; this information contains all the installed software, including OS. Based on this information they compute a score named the "Cyber Exposure Score". This score is calculated by an Asset Criticality Rating machine learning algorithm. All of the outputs of this scoring engine are computed using a geometric mean. Depending on the score given for each piece of software they have a recommendation system which reports to the user to update or keep the same software. In contrast, VERCASM-M can recommend to update and also to downgrade the software where applicable. Furthermore, VERCASM-M relies on weighted Fibonacci series for cyber risk assessment, instead of geometric mean. In addition, VERCASM-M can evaluate cyber risks for newly designed systems and select least vulnerable components to be used.

## III. Core Design

In our design of VERCASM-M, we use 4 main services as seen in Figure 1: Client System Configuration Detection Service, VERCASM-M Engine, Client Policy Service, and a Reporting Service.

### A. Microservice-based System Configuration Detection

In our solution, we heavily rely on a microservice architectural design. By using services we are able to fit the needs in finding the environment configuration architecture of another service. Client System Configuration Detection Service (shown as Component 1 in Figure 1) can detect the host or container OS; installed software (name and version); and permissions to execute the service. Given these parameters we are able to generate a report with guidelines on how to harden the system. We use details such as name, version, and vendor, of each piece of software on the computing system to generate a cyber risk score of an entire system. We also use details about the runtime environment: OS and version. Runtime environment can be Docker®, LXD, or a barebones system. We fully support the Linux®-based package detection. This can be done with the major software package providers, including AppImages, Flatpaks (All); apt (Debian); AUR (Arch); dnf, yum (RHEL); zypper (SUSE).

For our evaluation below we use Debian®-based OS which ships with the Advanced Package Tool (apt) package manager. Installed packages on this type of a system can be found in "/var/log/dpkg.log*". This file contains information including timestamp, status, software name, version, and architecture of packages. We can filter this file with two GNU/Linux commands, *grep* and *cut*, as seen below:

Listing 1: Filtering Debian Package Log File

```
$ grep " installed " /var/log/dpkg.log* |
cut -d' ' -f5,6  --output-delimiter=':' |
cut -d: -f1,3
```

An example of what each command does is shown in Table I, where we are filtering from an example line: "/var/log/dpkg.log.1:2022-05-31 23:21:07 status installed lua5.3:amd64 5.3.3-1.1ubuntu2"

| Command | Expected Output |
|---|---|
| grep " installed " /var/log/dpkg.log* | example line |
| cut -d' ' -f5,6 | lua5.3:amd64 5.3.3-1.1ubuntu2 |
| cut -d' ' -f5,6 | lua5.3:amd64:5.3.3-1.1ubuntu2 |
| cut -d: -f1,3 | lua5.3:5.3.3-1.1ubuntu2 |

TABLE I: GNU/Linux Commands to Retrieve Software Names and Versions for Debian-based Systems

Using these commands we are able to extract the programming language Lua with version 5.3.3-1.1ubuntu2. As of June 24, 2022, there are 94 CVE's that relate to Lua, from years 2006 to 2022. This shows the importance of determining the cyber risk score of a system. If a hacker was able to access the computing system outside of a service, they would be able to run a Lua script with ease and if one of the vulnerabilities allowed for root access they could gain the root access. We can further extend this into searching over all Common Platform Enumeration (CPE) data, where it finds the first available vendor to match a piece of software. While this method is not full proof in cases where a software name may have multiple vendors, it is an item that is included in future work.

### B. Cyber Risk Evaluation

Once the Client System Configuration Detection Service has detected all the software installed in a computing system, including software names and versions, this information is sent to VERCASM-M Engine (shown as Component 2 in Figure

1) using a RESTful API. VERCASM-M Engine computes cyber risk scores for individual software component and the Total Cyber Risk Score (TCRS) for the entire computing system. The cyber risk score is a number that relates to the vulnerability of the system from 0 to 10.0, with 10.0 being the most vulnerable and risky. The formula (1) to calculate the Individual Cyber Risk Score (ICRS) of an individual software component, shown below, uses the weighted Fibonacci algorithm [4].

$$ICRS = \frac{\sum_{i=1}^{n} w(x_i)x_i}{\sum_{i=1}^{n} w(x_i)} \quad (1)$$

In this formula, $x_i$ is the Common Vulnerability Scoring System (CVSS) [8] score of the vulnerability (CVE) related to the given software component (from 0.0 to 10.0); n is the total number of vulnerabilities (CVEs) for the given software component; $w(x_i)$ is the Fibonacci weight defined by the piecewise function:

$$w(x_i) = \begin{cases} 0.1, & x_i \in [0.1 - 3.9] \\ 0.2, & x_i \in [4.0 - 6.9] \\ 0.9, & x_i \in [7.0 - 8.9] \\ 4.4, & x_i \in [9.0 - 10.0] \end{cases}$$

,each

$$x_i \in [.1 - 10]$$

Fibonacci weights, shown in Table II, are used for CVE weights since more critical vulnerabilities should contribute more to the software cyber risk score, whereas minor vulnerabilities should have minor impact on the cyber risk score. We use this scoring model since it assigns high cyber risk scores to vulnerable software from Top-50 Vulnerable software list [9] and low cyber risk scores for software that does not have critical vulnerabilities with CVSS greater or equal than 9.0. Paper [4] provides more details. In VERCASM-M, in contrast to VERCASM, users can flexibly select weighting schemas to customize and reflect cyber risks in specific computing systems. The scoring is based on the CVSS Version 3.

| Severity Ranges | Weights |
|---|---|
| Low (0.1–3.9) | 0.1 |
| Medium (4.0–6.9) | 0.2 |
| High (7.0–8.9) | 0.9 |
| Critical (9.0–10.0) | 4.4 |

TABLE II: Weights Assigned per CVSS Range (taken from [4] Table 3 with permission)

The formula (2) to calculate the TCRS of the entire computing system is as follows:

$$TCRS = \frac{\sum_{i=1}^{m} w(ICRS_i)ICRS_i}{\sum_{i=1}^{m} w(ICRS_i)} \quad (2)$$

| Octal | Permission | Modified TCRS |
|---|---|---|
| 0777 | .rwxrwxrwx | TCRS $\times$ 2 |
| 0766 | .rwxrw-rw- | TCRS $\times$ 1.9 |
| 0733 | .rwx-wx-wx | TCRS $\times$ 1.7 |
| 0763 | .rwxrw–wx | TCRS $\times$ 1.7 |
| 0760 | .rwxrw—- | TCRS $\times$ 1.1 |

TABLE III: VERCASM-M Modified Total Cyber Risk Scores

In this formula, $ICRS_i$ is the cyber risk of the individual software component (from 0.0 to 10.0) calculated in formula (1); $w(ICRS_i)$ is the user-defined weight (from 0.0 to 1.0) for an individual software (the more important the software the greater this weight should be); m is the number of software components in the computing system.

Our scoring model allows to modify the TCRS depending on attributes of microservices we aim to protect. Table III illustrates how permissions of microservices impact the TCRS score. The rationale behind this conclusion, is that if the microservice runs with root privileges, then its compromise is more critical and dangerous for the computing system. The maximum TCRS score is 10.0, so if the modification results in a score greater than 10.0, we have a ceiling of 10.0. If multiple microservices need to be protected then modified TCRS score can be conservatively selected as maximum between all microservices. Corrective weights for modified TCRS are set by client policies and managed by the Client Policy Service, shown as Component 3 in Figure 1. These weights are found experimentally depending on the importance, probability of being exploited [10], and other features of microservices. As an example, if a java® service configuration has TCRS score of 4.0, and the jar was setup with maximum permissions of '0777' (read, write, and execute by everyone), then we would take 2 $\times$ 4.0 to assign a modified TCRS score of 8.0 to the computing system.

We employ the pre-determined Static Analysis (SA) of specific software, if its source code is available. Using static analysis for cyber risk evaluation is essential in cases when there are no CVEs registered. It could be when the software is not publicly available. In cases when the new open-source software is released by a GNU/Linux and a buffer overflow vulnerability is found then we can assign ICRS score based on the number of vulnerabilities found, their probability of being exploited, the importance of the software for a given system, and other metrics. Furthermore, we can escalate the scoring higher based on a clients policies. A client may regard buffer overflow has the highest priority in mission critical systems so they may modify the scoring by taking the maximum ICRS of 10, which may trigger further steps to be taken. Even if CVEs exists for a microservice or a hosting system's software component, SA can also be performed to validate CVSS, especially for cases when there are very few CVEs registered. Small numbers of discovered vulnerabilities and registered CVEs could be the result of the software being not very popular and not widely used.

Another input for the cyber risk evaluation of microservices and systems that host microservices is penetration testing.

Its engine will be used to find which software has critical vulnerabilities. Penetration testing can be used in conjunction with the SA and CVE-based cyber risk assessment discussed above. Penetration testing is essential for evaluating cyber risks of those components in the system for which there are no CVEs registered and the source code is not available. The methodology to assign cyber risk scores to discovered vulnerabilities can be similar to the CVSS specification [8].

### C. Building Secure Environment for Microservices

Reporting service, shown as Component 4 in Figure 1, gives the final ICRS and TCRS scores for the existing systems ("On-the-fly Re-configuration" mode) or for newly designed systems ("Design from Scratch" mode). VERCASM-M supports "What If" feature so that the user can select different alternatives for the software and see how they impact the TCRS. For example, it may make the system less vulnerable if the OS is updated or downgraded or changed. For some system components, such as web browsers, VERCASM-M can automatically install the less vulnerable version of the browser currently used or install another web browser on-the-fly which results in lowering the TCRS. For some components, it is hard to update them on-the-fly. For example, migrating from Windows to Linux or the other way may be not easy since it would require re-installing and re-configuring many other software components for which alternatives may not exist in other OS or alternatives would cause compatibility issues. In such cases VERCASM-M generates the recommendations of what software should be updated or downgraded or replaced to lower cyber risks. The recommendations include software names and versions. VMs and containerized runtime environments can be evaluated by VERCASM-M as well. Reporting Service hosts a front-end that allows a client to download the reports of final ICRS and TCRS, and recommendations with different alternatives in formats including Javascript Object Notation (JSON), Portable Document Format (PDF), or eXtensible Markup Language (XML).

VERCASM-M supports cyber risk evaluation for newly designed system in "Design from Scratch" mode. The client can select different OS with different versions and build a compatible least vulnerable configuration that includes all the software the client needs. As a future work, in both modes the suggested least vulnerable software components will be checked to be compatible. Furthermore, building least vulnerable configurations for selected list of features will be supported. As an example, a user will be able to build a secure configuration for a Linux-based OS with all the open-source software packages, and with a Relational Database Management System that supports triggers and stored procedures.

## IV. Evaluation

To assess our VERCASM-M cyber risk scoring evaluation model, we compared it with the other model presented in [11]. VERCASM-M model is based on weighted Fibonacci series and assigns higher cyber risk score (8.39 vs. 5.9) for Microsoft® SQL Server 2012, as shown in Table IV. Assigning higher cyber risk scores is more adequate since Microsoft® SQL Server 2012 has a critical vulnerability with CVSS greater than 9.0, and several vulnerabilities with CVSS = 8.8 - see Table V. As shown in Table II, weighted Fibonacci formula puts high weight on critical vulnerabilities so that they significantly increase the ICRS score and draw user's attention. Critical vulnerabilities contribute significantly to the software ICRS, whereas minor vulnerabilities do not have large impact on the ICRS. Weighted Fibonacci-based model allows to achieve that, whereas computing the median or mean value for all CVSS of CVEs for a given software would not achieve that. Assigning the ICRS as a highest CVSS of CVEs for a given software would be too conservative.

| Software Name | Software Version | Max Score | VERCASM ICRS |
|---|---|---|---|
| Microsoft® SQL Server 2012 | 11.1.3128.0 | 5.9 | 8.39 |
| Microsoft® SQL Server 2016 | 2016 | 7.1 | 8.72 |
| Python | 3.4.2 | 8.4 | 8.63 |
| Python | 3.6 | 8.4 | 8.62 |
| Microsoft® .NET® Framework 4 | 4.0.30319 | 8.6 | 9.11 |
| Microsoft® .NET® Framework 4.6 | 4.0.30319 | 8.6 | 9.10 |

TABLE IV: Cyber Risk Score Comparison between Maximum Score Model [11] and VERCASM

| CVE-ID | CVSS ver.3 | CVSS ver.2 |
|---|---|---|
| CVE-2016-7249 | **8.8** | 6.5 |
| CVE-2016-7250 | **8.8** | 6.5 |
| CVE-2016-7251 | 6.1 | 4.3 |
| CVE-2016-7252 | 6.5 | 4.0 |
| CVE-2016-7253 | **8.8** | 6.5 |
| CVE-2016-7254 | **8.8** | 6.5 |
| CVE-2017-8516 | 7.8 | 5.0 |
| CVE-2018-8273 | **9.8** | **10.0** |
| CVE-2019-0819 | 6.5 | 4.0 |
| CVE-2019-1068 | **8.8** | 6.5 |
| CVE-2020-0618 | **8.8** | 6.5 |
| CVE-2021-1636 | **8.8** | 6.5 |

TABLE V: Subset of CVEs for Microsoft® SQL Server 2012

In our second experiment we assessed a wide range of software including OS and programming languages. Results are shown in Table VI. As we can see, the least vulnerable OS if we select between Microsoft® Windows 10, Fedora® Linux® ver. 34, Ubuntu® Linux® ver. 20.04, and OpenSuse® Leap ver. 15.2, is Windows® 10 20H2. Linux® kernel of a

| Software Name | Software Version | ICRS Score |
|---|---|---|
| PHP | 7.4 | 8.90 |
| Node.js | 14 | 5.35 |
| JRE® | 11 | 9.5 |
| MySQL® | 8.0 | 6.49 |
| MongoDB® | 4.2 | 6.47 |
| Debian® | 10 | 8.82 |
| Ubuntu® Linux® | 20.04 | 8.41 |
| Windows® 10 | 20H2 | 8.2 |
| Fedora® Linux® | 34 | 8.92 |
| Linux® Kernel | 5.10 | 7.56 |
| Linux® Kernel | 5.4 | 7.44 |
| OpenSuse® Leap | 15.2 | 8.35 |

TABLE VI: VERCASM ICRS Scores (*Data is taken from [12] Tables 12,13 with permission)*

newer version 5.4 is more secure than Linux® kernel version 5.1. For compared database management systems, MongoDB® is slightly more secure than MySQL® (6.47 out of 10 vs. 6.49 out of 10). Java® Runtime Environment (JRE®) ver. 11 turns out to be vulnerable since its cyber risk score is 9.5 out of 10.

In our third experiment, we evaluated the configurations with one OS (Linux Debian 10 vs. Linux® OpenSUSE® Leap) and three services: MySQL® version 8.0, MongoDB® version 4.2, and JRE® version 11. The results are shown in Tables VII and VIII below. ICRS column shows Individual Cyber Risk Scores, TCRS column shows the Total Cyber Risk Score, "TCRS for JRE® 0777" column shows the TCRS of the configuration if JRE® service runs with full privileges, i.e. 0777 in Unix, which is ".rwxrwxrwx", read-write-execute for everyone. As seen in Table VII, running JRE® service, which is already vulnerable (ICRS score 9.5 out of 10) makes the system configuration extremely vulnerable (TCRS 10 out of 10). This configuration should be avoided since it opens large attack surface for cyber attacks. As shown in Table III, TCRS has to be multiplied by 2, to get 8.27*2 = 16.54. However, since this is at our ceiling and TCRS is in the range between 0.0 to 10.0, we assign the modified TCRS to the maximum cyber risk score of 10.0.

The second configuration represented in Table VIII, is less vulnerable that the first one and its TCRS score is 8.11 out of 10. Running the JRE® with the executable permission of 0760, which is '.rwxrw—-', results in the TCRS score of 8.11 * 1.1, which is 8.92. Thus, the second configuration is less vulnerable (8.92 vs. 10) and more preferable to host services. Looking at Table VI, we can conclude that using Node.js® instead of Java® would make the environment less vulnerable, but for computing systems used in real life it is not always feasible to change the programming language for your service architecture from Java® to Node.js®. Also, changing permissions for JRE® service helps to reduce cyber risk of the system. We can also conclude that Linux® OpenSuse® Leap is less vulnerable than Linux® Debian® 10, based on their public CVEs. Windows® 10 20H2 is, in turn, less vulnerable than Linux® OpenSuse® Leap. However, in real computing system, depending on a specific use case, it might be not possible to migrate from Linux® to Windows-based OS due to compatibility issues and other constraints and conditions.

| Software | ICRS | TCRS | TCRS for JRE® 0777 |
|---|---|---|---|
| Debian® 10 | 8.82 | | |
| JRE® 11 Service | 9.5 | 8.27 | 10 |
| MySQL® 8.0 | 6.49 | | |

TABLE VII: Total Cyber Risk Scores for Vulnerable Environment Configuration (*Data is taken from [12] Tables 12,13 with permission)*

| Software | ICRS | TCRS | TCRS for JRE® 0760 |
|---|---|---|---|
| OpenSuse® Leap | 8.35 | | |
| MongoDB® 4.2 | 6.47 | 8.11 | 8.92 |
| JRE® 11 Service | 9.5 | | |

TABLE VIII: Total Cyber Risk Scores for Less Vulnerable Environment Configuration (*Data is taken from [12] Tables 12,13 with permission)*

## V. Conclusions

In this paper, we presented a methodology to evaluate cyber risks of microservice environments, including software and hardware, in automated and continuous mode. Our methodology considers attributes of microservices and containers. It allows to select less vulnerable software configurations, harden an environment and, thus, protect hosted microservices from cyber attacks. This makes microservice-based architectures more reliable and resilient. We evaluated cyber risk scores for several popular software products and two system configurations and then selected the less vulnerable configuration. This work is in progress and we continue working on penetration testing and static analysis engines and their integration into VERCASM-M.

## Acknowledgements

This project is supported by the Department of Computer Science and Cybersecurity Education, Research, and Outreach Center (CEROC) at Tennessee Technological University.

## References

[1] "Hacker breaks into Florida water treatment facility, changes chemical levels," https://www.securitymagazine.com/articles/94552-hacker-breaks-into-florida-water-treatment-facility-changes-chemical-levels, accessed: June 23, 2022.

[2] "Data breaches expected to cost $5 trillion by 2024," https://www.scmagazine.com/news/research/annual-global-data-breach-costs-to-exceed-5-trillion-by-2024-report, accessed: June 23, 2022.

[3] "Common Vulnerabilities and Exposures," https://cve.mitre.org/, 2022, accessed: June 24, 2022.

[4] B. Northern, T. Burks, M. Hatcher, M. Rogers, and D. Ulybyshev, "Vercasm-cps: Vulnerability analysis and cyber risk assessment for cyber-physical systems," *Information*, vol. 12, no. 10, p. 408, 2021.

[5] "Your Search Engine for Security Intelligence," https://vulners.com, accessed: June 22, 2022.

[6] V. Jain, B. Singh, M. Khenwar, and M. Sharma, "Static vulnerability analysis of docker images," in *IOP Conference Series: Materials Science and Engineering*, vol. 1131, no. 1. IOP Publishing, 2021, p. 012018.

[7] Tenable, "Calculating known and unknown risk: The math behind the cyber exposure score," https://www.tenable.com/whitepapers/calculating-known-and-unknown-risk-the-math-behind-the-cyber-exposure-score, Tenable, Tech. Rep., 2020.

[8] FIRST.Org, Inc., "Common vulnerability scoring system version 3.1: Specification document," 2019, accessed: June 22, 2022. [Online]. Available: https://www.first.org/cvss/v3-1/cvss-v31-specification_r1.pdf

[9] "CVEdetails, Top 50 Products," https://www.cvedetails.com/top-50-products.php, accessed: June 22, 2022.

[10] Microsoft Corporation, "Microsoft exploitability index," 2022, accessed: June 22, 2022. [Online]. Available: https://www.microsoft.com/en-us/msrc/exploitability-index

[11] S. Treetippayaruk and T. Senivongse, "Security vulnerability assessment for software version upgrade," in *2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2017, pp. 283–289.

[12] B. Northern, "Vulnerability analysis and cyber risk assessment," Master's thesis, Tennessee Technological University, 2022, in press.