ARCS-R: Mission Critical Combined Reliability and Cybersecurity Systems Engineering Analysis

Conference Paper · January 2024

DOI: 10.1109/RAMS51492.2024.10457626

Douglas Lee Van Bossuyt
Naval Postgraduate School
135 PUBLICATIONS **937** CITATIONS
SEE PROFILE

Nikolaos Papakonstantinou
VTT Technical Research Centre of Finland
81 PUBLICATIONS **1,045** CITATIONS
SEE PROFILE

Britta Hale
Naval Postgraduate School
21 PUBLICATIONS **167** CITATIONS
SEE PROFILE

Ryan Arlitt
Singapore University of Technology and Design
43 PUBLICATIONS **208** CITATIONS
SEE PROFILE

ARCS-R: Mission Critical Combined Reliability and Cybersecurity Systems Engineering Analysis

Douglas L. Van Bossuyt, PhD, Naval Postgraduate School

Nikolaos Papakonstantinou, PhD, VTT Technical Research Centre of Finland

Britta Hale, PhD, Naval Postgraduate School

Ryan Arlitt, PhD, Unaffiliated

Srinivasa Rao Palatheerdham, University of South Brittany, Lorient

Key Words: Large Language Model, Artificial Intelligence, Machine Learning, LLM, AI, ML, Cyber-Physical System, Failure Rate, Resilience

SUMMARY & CONCLUSIONS

This paper explores how reliability analysis and cyber-security analysis can be combined using Artificial Intelligence and Machine Learning (AI/ML), and Large Language Models (LLM) to produce a continuously updated resilience analysis. This is achieved by modeling both the hardware and software of the system, and employing LLMs and AI/ML to continuously search for new software vulnerabilities and feed that information into continuously updating resilience models. A case study of a drone is presented that demonstrates the promise of the proposed method. It is expected that using the proposed method, named Assessment for Risk in Cybersecurity and Safety - Resilience (ARCS-R), will reduce failure rate of mission-critical cyber-physical systems by reducing the likelihood of a potential initiating event causing a prolonged degradation in system performance that impacts system resilience.

1 INTRODUCTION

Modern cyber-physical mission-critical socio-technical systems must operate in a rapidly evolving and worsening dynamic threat environment. Further, such systems continue to increase in complexity both from a hardware and a software perspective. The introduction of AI/ML has added a new element of risk that can cause systems to fail in new and unexpected ways. Practitioners are challenged to evaluate the combined failure probability of a system when considering both reliability and security from a system resilience perspective.

Safety engineering has exploited decades worth of reliability data on the behavior of materials, components, devices and systems to establish libraries of failure probabilities and mechanisms which were used to develop an extensive and thorough set of failure modelling and safety assessment methods and analyses. Of special interest to this paper is the well-known “bathtub” curve which is a failure rate graph that captures the typical lifecycle reliability phases of a system starting from infant mortality when a system first comes online, and then moves into the random failures phase during the main part of the system’s lifecycle, and lastly moves into the wear-out failures phase near the end of life of the system. Unfortunately, in the security domain from a cyber perspective, the threat environment is very volatile and thus the challenge of risk modelling is more complex and dynamic. A significant part of this challenge is the rapid collection and processing of information as well as a good awareness of the system under study.

While AI/ML technologies have had disruptive effects on the cyber security of systems, recent developments of LLM - based AI have shaken the traditional status quo with disruptive extensions to cyber security threats, assessment, design, and operation. This paper explores what role that LLM – based AI can play in increasing the cadence of failure rate modeling of critical systems. The research presented here focuses on the challenge of modeling a reliability curve for the lifetime of the system, including reliability and cybersecurity events. It has been observed that when a new software intensive cyber-physical system is deployed, there are many vulnerabilities linked to bugs and design/implementation errors that can lead to system failures. Then there is a more secure period where the early issues are addressed (patched/fixed). Towards the end of the system’s life there is again a more dangerous period when system software/hardware components become deprecated/obsolete, people who can maintain/fix the code have left, the attack tools/methods have become better in bypassing defenses, confidential info about the system’s security may have been leaked, etc. This produces a similar picture to the classic “bathtub curve” of system reliability. However, we have observed that in recent years, even well-resourced cyber-physical systems can have premature failure due to cybersecurity issues due to the rapidly increasing velocity of cyberattack development. Often, cybersecurity audits are conducted on a semi-annual basis for industrial cyber-physical systems; however, LLM-based AI may significantly increase the needed frequency of analysis to understand the current failure curve so that appropriate remediation steps can be taken if needed.

The scientific contribution of this paper is an early study of the information needed to extend the traditional reliability bathtub curve to also include cybersecurity-related events. The paper identifies which parameters may be relevant; how to develop dynamic reliability curves based on the cyber threat landscape; whether tools like LLM-based AI, possibly combined with an expert panel of humans, can help to quickly update the reliability curve; and how a high-level overview model of a system can be used to represent the current and forecasted resilience of a system on a per system function basis.

A case study of an internet-connected industrial inspection drone with AI/ML components located at a remote, uncrewed installation is presented. The drone is connected via the open internet and a virtual private network to several resources. The case study shows that the proposed methodology (ARCS-R) can better forecast resilience of the system by combining reliability and cybersecurity analyses.

2 *LITERATURE REVIEW*

This section provides a review of background information and related research necessary to the research in this paper.

2.1 *Resilience engineering*

The engineering of resilient systems has seen a surge in interest over the last several years, especially in the critical infrastructure sectors [1]. Resilience is generally defined as the ability of a system to recover from an event that causes partial or total loss of system performance [2]. In the context of cyber-physical mission-critical systems, resilience is yet another "-ility" that must be balanced when conducting trade-off studies in conceptual systems design [3]. In plain language, resilience is the ability of a system to take a punch, recover, and resume operations at an acceptable level. Generally, one or multiple initiating events (sometimes called disturbances), either at the same time or over a period of time, causes system performance to degrade beyond nominal bounds. Then the system enters a stabilization phase where degradation is stopped, and a new baseline system performance is established – this baseline performance is often well below acceptable performance for the system. Next, the system enters a recovery phase where performance increases until it reaches an acceptable level. Finally, the system enters a post-recovery operation phase where either the system is back to its pre-disturbance performance level or stabilizes at an acceptable performance level.

Resilience improvements are implemented in the pre-disturbance phase before the initiating event has occurred. Engineering analysis of potential initiating events that can challenge a system's resilience is conducted to determine potential failure scenarios that must be addressed. Then engineers develop mitigation strategies to either avoid the initiating event(s), arrest the performance degradation before it reaches an unacceptable level, or implement remediation measures to more quickly recover the system to acceptable levels of performance after the system is stabilized at the new lower baseline [4].

2.2 *Software security*

Today, software is a key component across systems and the interconnections between them, including control of traditionally hardware components. With this dependency comes an added risk: any vulnerability in the code could lead to a vulnerability in the overall system, system failure, and safety hazards for personnel. Software supports automation, monitoring, and overall increased ease of use for systems – but introduces a complexity that would not otherwise be present. Moreover, with multiple code instances running on different components and cross-feeding information, a further increase in system complexity is introduced, and that complexity can make detection of potential vulnerabilities difficult [5]. To address this, multiple approaches and considerations have been introduced and proposed to assist in the vetting of software and continuous monitoring for identification of potential vulnerabilities.

Software security starts at the time of development, which offers an opportunity to build in a zero-trust approach early on [6-8]. A controlled software development platform can control who can develop and add to the code and view it [9-10], minimizing the risk of unintended injects by malicious actors. Version tracking can further support not only the developer in their ability to roll back to older versions of the code or branch off copies, but also source tracking on the software being developed; maintainers can see when and who edited the code and what changes they made, meaning that the possible issues can be more easily determined if an insider threat is eventually identified.

More recently, software tracking and source identification has taken on the term Software Bill of Materials (SBOM) [11]. For a given program, an SBOM should identify who implemented what code and when, what entities had either read or write access throughout the stages of code development, and what security tests were performed to assess the code, as well as SBOMs for any externally developed code that is brought into the development process. Thus, an SBOM may have dependencies on other SBOMs, creating a nested set of documentation for the final end product. Naturally, the existence of an SBOM has little meaning if it is not used; thus, the SBOM should not only be checked by an unbiased security expert before the software is deployed, but it should also be referenced throughout the lifetime of the software and issues come to life (e.g., vulnerabilities are identified that could affect externally developed code dependencies, insider threats who may have had access, etc.).

Another security approach applied during the development process is often termed DevSecOps [12], namely that the developer works with the cybersecurity expert and end operator during the code development. As in other fields, software programming is an expertise of its own and it is a faulty approach to assume that the same person has sufficient security expertise for the needed use case. Furthermore, even if the programming and security approach are sound, they may be insufficiently tailored to the use case, which in itself can create risk hazards. Thus, DevSecOps, as a concept, is born from a motivation to ensure good code is developed in a way that is also secure within the use case needs.

Ultimately, the above methods are intended to minimize software risks before deployment; however, software security does not stop once the coding is complete. Due to potential human error, as well as unanticipated vulnerabilities in the code dependencies or interdependencies once in the deployment system (i.e., zero days), software security is a continuous undertaking. Some companies employ red teams and white hat hackers to try to identify vulnerabilities in the system(s). Bounty hunting, e.g., via gray hat hackers, may also be employed as a means to encourage skilled individuals to report issues that they have found [13]. Finally, if malicious entities, e.g., black hat hackers, identify and try to exploit vulnerabilities in a system and are detected, a company may register the vulnerability themselves. Frequently, companies are under an ethical, business, and even legal obligation to disclose vulnerabilities and patches for them regardless of how the issue was discovered – this enables end-users to patch software and other companies to harden their own systems from attacks. Once such reporting method is through the Common Vulnerabilities and Exposures (CVEs) program originally launched by MITRE [14]. Vulnerabilities that are reported are assigned a CVE number and logged with a public description of the issue, implications, and how it was fixed (if a patch has been deployed).

2.3 *Software source code analysis tools*

Software is an important part of any system and thereby keeping the software secure is the ultimate goal of almost every organization. Yet, we encounter security issues like data leaks, credentials theft, and etc. Static Analysis Security Testing (SAST) helps in preventing security issues while the software is in its development phase. SAST analyses source code for vulnerabilities also it helps developers who are from non-security background to keep the source code secure and also avoid insecure programming patterns [15].

Tools like Snyk help in such SAST analysis by providing Scans in early development phase, to indicate the issue in the source code also explain the issue found. They help in identifying the issues without the need for executing the code. It also helps in automating the scanning of source code files at any point of Software Development Life Cycle (SDLC) [16].

SAST also checks for code issues & security vulnerabilities, quality of documentation, consistency with overall software design, compliance and best programming practices, violation of rules that affect code execution, quality aspect of software like system complexity or maintainability.

2.4 *Vulnerability forecasting*

Vulnerability forecasting is done based on the volume of CVEs being produced within discrete time spans and it requires only 8% of the value between now and a year from now. There are multiple predictive algorithms to predict data for monthly, quarterly, annually, and other periods. The main forecasting is done with the data of publicly disclosed vulnerabilities. The primary purpose of CVEs is to reduce the risk in vulnerability management. The vulnerability of computers or networks can be predictive based on the unpatched vulnerabilities. There can also be many undiscovered/disclosed vulnerabilities in software which can occur during software development lifecycle or can be intentional too to make the quality team find but eventually they were unidentified. The other type of vulnerabilities can occur due to vendor specific product/software.

In [17] the authors strictly took the data from 2016 and training data up to the end of 2015. The 2016 data employs root mean squared percentage error (RMSPE) for validation and includes larger categories, including all vulnerabilities, browser CVEs, operating system CVEs, and video codec CVEs. metric achieving a median of 15% RMSPE for global vulnerability predictions over 24 months. This approach predicts cumulative CVEs of all time which makes the percentage of error metrics not consistent over time.

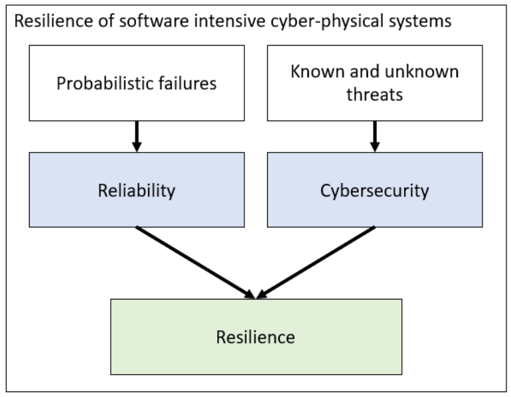
To predict the number of CVEs as a whole as well as sub-types of CVEs across different time spans (1 month, 3 months, half yearly, and annually) different models are required. Predictive model can be a good method but limited to lab data set only. Performing standard time series and forecasting analysis using predictive models which involves both statistical and machine learning models. To exploit the metadata from the CVE database minimum variance unbiased estimator (MVUE) and Little's Law exit rate [18] are used.

In the case of Adobe Acrobat Reader, the Mean Absolute Error for a period of 12 months was 163 with combined model. For Microsoft Windows 10 it was 156. Similarly for Linux Kernal it was 191.

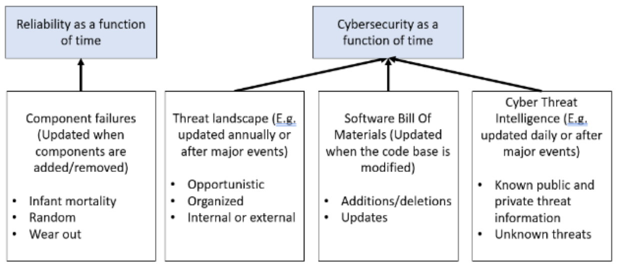
Thus, the choice of predictor is validation set-based even for the combined model.

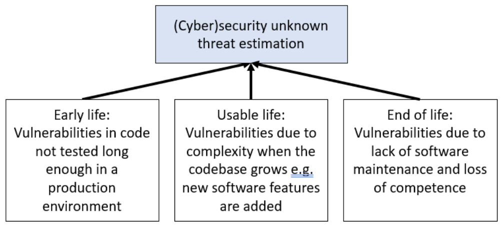
3 METHODOLOGY

The backbone of the proposed ARCS-R methodology is the combination of safety and security towards an overall resilience risk assessment as shown in Fig. 1. While the combined system failure probability due to component failures is a function of time (shown as phases of the classic "bathtub" failure rate graphic [19]), in the cyber world the conditions are much more


Resilience of software intensive cyber-physical systems
Probabilistic failures
Known and unknown threats
Reliability
Cybersecurity
Resilience
Figure 1: The Combination of Reliability and Cyber-Security into Resilience

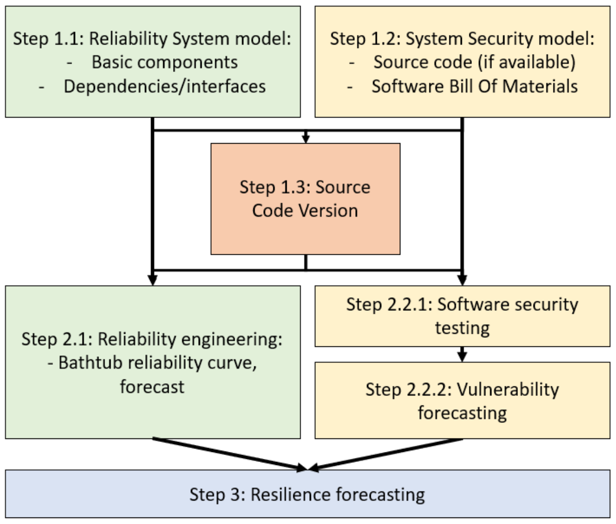
dynamic. The threat landscape, the software configuration, and new cyber threat intelligence inputs can substantially affect the overall cyber risk assessment of the system as shown in Fig. 2. Apart from the known vulnerabilities, unknown ones can be expected to exist, taking into consideration the lifecycle phase of the software used (early versions vs mature releases vs near end-of-life usage), its complexity, its popularity, and its attractiveness as a target, as shown in Fig. 3.


Reliability as a function of time
Cybersecurity as a function of time
Component failures (Updated when components are added/removed)
• Infant mortality
• Random
• Wear out
Threat landscape (E.g. updated annually or after major events)
• Opportunistic
• Organized
• Internal or external
Software Bill Of Materials (Updated when the code base is modified)
• Additions/deletions
• Updates
Cyber Threat Intelligence (E.g. updated daily or after major events)
• Known public and private threat information
• Unknown threats
Figure 2: Reliability/Cyber-security as a function of time.


(Cyber)security unknown threat estimation
Early life: Vulnerabilities in code not tested long enough in a production environment
Usable life: Vulnerabilities due to complexity when the codebase grows e.g. new software features are added
End of life: Vulnerabilities due to lack of software maintenance and loss of competence
Figure 3: Cybersecurity – unknown threats

The proposed ARCS-R methodology shown in Fig. 4 contains two parallel tracks, one related to safety/reliability and one for cyber security. Step 1.1 is part of the traditional safety engineering work and more specifically the development of a model (topology, behavior and internal/external interactions) to be used for calculating the reliability of the system. Step 1.2 calls for a similar activity, but this time focusing on the cyber aspect of the system. Step 1.3 provides a source code version of 1.1. and 1.2, namely moving the reliability assessment a further layer down into the code itself. In Step 2.1 the practitioner performs a reliability assessment given the system model and information about the components to estimate the failure rate graph. In the cybersecurity side, the software is tested in Step 2.2.1 using white box, black box or grey box techniques [20] to identify as many as possible existing vulnerabilities. After the security testing process has been performed over a large enough period of time, a vulnerability forecasting method like the Vuln4Cast [17] can be used in Step 2.2.2 to provide a statistical prediction of the vulnerabilities to be discovered in the future.

The LLM code generation foundation model can be trained on the language(s) of our software components, and can be fine tuned on those software components themselves. The prompt to the LLM can include the particular piece of code to be analyzed along with a general request to search for and patch vulnerabilities. Vulnerability databases are included via Retrieval Augmented Generation (RAG). A second adversarial LLM can be used to write and/or run tests to demonstrate the performance of the new code – that it still works as intended and that the vulnerability no longer exists. It then falls to the practitioner to assess the outcomes and provide a circuit breaker against automating the introduction of new defects. In the


Step 1.1: Reliability System model:
- Basic components
- Dependencies/interfaces
Step 1.2: System Security model:
- Source code (if available)
- Software Bill Of Materials
Step 1.3: Source Code Version
Step 2.1: Reliability engineering:
- Bathtub reliability curve, forecast
Step 2.2.1: Software security testing
Step 2.2.2: Vulnerability forecasting
Step 3: Resilience forecasting
Figure 4: Methodology

context of the ARCS-R methodology, this approach can be performed over the entire software bill of materials and used to create reliability and security statistics over the full system.

Step 2.2.1 is interesting to describe further. In white box software security testing, a tool like the Shift Left Scan [21] can perform a static application security testing by analyzing the source code for potential vulnerabilities. The same tool supports black box testing where the goal is to discover vulnerabilities in the functionalities of the software while it is running. In addition, a dependency-based tool like SNYK [22] can discover vulnerabilities by checking the library dependencies of a software project against Common Vulnerabilities and Exposures (CVEs) databases (like VulDB [23] or NIST/NVD [24]) and then compile a report summarizing the results, often with a priority label based on a scoring method like the Common Vulnerability Scoring System (CVSS) [25].

In step 2.2.2 a tool like Vuln4Cast can be used to forecast vulnerabilities monthly, quarterly, or yearly based on the vulnerabilities discovered so far. This method is still in active development and its predictive algorithms have been used for software products like operating systems (Windows, MacOSX, Linux kernel) as well as smaller products like Adobe Acrobat [17]. Its applicability is constrained mainly on the availability of a large enough data set of discovered CVEs over time.

The results of Step 2.1 and 2.2.2 are combined in Step 3 to provide an overall resilience forecast.

4 CASE STUDY

A case study of an internet-connected industrial inspection drone with AI/ML components is now presented. The drone is connected via the open internet back to the manufacturer for usage data, maintenance and diagnostic purposes, and asset management. In addition, it is connected via a company virtual private network to a remote control facility where the drone's data is processed and analyzed. The drone conducts periodic assessment and site surveys of a remote industrial facility that typically operates without a human presence such as petroleum production fields, flood control infrastructure, or similar. It follows a fixed patrol route that it can optionally deviate from when an anomaly warranting further inspection is detected. A traditional bathtub curve reliability analysis indicates the drone can operate at the remote industrial facility for several years with only minimal maintenance. However, the increased cadence of cyberattacks has led to the potential of cybersecurity related vulnerabilities that could impact the AI/ML components aboard the drone. This could cause the drone to malfunction and fail which could lead to both the loss of the drone as an asset and also damage to the industrial facility. The company currently does a security audit every six months and updates the AI/ML and cyber components on an as-needed basis based on the security audit. However, the company lacks resources for in-depth cybersecurity assessment related to the drone, as it is a minor part of a complex system within the management scope. Thus, an LLM-based AI is then introduced to continuously scan the internet for information related to the cyber components aboard the drone as well as any specific AI-based threats that should be considered at the next audit. The LLM-based AI further can analyze how CVEs impact system functionality, and produces a new reliability curve to help communicate information to the team tasked with keeping the drone secure and operational. This allows the team to more rapidly address cybersecurity threats and delay addressing threats that will not cause an immediate threat to the drone system. The case study next picks up at Step 1 of the proposed ARCS-R methodology.

Step 1.1: Reliability System Models

System reliability models are developed of the drone. A reliability block diagram is developed, individual component reliability data is either identified from existing data sources or developed through accelerated lifecycle testing. Dependencies and interfaces between the system hardware (e.g.: motors, batteries, controllers, etc.) and the system software are identified.

Step 1.2: System Security Models

The system interaction, including communication links, user interfaces, device internal software module interactions, is modeled to identify device internal dependencies and to provide a foundational design vulnerability analysis. While code review in later steps will be used to identify issues in the code itself, this step ensures that the type of code is appropriate to the system design and threat model.

Step 1.3: Software System Models

The source code for the drone is collected and analyzed to develop a software bill of materials. This is a common cybersecurity requirement and audit method found in industry and government guidance. The software system models identify package dependencies as well as any published vulnerabilities in those dependencies. This step does not include any new software analysis, but identifies known vulnerabilities. The package dependencies can be used in the future to scan for new vulnerabilities as they are identified in the software packages.

Step 2.1: Reliability Engineering

The reliability models indicate that the drone's reliability follows the classic "bathtub curve" where, after an initial "infant mortality" phase, the drone is expected to have a long service life with low random failure. At the end of the service life, the drone enters the "wear-out" phase of reliability where components increasingly fail as the reach end of life.

Step 2.2.1: Software Security Testing

Next software security testing is performed. This goes hand-in-hand with the AI/ML model development, any new software development, and new software analysis tests on the dependencies identified in Step 1.2. This may include use of fuzz testing tools and an AI/ML vulnerability analysis.

Step 2.2.2: Vulnerability Forecasting

Vulnerability forecasting is now conducted. In this case study, ShiftLeft code analysis is used. However, many different vulnerability techniques are available in the literature.

ShiftLeft code analysis does various sorts of analysis using a single code representation known as a "code property graph." Various approaches like Data Flow analysis, Control Flow analysis, Taint analysis, Lexical analysis, Configuration analysis, and Buffer overflow analysis, are typically used to do static analysis. On the other hand, ShiftLeft starts by constructing the code property graph (CPG), a single data structure. Through this single data structure (code property graph), it then does all of the aforementioned analysis on the complete code base (third-party libraries, open-source code), achieving enormous time/memory savings and thus saving time on code analysis. It's possible that a Shift Left testing strategy can't always give the best performance and functionality in a real-world setting. A Shift Right testing technique may be helpful in these circumstances.

The benefits of ShiftLeft scan can be Faster delivery, Improved security posture, Reduced costs, improving security integration and pace, Enabling trust in the overall software. Additionally, ShiftLeft scan reduces cost involved in fixing issues lately in CI/CD pipelines.

The set of open-source projects scanned using Snyk for dependency-based analysis shows us that the vulnerabilities for open-source code or software code are generally dependent on dependencies too. The more the dependencies which have unsolved vulnerabilities, the more insecure the program or source code becomes thereby affecting the system in which the software present or works for. In turn this can affect the entire network or production systems which when used for Live systems without patching vulnerabilities.

Step 3: Resilience Forecasting

Finally, the results of the reliability engineering and vulnerability forecasting steps are combined. Fig. 5 shows how the classic reliability "bathtub" curve can change with vulnerability forecasting, and further shows how the reliability curve can change over time as the LLM continuously scans for new software vulnerabilities. This can be fed into a resilience calculation to develop updated resilience information as shown in Fig 6.

5 CONCLUSION AND DISCUSSION

This paper has proposed a new way of automatically assessing reliability and vulnerability of cyber-physical systems to develop new resilience analyses – the ARCS-R method. This allows for potential critical vulnerabilities to be patched rapidly and provides an understanding of how emergent vulnerabilities impact reliability and resilience. Using AI/ML and LLMs to continuously scan for new vulnerabilities and then automatically update reliability and resilience analysis shows promise as a method to keep critical systems working.

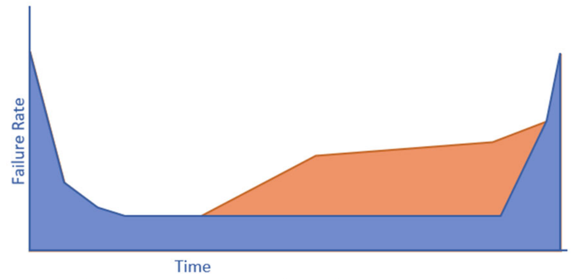


Figure 5: Failure rate without vulnerability (blue) and with vulnerability (orange) included

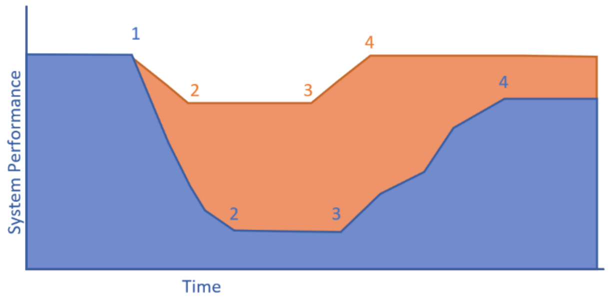


Figure 6: Resilience curve of system without vulnerabilities assessed (blue) and with vulnerabilities assessed (orange). Numbers indicate specific regions of resilience (1: initiating event, 2: start of new baseline, 3: start of recovery, 4: recovered).

ACKNOWLEDGEMENTS

The views presented are those of the authors and do not necessarily represent the views of the US Department of Defense, US Navy, VTT Technical Research Centre of Finland, or other organizations. This work was partially supported by the AI-NET-ANTILLAS project (CELTIC-NEXT, C2019/3-3), funded by Business Finland. Approved for Public Release; distribution is unlimited.

BIOGRAPHIES

Douglas L. Van Bossuyt, PhD
Department of Systems Engineering
Naval Postgraduate School
e-mail: douglas.vanbossuyt@nps.edu

Douglas L. Van Bossuyt is an associate professor of systems engineering at the Naval Postgraduate School in Monterey, California. He earned his doctorate in mechanical engineering at Oregon State University in 2012. His research focuses on the nexus of risk and failure analysis, RAMS, and systems design methods. He earned his honors bachelor's of science in mechanical engineering and honors bachelor's of arts in international studies (2007), and his master's of science in mechanical engineering (2009) at Oregon State University.

Nikolaos Papakonstantinou, D.Sc. (Tech.), Docent
VTT Technical Research Centre of Finland
e-mail: nikolaos.papakonstantinou@vtt.fi

Nikolaos Papakonstantinou is an electrical and computer engineer (Univ, of Patras/Greece, 2008), has a doctorate degree in information technology in automation from Aalto University, Finland (2012) and he is a docent in the field of information technologies in industrial applications (2020). He is leading the Applied cybersecurity team at the VTT Technical Research Centre of Finland. VTT is a large non-profit research organization with both commercial and public research activities. The interests of the team include security training, device testing, security design/architectures, platform security as well as holistic security assessment of industrial systems and other critical infrastructure. Papakonstantinou's personal interests focus on early resilience (safety/security) engineering for complex sociotechnical systems.

Britta Hale, PhD
Department of Computer Science
Naval Postgraduate School
e-mail: britta.hale@nps.edu

Britta Hale is a cryptographer and Assistant Profession in the Computer Science Department at the Naval Postgraduate School in Monterey, California. She holds a PhD from the Norwegian University of Science and Technology and a Master's of Science from Royal Holloway University of London. Dr. Hale's research areas include cryptographic protocol design and analysis, applications to uncrewed systems and counter-uncrewed systems, and security for other emerging environments and technologies.

Ryan Arlitt, PhD
Unaffiliated
e-mail: arlitt.ryan@gmail.com

Ryan Arlitt is an unaffiliated researcher. Previously he was an assistant professor of mechanical engineering in the Department of Civil and Mechanical Engineering at the Technical University of Denmark. He holds a PhD in mechanical engineering (Oregon State University), a MS in Systems Engineering (Missouri University of Science and Technology), and a BS in Interdisciplinary Engineering (Missouri University of Science and Technology). His research focus is on design methods in the fuzzy front end, at the interface of human expertise and computational tools.

Srinivasa Rao Palatheerdham
Masters in Cyber Security
University of South Brittany, Lorient
e-mail: palatheerdham.e2205110@etud.univ-ubs.fr

After finishing his bachelors in Computer Science and Engineering from JNTU Karimnagar, India, he worked for 3 MNCs as a Middleware administrator, Lead Product Support engineer and Application administrator in India for 9 years. His bachelors thesis involves Verification through Iris recognition. His thirst for Cyber Security made him apply for different programs in Europe and thereby getting into prestigious Erasmus Mundus program from University of South Brittany, Lorient during 2022. His interest in the field of Cyber Security is mainly into threat information landscape and GRC (Governance, Risk, and Compliance) Risk management.

REFERENCES

1. R.E. Giachetti, D.L. Van Bossuyt, W.W. Anderson, & G. Oriti, "Resilience and cost trade space for microgrids on islands," *IEEE Systems Journal*, 16(3), pp 3939-3949, Sept. 2021. Available: https://doi.org/10.1109/JSYST.2021.3103831
2. E. Anuat, D.L. Van Bossuyt, & A. Pollman, "Energy resilience impact of supply chain network disruption to military microgrids," *Infrastructures*, 7(1), 4. Dec. 2021. Available: https://doi.org/10.3390/infrastructures7010004
3. D. L. Van Bossuyt, I.Y. Tumer, & S.D. Wall, "A case for trading risk in complex conceptual design trade studies," *Research in Engineering Design*, 24, pp. 259-275. July 2013. Available: https://doi.org/10.1007/s00163-012-0142-0
4. N. Papakonstantinou, B. Hale, J. Linnosmaa, J. Salonen, & D.L. Van Bossuyt, "Model driven engineering for resilience of systems with black box and ai-based components." *In 2022 Annual Reliability and Maintainability Symposium (RAMS).* pp. 1-7. IEEE. Jan. 2022. Available: https://doi.org/10.1109/RAMS51457.2022.9893930
5. McCabe Software, Inc. "More Complex = Less Secure: Miss a Test Path and You Could Get Hacked." 2021. Available: http://www.mccabe.com/pdf/More%20Complex%20Equals%20Less%20Secure-McCabe.pdf
6. N. Papakonstantinou, D.L. Van Bossuyt, J. Linnosmaa, B. Hale, & B. O'Halloran, "A Zero Trust Hybrid Security and Safety Risk Analysis Method," *Journal of Computing and Information Science in Engineering*, 21(5): 050907, Oct. 2021. Available: https://doi.org/10.1115/1.4050685
7. B. Hale, D. L. Van Bossuyt, N. Papakonstantinou, & B. O'Halloran, "A Zero-Trust Methodology for Security of Complex Systems With Machine Learning Components," *Proceedings of the ASME 2021 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Volume 2: 41st Computers and Information in Engineering Conference (CIE). Virtual, Online. August 17–19, 2021.* Available: https://doi.org/10.1115/DETC2021-70442
8. S. Rose, O. Borchert, S. Mitchell, S. Connelly, "Zero trust architecture." NIST. Gaithersburg, MD. Special Publication 800-207. 2020. Available: https://doi.org/10.6028/NIST.SP.800-207
9. A. W. Brown, M. Delbaere, P. Eeles, S. Johnston and R. Weaver, "Realizing service-oriented solutions with the IBM Rational Software Development Platform," *IBM Systems Journal*, vol. 44, no. 4, pp. 727-752, 2005, doi: 10.1147/sj.444.0727.
10. A. Stuckenholz, "Component evolution and versioning state of the art." *ACM SIGSOFT Software Engineering Notes* Vol. 30, no. 1. 2005. Available: https://doi.org/10.1145/1039174.1039197
11. N. Zahan, E. Lin, M. Tamanna, W. Enck, & L. Williams. "Software Bills of Materials Are Required. Are We There Yet?" *IEEE Security & Privacy* 21, no. 2 pp. 82—88. 2023. Available: https://doi.org/10.1109/MSEC.2023.3237100
12. M.A. Akbar, K. Smolander, S. Mahmood, and A. Alsanad. "Toward successful DevSecOps in software development organizations: A decision-making framework." *Information and Software Technology* 147: 106894, 2022. Available: https://doi.org/10.1016/j.infsof.2022.106894
13. M. Ashong, "Bug the Bounty Hunter: Recommendations to Congress to Best Effectuate the Purpose of the Secure Technology Act." *Public Contract Law Journal*, 49:173, 2019.
14. CVE. "CVE Program Mission: Identify, define, and catalog publicly disclosed cybersecurity vulnerabilities." Accessed June 2023. Available: https://www.cve.org/
15. B. Chess, & G. McGraw, "Static analysis for security," *IEEE Security & Privacy*, 2(6), 76-79. 2004.
16. Y.B. Leau, W.K. Loo, W.Y. Tham, & S.F. Tan, "Software development life cycle AGILE vs traditional approaches," *Proceedings of International Conference on Information and Network Technology*, Vol. 37, No. 1, pp. 162-167. 2012.
17. É. Leverett, M. Rhode, & A. Wedgbury, "Vulnerability Forecasting: Theory and practice," *Digital Threats: Research and Practice*, 3(4), 1-27. 2022. Available: https://doi.org/10.1145/3492328
18. J.D. Little, & S.C. Graves, "Little's law," in *Building Intuition. International Series in Operations Research & Management Science* D. Chhajed, T.J. Lowe, vol. 115, pp. 81-100. 2008. Available: https://doi.org/10.1007/978-0-387-73699-0_5
19. G.A. Klutke, P.C. Kiessler, & M.A. Wortman, "A critical look at the bathtub curve," *IEEE Transactions on reliability*, 52(1), 125-129. Mar. 2003. Available: https://doi.org/10.1109/TR.2002.804492
20. O.B. Tauqeer, S. Jan, A.O. Khadidos, A.O. Khadidos, F.Q. Khan, & S. Khattak, "Analysis of security testing techniques," Intelligent Automation & Soft Computing, 29(1), 291-306. 2021. Available:

https://doi.org/10.32604/iasc.2021.017260
21. G. Alvarenga, "Shift Left Security Explained," CrowdStrike, 11 Jan. 2022. Available: https://www.crowdstrike.com/cybersecurity-101/shift-left-security/
22. Synk Limited, "Developer Security, Develop Fast," Accessed: 12 July 2023. Available: https://snyk.io/
23. Pyxpy inc. "VulDB," Accessed: 10 July 2023. Available: https://vuldb.com/
24. NIST, "National Vulnerability Database," Accessed: 8 July 2023, Available: https://nvd.nist.gov/
25. Forum of Incident Response and Security Teams, Inc., "Common Vulnerability Scoring System SIG," Accessed: 14 July 2023. Available: https://www.first.org/cvss/