Evaluating the top application security tools: from static analysis to runtime protection

Abstract. This review article evaluates the effectiveness of application security tools, including static analysis techniques and runtime protection mechanisms, against the backdrop of the growing global cybersecurity market and evolving cyber threats. Through a comprehensive review, the study aims to assist developers, security professionals, and organizations in selecting the most effective tools to enhance application security. Employing a mix of theoretical analysis and empirical benchmarking, the paper analyzes static application security testing (SAST), dynamic application security testing (DAST), and runtime application self-protection (RASP) technologies. Findings indicate that while SAST tools are essential for early vulnerability detection, they may generate false positives and overlook runtime vulnerabilities. DAST tools, in contrast, effectively identify runtime issues but lack insight into internal application processes. RASP technologies offer real-time protection but face integration and performance challenges. The paper concludes with a recommendation for a layered security approach, combining SAST, DAST, and RASP tools to achieve comprehensive application security, thus contributing a novel perspective to the discourse on cybersecurity tool efficacy.

Keywords: application security, cybersecurity, SAST, DAST, RASP, vulnerability detection, software development lifecycle, real-time protection, cyber threats, security tools.

Introduction

In the digital age, application security has become a paramount concern for organizations around the world. With the increasing dependence on software applications in mission-critical operations, the need to protect these applications from potential threats and vulnerabilities has become more important than ever. The global cybersecurity market size was valued at USD 222.66 billion in 2023 and is projected

to grow at a compound annual growth rate (CAGR) of 12.3% from 2023 to 2030 (Figure 1) [1,17,18,19]. The objective of this paper is to evaluate application security tools, ranging from static analysis techniques to runtime protection mechanisms, and to provide a comprehensive review of their effectiveness in enhancing application security.

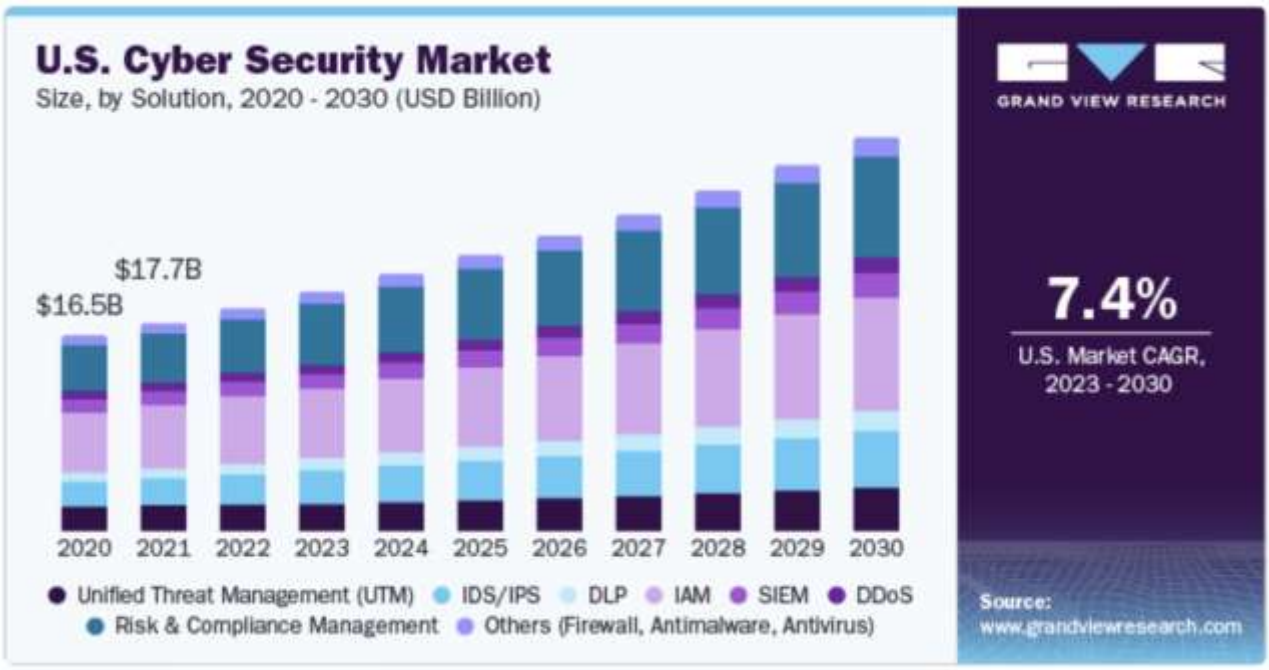

Figure 1 - Cybersecurity market size, 2020 - 2030 [1]

The importance of application security cannot be overemphasized given the changing landscape of cyber threats that organizations face today. Cyber attackers are constantly developing sophisticated methods to exploit vulnerabilities in applications, resulting in unauthorized access, data leakage, and other malicious activities (Table 1). In response, the field of application security has grown, offering a variety of tools designed to identify, mitigate, and prevent security weaknesses throughout the software development life cycle (SDLC).

Table 1: Types of Cyber Threats

Type of Cyber Threat	Description
Injection	Malicious code is injected into the program, leading to unauthorized access or data loss.

Broken Authentication	Attackers exploit weak or stolen credentials to gain unauthorized access.
Sensitive Data Exposure	Sensitive data such as financial, healthcare, or personal information is exposed to unauthorized parties.
XML External Entities (XXE)	Attackers exploit poorly configured XML processors to execute unauthorized commands or access data.
Broken Access Control	Users gain unauthorized access to certain functionalities or data, violating access controls.
Security Misconfiguration	Misconfigurations in security settings that leave applications vulnerable to attacks.
Cross-Site Scripting (XSS)	Malicious scripts are injected into content that is then viewed by other users, compromising their data or user experience.
Insecure Deserialization	Attackers exploit serialization flaws to execute malicious commands or access unauthorized data.
Using Components with Known Vulnerabilities	Applications use components with known vulnerabilities, exposing them to potential exploits.
Insufficient Logging & Monitoring	Failures in logging and monitoring allow attackers to further exploit vulnerabilities without detection.

This study is motivated by the need to understand the effectiveness of different application security tools in detecting and remediating security issues. By comparing static analysis tools, dynamic analysis tools, and runtime security solutions, this paper seeks to provide insight into their relative merits and limitations. It also aims to help developers, security professionals, and organizations select the most appropriate tools to improve the security of their applications.

The goals of this paper are twofold. First, it is to critically analyze the current state of application security tools, including static and dynamic analysis techniques and runtime security measures. Second, it aims to propose a set of criteria for evaluating these tools, thereby helping stakeholders to make informed decisions about their implementation and use.

1. Theoretical aspects of application security

The field of application security encompasses a wide range of techniques, tools, and practices aimed at preventing unauthorized access and modifications, ensuring confidentiality, integrity, and availability of application resources and data [6].

Application security threats can manifest themselves in various forms, ranging from injection vulnerabilities such as SQL injection to more sophisticated attacks such as cross-site scripting (XSS) and cross-site request forgery (CSRF), as outlined in Table 1. These vulnerabilities appear mainly in the design and development phases of the software development life cycle (SDLC). The Open Web Application Security Project (OWASP) periodically publishes a list of the ten most critical web application security risks to serve as a guide for developers and security professionals in identifying and remediating common threats (Figure 2) [2,7].

Figure 2 - The OWASP TOP 10 Web Application Threats [2]

Securing applications requires a multifaceted approach that integrates security practices throughout the SDLC, from initial design through deployment and maintenance. This approach, often referred to as secure by design, emphasizes incorporating security measures early in the development process rather than at the end.

Techniques such as threat modeling, risk assessment, and secure coding practices are very important in this regard. In addition, adherence to security standards and frameworks such as ISO/IEC 27001 provides a structured methodology for managing and improving security processes [8].

ISO/IEC 27001 is a widely recognized international standard for information security management. Published jointly by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC), it defines the requirements for establishing, implementing, maintaining and continuously improving an information security management system (ISMS). The standard is intended to ensure that adequate and proportionate security controls are selected that protect information assets and provide assurance to stakeholders, especially customers.

ISO/IEC 27001 is built on a process-based approach to establishing, operating, monitoring, reviewing, maintaining and improving an organization's ISMS. It uses the Plan-Do-Check-Act (PDCA) cycle to ensure continuous improvement in a systematic and ongoing manner. This approach is not only systematic and iterative, but also ensures that the ISMS is integrated into the organization's processes and overall governance structure [3].

In the application security arena, various testing methodologies are used to identify and remediate vulnerabilities. These methods, including static application security testing (SAST), software composition analysis (SCA), dynamic application security testing (DAST), mobile application security testing (MAST), interactive application security testing (IAST), and runtime application self-protection (RAST), each fulfill different roles in the software development life cycle (SDLC) and utilize unique operating principles (Figure 3) [4,5,9].

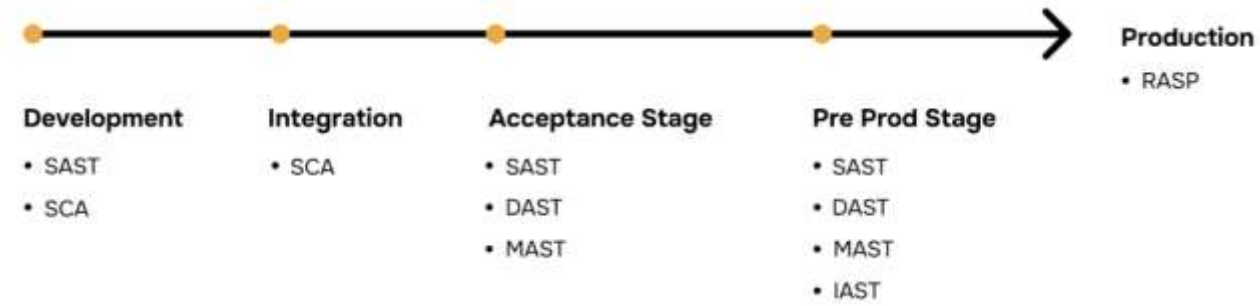

Figure 3 - Security checks in the application development cycle

Static application security testing (SAST) involves examining an application's source code without executing it to identify potential security vulnerabilities. This automated process can detect issues ranging from syntax errors to complex security flaws such as buffer overflows and SQL injection vulnerabilities. By integrating static code analysis tools into the development process, organizations can identify and fix security issues early, significantly reducing the risk of vulnerabilities in the final product.

Unlike static code analysis, dynamic application security testing (DAST) evaluates an application in a running state by simulating attacks and testing it for vulnerabilities. This approach is particularly effective in identifying runtime issues that static analysis may miss, such as authentication and authorization errors, session management issues, and vulnerabilities specific to the application's runtime environment. Dynamic analysis tools, often part of a broader application security testing (AST) strategy, play a critical role in securing complex interactive applications.

DAST can also shed light on runtime issues such as:

- authentication and server configuration issues
- flaws visible only when a known user logs in

Interactive Application Security Testing (IAST) combines elements of static and dynamic analysis, tracking the runtime behavior of an application to identify vulnerabilities with high accuracy. Using information from the application's runtime environment, IAST tools can provide context-sensitive information about security issues, facilitating targeted remediation efforts. This hybrid approach bridges the gap between static and dynamic analysis, offering a comprehensive view of an application's security posture.

Because IAST runs inside the application, it can analyze:

- application code
- data flows
- configurations

- HTTP requests and responses
- Libraries, frameworks, and other components
- internal connection information

The final stage of testing is runtime application self-protection (RASP), referred to in some contexts as RAST (runtime application security testing), is an innovative security technology designed to protect applications from within at runtime. Unlike traditional security measures that operate externally, RASP integrates security features directly into the application, allowing it to detect and defend against attacks in real-time. This approach provides a significant advancement in application security by providing immediate protection against the exploitation of vulnerabilities, malicious inputs, and other attack vectors.

RASP works by embedding or integrating security features into the application runtime environment. This integration allows RASP to have insight into the application's data flow, control flow, and operational context, enabling it to accurately detect and mitigate attacks [10].

Table 2: The comparison of Features SAST, DAST, and RASP

Feature	SAST	DAST	RASP
Method of Operation	Examines source code without executing it	Evaluates running applications by simulating attacks	Integrates into the application to detect and defend against attacks in real-time
Stage in SDLC	Early in the development cycle	Post-development, pre-production	Production/runtime
Type of Issues Detected	Syntax errors, security flaws such as buffer overflows, SQL injections	Runtime issues, authentication/authorization errors, session management issues	Real-time attacks, malicious inputs, runtime vulnerabilities
Integration	Integrated into the development process	Part of broader AST strategy, used in staging environments	Embedded within the application, operates in production environment
Granularity	Examines code at a granular level	Evaluates the application as a whole	Monitors and protects at runtime, providing contextual insights

Remediation Assistance	Provides early detection and fixing of issues	Identifies issues in a running state, providing context for runtime vulnerabilities	Provides immediate protection and mitigation
Coverage	Source code, configuration files	HTTP requests, responses, session data	Data flow, control flow, internal connection information
False Positives	Can be higher due to lack of runtime context	Generally lower as it evaluates actual runtime behavior	Low, as it operates in the actual runtime environment
Advantages	Early detection, wide range of detectable issues	Effective in identifying runtime-specific vulnerabilities	Immediate and continuous protection, context-aware defense
Disadvantages	May miss runtime-specific issues	Requires a running application, potential for environmental dependencies	Overhead on application performance, complexity in integration

Each of these methodologies plays a critical role in a comprehensive application security strategy, addressing different aspects of application vulnerabilities at different stages of the SDLC. The selection and implementation of these methodologies depends on the specific requirements, risks, and resources of the development project, emphasizing the importance of a layered and informed approach to application security.

2. Theoretical aspects of application security

The application security field has developed many tools to identify, remediate, and prevent vulnerabilities in software applications.

The selection of application security tools should be guided by several key criteria that reflect the specific needs and context of software development projects. These criteria include the ability to detect a wide range of vulnerabilities, the accuracy of vulnerability identification (to minimize false positives and negatives), the ability to integrate with development environments and CI/CD pipelines, scalability to handle large code bases, and the provision of actionable recommendations for remediation. In

addition, cost is an important consideration for organizations, both in terms of financial investment and impact on development timelines.

3. Overview of application security tools

Static Application Security Testing (SAST) tools analyze the source code, bytecode, or binary code of applications without executing them. These tools help detect vulnerabilities early in the software development life cycle (SDLC), including but not limited to code injection, cross-site scripting (XSS), and insecure authentication mechanisms [11,12].

- **SonarQube**: SonarQube offers comprehensive code quality and security analysis, identifying bugs, and code vulnerabilities in different programming languages. It is preferred for its detailed feedback and integration with various CI/CD tools.
- **Fortify Static Code Analyzer**: HP Fortify provides deep insight into application vulnerabilities, supporting a wide range of languages and frameworks. Its ability to integrate security testing into development and deployment processes makes it a valuable tool for enterprises.

Dynamic Application Security Testing (DAST) tools evaluate applications from an external perspective, focusing on running application instances to identify runtime vulnerabilities such as session management issues and authentication/authorization flaws.

- **OWASP ZAP (Zed Attack Proxy)**: ZAP is an open source DAST tool that offers automated scanners as well as tools for manual vulnerability assessment. It is especially useful for developers and functional testers who are just beginning penetration testing [13].
- **Burp Suite**: Burp Suite is a comprehensive web application security testing solution with an intuitive interface and powerful scanner. Its detailed analysis and reporting capabilities make it popular among security professionals.

Runtime Application Self-Protection (RASP) tools provide continuous security by monitoring application behavior in real-time and preventing or mitigating attacks as

they occur. RASP solutions integrate into the application runtime environment and provide protection against a range of threats without requiring changes to application code [14].

- **Contrast Security**: Contrast Security offers a RASP solution that utilizes tools to detect and prevent attacks in real-time, providing visibility into application usage and attack attempts. This solution is characterized by ease of use and minimal impact on performance.
- Imperva **RASP**: Imperva aims to protect applications from known and unknown vulnerabilities by monitoring and analyzing application traffic and behavior. It offers robust defense mechanisms against attacks such as SQL injection and cross-site scripting.

The effectiveness of these tools depends on the specific security requirements, development practices, and operating environment of organizations. Static analysis tools are important for the early detection of vulnerabilities, while dynamic analysis tools are important for identifying runtime issues that static tools may miss. RASP solutions provide an additional layer of protection by actively monitoring and protecting applications from runtime attacks.

When selecting the right tools, organizations must consider trade-offs between comprehensiveness of security coverage, accuracy of vulnerability detection, and impact on development and operational workflows. Integrating these tools into the SDLC plays an important role in improving application security, enabling organizations to develop and deploy secure software in an efficient and effective manner [15].

4. Benchmarking results

A comparative analysis of application security tools, including static application security testing (SAST), dynamic application security testing (DAST), and runtime application self-protection technologies (RASP), provides insightful conclusions about their effectiveness, utility, and applicability for improving software application security [16].

SAST tools, despite their invaluable assistance in early vulnerability detection, exhibit certain limitations. Analysis has shown that SAST tools, while helpful in identifying common vulnerabilities, often generate a significant number of false positives. This problem can lead to development delays as teams spend time validating and troubleshooting these reports. Additionally, because SAST is source code-centric, it can miss runtime vulnerabilities or environmental configuration issues that can only be detected while the application is running.

SonarQube and **Fortify Static Code Analyzer** were analyzed. The results showed that both tools offer comprehensive detection mechanisms, but differ in ease of integration and impact on the development cycle. SonarQube was noted for its developer-friendly interface and lower false positive rate compared to Fortify, which, while more thorough, requires more time to reduce false positives.

DAST tools provide an outsider's view of running applications, identifying vulnerabilities that can be exploited in a deployed application. The analysis showed that while DAST tools are effective at modeling external attacks and identifying problems at runtime, they are limited in what they can do. Because DAST tools work from outside the application, they may not have a complete view of the application's internal processes, which can lead to missing vulnerabilities that are not exposed through external interfaces.

OWASP ZAP and **Burp Suite** were noted for their effectiveness in simulating real-world attacks. OWASP ZAP was recognized for its accessibility and ease of use, making it suitable for integration into the development process. Burp Suite, with its detailed analysis capabilities, was deemed more suitable for security professionals.

RASP tools offer a new approach by integrating defense into the application and providing immediate real-time response to attacks. The analysis showed that RASP tools effectively address vulnerabilities by directly monitoring application behavior and preventing exploit attempts. However, the adoption of RASP technology is hampered by issues related to performance impact and the difficulty of integrating these tools into existing application architectures.

Contrast Security and **Imperva RASP** demonstrated their extensive capabilities to protect applications from known and unknown vulnerabilities. Contrast Security was particularly noted for its developer-friendly approach and minimal impact on application performance. Imperva RASP, while providing robust protection, requires more careful tuning and configuration to achieve optimal performance.

5. Discussion

The comparative analysis suggests a layered approach to application security that leverages the strengths of the SAST, DAST, and RASP tools in addition to each other. Organizations are encouraged to implement SAST tools early in the development lifecycle for early vulnerability detection, use DAST tools for comprehensive external testing of deployed applications, and apply RASP solutions to protect against real-time attacks. This strategy enables a holistic security posture by addressing vulnerabilities at different stages of the application lifecycle.

Conclusion

In conclusion, this comprehensive evaluation of application security tools underscores the critical importance of integrating a multi-faceted approach to safeguard software applications in the digital age. Through the comparative analysis of static analysis techniques, dynamic analysis tools, and runtime protection mechanisms, the study illuminates the strengths and limitations inherent to each method. Static Application Security Testing (SAST) tools are indispensable for early detection of vulnerabilities, yet they are not without their drawbacks, such as the propensity to generate false positives and the inability to detect runtime issues. Dynamic Application Security Testing (DAST) tools offer valuable insights into runtime vulnerabilities, albeit with limitations in accessing the application's internal logic. Runtime Application Self-Protection (RASP) technologies emerge as a promising solution for real-time threat mitigation, though challenges in integration and performance impact merit consideration.

The research advocates for a layered security strategy, leveraging the complementary capabilities of SAST, DAST, and RASP tools to establish a robust defense against the spectrum of cyber threats. This holistic approach not only enhances the security posture of applications but also aligns with the evolving landscape of cyber risks and the growing sophistication of cyber attackers.

Furthermore, the paper contributes to the field of cybersecurity by proposing a set of criteria for evaluating application security tools, aimed at guiding stakeholders in making informed decisions about their adoption and integration into the software development lifecycle (SDLC). By doing so, it provides a roadmap for developers, security professionals, and organizations to navigate the complex terrain of application security, ensuring the confidentiality, integrity, and availability of application resources and data.

References

1. Cyber Security Market Size, Share & Trends Analysis Report By Component, By Security Type, By Solution, By Services, By Deployment, By

Organization Size, By Verticals, By Region And Segment Forecasts, 2024 - 2030. URL: https://www.grandviewresearch.com/industry-analysis/cyber-security-market

2. OWASP Top 10: The Most Critical Web Application Security Risks. 2023. URL: https://securityboulevard.com/2023/03/owasp-top-10-the-most-critical-web-application-security-risks/

3. ISO/IEC 27001:2013. "Information technology — Security techniques — Information security management systems — Requirements." URL: https://www.iso.org/obp/ui/#iso:std:iso-iec:27001:ed-2:v1:en

4. Cybersecurity C. I. Framework for improving critical infrastructure cybersecurity //URL: https://nvlpubs. nist. gov/nistpubs/CSWP/NIST. CSWP. – 2018. – Vol. 4162018. – pp. 7.

5. Horvath M. et al. Magic Quadrant for Application Security Testing. – 2023.

6. Suhaemi, H., Sunarto, A., Murtiningsih, D., Napitupulu, D., & Rahim, R. (2019, November). Implementation of Square Methods in Analyzing the Security of an Application System. In *ICASI 2019: Proceedings of The 2nd International Conference On Advance And Scientific Innovation, ICASI 2019, 18 July, Banda Aceh, Indonesia*(p. 390). European Alliance for Innovation.

7. Zhang, Y., & Zhang, T. (2022). Research Into the Security Threat of Web Application. *Journal of Web Engineering*, *21*(5), 1707-1726.

8. Anwar Mohammad, M. N., Nazir, M., & Mustafa, K. (2019). A systematic review and analytical evaluation of security requirements engineering approaches. *Arabian Journal for Science and Engineering*, *44*, 8963-8987.

9. Pan, Y. (2019, August). Interactive application security testing. In *2019 International Conference on Smart Grid and Electrical Automation (ICSGEA)* (pp. 558-561). IEEE.

10. Tenev, T., & Tsvetanov, S. (2020, September). Recommendations for enhancing security in microservice environment altered in an intelligent way. In *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)* (pp. 1-6). IEEE.

11. Aloraini, B., Nagappan, M., German, D. M., Hayashi, S., & Higo, Y. (2019). An empirical study of security warnings from static application security testing tools. *Journal of Systems and Software*, *158*, 110427.

12. Oka, D. K., Makila, T., & Kuipers, R. (2019, July). Integrating application security testing tools into ALM tools in the automotive industry. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)* (pp. 42-45). IEEE.

13. Kondraciuk, A., Bartos, A., & Pańczyk, B. (2022). Comparative analysis of the effectiveness of OWASP ZAP, Burp Suite, Nikto and Skipfish in testing the security of web applications. *Journal of Computer Sciences Institute*, *24*, 176-180.

14. Kritikos, K., Papoutsakis, M., Ioannidis, S., & Magoutis, K. (2019, May). Towards configurable cloud application security. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)* (pp. 684-689). IEEE.

15. Kunda, M. A., & Alsmadi, I. (2022, September). Practical web security testing: Evolution of web application modules and open source testing tools. In *2022 International Conference on Intelligent Data Science Technologies and Applications (IDSTA)* (pp. 152-155). IEEE.

16. Seth, A. (2022). *Comparing effectiveness and efficiency of interactive application security testing (iast) and runtime application self-protection (rasp) tools*. North Carolina State University.

17 Olabanji SO. Advancing Cloud Technology Security: Leveraging High-Level Coding Languages like Python and SQL for Strengthening Security Systems and Automating Top Control Processes. J. Sci. Res. Rep. [Internet]. 2023 Sep. 13 [cited 2024 May 29];29(9):42-54. Available from: https://journaljsrr.com/index.php/JSRR/article/view/1783

18 Okolie AC. Microcontroller-based Security System for an Industrial Complex: Design, Fabrication and Testing. Curr. J. Appl. Sci. Technol. [Internet]. 2024 Mar. 30 [cited 2024 May 29];43(4):56-73. Available from: https://journalcjast.com/index.php/CJAST/article/view/4369

19 Macher G, Sporer H, Brenner E, Kreiner C. Supporting cyber-security based on hardware-software interface definition. InSystems, Software and Services Process Improvement: 23rd European Conference, EuroSPI 2016, Graz, Austria, September 14-16, 2016, Proceedings 23 2016 (pp. 148-159). Springer International Publishing.