Journal of Advances in Developmental Research (IJAIDR)
E-ISSN: 0976-4844 ● Website: www.ijaidr.com ● Email: editor@ijaidr.com

# Challenges and Limitations of Dynamic Application Security Testing (DAST) in Modern Software Development Environments: A Systematic Review

**Vivek Somi**

Information Security Engineer

**Abstract**

**DAST or Dynamic Application Security Testing can be considered a critical methodology in the contemporary approach to software security, including those flaws that emerge during application execution. While compared to static testing approaches, DAST delivers application assessments from an external point of view to mimic genuine attack vectors without the need for direct access to the source code. This systematic review aims to understand some of the limitations of DAST ranging from the use in CI/CD pipelines, DAST performance in microservices and cloud-native architectures and the ability of DAST to detect vulnerabilities. Technical constraints including the false positive rates, false negative rates, and missed areas are also explored. Furthermore, this review looks into new trends with solutions such as the synergy of DAST with VST and the incorporation of AI in boosting the DAST effectiveness. Thus, with the help of identifying these challenges and discussing new approaches to them, this work is to give a guideline on enhancing the efficiency of DAST for protecting current software applications. Lastly, the review gives information about newer potential future trends and advancements that may enhance the effectiveness, efficiency, and extensibility of DAST in more complex software systems.**

**Keywords: Application Security, Cloud-Native Architectures, Continuous Integration/Continuous Delivery (CI/CD), Dynamic Application Security Testing (DAST), Microservices, Runtime Vulnerabilities, Security Testing, Software Development Lifecycle (SDLC)**

## Introduction

DAST commonly stands for Dynamic Application Security Testing, and plays an important role in today's software development, especially for the assessment of the external Security weakness in web applications. DAST is different from other types of testing, such as SAST, which deals with analysing the source code of an application, but DAST is focused on the conditions observed at runtime, providing the review of the vulnerabilities which appear at runtime. Compared to other software testing methodologies this testing method uses real attack scenarios to find out security flaws such as SQL injections, cross-site scripting (XSS), and other security vulnerabilities that are invisible in code review, white box, and black box testing.

The evolving use of sophisticated software applications in economically sensitive sectors such as finance, healthcare, and e-commerce all support the need for DAST. Especially with the enlargement of

application complexity and involving microservices, APIs, and cloud-native architectures, the demand for strong staking on security testing tools and frameworks has increased. Since it doesn't demand source code access to work, DAST is most effective in securing third-party and legacy applications. Besides, as CI/CD pipelines become more integrated into DevOps processes, DAST enables security to remain a consideration during the processes of rapid software development.

The purpose of this paper is to provide a comprehensive analysis of both the technical and integration issues that limit DAST in contemporary software settings. Recognizing these deficits, this study aims to identify possible enhancements and evolution regarding DAST to bolster the efficacy and dependable safety it provides for applications against dangerous threats. The current goals of the study project include the definition of the current challenges arising in connection with DAST, analysis of its performance problems and the search for new solutions, including AI-related ones.

## Overview of Dynamic Application Security Testing

### Definition and Key Concepts

DAST is a subcategory of black box testing which mainly aims to discover security flaws in Web applications by emulating attacks that are launched in the course of the application[1]. DAST on the other hand works differently from what SAST (Static Application Security Testing) does, as the latter analyses source code for security vulnerabilities or penetration, DAST indirectly tests applications, by emulating an attacker's attempt to attack the application interfaces[2]. The process of testing in real-time helps identify intrusions and penetrations including, SQL injections, cross-site scripting (XSS) and remote code execution vulnerabilities[3].

DAST, in contrast, makes no assumptions about the functionality of the application and does not need access to the source code meaning that it can be applied to third-party applications, legacy systems and external web services[4]. Like penetration testing, through a variety of real or mimic tests, DAST identifies vulnerabilities and assesses the reactions to determine any problems that could result in data leaks, user rights elevation or other types of security breaches. DAST is good for post-deployment testing because it mimics the functioning of applications in live paradigms, giving testers an insight into the runtime problems that can occur when the application is fully functional[5].

Thus, DAST expands its area of application to APIs and other cloud-based services because they are already attractive to attackers. Nevertheless, the growing intricacy in applications resulting from transformation toward microservices as well as serverless computing has forced DAST to assume new methodologies though still being an integral component of modern security testing models.

### DAST in Modern Software Development Lifecycle

As new methodologies are adopted in development, DAST has evolved from its role concerning the new methodologies of Agile Software Development. Historically, security testing, and thus DAST, was performed much later in the software development cycle, possibly even at the deployment stage or at a late testing stage[6]. However, CI/CD integrated pipelines and the security shift-left movement have relocated DAST earlier in the software development life cycle[2].

DAST are widely integrated in CI/CD environments and are regularly used in security testing sets that allow to find and eliminate the weaknesses in the early stages of SDLC[7]. This shift assists in decreasing costs and increasing the simplicity of eradicating security problems because it is easier to treat them during the development period than during deployment. Gartner research conducted in 2019 pointed out that 85 percent of the companies, that had integrated the DAST tools into CI/CD pipelines, had a positive impact on enhancing the security standings of the organizations and decreasing production vulnerabilities by 35 percent on average[8].


**Figure 1: Hype Cycle for Application Security**

**Source: Adapted from [8]**

DAST tools were also adapted for the modern application architecture, including APIs, microservices and containerization. Notably, the new applications equestrian on serverless frameworks present new challenges to DAST, which has always assumed stable and divider environments in which to conduct tests. Nevertheless, modern DAST solutions are evolving to accommodate their versatility as applications are becoming distributed today and the control should be early, even where it is in a somewhat dynamic and transient manner.

## Methodology of the Systematic Review

### Search Strategy and Selection Criteria

The systematic review utilized a focused search strategy on the data sources in different academic, industrial, and conference papers up to June 2019. This review examined issues and future trends in DAST strategies when applied to contemporary SDLCs, particularly CI/CD systems, microservices architecture, and runtime risks.

Keywords used during the search included "Dynamic Application Security Testing," "DAST implementation in CI/CD," "microservices security testing," and "DAST limitations in cloud-native applications." The search was performed in IEEE Xplore, ACM Digital Library, Google Scholar, and industry reports to find all relevant papers and literature. Only papers which were related to DAST implementation, technical issues related to it, performance issues associated with it and probable solutions were chosen.

### Data Extraction and Analysis

Information from the chosen sources was collected and categorized according to the frequently encountered issues that manifest in DAST, including technical problems of the tool, false positive/negative conclusions and such factors as complex structure of software. In the course of the analysis, all of the provided data sources were scrutinized to determine the gaps and trends for the modern software environments as well as new emerging solutions. Qualitative data comprised in the study with the performance of DAST in actual application settings are presented in numerical figures to give a quantitative dimension on the effectiveness of the instrument as a tool to detect vulnerabilities in different architectures such as the DAST success rates.

This data was then used to identify conclusions about the current state of DAST Usage and bring forward recommendations on future trends of research and development of DAST tools. This approach also helpsdiscover some areas of performance bottlenecks and integration problems that developers and security teams encounter when adopting DAST in an environment of Agile software development.

### Current Challenges in DAST Implementation

### Technical Limitations

DAST is an inherently black-box testing technique, stating that it only goes through an application's interface and cannot see the source code of the application. It tends to overlook some risks that dynamic analysis tools which analyse the working program can determine from the source code of the program[9]. A critical disadvantage of DAST is that it is unable to detect vulnerabilities not visible in the course of runtime; thus, it cannot detect susceptibilities embedded in the application. Veracode concluded that DAST tools in tested applications find only 66% of total vulnerabilities while static testing finds 86%[10].

In addition, DAST is challenged when it comes to applications with complex microservices and API architectures, though not all of them can be easily accessed through the UI[11]. This is much as leads to half-baked vulnerability scans and missing security flaws, especially in applications where a lot of logic is coded on the back end.

### Integration Issues with CI/CD Pipelines

A major challenge firms have encountered while using DAST in their automation is the fact that integrating CI/CD pipelines is still an arduous process that demands a lot of scanning time and resources. The DevSecOps community reported in 2019 that the challenges companies face in integrating DAST stemmed from it increasing the difficulty of developing which ultimately slowed down the development pipeline – 45% of development teams had issues with integrating DAST into CI/CD[12].

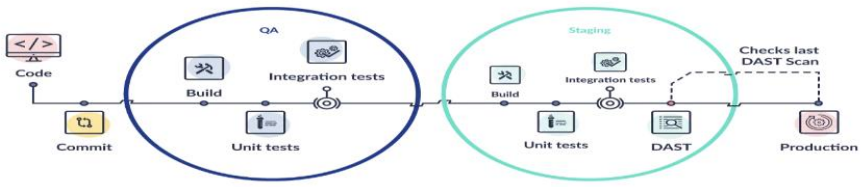


**Figure 2: automate DAST in CD**

**Source: Adapted from [12]**

DAST tools are created to test live applications and so, they may need fully deployed environments to work on and this is hard to automate in CI/CD models whereby continuous deployment is the norm[13]. This requirement has often resulted in bottlenecks within the testing phase and it slows development whenever extensive security scans are necessary.

**False Positive/Negative Rates**

Externally, false positives and false negatives are some of the greatest issues in DAST solutions. Since DAST tools report potential vulnerabilities which do not exist, these are considered false positives to developers because of the time taken to analyse, debug and fix the absence of vulnerabilities. On the other hand, false negatives that relate to situations where real vulnerabilities may not be detected during the screening of applications mean that applications may be vulnerable to attacks even after going through the security test.

While using DAST, the actual percentages were a 19% false positive average, and 13% false negative average thus making it complex for security teams to depend solely on DAST testing[14]. These false reports result in distrust of the tool and the need to use other testing methods such as SAST and IAST to supplement to offer a comprehensive security analysis.

**Coverage and Depth of Testing**

This testing type also has limitations in the depth it offers: the depth that DAST provides. As the name suggests, DAST stands for Dynamic Application Security Testing; the tool can be used to check the application for vulnerabilities when a user interacts with them through the user interface or a Public API, but it does not go deeper into the application[15, 16]. Such as Internal APIs, access to databases, and logical flaws are not very easy to be recognized by DAST.

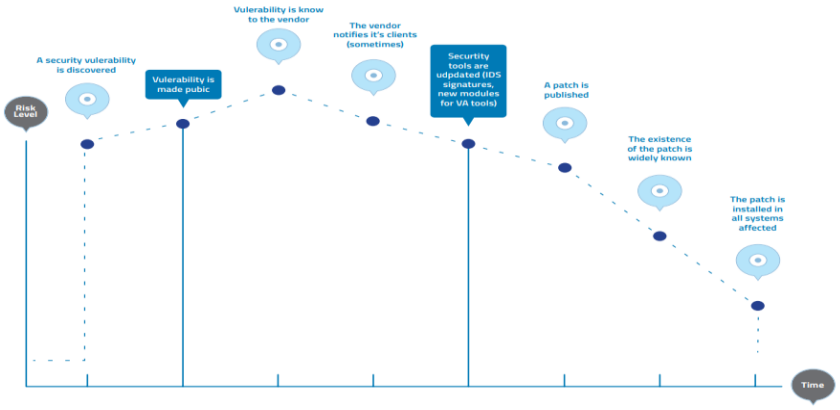


**Figure 3: Window of Vulnerability**

**Source: Adapted from [16]**

This limitation starts becoming even more apparent in many modern application architectures where business logic is split across multiple services, most of which are not directly bound to external access points. A 2018 study looking at application security revealed that 54% of the firms responding indicated that while using DAST tools, they realized that the tools offered insufficient coverage when it comes to testing layered applications, and therefore had an incomplete picture of potential vulnerabilities[17].

**DAST in Complex Application Architectures**

**Microservices and Containerized Applications**

An application that is decomposed into numerous smaller independent services in what is known as microservices architectures proves to be a weakness in DAST. While traditional applications, on the other hand, are monolithic, microservices are designed to work in a distributed architecture which can run in different containers or virtual machines respectively[18]. Most of the services can be different and have their security necessities and risks that are why DAST tools cannot check the totality of the service.

Containerization makes DAST assessment more challenging as well because the containers are commonly short-lived and can be created/terminated rapidly. Because container environments tend to be more time-bound than their VM counterparts, DAST tools may not be able to perform uniform, reliable scans, unless they're designed to factor in container orchestration frameworks such as Kubernetes or Docker Swarm[19].

**Serverless and Cloud-Native Environments**

Serverless systems are another problem for DAST. In the serverless model, the computational code is run in stateless functions for short durations as noted by constraint, which makes it hard for DAST tools to test the actual runtime environments[20]. Since serverless applications are short-lived, running for only several milliseconds at a time, they pose challenges in testing, especially for DAST, which requires always running environments to perform scans.

Furthermore, applications which are developed to be deployed in cloud-like architectures, infrastructures or environments that have scale-out characteristics present extra challenges. Such applications frequently use auto-scaling and load balancing combined with dynamic service discovery, all of which may impact

DAST testing by influencing consistency and accuracy[21]. Also, selecting an appropriate security approach that comprehensively tests serverless applications and cloud-native environments has emerged as a significant challenge[22].

**API-driven Applications**

Application programming interface-driven development has become the new normal for web and mobile applications. Applications have interfaces, often called APIs, which enable one service to be integrated with another service, one platform with another, or one device with another, but on the same token APIs are vulnerabilities. The problem that DAST tools face is that many of the above interactions between APIs and their client do not get the users' participation at all[23].

The fact that APIs are interconnected and the interactions under test are complex, especially in systems that integrate multiple third-party APIs makes the testing even more challenging. DAST tools have to be tuned to recognize API-specific issues like rate limiting, authentication, and data validation but the truth is that at present most of the DAST tools are not so efficient in managing these issues adequately[24, 16]. Research by OWASP in 2018 revealed that only 40% of the DAST tools met the testing requirements of the new generation of API-based applications[16,24, 25].

**Performance and Scalability Issues**

**Time and Resource Consumption**

One of the most striking weaknesses of DAST is the great effort in terms of time and resources needed to perform a broad test. Applications having a large number of endpoints take many hours to days to complete scanning, and thus, cause a problem for development teams who want to stick to the application deployment timeline. A 2018 report by Gartner pointed out that DAST scans led to an increase of time to deployment by 18% in organizations, thus affecting the CI/CD.

However, tests using DAST tools are computationally intensive, especially when testing large applications. In cloud platforms whereby resources are elastic, this often causes interference with the reliability of tests, as well as the time taken to scan for vulnerabilities.

**Impact on Development Velocity**

The slowness of DAST tools has a direct effect on the speed of delivery teams to get their tasks done. In applications that release features every few hours, incorporating security testing means that development will be slowed down by the extra time it takes.Other significant security holes within the applications remain unseen and unaddressed until someone finds a way to exploit them in realistic scenarios. Currently, DAST vendors are actively addressing the question of tool speed and, while the problem is largely relevant, its impact on development speed will be partially felt[26].

**Emerging Solutions and Future Directions**

**AI and Machine Learning in DAST**

AI and ML are evolving quickly as the newest and fastest-growing tools in security testing in DAST. DAST tools can also be enhanced using these technologies as a way of increasing the reliability of the tools by decreasing the number of false positives and false negatives that can be identified through

pattern recognition. The nature of the scan results can also be learned by the AI tools and from there, make improvements to their algorithms to distinguish true threats and exclude the false ones such as the dummy problems which may have been detected on previous scans[27].That is why it is also possible to use ML algorithms for ranking the uncovered vulnerabilities by the severity of the impacts on the system. A 2019 report by Capgemini stated that advanced DAST tools with AI capabilities were effective at saving about a third of the time that was taken to discover the risks associated with past approaches[28].

**Hybrid Approaches (Combining DAST with SAST and IAST)**

Despite the tested method's constraint, there is an increasing trend of integrating methods such as DAST with SAST and IAST. SAST checks the source code in detail, on the other hand, DAST tries on the application in its runtime environment and at the same time, IAST gives feedback during its execution[29]. This compilation guarantees inclusiveness and the ability to identify purely logical, as well as runtime, vulnerabilities.

The organizations can observe that the deployment of hybrid security models will have great impacts on improving the situations. A study by Synopsys conducted in 2018 showed that hybridusers had a 50% lower number of missed vulnerabilities compared to DAST users[30].

**Automated Remediation Techniques**

Automated remediation is a relatively new approach that is on its way to changing the way that vulnerabilities are remediated in DAST. Traditional DAST, on the other hand, narrows the scope of analysis to the application's security issues and is only gradually moving towards the direction of automatically eliminating vulnerabilities without outside assistance[31]. These tools apply machine learning to come up with the patches or configurations that can help negate the risks that have been identified, sparing the development teams the time they would have spent fixing the problems on their own, time they could spend on creating new features instead.

**Conclusion**

Manual and automated approaches to Dynamic Application Security Testing (DAST) are still noteworthy, providing valuable information on issues that appear during an application's execution. However, DAST has problems when applied to the CI/CD integration, dealing with complex architectures, and minimize false-positive and false-negative reports. Nevertheless, the work of DAST goes on; there are new ideas in the application of artificial intelligence and mixed approaches as a part of the way of overcoming the weaknesses of DAST.

Therefore, the prospect of DAST depends on the dynamics in software development as cloud-native, serverless, API-first applications dominate software development. Through leveraging newer technologies such as AI and machine learning, for instance, DAST tools can become more effective and precise ensuring the developer gets the security information they need within the shortest time possible all while coding. With organizations focusing on security as the key consideration in development, DAST will continue to offer its value in securing the applications of today and the future.

**Bibliography**

[1] J. Im, J. Yoon, and M.-S. Jin, "Interaction Platform for Improving Detection Capability of Dynamic Application Security Testing," *International Conference on Security and Cryptography*, Jan. 2017, doi: https://doi.org/10.5220/0006437104740479.

[2] NowSecure, "5 Mobile DAST Misconceptions | Dynamic Application Security Testing," NowSecure Blog, Sep. 25, 2019. [Online]. Available: https://www.nowsecure.com/blog/2019/09/25/5-misconceptions-about-dynamic-application-security-testing-dast-for-mobile/.

[3] Rapid7, "Application Security Testing & Monitoring with DAST and RASP: A Two-Pronged Approach," Rapid7 Blog, Oct. 28, 2019. [Online].
Available: https://www.rapid7.com/blog/post/2019/10/28/application-security-testing-monitoring-with-dast-and-rasp-a-two-pronged-approach/.

[4] Internal Admin, "How to Improve DevOps Quality Assurance | ThreatModeler," *ThreatModeler*, Oct. 28, 2017. https://www.threatmodeler.com/improve-devops-quality-assurance/.

[5] Abdollah Shajadi, "Automating Security Tests for Web Applications in Continuous Integration and Deployment Environment," Jan. 2019.

[6] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich, "FeatureIDE: An extensible framework for feature-oriented software development," *Science of Computer Programming*, vol. 79, pp. 70–85, Jan. 2014, doi: https://doi.org/10.1016/j.scico.2012.06.002.

[7] T. H.-C. Hsu, "Practical Security Automation and Testing," *Google Books*, 2019. https://books.google.com/books?hl=en&lr=&id=z4KGDwAAQBAJ&oi=fnd&pg=PP1&dq=DAST+are+widely+integrated+in+CI/CD+environments+and+are+regularly+used+in+security+testing+sets+that+allow+to+find+and+eliminate+the+weaknesses+in+the+early+stages+of+SDLC.+&ots=otPNR-CgSW&sig=lI9tMhcLANcczYrIo0HudLwbFM0.

[8] Dmitry Sotnikov, "Issue 68: API security in Gartner Hype Cycle, McAfee threat predictions for 2020," *API Security News*, Jan. 2020. https://apisecurity.io/issue-68-api-security-in-gartner-hype-cycle-threat-predictions-for-2020/

[9] Satish Govindappa, "What is Dynamic App-sec Vulnerability Scanning (DAST)? Vulnerability scanning is first phase of penetration testing. Vuln-scanning can be manual or dynamic.," *Linkedin.com*, Jul. 20, 2018. https://www.linkedin.com/pulse/automation-app-sec-vulnerability-scanningdast-cicd-satish-govindappa.

[10] B. Higuera and J. Ramón, "Assessment methodology of web applications automatic security analysis tools for adaptation in the development life cycle," Jun. 2014.

[11] M. Rao, "Common security challenges in CI/CD workflows," *Blackduck.com*, Jul. 10, 2018. https://www.blackduck.com/blog/security-challenges-cicd-workflows.html.

[12] N. Loureiro and Davor Petreski, "Integrating Web Vulnerability Scanners in Continuous Integration: DAST for CI/CD," *https://probely.com*, Jul. 2019. https://probely.com/blog/integrating-web-vulnerability-scanners-in-continuous-integration-dast-for-ci-cd/.

[13] M. Press, B. Briggs, and E. Kassner, "Enterprise Cloud Strategy 2nd Edition," 2017. Available: https://info.microsoft.com/rs/157-GQE-382/images/EN-US-CNTNT-ebook-Enterprise_Cloud_Strategy_2nd_Edition_AzureInfrastructure.pdf

[14] B. Aloraini, M. Nagappan, D. M. German, S. Hayashi, and Y. Higo, "An empirical study of security warnings from static application security testing tools," *Journal of Systems and Software*, vol. 158, p. 110427, Dec. 2019, doi: https://doi.org/10.1016/j.jss.2019.110427.

[15] A. Oprea, Z. Li, R. Norris, and K. Bowers, "MADE," *Proceedings of the 34th Annual Computer Security Applications Conference*, Dec. 2018, doi: https://doi.org/10.1145/3274694.3274710.

[16] OWASP, "4.0 Testing Guide," 2014. Available: https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Testing_Guide_v4.pdf

[17] C. Chapman, *Network Performance and Security*. Syngress, 2016.

[18] A. W. S. Startups, "Using Containers to Build a Microservices Architecture," *Medium*, Apr. 23, 2015. https://medium.com/aws-activate-startup-blog/using-containers-to-build-a-microservices-architecture-6e1b8bacb7d1

[19] L. Zheng and B. Wei, "Application of microservice architecture in cloud environment project development," *MATEC Web of Conferences*, vol. 189, p. 03023, 2018, doi: https://doi.org/10.1051/matecconf/201818903023.

[20] N. Kratzke, "A Brief History of Cloud Application Architectures," *Applied Sciences*, vol. 8, no. 8, p. 1368, Aug. 2018, doi: https://doi.org/10.3390/app8081368.

[21] S. Taherizadeh and V. Stankovski, "Dynamic Multi-level Auto-scaling Rules for Containerized Applications," *The Computer Journal*, vol. 62, no. 2, pp. 174–197, May 2018, doi: https://doi.org/10.1093/comjnl/bxy043.

[22] Apriorit, "Security of Serverless Applications: Key Challenges and Best Practices to Overcome Them," Apriorit Dev Blog, 2020. [Online]. Available: https://www.apriorit.com/dev-blog/657-web-security-of-serverless-applications

[23] R. Degges, "Please Stop Using Local Storage," *The DEV Community*, 2018. https://dev.to/rdegges/please-stop-using-local-storage-1i04

[24] OWASP, "OWASP Application Security Verification Standard 4.0," *Owasp.org*, 2019. https://wiki.owasp.org/images/8/88/OWASP_Application_Security_Verification_Standard_4.0-en.docx.

[25] O. Hämäläinen, "API-First Design with Modern Tools," 2019.[Online]. Available: https://www.theseus.fi/bitstream/handle/10024/226493/Hamalainen_Oona.pdf?sequence=2

[26] D. Rigby, J. Sutherland, and A. Noble, "Agile at Scale," *Harvard Business Review*, May 2018. https://hbr.org/2018/05/agile-at-scale

[27] M. Chora and R. Kozik, "Machine learning techniques applied to detect cyber attacks on web applications," *Logic Journal of IGPL*, vol. 23, no. 1, pp. 45–56, Dec. 2014, doi: https://doi.org/10.1093/jigpal/jzu038.

[28] Capgemini, "Reinventing Cybersecurity with Artificial Intelligence," 2019. Available: https://www.capgemini.com/wp-content/uploads/2019/07/AI-in-Cybersecurity_Report_20190711_V06.pdf

[29] Rapid7, "App-a-bet Soup: Should You Use a SAST, DAST, or RASP Application Security Tool?," Rapid7 Blog, Aug. 6, 2019. [Online]. Available: https://www.rapid7.com/blog/post/2019/08/06/app-a-bet-soup-should-you-use-a-sast-dast-or-rasp-application-security-tool/.

[30] Synopsys, "DevSecOps Realities and Opportunities," 2018. Available: https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/devsecops-realities-opportunities-451.pdf

[31] D. Danielecki and T. Beekman, "Security First approach in development of Single-Page Application based on Angular," 2019. [Online]. Available: https://essay.utwente.nl/79862/1/Danielecki_MA_EEMCS.pdf