Application Security Verification Standard Compliance Analysis of a Low Code Development Platform

Master of Science in Technology
Thesis
University of Turku
Department of Computing
Cybersecurity
2022
Sami Spets

Supervisors:
Tapani Joelsson Univ. of Turku
Martti Ala-Rantala Solita Oy
Seppo Virtanen Univ. of Turku

UNIVERSITY OF TURKU
Department of Computing

SAMI SPETS: Application Security Verification Standard Compliance Analysis of a Low Code Development Platform

Master of Science in Technology Thesis, 57 p.
Cybersecurity
December 2022

Low-code development platforms (LCDPs) are software development platforms that use artificial intelligence to help automate simple and routine tasks and make the software development process faster. By 2024, 60% of application development expect to be done using these platforms. Even though these platforms are gaining popularity, they have not been popular research topics, and their security features have not been assessed. One way to conduct such an assessment is by using Application Security Verification Standard (ASVS).

ASVS is a community-driven security standard for web applications and services. ASVS is made of three requirement levels, and the security controls become more strict when moved up. ASVS is designed to give organizations a tool to develop and maintain more secure applications.

One example of an LCDP is OutSystems, which is said to be "designed for the developers, by the developers". OutSystems belongs to the Leader category in the 2021 release of Gartner® Magic Quadrant™ for Enterprise Low-Code Application Platforms. In this thesis, we will conduct a first of its kind compliance analysis between OutSystems and ASVS levels 1 and 2 to find out if and how compliant OutSystems is with the standard. This kind of compliance analysis has not been done before. Based on our analysis, we will do a "lessons learned" and write a guideline on how to evaluate LCDPs' security features in the future.

The results themselves show that OutSystems, for the most part, is compliant with ASVS. The biggest deficiencies in OutSystems are with authentication and input validation. We show that the deficiencies with authentication are trivial to fix, but meeting the requirements with the input validation requires some work.

From the assessment, we learned that assessing LCDPs is not completely similar to a traditional security assessment. We learned that some functionalities are pre-made, and the developer can not customise them. We found that it is easier to evaluate first if the platform meets the requirement. If not, then see if the developer can do something about it.

Keywords: OutSystems, OWASP ASVS, low code, security standard, cyber security

Contents

List of Figures

List of Tables

List of acronyms

AI Artificial Intelligence

ASVS Application Security Verification Standard

LCDP Low-code development platform

LCSD Low-code software development

MFA Multi Factor Authentication.

OTP One Time Password

OWASP Open Web Application Security Project

U2F Universal 2nd Factor

1 Introduction

Software development methodologies have existed for over 50 years, beginning with the Systems Development Life Cycle (SDLC) [1]. It consists of five elements: planning, analyzing, designing, implementing, and maintaining the system. The elements themselves have not much changed, but the way they are implemented have.

In the 1990s, Rapid Application Development (RAD) emphasized building prototypes and proofs-of-concept rather than designing or planning. The user could interact with the prototypes and give feedback, leading to better quality products and staying within budgets since the focus is on implementation [2]. The process resembled that of a modern-day Agile methodology. For example, in Scrum, the development is typically done in two weeks "sprints". These sprints consist of choosing a feature to be developed, developing it, and then demonstrating the developed feature to the customer. These features are given to the customer in small pieces, and the customer can ask for improvements based on their needs [3]. Emphasis was on providing working software fast and incrementally enhancing it rather than delivering software all at once.

1.1 Research motivation

Coming to the 2010s, the rapid software delivery has become a more crucial part of the software development cycle. One of the biggest bottlenecks in the delivery process is converting the design with many details to an actual working application.

For some time dissolving this bottleneck has had two ways, first is by increasing the number of skilled software developers working on the software. The problem with this is that the industry has a severe lack of skilled developers but the need for software keeps increasing. The second way is to use different types of software accelerators to try and make the development process faster. Typically these accelerators create some general boilerplate code and give some kind of template for the developer to work with.

Low-code development platforms (LCDPs) approach this bottleneck from a new perspective. They utilize Artificial Intelligence (AI) and static code analysis based tools. AI-based tools help the developer by creating boilerplate code based on the context and handling repetitive tasks. The static code analysis tools are used to analyze the code, spot bad practices, and warn the developer of potentially harmful code.

Working with LCDPs requires little to no programming and may utilize a new class of developers, called citizen developers. These are people who do not have a background in IT and do not have existing knowledge or skills in software engineering, but are subject matter experts and know the business domain the software is built for. Their status is somewhat debatable, and LCDPs differ starkly in the degree to which they support citizen developers. How can someone do even little programming if they have never before programmed and if they do not understand the basics of programming? Citizen developers use the same tools, frameworks, and libraries that require lots of skills from an expert developers. Is there a point in showing a programming error on the screen if the citizen developer does not know what that error means? If an LCDP would take all of this into account, would that limit the capabilities of expert developers who have a background in IT?

With LCDPs, complete systems and applications are created using different accelerators to add functionalities and pre-fabricated components. LCDPs, such as

OutSystems, have existed for 20 years, but they have become more popular and widely used in software development in recent years. Estimates have it that by 2024, over 65 % of all application development uses LCDPs. [4]

Because estimates predict that LCDPs' popularity will skyrocket, the security of these platforms is becoming a more relevant topic. New vulnerabilities are found in different software products constantly. Cybersecurity standards help mitigate them. One such standard is Open Web Application Security Project's (OWASP) Application Security Verification Standard (ASVS). OWASP is a community-driven non-profit foundation that aims to create more secure software [5]. They have open-source projects such as a top 10 list of most common vulnerabilities found in web applications [6], a web application security scanner [7], and a security standard to develop secure software [8].

1.2 Research questions

In this thesis, we will do a compliance analysis between OutSystems and ASVS and based on our findings share our thoughts how such analysis could be done in the future. This research question came as a commission from Solita Oy.[1] We did some initial background investigation to see if there were any studies or frameworks on how to conduct a security assessment on an LCDP. We found that the research field was scarce and somewhat scattered on the topic.

To conduct a compliance analysis it is necessary to do an LCDP literature review from a security point of view. We also decided to look into industrial studies to see if there were any industry standards or frameworks to help us with our question. In the end, we came up with the following three research questions (RQs).

[1]Solita Oy https://www.solita.fi/en/

RQ1: WHAT IS THE CURRENT STATE OF THE FIELD OF LCPD?

In our initial background investigation, we did not find many previous research papers about the topic. From those we did manage to find, we did not find any comprehensive literature review from an academic or industrial point of view. Thus, we decided to make two-part multi-vocal literature review.

We will be looking at previous academic studies and categorizing them based on the research papers' topic. We will also review if any of the categories present security in low-code. Based on these research papers we will also present our thoughts on LCDPs' future.

Garousi et al. [9] argued that industrial materials give an insight to the current state of the topic in literature review. Based on the initial familiarization with the field, it seems that LCDP research is driven by industry, with companies such as OutSystems. Thus it is vital to look at industry materials, such as blog posts. Due to the focus of this thesis, we will be focusing on security-related industry materials.

After we have looked at academic studies and industrial materials, we want to find a way to assess OutSystems' compliance with ASVS. We will use the literature review to look if there are any previous case studies and if there are, how they have been conducted, to help conduct our assessment.

RQ2: IS OUTSYSTEMS COMPLIANT WITH ASVS?

After defining a way to assess OutSystems' security, we want to determine how compliant OutSystems is with ASVS. If the platform is not fully compliant, we want to identify the missing components required to become fully compliant. ASVS comprises three levels, and the security requirements become more strict when we move up the levels. We will be assessing OutSystems compliance with levels 1 and 2, and it is worth noting that some of the level 2 requirements are related to developer actions and are not solvable with the platform itself. Level 3 will not be assessed,

because it did not belong to the scope given by Solita.

Since OutSystems provides pre-made logic, which the developers use in the software development phase, we need to find out what is on OutSystems' responsibility and what is on the developer's responsibility.

RQ3: HOW TO ASSESS LCDP SECURITY IN FUTURE?

In low-code software development, the LCDPs have pre-made modules and functionalities. Since some LCDPs are proprietary software, developers cannot always access the source code and cannot assess them as they would with traditional software development. Based on the RQ1 and RQ2, we will define a guideline to evaluate OutSystems' security features.

1.3 Research method and process

This thesis uses multiple different research methods. Chapter 3 will be done as a multi-vocal literature review from an academic and industrial point of view. Chapter 4 is a mixture of case studies and expert reviews. To conduct the expert review, we will use OutSystems' public information about their platform. We will also use the help of two senior experts who are OutSystems professionals, one working at OutSystems and one working at Solita.

1.3.1 Literature review process

With the rise of LCDP, the amount of studies and research on the subject matter is also rising. Before beginning the literature review, we did some preliminary familiarization with the field and even though there are some studies on the subject, there are almost no literature reviews. In our premilinary familiarization we found one paper from 2020 [10] that had one subchapter about previous studies and con-

cluded that LCDP as a research topic seems to be under-researched, but has gained some popularity during the last few years. Then we found one full literature review from the year 2021 [11] which claims to be the first one on the subject. The study focused on assessing the socio-technical aspect of previous LCDP studies and found that most of them focused on the technical system.

We wanted to do a literature review on low-code and security. We found that there is a small number of papers about low-code in general, so we decided to do a literature review on a larger field-wide topic. We chose the terms ”low-code” and ”low code” as our search engine keywords. Other criteria we had for our searches were as follows:

- Research paper is in English,
- Only scientific articles,
- The paper is related to low-code development platforms,
- The paper is published between the years 2000 and 2021

For this literature review, we went through five databases between October 4th and October 14th 2021, and the process is presented in Figure 1.1.

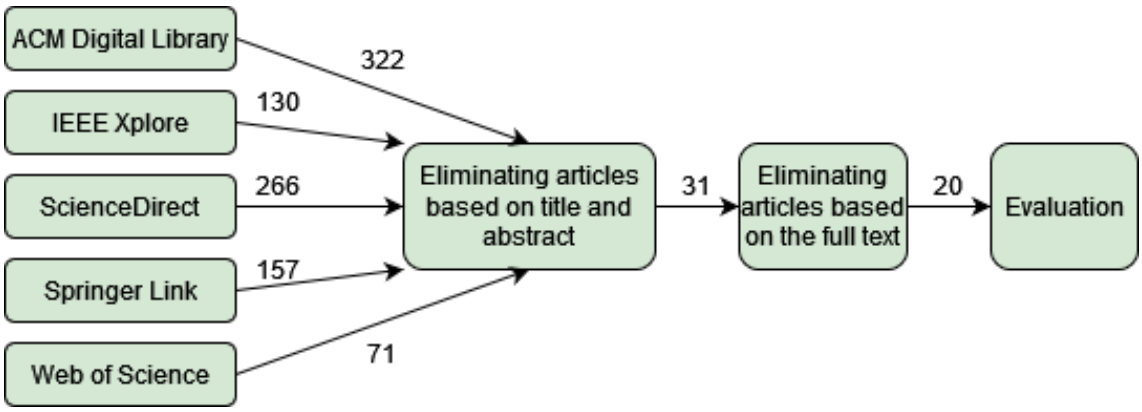


Figure 1.1: Academic research process.

The included databases were ACM Digital Library[2], IEEE Xplore[3], ScienceDirect[4], Springer Link[5], and Web of Science[6]. We found a total of 946 articles that met the initial criterias. The first phase of the process was going through them and based on the search results' title, and abstract deciding whether or not the article was relevant. Parts of the article were also examined if the title and abstract were not clear enough. After this phase we were left with total of 31 articles. They were read in full and based on the full article we decided if they met the inclusion or exclusion criterias. After these elimination phases, there was a total of 20 papers left to be evaluated.

In addition to the fact that we went through five different search engines, we also searched for industrial papers and blog posts from Google and DuckDuckGo between December 6th and December 7th 2021 with the keywords "low-code security" and "low code security". We thought that using "low-code" as a keyword would produce too many search results with general search engines so we wanted to narrow the focus to industry papers on security. In the end, we found four industrial papers from a practical point of view that we think will help us predict the direction of low-code software development.

1.3.2 Compliance analysis

The goal of the compliance analysis is to find out OutSystems' compliance with ASVS. We will use OutSystems' documentation to define the shared responsibility between OutSystems and the developer. The assessment itself will be done as an expert review, where we will be also using OutSystems-provided materials and se-

[2]https://dl.acm.org/

[3]https://ieeexplore.ieee.org/Xplore/home.jsp

[4]https://www.sciencedirect.com/

[5]https://link.springer.com/

[6]https://www.webofknowledge.com/

nior experts. These senior experts are OutSystems professionals, who work with OutSystems.

1.3.3 Defining an LCDP assessment guideline

Based on compliance analysis we identify methods to assess LCDP's security features in the future. We will look at findings from the previous studies and the compliance analysis to help us determine this guideline. It will hopefully help other researchers and practicioners assess LCDPs' security features in the future.

1.4 Structure of the thesis

The structure of this thesis is as follows. Chapter 2 explains the background needed to understand this thesis. We will be opening the topics of cybersecurity standards, LCDPs, and what kind of cybersecurity assessments there are.

Chapter 3 is a literature review of previous academic studies about low-code. We will also review industrial materials, such as blog-post, projects, and papers related to low-code security, and then predict where it is heading. After we have reviewed the studies and industrial materials, we will be assessing where the field of LCDP is heading towards.

In Chapter 4 we will look at how we can apply ASVS to OutSystems and what kind of shared responsibility needs to be defined. We will also present the results of OutSystems' compliance with ASVS and interpret the results. Finally, we will take a closer look at the requirements that did not pass the evaluation. In chapter 5, we will conclude this thesis by recapping and answering the research questions.

2 Background

The two main components for this thesis are ASVS and OutSystems. We will look into what a cybersecurity standard is and what types of cybersecurity standards there are. We will also look at what an LCDP is, and look at two LCDPs, Microsoft Power Platform and OutSystems. Lastly, we will look at what types of cybersecurity assessments there are available and how they can be used.

2.1 Cybersecurity standards

A cybersecurity standard is a set of instructions and techniques that helps an organization or an individual create more secure systems and mitigate cybersecurity threats. The standards themselves can vary much, some can be specific and only address application security. Other standards can address information security management in an organization as a whole. Standards are designed for commercial use, such as the International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 27000 series. However, they can also be community-created, and free to use, such as the ASVS.

2.1.1 ISO/IEC 27000-series

ISO/IEC 27000-series was created in 2005 by ISO and IEC. It is an internationally used set of cybersecurity standards designed for information security management.

The series defines a set of best practices that help organizations improve their information security and tell how to implement these best practices to create an overall information security management system.

The heart of the series is the ISO/IEC 27001 standard, which defines the requirements for an information security management system and how to maintain and improve it [12]. The system is in charge of protecting the confidentiality, integrity, and availability of the organization's information. It also signals that the organization addresses risks with severity. Even though the ISO/IEC 27001 is the heart of the series, the better-known standard is the ISO/IEC 27002 which defines a set of information security controls that the organization can use [13]. The standard also has variations, such as ISO/IEC 27011, which is an implementation designed for telecommunication companies, [14] and the ISO/IEC 27017 is designed for cloud services [15].

Even though ISO develops and maintains the ISO/IEC 27001, they do not grant any certificates. Instead, external certification bodies conduct audits and award certificates. The price for a security audit can range from thousands of dollars to tens of thousands of dollars, depending on factors such as the number of employees in the audited organization and the amount of time the audit takes [16].

2.1.2 OWASP ASVS

ASVS is a community-driven security standard for web applications and web services. It defines the security controls needed when the application or service is designed, developed, and tested, from both functional and non-functional point-of-view. The goal of the standard is to give organizations a tool to develop and maintain more secure applications and allow different security vendors and consumers to have a meeting point for their offers and requirements. In bids, the procurer can use ASVS to define the wanted security features for the application.

Even though ASVS is a standard, there is no actual certification issued to organizations that meet these standards. It can be used as a playbook for application security, but to be used correctly, completely open access is needed to resources such as architects, source code, and test systems. The standard can also be used as a coding checklist, guide for automated unit and integration tests, or detailed security architecture guidance. In this thesis, we will use ASVS version 4.0.3.

ASVS has three security levels designed for different systems and applications. Level 1 is for low assurance applications, level 2 is for today's average applications, and level 3 is for applications and systems running critical infrastructure.

The ASVS defines level 1 as the bare minimum security level that every application should meet. It covers vulnerabilities in the OWASP Top 10 list and other vulnerabilities that are as common and easily discoverable. An attacker who is trying to break into this kind of application most likely uses simple techniques that require little to no effort, such as automated scripts. Application belonging to level 1 does not store or handle any sensitive data. Level 1 applications can be tested manually or automatically without having access to the documentation or the source code.

Applications that achieve level 2 can sufficiently defend against most of the threats and risks that an application might be subjected to today. They handle business-to-business transactions, store or handle sensitive data, or implement business-critical or sensitive functions. In short, level 2 covers most of today's applications. The threats a level 2 application may encounter are target-specific. The attacker is typically motivated, skilled with tools and techniques and knows how to discover and exploit weaknesses in the application. To mitigate these threats and risks, level 2 introduces new controls that should be implemented in processes and phases of software development. These new controls include security architecture and reviews, unit and integration tests, and require the developers to follow

standards and checklists.

In level 2, some of the requirements become more abstract when compared to level 1 requirements. For example, level 1 has requirement id V2.8.1, which states ”Verify that time-based OTPs have a defined lifetime before expiring.”. The requirement following that, requirement id V2.8.2 states ”Verify that symmetric keys used to verify submitted OTPs are highly protected, such as by using a hardware security module or secure operating system based key storage”. Although it gives examples of how to highly protect OTP, it also leaves some room for the developer to determine how to do so.

The highest level an application can achieve in ASVS is level 3. Level 3 applications require significantly more security verification than level 2 applications. These applications usually belong to critical infrastructures, such as health care, banking, or the military. The attackers targetting level 3 applications are usually highly motivated and skilled, with lots of resources to conduct their attacks. Level 3 applications, for the most part, use the same controls as level 2 applications but are implemented with more precision and more depth. For example, it is not enough for a level 3 application to encrypt data, but the application also needs to be sure that the encryption is signed so the encryption is verifiable. Another example would be that in level 2 backups need to be taken periodically, in level 3 the backup restoration also needs to be tested.

2.2 Low-code development platforms

LCDPs are software developing platforms where the development is done with little coding to no coding. The traditional coding has been replaced with an AI-based tool that can generate, for example, boilerplate code or otherwise automate trivial tasks. The developer can drag and drop ready-made modules to build the application. Users can create whole applications and systems this way, and developing with

LCDPs allows faster delivery times [17]. As we will see in the literature review, LCDPs have been steadily increasing in popularity. In this chapter, we will look at two LCDPs, OutSystems and Microsoft's Power Platform.

2.2.1 Microsoft's Power Platform

Microsoft's Power Platform is a cloud-based LCDP. It is a collection of four Microsoft products, Power Automate, Power Apps, Power BI, and Power Virtual Agents. It is a data-driven platform, and the heart of the platform is Microsoft Dataverse, where the data is stored. The data is divided into two categories. Tabular data resembles traditional data and function-based data can also perform functionalities, such as sending an email. The data is moved between Dataverse and applications with Connectors. Triggers and Actions are used to start an event in the application. Actions are user-initiated, and triggers are automated events. AI Builder is used to helping build the application [18][19].

Whereas OutSystems is a low-code platform, in the code writing aspect, Power Platform is a low-code platform in the coding skills aspect. There are over 275 connectors that work with the most common data sources [19]. Citizen developers can use them to develop in-house applications for organizations, automate business processes, and create reports. Expert developers can build their applications traditionally and use Dataverse as their data storage.

Since Power Platform is a data-driven platform, identity management, and access control play a big part in its security features. Dataverse uses Azure Active Directory to authenticate users. A single organization has one Azure Active Directory Tenant, and that tenant can have multiple environments that run the same application but have a small difference between them. These differences can be, for example, that they are regional or that they have different security needs [20]. Power Platform's security features also include data loss prevention policies. These policies can be

scoped to a tenant or environment level [21]. The environments themselves are managed through Power Platform's Admin Center.

2.2.2 OutSystems

OutSystems is an LCDP whose development started in 2001 and that is said to have been "created by developers for developers". It is used for developing mobile and desktop web applications. The applications are hosted either through OutSystems' cloud-hosting service, another public cloud provider, or in some rare instances, locally. Figure 2.1 shows OutSystem's architecture [22][23].

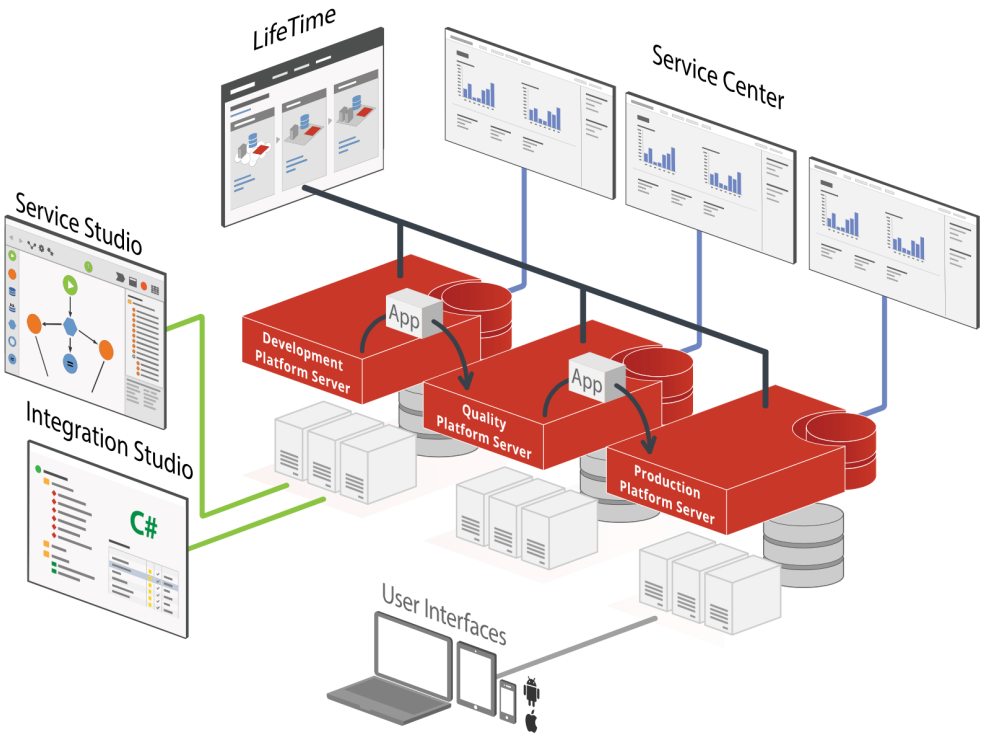


Figure 2.1: OutSystems architecture structure [24]. ® 2022 OutSystems

Service Studio and Integration Studio run on a development environment, although they can be connected to test and production environments as well. After

a feature is ready, it is deployed to a testing environment, where it is subjected to testing and quality assurance is done. After testing and quality assurance, the feature gets promoted to the production environment, where the end-users use it. Each environment has a Service Center instance for managing purposes, and there is one LifeTime instance to manage the infrastructure across all environments. The infrastructure has at least three environments, except for the personal environment where there is only one, and the OS demo environment, where there are two. Typically an infrastructure has four or five environments.

For the developers, the key components are Integration Studio and Service Studio. The application development is mostly done in Service Studio but if a developer wants to write external integration modules with C# or .NET, it can be done with Integration Studio. Figure 2.2 shows the graphical user interface for OutSystems Service Studio 11.

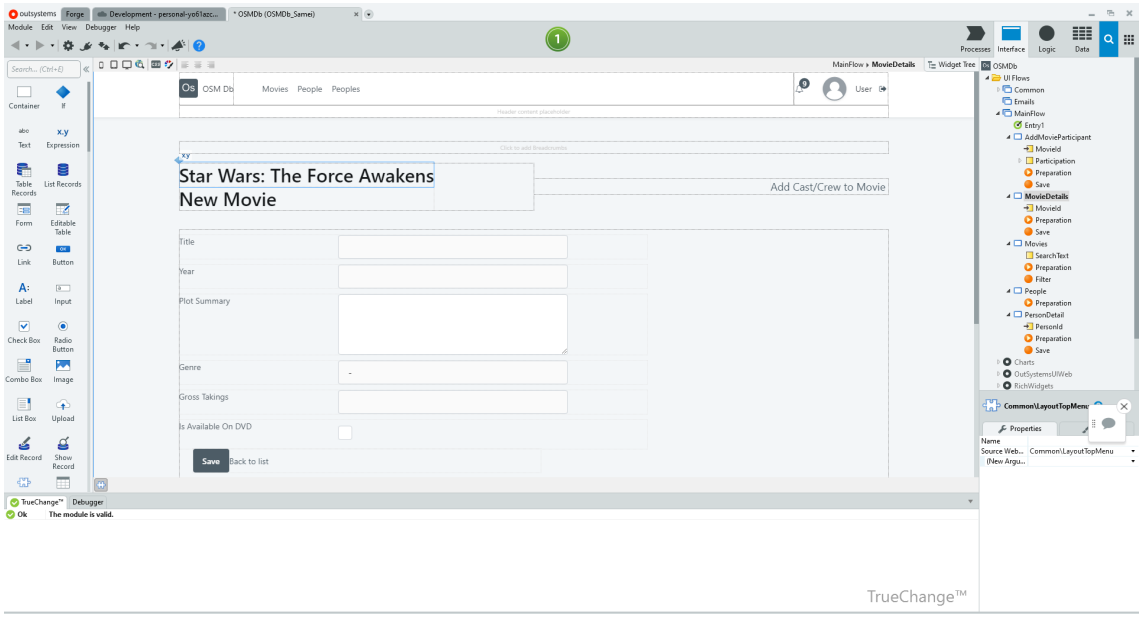


Figure 2.2: OutSystems Service Studio 11.

The left panel contains application components that can be dragged to the center

panel. The center panel shows the application you are building. The right panel displays the structure of the application in one of the four views selected at the top of the rightmost panel: Process, Interface, Logic, and Data.

OutSystems AI automates repeating and routine tasks, for example, by creating "pre-fab" screens for manipulating data in a database entity. An AI-based static code analysis tools issue warnings about security risks and performance problems, and there are tools for analyzing the code quality, e.g., the amount of technical debt [22][25].

OutSystems is a proprietary LCDP, and that may raise some security concerns since the source code is not open-sourced. At the design time, OutSystems uses static code analysis to warn the developer of possibly vulnerable code, such as SQL injection threats, and unvalidated redirects. Developers can customize the configurations used for security checks, for example, to get notifications of customized security issues.

OutSystems integrates to third-party identity management services such as These tools include Active Directory using LDAP, SAML, and OAuth. These integrations provide good secure identity management solutions, but their downside is that they cannot be used to identify the developers. They can be only identified using OutSystems' login credentials.

OutSystems is capable of enforcing HTTPS/SSL encryption on all web applications. All requests from web services to external systems are logged for security reasons. The system also logs changes made by developers, system administrators, and application managers for future auditing purposes [26]. In this thesis we will be assessing applications developed with Service Studio 11.52.

2.3 Security assessment in general

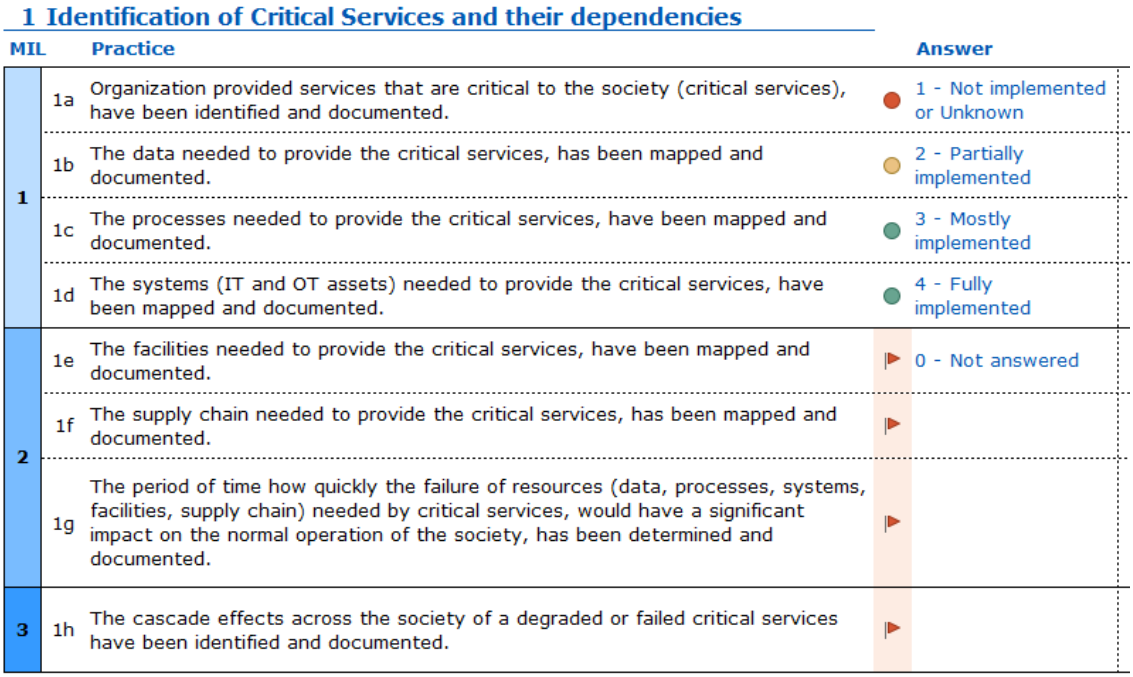
A security assessment is a process where an organization tests its security. A security assessment can be done by someone from an external organization or by someone who works for said organization.

An external assessment is usually done when a company wants to be seen as legitimate by others. Probably the most common reason for an external assessment is when an organization is trying to get certified for a cybersecurity standard. Being audited by an external body brings more credibility than self claiming to meet the standard. The proof of meeting the security standard is the certificate. These certificates typically need to be re-evaluated within a certain amount of time. For ISO/IEC 27001, the certification is valid for three years [27], and the company needs to do maintenance in between the re-evaluation. During a cybersecurity certification audit, the auditors have a list of predefined requirements that they will compare against the company.

Externally done security assessments can also occur during a buyout or when two companies are about to merge. Typically in these cases, the auditors pay special attention to see if the systems have had a previously unknown security incident. If one of the companies is very large department, they can sometimes use their internal cybersecurity experts.

There can be many reasons why a company might want to do an internal security assessment. The first reason is to get a baseline of their current security. When a company reaches a certain milestone or becomes big enough, it might want to start figuring out its security maturity. This security maturity then can be used to demonstrate to a customer that the organization takes care of their security and is safe to use. Figure 2.3 shows examples from the security maturity questionnaire from the Finnish National Cyber Security Center (NCSC-FI).

Whereas the technical security assessment might need external help, the admin-



1 Identification of Critical Services and their dependencies

MIL		Practice	Answer
1	1a	Organization provided services that are critical to the society (critical services), have been identified and documented.	1 - Not implemented or Unknown
	1b	The data needed to provide the critical services, has been mapped and documented.	2 - Partially implemented
	1c	The processes needed to provide the critical services, have been mapped and documented.	3 - Mostly implemented
	1d	The systems (IT and OT assets) needed to provide the critical services, have been mapped and documented.	4 - Fully implemented
2	1e	The facilities needed to provide the critical services, have been mapped and documented.	0 - Not answered
	1f	The supply chain needed to provide the critical services, has been mapped and documented.	
	1g	The period of time how quickly the failure of resources (data, processes, systems, facilities, supply chain) needed by critical services, would have a significant impact on the normal operation of the society, has been determined and documented.	
3	1h	The cascade effects across the society of a degraded or failed critical services have been identified and documented.	

Figure 2.3: Example questions from the NCSC-FI's security maturity questionnaire. [28]

istrative tasks will most likely need to be done internally.

The second reason to use internal security assessment is to improve the existing security in the company. For example, in an IT product company, the employees most likely know the company's products better than an external assessor. Since they know the product, they could be used to do threat modeling, which then could be used to make the product more secure. If the company periodically does security assessments internally, it can also have custom-made technical assessment tools. These tools can be more effective when assessing internal security than the off-the-shelf tools that an external tester might use. The company might also do an internal security assessment because they want to see if the employees follow the company's security program. Do the software developers follow the security guidelines and checklists they have been provided? Do the sales employees inform

the security department about a suspicious email they have been receiving lately?

The third reason a company might want to do an internal security assessment is because of a security incident. When a security incident happens, the company usually goes into an alert mode. If this happened here, where could it happen next? The last chapter of a typical incident report is ”Lessons learned”. As the name suggests, this chapter describes what the company has learned from the incident and what security measures they should place so it will not happen again.

2.4 Summary

We have now opened the topics that are necessary to understand this thesis. Cybersecurity standards come for different purposes. Some standards, such as the ISO/IEC 27000 series, are designed to show that the organization meets predefined criteria and qualifies for that standard. The organizations are then given a certificate to prove that they meet these criteria. Others, such as ASVS are community-driven and designed for more internal use. It can be used as a security checklist when developing an application or as a guide for automated unit tests. In biddings, a procurer can use it to define a set of security features required for the application.

Security standards are closely related to security assessments. When a company wants a security certificate, they call for an external assessor to conduct the audit. In internal security audits, the company typically wants to see and test its current level of security. The results of the internal security audit can be used, for example, to help evaluate the company’s security maturity index.

LCDPs also come in different shapes and sizes. There are platforms where the developer does not need to know how to code, like in Microsoft Power Platform. Then some LCDPs are specifically created for the developers, such as OutSystems. In both cases, the platforms use AI to speed up the development process. The AI can create boilerplate code and do routine tasks.

3 Multi-vocal literature review

3.1 Academic research

During the academic research process we began to think how we could categorize the academic papers to present them once we have found them. In the end, we identified four categories that hold most of the research papers. These categories are ”analyzing low-code community discussions”, ”testing in low-code”, ”researching low-code”, and ”developing new LCDPs”.

We also wanted to see different themes throughout the papares, and on top of the categorization, we decided to sort them according to their themes. Whereas one paper would belong to only one category mentioned before, the same paper could have multiple themes. The themes we identified were ”Platform comparison”, ”Citizen developer”, ”OutSystems”, ”Community”, ”Testing”, ”Security”, and ”Reserach”. The papers, that did not seem to fit into these categories and themes were put into a category and theme called ”Mixed”.

ANALYZING LOW-CODE COMMUNITY DISCUSSIONS

Low-code software development is on the rise in developer communities. Alamin et al. [29] analyzed 3597 questions and 1188 answers from Stack Overflow that dealt with the nine popular low-code development platforms. They applied a topic modeling algorithm, Latent Dirichlet Allocation, and identified 13 topics. These topics got grouped into one of the following categories Customization, Platform

Adaption, Database Management, or Third-Party Integration. The customization category was the most popular, covering five topics and having 39.7% of the questions categorized into it. Platform Adaption had 21.9% of the questions, Database Management 21.5%, and Third-Party Integration 16.8%. The topic with the most questions was "External Web Request Processing" covering 10.4% of the questions. The topic with the most unanswered questions was also the most popular, "Dynamic Event Handling" in the Customization category had the highest average view count. They also discovered that from the point of software development life cycle, 85% of the questions were related to implementation. They also found that testing is seen as challenging due to the graphical nature of the platforms.

Luo et al. [30] did another study where they analyzed Stack Overflow and Reddit's low-code development communities. They extracted 73 posts from Stack Overflow and 228 from Reddit. Their study focused on the developers' opinions on low-code platforms and analyzing the platforms based on their features. They found that when expert developers talk about low-code they mean you can create software with significantly small amounts of actually coding and sometimes not needing to code at all. In total, they identified 21 development platforms that got mentioned at least ten times, Bubble.io being the most popular with 96 posts mentioning it. Expert developers prefer open-source platforms, but only seven of the 21 platforms were open-source. Luo et al. also found that platforms, designed for mobile application development, are in the highest demand. This argument gets support from the fact that the most popular programming language used in these platforms are Java and JavaScript. React Native, a JavaScript mobile applications framework, is implemented in many of these platforms. In many posts, the developers feel that LCDPs are intuitive, easy to use, and newbie-friendly compared to programming languages. At the same time, they feel that these platforms have a high learning curve, especially those that provide complex functions. Developers also feel that

even though LCDP's are newbie-friendly, expert developers may find them hard to use and feel that they are being limited by the environment. This mostly becomes an issue once the developers begin adding custom code to the platform.

Although Silva et al. [31] did not study developer discussions, they did develop a tool to analyze developers' first experience when using LCDP. They observed how expert programmers and novice programmers handled the same task. The experts had a software-engineering background, and the novices had backgrounds in social sciences, economics, and finance. They found that expert programmers are more explorative in the platform and face unique issues frequently, but these issues are more interaction-related and appear less often. The novice programmers, on the other hand, faced recurring issues frequently. These issues seemed to arise from the lack of programming experience and skills. The novice programmers also took more time to complete the tasks.

TESTING IN LOW-CODE

Khorram et al. [32] analyzed the testing components of five commercial LCDPs. They identified the characters of low-code testing and created a feature list based on those characters. They then used that list to compare the testing components in current platforms and argue that it can be used as a baseline when creating new components in the future. Their feature list was used to determine the current state of the testing components in the selected five LCDPs. Present platforms have good support for testing, although mainly because of their integrated third-party testing tools such as SoapUI, Selenium, Jasmine, and others. The tools provide a medium level of automation and still typically require a technical person to create the tests. The testing tools work as a good collaboration-tools between the developer and the tester, and the platforms have test monitoring and reporting features. Based on these findings, they also discuss the challenges and opportunities for low-code

testing. The three main topics they found were citizen developers' role in testing, cloud testing, and the need for high-level test automation.

RESEARCHING LCDP

Although LCDPs have become popular recently, few traditional research papers about the subject exist. Horvath et al. [33] propose their views for the next generation of low-code platforms, which they call low-code engineering platforms (LCEPs). They present three research lines in their paper. The first research line is utilizing multi-tenant architecture, where different teams can work on the same project concurrently. The second research line they propose is reactive model transformation to improve scalability. Finally, their last proposed research line is to combine the two research lines, to create a multi-tenant reactive model transformation benchmark research line. This research line's goal should be that future LCDPs can understand the context and select the most suitable transformation engine.

Jahanbin et al. [34] researched how LCDPs could intelligently partition models during run-time. With LCDPs becomming more popular, the projects they are used also become larger. Jahanabin et al. therefore argue that model managements programs, which perform the validation, transformation, and merging of models and such, will soon reach their limits. Their approach uses static analysis to create an plan of execution, called effective metamodel. This effective metamodel tells the model managements program's which elements are necessary for executing the program at compile time. After creating the effective metamodel, only the relevant parts of it are loaded into the memory to reduce loading times. Likewise, when the parts are not relevant anymore, they are removed from the memory.

Philippe et al. [35] did a similar study. Their study focused on combining multiple execution strategies and seeing what kind of impact doing so would have on the model management. They found that combining these strategies can be used for performance optimization, but note that it can also affect calculation efficiency.

Bexinga et al. [36] developed a new approach to converting designs into web-application artifacts. This approach takes advantage of transformation and meta-modeling techniques, and it was implemented into the OutSystems development process. The preliminary results show that implementing this approach, the conversion from design into a web component can be made up to 75% more efficiently and effectively, depending on the project's complexity.

Since LCDPs use visual programming languages and their users include citizen developers, their recommendation systems are different from traditional development platforms. Almonte et al. [37] studied the recommendation systems for LCDPs. Their study focused on creating a framework that would automate the construction of recommendation systems. They used three different datasets and found that they could build a recommendation system that can recommend methods, attributes, and superclasses for a given class.

Another study about recommendation systems was done by Di Sipio et al. [38] They used metamodeling and dedicated supporting tools to help developers build custom their own recommendation systems. Whereas Almonte et al. did research on LCDPs recommendation systems, Di Sipio et al.'s research covers other software engineering IDEs as well.

DEVELOPING NEW LCDPs

A theme we came across alot was what could be described as "we created an LCDP for purpose X". Daniel et al. [39] created the Xatkit framework to make developing and deploying chatbots and voice bots easier. Arora et al. [40] developed a platform where developers can generate code for a more general-purpose use such as data integration. Zolotas et al. [41] on the other hand, created an LCDP in 2018 to help enterprises automate their "webification" of their applications in a secure way.

MIXED

The academic research on LCDP's is still a bit scattered. The papers presented in this section are mostly unique. A theme that could have been its subsection, but we decided not to make one, was papers that somehow focused on the OutSystems platform. OutSystems seems to be actively developing its platform and publishing research papers based on the development. Jacinto et al. [42] created test mocks to enhance OutSystems testing ability. Their preliminary results show that these test mocks could speed up the software development process. Martins et al. [43] used OutSystems to develop an application for business automation and innovation.

In 2019 Lourenço and Eugenio [44] published an article about OutSystems TrueChange™ engine and how it manages to handle large models that have over 200 000 individual elements. A year later, Lourenço et al. [45] did a case study about changing the OutSystems delivery process. In their previous delivery process, they released a new version once every two years, resulting in the new version not working with older components. Their current process is continuous delivery, and now they can implement changes to their platform more rapidly. The customer can also decide whether or not they want to use those changes.

In 2020 Fernandes et al. [46] studied the effects when OutSystems got implemented in a project-based learning software engineering course. They found that they could give individual feedback to the students even though there were more than 200 students enrolled in the course. The students were majoring in different subjects, such as Informatics Engineering, Design and Multimedia, and Electrical and Computer Engineering. Even though students were majoring in different subjects, this did not affect their grades. On the contrary, using LCDP seemed to even the playground drastically.

We also found two papers from 2020, which looked at platform comparisons. Sanchis et al. [10] study focused on comparing virtual factory open operating system,

vf-OS, to other LCDPs. The comparison was done with a feature list, they had created. They found that vf-OS met the most criteria, meeting 15 of the 16 criteria by default and the last one with configuration. The second best platform met 14 criteria, and the poorest performing platform only met 10 criteria from the 16. Sahay et al. [47] compared 8 LCDP's to a list of 35 control features they had made. The poorest performing LCDP's score was 21 out of 35 control features, and the highest score was 30 out of 35.

3.2 Industrial research

Based on chapter 3.1., it appears that the research on LCDPs is industry-driven by companies such as OutSystems[1] and Mendix[2]. Thus it is also worth looking for industrial materials, such as industrial papers, projects, and blog posts. Since this thesis focuses on the security of a LCDP, the industrial research part of the literature review will focus purely on the security aspect. Narrowing the industrial research to only cover security will also reduce the amount of results that we will get.

The first industrial paper worth mentioning is *7 Deadly Sins of Low-Code Security and How to Avoid Them* by Zenity [48]. Zenity is a company that specializes in low-code/no-code application security[3]. They list the seven most common risks and concerns low-code applications might have, and using their expert knowledge, give their tips on avoiding them. The risks and concerns are not in order, and they are as follows; Privilege escalation, Data leakage, Insecure authentication, Misconfigurations, Dependency injection, Oversharing, and Application impersonation.

A second industrial material, which is still in progress, is OWASP Top 10 Low-Code/No-Code Security Risks [49]. The project is still in progress, and currently

[1]https://www.outsystems.com/

[2]https://www.mendix.com/

[3]https://www.zenity.io

holds the same seven risks that the Zenity's project has.

We also found some blog posts relating to low-code security and concerns. One post made 2021 by Bill Doerrfeld [50] points out these concerns. These concerns derive from a conversation that the writer had with the CTO of Veracode[4], a company that specializes in application security. Doerrfeld points out three concerns; API integrations, the lack of security awareness among citizen developers, and using 3rd-party libraries with vulnerabilities. Another blog [51] states that developers working outside of the IT department can be a risk for an application for not following the companies cybersecurity guidelines.

There is another security concern related to the visibility of the code the platforms provide. Blogs [51][52] point that companies have their guidelines on software development but cannot verify that these platforms follow these guidelines.

3.3 Conclusion

As our findings in Table 3.1 shows, nine out of the 20 academic papers and two of the four industry papers got categorized into two or more topics. The table shows that especially testing and security are not frequently researched topics in the academic community. It also shows that while platform comparison is a researched topic, it is usually not combined with other topics. Current platforms provide built-in security that developers need to rely on but so far no one has compared them or evaluated them. It also seems tat there are not any previous case studies or frameworks on how to conduct a security assessment on an LCDP.

[4]https://www.veracode.com

	Platform comparison	Citizen developer	OutSystems	Community	Testing	Security	Research	Mixed
Alamin et al. 2021				x				
Luo et al. 2021				x				
Silva et al. 2021		x						
Khorram et al. 2020	x	x			x			
Horvath et al. 2020							x	
Jahanbin et al. 2020							x	
Philippe et al. 2020							x	
Bexinga et al. 2020			x				x	
Almonte et al. 2020							x	x
Di Sipio et al. 2020							x	x
Daniel et al. 2020								x
Arora et al. 2020								x
Zolotas et al. 2018						x		x
Fernandes et al. 2020		x	x					x
Sanchis et al. 2020	x							
Sahay et al. 2020	x							
Jacinto et al. 2020			x		x			
Martins et al. 2020			x					x
Lourenço and Eugenio 2019			x					
Lourenço et al. 2020			x					x
Zenity 2020						x		
OWASP TBA						x		
Bill Doerrfeld 2021		x				x		
IT Pro Today 2021		x				x		

Table 3.1: Research papers and their topics.

OutSystems seems to be an active contributor to the community. Their research tends to be cross-topic, and they aim to combine their platform with the research. ”Citizen developer” is a somewhat researched topic. Khorram et al. researched citizen developers and testing and noticed that there is still work to be done before citizen developers can work more independently. However, there are no papers that handle citizen developers and security. Citizen developers are said to improve the developer shortage, but they might impact the overall security. If an LCDP has built-in security features, but the citizen developer ignores or does not understand the warnings, what are the features' purposes?

The industry has noticed this. Two of the four industry papers cover both topics. The industry papers paint a picture that companies will solve this problem by putting their citizen developers in more extensive security training. So far, the Zenity's or OWASP's projects have not taken into account citizen developers from the security point of view.

4 OutSystems compliance with ASVS

As we saw in chapter 3, there are no previous studies about compliance analysis between an LCDP and a cybersecurity standard. This further supports our claim that there is a need for one, especially since citizen developers might have a lower level knowledge in cybersecurity. They most likely will have a basic understanding in cybersecurity so they will need to rely on the pre-made functionalities and security features in LCDPs.

Since OutSystems is proprietary software, and we do not have clear visibilty inside the pre-made components that we use, the traditional methods of conducting security assessment do not work. There are also some cases where developers can write their custom C#/.NET code for special integrations and we need to take these into account as well. To take all of these special requirements into account, we have developed an evaluation scheme for the compliance analysis.

4.1 Compliance analysis of ASVS Level 1

For ASVS Level 1 we have divided compliance into four compliance levels (CL). CL 1 requires that the feature is built-in into the OutSystems, and developers can use it out-of-the-box. CL 2 means the feature is implementable through the OutSystems Forge as a trusted or OutSystems supported component. In CL 3, the feature is im-

plementable with a Forge component, but not a trusted, or OutSystems supported component. OutSystems Forge is OutSystems' marketplace where anyone can publish OutSystems applications and components that serve a wide range of needs. The components are open source and distributed under the BSD license, "take it and use as it is." The final compliance level, CL 4, can be implemented by programming the feature. In addition to the compliance levels we will also use Developer Action (DA), because the requirements themselves are implemented as part of software production process and cannot be measured with the defined compliance levels.

4.2 Compliance analysis of ASVS Level 2

As mentioned in chapter 2, in ASVS level 2 the requirements become more abstract. The requirements might need implementation in the organization, which uses OutSystems, but might not concern OutSystems. For this reason, this level will look at compliance in OutSystems and user organization. Each requirement will be assessed individually for both parties, i.e., "Does OutSystems support this requirement? If yes, does the user organization need to implement or enable it or is it enabled by default?". The possible outcomes for a requirement are seen in table 4.1.

Possible answers for a requirement	OutSystems	Developer
OutSystems supports this requirement as a pre-made feature.	Yes	N/A
OutSystems supports this requirement, developer needs to enable it.	Yes	DA
OutSystems does not support this requirement, developer needs to impelement it.	No	DA
This requirement does not concern OutSystems, just the developer.	N/A	DA

Table 4.1: The possible answers for a requirement, and what the answers mean for OutSystems and for the IT-user. N/A means Not Applicaple and DA means Developer Action.

The possible answers for OutSystems are Yes, No and N/A. Yes means that

OutSystems supports the requirement, no means it does not support the requirement, and N/A means that the requirement is not applicable. The Possible answers for the developer are N/A, and Developer Action, DA. In this context, our use of DA means that to meet the requirement, the developer needs to do something actively. Either configure OutSystems correctly, create their implementation to meet the requirement, or create some other routine to meet the requirement.

4.3 Results

In total, the ASVS 4.0.3 has 279 requirements. We will be evaluating OutSystems' compliance with levels 1 and 2 which have 256 requirements. These requirements are divided into 14 categories, which are divided into subcategories. In this chapter, we will present the results by category. The results are combined and shown in Table 4.2.

4.3.1 Defining the shared responsibility between OutSystems and the Developer

Evaluating an LCDP-based application is not as trivial as evaluating a traditional application. In traditional software development, the developer has full control of the application and can modify it freely. In low-code software development (LCSD), the developer relies heavily on the built-in functionalities and needs to trust them. In open-source LCDPs the developer can build trust by reading the source code but in proprietary software, the source code is not available. In cases like this, the platform provider needs to somehow compensate for the lost transparency, and in OutSystems' case, they provide a security portal [53] where the developer can learn about OutSystems' practices. The security portal provides a comprehensive list of security reports done by third party vendors, so the trust is not blind.

Category	**ASVS Level 1**					**ASVS Level 2**			
	CL1	CL2	CL3	CL4	DA	NA/DA	Yes/DA	No/DA	Yes/NA
Category 1	-	-	-	-	-	15	19	1	3
Category 2	24	-	2	1	-	5	12	5	1
Category 3	10	-	-	-	2	3	2	1	-
Category 4	5	-	1	-	2	1	-	-	-
Category 5	10	-	-	16	1	-	1	-	2
Category 6	1	-	-	-	-	-	8	1	3
Category 7	2	-	-	-	1	-	7	-	2
Category 8	2	-	-	-	5	3	3	-	2
Category 9	3	-	-	-	-	2	2	-	-
Category 10	2	-	-	1	-	2	-	-	-
Category 11	3	1	-	1	-	-	2	-	1
Category 12	8	-	1	-	2	2	1	-	1
Category 13	5	-	-	-	1	2	5	1	1
Category 14	9	5	-	1	-	-	6	-	2
Total	84	6	4	20	15	35	66	9	18

Table 4.2: Results for OutSystems' compliance with ASVS.

Because the LCSD relies on pre-made components and configurations, there should be a division on which categories should be on OutSystems responsibility and which ones are on the developer's responsibility. To create the division, we need to look at what OutSystems provides. According to their technical white paper [54], OutSystems protects against nine out of ten of the 2017 OWASP Top 10 list. The only vulnerability that they do not cover is insufficient logging which is typically a developer action case. For mobile devices, OutSystems also provides coverage for nine out of ten of the 2016 OWASP Top 10 Mobile list.

Based on these factors, we can create a rough draft of the categories that OutSystems should support as much as possible. The most critical categories, that

OutSystems should support are session management, authentication, and access management. In OutSystems' technical white paper they clearly state that they provide secure session data, a secure authentication mechanism, and role-based access control for developers and end-users.

In the same technical paper [54] they also clearly state that OutSystems escapes content before it is shown in the user interface, mitigating reflected and stored cross-site scripting vulnerabilities. OutSystems also warns the developer about potential injections and provides sanitization functions for the developers to use for HTML, JavaScript, and SQL. Even though they provide these functions, they do not enforce their usage. Therefore the validation, sanitization, and encoding category is mostly the developer's responsibility. While OutSystems cannot provide full support for logging, their paper does tell how to implement application logging and proper error handling. They also have pre-made error handling and for that reason, error handling and logging should have shared responsibility.

Figure 4.1 shows our thoughts on the shared responsibility between OutSystems and the developer.

The categories are distributed quite evenly. There also seems to be a trend with the categories, the ones that are on the right seem to have more level 2 requirements. The ones on the left seem to have more level 1 requirements. Even though OutSystems does not support logging, it does support error handling quite well, and thus it is put into the middle. Stored cryptography is also in the middle. We believe that OutSystems provides a sufficient level of cryptography, but we think that the developer still has the responsibility to implement them correctly. Data protection could also be in the middle, but we believe classifying data plays an important role in that category and for that reason put it more towards the developer's responsibility. We also put Business logic, API and web services, and configuration more towards the developer's responsibility. The application's business logic is tricky to place because

it relies both pre-made components that come with the platform and the logic created by the developer. However, we feel it relies more on the developer and thus we placed it more to the right. We also placed API and web services, and configuration more towards right. OutSystems does not need provide external services and we also that configurations are application specific and thus more developer's responsibility.

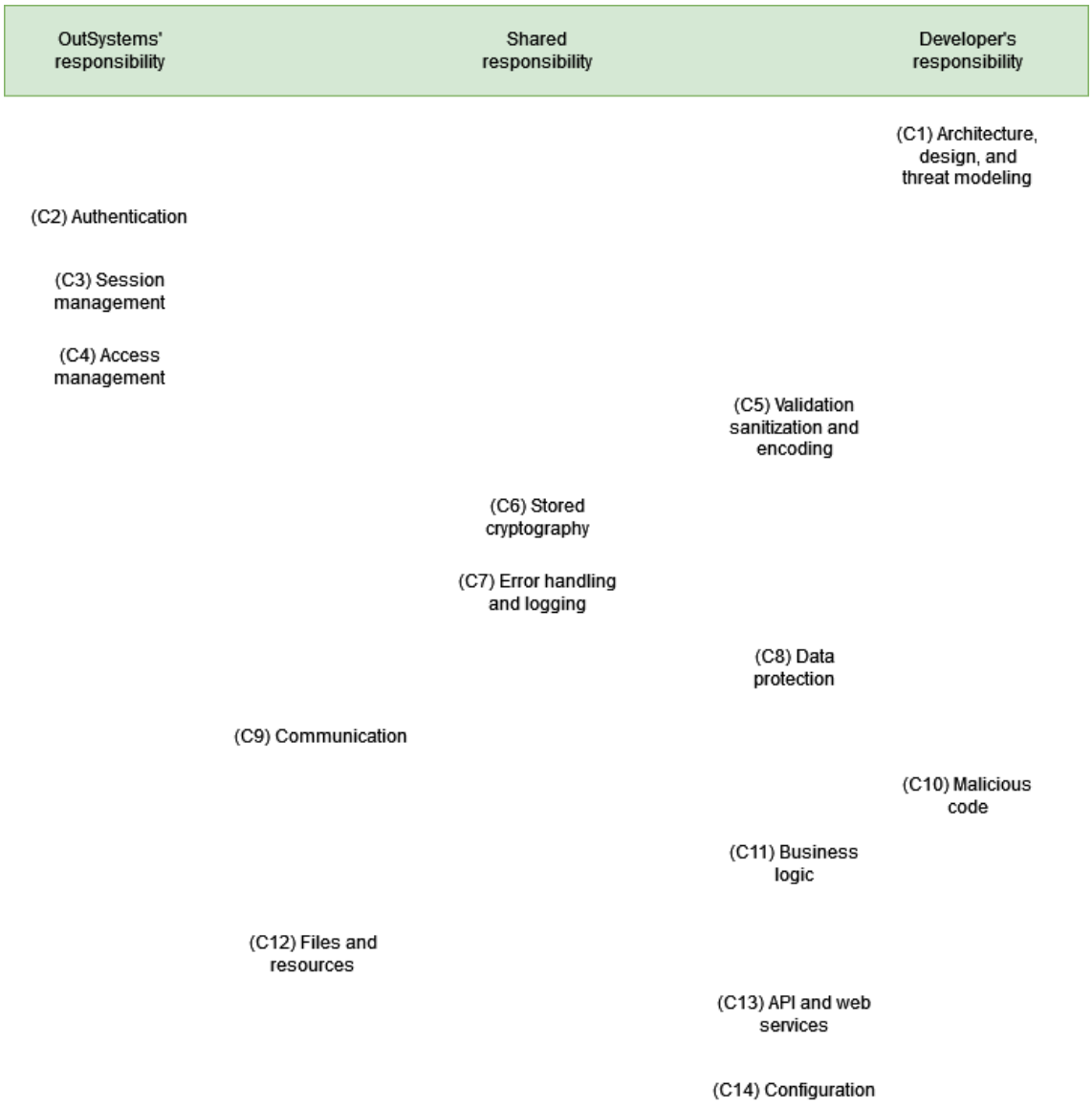


Figure 4.1: Expectations for the shared responsibility between OutSystems and developer.

4.3.2 Findings

Our findings are somewhat interesting. Categories 2 (Authentication) and 3 (Session management), which we defined OutSystems responsibility and thus pre-made into it, were not fully supported by OutSystems. Category 2 had the most requirements that did not have built-in support in OutSystems.

Out of all the categories, only three were fully compliant with ASVS levels 1 and 2. These categories were 7, 8, and 9. We will be going through all of the categories and our findings in them. We have compiled all of the results into Table 4.3 on page 33.

C1 ARCHITECTURE, DESIGN AND THREAT MODELING

The first category deals with the application's architecture, design, and threat modeling. This category has 38 requirements, all of which were ASVS level 2 requirements. Most of them are either supported by OutSystems or do not apply to OutSystems. One example that OutSystems does not natively support, is requirement id V1.6.2. It requires that key vaults or API-based alternatives are used to protect key material and other secrets. OutSystems does not provide this kind of service in the standard edition, but they offer a Key Management Service for enterprises.

C2 AUTHENTICATION

The authentication category had a total of 50 requirements. Of the 50 requirements, 27 were level 1 requirements, and 25 were level 2 requirements.

Requirements related to password security, general authenticator security, credential recovery, look-up secret verifier, and out-of-band verifier sections have overall good support in OutSystems, mostly requiring the developer to enable them. Cryptographic verifier and credential storage sections are also easily implementable. Some

of their requirements might require the developer to create them or do additional work to enable them. The additional work isn't too complex but still requires some manual work to meet the requirement.

The authenticator lifecycle and one-time verifier are sections that require the most work to meet the requirements in this category. OutSystems does not provide MFA natively or have a trusted or OutSystems-supported module downloadable from the OutSystems Forge to authenticate the developer. Creating a module to handle it requires significant work. The developer also needs to build a component so that the end-user has the option to use physical authentication devices, such as U2F tokens.

C3 SESSION MANAGEMENT

Session management had a total of 18 requirements. Of the 18, 12 were level 1, and six were level 2 requirements. OutSystems supports this category quite well since session management is a built-in feature in OutSystems. Out of the six sections in this category, only one has requirements that the developer needs to implement. The requirements relate to terminating all active sessions for a single user. They are business logic requirements and thus do not apply to OutSystems.

C4 ACCESS MANAGEMENT

The fourth category is access management. This category has nine requirements, and all but one are level 1 requirements. As with the previous category, this category is also quite well supported by OutSystems. The only requirement not supported by it deals with multi-factor authentication for the admin user, including developers and testers. OutSystems Forge has a module that the developer can implement, but it is not trusted, and it is not an OutSystems Supported module.

C5 VALIDATION, SANITIZATION, AND ENCODING

The validation, sanitization, and encoding category had the most failing requirements. Of the 30 requirements, 27 were level 1 requirements, and all 16 failing requirements belonged to that level. Having over half of the requirements fail in one category might appear bad, but in reality, it is not. Many of these requirements are business logic types and require the developer to implement them. An example of these business logic type requirements is requirement id V5.2.2, "Verify that unstructured data is sanitized to enforce safety measures such as allowed characters and length."

C6 STORED CRYPTOGRAPHY

Stored cryptography has 13 requirements, of which one is a level 1 requirement and 12 are level 2 requirements. As with other categories, this category is also quite well supported by OutSystems. The data classification section does not apply to OutSystems, since it is a business process. However, OutSystems does provide some help with that. It provides CryptoAPI, which is useful when encrypting the data. The only requirement on the developer's responsibility is implementing a secret management solution.

C7 ERROR HANDLING AND LOGGING

Category "error handling and logging" has 12 requirements, of which 3 are level 1 requirements and nine are level 2 requirements. OutSystems supports all the requirements as built-in features or helps the developer implement them.

C8 DATA PROTECTION

Data protection is a category where the responsibility between OutSystems and the developer is more shared. There are 15 requirements, seven of which are level 1 requirements and eight of which are level 2 requirements. Due to the nature of this

category, most of the requirements are the developer's responsibility. Even though most of them are the developer's responsibility, OutSystems still has some features that support these requirements. One of these features is CryptoAPI which provides extensive cryptographic tools to protect sensitive data.

C9 COMMUNICATION

The ninth category is communication. This category has seven requirements, four of which are level 1 requirements and three which are level 2 requirements. OutSystems either supports all of the requirements, or they do not apply to it.

C10 MALICIOUS CODE

Category 10, malicious code, has a total of 5 requirements. Even though one answer is compliance level 4, it is not that bad from OutSystems' point of view. The requirement relates to protecting DNS names so that subdomain takeovers cannot occur, and could be seen more like a business process.

C11 BUSINESS LOGIC

Category 11 relates to business logic security. It has a total of eight requirements. As OutSystems' technical white paper [54] describes, OutSystems has lots of built-in features. Because of these features, OutSystems is fully compliant in this category.

C12 FILES AND RESOURCES

Category 12 has a total of 15 requirements. OutSystems is fully compliant with this category except for one requirement, which requires that files obtained from untrusted sources are run through an antivirus scanner to prevent uploading and serving known malicious content.

C13 API AND WEB SERVICES

The second to last category is API and Web Services, which has 13 requirements The category has one requirement, which requires more business logic, but even it is implementable with OutSystems. The requirement relates to validating JSON Schemas.

C14 CONFIGURATION

The final category is Configuration, which has 23 requirements. This category is also mostly supported by OutSystems. There is one requirement that is compliance level 4. The requirement requires that subresource integrity is used if application assets are hosted on some external host, such as Content Delivery Network. Since OutSystems does not use external hosts, but the developer might use them, this requirement falls to the developer's responsibility. Some requirements relate to HTTP headers, which require that the developer downloads the Factory Configuration module from OutSystems Forge.

4.3.3 Summary of results

Since OutSystems does not meet all of the requirements, it is worth looking more closely into the failing requirements to see how it could be improved. The failing requirements have been collected into two tables.

Requirements V2.1.8, V2.1.12, V4.3.1, and V12.4.2 meet CL 3 and thus appear easy to fix. They meet the requirement at compliance level 3 and already have an existing solution in OutSystems Forge. For V2.1.8, there is one popular module [55] to measure password strength. The same goes with the requirement V2.1.12, dealing with toggling the password [56]. These modules' creators are ranked in the top 100 in the OutSystems community, which has more than 430,000 members [57]. They are also marked as "MVPs" so it is safe to say that the developers are trustworthy.

OutSystems would only need to review these modules' source code and make them trusted modules promoting V2.1.8 and V2.1.12 to CL 2. OutSystems Forge also has an existing module [58] whose creators include OutSystems' employees. This module fulfills the requirement V4.3.1 by itself, and marking the module as Trusted or OutSystems supported would promote OutSystems to meet the requirement.

OutSystems Forge also has a module [59] that would meet the requirement for requirement V12.4.2. Requirement V12.4.2 deals with executing files uploaded from an untrusted source through a virus scanner before the upload. The creator of this module is ranked #13 in the community rankings and has the "MVP" title in their profile.

Requirements that meet Compliance Level 4 are more problematic. Some do not relate to OutSystems, such as V10.3.3, but some are a bit more tricky such as V2.8.1. V2.8.1's description states "Verify that time-based OTPs have a defined lifetime before expiring.". The reason this is a bit more tricky is how you interpret it. Adding a Multi-Factor Authentication is CL 2, so one might argue that this is a CL 2, but OutSystems does not help or support defining lifetime for OTPs. We concluded that this is a false positive CL4, but have decided to keep it as a CL4.

Other CL 4 requirements appear more business logic-related issues, such as requirement V11.1.3, or most of the requirements in category 5. As was mentioned previously, category 5 had 16 failing requirements. We will make an exception in this category and not go through all of these requirements, but instead handle them more generally. These requirements relate to either validating, sanitizing, or encoding user-given data. We see these requirements are more business logic types and require the developer to implement them. The static code analyzer in OutSystems does warn the developer of SQL injections, but they are only warnings and not mitigated. Thus OutSystems fails so many requirements in this category.

Some requirements resemble more business processes and could be seen as false

Requirement ID	Requirement Description	Compliance Level
V2.1.8	Verify that a password strength meter is provided to help users set a stronger password.	CL 3
V2.1.12	Verify that the user can choose to either temporarily view the entire masked password or temporarily view the last typed character of the password on platforms that do not have this as built-in functionality.	CL 3
V2.8.1	Verify that time-based OTPs have a defined lifetime before expiring.	CL 4
V4.3.1	Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.	CL 3

Table 4.3: Summary of ASVS Level 1 requirements that OutSystems does not fill.

positives, such as requirement V10.3.3 which deals with subdomain takeovers. Subdomain takeovers are not in the scope of OutSystems, and thus the CL4 is not that severe. Table 4.3, 4.3.3, 4.5, and 4.6 show level 1 failing requirements, their descriptions as they are in the ASVS and their compliance level.

There were a total of seven level 2 requirements that OutSystems did not support. Four of the failing requirements relate to category 2, authentication and of the four, two relate to OTPs. Table 4.7 shows all failing requirements in level 2.

V2.8.2 requires that symmetric keys used with OTPs are highly protected. One could argue here the same as with requirement V2.8.1, but OutSystems does not support this requirement. Requirement V2.8.6, which requires that physical OTPs have to be able to be revoked also gets failed for this reason. Requirement V2.3.2 relates to MFA as well, it requires that the user can use physical authenticators, such as U2F. Since OutSystems does not support MFA natively, this requirement fails as well. The final category 2-related requirement is V2.4.3. This requirement states that if the PBKDF2 algorithm is used, there should be at least 100 000 iterations used with it. OutSystems provides said algorithm in CryptoAPI, a Trusted module

Requirement ID	Requirement Description	Compliance Level
V5.1.3	Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc) is validated using positive validation (allow lists). ([C5](https://owasp.org/www-project-proactive-controls/#div-numbering))	CL 4
V5.1.4	Verify that structured data is strongly typed and validated against a defined schema including allowed characters, length and pattern (e.g. credit card numbers, e-mail addresses, telephone numbers, or validating that two related fields are reasonable, such as checking that suburb and zip/postcode match). ([C5](https://owasp.org/www-project-proactive-controls/#div-numbering))	CL 4
V5.1.5	Verify that URL redirects and forwards only allow destinations which appear on an allow list, or show a warning when redirecting to potentially untrusted content.	CL 4
V5.2.2	Verify that unstructured data is sanitized to enforce safety measures such as allowed characters and length.	CL 4
V5.2.3	Verify that the application sanitizes user input before passing to mail systems to protect against SMTP or IMAP injection.	CL 4
V5.2.4	Verify that the application avoids the use of eval() or other dynamic code execution features. Where there is no alternative, any user input being included must be sanitized or sandboxed before being executed.	CL 4
V5.2.5	Verify that the application protects against template injection attacks by ensuring that any user input being included is sanitized or sandboxed.	CL 4
V5.2.6	Verify that the application protects against SSRF attacks, by validating or sanitizing untrusted data or HTTP file metadata, such as filenames and URL input fields, and uses allow lists of protocols, domains, paths and ports.	CL 4

Table 4.4: Summary of ASVS Level 1 requirements that OutSystems does not fill.

in OutSystems Forge, but their implementation only uses 37 649 iterations.

The two requirements outside of category 2 are requirements V1.6.2 and V6.4.1. Both describe the usage of key vaults, V1.6.2 requires that key vaults or other API-based services are used to protect key materials for cryptographic service users. V6.4.1 requires that key vaults or other secrets management solution is used to create, store, destroy, and control access to secrets. OutSystems has nothing on to meet them and thus these requirements do not pass.

Tables 4.8, 4.9, 4.10, and 4.11 show level 2 requirements that were marked as N/A / DA.

Requirement ID	Requirement Description
V1.1.2	Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.
V1.1.3	Verify that all user stories and features contain functional security constraints, such as "As a user, I should be able to view and edit my profile. I should not be able to view or edit anyone else's profile"
V1.1.7	Verify availability of a secure coding checklist, security requirements, guideline, or policy to all developers and testers.
V1.11.2	Verify that all high-value business logic flows, including authentication, session management and access control, do not share unsynchronized state.
V1.12.2	Verify that user-uploaded files - if required to be displayed or downloaded from the application - are served by either octet stream downloads, or from an unrelated domain, such as a cloud file storage bucket. Implement a suitable Content Security Policy (CSP) to reduce the risk from XSS vectors or other attacks from the uploaded file.
V1.14.2	Verify that binary signatures, trusted connections, and verified endpoints are used to deploy binaries to remote devices.
V1.2.1	Verify the use of unique or special low-privilege operating system accounts for all application components, services, and servers. ([C3](https://owasp.org/www-project-proactive-controls/#div-numbering))
V1.2.3	Verify that the application uses a single vetted authentication mechanism that is known to be secure, can be extended to include strong authentication, and has sufficient logging and monitoring to detect account abuse or breaches.
V1.5.1	Verify that input and output requirements clearly define how to handle and process data based on type, content, and applicable laws, regulations, and other policy compliance.

Table 4.8: Summary of ASVS Level 2 requirements that do not concern OutSystems.

Requirement ID	Requirement Description
V1.5.2	Verify that serialization is not used when communicating with untrusted clients. If this is not possible, ensure that adequate integrity controls (and possibly encryption if sensitive data is sent) are enforced to prevent deserialization attacks including object injection.
V1.5.3	Verify that input validation is enforced on a trusted service layer. ([C5](https://owasp.org/www-project-proactive-controls/#div-numbering))
V1.6.4	Verify that the architecture treats client-side secrets–such as symmetric keys, passwords, or API tokens–as insecure and never uses them to protect or access sensitive data.
V1.8.1	Verify that all sensitive data is identified and classified into protection levels.
V1.8.2	Verify that all protection levels have an associated set of protection requirements, such as encryption requirements, integrity requirements, retention, privacy, and other confidentiality requirements, and that these are applied in the architecture.
V1.14.5	Verify that application deployments adequately sandbox, containerize and/or isolate at the network level to delay and deter attackers from attacking other applications, especially when they are performing sensitive or dangerous actions such as deserialization. ([C5](https://owasp.org/www-project-proactive-controls/#div-numbering))
V2.10.2	Verify that if passwords are required for service authentication, the service account used is not a default credential. (e.g. root/root or admin/admin are default in some services during installation).
V2.10.4	Verify passwords, integrations with databases and third-party systems, seeds and internal secrets, and API keys are managed securely and not included in the source code or stored within source code repositories. Such storage SHOULD resist offline attacks. The use of a secure software key store (L1), hardware TPM, or an HSM (L3) is recommended for password storage.
V2.4.5	Verify that an additional iteration of a key derivation function is performed, using a salt value that is secret and known only to the verifier. Generate the salt value using an approved random bit generator [SP 800-90Ar1] and provide at least the minimum security strength specified in the latest revision of SP 800-131A. The secret salt value SHALL be stored separately from the hashed passwords (e.g., in a specialized device like a hardware security module).
V2.6.2	Verify that lookup secrets have sufficient randomness (112 bits of entropy), or if less than 112 bits of entropy, salted with a unique and random 32-bit salt and hashed with an approved one-way hash.

Table 4.9: Summary of ASVS Level 2 requirements that do not concern OutSystems.

Requirement ID	Requirement Description
V2.9.1	Verify that cryptographic keys used in verification are stored securely and protected against disclosure, such as using a Trusted Platform Module (TPM) or Hardware Security Module (HSM), or an OS service that can use this secure storage.
V3.3.4	Verify that users can view and (having re-entered login credentials) log out of any or all currently active sessions and devices.
V3.3.3	Verify that the application gives the option to terminate all other active sessions after a successful password change (including change via password reset/recovery) and that this is effective across the application, federated login (if present), and any relying parties.
V3.5.1	Verify the application allows users to revoke OAuth tokens that form trust relationships with linked applications.
V4.3.3	Verify the application has additional authorization (such as step-up or adaptive authentication) for lower value systems, and/or segregation of duties for high-value applications to enforce anti-fraud controls as per the risk of application and past fraud.
V8.1.1	Verify the application protects sensitive data from being cached in server components such as load balancers and application caches.
V8.1.2	Verify that all cached or temporary copies of sensitive data stored on the server are protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data.
V8.3.5	Verify accessing sensitive data is audited (without logging the sensitive data itself), if the data is collected under relevant data protection directives or where logging of access is required.
V9.2.1	Verify that connections to and from the server use trusted TLS certificates. Where internally generated or self-signed certificates are used, the server must be configured to only trust specific internal CAs and specific self-signed certificates. All others should be rejected.
V9.2.3	Verify that all encrypted connections to external systems that involve sensitive information or functions are authenticated.
V10.2.1	Verify that the application source code and third-party libraries do not contain unauthorized phone home or data collection capabilities. Where such functionality exists, obtain the user's permission for it to operate before collecting any data.
V10.2.2	Verify that the application does not ask for unnecessary or excessive permissions to privacy-related features or sensors, such as contacts, cameras, microphones, or location.
V12.1.2	Verify that the application checks compressed files (e.g. zip, gz, docx, odt) against the maximum allowed uncompressed size and against a maximum number of files before uncompressing the file.

Table 4.10: Summary of ASVS Level 2 requirements that do not concern OutSystems.

Requirement ID	Requirement Description
V12.2.1	Verify that files obtained from untrusted sources are validated to be of the expected type based on the file's content.
V13.4.1	Verify that a query allows a list or a combination of depth limiting and amount limiting is used to prevent GraphQL or data layer expression Denial of Service (DoS) as a result of expensive, nested queries. For more advanced scenarios, query cost analysis should be used.
V13.4.2	Verify that GraphQL or other data layer authorization logic should be implemented at the business logic layer instead of the GraphQL layer.

Table 4.11: Summary of ASVS Level 2 requirements that do not concern OutSystems.

These requirements could have been combined with the Yes / BP category, but we wanted to keep them separated to bring more transparency. Some of the requirements could be seen as subjective, someone might say that a requirement is not met while another might say that it does not apply in the OutSystems case.

As the tables show, there are a total of 36 requirements that do not apply to OutSystems. These requirements focus heavily on categories 1 and 2. Category 1 is self-explanatory since it is architecture, design, and threat modeling, but category 2, authentication, is a bit surprising. It is the same category that OutSystems has the most problems with.

For the rest of the requirements, they seem to be more business logic-related requirements, such as V3.3.4 or V12.2.1

4.4 How to assess an LCDP's security in the future?

For the current standards, we can say that they do not work as they are when applied to LCDPs. Applications developed with them are partly developed by the developer and by the platform. Current security standards only assess a security feature as

Yes/No, but when assessing LCDPs you first need to find what security features are provided with the platform and what require the developer to implement them. Figure 4.2 shows the responsibility in the case of OutSystems.

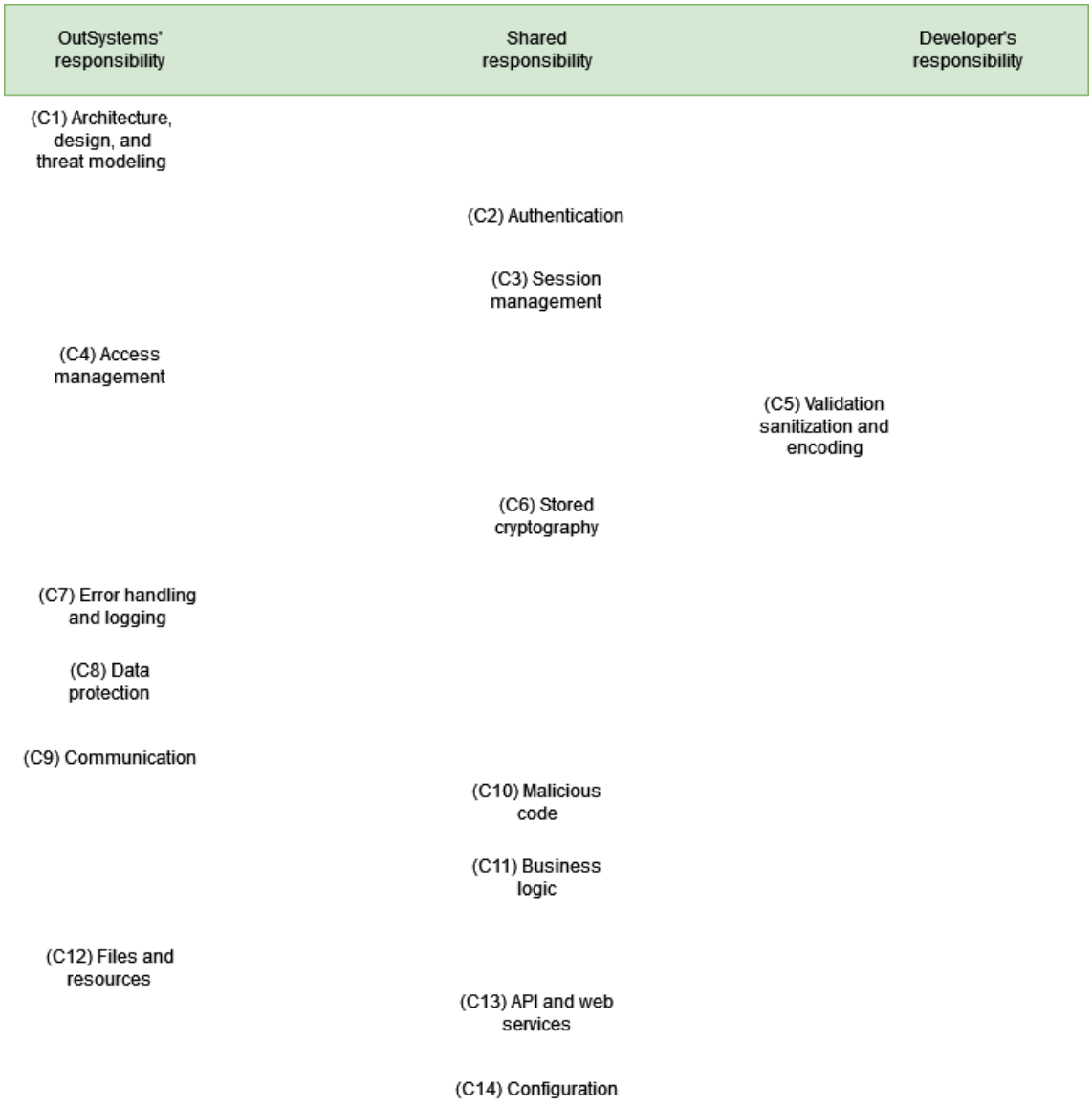


Figure 4.2: Reality for the shared responsibility between OutSystems and developer.

We see that in OutSystems' case, the security is quite well built-in. Six of the 14 categories are classified as OutSystems' responsibility, and only category 5 leans towards the developer's responsibility. If we were to use OutSystems as a baseline

for a more general assessment, we could say that future assessments should focus more on the platform's built-in security features. This kind of assessment is more effortless in an open-source platform since anyone with the expertise can review the source code.

The problems arise when the software is proprietary. For example, since OutSystems is proprietary software, we could not review the platform's source code, and with the lost transparency, it became a black box for us. Thankfully with the OutSystems case, they have solved this obscure. They provide a security portal where they have collected necessary security reports. Third-party auditors have conducted these audits. OutSystems have also extensively documented their ways of working there and, for example, lists the secure coding guides they use when developing their platform. These documents are important, if a security assessment were to be conducted in a company using OutSystems, OutSystems would be part of the development cycle, and their ways of working would also be part of said company's ways of working.

OutSystems also has the advantage in their community. We found their community to be very helpful when we needed some assistance on how its inner parts work. The community is mature, so most of the questions we had, were already answered by an expert and were easy to find.

Requirement ID	Requirement Description	Compliance Level
V5.2.7	Verify that the application sanitizes, disables, or sandboxes user-supplied Scalable Vector Graphics (SVG) scriptable content, especially as they relate to XSS resulting from inline scripts, and foreignObject.	CL 4
V5.2.8	Verify that the application sanitizes, disables, or sandboxes user-supplied scriptable or expression template language content, such as Markdown, CSS or XSL stylesheets, BBCode, or similar.	CL 4
V5.3.1	Verify that output encoding is relevant for the interpreter and context required. For example, use encoders specifically for HTML values, HTML attributes, JavaScript, URL parameters, HTTP headers, SMTP, and others as the context requires, especially from untrusted inputs (e.g. names with Unicode or apostrophes, such as O'Hara). ([C4](https://owasp.org/www-project-proactive-controls/#div-numbering))	CL 4
V5.3.4	Verify that data selection or database queries (e.g. SQL, HQL, ORM, NoSQL) use parameterized queries, ORMs, entity frameworks, or are otherwise protected from database injection attacks. ([C3](https://owasp.org/www-project-proactive-controls/#div-numbering))	CL 4
V5.3.5	Verify that where parameterized or safer mechanisms are not present, context-specific output encoding is used to protect against injection attacks, such as the use of SQL escaping to protect against SQL injection. ([C3, C4](https://owasp.org/www-project-proactive-controls/#div-numbering))	CL 4
V5.3.7	Verify that the application protects against LDAP injection vulnerabilities, or that specific security controls to prevent LDAP injection have been implemented. ([C4](https://owasp.org/www-project-proactive-controls/#div-numbering))	CL 4

Table 4.5: Summary of ASVS Level 1 requirements that OutSystems does not fill.

Requirement ID	Requirement Description	Compliance Level
V5.3.8	Verify that the application protects against OS command injection and that operating system calls use parameterized OS queries or use contextual command line output encoding. ([C4](https://owasp.org/www-project-proactive-controls/#div-numbering))	CL 4
V5.3.9	Verify that the application protects against Local File Inclusion (LFI) or Remote File Inclusion (RFI) attacks.	CL 4
V10.3.3	Verify that the application has protection from subdomain takeovers if the application relies upon DNS entries or DNS subdomains, such as expired domain names, out-of-date DNS pointers or CNAMEs, expired projects at public source code repos, or transient cloud APIs, serverless functions, or storage buckets (*autogen-bucket-id*.cloud.example.com) or similar. Protections can include ensuring that DNS names used by applications are regularly checked for expiry or change.	CL 4
V11.1.3	Verify the application has appropriate limits for specific business actions or transactions which are correctly enforced on a per user basis.	CL 4
V12.4.2	Verify that files obtained from untrusted sources are scanned by antivirus scanners to prevent upload and serving of known malicious content.	CL 3
V14.2.3	Verify that if application assets, such as JavaScript libraries, CSS or web fonts, are hosted externally on a Content Delivery Network (CDN) or external provider, Sub-resource Integrity (SRI) is used to validate the integrity of the asset.	CL 4

Table 4.6: Summary of ASVS Level 1 requirements that OutSystems does not fill.

Requirement ID	Requirement Description	Responsibility
V1.6.2	Verify that consumers of cryptographic services protect key material and other secrets by using key vaults or API-based alternatives.	No / DA
V2.3.2	Verify that enrollment and use of user-provided authentication devices are supported, such as U2F or FIDO tokens.	No / DA
V2.3.3	Verify that renewal instructions are sent with sufficient time to renew time-bound authenticators.	No / DA
V2.4.3	Verify that if PBKDF2 is used, the iteration count SHOULD be as large as verification server performance will allow, typically at least 100,000 iterations.	No / DA
V2.8.6	Verify physical single-factor OTP generator can be revoked in case of theft or other loss. Ensure that revocation is immediately effective across logged-in sessions, regardless of location.	No / DA
V2.8.2	Verify that symmetric keys used to verify submitted OTPs are highly protected, such as by using a hardware security module or secure operating system-based key storage.	No / DA
V6.4.1	Verify that a secrets management solution such as a key vault is used to securely create, store, control access to and destroy secrets.	No / DA

Table 4.7: Summary of ASVS Level 2 requirements that OutSystems does not fill.

5 Conclusion

Low-code development platforms are estimated to become mainstream in software development in the following years. This thesis was a compliance analysis into OutSystems' security features as well as finding out how to conduct a security assessment with an LCDP. The results from this thesis show that even though OutSystems has some deficits with ASVS requirements in levels 1 and 2, it is mostly compliant with it. The methods used in this compliance analysis can be used in the future when assessing an LCDP's security features.

Along with the compliance analysis, we did a literature review on the field of low-code. In our academic literature review, we focused on the whole field of low-code, and in our industrial review, we focused on the security of low-code. Our results show that while research on LCDPs is gaining popularity, the industry is leading the way in researching the topic.

5.1 Limitations

The biggest limiter for this thesis is the theme, looking at the security of low-code from a single platform's point of view. Low code and security as a theme does not have many previous studies, but as a scope, it is big. A dissertation would be more appropriate for this topic rather than a Master's thesis. This topic is also limited in a sense that this is the first study about assessing an LCDP's security and there are no existing cybersecurity standards.

The final limitation of this thesis is that we conduct the assessment against ASVS Levels 1 and 2. Assessing OutSystems' compliance with ASVS Level 3 requires more resources than were now available.

5.2 Future work

As mentioned in the previous section low-code and security is an unresearched topic. Future works could include reviewing multiple LCDP's security features and comparing them. This kind of research could tell us what security features LCDPs typically offer. We can use that information to define the shared responsibility between the platform and the developer. Because LCDPs have not yet seen an explosion in their popularity this kind of research could be in order. When transitioning to low-code becomes mainstream many organizations might begin to ask what security features belong to them, and what features the platform should provide. Having a clear answer could help the transition and choose the right platform.

Shared responsibility in security is not the only thing that has room for future work. Our literature review shows that of the eight categories we identified, five had three or fewer research papers. Not only is there lots of room for future research, but even our literature review could have been more comprehensive from the start. A topic that closely relates to low-code is no-code, and we could have used the keywords "no-code" and "no code" to conduct our database searches.

With this thesis, we also saw that the traditional security assessments might not always work when working with LCDPs. There is not only a need to come up with shared responsibility but also a need for a security assessment or security standard designed for LCDPs. This thesis also found the status of citizen developers' to be somewhat debatable. LCDPs have varying support for them

In their security portal [53] OutSystems claims to be compliant with HIPAA and PCI DSS. These are cybersecurity standards for critical infrastructure and would

suggest that OutSystems might also be compliant against ASVS Level 3. However since ASVS Level 3 was not in the scope of this thesis, future work should also include testing OutSystems compliance against that.

5.3 Summary

We answered RQ1 in chapter 3, and in chapter 4 we answered RQ2 and RQ3. Let us recap the research questions and our findings for them.

RQ1: WHAT IS THE CURRENT STATE OF THE FIELD OF LCDP?

While LCDP as a research field has not been popular topic to research, it has been gaining some attraction in the resent years. Table 3.1. shows how academic research is currently divided. We can see that security is the most under-researched topic and that there is room for multitopic research. We also see that academic research has previously not studied how to conduct a security assessment on LCDP.

As for the industry. Currently it appears the industry is leading the way for LCDPs' security. It seems that rather than an industry, we should be talking about individuals and individual companies leading the way. Individual companies such as Zenity might play a big role in the future in defining the LCDPs' security. They are already sponsoring the second major industrial material we found, OWASP Top 10 Low-Code/No-Code Security Risks. The industry studies also did not show that there is a previous case study on how to conduct a security assessment on LCDP. We truly see that this thesis is a pioneer when it comes to that.

RQ2: IS OUTSYSTEMS COMPLIANT WITH ASVS?

For the most part, OutSystems is compliant with ASVS. Looking at tables 4.6 and 4.7, we see that seven out of the 13 requirements that OutSystems did not fulfill are related to category 2, authentication. The single biggest issue that OutSystems has with the ASVS relates to the use of MFA. Nowadays, MFA plays a big part in application security, and while OutSystems Forge does have modules that would

make it fulfill these requirements, they are not official modules. OutSystems would need to also do this with another module for scanning antiviruses. The rest of the failing requirements would require OutSystems to provide a key vault to store secrets.

RQ3: HOW TO ASSESS LCDP SECURITY IN FUTURE?

Assessing an LCDP's security is not easy. Currently, there are no standards that advise you on how it should be done. We do not have a clear answer on how to assess an LCDP's security, but we believe that one crucial element is transparency. Part of the transparency is lost with the pre-made features, and the LCDPs have to compensate for that. In OutSystems' case, we have utilized their security portal to see the security reports done by 3rd party vendors, and to see their documentation on their ways of work. We found this to be working quite well.

Another crucial element in assessing an LCDP's security is to define a clear separation of responsibility. The responsibility needs to be defined so that the evaluation can take place. In an ideal case, the platform would have all the security features pre-made, but as our compliance analysis shows, we are not yet there. In our compliance analysis, we used this division to see if the platform met the requirement and if it did not, we looked if the developer could do something to meet the requirement. This method seemed to work and it was easy to implement.

References

[1] J. S. Geoffrey Elliott, *Global business information technology: an integrated systems approach*. Pearson Education, 2004, p. 87.

[2] B. W. Boehm, "A spiral model of software development and enhancement", *Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[3] Atlassian. "Scrum - what is it, how it works, and why it's awesome". (2022), [Online]. Available: `https://www.atlassian.com/agile/scrum` (visited on 06/05/2022).

[4] P. Vincent, Y. Natis, K. Iijima, J. Wong, S. Ray, A. Jain, and A. Leow, "Gartner magic quadrant for enterprise low-code application platforms", *Gartner Research*, 2019.

[5] OWASP. "OWASP | Open Source Foundation for Application Security". (2022), [Online]. Available: `https://owasp.org/` (visited on 09/30/2022).

[6] OWASP. "OWASP Top Ten Web Application Security Risks". (2022), [Online]. Available: `https://owasp.org/www-project-top-ten/` (visited on 09/30/2022).

[7] OWASP. "OWASP ZAP". (2022), [Online]. Available: `https://www.zaproxy.org/` (visited on 09/30/2022).

[8] OWASP. "OWASP Application Security Verification Standard". (2022), [Online]. Available: https://owasp.org/www-project-application-security-verification-standard/ (visited on 09/30/2022).

[9] V. Garousi, M. Felderer, and M. V. Mäntylä, "The need for multivocal literature reviews in software engineering: complementing systematic literature reviews with grey literature", in *Proceedings of the 20th international conference on evaluation and assessment in software engineering*, Limerick, Ireland, 2016, pp. 1–6.

[10] R. Sanchis, Ó. García-Perales, F. Fraile, and R. Poler, "Low-code as enabler of digital transformation in manufacturing industry", *Applied Sciences*, vol. 10, no. 1, p. 12, 2020.

[11] N. Prinz, C. Rentrop, and M. Huber, "Low-code development platforms–a literature review", Virtual Conference: AMCIS 2021, 2021.

[12] ISO/IEC. "ISO/IEC 27001:2013, Information technology — Security techniques — Information security management systems — Requirements". (2013), [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:27001:ed-2:v1:en (visited on 12/07/2021).

[13] ISO/IEC, "ISO/IEC 27002:2013, Information technology — Security techniques — Code of practice for information security controls", 2013. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:27002:ed-2:v1:en.

[14] ISO/IEC, "ISO/IEC 27011:2016, Information technology — Security techniques — Code of practice for Information security controls based on ISO/IEC 27002 for telecommunications organizations", 2021. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:27011:ed-2:v1:en.

[15] ISO/IEC, "ISO/IEC 27017:2015, Information technology — Security techniques — Code of practice for information security controls based on ISO/IEC 27002 for cloud services", 2021. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:27017:ed-1:v1:en.

[16] IT Governance USA Inc. "Typical ISO 27001 Certification Costs". (2021), [Online]. Available: https://www.itgovernanceusa.com/iso27001%20-certification-costs (visited on 09/30/2022).

[17] OutSystems, "What is low-code? [2021 update]", 2021. [Online]. Available: https://www.outsystems.com/blog/posts/what-is-low-code/.

[18] Microsoft. "What is Microsoft Power Platform?" (2021), [Online]. Available: https://docs.microsoft.com/en-us/learn/modules/introduction-power-platform/2-what-is-power-platform (visited on 11/30/2021).

[19] Microsoft. "Data connectors". (2021), [Online]. Available: https://docs.microsoft.com/en-us/learn/modules/introduction-power-platform/3-data-connectors (visited on 11/30/2021).

[20] Microsoft. "Security in Microsoft Dataverse". (2021), [Online]. Available: https://docs.microsoft.com/en-us/power-platform/admin/wp-security (visited on 11/30/2021).

[21] Microsoft. "Data loss prevention policies". (2021), [Online]. Available: https://docs.microsoft.com/en-us/power-platform/admin/wp-data-loss-prevention (visited on 11/30/2021).

[22] OutSystems, "OutSystems Platform Services", 2021. [Online]. Available: https://www.outsystems.com/evaluation-guide/platform-services/.

[23] OutSystems, "Supported stack configurations", 2021. [Online]. Available: https://www.outsystems.com/evaluation-guide/platform-runtime/#3.

[24] OutSystems. "Architecture | OutSystems". (2021), [Online]. Available: https://www.outsystems.com/enterprise-apaas/ (visited on 01/15/2022).

[25] OutSystems. "Does the OutSystems platform use AI and machine learning?" (2021), [Online]. Available: https://www.outsystems.com/evaluation-guide/does-OutSystems-platform-use-ai-machine-learning/ (visited on 12/05/2021).

[26] OutSystems. "OutSystems Security Overview ". (2021), [Online]. Available: https://www.outsystems.com/evaluation-guide/OutSystems-security-overview/ (visited on 01/11/2022).

[27] OutSystems. "ISO 27001 Certification Guide: What You Need to Know". (2022), [Online]. Available: https://www.itgovernance.co.uk/iso27001-certification (visited on 09/30/2022).

[28] "Cybermeter". (2021), [Online]. Available: https://www.kyberturvallisuus%20keskus.fi/en/our-services/situation-awareness-and-network-management/kybermittari-cybermeter (visited on 10/27/2022).

[29] M. A. A. Alamin, S. Malakar, G. Uddin, S. Afroz, T. B. Haider, and A. Iqbal, "An Empirical Study of Developer Discussions on Low-Code Software Development Challenges", *arXiv preprint arXiv:2103.11429*, 2021.

[30] Y. Luo, P. Liang, C. Wang, M. Shahin, and J. Zhan, "Characteristics and Challenges of Low-Code Development: The Practitioners' Perspective", in *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Bari Italy, 2021, pp. 1–11.

[31] C. Silva, J. Vieira, J. C. Campos, R. Couto, and A. N. Ribeiro, "Development and validation of a descriptive cognitive model for predicting usability issues in a low-code development platform", *Human Factors*, vol. 63, no. 6, pp. 1012–1032, 2021.

[32] F. Khorram, J.-M. Mottu, and G. Sunyé, "Challenges & opportunities in low-code testing", in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, Virtual Event Canada, 2020, pp. 1–10.

[33] B. Horváth, Á. Horváth, and M. Wimmer, "Towards the next generation of reactive model transformations on low-code platforms: three research lines", in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, Virtual Event Canada, 2020, pp. 1–10.

[34] S. Jahanbin, D. Kolovos, and S. Gerasimou, "Intelligent run-time partitioning of low-code system models", in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, Virtual Event Canada, 2020, pp. 1–5.

[35] J. Philippe, H. Coullon, M. Tisi, and G. Sunyé, "Towards transparent combination of model management execution strategies for low-code development platforms", in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, Virtual Event Canada, 2020, pp. 1–10.

[36] M. Bexiga, S. Garbatov, and J. C. Seco, "Closing the gap between designers and developers in a low code ecosystem", in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, Virtual Event Canada, 2020, pp. 1–10.

[37] L. Almonte, I. Cantador, E. Guerra, and J. de Lara, "Towards automating the construction of recommender systems for low-code development platforms", in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven*

Engineering Languages and Systems: Companion Proceedings, Virtual Event Canada, 2020, pp. 1–10.

[38] C. Di Sipio, D. Di Ruscio, and P. T. Nguyen, "Democratizing the development of recommender systems by means of low-code platforms", in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, Virtual Event Canada, 2020, pp. 1–9.

[39] G. Daniel, J. Cabot, L. Deruelle, and M. Derras, "Xatkit: a multimodal low-code chatbot development framework", *IEEE Access*, vol. 8, pp. 15 332–15 346, 2020.

[40] R. Arora, N. Ghosh, and T. Mondal, "Sagitec Software Studio (S3)-A Low Code Application Development Platform", in *2020 International Conference on Industry 4.0 Technology (I4Tech)*, IEEE, Pune India, 2020, pp. 13–17.

[41] C. Zolotas, K. C. Chatzidimitriou, and A. L. Symeonidis, "RESTsec: a low-code platform for generating secure by design enterprise services", *Enterprise Information Systems*, vol. 12, no. 8-9, pp. 1007–1033, 2018.

[42] A. Jacinto, M. Lourenço, and C. Ferreira, "Test mocks for low-code applications built with OutSystems", in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, Virtual Event Canada, 2020, pp. 1–5.

[43] R. Martins, F. Caldeira, F. Sá, M. Abbasi, and P. Martins, "An overview on how to develop a low-code application using OutSystems", in *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*, IEEE, Bengaluru India, 2020, pp. 395–401.

[44] H. Lourenço and R. Eugénio, "TrueChange (TM) Under the Hood: How We Check the Consistency of Large Models (Almost) Instantly", in *2019 ACM*

IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, Munich Germany, 2019, pp. 362–369.

[45] H. Lourenço, J. Tavares, R. Eugénio, M. Lourenço, and T. Simões, "LUV is not the answer: continuous delivery of a model driven development platform", in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, Virtual Event Canada, 2020, pp. 1–10.

[46] J. P. Fernandes, R. Araújo, and M. Zenha-Rela, "Achieving Scalability in Project Based Learning through a Low-Code platform", in *Proceedings of the 34th Brazilian Symposium on Software Engineering*, Natal Brazil, 2020, pp. 710–719.

[47] A. Sahay, A. Indamutsa, D. Di Ruscio, and A. Pierantonio, "Supporting the understanding and comparison of low-code development platforms", in *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, IEEE, Portorož Slovenia, 2020, pp. 171–178.

[48] M. Bargury, "The 7 deadly sins of low-code security and how to avoid them", 2021. [Online]. Available: `https://www.zenity.io/blog/low-code-security-risks-7-sins-and-how-to-overcome-every-single-one/`.

[49] OWASP. "OWASP Top 10 Low-Code/No-Code Security Risks". (2022), [Online]. Available: `https://owasp.org/www-project-top-10-low-code-no-code-security-risks/` (visited on 02/05/2022).

[50] B. Doerrfeld. "How to mitigate low-code security risks". (2021), [Online]. Available: `https://devops.com/how-to-mitigate-low-code-security-risks//` (visited on 11/29/2021).

[51] IT Pro Today. "App development: Staying secure using low-code platforms". (2021), [Online]. Available: https://www.itprotoday.com/application-security/app-development-staying-secure-using-low-code-platforms (visited on 12/03/2021).

[52] Appknox. "Top 7 security risks of a low code development for your enterprise". (2021), [Online]. Available: https://www.appknox.com/blog/top-security-risks-of-low-code-development (visited on 12/03/2021).

[53] OutSystems. "Security Portal". (2022), [Online]. Available: https://security.outsystems.com/ (visited on 09/30/2022).

[54] OutSystems. "Application security - Technical white paper". (2022), (visited on 01/11/2022).

[55] OutSystems. "OutSystems Forge - Password Strength Meter". (2022), [Online]. Available: https://www.outsystems.com/forge/component-overview/1131/password-strength-meter (visited on 05/15/2022).

[56] OutSystems. "OutSystems Forge - Toggle Show Password". (2022), [Online]. Available: https://www.outsystems.com/forge/component-overview/2627/toggle-show-password (visited on 05/15/2022).

[57] OutSystems. "About OutSystems". (2022), [Online]. Available: https://www.outsystems.com/company/ (visited on 09/30/2022).

[58] OutSystems. "OutSystems Forge - SAML Platform Authentication". (2022), [Online]. Available: https://www.outsystems.com/forge/component-overview/4311/saml-platform-authentication (visited on 05/15/2022).

[59] OutSystems. "OutSystems Forge - VirusTotal API". (2022), [Online]. Available: https://www.outsystems.com/forge/component-overview/4738/virustotal-api (visited on 05/15/2022).