Document Generation Security Assessment - A DocOrigin Case Study

University of Turku
Department of Computing
Master of Science (Tech) Thesis
Cyber Security
April 2026
Leo Tamminen

Supervisors:
Tahir Mohammad Univ. of Turku
Ismayil Hasanov Univ. of Turku
Jukka Brunfeldt ProDocsFormat

UNIVERSITY OF TURKU
Department of Computing

Leo Tamminen: Document Generation Security Assessment - A DocOrigin Case Study

Master of Science (Tech) Thesis, 102 p.
Cyber Security
April 2026

Enterprise document generation systems are widely used in modern organizations to automatically generate large volumes of business-critical documents such as invoices, payslips, and official communications. These systems process sensitive data and are often deeply integrated into enterprise backend environments, making their security a critical concern. Despite their importance, document production platforms are frequently treated as supporting systems, and their architecture-level security aspects may receive limited attention.

This thesis evaluates the security of a real-world ERP-to-DocOrigin document production pipeline using two industry-recognized standards: OWASP Application Security Verification Standard (ASVS) v5.0.0 and the AS&D Security Technical Implementation Guide (STIG) Version 6, Release 4.

The evaluation combines a systematic literature review of document pipeline security risks with a practical case study based on a production environment running the DocOrigin document generation platform. The application was evaluated against the OWASP ASVS at the application level and against AS&D STIG at the environment level.

The results show that DocOrigin operates as a component-style engine that delegates security entirely to its surrounding environment, a pattern termed the dumb engine hypothesis in this study. Proven vulnerabilities include input validation failures, a successful Billion Laughs denial-of-service attack, plain-text credential storage, and a total absence of drive encryption in the reference environment. Nineteen out of twenty-three high severity STIG findings were confirmed as active. The findings are applicable to organizations deploying similar document generation systems, particularly in regulated or defense-sector contexts.

Keywords: OWASP ASVS, STIG, cyber security, document generation, document pipeline security, ERP integration, CCM, penetration testing, security hardening, DocOrigin

Contents

List of acronyms

ASVS Application Security Verification Standard

BPMN Business Process Model and Notation

CCM Customer Communication Management

CRM Customer Relationship Management

CVE Common Vulnerabilities and Exposures

DFD Data Flow Diagram

DISA Defense Information Systems Agency

DoS Denial of Service

EEH Exaggerated Error Handling

ERP Enterprise Resource Planning

GUID Globally Unique Identifier

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

LLM Large Language Model

MFA Multi-Factor Authentication

NIST National Institute of Standards and Technology

NPM Node Package Manager

OTP One-Time Password

OWASP Open Worldwide Application Security Project

PII Personally Identifiable Information

SBOM Software Bill of Materials

SLR Systematic Literature Review

SSTI Server-Side Template Injection

STIG Security Technical Implementation Guide

TEE Trusted Execution Environments

TLS Transport Layer Security

TOCTOU Time-of-Check to Time-of-Use

XML Extensible Markup Language

XSS Cross-Site Scripting

XXE XML eXternal Entity

1 Introduction

Enterprise document production systems are used to automatically create, manage, and distribute business-critical documents as part of everyday organizational processes. As Nurmeksela states [1], document production in enterprise environments is closely connected to enterprise content management practices and the overall information infrastructure of an organization. Documents such as invoices, payslips, shipping labels, and order confirmations are generated automatically from data stored in core business systems, with minimal or no human intervention. Most modern enterprises rely on ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), or a combination of these systems to manage their core operations. A typical document production workflow begins with data from one of these systems, which is then transferred to a document production environment. There, specialized software processes the input, applies templates and business logic, and produces the final document output. The result is then distributed through channels such as print services, email, or digital customer portals.

From a cybersecurity perspective, these document production environments are complex pipelines with multiple connected components. Because they rely on integrations, configurations, and data transfers between components, security risks arise when the architecture is not carefully designed or when security controls are not correctly implemented.

1.1 Research Problem

Many enterprises rely entirely on their ERP systems for the majority of their operations. Document production and security controls are also handled within a single integrated platform. In these cases, document security-related risks are handled and largely managed as part of the ERP ecosystem.

However, there are industrial-level needs for document generation that require high-volume output, flexible template design, or advanced automation, which cannot be fully achieved within an ERP system alone. Organizations with more complex needs often adopt an external document generation platform alongside their existing ERP, either as part of a best-of-breed approach or to extend a system that cannot meet their requirements on its own. These organizations use platforms such as OpenText Exstream, DocOrigin, or older legacy systems, such as JetForm or Adobe Central.

The need for Customer Communication Management (CCM) systems is larger than it may seem. Major manufacturing and logistics companies depend heavily on automated document production for shipment labels, stock lists, transport documents, production tags, and other outputs that are critical for supply chains and regulatory compliance. A practical example is physical packaging labels used across many industries worldwide, steel industry being one of them [2].

When document generation is separated from the core ERP system, a distinct document production pipeline is formed. Inside this pipeline, data is transferred, processed, transformed, and distributed across multiple components and environments. This introduces security risks that are specific to the pipeline itself, independent of the ERP system.

There is a limited amount of research addressing these specific security risks. This thesis focuses on DocOrigin, a professional document generation engine, and evaluates its security posture in a real-production environment. The ERP system is

treated as a black box and is out of scope. The focus is on the DocOrigin environment itself: its data flow, configuration, access controls, and operations. The research questions are as follows:

1. **RQ1:** What security risks are present in the ERP-to-DocOrigin document production pipeline, and at which stages do they occur?
2. **RQ2:** How does DocOrigin perform against OWASP ASVS v5.0.0 and AS&D STIG v6r4 when evaluated as a production document generation system?
3. **RQ3:** What do the evaluation findings indicate about the security posture of DocOrigin-based architectures, and what are the implications for organizations using similar systems?

This study was commissioned by ProDocsFormat Oy.[1]

1.2 Research Objectives and Structure

The objectives of this research are as follows:

1. Analyze the ERP-to-DocOrigin document production pipeline and identify at which stages security risks can occur *(addresses RQ1)*
2. Identify security weaknesses in the DocOrigin environment, covering the application itself, its data flow, configuration, and access controls *(addresses RQ1 and RQ2)*
3. Evaluate DocOrigin using two industry-recognized security standards, OWASP ASVS v5.0.0 and AS&D STIG v6r4 *(addresses RQ2)*
4. Summarize the evaluation findings and assess the overall security posture of DocOrigin-based architectures *(addresses RQ2 and RQ3)*

[1]ProDocsFormat Oy URL: https://prodocsformat.fi/

5. Discuss the implications for organizations using similar document generation systems *(addresses RQ3)*

In this thesis, the security of the ERP-to-DocOrigin document production pipeline is evaluated using a systematic approach that combines a theoretical background, literature review, and a practical case study. The evaluation is relevant to organizations that operate in regulated environments, including the public and defense sectors, that rely on external document generation engines without vendor-provided security documentation. The findings reveal that significant security risks exist at both the application and environment levels, and that a structured evaluation using established standards is practical and necessary for this type of system.

The remainder of the thesis is organized as follows. Chapter 2 describes the research methodology and introduces the two evaluation standards used in the study. Chapter 3 sets the theoretical background by analyzing security risks in document generation pipelines in four areas: an overview of the field (Section 3.1), system architecture (Section 3.2), data flow (Section 3.3), and technical attack vectors (Section 3.4). Chapter 4 presents the case study, introducing DocOrigin and evaluating it against OWASP ASVS v5.0.0 at the application level and AS&D STIG v6r4 at the environment level. Chapter 5 discusses the findings, answers the research questions, evaluates the suitability of the standards, and concludes the thesis.

1.3 Use of generative AI

Large Language Model (LLM), specifically Claude by Anthropic (claude-sonnet-4-6), was used in the writing process of this thesis to improve language, fix grammatical errors, and enhance textual coherence. The same tool was also used to generate command-line and PowerShell commands during the reference environment evaluation in the case study as an efficient alternative to manual documentation searches

when verifying specific system states and configurations. All research, analysis, findings, penetration testing, evaluation decisions, and conclusions are the author's own work.

2 Research Methodology and Background

This study employs a Systematic Literature Review (SLR) as its primary research method, as an SLR is a structured and repeatable process that does not rely on personal judgment when selecting sources [3]. Direct academic databases were used to retrieve the most relevant sources. Table 2.1 lists all search queries used, the number of results screened, and the number of sources ultimately selected.

Because no studies have examined DocOrigin security directly, the search was divided into three categories: industry-level security, data flow integration, and technical attack vectors. The goal was to build a current and comprehensive overview of document generation security as a field, given that there is a limited amount of dedicated research on this specific, rather *niche*, but important topic. The following search queries were used:

- Q1 (Domain & Security) This query focuses on general cyber security and the intersection with document management systems. The term "document producti*" was excluded after testing, as it introduced legal and technical noise without adding relevant depth. Therefore the Q1 is formed as follows: **("document generation" OR "document output management" OR "document composition") AND (security OR vulnerabilit* OR threat* OR "risk assessment")**

- Q2 (Data Flow & Integration) This query is used to identify the risks specific to ERP-to-document output: **("data flow" OR "data pipeline" OR "integration" OR "middleware") AND (ERP OR "enterprise system" OR "BPM") AND ("document generation" OR "report generation") AND (security OR integrity)**

- Q3 (Attack Vectors) This query is used to search for document generation systems vulnerabilities that are relevant: **("document generation" OR "document composition" OR "report generation") AND (security OR attack OR vulnerability) AND (injection OR "input validation" OR "parsing" OR "attack vector" OR "OWASP ASVS" OR "STIG" OR "hardening")**

It is important to note that the search queries are not perfect. They are fine-tuned to retrieve relevant studies across the selected scientific databases and could have been refined further, but the priority was to keep the review repeatable as a systematic literature review.

The selected scientific databases are listed in Table 2.1. For arXiv, a full-text search was used because it was the only option that returned results. This introduces more noise and required additional screening, but as the table shows, the initial hit counts for arXiv remained manageable. For Google Scholar, only the first 100 most relevant results were screened for each query, given the scale of the total results.

2.1 Inclusion and Exclusion Criteria

The review applies carefully selected inclusion and exclusion criteria. The publication year range was set to 2015–2026 to ensure coverage of the most recent developments in document generation cybersecurity. A small number of sources published before this range are cited where they represent foundational standards or reference

Table 2.1: Systematic Literature Search Strategy

Database	**Query**	**Initial Hits**	**Screened**	**Selected†**
IEEE Xplore	Q1	18	18	7
	Q2	1	1	1
	Q3	6	6	2
ACM Digital	Q1	193	193	11
	Q2	154	154	3
	Q3	464	203	9
SpringerLink	Q1	292	292	4
	Q2	160	160	8
	Q3	421	421	15
WoS	Q1	13	13	2
	Q2	56	56	8
	Q3	75	75	7
arXiv	Q1	200*	200	12
	Q2	58*	58	8
	Q3	131*	131	2
Google Scholar	Q1	3590	100	10
	Q2	2440	100	12
	Q3	5560	100	4
Total		13,832	2281	125

* = Full Text Search

† = Candidates passing all screening rounds; not all are cited in the final thesis

models that remain authoritative, such as established architectural frameworks and methodology guidelines [3], [4].

During initial screening, sources that appeared relevant by keyword but missed the intended scope were excluded. Examples include blockchain-focused or heavily cryptographic studies, document authenticity verification research, and STIX document generation, which concern cyber threat reporting rather than document production security.

The "Initial Hits" column in Table 2.1 reflects the number of results for each query as of February 16, 2026, with a publication year filter from 2015 onwards applied. Language and content-type filters were not applied because non-English and non-academic results were negligible in practice. Therefore, the only active filters were the time range and sorting by relevance.

The search queries were kept consistent across all databases. Minor syntax adjustments were made where database-specific notation was required. After collecting the initial results, the screening process began in the following rounds:

- For databases with only tens of initial hits, every source was reviewed at the abstract level.
- For databases with hundreds of initial hits, the titles were reviewed first, then followed by abstracts of the relevant papers.
- For Google Scholar specifically, with over thousands of results in every case, the titles of the first 100 most relevant results were reviewed. Then the abstracts of the passing ones.

At this screening level, the focus was on breadth rather than depth, with a maximum abstract-level reading. The goal was to reduce all three query pools to a manageable size before conducting a deeper analysis.

The sources that passed the first round were recorded in a spreadsheet with the following fields: author(s), title, publication year, paper aim, source link, relevance to each query, linkages to the research topic, identified gaps, and a short identifier such as `mittal_offline_2025` for reference management.

Because the first round was intentionally broad, a second screening pass was conducted directly from the spreadsheet. A duplicate-detection procedure was applied to remove overlapping results across databases. After deduplication, each remaining source was examined to assess its relevance, contribution to the research questions, and the gaps it identified. The "Selected" column in Table 2.1 reflects only the sources that passed all screening rounds.

Topic-level exclusion was applied manually. Studies on blockchain technologies, geopolitics, medicine, or legal topics were excluded, even when keyword overlap existed. The first screening round was intentionally generous, and a number of

sources that appeared potentially relevant were excluded in the second round after closer reading confirmed that they fell outside the scope.

A note on Q3 database coverage: ACM results for Q3 included many full conference proceedings rather than individual articles, so only individual articles were screened. SpringerLink returned a cleaner dataset of 421 results, of which only 4 were full proceedings, meaning 417 individual articles were effectively screened. For consistency, the table records 421 as the screened count.

The review confirmed that dedicated research on document generation pipeline security is limited. Q1 produced sources on general enterprise system security and document management, which set the architectural principles and security concepts applicable to CCM systems. Q2 identified risks specific to ERP-to-document integration points, covering data flow integrity and middleware vulnerabilities. Q3 produced the technical foundation for the attack classes evaluated in the case study, including injection vulnerabilities, parser weaknesses, and configuration exposure. The selected sources are synthesized in Chapter 3, which organizes the findings into three areas directly inherited from the search queries: system architecture and security principles, data flow and integration integrity, and technical vulnerabilities and attack vectors. Together, Chapters 2 and 3 form the theoretical foundation for the case study in Chapter 4.

2.2 Selected Evaluation Standards and Their Justification

In highly regulated sectors, such as the public sector and defense industry, security cannot depend on informal checklists. Standardised frameworks provide consistent, verifiable baselines that save time and remove ambiguity, particularly for tools and systems that lack vendor-specific hardening guidance [5]. These sectors are critical

by nature, and compliance with recognised standards is essential for ensuring the safety and integrity of the systems they operate.

Measurable and verifiable security is a fundamental requirement in these contexts. International frameworks, including those mandated by NATO, establish this baseline. In this study, two major industry standards were used to evaluate the security of the document generation pipeline.

The first one is the OWASP Application Security Verification Standard (ASVS) Version 5.0.0, maintained by the Open Worldwide Application Security Project. ASVS provides a structured framework of security requirements for evaluating the security controls of modern applications. It is organised into levels of increasing rigour, making it suitable for assessing both basic and enterprise-grade systems [6].

The second is the Application Security and Development (AS&D) Security Technical Implementation Guide (STIG) provided by the Defense Information Systems Agency (DISA) for the United States Department of Defense. STIGs translate high-level security policies into specific actionable technical settings for securing information systems and software. The version used in this study is AS&D STIG Version 6, Release 4, published on November 26, 2025, on the DoD Cyber Exchange Public website operated by DISA [7].

Together, these two standards complement each other well for the purposes of this study. ASVS examines the security of the application itself, while the AS&D STIG evaluates the hardening of the environment in which it runs. Since no vendor-provided security documentation exists for DocOrigin, this combination provides a structured and justifiable basis for the evaluation.

3 Security Risks in Document Pipelines

3.1 Overview of Document Generation Security

Modern document generation is no longer a simple, stand-alone, single-application process. It is a distributed pipeline where the data that is being processed can originate from various different sources. The most typical source for this data is ERP or similar system that works similarly in terms of producing usable data for document generation. The data then is transmitted in this *data pipeline*, that includes various integration layers, validation and is finally rendered by specialized engines.

One of the most critical security risks in document generation pipelines is the leakage of sensitive data outside of the systems. As Mittal et al. [8] show, one way to manage this risk is to take the generation process and move it to offline environment. For example, local configuration JSON (JavaScript Object Notation) or XML (Extensible Markup Language) templates can be used (or any other configuration method). It is crucial to understand that an offline environment and a completely air-gapped or logically separated segment from the ERP are different concepts. The document generation engine (CCM) is rarely truly air-gapped in enterprise production use. Typically, such an engine or document-generating software is logically separated from the actual ERP but can not be fully offline because it constantly has

to receive critical data that is then used to produce the actual documents. Moreover, the segment is often remotely controlled, monitored and configured, therefore it requires very good security practices but simply can not be air-gapped for convenience and business critical reasons.

Securing the generation engine is not enough without properly securing the input data. In enterprise-level systems, automatic validation is necessary. In a typical CCM production pipeline (I.E. DocOrigin) this can mean that the data originating from the ERP has to always be validated before it reaches the generation engine. This way any injection attacks or malicious data can be stopped [9].

Also, with all these in place, the integrity of the document lifecycle itself actually poses a risk. Balakayeva et al. demonstrated that all stages of the document lifecycle must be systematically evaluated to detect and prevent errors that could compromise the process. This includes every stage from data collection, pre-transmission, transmission, storage, and delivery. They argue that security risks are not limited to just the data source or the final output. Vulnerabilities often appear in the "middle" stages, such as when data is being transmitted or processed. This supports the idea that the entire document pipeline must be well secured in order for the final output to remain accurate and secure [9].

Managing this complex lifecycle can be difficult and expensive, especially in highly regulated industries such as the pharmaceutical industry, where data integrity is vital. Chen and O'Connell [10] point out that while smart document automation can help follow highly regulated industry rules, it requires ongoing monitoring when ever the data is moving. Similarly, Hermawan et al. [11] state that although automated reports in ERP systems makes work more efficient but at the same time it also create new challenges in keeping the data intact and correct at integration points.

One way to approach these challenges is presented by Sadique et al. [12] where

they show protecting data early in the generation stage is crucial. This is because early-stage obfuscation or encryption can minimize the attack surface of the pipeline. If data is compromised in the middle layers, it remains unusable to an attacker without the corresponding decryption keys. This principle of early protection complements the use of isolated environments by ensuring that there are no *blind spots* where plain text data could been exposed to tampering before the final document is generated [8].

3.2 System Architecture and Security Principles

The security of a document production pipeline depends largely on how the system is built. Real-world deployments confirm this, as Ambati [13] demonstrated in a case study of on-demand financial document generation integrated with an ERP system, where multilayered security controls were required to protect sensitive payment document output. In many organizations ERP systems (SAP, MS D365 etc.) perform as the main hub for all or majority of their processes and business data. Companies prefer all-in-one solutions because they are convenient and often the most reasonable for most needs. However, like discussed earlier, this luxury is not available for every business, some require more. The standard ERP tools can lack the needed flexibility, automation, maintaining, configuration etc. needed for high-volume output automation. To address this, the businesses simply solve it by using another layer, integrated external CCM systems such as DocOrigin in their infrastructure.

To understand this properly, it is important to consider current industry standards for system architecture. Some type of reference model could help explain the situation. For this specific case, there is no out-of-the-box reference model that fits perfectly, but reference models, after all, are used as an abstract frameworks to understand relationships in an environment [4]. Therefore, a remarkable reference model that could be used, would be the OSI model originally designed for network

communication. Although the OSI model was not designed for enterprise application architecture, its core principles are directly applicable here. In the OSI model, data moving between systems passes through distinct, separate layers, and each layer has its own responsibility. In an ERP-to-CCM pipeline, the same logic applies, data comes from the ERP system and passes through integration and transformation layers until it is finally processed by the document generation engine. Each of these layer boundaries is a potential trust boundary and, therefore, a potential attack surface.

3.2.1 Service Isolation and Decoupled Architecture

Adding an external CCM layer gives organizations the flexibility they need, but it also expands the attack surface [14], [15]. According to Yadav [16] even though hybrid cloud integration is a strategic way to modernize ERP systems, it also introduces new regulatory risks and adds complexity to the integration across these distributed environments. Unlike in traditional *tight coupling*, where all tools are built directly into the main business logic infrastructure, a compromise in one component can spread quickly to the rest of the system. Lapatta also argues that tight coupling makes cloud or distributed systems more difficult to protect. If one part of a tightly coupled system is attacked, the risk can quickly spread from the document tools to the core ERP database. To address this, Lapatta recommends using the *Repository Pattern*. This is an architectural approach that separates data sources from the document generation engine, creating a *decoupled* system. In this model, the document engine does not access the ERP database directly, but through a middle layer that controls and filters the data flow [17].

Separation guaranties *change localization*. This simply means that if a security issue or a technical flaw or change occurs in the engine, it is less likely to affect the rest of the business environment [17]. For a CCM system like DocOrigin this is

crucial security principle. Because the document generation process is isolated, any possible attack will be more difficult to succeed. In this way, it is much more difficult for an attacker to use a vulnerability in the document engine to move laterally into the sensitive layers of the ERP system.

3.2.2 Software Dependencies and SBOM

Modern software typically uses external parts, such as libraries, for different tasks. Fucci [18], states that governments now often require companies to keep a Software Bill of Materials (SBOM). This serves as a list of ingredients that essentially states the software composition and all of its dependencies. It can be stated that because of the possibility of even using multiple libraries for a seemingly helpful reason, managing all the parts can become a massive task. As Mens describes [19], perhaps the worst scenario to end in is so-called *dependency hell*, a situation where a piece of software becomes *buggy* or insecure because of the many dependencies it uses start to conflicting with each other.

The dependency problem is growing because the number of available libraries is growing rapidly. For instance, in seven years, the number of the popular NPM (Node Package Manager) library, the number of packages increased more than ten times. A major part of the problem is known as *transitive dependencies*. These are essentially hidden layers, libraries that the libraries itself need in order to work properly. According to Mens, most software has only a few direct parts but many of these hidden transitive parts. It is clear that if even one of these has a vulnerability, the whole pipeline can be compromised. This is alarming because this *technical lag* can make the entire system critically fall behind on security because updating any small part might break the entire chain [19].

Simply listing the software components and following the regulations of the SBOM are surely not enough given the complexity of the modern software systems

and all the possible weak points. SBOMs must be augmented with up-to date vulnerability information, such as Common Vulnerabilities and Exposures (CVE) [18]. In a CCM environment like DocOrigin, this means that every library used in the integration and generation, all stages must be continuously monitored. Decoupled architecture, according to the repository pattern, could still have an unpatched vulnerability in the XML parser. Therefore, the combination of architectural isolation with automated software composition analysis is a fundamental principle for maintaining the integrity of the entire document production chain.

The practical way to follow these security rules is to treat document generation as an independent service. Sabbag Filho thoroughly explains that isolating document generation into its own dedicated service makes it much easier to apply uniform security and auditing policies. In this way, the *separation of concerns* is ensured. In other words, the layout design is completely separated from the main business data. By doing this, organizations can prevent rendering tools from needing direct access to sensitive databases [20].

The separation points (where data moves from one system or responsibility to another.) In a document pipeline, these are called trust boundaries because the security rules must be checked every time data crosses them. These boundaries can, for example, be an API gateway or even the document service itself; the system can control and monitor how information moves. Sabbag Filho recommends that the document service should only receive data that has already been normalized and validated. This prevents the generation engine from making risky calls back to the core database. Protecting these transition points is a key rule for keeping the system's data safe. Additionally, observability and tracing are essential at these boundaries to ensure that every generation request can be audited for security reasons [20].

3.3 Data Flow and Integration Integrity

To ensure the security of a document pipeline, it is necessary to understand how data moves in different types of data pipelines. There are many examples of data moving from one system to another across modern complex networks. Because an increasing number of systems have become digital, there are also numerous systems that co-exist, and these systems ultimately need to be able to communicate with each other seamlessly. There are different protocols that help communicating, such as the HTTP (Hypertext Transfer Protocol) mentioned by Chiș et al [21] as an example in a Data Flow Diagram (DFD).

Chiș et al [21] explains that a DFD is essential tool that are used to represent systems and data flow. It offers a clear way to see how information travels between different components and entities. Even though is has been never standardized to follow any specific notations, it can be stated that many of those emerged in recent years share the same concepts. For the diagram to be clear and effective, it needs to follow these guidelines based on four core concepts:

- **Processes**, These are the active parts of the system that take incoming data and change it into something else for a specific business reason.
- **Data stores**, These are the places where information is kept, acting as a library or a collection for the system.
- **External entities**, These are the participants outside the immediate process, for instance a user, an organization, or an ERP system that integrates with the data.
- **Data flows**, The are the arrows that show exactly how the data travels part of a system to another.

For the model to be accurate and useful for security analysis, guidelines must

be followed. A process can not just produce data without receiving any, and vice versa it can not receive data without producing a result. Furthermore, data can not simply jump from one storing space to another or move directly from an external entity to a storage center. There must always be a process in between for handling the movement. Every flow should be labeled clearly to describe what it is moving, arrows must be single direction and not bidirectional. Following these brief ruleset makes DFD a powerful tool for identifying where the data pipeline might or could be weak or open to attacks [21].

Decomposition is used when building DFDs. Chiș et al. [21] explain that the first step is to break the system down into its basic parts. These parts are then mapped to these four core concepts previously mentioned. For security purposes, a critical part of this step is identifying trust boundaries. These boundaries represent the borders between trusted and untrusted parts of the system. In a document pipeline, this means identifying the exact points where data moves from the ERP system into the processing layers.

However, building an effective DFD is not always simple and it comes with specific challenges. Chiș et al. [21] note that the value of a DFD depends heavily on its quality. If the model is incomplete or inaccurate, the security analysis will not identify all potential threats. DFDs fit better than BPMNs (Business Process Model and Notation) for security analysis, while BPMN is more suitable for business strategy related analysis [22]. Still, one of the main difficulties of DFDs are the fact that they are better at showing how data moves than identifying what might go wrong. Especially in complex environments where not all system details are fully known. For example, if the internal logic of an ERP system is not transparent, it can be difficult to map every data flow correctly. This makes it difficult to determine if a model is *good enough* for a professional security assessment. Because of these challenges, it is important to realize that the DFD is a starting point for discovering

threats rather than a final solution [21].

Once data flows and trust boundaries have been identified through DFD analysis, next step is to start planning and deciding how to actually protect those parts that are found to be the most relevant, weakest or open points for attacks. For this, for example in document production pipeline, one of the most vulnerable, or the critical points are actually those points where the data is in its most readable format, that is, plain text, for example. That means, the document production engine has to be one of those key points that has to see the data in plain text, since it produces the final document output. Mapping like this (DFD) will help understanding the path, it also reveals that certain parts of the pipeline require more than just standard software protection. If a process is located at a identified high-risk trust boundary, it needs a way to stay secured even if the surrounding environment could potentially be compromised.

Even though Lapatta's et al [17] presented decoupled isolates the services from one another, this can be taken even step further. Trusted Execution Environments (TEEs) are a hardware supported secure area that isolates computations from the rest of the system. [23] This addresses Lapatta's et al [17] issues or risks regarding that the admin-area is a critical point. With TEE-approach, not even OS admin or root-user, or a higher-privileged external software could technically even view the data-in-use.

It is crucial that in distributed architectures, protecting data only at the network or application level may not be enough if privileged system components are compromised. TEE technologies can vary a lot in how large the protected environment is. Application-level TEEs protect only a small part of an application's memory, examples of these include Intel SGX and ARM TrustZone. The other approach extend the protection boundary to an entire virtual machine. Technologies such as AMD SEV and Intel TDX isolate the memory and processor state of a full virtual

machine from the rest of the system [23].

3.4 Technical Vulnerabilities and Attack Vectors

A very significant yet often forgotten or overlooked vulnerability in decoupled architectures is the semantic gap between the data source (the ERP system) and the rendering engine (document generation engine in this case). Isolation appears to be the logical solution for protecting crucial systems. Still, this very separation is often not even possible to pull off properly, due to lack of resources or even knowledge, and on top of that, it can even lead to a scenario called *security vacuum* where the engine itself assumes that the data it receives has been already sanitized by the middle layer, and the middle layer assumes that the engine has these internal defenses in place. This creates a dumb engine scenario, where the software acts as a transparent processor of instructions, which makes it highly vulnerable to injection attacks if the trust boundary is breached [17], [23].

It is important to understand document generation more precisely to understand why this gap exists in the first place. At the level of data processing, document generation engine takes structured input data and produces a structured output document out of it. Transforming the internal representations into a formal language, Hermerschmidt et al. [24] call this process unparsing. They put it more formally, the process *(un)parsing round-trip*: the system is secure only if for every internal data representation m, the equation:

$$\text{parse}_{\text{decode}}\big(\text{unparse}_{\text{encode}}(m)\big) = m \tag{3.1}$$

holds, meaning that maliciously crafted input values cannot influence the structure or semantics of the output document. According to Hermerschmidt et al. injection vulnerabilities are not only a web problem. Essentially, every program that uses

input data to produce documents in a context-free language, such as XML or PDF, is potentially vulnerable to this attack. During the unparsing phase, when the engine embeds data values in template positions, the attack surface is larger. Without context-sensitive encoding applied at that exact point, a data value containing control tokens of the output language could potentially alter the document's structure. This essentially justifies why document generation engines are structurally open to injection regardless of how well the surrounding infrastructure is hardened. Fadlalla and Elshoush [25] confirm through a systematic review of input validation vulnerabilities that improper input handling are still one of the most widespread and severe vulnerability categories across all types of applications that process structured input data.

This theoretical risk could compound by the practical complexity of multiformat document parsing. Zhang et al. [26] show that for different types of documents, a system calls different parsers according to the document format identifier, and that mismatches between actual content and expected format cause parsing failures. From a security perspective, Zhang et al. demonstrated that attackers can embed malicious code into XML files and package them into well-formed document containers that are passed to the data processing document generation engine. This essentially means that the attack surface is far more extensive than just the data values themselves. A parser that prioritizes just passing the data through over strict validation accepts structurally ambiguous input that different processing layers may interpret differently.

XML Injection and External Entity Attacks

The most critical attack vector in any document generation pipeline that processes XML is XML injection, and especially the XML eXternal Entity (XXE) attack. Document generation systems are especially exposed because XML is often both

the input format and the primary structural language of the pipeline. Hermerschmidt et al. [24] show that the problem of correct context-sensitive encoding is not language specific, essentially according to them, every context-free language used for communication in distributed systems has the same injection risks, if the unparsing is implemented without explicit encoding rules. Their analysis of composed languages, where a sub-language is embedded within a super-language as in the case of a JavaScript function call embedded within an XML attribute, shows that the encoding must be applied in strict bottom-up order, first encoding the inner language, only then the outer. The resulting string into the outer language's token. If this safeguarding is not appropriately in place, it creates injection paths that go over language boundaries [24]. Essentially this is very critical, given the fact that document generation systems are by nature, vulnerable to injection attacks, and the fact that correct context-sensitive encoding is language-agnostic, not tied to any specific languages.

In the XML eXternal Entity (XXE) attack, attacker embeds a `DOCTYPE` declaration with a `SYSTEM` entity reference inside the XML input. If the parser resolves external entities without protections, it can be directed to read local files from the server.

Zhang et al. [26] confirm that the internal structure of XML-based document formats, specifically the use of XML files packaged within larger document containers, like the Microsoft Word's `.docx` format, creates a scenario where malicious code embedded in the XML layer goes to the document generation engine inside a structurally valid outer container. For a document generation pipeline where the engine processes XML data that is originally from an ERP system, a compromised or malicious data could cause a file exfiltration or server-side request forgery through the merge process itself.

Brodin et al. [27] show that PDF and XML parsers have frequently behavior that

is "forgiving" or in other words they might accept shortened or cut-off words that aren't actually allowed. Because different programs interpret these broken rules in different ways, this can end up the program viewing the same file differently. This allows an attacker to create a file that looks safe to one program but triggers a hidden, different action in another.

Resource Exhaustion and Denial of Service

Because document generation engines live by their input data, there need to be proper safeguards in place. resource exhaustion is dangerous for document generation engines because these attacks aim often is to make the engine unavailable. The most well-known variant is the Exponential Entity Expansion attack, commonly known as the Billion Laughs attack. It is a nested, limited amount of entity recursions. Essentially, this attack exploits recursive entity resolution in XML parsers by defining a chain of entities where each level references the previous one multiple times. When the parser resolves the final entity, it triggers exponential memory and CPU consumption. The attack requires only a very small input file, typically under 2 KB, to cause catastrophic resource consumption. [28] An example of this attack and a code snippet is shown in the Section 4.3, Figure 4.2.

Pakki et al. [29] provide a theoretical framework that explains why this attack type is so effective. They show that when a parser receives an input that is technically valid syntax of the specific language but is operationally far outside the expected resource use, the engine's error response determines whether the system fails safely or catastrophically. In their study of the Linux kernel, they found that 66 percent of identified Exaggerated Error Handling (EEH) bugs can crash the process, and they observed an average latent period of seven years before these bugs were even discovered. This means that exaggerated error responses can sit in production code for a very long time without being noticed. Their severity classification

maps directly onto document generation: a document engine that returns an error code and cleanly terminates a job is handling the error at an appropriate severity level. An engine that enters a non-responsive deadlock state is handling the error at a severity level that is higher than needed, which is exactly the EEH pattern. The deadlock variant prevents recovery and requires manual intervention, which in a production environment that generates time-sensitive documents such as invoices is a significant operational problem. According to Pakki et al. the most common cause of EEH bugs is incorrect reasoning about error severity. A total of 38 percent of cases, therefore not at all unusual case, more of a systematic pattern in systems that were not designed with error handling policy first in mind.

Template Injection and Scripting Risks

Document generation engines typically use template languages or scripting mechanisms to define how input data maps to output fields. This makes the Server-Side Template Injection (SSTI) possible. In SSTI, an attacker uses input data that contains template directives rather than plain data values. If the parser or engine interprets the input as a template expression rather than a literal string, the attacker may be able to execute arbitrary logic inside the generation process.

Blefari et al. [30] study SSTI as a primary attack vector and according to them, template-based systems are consistently exploitable when input data is passed to the template processing system with no sanitization done before to the data. SSTI is framed to more like a architectural risk and not a coding error. The vulnerability exists by design whenever a system uses dynamic input to apply and form the final output in cooperation with the existing template logic. concrete SSTI payload such as `{{__class__.__mro__[1].__subclasses__()}}` shows how input that references the internal object hierarchy of the template engine can expose or manipulate the runtime environment [30]. For a document engine that uses a JavaScript-based

scripting layer to control field processing logic, the same risk applies whenever data arriving from the ERP system can influence the execution path of that scripting layer.

A template language that allows arbitrary computation provides developers with maximum flexibility, but it also gives attackers a code execution path if input is not strictly separated from template logic, according to Ivanov et al. [31]. Their analysis shows that even well-designed template languages are difficult to secure against injection when the boundary between data and code is not made clear at every processing stage, especially when templates are generated dynamically from input-controlled values.

Yamazaki et al. [32] talk about Cross-Site Scripting (XSS) injection specifically in the context of HTML template generation and show that the structure of the output document breaks when injected content changes what the template was expecting. Their work groups injection risk based on where the untrusted data is placed inside of the actual document, for example inside a HTML body, an attribute value, or an event handler. Each of these require a different encode way. Browser-based XSS does not directly apply to batch document engines, but still the core problem is the same. When a template is expecting a plain text value but it gets a structured element or something executable the output document can end up malformed and potentially can have unintended content.

Hermerschmidt et al. [24] gets to the same conclusion from a more formal perspective, it then shows that the only reliable solution is basically a encoder that is based on the syntax rules of the output language. Such an encoder can automatically determine the correct encoding for every position in the document during the generation process, and it even removes the need for developers to manually check where each data value ends up.

Parser Blind Spots and Format Differentials

The blind spot problem is a very fundamental risk. It is essentially a deeper problem than just a single attack type [27]. A blind spot is essentially when a part of the input file can be changed freely without affecting what the parser produces as output, the parser just ignores this part completely. The definition is based on a data flow tracking, input byte is a blind spot if its value never affects, either directly or through any decision the program makes anything that the program outputs. The testing of Brodin et al. [27] of the MuPDF parser across over one thousand real PDF files found that on average at least five percent of each document was completely ignored by the parser. Some files had over one megabyte of entirely ignored data.

With document generation, this creates concrete problems. A potentially malicious payload can be hidden inside one of these blind spots of an input file. It can pass through validation without detection, (if there even is one in place) because the validation tool and the downstream processor may ignore different parts of the same file.

And on top of that, the lenient parsing behavior that creates blind spots, for example a PDF parser that accepts a shortened version of a `endobj` token where the full keyword is required, creates differences in behavior between the generation engine and any system that later processes the same output. Brodin et al. state that these kinds of differences have been exploited in practice in the past to create documents that try to display differently in different viewers. Fleury et al. [33] also show that PDF documents are an attractive attack vector because their rich feature set and widespread trust among users make them an effective carrier for malicious content that exploits parser inconsistencies.

This essentially shows that blind spots are a real problem, Zhang et al. [26] add to this from the document management side, that according to them, a mismatch between what a file actually contains and what its format label says causes parser

failures, and that attackers use exactly this kind of ambiguity to hide malicious content inside containers that look valid from the outside.

Prototype Pollution in JavaScript-Based Scripting

Document generation engines that use JavaScript for data filtering and for processing logic have a risk that is specific to how the language actually works. Prototype pollution, in JavaScript, is that every object automatically inherits properties from a shared global structure called the prototype chain.

Prototype pollution occurs when an attacker can insert or change properties on a shared structure, thereby causing objects in the application to inherit unexpected or harmful properties. This type of attack typically appears in code that stores values using a pattern like `obj[key] = value` without checking that if `key` is a "reserved" because they are part of the language's prototype inheritance such as pointers like `__proto__` or `constructor`. If the attacker can control what `key` contains, they can possibly modify the entire scripting environment.

Sachidananda et al. [34] showed with static analysis of software components that prototype-level vulnerabilities and code injection patterns are one of the most commonly found issues in systems that use dynamic scripting as part of their processing. Hardcoded credentials and prototype-level vulnerabilities appear regularly in systems where functional development was prioritized over security. For a document engine whose scripting layer handles field values coming from an external ERP system, prototype pollution is a risk in any place where an input-controlled value is used as a property name on a shared object without being sanitized first.

Path Traversal and Configuration File Exposure

Category risk specific to document generating engines that are heavily dependent on the filesystem is path traversal attack. A path traversal attack works like this:

the attacker manipulates a file path to reach files that are outside of the directory that the application was intended to access. In other words, the goal of the attacker is to gain access to other directories, possibly even attempting to take control of the entire host file system.

For a document engine that looks up template files, output directories, and configuration files based on values that come from the input data, any path component that is not properly cleaned up can direct the engine to read from or write to unintended locations on the server.

According to Witanto et al. [35] improper directory access controls are one of the most reliably exploitable weaknesses found in production deployments. Although they studied security misconfigurations in web applications, their findings apply directly to document generation environments, where configuration files that contain credentials and other secrets are stored inside the same directory structure as the rest of the application, without proper access restrictions at the filesystem level.

Moscosos et al. [36] show that even in regulated environments such as financial institutions, where security compliance requirements are very strict, production deployments regularly have security misconfigurations, like open TCP ports, unencrypted internal communications, and missing authentication on administrative connections.

Their case study found that a router used for testing remained improperly secured even after it had been scanned, and that open FTP ports exposed enough OS information for attacker to plan further attacks. Same applies directly to a document generation server, a deployment that works correctly and passes integration testing can at the same time expose directory structures, service accounts, and configuration files through misconfigured permissions, like a ports that should not be open, or missing access controls between the document engine and the rest of the organization's systems.

OS Command Injection via Web Service Interfaces

When a document generation engine exposes a web service interface that converts incoming HTTP requests into commands executed locally on the server, the interface becomes an entry point for OS command injection.

Alahmad et al. [37] survey OS command injection as one of the most severe vulnerability categories in the OWASP framework, noting that it occurs whenever user-controlled input is passed directly to the operating system shell without being sanitized.

The attack works in three steps: the attacker finds a vulnerable input point, adds a malicious command alongside the legitimate input using special shell characters, and the server runs the combined string using the same permissions as the application's service account. The result is that the attacker can run arbitrary commands on the server, which means they can access configuration files, steal output documents, change job queues, or even shut down the engine. In a document pipeline that produces business-critical or any sensitive data documents such as invoices, or regulated financial documents the consequences can be destructive.

Ahmad claims that applying OWASP ASVS Level 1 in a structured verification pipeline is very practical and effective for finding these mentioned risks before they reach production [38].

His framework uses findings all the way from static code analysis, dynamic testing, and dependency scanning to specific ASVS verification requirements, and demonstrates that Level 1 rules can actually detect the most critical injection vulnerabilities, like command injection, encoding failures, and configuration weaknesses and all this in a way that is repeatable and can be audited. The evaluation found that ASVS Level 1 covers approximately 40 percent of the verifiable security surface using checks that are observable from outside of the system, it essentially proves that a standard-based evaluation is very realistically achievable and also necessary

as a starting point for any document processing system.

4 Case study: DocOrigin

4.1 DocOrigin: System Overview

This chapter introduces DocOrigin software, dives deep to the integration with ERP (even though ERP is treated as a black box) takes into consideration all of the recognized security concepts that were discussed in the Chapter 3. The case study aims to do so with using AS&D STIG (Security Technical Implementation Guide) and OWASP (The Open Worldwide Application Security Project) ASVS v5.0.0, two (major) industry standards to evaluate the security of the pipeline. The first is the Application Security and Development (AS&D) STIG provided by the Defense Information Systems Agency (DISA). These are hardening guidelines that provide specific instructions for securing software and infrastructure [7]. The second is the OWASP Application Security Verification Standard version 5, a structured framework for testing the security controls of modern applications [6].

The specific guide used in this case study is the Application Security and Development (AS&D) STIG, Version 6, Release 4 (U_ASD_V6R4_STIG). The document was downloaded from the official DoD Cyber Exchange Public document library operated by the Defense Information Systems Agency (DISA) [39]. This document is designed to be used to bridge the gap between the National Institute of Standards and Technology (NIST) Special Publication 800-53 and the Risk Management Framework. It provides specific technical instructions for hardening software

applications when a vendor-specific guide is not available [7].

By using these established standards, it is guaranteed that the evaluation is based on proven industry methods. These specific standards were requested by the commissioning organization, ProDocsFormat, whose customers operate in the public and defense sectors and require documented security assessments of the systems they deploy. As the organization will eventually need to support security proposals for the public and defense sectors, including NATO level requirements. To make the process as effective as possible, the study does not use every single rule found in these large documents. Instead, it focuses on the most relevant requirements for document generation and data integration. For instance, rather than testing all 345 rules in the ASVS, only the controls that directly affect the DocOrigin pipeline are selected. Similarly, not all 286 rules from the AS&D STIG were used, only the ones relevant were chosen for the security assessment. More in-depth process of the screening, inclusion and exclusion criteria, as well as final rules used for evaluation, can be seen on the Chapter 4.2.

DocOrigin is a comprehensive software platform designed for enterprise-level document generation and Customer Communications Management (CCM). It is developed by Eclipse Corporation [40] (US). The system allows organizations to transform raw data from various business systems into professional documents. These documents include business critical documents such as invoices, purchase orders, and shipping labels. The platform is built to integrate with existing infrastructure like ERP systems to automate the production of high-volume output.

In this case study, DocOrigin is evaluated as a whole. DocOrigin is a set of applications for different purposes, that are intended to work together. The most important ones are the "Main merge engine", and the DocOrigin Web Services, that is referred as "DOWS" in this work. The merge engine is the main backend software that generates the final output, DOWS is a lightweight core or a wrapper to this

engine that provides an ease of access use, such as using merge remotely or trough an API for example, this of course raises security questions that are looked into in this chapter.

There are also good-to-know applications included in the DocOrigin package: Configuration Editor, Design, FilterEditor, FolderMonitor, and more, but are not in the main spotlight of this study. Some are not even included in this list, due to being either tools or non-GUI programs, treated as working on top of the actual engine.

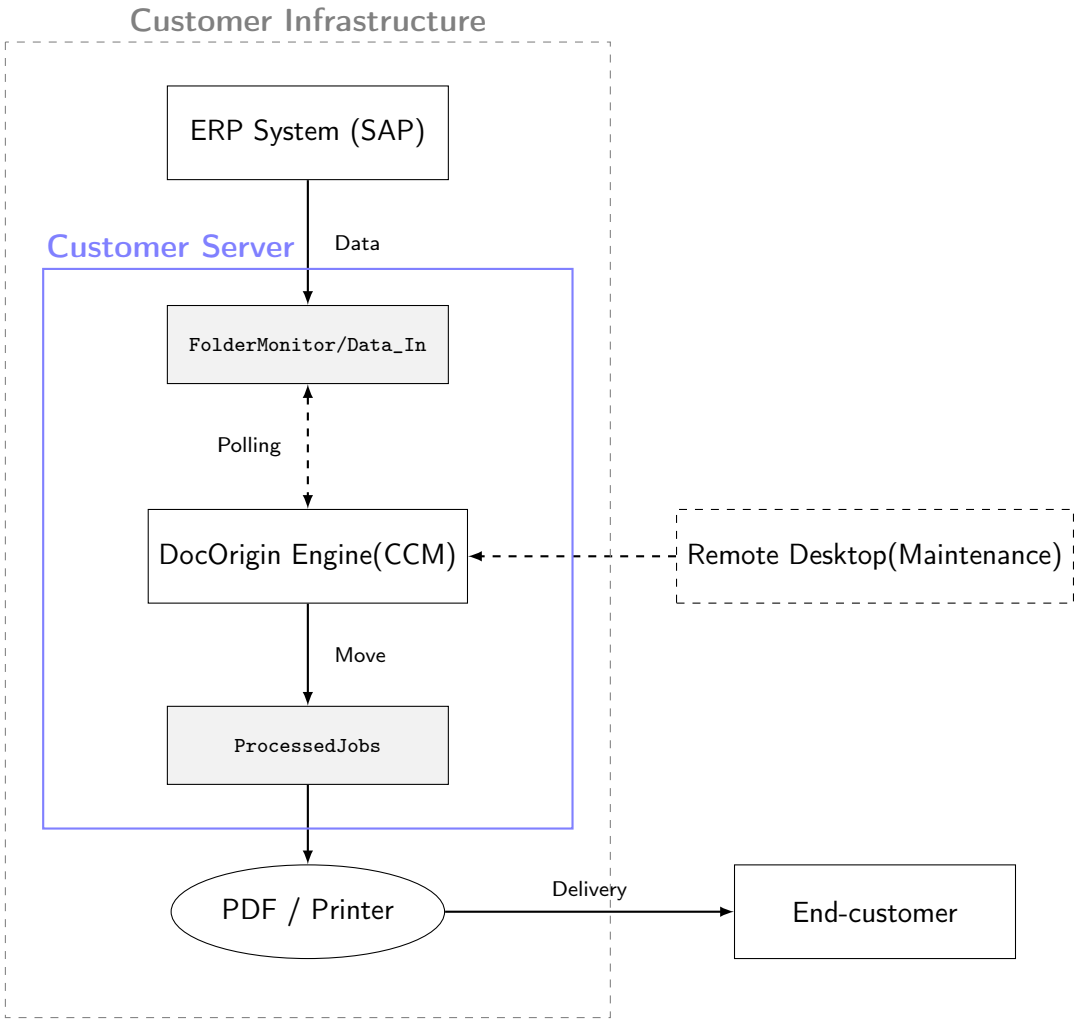

Figure 4.1: Logical system diagram and key components of the DocOrigin integration.

Figure 4.1 shows the logical structure of the DocOrigin environment and how

it connects to the customer infrastructure. It is important to note here that this is architectural description. It does not try to be a DFD, therefore it does not follow rules of formal Data Flow Diagram as discussed in the Section 3.3. That said, the goal of Figure 4.1 is to give the reader a simple and concrete overview of how different parts of the system are placed.

4.2 Assessment Methodology

4.2.1 OWASP ASVS

The OWASP Application Security Verification Standard (ASVS) v5.0.0 is a security standard for web application security. The standard can be used as a metric, as a guidance and also during procurement. In this study, the ASVS standard is used to investigate the internal logic of DocOrigin.

The OWASP ASVS v5 consists total of 17 chapters V1-V17. Each of these chapters are then divided into smaller sections for example section V1.1, with the section name Encoding and Sanitization Architecture. The sections are then finally separated to even smaller requirements (Req), for example Req ID V1.1.1 with the single requirements being the smallest objects. More clearly, the ASVS identification system goes as follows: `<chapter>.<section>.<requirement>` and even more specifically using the version number of the ASVS in front: `v<version>-<chapter>.<section>.<requirement>`. For example, ASVS Version 5.0.0, chapter 1, section 1, requirement 1 would be: `v5.0.0-1.1.1`. With the letter v in front always in lowercase.

The ASVS v5.0.0 consists total of 345 requirements. The requirements are divided to three security levels. Level 1 (L1, 72 requirements) is the basic-level and is often referred as the minimum baseline for web app security. It is focused on easily detectable, and high priority vulnerabilities, such as the OWASP Top 10. Level

2 (L2, 185 requirements) is more secure, especially intended to applications that handle personally identifiable information (PII), financial data, and other sensitive data. Level 3 (L3, 88 requirements) is for a high-value or critical systems. Because the DocOrigin platform is used in public and defense sectors, this study targets Level 2 as the baseline for the evaluation. This level guarantees that the document generation process meets the rigorous enterprise security standards.

First, the requirements of the ASVS must be screened, to those relevant and excluding the non-relevant ones for DocOrigin specifically. Out of the 17 chapters, we can clearly see right away that several of these can be excluded because DocOrigin is primarily a back-end document engine, and not an interactive social platform. Chapters V3 Web Frontend Security, V9 Self-contained Tokens, V10 OAuth and OIDC, and V17 WebRTC are considered Not Applicable (N/A) for this case study.

The focus is put on chapters that are critical for data processing and output generation. Chapters V1 Encoding and Sanitization, V2 Validation and Business Logic, are used to evaluate that the data from the ERP is handled safely. These two are grouped into group 1: Input security (What happens when XML data comes in? How it is interpreted, validated and processed?)

Group 2: Interface Security: V4 API and Web Service is relevant for DocOrigin Web Service (REST and SOAP), and partially relevant V7 Session Management.

Group 3: File and Pipeline Handling: V5 File Handling.

Group 4: Access Control: V6 Authentication, and V8 Authorization.

Group 5: Configuration and Deployment Security: V13 Configuration.

Group 6: Data Protection and Transport: V11 Cryptography (only relevant parts), V12 Secure Communication, and V14 Data Protection.

Group 7: Secure Design and Observability: V15 Secure Coding and Architecture, and V16 Security Logging and Error Handling.

All of the selected chapters are mapped into suitable groups, that help a little

when doing deeper analysis in the next Section 4.3. All Level 1 and Level 2 requirements from the selected chapters are included in the evaluation, of course, if not any surprisingly not-applicable requirements show in the process, those then will be excluded. Requirements are then classified by their applicability, applicability as Met, Partially Met, Not Met, or Not Applicable. This is done entirely based on the available documentation and system configuration; however, we do not delve too deeply into the application analysis to prove a single requirement, as there are hundreds of them; therefore, there must be some reasonable starting point for the case study.

4.2.2 AS&D STIG

The Application Security and Development (AS&D) STIG, Version 6, Release 4 contains 286 findings (rules), which will be screened in this chapter to those relevant and then analyzed more in Section 4.4.

The AS&D STIG cover a wide range of requirements throughout the software development lifecycle. The rules are divided into three severity categories. CAT I (High) contains 34 rules, CAT II (Medium) contains 230 rules, and CAT III (Low) contains 22 rules, a total of 286 rules. Not all of these are relevant for every deployment scenario, and applying all of them blindly would be both impractical and unreasonable.

The reference architecture for this case study is the Windows Server 2022 Standard installation. More specifically, the reference environment is a Windows Server 2022 Standard installation with DocOrigin deployed as a production application within it. This was selected because it represents a realistic and current production environment for DocOrigin deployments. there is no vendor-provided STIG available for DocOrigin itself. The AS&D STIG is designed exactly for this situation, is intended to be used when no application-specific STIG exists, it gives general hard-

ening guidance that is applicable to the deployment environment and the application running inside of it. [7]

A Windows Server 2022-specific STIG does in fact exist, and it was considered during the planning stage of this evaluation. It was deliberately not selected for this case study. The Windows Server 2022 STIG is designed to evaluate the hardening state of the Windows Server operating system itself. The findings are about OS-level configuration, therefore if it was chosen, the results would give more like a compliance statement of the OS, it would not touch or cover DocOrigin and the aimed ERP-to-DocOrigin data-flow security aspect at all. The commissioning organization also confirmed this choice, as their customer explicitly requested AS&D STIG compliance documentation for DocOrigin by name, because no vendor-provided STIG was available for the software.

Windows Server 2022 STIG and the AS&D STIG are not competing alternatives for the same evaluation target. They examine different subjects. The Windows Server 2022 STIG evaluates the OS. The AS&D STIG evaluates applications and their deployments. Since DocOrigin and its security posture are the subjects of this case study, the AS&D STIG is the appropriate and justified standard.

Important note on practical matters, STIG rules are called findings, like stated earlier. In the STIG, the findings are classified as Not Reviewed (grey), Not A Finding (green), Open (red), and Not Applicable (black). Therefore, in the STIG evaluation, the findings will use the exact same, official STIG classification system.

In STIG evaluation, no active penetration testing is performed. The reference environment is inspected and its configuration state is documented. Where relevant, command line, or PowerShell commands were used to verify the installation state of specific Windows Server features, these were useful already in the early screening process. The results are documented in this section. STIG Viewer checklist is a configuration audit tool, not a penetration testing framework. Therefore, this

specific approach to STIG assessment is used. After all, this is how STIG assessments are intended to be done.

All CAT I (High) rules were included by default, as there are only 34 total high severity rules, unless manual screening proves otherwise, the non relevant rules are then excluded. CAT II (Medium) rules were included if they were relevant to the Windows Server environment, the Java Servlet container (Apache Tomcat), or the DocOrigin application deployment. CAT III (Low) rules were included only if they are directly relevant to the identified risks from the ASVS analysis in Section 4.3.

CAT I Screening

First, CAT I high severity rules were evaluated and screened one by one. For the screening, the STIG Viewer 3.7 desktop application for Windows was used. In the STIG Viewer, it is possible to view all of the findings (rules), but even better methods for making the screening process more smooth, exist. For this, a specific STIG checklist was first created and then filled. As for this STIG evaluation, a manual spreadsheet cell filling is not the way to do it.

Rules specific to technologies not present in this environment were excluded. The scope of this study is a production server environment and not a development environment. For instance, mobile application development, browser-based client applications, or source code development pipelines are therefore excluded.

For specific rules, Group IDs were used as the main identifier. Group ID such as `V-222399` is precise enough, since the case study uses the specific AS&D STIG V6R4, there is no need for using the more precise, longer identifiers, such as the Rule IDs: `SV-222399r960759` that points to the same exact rule. All individual rules contain the following fields: **Rule Title** describing what the rule is about, **Discussion** that works as a more detailed description, **Check Text** giving instructions of how to confirm the rule state, **Fix Text** giving information how to fix the rule, and

References pointing to the corresponding security controls in the NIST framework.

The first exclusion category covers federated identity and message-level security protocols. Rules `V-222399`, `V-222400`, `V-222403` and `V-222404` address WS-Security (Web Services Security) timestamps and SAML (Security Assertion Markup Language) assertion handling. These rules are only relevant when Active Directory Federation Services (AD FS) is installed and configured on the server. In the reference environment, AD FS is confirmed not installed. This was verified through configuration inspection using the PowerShell command `Get-WindowsFeature ADFS-Federation`, that returned an Install State of *Available*, that confirms it has not been installed on the server. These four rules are therefore classified as Not Applicable.

The second, "easy" exclusion category is PKI-based (Public Key Infrastructure) client certificate authentication and DoD Common Access Card (CAC) technologies. Rules `V-222550`, `V-222551` and `V-222555` are about PKI certificate path validation and FIPS-validated (Federal Information Processing Standard Publication) cryptographic module authentication. The reference environment uses credentials (username and password) combined with one-time password (OTP) based multi-factor authentication, not certificate-based or CAC-based mechanisms. These three rules are therefore classified as Not Applicable.

Third exclusion category is browser-based attack surfaces. Cross-Site Scripting (XSS) could technically be relevant for DocOrigin, since it is mostly using JavaScript for scripting, however, DocOrigin is a server-side document production engine that produces output in formats such as PDF. It does not produce HTML content rendered in a browser. XSS in `V-222602` needs a browser to translate and execute injected script, which is not applicable in the reference environment. In addition, the rules about browser session cookie destruction `V-222578` and sensitive data in hidden HTML form fields `V-222601` would require a browser-based interface that

simply does not exist in the environment, therefore these three rules are classified as Not Applicable.

The fourth exclusion category is about SQL injection `V-222607` there is no relational database in the reference environment. DocOrigin does not connect to any SQL database on this server, therefore this rule is also Not Applicable.

CAT II Screening

There, CAT I high severity rules, out of 34 rules 23 passed and will be evaluated in the Section 4.4. Next, is the screening of the CAT II medium severity level rules. Because there are total of 230 CAT II rules, individual manual screening is not feasible. However, a similar category-based exclusion approach was used with the medium-severity rules. The STIG Viewer: STIG Checklists Filter Editor was used and proved to be a very powerful tool for this task.

Essentially, a two-pass bulk exclusion approach was used to narrow down the rule set. Entire technology categories were excluded in the first round. The second round excluded organizational process requirements.

The first exclusion pass removed rules that had already been confirmed not to be applicable, and technologies were used. The same way that in CAT I, SAML and WS-Security rules are excluded, as well as PKI, CAC, PIV, and FICAM rules were excluded on the same basis as in CAT I. Browser-based rules covering cookies, CSRF, and session cookie flags are excluded because DocOrigin produces document output files and has no browser-rendered interface. Relational databases, SQL, mobile, wireless-related rules, and DoD-internal registries; PPSM CAL and DoD Ports and Protocols Database were excluded. Because US federal administrative requirements not applicable to this Finnish enterprise deployment. Pure software development lifecycle rules addressing code reviews, defect tracking systems, and developer training programs were also excluded. NSA-approved cryptography for

classified information was excluded. The first pass removed 37 rules, leaving 193 rules.

The second pass, organizational process requirements was filtered out using the filtering tool (does not contain) administrative duties, policy documentation for organizations. For example, Configuration Control Board requirement, Software Configuration Management plan, ISSO-level audit trail review duties. These directly do not apply to the reference environment, or are not essential by any means from the perspective of this duty. Although they are still valid security controls in a full organizational audit, they could not be evaluated through a configuration inspection of a server. Therefore, they were classified as Not Applicable in the context of this technical assessment. In addition, three application-level session management rules were excluded because they had already been fully addressed in the ASVS analysis in Section 4.3. Re-evaluating them at the environment level would not provide any additional benefits. Therefore, the second pass removed a total of 25 additional rules, leaving 168 CAT II-level rules.

CAT III Screening

Finally, a screening of the CAT III level rules was done. Like stated, strict criteria was used. Because low-severity rules are also important, but the main focus of this thesis had to shifted towards the most critical rules by severity. CAT III rules were included only if directly relevant to ASVS findings. All of the following categories were classified as Not Applicable in the process: Account lifecycle notification rules (application level), decommission notification procedure, both organizational processes, not server configurations. DoD mandatory consent banner, DoD-specific US federal requirement, display last successful logon time, application-level UI feature. Notify ISSO of failed security verification, organizational notification process. Development process artifacts: test plans, code coverage, coding standards, design

documents. Production deployment scope. After the tough screening process, only 4 relevant CAT III rules remain.

4.3 Application Security Analysis: OWASP ASVS v5

Because input validation is perhaps the most critical point in the DocOrigin pipeline, practical penetration testing was conducted in a local test environment to verify the most critical requirements that could not be confirmed through documentation alone. For example, v1.1.1, v2.2.1 and v2.2.2 were proven not met.

However, because of resource constraints, a full penetration test of all 345 requirements was not feasible. Therefore, a representative feature, the Auto Translate mechanism, was selected for penetration testing. This feature serves as a proxy to evaluate the practical implementation of input validation controls.

4.3.1 Input Validation and Encoding

Chapter V1 in the OWASP ASVS focuses on how the application processes and data interpretation. For a document generation engine, the most significant risks involve how the system parses complex data formats, such as XML or JSON, and how it handles scripting languages, such as JavaScript.

A primary section in this chapter is V1.5 Safe Deserialization. Requirement V1.5.1 specifically talks about XML eXternal Entity (XXE) attacks. Because DocOrigin typically relies on XML data from the ERP to populate document templates, the XML parser must be configured securely. Therefore, XXE attacks are highly relevant to DocOrigin.

The DocOrigin Reference Manual reveals a lack of specific information about parser security configurations. The documentation does not explicitly state that

external entity resolution is disabled. Because of this, the specific requirements can not be classified as met without any formal proof. This is a representative proof-of-concept (PoC) to verify the most critical controls.

Successful attacks were performed to prove that requirements V1.1.1, V2.2.1, and V2.2.2 were not met. For the testing environment, a simple local DocOrigin environment was set up. DocOrigin's own sample XATW (DocOrigin Form file) was used, as well as their own sample translate INI file and sample XML. These were used as a base, and the language translations were tested as in the example. The files were then modified to test the behaviour of the system with unexpected data.

Test1: V1.1.1 (Canonical form and decoding): Not Met. URL-coding test was done. Replaced the language code with `%45%4e` that stands for "EN", the fact that the PDF output contained literally `"%45%4e"` informs that the system does not decode the input to canonical form before using it. The requirement v1.1.1 demands that the input is normalized before processing. If the system in this case treats both "EN" and `"%45%4e"` differently in the same context, without normalizing it, this means that it is vulnerable to code-based bypasses.

Test2: V2.2.1 (Input Validation against allow-list): Not Met. The requirement states that, input needs to be validated either against allow-list or against a pattern. The language code such as [EN] could be successfully replaced with any text, this proves that there are no allow-list for language codes.

The DocOrigin form expects lines starting with tildes (~) to be translated trough a dedicated INI file. By using the Auto Translate example directly from the DocOrigin Reference Manual [41], it proved successful that the DocOrigin software does not validate the language field in any way. Therefore, by modifying the XML file and the INI file containing any text directly in the language name and the field, both ran successfully through merge, and conducted the output PDF with potentially malicious data. In other words, when the language key was encoded, the engine

treated the symbols as a literal string rather than decoding them into a standard form.

Test3: V2.2.2 (Validation at trusted service layer): Not Met. Because DocOrigin Merge, which is the engine and the server layer, trusts the XML data value as it is, and uses it to find file path or INI-section, therefore no validation happens on the trusted layer.

These results prove that the requirements v1.1.1, v2.1.1, v2.2.1 and v2.2.2 are classified as not met.

Test4: V1.5.1 (XML External Entity - XXE): Not Met. Requirement V1.5.1 specifies that parsers should be configured to disable the resolution of external entities to prevent XXE attacks. This is critical point for DocOrigin or any document generation engine, because it uses XML as its primary data source, a vulnerability here would allow an attacker to exfiltrate local system files or perform internal network scans via the merge process.

The test4 involved injecting `DOCTYPE` declaration with a `SYSTEM` entity pointing to a local text file `secret.txt` into the sample XML data stream. The goal was to see if the DocOrigin Merge engine would resolve the entity and log the file content either in the log or in the resulting output PDF. Initially, the inclusion of a DOCTYPE declaration caused the parser to enter strict validation mode. If the DTD (Document Type Definition) did not perfectly match the XML structure, the engine returned a `Return code 53`. That indicates a fatal parsing error. This "fail-fast" behavior is a significant defensive trait. The system is more likely to crash or reject the file than to process invalid structure (potentially malicious payload).

After refining the DTD to match the expected schema, a successful merge `return code 0` was achieved. However with JavaScript logging, it turned out that the data field remained empty (`[]`). This indicates that the attack did not result in a successful data leak. Although it does not prove that the system is well secured

against XXE attacks, it suggests that the engine utilizes a validating parser that is highly sensitive to the consistency of data binding.

The XXE experiment was more detailed because it is a critical vulnerability for document generation engines. The fact that the engine frequently "drops" data items when it receives unexpected XML structures (could be unintentional) but provides effective security control. It can be stated that DocOrigin behaves as a "deterministic engine" that requires a perfect match between the XML schema and the form design. Security context, XML parsing layer provide a layer of structural hardening that makes exploitation difficult in practice.

The DocOrigin pipeline security relies heavily on trust and the integrity of the data stream and the hardening of the surrounding infrastructure. There are in total 43 requirements (out of which 30 V1 and 13 V2) and out of these 39 is level 1-2 relevant for DocOrigin architecture, to keep the text natural and easily accessible, only the most relevant, manually tested requirements are presented here in detail.

4.3.2 Interface and Session Security

DocOrigin is mainly a backend engine, but it also has a DocOrigin Web Service, which is referred to as DOWS. DOWS is the focus when analyzing interface and session security V4 and V7, which turns out to be only partially relevant for DocOrigin architecture as a whole.

First of all, it is important to understand the logic and the overall function of the DOWS. DOWS can actually be referred as a thin wrapper for the DocOrigin Merge engine itself. DOWS is not a native, deep-logic full web application, it acts as a translation layer that converts incoming HTTP requests (REST/SOAP) into locally working command-line executions on the server. Therefore, its primary function is convenience, as it allows for remote system control. The DocOrigin reference manual clearly states that the most obvious use case for the DOWS is to integrate one's own

applications, access them from web pages, and use DocOrigin from a non-supported OS.

The wrapper thinness is nor a good or a bad thing from a security perspective. One way to look at it is that it keeps the attack surface small by not actually having complex internal logic. Therefore it creates a Security Delegation model. DOWS does not protect itself; it relies only on the Java Servlet Container (Tomcat) for identity and the operating system for file integrity.

The command-line tools DOWS uses: `RunScript` and `Merge` need a standardized gateway, DOWS has built REST and SOAP for this. This *pass-through* architecture is a central component of the DO system's decoupled design. It also, however introduces some very specific security characteristics.

The DOWS architecture is built on Security Delegation model. This means it does not perform internal business logic validation. It relies on the hosting Java Servlet container Apache Tomcat, (or similar that has been configured in specific cases this may vary), for authentication and on predeployed JavaScript files for processing logic.

How the `options` parameter is handled, is a prime example of evidence that supports the theory of "dumb engine". The technical manual explicitly states that this is an "arbitrary string" which the web service is "oblivious to." If this string is passed directly to these scripts without inspection, the service creates a semantic trust gap, where it essentially assumes that either the calling system has already sanitized the input or the target script will do so. Therefore the interface is a neutral undefended platform for potentially malicious data (even though from the inside).

Even though the "dumb engine" hypothesis stands still, the DOWS actually implements very strong resource isolation pattern. As, for every incoming request to run a job, the service creates a unique GUID-named temporary directory (Globally Unique Identifier), for example `1fb4e383-fe9f-475e-ad90-45ffc981f5e2` and this

is at the same time the corresponding Job ID. All input files, logs and generated outputs are restricted to this directory. This is in line with the principles of isolation, ensuring that concurrent jobs do not interfere with each other's data. This creates a baseline type of defense against certain types of file system-based attacks.

Because of the stateless nature of the DOWS, the most logical way to analyze it is to not active penetration testing, but to systematically review the technical documentation and the default configuration files.

V4.1.1 (Content-Type Headers): Met. According to the processing details [41], DOWS uses a communication file `jobInfo.properties`. The script responsible for the job must explicitly set a `mime` property usually application/pdf as well as plain/-text as applicable. The web service then uses this value to set the `Content-Type` header in the HTTP response. Therefore, is confirmed that while the engine is "oblivious" to the content itself, provides the standard required mechanisms for the caller to receive correctly labeled data, satisfies the V4.1.1 requirement of explicit media, safe character encoding.

V4.1.4 (HTTP Method Enforcement): Not Met. V4.1.4 requires that only explicitly supported HTTP methods are allowed and those unsupported are blocked. This requirement is clearly not met, since the DOWS documentation does not provide instructions for restricting HTTP methods in the application logic. Because like stated earlier, DOWS acts as a "pass-through" to the command line, it delegates method filtering to the Java Servlet Container (Tomcat). Therefore, in a default installation, without manual hardening of the `web.xml` file, interface is open to all methods supported by the container.

V4.2.4 (Header Injection and CRLF): Not Met. Not allowing any CR (Carriage Return), LF (Line Feed) or a combination of these two: CRLF (Carriage Return Line Feed). Basically the application needs to accept only HTTP/2 or HTTP/3 requests with none of the previous to prevent header injection attacks. The DOWS

reads `name=value` pairs from the `jobInfo.properties` file, then converts them to `do.property.xxx` headers. Since the web service does not "know" about these strings there are no documented evidence that there is a sanitization layer that blocks these characters from being in the header stream. If a malicious script or unsanitized input reaches the properties file, it could potentially lead to HTTP response splitting. Due to time constraints these can not be all tested in a manual penetration testing setup, even this one especially seems very tempting and interesting.

Over half of the V4 requirements are classified as not applicable because of the use case and triviality of the functions of DOWS. These three included here were the most relevant, and the one's that could be best answered with analyzing the DOWS documentation as well as some minor manual configuration files exploring. The V7 Session Management is even more towards not applicable as a whole, but it was still analyzed because of the couple of interesting findings that turned out more relevant that one would have figured initially.

Chapter V7 is all about session management. The stateless nature of DOWS, again, because of this, needs to be analyzed very differently than an actual online website. One way to possible look at this would have been just completely classified it as N/A. But, the analysis can be done if we look at the DOWS Job IDs (GUID) that they actually act as a temporary session tokens for the software. It is very clear that this is not typical session management, as multi-factor authentication (MFA), logging out, concurrent sessions are not applicable, for example. DOWS is designed to be stateless, meaning it treats each request as an independent event. Authentication is delegated to the Java Servlet Container (Tomcat), and the "session" is effectively reduced to the lifecycle of a single job. Nevertheless, many V7 requirements are intended for interactive web frontends and are therefore not applicable. Several fundamental security controls regarding token generation and timeouts are highly relevant to how DOWS handles Job IDs and temporary file storage.

The DOWS does not care about the user's state, it only keeps track of the Job's date. The most crucial security controls found here are the `KeepJobHours` and `RunTimeoutSec` parameters. These can be considered the primary mechanisms for Resource Lifecycle Management. By automatically deleting the Job directories (GUID) after a set time, the system mitigates the risk of long-term data exposure on the server file system. However, the lack of a true session logout, as in V7.4.1, and the dependency of static credentials for initial access, as in V7.2.2, mean that the security of the pipeline still depends entirely on the environment security itself, secure integration with the ERP, and hardening of the environment.

V7.2.2 (Static vs dynamic tokens): Partially Met. The requirement requires the application to use either self-contained or dynamically generated reference tokens for session management and not static secrets such as API keys. In DOWS this requirement is split, initial authentication typically relies on static credentials (stored in `tomcat-users.xml`), fails the requirement. Once request is accepted, the system generates a unique dynamic **Job ID (GUID)**. It can be stated as the system not fully passing the requirement as a default. Transition from a static user session to dynamic data session is clear.

V7.2.3 (Token entropy): Met. Session tokens or reference tokens must be unpredictable. They need to be unique and generated using a cryptographically secure random generation. By utilizing Version 4 UUIDs (GUIDs) for Job IDs, DocOrigin provides 128 bits of entropy. This is met.

V7.3.2 (Absolute Session Lifetime): Met. This requirement is met as functional equivalent. The requirement is about demanding a maximum lifetime for a session to prevent long-term data exposure. The DOWS `KeepJobHours` parameter defaults to 1 hour, can be adjusted by the user. This parameter serves as the absolute boundary for the session lifetime. Theoretically any user with enough privileges and the corresponding `jobId` can access the finished document output or log files, the

"session" only truly ends when the directory is deleted. By treating `KeepJobHours` as data retention timeout, it allows the engine to satisfy the spirit of V7.3.2, keeping in mind, for anyone reviewing the full audit, should be on track of the perspective here.

V7.4.1 (Session Invalidation/Logout): Not Met. This requirement requires the termination of expired session. For example by maintaining list of terminated session tokens. DOWS does not have a native logout mechanism, it is possible to call a `deleteJob` to remove a specific directory. Still the authentication process is technically active for future requests, therefore the requirement is not met.

The analysis confirms that DocOrigin Web Services acts as a stateless interface adapter. It can be said that the majority of the chapters V4 and V7 are either N/A or not met. The security of the system, when viewed specifically compared to these two chapters, seems to be strongly related to delegation. The system does technically provides some mechanisms that have been discussed in the Chapter 3, such as isolation (unique directories) and basic lifecycle management (`KeepJobHours`), the actual decision-making is delegated to the infrastructure.

The dumb engine hypothesis is also confirmed at the interface level. The system provides a secure "container" for specific jobs to run in, while the system itself remains oblivious to the actual content of the input data and requests. Therefore, the full integrity of the document pipeline depends entirely on the temporal isolation provided by the timeouts and the hardening of the actual environment.

4.3.3 File and Pipeline Handling

At its core, DocOrigin obtains input data, processes it, and produces output data. DocOrigin ecosystem essentially manages the lifecycle of the files on the server. File handling is one of the most crucial security points for DocOrigin because of its file-centric engine. The engine relies heavily on different types of files, configuration files

(Printer Configuration file extension) (`.prt`), and parameter file extension (`.prm`) and the data directories. In addition, almost all other configuration-related settings are set and configured in various files. The previously mentioned `.ini` files as well as JavaScript-based `.wjs` ("Wonderful" JavaScript file extension) files control many functionalities, such as data filtering, job name discovery, processing, and merge scripting. Therefore, the file system is its primary operational environment and is one of the main attack surfaces.

V5.1.1 (Documentation states max file size, permitted file types and malicious file handling): **Not Met.** V5.1.1 requires documentation to define the permitted file types and maximum size for every upload, as well as the expected file extensions. Additionally, how the files are processed to be safe for the end-users, and how the system reacts to a malicious files. DocOrigin's documentation is clear about its functional types (XML data and XATW forms) but do not state clearly any maximum file size limits for uploads to DOWS. There are also not a specific policy for handling malicious files beyond basic error return codes (e.g. Return 53 for invalid XML) and error handling. This goes well with the "Dumb Engine" hypothesis and confirms again, that DocOrigin views security as an environmental concern rather than an internal application feature.

V5.2.1 (Only accept files of size the application can process): **Not Met.** V5.2.1 requires the application to accept only files of a size that it can process without causing a loss of performance or a denial-of-service attack. As it turned out that there is no clearly stated maximum file size for input data or upload files, it could be very interesting to test it.

In a local penetration testing environment, this was tested using several methods and "tricks" to attempt to either crash the application or cause a significant loss of performance. If either of these attacks proves to be successful, it has been shown that the application once again delegates the security to the environment itself and

further proves the "Dumb Engine" hypothesis.

The most interesting theory to test this on is actually trying out the Exponential entity attack (Billion Laughs Attack) that works by using nested, but limited, level of entity recursions. A clear example of this would be referring to existing previous entities ten times and again ten times, doing this until the last line having the references increased to a large amount; this should use a large amount of memory during parsing. Figure 4.2

The first initial thought is to intuitively to try to run the DocOrigin merge with a excessively large file and review the task manager, try to find unusual memory usage, processor usage to get a hint if the plan would work in action. The billion laughs attack was in fact, even better way to approach the testing. This way, a small file, under 2 KB is used to trigger exponential resource consumption during the parsing phase.

When loading the payload into the DocOrigin Design data explorer, the application triggered a structural error *"no declaration found for element 'data'."* However, after acknowledging the error, the application entered a non-responsive state (deadlock). According to Windows Task Manager, a persistent 7 % CPU load as well as a static memory footprint of 30 MB. The application could not be shut down without forcing it with the Task Manager for example. It is important to note that this alone does not prove anything, as only the GUI froze on several tests. The next step was to configure the XML data to work seamlessly with the engine. To achieve this, it had to be done with a similar XML file, but only a few referred to previous entities to avoid any performance loss. When the form and data worked well together, the data was then changed back to nine lines of continuous references to the previous ones.

The behavior of the application initially shows that the DocOrigin XML parser attempts to resolve or expand the entity definitions before completing the structural

validation of the XML schema. It appears that even though the parser eventually identified the lack of a declaration for the `<data>` element, the recursive expansion of the entities had already triggered a computational loop that exhausted the application's processing thread.

The original XML was modified to fit and work well with DocOrigin. The XML file includes a root element and basic element declarations. By doing so, the parser could proceed past the initial validation stage and resolve the entities.

When performing a even better, more successful version of this test, the results were even more severe. The engine usually processes documents very quickly and the Merge.exe terminal only flashes quickly, this time a successful Denial of service attack (DoS) was performed. The recursive entity payload caused a total computational deadlock. This time Windows Task Manager showed persisting 7 to 8 percent processor usage for Merge.exe and about 10 MB memory usage. Not the GUI, but the merge engine itself. A large linear file size attack would spike the memory usage, this more complex attack maintained low memory usage. These indicate that the engine was trapped in an infinite internal loop while attempting to resolve the nested entities.

It is fair to say that there are not any reasonable time limit like in the DocOrigin Web Services `RunTimeoutSec=60` for example. While this did not test maximum file size for upload, this still showed significant loss of performance, which is exactly what V5.2.1 is about, even though it talks only about the file size. "Verify that the application will only accept files of a size which it can process without causing a loss of performance or a denial of service attack."

The fact that a small file can cause a persistent Denial of Service (DoS) confirms the "Dumb Engine" hypothesis. The application assumes that the input data is filtered in the previous stages of the document pipeline. The application does not verify the input, it is vulnerable to these types of resource exhaustion attacks.

Figure 4.2: Modified Exponential Entity Expansion Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [
 <!ELEMENT root (data+)>
 <!ELEMENT data (#PCDATA)>
 <!ENTITY a "expansion">
 <!ENTITY b "&a; &a; &a; &a; &a; &a; &a; &a; &a; &a;">
 <!ENTITY c "&b; &b; &b; &b; &b; &b; &b; &b; &b; &b;">
 <!ENTITY d "&c; &c; &c; &c; &c; &c; &c; &c; &c; &c;">
 <!ENTITY e "&d; &d; &d; &d; &d; &d; &d; &d; &d; &d;">
 <!ENTITY f "&e; &e; &e; &e; &e; &e; &e; &e; &e; &e;">
 <!ENTITY g "&f; &f; &f; &f; &f; &f; &f; &f; &f; &f;">
 <!ENTITY h "&g; &g; &g; &g; &g; &g; &g; &g; &g; &g;">
 <!ENTITY i "&h; &h; &h; &h; &h; &h; &h; &h; &h; &h;">
]>
<root>
 <data>&i;</data>
</root>
```

V5.3.2 (Use trusted data for file paths): Not Met. In the DocOrigin environment, the engine relies on the `HardenPath` logic to resolve folder mappings. The requirement demands that no user's own named file names. Or if there is, there needs to be a clear validation for file names to avoid any path traversal attempts. According to DocOrigin's documentation, restricted characters are only replaced by underscores for output file names. Therefore, there is no valid character check for file naming.

File handling remains, for the most part, the responsibility of the developers to ensure that everything is configured correctly, for example, by creating custom validation scripts for specific cases and needs.

4.3.4 Authentication and Authorization

DocOrigin lacks internal password management and multi-factor authentication (MFA) logic. DocOrigin is primarily on-premises software. It delegates authentication to the Java Servlet Container (Tomcat); DocOrigin is stateless and lightweight in this regard. Most V6 requirements, such as password complexity (V6.2) and cre-

dential rotation (V6.2.10), are Not Applicable at the application level. However, there are requirements that are unintentionally classified as Met by the standard.

V6.1.3 (Multiple pathways documented): Met. The documentation clearly states REST and SOAP as the two independent authentication pathways.

V6.4.2 (No password hints/questions): Met. The system lacks any password recovery or hint features, which unintentionally satisfies this requirement.

V6.1.1: Not Met. The requirement demands that there needs to be a clear mention in the documentation of how controls such as rate limiting, anti-automation and adaptive response are used against attacks such as credential stuffing and password brute force. DocOrigin documentation contains no guidance on defending against these types automated attacks. Therefore the requirement is classified as Not Met.

V6.3.2 (no default users used): Not Met. The requirement demands that no default users are used such as `"admin"` or `"root"` or they are disabled. DocOrigin does not ship with a hardcoded `"root"` user, the deployment guide provides specific XML snippets for `tomcat-users.xml` using `username="admin"` and `password="admin"`. This creates a risk, if the prerequisites of the documentation is followed literally, and these examples are not removed for any reason, the production environment then will have a high-risk default credential set.

V6.3.3: Not Met. There is no documented use of MFA in the application, even though in a modern enterprise environment, Multi-Factor Authentication (MFA) is a baseline requirement (Level 2). DocOrigin Web Services fails this requirement entirely at the application level. Because, like stated, the DOWS adapter is a "thin wrapper", it does not require the user to complete a second factor. This is significant for the entire document pipeline's security, when user gives a valid single-factor credential to Tomcat, the baseline is that they have full access to trigger any script or job.

There is no clear authorization documentation provided in the official DocOrigin

documentation, besides the DOWS roles and the job isolation logic. The software assumes that if a user is authenticated by Tomcat, they are authorized to initiate jobs. This supports the "Dumb Engine" hypothesis. However, the system does implement a form of resource-level authorization through the use of unique, GUID-named folders.

V8.1.1: Not Met. A complete lack of authorization policy documentation is a notable finding in this case study. It is clear that a lot of effort has been put into the practical functionality, ease of access and overall user experience. But at the same time, not into the security aspects of the software. The vendor provides instructions on how to make the system function, but not on how to make it secure. The documentation is intended to be used as a reference manual, it even states that it leaves the "web server portion entirely to you".

V8.2.2: Partially Met. The requirement demands that data-specific access is restricted to only those that have the permissions to the specific data. The aim is to mitigate insecure direct object reference (IDOR) and broken object level authorization (BOLA). The DocOrigin Web Service uses a GUID-named folder strategy to store job data. From the perspective of a remote API caller, this mitigates Insecure Direct Object Reference (IDOR) because the 128-bit identifier is computationally impossible to guess. If an attacker or an insider can view the server's file system or logs, the GUID-named folders provide no actual protection. The system provides the naming scheme for isolation, but the enforcement is delegated to the operating system's file permissions.

The documentation for script security shows that restricting specific scripts to specific users is possible but is disabled by default as of version 1.3.5. This indicates the idea very well, functional accessibility is prioritized over strict authorization. It is left to the developer and system administrator to implement and document any custom authorization logic in to the `WsSecurity.wjs` script.

The remainder of the requirements are classified as either not applicable or not met.

4.3.5 Configuration and Deployment Security

Because the DocOrigin software delegates the security so heavily to developers and system administrators, configuration files can become a primary target for attackers.

The DocOrigin ecosystem is not a single application. It is more accurately described as a modular collection of specialized tools and engines. Components such as the Merge engine, FolderMonitor, FilterEditor, Design and more work together to form a complete document production pipeline from the DocOrigin perspective. Of course, the pipeline extends beyond the DocOrigin ecosystem. That said, these tools are connected to each other and form a toolpack together, they are connected by a variety of configuration files, such as the `.wjs`, `.ini`, as well as printer configuration files `(.prt)`, and parameter files `(.prm)`.

Because of this modular design, the software provides an open architecture. The DocOrigin vendor, Eclipse Corporation provides the functional parts of the software, but the actual configuration of the system is left to the developer. This implies that the system integrator is responsible for creating and planning directory structures, implementing version control, and verifying the integrity of the configuration files.

The DocOrigin is powerful engine designed for high volume document output, documentation states that for parallel jobs the maximum value for n is 62 (`concurrent` parameter). However, it is recommended to use a sensible value, typically the number of logical processors that the server has. The open approach is efficient for complex business needs, but the configuration phase could easily become a security blind spot if the developers in charge are not experienced and have the security first approach. It can be stated that even though the security first configuration is possible, it may not even be feasible for the developers due to lack of instructions

to configuring and building a secure platform for DocOrigin to work in.

Because the engine executes logic based on these external configuration files, the security of the entire pipeline is only as strong as the protections placed on the configuration environment. If the directory structures are unorganized or if the configuration files are easily accessible to unauthorized users, the "Dumb Engine" logic can be manipulated and exploited to perform attacks.

V13.3.1: Not Met. This requirement demands that a secrets management solution, such as a key vault must be used to control backend secrets. Additionally, secrets must not be included in the application source code or in build artifacts. DocOrigin environment relies heavily on plain-text storage for instance, in `.ini` and `.wjs` files. The credentials for the DocOrigin Web Service are typically stored in the `tomcat-users.xml` file as a default. This by itself confirms that software treats the server environment as a trusted space. Any implementation of such key vaults or other secrets management solution are left to the developers responsibility.

V13.2.3: Not Met. This requirement demands that a credential must be used for service authentication that can not be default credential such as `admin/admin` or `root/root`. The documentation for DocOrigin provides clear examples for setting up the system, such as using "admin/admin" or "user/pass" for Tomcat roles. These are at the same time examples in the documentation that are intended to be used as placeholders for initial testing, still, they represent a "Documentation-Induced Risk." If the developer configuring the system follows the documentation blindly to reach a "proof of life" state and forgets about changing these secrets in production deployment, this is a significant vulnerability. Therefore again, it is possible to harden the environment but DocOrigin is a professional tool that assumes the user has the expertise to do so, and leaves that open for their responsibility.

V13.1.2: Partially Met. This requirement requires the documentation to define maximum number of concurrent connections and how the application behaves when

the limit is reached to prevent denial of service attacks for example. Like previously mentioned, with the DocOrigin parameters `concurrent` and `RunTimeoutSec`, the system administrator can define performance limits. With these settings it is possible to prevent a single job from consuming all server resources. However, although these limits are documented clearly, the application does not have a native recovery or fallback system documented. The job is simply terminated after timeout, and with the `concurrent` tag it is predefined how many jobs can run at the same time initially.

V13.4.6: Not Met. To meet the requirement the application can not expose detailed version information of backend components. The Web Service headers, such as `do.service.base` often expose specific version and build information. The transparency could be useful for debugging and support. Detailed versioning should not be accessible to potential attackers as it can be used to identify known vulnerabilities in specific software builds.

In V13, the majority of the evaluated requirements for DocOrigin were classified as Not Met or N/A. At this point of the case study, this does not come as a surprise. DocOrigin software is clearly not intended to be used in an insecure environment. The study confirms that DocOrigin's security is an emergent property of its configuration. The software is intended for businesses with high volume and highly customizable document output needs, but the system administrator and the developers must have experience with scripting. As the software provides the necessary places for security implementations but does not show how to setup them internally.

Because the DocOrigin engine is "oblivious" and its configuration is stored in easily readable files, the integrity of the entire pipeline depends on the hardening of the server. This proves the importance of the upcoming STIG analysis because the engine's configuration is only as secure as the file system permissions protecting it.

4.3.6 Data Protection and Secure Communication

In the evaluation of the chapter V11 Cryptography, DocOrigin software performs poorly. Having difficulty in classifying the entire chapter V11 as either **N/A**, most of it **delegated**, or **Not Met**, this chapter is justified briefly because there is no suitable material for a full in-depth analysis. If the software is treated as a modern full-stack or web application, it would be clear that not any of the V11 requirements would not necessarily be not applicable. However, in the DocOrigin scenario, there are two main directions: either brutally classify all as N/A or Not Met. Because cryptography is heavily delegated to the external tools and a lot of the security depends on the environment hardening. It is mainly left open like this because cryptography remains one of the most important areas for supposedly sensitive data. The chapter can not be ignored, it became clear that like said, most of the encryption is simply delegated to external tools or software or not taken into the consideration at all. For future work, this chapter is important to keep here.

In DocOrigin, it is possible to use external tools, such as the OpenSSL binary to perform signing and encryption tasks. The use of these tools is recommended in the documentation but for the most part, any encryption is not done. The OpenSSL is only mentioned briefly in the Signing and Encrypting mail -part of the documentation [41]. Although it is advised to be installed, it is still an external tool that the system administrator may ignore and therefore will not be installed and used. It does not come as a default with the software in any way.

The software does not have its own security tools built-in. The scripting part of the documentation reveals a lot. Scripts can use functions like `_toBase64`, which are only used to change the format of text. Encoding data is not the same as protecting it with encryption. Encoding does not hide the information or prevent others from seeing it. It only makes sure the data can be sent easily between different systems.

Most of the V11 Cryptography can be either classified as not met or even N/A

if we the engine is looked as a pure core engine that is not even intended to do any encrypting in the first place. The lack of internal security is clear in the email section of the documentation. The documentation states that other tools must be used for signing and suggests using the OpenSSL program. This proves that DocOrigin acts as a middleman. It does not handle security rules or keys by itself. Instead, it calls for help from other programs already on the server. The documentation provides a brief installation guide for OpenSSL.

V11.2.1: Not Met or Delegated. To meet the requirement, it is necessary to verify that only industry-standard software and hardware are used for cryptography. DocOrigin does not have security tools built-in. It only provides some instruction on how to install external programs like OpenSSL. The software does not check if that external program is safe or up to date. It seems that the software also does not verify this in anyway, the documentation refers to OpenSSL v.1.1.1 more specifically. OpenSSL v1.1.1 was released on September 11, 2018. It has even confirmed to have reached the end of life as of 2023 [42]. It seems that the DocOrigin documentation does not get regular updates.

V11.1.1: Not Met. To meet this requirement, there needs to be documented policy of how the application manages cryptographic keys. Also, a cryptographic key lifecycle that follows a valid key management standard such as NIST SP 800-57. The engine treats security certificates like any other simple file. It does not have a built-in system to manage or protect these keys. Certificates are handled as flat files by the user, and the software provides no safe key vault to store them.

V11.5.1: Not Met. To meet the requirement, all random numbers and strings that are intended to be non-guessable, must be generated with using cryptographically secure pseudo-random number generator (CSPRNG) and have at least 128 bits of entropy. The ASVS standard is very strict about how random strings are created. It explicitly states that standard UUIDs (or GUIDs) do not meet the requirement

for 128 bits of entropy. DocOrigin uses "guid-like" names for its job folders, it technically fails this requirement.

V11.7.2: Not Met. To meet the requirement, there needs to be a minimum data exposure during process and the data needs to be encrypted immediately after use or as soon as feasible. The `KeepJobHours` feature automatically deletes temporary job folders after a set time. This is more of a step to the right direction but does not alone meet the requirement. More specifically, the deletion process only targets the job folder. It does not clean up the log files containing processedjobs. The log show details about the processing job, therefore may contain fragments of sensitive data. Log files itself do not get deleted from the server automatically as a default, this creates a digital trail that stays unprotected.

Moving to **V12 Secure Communication**, it becomes clear that DocOrigin does not manage its own transport security and provides no native support for modern encryption features. The whole chapter V12 Secure Communication is focused on how the application protects data while it is moving through the network. The chapter V12 is divided into three main parts; General TLS (Transport Layer Security) Security Guidance, HTTPS Communication with External Facing Services and General Service to Service Communication Security. For a usual, modern web application V12 is about managing the "tunnel" (HTTPS/TLS) between the user and the server. However, for DocOrigin specifically, this chapter confirms a total reliance on external systems.

Because, like stated earlier, DocOrigin Web Services (DOWS) only acts as a thin wrapper for the merge engine itself, it does not have its own code to manage encryption protocols or certificates. It is in its core, a java application that is inside a web server like Apache Tomcat. The application trusts the platform, does not natively validate data, it assumes that if a request reached the engine, the web server has already handled the security.

For an on-premises tool that DocOrigin mainly is, besides the thin wrapper, DOWS, almost every requirement in V12 is delegated to the environment.

V12.1.1: Not Met. The requirement is met only if the latest recommended versions of TLS protocol are used, such as TLS 1.2 or 1.3. The latest version of the TLS must be the preferred option. DocOrigin does not have the internal logic to choose a TLS version. In this sense, DocOrigin sits behind Apache Tomcat, if an administrator wants to use a secure TLS tunnel, they must configure it in the Tomcat `server.xml` file. DocOrigin code itself does not have a way to use or even see which version of TLS is being used, it can't meet the requirement natively.

V12.2.1: Not Met. To meet this requirement, an application must use TLS for all connections and never fall back to insecure or unencrypted communication. The DocOrigin documentation almost encourages the opposite for initial setup. "Calling The WS" -section suggests using port 8080 (unencrypted HTTP) and also shows example of the `-k` or `-insecure` flag with the curl command. This option ignores any invalid and self-signed certificate errors, essentially allows Curl to use unsecured SSL connection, skips certificate checks without compromising encryption [43]. Of course, the documentation explains that this is done to avoid hassles during the start of a project. It also confirms that the system is not secure by default. The documentation allows and shows these insecure paths openly, therefore requirement is not met.

V12.3: Not Met. General Service to Service Communication Security (V12.3) requires the application to use TLS or another appropriate transport encryption mechanism in the first place, and more specifically for all inbound and outbound connections, as well as internal communications. Any of these should not fall back to insecure or unencrypted protocols. Any of these requirements are not met because first of all, the outbound and inbound communications is left to the system administrator to implement, according to the DocOrigin documentation. And in-

ternal communications between different parts of the system, such as FolderMonitor and the Merge engine communicate regularly each other by placing plain text files in shared folders on the server.

DocOrigin is not designed to manage its own network security. The documentation mentions insecure connections and their use, which is very usable in the development stages of an application but should not be used in production. The secure communication of the software, even internally, has not been implemented, as it seems it was not intended to.

V14 Data Protection is up next. The V14 has total of 13 requirements, and it is roughly divided into V14.1 Data Protection Documentation, V14.2 General Data Protection and V14.3 Client-side Data Protection.

DocOrigin is actually very clever in a sense of filtering incoming data, most commonly XML. It has internal, effective validation for XML, if the syntax is off, it breaks, it has the capabilities built-in to fix slightly corrupted data and multiple filtering options for the incoming data as a default. As proved in the test1 in 4.3.1, the data for the DocOrigin merge engine is just data. The XML data it processes can contain anything, like a private medical record, it does not make any difference, to the engine, all data is simply a series of tags to be placed on a page. The software therefore does not provide any native tools for classification or masking for data.

V14.1.1: Not Met. The requirement states that "all sensitive data" needs to be identified and classified into protection levels. To meet the requirement, it essentially requires the application to first identify sensitive data, DocOrigin treats all data the same. There are no default feature in DocOrigin to classify data to different priority levels or recognizing sensitive data automatically. The software does not provide these tools out of the box, the documentation shows that a developer can build them manually. For example, it is possible to use Object Tags to add custom labels to fields or use Processing Instructions to change how the engine handles specific

data. Also, it is possible to use JavaScript filters to remove sensitive tags before the merge process begins. However, because these features are not part of the standard engine and must be custom-made by the user, the software does not meet the requirement on its own.

V14.1.2: Not Met. OR N/A This is classified as Not Applicable because the requirement requires documentation to state certain aspects of the sensitive data protection levels, and as it revealed that DocOrigin does not have these as a default.

V14.2.6: Not Met. This requirement says that an application should only show the minimum amount of sensitive data needed. Like stated, DocOrigin does not offer recognize sensitive data, therefore there are not any native features to do this automatically.

V14.2.7: Partially Met. Requires that sensitive data is removed when it becomes outdated or unnecessary for example. In DocOrigin's case it seems simple, all temporary data is simply treated the same. Like stated previously, the software handles data retention through the `KeepJobHours` setting that essentially deletes temporary folders after set time. This still remains only partially met or even not met, as this is not complete security control. The system as a default does not securely wipe the disk or clean up the logs, this could be seen more as a convenience matter or done for performance reasons, but not giving full digital trail removal.

Client-side Data Protection **V14.3.1**, **V14.3.2**, and **V14.3.3**: N/A. These requirements focus on client-side or browser-based data protection. DocOrigin is a backend engine designed to run alongside existing business applications on a server, it does not interact with a user browser storage or local cookies. Client-side storage do not apply to the core engine.

4.3.7 Secure Design and Logging

V15 Secure Coding and Architecture consists of total of 21 requirements. These are further divided into sections: V15.1 Secure Coding and Architecture Documentation, V15.2 Security Architecture and Dependencies, V15.3 Defensive Coding, and V15.4 Safe Concurrency.

V15.1.1: Not Met. To meet this requirement the application documentation must provide risk based remediation time frames for third party component versions with vulnerabilities and for updating libraries in general, to minimize the risk from these components. It is difficult to tell about the component updating part, but safe to say that the documentation is not being open about this. Therefore, if the regular updates of DocOrigin do have any of these minimizing component risk updates, the documentation lacks the required time frames, requirement is not met.

V15.1.2: Not Met. To meet this requirement application must provide inventory catalog such as SBOM. Essentially a clear list of every library used in their software. The documentation does not list its internal components or does not provide any sort of SBOM either. Therefore the requirement is clearly not met.

V15.1.3: Met. This is met if the application documentation identifies functionality which is time-consuming or resource-demanding and provides instructions to keep the system available. DocOrigin reference manual explicitly states how to handle high-volume processing. It gives multiple tools to prevent the system from slowing down or crashing. The `-concurrent` tag is one of these, it is used to process multiple jobs at the same time. It warns the user to use a "sensible value" that matches the number of processors on the server. It is also possible to use Queues to manage work. User can set up to 10 different folders with different priorities. The documentation explains that jobs in "Queue1" are always finished before the system looks at "Queue2." It is even possible to run several instances of the FolderMonitor program at the same time. Each instance watches its own set of folders.

Additionally, the role of the `FMTransaction` helper app is explained in the reference manual. The main FolderMonitor does not actually process the jobs itself. It only "spawns" a helper app to do the work. Once the job is done, the helper app closes. The manual states this is done to keep the main system safe from memory leaks or open files.

V15.2.2: Not Met. This requirement is essentially similar to the V15.1.3, as it demands the system to have implemented these documented functionalities that are time-consuming or resource-exhausting for the system. This is not met as a default, although the documentation gives the explanation how to implement these strategies as discussed in V15.1.3, it is still the developers responsibility to actually implement and use them correctly.

V15.2.3: Not Met. This requirement is met if the production environment includes no extra files, only those required for the application to function. DocOrigin installer is designed for convenience and includes many extra files. It installs sample forms as a default, example scripts and executables that are not necessary for all users in the same directories as the production engine. The installer should have checks at least to only check those the user will be needing and different installation for development stage and production. Current situation, the user has the uncertainty of what could be deleted from the bin folder, so it becomes almost obvious not to touch anything in order for not breaking the system.

V15.3.1: N/A This requirement demands that the application only returns the required subset from a data object. Not the entire data object for example, like the full XML data. This is actually a tricky one because DocOrigin typically uses XML data, the application is designed to ask the user (developer) in this sense what to use as the XML root element. The developer could use the whole root element such as the `<document>` and in the form specifying what fields to actually return in the final output. Because this is the whole key point of the DocOrigin Design

application, and essentially the whole DocOrigin platform, is to filter and produce the output data with only desired fields of the data and not expose anything else to the end-user, this is completely in the hands of the developer. Therefore, the "user" is not the end-user, and the developer has access to the data, which the end-user does not have in a typical scenario. Therefore, the classification for this requirement is N/A, although it is very relevant and crucial for evaluating DocOrigin.

V15.3.6: Not Met. To meet this requirement, JavaScript code must be written in a way that it prevents prototype pollution, for example with using `Set()` or `Map()` functions instead of object literals. Prototype pollution is a specific attack where an intruder changes the behavior of all objects in a program by modifying a basic template. DocOrigin uses JavaScript (the `.wjs` files) to handle data logic, it is vulnerable to common web-based coding flaws. A sample script for Job Name Discovery provides clear evidence that the requirement is not met. The script uses basic string searching and standard variables to decide how to process a file [41].

V15.4.2: Unverified. Checks that check the resources state must be performed as a single atomic operations to prevent time-of-check to time-of-use (TOCTOU) race conditions. This could be for example checking if a file exists before opening it. DocOrigin documentation mentions concurrent processing but do not mention anything about being atomic in a sense of checking resources, therefore this requirement remains unverified.

The evaluation of V15 Secure Coding and Architecture shows that the assumptions of DocOrigin could be true. This means that the software is intended to be used to "replace" one's old legacy systems, while not actually replacing them. The marketing for DocOrigin is that it is possible to integrate DocOrigin on top of the legacy system without changing the existing data or the server architecture. DocOrigin is a professional tool built for high performance. It satisfies the identifying resource risks and provides instructions to manage them, which is why it meets the

requirements for system availability. However, it became clear that it fails when it comes to modern defensive coding and modern coding security standards. Out of all of V15, only one requirement is clearly met (V15.1.3) because of sufficient documentation of this area, which is very alarming for every other requirement. Secure Coding and Architecture is something DocOrigin does not satisfy as a default, except for the one requirement, therefore to truly satisfy this area, the developers and the system admin has to almost build majority of the functionalities from scratch. DocOrigin uses "older" JavaScript, such as basic string searching and saving values from the XML input data to default JavaScript objects, doing exactly what requirement V15.3.6 tells not to do, is to use object literals. In a worst case scenario, this could allow the prototype pollution attack that the requirement warns about. V15.4 Safe Concurrency is a section that was not been able to prove false nor true. It is notable to state that a complete deadlock of the application was performed in chapter 4.3.3 V5.2.1 successful "billion laughs" attack. Still, safe concurrency remains a mystery. DocOrigin gives valuable information of concurrent processing, and instructions to that, but is is unclear if the multi-threaded code is safe and has all the defenses in place to mitigate thread starvation or similar attacks. This is one area that is not feasible for manual penetration testing for this case study, because the documentation explains well how to run concurrent jobs the correct way, but it requires a lot of manual configuration. And also, the maximum amount for `-concurrent n` tag for n is 62, ("A sensible value would be in the vicinity of the number of logical processors that your server has."). While default value is set to 1, this would have to be changed for testing. It is also very difficult to determine what appropriate value it should be set to in order for the results to be academically reliable and reasonable.

Moving to the last chapter of the OWASP ASVS v5.0.0 for this case study is V16 Security Logging and Error Handling. The chapter consist total of 17 requirements.

These are divided into sections: V16.1 Security Logging Documentation, V16.2 General Logging, V16.3 Security Events, V16.4 Log Protection and V16.5 Error Handling.

The log files are crucial. The way DocOrigin stores log files is simple, essentially text files (`.log`) are saved on the application directory as a default. `C:\DocOrigin\User\Logs\15_03_2012_logfile.txt` for example (from the documentation). This directly proves that **V16.4.2** is not met. For this requirement to be met, the logs must be protected from unauthorized access and cannot be modified. Plain text files in the application directory structure as a default do not satisfy this requirement.

V16.1.1: Not Met. This is the only requirement in the V16.1 Security Logging Documentation. To meet the requirement, the documentation needs to be open about how the logging is performed at each level of the application's technology stack, how and what are being logged, where they are stored, how they are used, access control and how long they are kept. The DocOrigin documentation does not give any of these, essentially every event is being logged and stored as a text file as a default, the manual suggests "keeping 20 GB of disk space" on the server, which refers to manually cleaning or indirectly telling the developer to make scripts for automatically deleting the logs after set time. The documentation is open about the possibilities to configure the logging in a way that is desired, offering a lot of freedom for the developer, but at the same time configuration is left for the developers responsibility to do.

V16.2.1: Partially Met. This requirement requires the application to each log entry to include necessary metadata for allowing detailed investigation of the timeline for when event happens. This metadata should include things like "when, where, who, what". DocOrigin actually does OK job to meet this requirement partially. The DocOrigin logging as a default is very "throughout", it includes very precise

timestamps. The logs also clearly state "what" is happening by using descriptive labels like `Merge:JobStart`, `Merge:OpenFile`, and `Merge:ProcessEnd`. It also logs the logon user name from the operating system. The problem is that this is only the system-level user. It shows the account that is running the DocOrigin service, but it does not show the specific person who might have sent the document request through a web interface. The "where" part is missing or incomplete. The log shows the internal directory structure and modules being used on the server, but they lack any identifier of where. Therefore the requirement is only partially met.

V16.2.5: Not Met. The requirement is met if logging takes into consideration the data's protection level and highly confidential data should not be logged or should be masked or hashed either partially or fully. This confidential data includes data such as credentials for example. As previously shown, DocOrigin does not offer default filtering or masking for sensitive data, it does not "know" what data is because the engine is oblivious for the data. Therefore the logging will include everything that the developer sees, it is the developers responsibility to implement masking or hashing for data logging.

V16.4.3: Not Met. To meet the requirement, the logs need to essentially be separated from the core system. This needs to be done via securely transmitting them to logically separate system. This is not met as the like stated, the logs go in to the same directory structure as the application itself. The aim of the requirement is that if an attacker gets access to the application, they would not get access to the logs and they would not be compromised.

V16.5.1: N/A This requirement is met if only a generic error message is shown to the end-user or "consumer". This needs to be looked at a slightly different perspective, as the developer should get the "sensitive internal system data" such as stack traces. But end-user typically does not see these because the end-user would only be viewing the output such as a pdf-file. Therefore, even though this is

highly important and relevant, it is impossible to classify either as met or not met in the context of DocOrigin, it is ultimately the developers responsibility to not allow sensitive logs to the end-user to see, it seems rare almost irrelevant, therefore this requirement is classified as N/A. If this is looked from the perspective of an attacker getting access to the application, they will see the logs but they also has access to everything at that point.

Evaluation of V16 show that DocOrigin tries not to be a secure software. This is very alarming from the security aspects what the evaluation reveals. but the application itself is a powerful component that requires a highly skilled administrator to build a secure architecture around it. It is clear that the application on its own is not secure as it is not built security in mind. The application requires external hardening and a secure environment to work in; without this, the convenience of the software creates a significant gap in the overall security of the document generation pipeline.

4.4 Environment and Hardening: AS&D STIG

Like stated earlier, the specific AS&D STIG used in this study is the latest as of 17.3.2026, uploaded by the DoD Cyber Exchange Public DISA official website, is version 6, Release 4. It is also important to note here, that to view the actual STIG, one will need a designated software, a STIG Viewer, which is also provided by the DoD Cyber Exchange Public, and can be downloaded and installed from SRG and STIG Tools [44] suitable for one's specific OS. While it is also possible to view the raw data, usually XML or JSON, it is preferred and makes navigating the rules much more accessible in an intuitive graphical user interface, with added helpful functions, such as the search and sort functionality.

The AS&D STIG is very different from the OWASP ASVS. Both are essentially rule sets that can be used to evaluate the security of a software, but ASVS focuses on

application code and STIG focuses on the technical implementation and hardening of the application in the environment.

The ASVS evaluation was possible to conduct on DocOrigin, the area of investigation was clear. In the STIG evaluation, a environment needs to be specifically selected for the study. The environment refers to the server where the application (DocOrigin) is in production and where it operates. To define the environment, reference architecture is used based on Windows Server.

4.4.1 High severity CAT I rules

The **reference environment** is simply referred as **"RE"** in this subsection for convenience reasons. All rules are referred by using the **Group ID** as the identifier. It is important to note that Tomcat was confirmed not installed in the RE through a full recursive search of both `D:\DocOrigin` and the entire D: drive, no results for `tomcat-users.xml`. Therefore, findings that are based purely on DocOrigin Web Services (DOWS) behavior are not valid environment-level findings for this RE and are classified as Not Applicable.

V-222425: Open. The rule states that "The application must enforce approved authorizations for logical access to information and system resources in accordance with applicable access control policies." In practice, this requirement sets rules for what each user can do in the system and what information they are allowed to access. `icacls "D:\DocOrigin"` command ran in the command prompt inside the RE was used to confirm this. The relevant part of the output: `BUILTIN\Users:(I)(OI)(CI)(RX)` means all regular domain users have read and execute access to the entire DocOrigin directory. This includes every configuration file as well, since it was successfully proved in chapter 4.3 that DocOrigin stores configuration files containing credentials in the same directory structure as the application itself. This is a finding.

V-222432: Open. The rule states that "The application must enforce the limit of three consecutive invalid logon attempts by a user during a 15 minute time period." Essentially, the rule demands a lockout after three failed login attempts. The lockout threshold is 50 in the reference environment, not 3. This is a clear finding. This was revealed by running `net accounts` command in a command prompt inside the RE.

V-222536: Open. The rule states that "The application must enforce a minimum 15-character password length." The minimum password length in the RE is 12, not 15. This was also confirmed by the `net accounts` command. This is a clear finding.

V-222542: Open. The rule states that the application must only store cryptographic representations of passwords. As confirmed in the ASVS analysis in Section4.3 DocOrigin uses configuration files such as `.ini` and `.wjs` to store credentials in plain text. The plain text credential storage was confirmed by reviewing the documentation. Also, with the result in V-222425 added: regular domain users have read access to the DocOrigin directory, any user with access to the RE can read these credentials directly. Even though Tomcat is confirmed not installed in RE, these configuration files exist regardless of DOWS deployment and are part of the standard DocOrigin installation. This is a finding.

V-222585: Open. The rule requires the application to fail to a secure state when initialization, shutdown, or abort fails. Proven in the ASVS analysis in Section 4.3, a successful Billion Laughs attack caused the DocOrigin Merge engine to enter a persistent deadlock. The engine did not terminate cleanly, did not release resources, and required forced termination through the Task Manager. Even that was, in fact, not enough, as reboot was necessary for the system to start working properly again. This is not a controlled failure to a secure state. This is a finding.

V-222588: Open. This rule states that the "application must implement ap-

proved cryptographic mechanisms to prevent unauthorized modification of organization-defined information at rest on organization-defined information system components." Essentially cryptographic mechanisms must prevent unauthorized modification of information at rest. The `manage-bde` command was not recognized on the RE even when executed as administrator. This confirms that the BitLocker Drive Encryption feature is not installed on the RE. Basically all XML file, processed job data, configuration files containing credentials and log files are all stored unencrypted on the D: drive. This is a finding.

V-222589: Open. The rule demands appropriate cryptography to protect stored information. The same basis as in V-222588, and since BitLocker is confirmed not installed on the RE, therefore no data-at-rest encryption is in place for any data stored on this server. This is a finding. This was also confirmed `manage-bde` command not being recognized.

V-222596: Open. This rule states that the application must protect the confidentiality and integrity of transmitted information. Internal communication between FolderMonitor and the Merge engine operates by placing plain text XML files in shared directories on the server. This is the core data pipeline of the RE and it operates without any transport encryption. This is a finding.

V-222608: Open. The rule demands that the application is not vulnerable to XML-oriented attacks. As proven in the ASVS testing in Section 4.3, successful Billion Laughs exponential entity expansion attack was done. It caused a complete denial of service on the DocOrigin Merge engine, the engine went to a stage of a computational deadlock and could not recover without force quitting. The XML denial of service vulnerability is confirmed. This is a finding.

V-222609: Open. The rule demands that the application is not subject to input handling vulnerabilities. A total of three separate failures were proven in the ASVS testing. URL-encoded input was not normalized before processing (Test1),

no allow-list validation exists for language codes (Test2), and the trusted service layer performs no validation on incoming data values (Test3). The RE provides no compensating controls at the OS or network layer to filter input before it reaches the engine. This is a finding.

V-222612: Open. The rule states that the application can not be vulnerable to overflow attacks. In Windows Server environment, Windows Server Exploit Protection on Windows Server can be used to apply a suite of hardening techniques directly to the OS or individual applications, the main mitigation categories such as Address Space Layout Randomization (ASLR) and Data Execution Prevention (DEP) [45] was confirmed not set on the RE. Running `Get-ProcessMitigation -System` in PowerShell on the RE showed that all exploit protection settings are `NOTSET`. Therefore, no system-level exploit protection policy has been explicitly configured. This is a finding.

V-222642: Open. The rule states that the application must not have any embedded authentication data. This essentially includes all instances of sensitive data included in code, not just credentials, it could be for example certificates or configuration files such as global.asa [7]. As confirmed in the ASVS analysis in Section 4.3, DocOrigin documentation uses `admin/admin` as example credentials, and configuration files like `.ini` and `.wjs` files have plain text credentials on the directory structure. The V-222425 finding confirmed that all domain users have read access to the DocOrigin directory on the RE, this means that embedded authentication data in configuration files is also accessible to any authenticated user on the RE. This is a finding.

V-222643: Open. The rule demands that the application has the capability to mark sensitive or classified output when required. DocOrigin treats all input data identically and there is no built-in function to classify or mark data by sensitivity level. Custom implementations are possible but are not part of the default appli-

cation. This finding is particularly relevant given the defense sector deployment context for which this evaluation was commissioned. This is a finding.

V-222658: Open. The rule requires that all products are supported by the vendor or development team. DocOrigin itself is actively supported by Eclipse Corporation. However, the DocOrigin reference manual references external tool, more specifically OpenSSL version 1.1.1 for signing and encryption tasks. Like already mentioned, OpenSSL 1.1.1 reached end of life on September 11, 2023 and no longer receives security updates [42]. A component referenced in active production documentation that is no longer maintained is an unsupported dependency. This is a finding.

V-222659: Open The rule states that an application should be decommissioned when maintenance or support is no longer available. This finding follows directly from V-222658. OpenSSL 1.1.1 is past end of life and has not been replaced. Therefore the continued reference to an EOL component in active deployment documentation without a documented replacement makes this also a finding.

V-222430: Not A Finding. The rule simply demands that "The application must execute without excessive account permissions." The application here is DocOrigin, more specifically DocOriginFolderMonitor. DocOriginFolderMonitor runs as `SVC002896@company.com`, that is a dedicated domain service account, not `SYSTEM` or a local administrator. This is correct practice, not a finding. This was confirmed by `Get-WmiObject Win32_Service` command and filtering that was ran in PowerShell inside the RE.

V-222522: Not A Finding. The rule demands that the application uniquely identifies and authenticates organizational users. Access to the RE requires individual domain credentials through Citrix Workspace, and then one-time password (OTP) sent to the user's mobile device, and then separate Windows credentials for the Remote Desktop ICA (Independent Computing Architecture) sessions. Each

user is uniquely identified in this process. This is not a finding.

V-222543: Not Applicable. The rule demands that the application transmits only cryptographically protected passwords. This rule applies to network transmission of credentials by the application. Since Tomcat and DOWS are confirmed not installed in the RE, the application does not perform any network-based password transmission. Therefore rule is Not Applicable in this environment.

V-222554: Not Applicable. The rule demands that the application does not display passwords or PINs as clear text. It would require first a interactive user interface, where credentials are entered and displayed. Even though Windows security log in UI satisfies this, DocOrigin in this RE operates as a backend batch processing engine with no interactive login interface. This rule is Not Applicable.

V-222577: Not Applicable. The rule demands that the application does not expose session IDs. GUID-based Job IDs are a feature of DocOrigin Web Services. Since DOWS and Tomcat are confirmed not installed in the RE, no session IDs are generated or exposed at the environment level. This rule is Not Applicable.

V-222604: Not Applicable. The rule demands that the application protects from command injection. The command injection risk identified in the ASVS analysis in Section 4.3 is specific to the DOWS pass-through architecture, where arbitrary strings are passed to command-line execution without sanitization. Because, again DOWS is not deployed in the RE, this attack vector does not exist at the environment level. This rule is therefore Not Applicable.

V-222620: Not Applicable. The rule requires web servers to be on a separate network segment from application and database servers when operating in the DoD DMZ. The RE is a Finnish enterprise deployment, not operating in a DoD DMZ.

V-222662: Not Applicable. The rule states that default passwords must be changed. Since DocOrigin does not have any credential based login, Tomcat is the focus here, but as Tomcat was confirmed not installed on the RE through a full

recursive search of both `D:\DocOrigin` and the entire D: drive, which returned no results for `tomcat-users.xml`.

4.4.2 Medium severity CAT II rules

There is total of 168 remaining CAT II rules after the first two screening rounds. Because that amount is not feasible to go through one by one, again a mapping to groups strategy is used. And only the most significant findings in each group are shown here. Basically, the 168 rules are grouped into six functional groups or more specifically, clusters. Rules in each cluster that are DOWS specific are classified as Not Applicable, because Tomcat is already confirmed not installed in the RE.

The whole 168 remaining CAT II rules can be divided into groups as follows: Group A: authentication and password policy (41 rules), Group B: logging and audit (76 rules), Group C: access control and privilege management (16 rules), Group D: cryptography and transport security (11 rules), Group E: DoS mitigation, input handling, app resilience (13 rules), and Group F: configuration and availability (11 rules).

Group A: authentication and password policy (41 rules) This group is about session management, password policy, MFA requirements, and remote maintenance session security. Specific rules are classified as Not A Finding based on configuration evidence already found. MFA is confirmed as implemented through Citrix Workspace and on top of that, OTP authentication. This essentially fills the requirements of the rules V-222523, V-222526, V-222527 and V-222528. All of these rules are classified as Not A Finding, because they demand that, the application must use MFA for network access to privileged accounts (V-222523), also the same but for non-privileged accounts (V-222526), MFA is used for local access to privileged accounts (V-222527), and local access to nonprivileged accounts (V-222528). Because MFA is used for all accounts, all of these are not a findings.

The confirmed password history of 24 generations satisfies the requirement of the rule V-222546 (minimum of five generations). The `net accounts` command output shows this: "Length of password history maintained: 24". The `net accounts` command output also confirms two clear findings. The maximum password age is set to 360 days, fails the requirement of 60 days of the rule V-222545. The minimum password age is set to 0 days, this essentially allows immediate and repeating password changes, fails the required minimum password lifetime 1 day or 24 hours of the rule V-222544. Therefore V-222545 and V-222544 are findings.

Network Level Authentication (NLA) is confirmed enabled via registry inspection (`UserAuthentication = 1`), fills the requirement of V-222565 which requires strong authenticators for non-local maintenance session establishment. RDP SecurityLayer value 2 confirms that TLS is enforced for all remote connections to the RE. This fills the requirements of V-222562 and V-222563 which require cryptographic protection of integrity and confidentiality for non-local maintenance sessions. This is an important finding in the context of this deployment, since all administrative and configuration work on DocOrigin is performed remotely through these sessions.

The most significant finding in this group is about session timeout policy. Registry inspection using `Get-ItemProperty` on the RDP-Tcp key confirmed that both `MaxIdleTime` and `MaxDisconnectionTime` are set to 0. This means no idle timeout and no automatic disconnection is configured. Essentially admin RDP sessions to the RE can remain open regardless of idle or not. This fails V-222389 which requires automatic termination of non-privileged sessions after 15 minutes of inactivity, V-222390 which requires automatic termination of administrator sessions after 10 minutes of inactivity, and V-222566 which requires that all sessions and network connections are terminated when non-local maintenance is completed. The fact that there is no session timeout enforcement is a notable finding because the RE hosts sensitive document production data and configuration files containing credentials.

Rules about mutual SSL/TLS authentication for service-oriented applications (V-222534) and audit logging of non-local maintenance sessions (V-222561) are Not Applicable as DOWS is not installed in the RE. No application-level session audit mechanism exists for the RDP-based maintenance workflow.

Group B: logging and audit (76 rules) The largest group with 76 rules. The group is about audit record generation, content requirements, log protection and log management. Evaluation of this group is highly based on two already proved facts. DocOrigin stores log files as plain text `.log` files in the application directory structure, as confirmed in the ASVS V16 analysis in Section 4.3. Also, the permissions issue finding from V-222425 confirmed that all domain users have read and execute access to the entire DocOrigin directory, including all log files.

Rules V-222500, V-222501 and V-222502 require that audit information is protected from unauthorized read, modification and deletion. The logs are plain text files in a directory accessible to all domain users, all of these rules are clear findings. In addition, the rules stating that audit records are offloaded to a separate system (V-222481) and written to a centralized log repository (V-222482) are also findings (Open). Because DocOrigin logs stay on the same server in the same directory structure as the application itself by default. The rule that sensitive data must not be written into logs (V-222444) is also Open because DocOrigin has no data masking capability and logs all processed data without filtering.

DocOrigin does produce timestamped log entries with descriptive event labels, such as `Merge:JobStart` and `Merge:ProcessEnd`, which meet the rules V-222446 and V-222473. The application must record a timestamp when an event occurred (V-222446), which was not a finding. It is required to make audit records of when (date and time) the events occurred (V-222473), which was also not a finding. DocOrigin has valid logging in the time stamp point of view, but as proved in the Section 4.3, chapter V16, not in terms of logging the user or the location. Therefore, because

the log entries only record the OS-level service account and not the identity of the individual user who triggered the job, V-222449 demands to record the username or user ID of the user associated with the event, is a finding. Also, the rule V-222477 demands generating audit records with the identity of any individual or process associated with the event, is a finding, since the OS-level service account does not state anything about the user's identity.

Rules about session ID audit capability (V-222441, V-222442, V-222443, V-222445) were re-evaluated after it became clear that RDP and Citrix ICA administrative sessions to the RE are actually the non-local maintenance workflow for DocOrigin. Configuration inspection using `Get-WinEvent` confirmed that the Windows Security Event Log is actively recording session lifecycle events with unique IDs.

The following IDs are used by Microsoft and precisely in older versions of Windows, which Windows Server 2022 is. Auditing event-4624 is successful log on [46], event-4634 is An account was logged off, event-4647 User initiated logoff [47], event-4778 A session was reconnected to a Window Station, event-4779 A session was disconnected from a Window Station [48]. Using these with the `Get-WinEvent` command confirms that the rules V-222441 and V-222442 are classified Not A Finding at the OS level. No Citrix server-side components were found on the RE which confirms that session management is handled entirely through standard Windows RDP session infrastructure. Session ID renewal events (V-222443, event 4778) were not found any of in the sample output, so this rule remains Not Reviewed. Session timeout auditing (V-222445) cannot be confirmed as active because timeout enforcement itself is not configured so therefore this rule is also classified as Not Reviewed.

HTTP header logging (V-222447) is Not Applicable as DOWS is not installed in the RE and there is no HTTP interface. Transaction recovery logs (V-222479) are

Not Applicable as DocOrigin is not a transaction-based system.

Group C: access control and privilege management (16 rules)

This group is about flow controls, privilege separation and configuration access restrictions. The significant finding from V-222425 directly applies all across this group. The fact that all regular domain users have read and execute permissions to the entire DocOrigin directory. The rule stating that access restrictions are enforced for changes to application configuration (V-222511) is a finding, because the read and execute permissions. The rule that configuration and control files must not be stored in the same directory as user data (V-222626) is also a finding. This was confirmed in the ASVS analysis in the Section 4.3 where DocOrigin stores configuration files, scripts, credentials and data in the same directory structure.

Windows Server 2022 offers native process isolation as a default OS feature, meaning each process runs in its own execution domain. This fills the requirements of the rule V-222590 (isolate security functions from non-security functions) as well as rule V-222591 (must maintain a separate execution domain for each executing process). Making these two rules not findings. The rule stating that the user interface is physically or logically separated from data storage and management interfaces (V-222574) is Not Applicable, as DocOrigin operates as a backend processing engine with no user interface in the RE.

By running `Get-WmiObject Win32_Service | Where-Object {$_.State -eq "Running"} | Select Name, StartName | Sort Name` command, essentially listing all running Windows services in a clear list in the RE, is confirmed that there is a lot of other processes running in the RE. Other services, not just DocOrigin related, such as backup agents, monitoring tools and other business applications. This indicates that the server is a general-purpose machine rather than a dedicated DocOrigin server. The rule V-222635 states that applications designated as critical or high availability must not be hosted on a general-purpose machine shared with

other applications. This is currently pending and can not be classified. Whether DocOrigin is formally designated as critical or high availability is an organizational decision outside the scope of this technical evaluation; however, the configuration evidence confirms that the prerequisite condition is present. This was flagged as a finding pending organizational classification confirmation.

Group D: cryptography and transport security (11 rules)

This group is about encryption requirements for data in transit and at rest, FIPS validation and endpoint authentication. The BitLocker finding from V-222588 and V-222589 in the CAT I evaluation is directly relevant here. Therefore one clear finding that can be stated right away, is the rule to protect confidentiality and integrity of stored information (V-222587), it is a finding. Because no drive encryption is installed on the RE.

Remote access to the RE is done via RDP with SecurityLayer value 2, was confirmed by doing registry inspection. TLS is enforced for all remote sessions, that makes the rules V-222396 and V-222397 not findings, they require encryption and cryptographic integrity protection for remote access sessions. Internal communication between FolderMonitor and the Merge engine is done plain text XML files placed in shared directories on the server directory structure, which also makes the rules V-222597, V-222598 and V-222599 findings because they require cryptographic mechanisms to protect data confidentiality and integrity during transmission and reception.

DocOrigin application does not use FIPS-validated cryptographic mechanics for signing or hashing or protecting unclassified information. This makes also the rules V-222570, V-222571 and V-222572 findings. Rules that require FIPS-approved session identifier generation (V-222583) and encryption for key exchange (V-222641) are Not Applicable because DOWS is not installed in the RE.

Group E: DoS mitigation, input handling, app resilience (13 rules)

This group pertains to input validation, DoS protection, canonical representation, error handling, and application resilience. The findings from the ASVS penetration testing in Section 4.3 apply directly here at the environment level because the RE does not offer compensating controls at the OS or network layer for filtering or blocking potentially malicious input before it reaches the engine.

The used Billion Laughs attack confirms three rules as Open. The rule V-222593 requires XML-based applications to mitigate DoS attacks using XML filters, parser options or gateways, out of which none are used deployed in the RE pipeline. Also, V-222594 requires that the application restricts the ability to launch DoS attacks against itself. The deadlock that was successful proves this is a finding. The rule V-222667 requires that protections against DoS attacks are implemented, which is also a finding. The URL encoding bypass proven in Test1 confirms V-222605 as a finding. Also, the input validation failures proven in Test1 to test3 confirms V-222606 as a finding. No administrator alert mechanism for low resource conditions is in place, that confirms V-222668 as a finding. The RE is a single server deployment with no documented redundancy, this confirms V-222595 as a finding. The error message restriction rule V-222611 is Not Applicable because DocOrigin produces document output files and has no interactive error display surface to the end users.

Group F: configuration and availability (11 rules)

This group is about software update management, security function verification, backup and deployment configuration. The Windows Update check confirmed that the most recent hotfixes on the RE were installed on December 23, 2025, so almost four months before the time of this evaluation. And only three patches was in the recent history. This confirms rule V-222614 as Open because relevant security updates are not being applied on a current basis.

DocOrigin does not have an internal mechanism to verify its own security functions at startup, restart, or on a 30-day schedule. These confirm the rules V-222615

and V-222616 findings (Open). Rule V-222515 is not a finding because this evaluation is exactly what the rule demands.

One thing, the Rubrik Backup Service was confirmed to be running on the RE. This indicates that a backup solution is in place. The rule V-222638 demands data backup done in intervals in accordance with DoD policy, meaning for low risk applications at least weekly. And for medium risk applications at least daily. The actual interval of the data backups could not be verified. In addition, if the backup procedures follow a documented schedule, or whether the backups are stored offsite or in a fire-rated container (V-222639), and whether physical and technical protection of backup media is assured (V-222640) cannot be confirmed through configuration inspection alone. These rules are still classified as Not Reviewed after the evaluation. These are pending organizational verification, and it is not even feasible or in the scope of this study to address every single rule; these are good and worth looking into in future work.

The rule that unnecessary software components are removed after updates (V-222613) also remains Not Reviewed. Component lifecycle procedures are not visible from the server configuration.

4.4.3 Low severity CAT III rules

After the screening rounds, only 4 CAT III rules remain for evaluation out of the original 22. These are evaluated individually here because the number is small enough to handle one by one.

V-222647: Open. The rule requires that security verification tests are executed on system initialization and shutdown to confirm the system remains in a secure state. DocOrigin has no built-in self verification mechanism that runs at startup or shutdown. It is confirmed in the ASVS analysis in Section 4.3 that the engine processes whatever input it receives without any security checks. The Billion Laughs

attack also proved that the engine does not shut down cleanly. There is no documentation or configuration evidence of any startup or shutdown verification procedure in the RE. This is a finding.

V-222669: Open. The rule states that at least one application administrator has to be registered to receive update notifications or security alerts when automated alerts are available. Because there is no documented notification system in the DocOrigin configuration or documentation, this is open. The OpenSSL 1.1.1 end-of-life situation identified in V-222658 shows an example security update that has not been done anything about. This suggests no active alert mechanism in place. This is a finding.

V-222670: Open. The rule requires that the application provides notifications or alerts when product updates and security patches are available. DocOrigin does not have a built-in update notification mechanism. Update awareness is not being maintained actively, this connects directly to V-222669. This is a finding.

V-222672: Not A Finding. The rule requires that the application generates audit records when concurrent logons from different workstations occur. Confirmed in the Group B evaluation, Windows Security Event Log records logon events like the event-4624 that stands for successful logon, with the source network address for each session. This means in practice that concurrent logons from different IPs would produce completely different separate 4624 events with the corresponding workstation address. This makes them possible to do investigation on and they are therefore auditable. The OS-level audit infrastructure fills the requirements of this rule. This is not a finding.

4.5 Compliance Summary

The case study used two industry standards to evaluate DocOrigin and its reference environment successfully. The goal was not to produce full security assessment, a

line had to be drawn clearly to separate this work from consulting work, as this is academic work after all. The OWASP ASVS v5 evaluated the application itself with seven groups of selected ASVS chapters after screening. The AS&D STIG Version 6, Release 4 evaluated the deployment environment, a Windows Server 2022 Standard installation with DocOrigin in production, using all three severity categories after screening.

The key hypothesis, that DocOrigin is not built with security in mind in the first place, was carried through the whole case study, especially in the ASVS part where DocOrigin was the main focus area. This hypothesis, the "Dumb engine" idea turned out to be true, as even the official marketing of DocOrigin points to this idea.

The ASVS evaluation used a total of 39 Level 1 and Level 2 requirements in all of the ASVS chapters. It can be stated that the results clearly indicate that DocOrigin only meets very few requirements natively. The majority of the addresses requirements were classified as either Not Met or they were delegated entirely to the environment, the system administrator or the developers responsibility.

Those requirements that were classified as Met, in multiple cases these were actually met unintentionally and by design, because the evidence points to the no security in mind design very heavily and there are concrete examples in the official documentation that recommends the more convenient way, not the more secure way of implementing things. For example, the fact that there is no password hint, is due to the fact that there are no login UI in the first place, credentials are passed in plain text in the command line when using DOWS.

The most critical findings are the proven input handling failures (v1.1.1, v2.2.1, v2.2.2) as well as notable mention, the successful Exponential Entity Expansion Example, the Billion Laughs attack, as well as total absence of internal cryptographic controls. There were also proven plain text credential storing in configuration files

and a total lack of log protection. All of these support the key hypothesis that DocOrigin operates as a dumb engine that assumes a hardened and trusted surrounding environment.

The AS&D STIG evaluation started also with the screening of total of 286 rules. After two rounds of screening, 91 rules were classified as Not Applicable. That left total of 195 rules for evaluation, out of which 23 high severity CAT I rules were all evaluated individually, because these were considered the most important, given their high severity status. A total of 19 Open findings and 13 classified as Not Applicable, only two was classified as Not A Finding, 0 remained Not reviewed out of the CAT I rules. The CAT I evaluation result alone is significant as 19 out of 23 evaluated rules represent active findings against the reference environment. The most severe findings of the CAT I class is the total missing of drive encryption (V-222588, V-222589), plain text credential storage that is accessible to all domain users (V-222542, V-222425), the proven DoS vulnerability (V-222608), and the EOL OpenSSL dependency (V-222658, V-222659). The remaining medium severity CAT II and low severity CAT III rules show similar patterns but in wider scale, essentially the most significant additional findings were the absence of session timeout enforcement, not properly filling password policies, unprotected and non-centralized log storing. Also, it got clear that the reference environment server is not only a dedicated DocOrigin server, but more of a general purpose machine.

It is proven that ultimately the reference environment server does not offer the needed hardening and secure environment that are needed by the DocOrigin software set. ASVS identified weaknesses at the application level showed that DocOrigin is not secure by itself, it should be treated as a component that needs secure environment to work on. And, the STIG evaluation confirmed that the deployment environment does not compensate for these weaknesses. Essentially, there are no environmental controls that mitigate the application-level risks, and there are no

application-level controls that compensate for the environmental gaps. The ASVS evaluation showed a clear need for the STIG evaluation, but the STIG evaluation did not give the results that was needed, it only showed that the overall security posture is weaker than what the ASVS results needed.

The evaluation reveals a systematic pattern of security delegation that can be formalised as a shared responsibility model. DocOrigin provides the processing capability while delegating all security enforcement to the deployment environment. Where that environment does not compensate, no security exists at either layer. This model is not unique to DocOrigin. Any document generation engine designed as an integration component rather than a complete application will exhibit the same structural characteristic, making the findings of this study applicable not to just this specific deployment.

5 Discussion, Recommendations and Conclusion

5.1 Answers to Research Questions

RQ1: What security risks are present in the ERP-to-DocOrigin document production pipeline, and at which stages do they occur?

The risks are present at every stage of the pipeline. Chapter 3 provides the theoretical background of what the most dangerous threats are and why, which clarifies what to look for in the case study. When document generation is separated from the ERP system and turned into its own data pipeline, it creates a security gap. The document generating engine, in this case, DocOrigin, in the end of the pipeline receives data and trusts it completely. It does not validate, sanitize, or encode the incoming data on its own. Essentially, this means, and as it proved to be, the engine is oblivious to the data it receives, if anything in the pipeline before the engine is compromised or misconfigured, the engine will process whatever it receives.

The most critical risks discussed in the Chapter 3 connect well with the findings of the Chapter 4. XML injection and XXE are relevant because DocOrigin uses XML as its primary input format. Resource exhaustion by for example nested and malformed input is relevant because the Billion Laughs attack was successfully proven in the test environment.

Template injection and scripting risks are also relevant because DocOrigin uses JavaScript-based scripting in its own template building logic, functions such as filtering, or modifying the output based on input data, all of this logic is essentially implemented with this scripting logic.

Path traversal is also relevant because it was proven that configuration files, credentials, and data are stored inside the same directory structure as the application itself.

OS command injection is relevant because the DocOrigin Web Service translates incoming HTTP requests into locally executed commands.

Security risks are present throughout the entire process. Input validation failures are present at the beginning stage, specifically at the entry point where the ERP data reaches the engine. For example, sensitive business-critical XML data arrive at the document generation engine having traveled through a channel that may not be validated at all. This sensitive XML data will be in transit for an indefinite period until it reaches the engine. Although there are timeouts and other safeguards in place, this is a major risk zone; evaluating it is difficult because of the lack of concrete evidence, and the evaluation of this area remains highly hypothetical. In addition, the ERP system itself remains a "black box" deliberately and is outside the scope of this thesis; nevertheless, the data must travel from the ERP to the DocOrigin engine through some network channel, and that channel carries its own risks regardless of the ERP system used.

The exact transport protocol depends entirely on the ERP vendor and the integration design chosen by the organization. In practice, the most common scenarios are REST or SOAP calls over HTTP or HTTPS, which would mean the data would be moving on Port 80 or Port 443. Multiple possible scenarios exist, and it is impossible to fully address the stage. Nevertheless, the main point is that if the data travel unencrypted, which is entirely possible if HTTP or an unencrypted message

queue is used, the XML payload containing sensitive business data is readable by anyone with access to the network path. Given that the STIG evaluation confirmed that the reference environment server is not a dedicated machine and runs other services apart from DocOrigin, the network attack surface of that server is larger than it should be. It is realistic to assume that the port used for DocOrigin data intake is not isolated from other services running on the same host.

Apache Tomcat was confirmed not installed on the reference environment server, still, there are strong indicators that the full HTTPS configuration with proper certificate management is not enforced by the application itself, and is left entirely to the system administrator. This means the transport encryption decision, one of the most fundamental data-in-transit protections, is a manual configuration step that may or may not have been taken in any given deployment.

Scripting and template risks also require attention. They occur during the actual processing of the data. Credential and secret exposure risks operate at the deployment and storage levels rather than at a specific pipeline stage. The finding that all regular domain users have read and execute access to the entire DocOrigin directory is a good example: it is not tied to any single data flow step, but the risk is present in every operation until it is acted upon.

RQ2: How does DocOrigin perform against OWASP ASVS v5.0.0 and AS&D STIG v6r4 when evaluated as a production document generation system?

The ASVS evaluation covered 39 Level 1 and Level 2 requirements across seven grouped sections. DocOrigin meets very few of these natively. In fact, if the documentation is followed literally, the configurations are more likely to be insecure. The DocOrigin documentation works very well for experienced system integrators, system admins and developers in the field, but an inexperienced user could easily misconfigure something, because in places the manual should not be understood

literally, but used for its guidelines and examples.

Most ASVS v5 requirements were either Not Met or fully delegated to the environment, the system administrator, or the developer. In several cases where a requirement appeared to be met, it was unintentional. The absence of a login interface means there is no password hint field, but it also means credentials are passed in plain text on the command line, which is even worse, but seemingly "meets" the requirement.

The most directly proven failures came from input handling. Three requirements were confirmed Not Met by practical penetration testing: URL-encoded input was not decoded before use, arbitrary text got accepted in fields that should have an allow-list, and the merge engine used XML data values directly as file paths without any validation at the trusted service layer. The Billion Laughs attack also confirmed a failure in a concrete way, the application entered a non-recoverable deadlock state, required a forced kill, and showed 7 % persistent CPU load and a static 30 MB memory footprint. The engine did not reject the payload, and the deadlock persisted for hours, ultimately requiring a full reboot. Because the ASVS requirements were screened and the evaluation paid closer attention to those that were identified as the most critical ones, it is safe to say that the results were alarming.

It is important to understand the context here, because DocOrigin is clearly not built with security first in mind. Therefore, the outcome was somewhat predictable. DocOrigin is a component-style engine designed for integration into existing systems, so it will naturally miss many requirements that were written for full web applications. What is the actual value of this evaluation?

The evaluation results probably did not come as a surprise when evaluating against the latest application security standards by OWASP, but it was not the intended outcome. The commissioning organization was in need of one. Some of their customers are currently preparing proposals for the public sector and the

defence sector, including NATO-level engagements, and as part of those processes they are required to provide accurate and detailed documentation of the security posture of the systems they use, including DocOrigin. No vendor-provided security documentation existed for this purpose, so the evaluation had to be done from scratch.

The STIG results are where the evaluation gets more interesting, and arguably more alarming. DocOrigin not meeting ASVS requirements for input validation is expected and understandable given what the product is. The reference environment running on Windows Server 2022 Standard is one of the more secure Windows Server versions by default. It claims to be more secure than older versions, but as proven in the STIG evaluation, 19 active CAT I findings were identified on this platform, which means that the default protections were insufficient, and in some cases it looks like some of them may have been deliberately turned off or at least never verified after initial setup. For instance, drive encryption is not an obscure setting, but it is still not a default setting. It requires someone to have consciously decided not to enable it, or to have never checked.

DocOrigin and its reference environment did not perform well against the two standards, but the more meaningful finding is about where the responsibility actually lies. DocOrigin behaves exactly as it is designed to, it is a component, and components are not expected to be complete security solutions on their own. The environment that hosts it is expected to provide the security controls the component does not. In this case, that expectation was not met.

RQ3: What do the evaluation findings indicate about the security posture of DocOrigin-based architectures, and what are the implications for organizations using similar systems?

The most important conclusion of this work is that the dumb engine finding does not automatically mean that DocOrigin is an unusable product. DocOrigin

is a capable and powerful document production engine that is designed to be integrated into an existing production environment, not to stand alone as a complete security solution. The product is built with this assumption in mind, and the official documentation reflects it clearly. The problem is not the design choice itself. The problem is that organizations that decide to deploy a system like this need to actively recognize this assumption and take full responsibility for building the secure environment the engine requires.

This sounds simple, but in practice it often does not happen. The reference environment evaluated in this study showed that the DocOrigin software was deployed and running in a production setting, still the reference environment missed some basic security controls. Drive encryption was not in place. Credentials were stored in plain text and were accessible to all domain users. None of these issues are hard to notice if someone is specifically looking for them, but the evaluation proved that they had been there for some time without being acted upon. This is probably the most realistic takeaway from this study: the gap was not about a lack of knowledge, but about a lack of accountability. It is not unrealistic to assume that no one was responsible for making the DocOrigin environment secure, so it stayed as it was.

For organizations using DocOrigin or any similar document generation engine, there are a few concrete things the findings point to. The incoming data pipeline must be treated as untrusted from the engine's perspective. This means that input validation, technology-specific (in this case, XML) schema enforcement, and parser hardening must be performed at the infrastructure or middleware layer before the data reaches the engine. The engine will not do this itself, and assuming the ERP system takes care of it is exactly the kind of assumption that creates the security vacuum discussed in Chapter 3. The deployment environment must be actively hardened. The STIG framework is a practical tool for working through these issues systematically, and the fact that 19 out of 23 high-severity CAT I rules were active

findings suggests that the STIG had not been applied before this evaluation. An End-Of-Life (EOL) component, such as OpenSSL, will never receive a security patch, and the list of known vulnerabilities against it will not get any smaller. Every organization requires a formal dependency review process for any system that uses sensitive data.

For organizations using similar systems, there is no way around the fact that the environment needs to be built with security first in mind. A formal way of building a new production deployment environment is to use STIGs as the primary configuration standard. In this way, if the entire environment is built using the appropriate STIG, the environment has a good starting point. In addition, some checks and controls come built into the STIG process itself, such as the classification system of CAT I, CAT II, and CAT III rules that help prioritize what needs to be fixed first. Starting with all CAT I rules alone already gets an environment to a meaningfully more secure state than most default deployments.

That said, building with STIG is a justifiable starting point. But, "absolute security" is never met, in other words, there is no point where the environment reaches perfect security. Active and ongoing evaluations are still needed, and automated checks and regular dependency reviews can help in the process. Security should not be treated as a project; any organization that treats security as a project with a finish line will eventually find themselves in the same situation as the reference environment in this study.

5.2 Evaluation of the Standards

Both OWASP ASVS v5.0.0 and AS&D STIG v6r4 proved to be suitable and practical tools for this evaluation, but they were suitable in different ways and each came with its own challenges.

ASVS worked well as a structured framework for examining the application itself.

The requirement groupings made it possible to focus the evaluation on the most relevant security areas for a document generation engine, which was important because applying all 345 requirements equally would have been impractical. The screening and grouping approach described in Section 4.2.1 allowed the evaluation to remain focused while still producing meaningful coverage. The main challenge with ASVS in this context was the lack of vendor-provided security documentation from DocOrigin. Many requirements could not be confirmed as met or not met through documentation alone and required practical testing in the local environment. This is a genuine limitation of evaluating a closed commercial product.

The AS&D STIG worked well as a complement to ASVS because it is specifically designed for situations where no vendor-provided STIG exists. It gave the evaluation a structured way to assess the deployment environment independently of the application. The CAT I findings alone told a clear story without needing to go through all 195 evaluated rules in detail. The challenge with the STIG in this case was that some rules are written with software development lifecycle scenarios in mind rather than third-party application deployment, which required careful judgment during the screening phase to avoid producing irrelevant findings.

5.3 Limitations

The most significant limitation of this study is that the ERP system was deliberately kept out of scope and treated as a black box. This was the right call for keeping the work focused, but it means the first stage of the pipeline, the point where data is generated and initially sent from, was not evaluated at all. The transport channel between the ERP and DocOrigin was analyzed theoretically but not tested. That is a real gap, and in a production context it could be the most critical gap of all, since the data in transit is sensitive and the channel configuration varies entirely by deployment.

The scope of the thesis was on paper very clear, but turned out to be challenging to manage. The software (DocOrigin) that the OWASP ASVS v5 part of the case study focuses on, was not a single executable, or otherwise easily delimited with clear boundaries. The Apache Tomcat part which turned out to be only a thin wrapper, and not actually studied in depth, could have been easily scoped out in the planning phases of the thesis. But it was actually deliberately included because it was requested by the commissioning organization.

The reference environment was also difficult to scope. As there was no physical access to the servers, and certain details about the commissioning organization had to be kept confidential. It made the case study more difficult as remote access to the reference servers at any given time was not possible. The environment had concurrent session constraints.

The scope of the work also had to draw a clear line between security consulting work and academic work. It was not intended to deliver a full security assessment. A full evaluation of all 345 ASVS requirements was neither feasible nor the goal. The same applies to the STIG: only the most relevant rules were evaluated in detail, and a different evaluator might have prioritized different rules and arrived at a different picture.

Additionally, the STIG evaluation was done on one very specific reference environment. The results reflect that specific environment's configuration and cannot be generalized to all DocOrigin deployments. Organizations with different infrastructure, different Windows Server configurations, or different network setups will get completely different STIG results.

The available literature on document pipeline security specifically is very limited. Therefore creating a good theoretical background required careful source selection, but the identified attack classes connected directly to the empirical findings, which confirms the background was well-founded.

5.4 Conclusion

The thesis studied the security of the ERP-to-DocOrigin document production pipeline by first forming a good theoretical background of the document generation field, and then by using two industry-recognized security standards to evaluate the security of the selected software, DocOrigin, a professional document generation engine, as well as the scoped reference environment that represents a real production environment. The theoretical background built in Chapter 3 was proven accurate. The attack classes that were identified connected well with the practical findings in the case study in Chapter 4.

The central finding of the work, DocOrigin is a "dumb engine" is not to put it down, but an effort to try to describe it in layman terms. DocOrigin is a dumb engine, by design, not by accident. DocOrigin is not the problem clearly in this study. The problem is that the environment in this study did not fulfill the role that design choice requires. The application delegated security to the environment, and the environment was not hardened to receive that responsibility.

One of the goals is to generalize the findings to other CCM systems and organizations using document generation engines not limited to only DocOrigin. The dumb engine pattern is not unique to DocOrigin. Any document generation engine that is designed as a component to be integrated into an existing infrastructure will have the same fundamental characteristics. The engine will trust the data it receives, and it will trust the environment it runs in. OpenText Exstream, legacy systems like JetForm, and similar platforms all operate on the same architectural assumption. The key for organizations that deploy any external document generation engine is to treat the engine as a component that requires a secure and hardened environment.

A few natural directions for future work exist. Perhaps the most important direction would be to implement the mitigations discussed in the case study and re-evaluate the environment against the same STIG checklist. This should be done

as an iterative process, not a one-time assessment. Another direction would be to repeat the STIG evaluation on a reference environment where Apache Tomcat and DocOrigin Web Services are fully deployed. The DOWS-specific attack surfaces, such as OS command injection through the pass-through architecture, could not be tested here because Tomcat was not installed in the reference environment. Another direction is to test the transport channel between the ERP system and the engine. This was out of scope for this study, but it is a real security gap in any production deployment.

References

[1] R. Nurmeksela, "Implementing structured document production to support enterprise content management", Ph.D. dissertation, University of Jyväskylä, 2017, ISBN: 978-951-39-7210-3. [Online]. Available: https://urn.fi/URN:ISBN:978-951-39-7210-3.

[2] A. Koivula, *Centralised product label printing system: Survey and specification*, fi, Bachelor's thesis, Oulu, Finland, 2024. [Online]. Available: https://www.theseus.fi/handle/10024/858309.

[3] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering – a systematic literature review", *Information and Software Technology*, vol. 51, no. 1, pp. 7–15, 2009. DOI: 10.1016/j.infsof.2008.09.009.

[4] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, and R. Metz, *Reference model for service oriented architecture 1.0*, OASIS Standard, 2006. [Online]. Available: http://docs.oasis-open.org/soa-rm/v1.0/.

[5] A. L. Mesquida and A. Mas, "Implementing information security best practices on software lifecycle processes: The ISO/IEC 15504 security extension", *Computers & Security*, vol. 48, pp. 19–34, 2015. DOI: 10.1016/j.cose.2014.09.003.

[6] OWASP Foundation, *OWASP Application Security Verification Standard (ASVS) v5*, https://owasp.org/www-project-application-security-verification-standard, 2025.

[7] Defense Information Systems Agency (DISA), *Application security and development stig, version 6 release 4 (u_asd_stig_v6r4_manual-xccdf.xml)*, https://dl.dod.cyber.mil/wp-content/uploads/stigs/zip/U_ASD_V6R4_STIG.zip, 2026.

[8] N. Mittal, D. V. D, J. K. P, and P. Santhya, "Offline llm-based expert systems for confidential software documentation: Integrating ollama and json templates for regulatory adherence", in *2025 IEEE Space, Aerospace and Defence Conference (SPACE)*, Bangalore, India, 2025, pp. 1–5. DOI: 10.1109/SPACE65882.2025.11170588.

[9] G. Balakayeva, M. Zhanuzakov, U. Zhapbasbayev, and K. Nurlybayeva, "An intelligent enterprise system with processing and verification of business documents using big data and ai", *Journal of Intelligent Systems*, vol. 34, no. 1, p. 20 240 446, Sep. 2025. DOI: 10.1515/jisys-2024-0446.

[10] E. V. Chen and S. R. O'Connell, "Leveraging intelligent document automation for enhanced data integrity and compliance in the pharmaceutical industry", *Frontiers in Emerging Artificial Intelligence and Machine Learning*, vol. 2, no. 9, pp. 23–33, 2025, Accessed: Mar. 9, 2026. [Online]. Available: https://irjernet.com/index.php/feaiml/article/view/232.

[11] H. Hermawan, K. Shanmugam, and M. E. Rana, "Leveraging Artificial Intelligence in Enterprise Resource Planning (ERP): Enhancing Efficiency, Decision-Making, and Future Prospects", in *2025 International Conference on Metaverse and Current Trends in Computing (ICMCTC)*, Subang Jaya, Malaysia, Apr. 2025, pp. 1–6. DOI: 10.1109/ICMCTC62214.2025.11196219. Accessed:

Mar. 9, 2026. [Online]. Available: https://ieeexplore.ieee.org/document/11196219.

[12] F. Sadique, S. Cheung, I. Vakilinia, S. Badsha, and S. Sengupta, "Automated structured threat information expression (stix) document generation with privacy preservation", in *2018 9th IEEE Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, New York, NY, USA, 2018, pp. 847–853. DOI: 10.1109/UEMCON.2018.8796822.

[13] R. Ambati, "Secure dynamic check rendering: NetSuite's on-demand financial document generation system", *Journal of Engineering and Computer Sciences*, vol. 4, no. 9, pp. 130–144, 2025. DOI: 10.5281/zenodo.17079802.

[14] J. A. Chamberlain, "SAP-Integrated Large Language Models for Secure Cloud-Based Enterprise Analytics and Risk Detection", en, *International Journal of Engineering & Extended Technologies Research (IJEETR)*, vol. 5, no. 6, pp. 7567–7574, Dec. 2023. DOI: 10.15662/IJEETR.2023.0506014. Accessed: Mar. 9, 2026. [Online]. Available: https://www.ijeetr.com/index.php/ijeetr/article/view/224.

[15] A. P. Mishra, M. Dublish, and D. Kumar, "Cyber security application in ERP implementation", *Journal of Pharmaceutical Negative Results*, vol. 13, no. S06, pp. 2507–2522, 2022. DOI: 10.47750/pnr.2022.13.S06.325.

[16] S. Yadav, "HYBRID CLOUD STRATEGIES FOR SAP ERP MODERNIZATION: BRIDGING S/4HANA AND LEGACY SYSTEMS", en, *International Journal of Applied Mathematics*, vol. 38, no. 3s, pp. 1114–1129, Sep. 2025, ISSN: 1314-8060. DOI: 10.12732/ijam.v38i3s.207. Accessed: Mar. 9, 2026. [Online]. Available: https://ijamjournal.org/ijam/publication/index.php/ijam/article/view/207.

[17] N. T. Lapatta and Syahrullah, "Architectural Analysis of the Repository Pattern in Web-Based Credit Score Conversion Assessment System Based on PermenPAN-RB No. 1 of 2023", en, *Tech-E*, vol. 9, no. 2, pp. 192–205, Feb. 2026, ISSN: 2581-1916. DOI: 10.31253/te.v9i2.4362. Accessed: Mar. 9, 2026. [Online]. Available: https://jurnal.buddhidharma.ac.id/index.php/te/article/view/4362.

[18] D. Fucci, M. Di Penta, S. Romano, and G. Scanniello, "Augmenting Software Bills of Materials with Software Vulnerability Description: A Preliminary Study on GitHub", in *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, Trondheim, Norway: Association for Computing Machinery, Jul. 2025, pp. 631–635, ISBN: 979-8-4007-1276-0. Accessed: Mar. 9, 2026. [Online]. Available: https://dl.acm.org/doi/10.1145/3696630.3728513.

[19] T. Mens and A. Decan, *An Overview and Catalogue of Dependency Challenges in Open Source Software Package Registries*, Nov. 2024. DOI: 10.48550/arXiv.2409.18884. Accessed: Mar. 9, 2026. [Online]. Available: http://arxiv.org/abs/2409.18884.

[20] N. S. Filho, "Questpdf and .net: Architectural patterns for document generation in distributed systems", *Zenodo*, vol. 3, pp. 1–6, Jan. 2026. DOI: 10.5281/zenodo.18458517.

[21] A. Chiş, O. I. Stoica, A.-M. Ghiran, and R. A. Buchmann, "A Knowledge Graph Approach to Cyber Threat Mitigation Derived from Data Flow Diagrams", in *2024 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, Cluj-Napoca, Romania, May 2024, pp. 1–6. DOI: 10.1109/AQTR61889.2024.10554074. Accessed: Mar. 16, 2026. [Online]. Available: https://ieeexplore.ieee.org/document/10554074.

[22] S. Zareen, A. Akram, and S. A. Khan, "Security requirements engineering framework with BPMN 2.0.2 extension model for development of information systems", *Applied Sciences*, vol. 10, no. 14, p. 4981, 2020. DOI: 10.3390/app10144981.

[23] D. Basile, V. Goretti, L. Barbaro, H. A. Reijers, and C. Di Ciccio, "A TEE-based approach for preserving data secrecy in process mining with decentralized sources", *Journal of Information Security and Applications*, vol. 98, p. 104 381, May 2026, ISSN: 2214-2126. DOI: 10.1016/j.jisa.2026.104381. Accessed: Mar. 15, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212626000116.

[24] L. Hermerschmidt, S. Kugelmann, and B. Rumpe, "Towards more security in data exchange: Defining unparsers with context-sensitive encoders for context-free grammars", in *2015 IEEE Security and Privacy Workshops*, San Jose, CA, USA: IEEE, May 2015, pp. 134–141. DOI: 10.1109/spw.2015.29. [Online]. Available: http://dx.doi.org/10.1109/SPW.2015.29.

[25] F. F. Fadlalla and H. T. Elshoush, "Input validation vulnerabilities in web applications: Systematic review, classification, and analysis of the current state-of-the-art", *IEEE Access*, vol. 11, pp. 40 128–40 161, 2023. DOI: 10.1109/ACCESS.2023.3266385.

[26] H. Zhang and J. Shang, "Multiformat document parsing and management", in *Natural Language Processing and Applications*. Singapore: Springer Nature Singapore, 2025, pp. 115–132, ISBN: 978-981-97-9739-4. DOI: 10.1007/978-981-97-9739-4_6. [Online]. Available: https://doi.org/10.1007/978-981-97-9739-4_6.

[27] H. Brodin, E. Sultanik, and M. Surovič, *Blind spots: Automatically detecting ignored program inputs*, 2023. DOI: 10.48550/arXiv.2301.08700. arXiv: 2301.08700. [Online]. Available: https://arxiv.org/abs/2301.08700.

[28] C. Späth, C. Mainka, V. Mladenov, and J. Schwenk, "SoK: XML parser vulnerabilities", in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, Austin, TX: USENIX Association, Aug. 2016. [Online]. Available: https://www.usenix.org/conference/woot16/workshop-program/presentation/spath.

[29] A. Pakki and K. Lu, "Exaggerated error handling hurts! an in-depth study and context-aware detection", in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1203–1218, ISBN: 9781450370899. DOI: 10.1145/3372297.3417256. [Online]. Available: https://doi.org/10.1145/3372297.3417256.

[30] F. Blefari, C. Cosentino, A. Furfaro, F. Marozzo, and F. A. Pironti, "Secflow: An agentic LLM-based framework for modular cyberattack analysis and explainability", in *CEUR Workshop Proceedings*, vol. 4075, 2025. [Online]. Available: https://ceur-ws.org/Vol-4075/paper4.pdf.

[31] V. Iatsiuta, V. Kobets, and O. Ivanov, "Creating of a general purpose language for the construction of dynamic reports", in *Digital Transformation*, J. Maślankowski, B. Marcinkowski, and P. Rupino da Cunha, Eds., Cham: Springer Nature Switzerland, 2023, pp. 16–37, ISBN: 978-3-031-43590-4.

[32] K. Yamazaki, D. Kotani, and Y. Okabe, "Xilara: An XSS Filter Based on HTML Template Restoration", in *Security and Privacy in Communication Networks*, R. Beyah, B. Chang, Y. Li, and S. Zhu, Eds., Cham: Springer

International Publishing, 2018, pp. 332–351, ISBN: 978-3-030-01704-0. DOI: 10.1007/978-3-030-01704-0_18.

[33] N. Fleury, T. Dubrunquez, and I. Alouani, *PDF-malware: An overview on threats, detection and evasion attacks*, arXiv preprint arXiv:2107.12873, 2021. DOI: 10.48550/arXiv.2107.12873.

[34] V. Sachidananda, S. Bhairav, and Y. Elovici, "Over: Overhauling vulnerability detection for iot through an adaptable and automated static analysis framework", in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ser. SAC '20, Brno, Czech Republic: Association for Computing Machinery, 2020, pp. 729–738, ISBN: 9781450368667. DOI: 10.1145/3341105.3373930. [Online]. Available: https://doi.org/10.1145/3341105.3373930.

[35] S. Kumi, C. Lim, S.-G. Lee, Y. O. Oktian, and E. N. Witanto, "Automatic detection of security misconfigurations in web applications", in *Proceedings of International Conference on Smart Computing and Cyber Security*, P. K. Pattnaik, M. Sain, A. A. Al-Absi, and P. Kumar, Eds., Singapore: Springer Singapore, 2021, pp. 91–99, ISBN: 978-981-15-7990-5. DOI: 10.1007/978-981-15-7990-5_8.

[36] M. A. Lazo Moscosos, P. E. Sánchez Anccori, and W. N. Choquehuayta, "Hardening to improve the security of financial institutions", in *2024 IEEE XXXI International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*, Lima, Peru, 2024, pp. 1–7. DOI: 10.1109/INTERCON63140.2024.10833467.

[37] M. Alahmad, A. Alkandari, and N. Alawadhi, "Survey of os command injection web application vulnerability attack", *Journal of Engineering Science and Technology*, vol. 17, no. 1, pp. 75–84, 2022. [Online]. Available: https:

//jestec.taylors.edu.my/Vol%2017%20Issue%201%20February%20%202022/17_1_6.pdf.

[38] A. Ahmad, "Operationalizing owasp asvs level 1 in ci/cd: Reproducible security testing using one open source and one industrial case study", M.S. thesis, Tampere University, 2025. [Online]. Available: https://trepo.tuni.fi/handle/10024/233001.

[39] Defense Information Systems Agency (DISA), *STIGs document library*, DoD Cyber Exchange, Accessed: Mar. 17, 2026. [Online]. Available: https://www.cyber.mil/stigs/downloads.

[40] Eclipse Corporation, *Eclipse – premier document generation software for businesses*, Accessed: Mar. 16, 2026. [Online]. Available: https://eclipsecorp.us/.

[41] Eclipse Corporation, *DocOrigin reference manual*, Accessed: Mar. 18, 2026. [Online]. Available: https://docs.docorigin.com/reference-manual.

[42] OpenSSL Corporation, *OpenSSL 1.1.1 end of life*, Accessed: Apr. 2, 2026. [Online]. Available: https://openssl-corporation.org/post/2023-09-11-eol-111/.

[43] curl contributors, *Curl – how to use*, Accessed: Apr. 1, 2026. [Online]. Available: https://curl.se/docs/manpage.html.

[44] Defense Information Systems Agency (DISA), *SRG and STIG tools*, DoD Cyber Exchange, Accessed: Mar. 2026. [Online]. Available: https://www.cyber.mil/stigs/srg-stig-tools.

[45] Microsoft, *Apply mitigations to help prevent attacks through vulnerabilities*, Microsoft Defender for Endpoint | Microsoft Learn, Accessed: Apr. 12, 2026. [Online]. Available: https://learn.microsoft.com/en-us/defender-endpoint/exploit-protection.

[46] Microsoft, *Audit logon*, Microsoft Learn, Accessed: Apr. 13, 2026. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/audit-logon.

[47] Microsoft, *Audit logoff*, Microsoft Learn, Accessed: Apr. 13, 2026. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/audit-logoff.

[48] Microsoft, *Audit other logon/logoff events*, Microsoft Learn, Accessed: Apr. 13, 2026. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/audit-other-logonlogoff-events.