

Tools4LEAs |

A project of the European Anti-Cybercrime Technology Development Association
(EACTDA)



D3.17 Tools packaging, release and delivery handbook



Co-funded by
the European Union

Version:	1.0	
Delivery date:	October 2023	
Dissemination level:	Public	
Status	FINAL	
Nature:	Report	
Main author(s):	Jon Elduayen	EACTDA
Contributor(s):	Miguel Angel Blanco	EACTDA
	Laura Peralta	EACTDA
	Juan Arraiza	EACTDA

DOCUMENT CONTROL

Version	Date	Author(s)	Change(s)
0.1	14/09/2023	Juan Arraiza (EACTDA)	TOC and initial text
0.2	28/09/2023	Jon Elduayen (EACTDA) Miguel Ángel Blanco (EACTDA) Laura Peralta (EACTDA)	First version for all sections completed.
0.3	09/10/2023	Jon Elduayen (EACTDA)	Updated with received feedback
1.0	25/10/2023	Jon Elduayen (EACTDA)	QA review and final version, ready to be submitted

TABLE OF CONTENTS

1. Introduction	4
1.2. Overview of the Tools4LEAs project	4
1.3. Main objective of this document	4
1.4. Relation to other deliverables	5
1.5. Structure of the deliverable	5
2. Packaging of tools	6
2.1. Definition	6
2.2. Processes involved	6
2.2.1. Compilation	6
2.2.2. Build documentation and packaging	10
2.3. Specificities within the Tools4LEAs project	13
3. Release of tools	16
3.1. Definition	16
3.1.1. Release preparation	23
3.1.2. Release execution	28
3.2. Specificities within the Tools4LEAs project	32
4. Delivery of tools	33
4.1. Definition	33
4.2. Processes involved	33
4.2.1. Delivery planning	33
4.2.2. Delivery preparation	36
4.2.3. Delivery execution (or release distribution)	39
4.3. Specificities within the Tools4LEAs project	44
5. Summary	45
5.1. Conclusion	45
5.2. Evaluation	45
5.3. Future work	45
ANNEX I – Overview of the SbomIPRChecker	46
ANNEX II – Software signing	47
ANNEX III – Overview of EACTDA’s repository of tools	48

1. Introduction

1.2. Overview of the Tools4LEAs project

EACTDA is the acronym of the European Anti-Cybercrime Technology Development Association, which is a private non-profit association, established in San Sebastian, Spain. The members of the Association include European Union (EU) public entities fighting cybercrime, universities and research technology organisations, for-profit private companies, and other relevant actors in the field of the EU security research and innovation.

The Tools4LEAs projects are a series of projects that receive a Direct Award under the ISFP programme, and which main goal is to facilitate and promote the uptake of innovative technologies by EU public entities fighting cybercrime. EACTDA, via the Tools4LEAs projects, aims at further developing pre-existing assets, mainly from EU-funded security research and development projects, so that they are offered with no license cost and with access to the source code to EU public entities fighting cybercrime.

In the first Tools4LEAs project (v1; Jul'21 to Jun'23), the focus was on designing and setting up the infrastructures, processes, and governance / decision-making mechanisms, whilst delivering the first set of “fully-tested and operational-ready” tools via Europol’s Tool Repository. Though 11 tools were further developed in the v1 project, it is expected that 3 of them will not be released to their targeted audience as they do not pass the pre-established quality threshold of “operational-ready”. Also, an End-User Advisory Board (EUAB) composed as of Jul'23 by 23 members from 14 EU member states and co-chaired by two Europol units (EC3 and Innovation Lab) was established and it is the body responsible for identifying and prioritising end-user needs and which has veto right over the decisions done by EACTDA/Tools4LEAs with regard to the tool development roadmap.

In the second Tools4LEAs project (v2; Jul'23 to Jun'25), it is proposed to double the number of tools delivered. Also, the repository of tools implemented in v1, and currently used to host the results of the Tools4LEAs projects, will be enhanced and reused to host the results of EU-funded security research projects (when relevant in the field of cybercrime). EACTDA will play the role of custodian of these results, and the technical, IPR, and administrative aspects needed to create this new repository of security research results will be put in place. In addition, the v2 project will include a pilot to proof the concept of initial and limited support&maintenance periods for a selection of tools. Besides, a pilot of the concept of EACTDA National Nodes (NN) will be included, with nodes planned in Lithuania, France, Spain, and maybe one or two additional ones. Also, a platform for end-users to evaluate online tools will be implemented. Finally, the v2 project will include activities to further build the community of Tools4LEAs stakeholders and to promote the creation and/or adoption of technical blueprints, and in general, of commonly accepted best practices.

1.3. Main objective of this document

This deliverable is part of Task 3.9 of the Tools4LEAs-v2 project, which definition is as follows:

T3.9 - Tools packaging, release and delivery

This task includes all the activities related to packaging, releasing and delivering (distributing) the tools after they have been successfully demonstrated and evaluated

The main objective of the deliverable is to serve as a handbook for EACTDA Secretariat staff as well as for those EACTDA members participating in “last-mile” development projects as seconded personnel to EACTDA when working on the packaging, release, and delivery of software products.

In addition, the deliverable has been marked as public, so that it is shared openly with the community. By doing it so, it is intended to provide greater visibility of the EACTDA DevSecOps practices, particularly on these three aspects (packaging, release, and delivery of software products), and including considerations that are specific to the software that is produced with the public entities fighting cybercrime as their main targeted end-users.

1.4. Relation to other deliverables

This deliverable is closely related to the following deliverables:

- **D3.18. Report on tools packaging, release and delivery activities:** The activities reported in D3.18 should reflect the considerations presented in this deliverable D3.17.
- In addition, **all WP3 deliverables are related to this deliverable**, in the sense that they all refer to different phases and activities within the DevSecOps cycle of the “last-mile” development projects that are part of the Tools4LEAs-v2 project.

1.5. Structure of the deliverable

Section 2 of this document defines the Packaging phase of the DevSecOps cycle, it presents the processes involved in that phase, and it concludes with some specificities that apply to the Tools4LEAs-v2 project.

Section 3 of this document defines the Release phase of the DevSecOps cycle, it presents the processes involved in that phase, and it concludes with some specificities that apply to the Tools4LEAs-v2 project.

Section 4 of this document defines the Delivery phase of the DevSecOps cycle, it presents the processes involved in that phase, and it concludes with some specificities that apply to the Tools4LEAs-v2 project.

Finally, section 5 summarises which is the goal and key aspects of this document, it acknowledges that there is still work to be done to improve the document, and it presents some of the areas of future work that have already been identified.

2. Packaging of tools

2.1. Definition

This phase includes building and integration of the different components of the software system/solution being developed. This is a phase that it repeats itself each time that a development team prepares a computer executable of its software to the development team, so it is a phase that is usually automated as much as possible. The processes within this phase are compilation, which is the process of converting source code files into standalone software artefact(s) that can be run or executed on a computer, and the build documentation (or creation of the build notes).

When this phase is automated, it is common to include a testing process to ensure the quality of the final product before it is released, this process includes two parts of testing known as static testing and dynamic testing. Static testing is an approach to testing that consists of performing an analysis of the product's source code using a static code analysis tool to identify its security flaws and possible vulnerabilities without executing it. On the other hand, dynamic testing is an approach to testing that consists of performing tests by executing the source code of the product; in a build automation, this approach is taken by running automated tests belonging to different test phases (e.g., integration, regression and/or smoke tests) on test environment(s) that are prepared as much as possible to resemble the real scenario where the software is also "packaged", "released", "delivered" and "deployed".

2.2. Processes involved

2.2.1. Compilation

	DevOps aspects	Security considerations	Fighting cybercrime domain considerations
Description	Compilation in software development means to transform a program written in a high-level programming language from source code into object code or machine code that is used to create the executable programme that is compliant with its targeted platform/machine.	The compilation process itself can raise multiple security issues when transforming the source code from its high-level language to a lower-level one (object/machine code). There is an emerging area of research around the concept of secure compilation. Secure compilation aims to protect high-level language abstractions in compiled code, even against adversarial low-level contexts, and to allow sound reasoning about security in the source	-

	<p>Also, note that in the context of a CI/CD pipeline testing activities can be linked to the build process (i.e., smoke test).</p>	<p>language by: (1) identifying and formalizing properties that secure compilers must possess; (2) devising efficient enforcement mechanisms; and (3) developing effective formal verification techniques.</p> <p>Also, the source code as well as the object/machine code can be analysed to detect vulnerabilities. Static Application Security Testing (SAST) is a technique that focuses precisely on these matters. Scans of the code can be done early in development, aiding the developers to identify problematic code locations and suggesting solutions. These scans do not require executing the code, and they can be easily automated. However, there are some drawbacks such as for example that it is important to keep an eye on false-positives. Also, SAST has a strong dependency on the source code programming language. And it is also important to note that not all type of vulnerabilities can/are detected with this technique.</p>	
Roles & responsibilities	Developers are responsible for this process.	Testers and developers are responsible for this process.	-
Inputs	<ul style="list-style-type: none"> • The source code • The software versioning policy (and previous build's identifier) 	<ul style="list-style-type: none"> • The source code 	-
Outputs	<ul style="list-style-type: none"> • The object code or machine code that is used to create the executable programme. 	<ul style="list-style-type: none"> • Static code analysis report 	-

Benefits	An executable programme, which is the output of the compilation process can provide the following benefits: <ul style="list-style-type: none"> • Better performance. • Reduced system load. • Protection by obscurity of the source code of the programme. • Better portability of compiled programs (within the targeted/supported platform/s). 	A static code analysis report gives as benefits: <ul style="list-style-type: none"> • Better understanding of the source code. • Early defects detection. • Cheap to conduct it. • Helps to reduce the technical debt of a software product. 	-
Exchange data formats	<ul style="list-style-type: none"> • Executable formats (exe, app, binary executable scripts, ...) • Software library formats (dll, o-files, dylibs, ...) • Build file (YAML, jenkinsfile, dockerfile, makefile, ...) 	<ul style="list-style-type: none"> • Plain text structured files (XML, CSV, JSON, ...) 	-
Taxonomies, ontologies, ...	<ul style="list-style-type: none"> • <u>Dynamic compilation</u>: This process translates the code to machine code at runtime or before executing it. It is commonly used in JIT compilation and dynamic language environments. • <u>Static compilation</u>: This process translates the code to machine code at compile time, creating an executable binary file. 	<ul style="list-style-type: none"> • <u>Code smells</u>: A code smell is known as a characteristic that it can indicate a deeper problem in the source code. What or what not it is a code smell varies depending on the language, the developer or the development methodology. • <u>Static testing</u>: Method of debugging where the source code is automatically examined without having to execute it to assure that it is compliant, safe, and secure. 	-
Standards	<ul style="list-style-type: none"> • IEEE 2675-2021 IEEE Standard for DevOps: Building Reliable and Secure Systems Including 	-	-

	Application Build, Package, and Deployment		
Techniques	<ul style="list-style-type: none"> Continuous Integration (CI) and Continuous Delivery (CD), also known as CI/CD, to support automation in building, testing and deployment of applications. 	<ul style="list-style-type: none"> SAST: There are three basic types of SAST testing: source code analysis, byte code analysis, and raw binary code analysis. SAST security solutions can be integrated directly into the development environment, allowing developers to fix issues before they pass into the next phase of the SDLC. Secure Compilation 	-
Tools	Type or category of tools	Key features	Commonly used open-source tools
	SAST tools	Analyse the source and/or the object code to detect known vulnerabilities and to propose fixes	SonarQube ; MegaLinter
	Vulnerabilities and dependencies tracking	Vulnerability detection, full-stack inventory, impact analysis, policy evaluation, outdated version detection,	Dependency Track ; Syft ; Grype
	Compiling tools	Automatization system to build mobile apps, microservices and software from source code, Task runners	Gradle ; Apache Maven ; PyBuilder ; Grunt

2.2.2. Build documentation and packaging

	DevOps aspects	Security considerations	Fighting cybercrime domain considerations
Description	<p>When a new build is created it has to be prepared to be transferred to the testing processes. This requires documenting and packaging the build in a way that it includes all the necessary elements for those that are going to test it.</p> <p>A build identifier, the source code (including dependencies and libraries), and the executable programme are three elements that must be part of the build package. In addition, there are other elements which can be provided, such as:</p> <ul style="list-style-type: none"> • The build notes (specifying what is new and/or what has changed from previous builds, which are the known issues/errors that have not been fixed yet, etc.) • Updated versions of the installation and/or user manuals • Test reports (including security related, such as SAST test reports, and other type of tests, such as unit 	<p>Creating a build for a piece of software has a number of security issues that need to be covered in order to ensure that the final package does not contain something that should not be distributed. Therefore, during this process, all components of the package must be identified to avoid further problems.</p> <p>The development team is mainly responsible for this task, with the support of the testing team to check the contents in case something sensitive could be released that has not passed the security filters in place. Among the elements that can be identified as sensitive, it can find:</p> <ul style="list-style-type: none"> • Connection strings of services used for development and stored in environment files or hard-coded in the source code. • Test or any kind of data related to individuals or real entities. • Modules of the source code that must not be shared because of an IPR agreement or other security reasons given by the development team. 	-
Roles & responsibilities	<ul style="list-style-type: none"> • The Developers are responsible for this process. 	<ul style="list-style-type: none"> • The Developers and Testers are responsible for this process. 	-

	<ul style="list-style-type: none"> • Testers can contribute by defining their needs. 		
Inputs	<ul style="list-style-type: none"> • Source code • Executable programme 	-	-
Outputs	<ul style="list-style-type: none"> • Build package, including: <ul style="list-style-type: none"> ○ Build identifier ○ Source code ○ Executable programme ○ Build notes ○ Installation and/or user manuals ○ Test reports 	-	-
Benefits	<p>Properly documenting and packaging builds so that they are prepared to be ingested by other processes (i.e., testing) that depend on the build as an input can make this transition much efficient and less error prone.</p>	<p>Taking all the necessary precautions to assure that the final package is not prepared with something sensitive brings as the main benefit that this problem does not involve a higher cost on the future to mitigate.</p>	-
Exchange data formats	<ul style="list-style-type: none"> • Standard document formats (.pdf, .docx, ...) • Plain text formats (.md, .txt, .xml, ...) • Compressed file formats (.zip, .tar.gz, ...) • Video formats (.mkv, .mp4, ...) 	<ul style="list-style-type: none"> • Environment variables file (.env) • Ignore files/folders project control file (.gitignore) 	-
Taxonomies, ontologies, ...	-	-	-
Standards	<ul style="list-style-type: none"> • IEEE 2675-2021 IEEE Standard for DevOps: Building Reliable and 	-	-

	Secure Systems Including Application Build, Package, and Deployment		
Techniques	Software versioning to assign either unique version names or unique version numbers to unique states of computer software, such as a new build. There are multiple different version number schemes, being one of the most commonly used ones the “semantic scheme”, which uses a three-part numbering system: major.minor.patch (e.g. 1.2.3) which may or may not be suffixed with special identifiers such as -alpha or -rc1.	<ul style="list-style-type: none"> • Use of control files (e. g. gitignore) which excludes resources that are sensitive or unnecessary. • Use of an environment file not shareable for environment variables commonly sensitive (e. g. development database connectionstring, email server credentials) • Reviewing existing documentation related to sensitive parts that should not be shared (e. g. test data related to individuals, modules protected by security reasons). 	-
Tools	Type or category of tools	Key features	Commonly used open-source tools
	Documentation generation tools	Extracting and structuring source code documentation, large formats exporting (HTML, PDF, MSWord...), large most common programming languages supported, automatic generated UML diagrams, fully customisable, extensions support, cross-platform, static-site generator	Doxygen
	Static-site generator tools	Engines that use text input files (such as Markdown, reStructuredText, and AsciiDoc) to generate static web pages. Static sites generated by static site generators do not require a backend after site generation.	Jekyll ; NextJS ; MkDocs ; Docusaurus

2.3. Specificities within the Tools4LEAs project

In this case, in addition to the techniques and recommendations discussed above, we comment here on the matter according to the EACTDA's caustics, identifying which important elements would be indispensable to prepare and how. For a final package of a tool according to EACTDA, the following checklist of items should be taken completely into account (unless otherwise stated) in order to proceed later with the release of the tool.

Item	Sub-item	y/n	Observations
Documentation	RELEASE notes		Use the (TO BE CREATED) template
	Installation manual		
	User manual		
	Technical Specifications		
	Technical Testing Report		
	UAT report		
Training materials	Short presentation video		
	Training course		
Test/evaluation data	Test data		
	Test data manual		Describe the content, how it has been created, ...
Installation files			If no executable, add a README.txt. It is crucial to use certificates that sign the artifact to get trusted, whether the application is a desktop or web application. (See ANNEX II)
Source code	Tool's source code		
	Automated test scripts		
License			ETR license (from EACTDA to Europol)
SBOM	SBOM file		XML or JSON in CycloneDX format.
	SBOM Security analysis report		PDF report of an internal use tool (e.g., DependencyTracker).
	SBOM IPR analysis report		PDF report of an internal use tool called S bomIPRChecker (see ANNEX I)

Table 1. Tool package items checklist table

A final software package, according to EACTDA, is a compressed file (split or not) or a folder containing all the elements previously presented in folders structured according to their contents and presented with the tool. In order to package these items, the following folder structure should be followed, as shown in the figure below.

```
C:\Windows\System32\cmd.e x + v
C:.\
├── ToolName_vToolVersion (FINAL)
│   ├── 01_Documentation
│   │   ├── RELEASE notes.pdf
│   │   ├── Tools4LEAs_Tech Specs_ToolName_v1.0.pdf
│   │   ├── Tools4LEAs_Technical-Testing-report_(PROJ-00XX-ToolName)_Jan01_v1.0.pdf
│   │   ├── Tools4LEAs_ToolName_Installation-manual_v1.0.pdf
│   │   ├── Tools4LEAs_ToolName_User-manual_v1.0.pdf
│   │   └── Tools4LEAs_UAT-report_ToolName_v1.0.pdf
│   ├── 02_Training materials
│   │   └── ToolName_presentation_video.mp4
│   │       └── Training course
│   ├── 03_Test data
│   │   ├── README.txt
│   │   └── ToolName_Test_Data_Manual.pdf
│   │       └── ToolName_TestData
│   ├── 04_Installation files
│   │   └── README.txt
│   ├── 05_Source code
│   │   ├── Automated test scripts
│   │   └── Tool's source code
│   │       ├── LICENSE
│   │       └── source-code.zip
│   ├── 06_License
│   │   └── LICENSE.txt
│   └── 07_SBOM
│       ├── README.txt
│       ├── SBOM IPR analysis report.pdf
│       ├── SBOM Security analysis report.png
│       └── sbom-toolname.xml
```

Figure 1. Tool example package folder's structure displayed on a terminal

Once all the items are in order, the next step will be to compress the entire package folder in a unique (**ToolName_vToolVersion (FINAL).zip**) or multiple part(s) (**ToolName_vToolVersion (FINAL).Part-X.zip**) and continue with the release process.

3. Release of tools

3.1. Definition

The release phase includes all the activities needed to prepare the software product (the package) to be distributed to its intended customers. This includes planning, designing, and preparing all the means necessary to facilitate the installation, initial configuration, and use of the software product when in the environments of the targeted customers. Typical examples of release artefacts are installation auxiliary tools, installation and user manuals, licence(s), complementary services such as training or customer support. This phase repeats every time the business decides to launch a new version of the software product to its intended customers.

The processes within this phase are release policy definition, planning, design, preparation, and release execution.

Note that not all software versions that are built become a release because only the software versions that are published or delivered to the intended end-users/customers are considered software releases.

From a security point of view, during the release phase it is important to conduct a security readiness check to make sure that all the security tests have been conducted as planned and that all identified vulnerabilities have been appropriately handled. No release should be completed and allowed to pass to the delivery phase without the validation of the security readiness check. Also, during the release phase, code signing certificates should be used to digitally sign the software product's applications and executables (See [ANNEX II](#)), so that, when the software is deployed, the end-users can verify that the code has not been altered or compromised by a third party. If the product has already been signed, it should be checked that the signature is correctly apply and working. Also, during this phase, the components of the packaging are reviewed and their hashes extracted, in order to keep track of the digital fingerprint of the tool and ensure the integrity of its components to the end-users.

3.2. Processes involved

3.2.1 Release planning

	DevOps aspects	Security considerations	Fighting cybercrime domain considerations
Description	<p>This phase involves planning, scheduling, and controlling the software development and delivery process.</p> <p>DevOps teams share the responsibility for the services they deliver, and their code. When software developers and IT professionals are involved in the entire delivery lifecycle, incidents are detected and resolved faster. This process is followed during and after the release process.</p> <p>The key areas of the release cycle are the following:</p> <ul style="list-style-type: none"> • Continuous exploration • Continuous integration • Continuous deployment • Release on demand 	<ul style="list-style-type: none"> • Provide mechanisms for verifying software release integrity. • Archive and protect each software version release 	
Roles & responsibilities	<ul style="list-style-type: none"> • The development team is responsible for creating new code or features. 	-	-

	<ul style="list-style-type: none"> • The testing team is responsible for testing the new code/features in a staging environment. • The release manager is responsible for coordinating the entire process and approving it so that the product is released to production environment. 		
Inputs	<ul style="list-style-type: none"> • Product Backlog • Stakeholder Feedback • Market Research • Project Constraints 	-	-
Outputs	<ul style="list-style-type: none"> • Release Plan • Release Backlog • Prioritized Features • Release Roadmap 	-	-
Benefits	<ul style="list-style-type: none"> • Fewer and shorter feedback loops. • Faster releases. • Enhanced Communication. • Alignment between development efforts and business goals. • Improved visibility of what to expect in the upcoming release. • Risk mitigation by identifying potential risks. 	-	-

<p>Exchange data formats</p>	<ul style="list-style-type: none"> • JSON • XML • YAML • CSV • Standard Document Formats (Word, PDF...) 	<p>-</p>	<p>-</p>
<p>Taxonomies, ontologies...</p>	<p>-</p>	<p>-</p>	<p>-</p>
<p>Standards</p>	<ul style="list-style-type: none"> • ISO/IEC/IEEE 32675:2022 • ISO/IEC 12207 • ISO/IEC 27001 • NIST SP 800-64 • DevOps Planning Standards. 	<p>Compliance and Regulations: Ensure that the release complies with relevant industry-specific regulations (e.g., GDPR, HIPAA) and security standards (e.g., ISO 27001).</p> <p>NIST SP 800-64: Offers guidance on security considerations during software development and release planning.</p>	<p>-</p>
<p>Techniques</p>	<ul style="list-style-type: none"> • State/define the criteria for success. • Aim for minimal customer impact • Take advantage of the staging environment. • Optimise CI/CD and QA. • Use automation whenever possible. • If possible, make things immutable 	<p>Security Threat Assessment: Identify potential security threats and risks relevant to the release. Consider factors like the sensitivity of data, potential attack vectors, and the impact of security breaches.</p>	<p>-</p>

Tools	Type or category of tools	Key features	Commonly used open-source tools
	Project Management	Task management, time tracking, analytics, reporting,	GitLab ; Jira ; Zoho ;
	CI/CD platforms	<p>Continuous integration (CI), for Frequent merging of several small changes into a main branch.</p> <p>Continuous Delivery (CD), for building, testing, and releasing software with greater speed and frequency.</p> <p>Continuous deployment (CD), for adding the deployment of the code to the continuous delivery activities.</p>	Jenkins ; Gradle ; GitLab ; GoCD ; Buddy ; Argo CD ; Spinnaker ; Ansible ;

3.2.2. Release design

Description	DevOps aspects	Security considerations	Fighting cybercrime domain considerations
	<p>Inside the release design process, the methodology for the release execution is defined.</p> <p>Teams define the technical details of the release, design the architecture, create release-specific requirements, and plan for any necessary infrastructure changes.</p> <p>There are different ways to plan the release:</p> <ul style="list-style-type: none"> • Planning-based: Structured planning, testing and deployment of software, suitable for large/complex projects with long release cycles. • Automated: Use of automated tools and processes to manage the entire software delivery pipeline, ideal for organizations with frequent and rapid releases. • Agile: Flexible and focused on delivering incremental changes and continuous improvements to releases, used in organizations that require quick and frequent releases, as well as adaptation. 	-	-

Roles & responsibilities	<p>The release manager is responsible for creating a release design that adapts the characteristics and requirements of the project.</p> <p>Software architects, developers, and technical leads are heavily involved in the release design phase.</p>	Compliance: Ensure that the release complies with industry standards, security regulations, and best practices.	-
Inputs	A release plan.	-	-
Outputs	<ul style="list-style-type: none"> • Detailed design documents. • Technical requirements. 	-	-
Benefits	<ul style="list-style-type: none"> • Improved reliability and stability of the releases. • Delivery of high-quality software releases that satisfy the customer requirements. • Fewer and shorter feedback loops. • Faster releases. 	-	-
Exchange data formats	<ul style="list-style-type: none"> • JSON • XML • YAML • CSV 	-	-
Taxonomies, ontologies, ...	-	-	-
Standards	<ul style="list-style-type: none"> - ISO/IEC 19770 - ISO/IEC 12207 	OWASP Secure Software Development Lifecycle (SSDL): Offers best practices and guidelines for secure release design.	-

	<ul style="list-style-type: none"> - OWASP Secure Software Development Lifecycle (SSDL) - DevOps Design Standards 		
Techniques	<ul style="list-style-type: none"> • Definition of technical specifications. • Decisions on deployment strategies. • Identification of third-party integrations. 	Risk assessment and mitigation planning.	-
Tools	Type or category of tools	Key features	Commonly used open-source tools
	Project Management	Task management, time tracking, analytics, reporting,	GitLab ; Jira ; Zoho ;
	CI/CD platforms	<p>Continuous integration (CI), for Frequent merging of several small changes into a main branch.</p> <p>Continuous Delivery (CD), for building, testing, and releasing software with greater speed and frequency.</p> <p>Continuous deployment (CD), for adding the deployment of the code to the continuous delivery activities.</p>	Jenkins ; Gradle ; GitLab ; GoCD ; Buddy ; Argo CD ; Spinnaker ; Ansible ;

3.1.1. Release preparation

DevOps aspects	Security considerations	Fighting cybercrime domain considerations
-----------------------	--------------------------------	--

<p>Description</p>	<p>During the review, the QA team will conduct final checks to ensure the build meets the minimum acceptable standards and business requirements.</p> <p>This phase sets the stage for delivering software that meets customer expectations and operates smoothly in a production environment.</p> <ul style="list-style-type: none"> • Review of release plan. • Verification of Completion. • Testing and validation. • Security assessment. • Quality assurance. • Documentation and User manuals. • Archiving Previous Releases. • Final Review and Approval. • Backup and Recovery plans. • Stakeholder Communication. • Final Approval. • Monitoring and Support. 	<p>Identify vulnerabilities or possible exploits through the Quality Tests to identify and fix them before release or in order to communicate them to the End Users.</p> <p>Security Documentation: Create comprehensive security documentation, including security guidelines, best practices, and incident response plans. All team members should be aware of and follow these security protocols.</p> <p>Cybersecurity Awareness: Conduct cybersecurity awareness training for all team members. Human error is a common cause of security breaches, so educating the team is crucial.</p>	
<p>Roles & responsibilities</p>	<ul style="list-style-type: none"> • The release manager is responsible for conducting a release preparation that can 		

	<p>achieve a final product that meets the requirements and prevents possible post release problems. Also needs to adapt to the characteristics and requirements of the project. Communicate the release schedule to stakeholders.</p> <ul style="list-style-type: none"> • The QA team to conduct the final tests. Identify areas for improvement in the release process. Verify that the release aligns with the organization's quality goals. • The Release Advisory Board (RAB) will approve or reject the release based on business and technical considerations. 		
<p>Inputs</p>	<ul style="list-style-type: none"> • Release plan. • Code Repository. • Defect Reports. • Test Cases and Test Data. • Release Documentation. • Quality Assurance (QA) Reports. • Requirements. • Operational Documentation. 	<p>Security Findings and Vulnerability Assessments: Reports from security assessments and vulnerability scans identify potential security issues. These findings must be addressed and resolved before the release to ensure the software's security.</p>	

	<ul style="list-style-type: none"> • Dependencies and Third-Party Components. • Historical data. • Feedback from Pre-release testing. 		
<p>Outputs</p>	<ul style="list-style-type: none"> • Release Package. • Release notes. • Deployment plan. • Operational Runbooks. • Test reports. • Security Documentation. • Database Change Scripts. • Approval and Sign-off. • Backup and Rollback Plans. • Quality Assurance (QA) Reports. • Validation of Compliance. • Historical Data. • Feedback from Pre-release Testing. • Release Package Distribution. • Operational Team Training. • Lessons Learned Report. 		
<p>Benefits</p>	<ul style="list-style-type: none"> • Enhanced reliability. • Reduced Deployment Risks. • Better Security and Compliance. 		

	<ul style="list-style-type: none"> • Optimized Deployment. • Customer Satisfaction. 		
Exchange data formats	-	-	-
Taxonomies, ontologies, ...	-	-	-
Standards	<ul style="list-style-type: none"> • ISO/IEC/IEEE 32675:2022 • ISO/IEC 20000 • ITIL (Information Technology Infrastructure Library) • ISO/IEC 27001 • IEEE 12207 • NIST SP 800-53 • Industry-Specific Standards • Customized Internal Standards 	<p>The ISO/IEC 27001 is the international standard for information security management systems. It is relevant to release preparation as it emphasizes the importance of security in the release process. Complying with ISO/IEC 27001 helps address security considerations during release preparation.</p> <p>NIST SP 800-53: Provides security controls and guidelines for the preparation phase, addressing security considerations.</p>	
Key Activities	<ul style="list-style-type: none"> • Test planning and execution, including unit, integration, and user acceptance testing. • Security testing and vulnerability scanning. • Performance and scalability testing. • Documentation creation, including release notes and installation guides. • Deployment planning and coordination. 		

Tools	Type or category of tools	Key features	Commonly used open-source tools
	Project Management	Task management, time tracking, analytics, reporting,	GitLab ; Jira ; Zoho ;
	CI/CD platforms	<p>Continuous integration (CI), for Frequent merging of several small changes into a main branch.</p> <p>Continuous Delivery (CD), for building, testing, and releasing software with greater speed and frequency.</p> <p>Continuous deployment (CD), for adding the deployment of the code to the continuous delivery activities.</p>	Jenkins ; Gradle ; GitLab ; GoCD ; Buddy ; Argo CD ; Spinnaker ; Ansible ;

3.1.2. Release execution

DevOps aspects	Security considerations	Fighting cybercrime domain considerations
----------------	-------------------------	---

<p>Description</p>	<p>Once the release has been correctly planned, designed, and prepared, it comes the release execution phase.</p> <p>The main goals of the release execution are:</p> <ul style="list-style-type: none"> • Run the appropriate security readiness tests (if needed), to validate that the build package is appropriate to be released. • Extract the hashes of the prepared package to ensure the security of the tool delivery and tracking. 	<p>The readiness and security tests are important to check that the release version of the tool does not involve any vulnerabilities or exploits that can be used to attack the End Users.</p> <p>The signing of the product with the certificate is also important for the End Users to trust that the delivered software has not been compromised or altered by external sources.</p> <p>Also, the extraction of the hashes to keep track of the tool and avoid third parties to deliver a compromised version of it. By keeping track of the hash end-users can ensure that the downloaded product is original and safe to use.</p>	
<p>Roles & responsibilities</p>	<ul style="list-style-type: none"> • The release manager is responsible for supervising the execution of the release and reviewing that the final release product is ready for delivery. • The testing team is responsible for testing the readiness and security of the product and sign them digitally with the certificate. 		

Inputs	Build package including all the deliverables.		
Outputs	The components of the build package correctly tested, and ready to make the delivery to production.		
Benefits	<ul style="list-style-type: none"> • Discard build versions that do not satisfy with the requirements to become a final release product to be delivered to the End Users. • Smooth and controlled deployment. • Real-time issue detection. • Satisfied end-users. 	Detection of vulnerabilities of the tool in order to report them for future enhancements or updates of the tool and to report them to the End-users.	
Exchange data formats	-	-	-
Taxonomies, ontologies, ...	-	-	-
Standards	<ul style="list-style-type: none"> • ISO/IEC 12207 • ISO/IEC 19770 • NIST SP 800-53 • Continuous Monitoring Standards 	NIST SP 800-53: Includes controls for monitoring and securing the execution phase of the release.	
Techniques	<ul style="list-style-type: none"> • Testing and quality assurance involve ensuring that the software release meets the required quality standards. It includes activities such as 		

	<p>functional testing, regression testing, load testing, and user acceptance testing (UAT).</p> <ul style="list-style-type: none"> • Deployment strategies. • Real-time monitoring. 		
Key Activities	<ul style="list-style-type: none"> • Deployment of the release package to production servers. • Monitoring for any issues or incidents during deployment. • Immediate response to any deployment problems. • Verification and validation of the deployed release. • Transition to post-release monitoring and support. 		
Tools	Type or category of tools	Key features	Commonly used open-source tools
	Project Management	Task management, time tracking, analytics, reporting,	GitLab ; Jira ; Zoho ;
	CI/CD platforms	<p>Continuous integration (CI), for Frequent merging of several small changes into a main branch.</p> <p>Continuous Delivery (CD), for building, testing, and releasing software with greater speed and frequency.</p>	Jenkins ; Gradle ; GitLab ; GoCD ; Buddy ; Argo CD ; Spinnaker ; Ansible ;

		Continuous deployment (CD), for adding the deployment of the code to the continuous delivery activities.	
--	--	--	--

3.2. Specificities within the Tools4LEAs project

For EACTDA’s release process, once the tool package has been generated and it is ready to be released, some hashes must be generated to ensure the integrity of the data within the package. To do so, two different scripts have been prepared. The first one obtains the hash for each file inside the folder structure (SHA256 format) and lists them in a .csv file, which is later converted into a .ods file. The following image shows the table format and the type of information in it.

Column1	Column2	Column3
#TYPE Microsoft.PowerShell.Utility.FileHash		
Algorithm	Hash	Path
SHA256	[REDACTED]	[REDACTED]

Once the tool package is ready to be released, it is saved inside a zip file. At this point, a second script has been prepared that extracts the hash for that zip file with the format displayed in the following image (txt file):

```
@{Algorithm=SHA256; Hash=[REDACTED]; Path=.\<[REDACTED].zip}
```


4. Delivery of tools

4.1. Definition

The delivery phase includes the activities needed to effectively distribute the software product to its targeted customers. It is the stage in the software development and deployment lifecycle where application code, infrastructure changes, or updates are prepared for production release. It is a pivotal part of the DevSecOps pipeline that integrates security practices, automation, and collaboration into the delivery process. This phase repeats every time a software release is ready and the business decides to distribute it to its targeted customers. It is important to note that the targeted customer base can be segmented, and different objectives and distribution means (e.g., communication and/or dissemination materials and channels) can be defined for each segment.

The processes within this phase are planning, preparation, and execution of the software product distribution.

From a security point of view, secure delivery focuses on making sure that the distribution of the software product to its targeted customers is done in a secure way, by secure channels, minimising the risk of unauthorised (or unwanted) access to the software product and making sure that the software product is not compromised in the process. It emphasizes continuous testing, automation, and security measures to minimize risks and maintain the integrity of applications in a dynamic and fast-paced development environment.

4.2. Processes involved

4.2.1. Delivery planning

	DevOps aspects	Security considerations	Fighting cybercrime domain considerations
Description	This stage involves the creation of a detailed plan to guide the delivery process. This plan outlines how the delivery of the product will be conducted (who will receive the product, the method that will be used to deliver it and where will it be uploaded)	-	-

Roles & responsibilities	<ul style="list-style-type: none"> The Delivery Manager, as the person responsible for the coordination of the whole process and deciding about the delivery method to be used. 	<ul style="list-style-type: none"> The Delivery Manager is responsible for this role. 	-
Inputs	<ul style="list-style-type: none"> Released product package. 	<ul style="list-style-type: none"> As described during the Packaging phase, a Static Application Security Testing (SAST) analysis can help verifying that the final code of the product is ready to be released and that the whole product package is ready to be uploaded to the delivery platform. 	-
Outputs	<ul style="list-style-type: none"> Organisation’s delivery plan. 	-	-
Benefits	<ul style="list-style-type: none"> Higher security in the delivery process. Increases the speed in the delivery of the product to the target user. 	<ul style="list-style-type: none"> Having a proper delivery plan prepared beforehand will help in assuring that the product is delivered to the customers (end-user in this case) in time or within the promised delivery window. 	-
Exchange data formats	<ul style="list-style-type: none"> JSON YAML XML CSV TOML HCL OpenAPI/Swagger Custom Data Formats Protocol Buffers Custom API Endpoints 	<ul style="list-style-type: none"> In DevSecOps, data exchange formats should consider security from multiple angles, including data confidentiality, integrity, authentication, and authorization. Additionally, integrating encryption and digital signatures can further enhance the security of data exchange within the pipeline. The choice of data format should align with the organization's specific security requirements and best practices. 	-

<p>Taxonomies, ontologies, ...</p>	<ul style="list-style-type: none"> • <u>Vulnerability taxonomy</u>: It helps in prioritizing vulnerabilities and planning for remediation. • <u>Compliance taxonomy</u>: It helps in aligning the software delivery process with relevant compliance mandates (GDPR, ISO, W3C). • <u>Threat taxonomy</u>: It helps in identifying and mitigating security threats during delivery planning (CVE). • <u>Data Sensitivity Taxonomy</u>: It classifies data based on its sensitivity level, regulatory implications and privacy considerations. It helps in planning for data protection and privacy measures during delivery. 	-	-
<p>Standards</p>	<ul style="list-style-type: none"> • Information Technology Infrastructure Library (ITIL). 	-	
<p>Techniques</p>	<ul style="list-style-type: none"> • Agile planning: When working Agile the user can plan the deliver with a fixed release date or with an open release date. 	<ul style="list-style-type: none"> • Make sure that there is proper documentation that describes the delivery process. When there is a lost of track of process documentation, the user cannot ensure that all security protocols are being followed because the reference documentation might not exist, or storing it in a consistent manner was not a priority. 	-
<p>Tools</p>	<p>Type or category of tools</p>	<p>Key features</p>	<p>Commonly used open-source tools</p>

	Project Management	Task management, time tracking, analytics, reporting,	GitLab ; Jira ; Zoho ;
	CI/CD platforms	Continuous integration (CI), for Frequent merging of several small changes into a main branch. Continuous Delivery (CD), for building, testing, and releasing software with greater speed and frequency. Continuous deployment (CD), for adding the deployment of the code to the continuous delivery activities.	Jenkins ; Gradle ; GitLab ; GoCD ; Buddy ; Argo CD ;

4.2.2. Delivery preparation

	DevOps aspects	Security considerations	Fighting cybercrime domain considerations
Description	<p>This stage involves the preparation, implementation and configuration of the environments that will be used to upload the product, so that then it can be ready to be downloaded by the end-users.</p> <p>This stage also involves preparing the communication with the end-users and deciding the mechanisms that will be applied in order to ensure that they will have the necessary tools and</p>	<ul style="list-style-type: none"> Aim for immutable infrastructure, where deployments are consistent, repeatable, and not subject to change once deployed. This reduces the attack surface and enhances security. 	-

	configurations ready to receive the product.		
Roles & responsibilities	<ul style="list-style-type: none"> The Delivery Manager, as the person responsible for the coordination of the whole process and verifying that the selected delivery method has been successfully implemented, meaning the product package is ready to be delivered. The Systems Administrator (also tester in this case) as the person responsible for implementing and configuring the system/platform that will be later used to upload and distribute the product package. 	<ul style="list-style-type: none"> Implement access controls and authentication mechanisms. Apply patches and updates to keep systems secure. Ensure that the infrastructure has been configured securely. 	-
Inputs	<ul style="list-style-type: none"> Organisation’s delivery plan. 	-	-
Outputs	<ul style="list-style-type: none"> Delivery system/platform 	-	-
Benefits	<ul style="list-style-type: none"> Higher control of the systems and services that are being prepared to deliver the product package. Higher control of the implemented access method for the end-users. 	<ul style="list-style-type: none"> Faster and More Reliable Delivery. Faster Incident Response: With a well-prepared incident response plan in place during delivery preparation, organizations can respond more effectively to security incidents, minimizing their impact and downtime. 	-
Exchange data formats	<ul style="list-style-type: none"> JSON YAML XML CSV 	<ul style="list-style-type: none"> The choice of data exchange format depends on the specific requirements of the organization, the tools and systems in use, and the need for compatibility and interoperability. Using 	-

	<ul style="list-style-type: none"> • TOML • HCL • OpenAPI/Swagger • Custom Data Formats • Protocol Buffers • Custom API Endpoints 	<p>standardized and well-documented formats enhances the efficiency of the DevSecOps pipeline and promotes collaboration between development, security, and operations teams.</p>	
Taxonomies, ontologies, ...	<ul style="list-style-type: none"> • <u>Artifact and Configuration Taxonomy</u>: It classifies software artifacts (containers, deployment scripts, etc) and configuration items based on their purpose and their impact. • <u>Change Management Taxonomy</u>: it organizes and categorizes different types of changes, including routine updates, security patches configuration changes, and major releases. It helps in tracking changes during delivery preparation. 	<ul style="list-style-type: none"> • Security control taxonomies classify and categorize various security controls and measures that can be applied to mitigate risks, vulnerabilities, and threats. This helps in selecting the most appropriate controls for the delivery preparation process. 	-
Standards	<ul style="list-style-type: none"> • Information Technology Infrastructure Library (ITIL). 	-	-
Techniques	<ul style="list-style-type: none"> • Zero Trust Security Model: Apply the principle of least privilege and the zero trust model, where access to systems and data is restricted to only what is necessary for a given user or system to perform its job. 	<ul style="list-style-type: none"> • Zero Trust relies heavily on strong authentication mechanisms such as multi-factor authentication (MFA) and continuous authentication. Users and devices are required to prove their identity and trustworthiness before accessing resources or systems. 	-
Tools	Type or category of tools	Key features	Commonly used open-source tools

	Project Management	Task management, time tracking, analytics, reporting,	GitLab ; Jira ; Zoho ;
	CI/CD platforms	Continuous integration (CI), for Frequent merging of several small changes into a main branch. Continuous Delivery (CD), for building, testing, and releasing software with greater speed and frequency. Continuous deployment (CD), for adding the deployment of the code to the continuous delivery activities.	Jenkins ; Gradle ; GitLab ; GoCD ; Buddy ; Argo CD ;

4.2.3. Delivery execution (or release distribution)

	DevOps aspects	Security considerations	Fighting cybercrime domain considerations
Description	This stage involves the upload of the product package to the previously implemented and prepared environments. This stage also involves communicating the end-users that the package is finally uploaded to the defined platform and that it is ready to be retrieved by them. Besides, this stage also involves verifying that the end-users have	<ul style="list-style-type: none"> Ensure that sensitive data is handled securely and is encrypted both in transit and at rest. Implement access controls and data minimization techniques to protect user privacy. 	-

	successfully managed to obtain the product package.		
Roles & responsibilities	<ul style="list-style-type: none"> The Delivery Manager, as the person responsible for the coordination of the upload of the product package to the environment and for the later communication with the end-users. The Systems Administrator as the person responsible for uploading the product package to the system/platform and ensuring that the environment is ready to be used by the end-users The end-users, as the people responsible for accessing the system/platform and correctly downloading the product package. 	<ul style="list-style-type: none"> Ensure that the infrastructure has been configured securely. Apply patches and updates to keep systems secure. End-users shall: <ul style="list-style-type: none"> Follow security policies and guidelines when using the service/platform. Report security incidents and vulnerabilities promptly. Participate in security awareness training. 	-
Inputs	<ul style="list-style-type: none"> Delivery system/platform 	-	-
Outputs	<ul style="list-style-type: none"> Product package successfully delivered. 	-	-
Benefits	<ul style="list-style-type: none"> Ensuring that the product package is received by the intended users with no issues. 	<ul style="list-style-type: none"> By proactively addressing security concerns during delivery preparation, organizations can mitigate security risks and reduce the likelihood of security breaches, data leaks, or other security incidents. Security checks and tests integrated into the CI/CD pipeline ensure that only secure and 	-

		<p>compliant code is promoted to production. This leads to faster and more reliable software delivery with fewer post-release issues.</p>	
<p>Exchange data formats</p>	<ul style="list-style-type: none"> • <u>Container Images</u>: Container image formats like Docker's .tar files or Open Container Initiative (OCI) formats are essential for distributing containerized applications. • <u>Package Managers</u>: Package manager formats such as RPM (Red Hat Package Manager) or DEB (Debian package) are used for distributing software packages and libraries in Linux environments. • <u>ZIP archives</u>: ZIP archives are commonly used to bundle and distribute files, applications, and related assets in a compressed format. • <u>Artifact Repositories</u>: Artifact repository formats like JFrog Artifactory and Sonatype Nexus store and distribute binary artifacts, including libraries, dependencies, and application builds. • <u>Custom Data Formats</u>: In some cases, organizations may create custom data exchange formats tailored to their specific release 	<ul style="list-style-type: none"> • <u>Git Repositories</u>: Git repositories can be used to distribute code and version-controlled assets. Git supports both code and documentation distribution. • <u>HTTP/HTTPS</u>: These protocols are commonly used for distributing files, documents, and artifacts securely over the internet. They are widely used for software updates and patches. • <u>Blockchain</u>: Blockchain-based formats can be used for secure and immutable distribution of software updates, ensuring the integrity of the delivered code. • <u>P2P (Peer to Peer) Networks</u>: P2P networks may employ their own data exchange formats for distributing software updates and patches in a decentralized manner. • <u>FTP (File Transfer Protocol)</u>: It is a network protocol used for transferring files, and it can be employed for distributing software and related files to remote servers and locations. • <u>CDN (Content Delivery Network)</u>: CDNs often use their own data formats for distributing content efficiently to users worldwide. This can include caching strategies and content optimization. 	<p>-</p>

	<p>distribution needs. These formats can be optimized for their unique use cases.</p>		
<p>Taxonomies, ontologies, ...</p>	<ul style="list-style-type: none"> • <u>Authentication and Authorization Ontology</u>: It categorizes different authentication and authorization mechanisms. It helps in planning the access control during delivery preparation (MFA, role-based and attribute-based access control, single sign-on, OAuth, SAML, OpenID Connect, LDAP). • <u>Distribution Channel Taxonomy</u>: It categorizes distribution channels based on their security features and usage, such as public and private repositories, and content delivery networks (CDNs). It helps in selecting the most secure distribution channels. • <u>Secure Transport Protocol Ontology</u>: It defines secure transport protocols (HTTPS, SFTP, SCP, FTPS, SMTPS, IMAPS, POP3S, IPsec, SSH) and categorizes them based on their encryption strength and authentication mechanisms. It helps in choosing secure transport methods for distribution. 	<ul style="list-style-type: none"> • These taxonomies and ontologies serve as a guide in the selection of appropriate distribution channels, methods and verification processes to minimize security risks. 	<p>-</p>

	<ul style="list-style-type: none"> • <u>Secure Content Delivery Taxonomy</u>: It categorizes CDNs and content delivery methods based on their security controls, distributed denial of service (<u>DDos</u>) mitigation capabilities, and secure caching mechanisms. 		
Standards	<ul style="list-style-type: none"> • Information Technology Infrastructure Library (ITIL). 	-	-
Techniques	<ul style="list-style-type: none"> • Agile Delivery: Some of its benefits are: <ul style="list-style-type: none"> ○ Teams are kept small and iterations short. ○ Feedback from customers is fast. ○ Business priorities are value-based. ○ Users are engaged in the refining of end-product requirements. 	<ul style="list-style-type: none"> • The following methods can help ensuring and controlling that the product package can only be received by the intended users: <ul style="list-style-type: none"> ○ Use of PGP keys. ○ Preparation of VPN credentials. ○ Identification of delivery channels. One of the possible and most secure delivery channels is a Secure File Transfer Protocol (SFTP). Also, the credentials to access the server will be required. 	-
Tools	Type or category of tools	Key features	Commonly used open-source tools
	Project Management	Task management, time tracking, analytics, reporting,	GitLab ; Jira ; Zoho ;
	CI/CD platforms	Continuous integration (CI), for Frequent merging of several small changes into a main branch. Continuous Delivery (CD), for building, testing, and releasing software with greater speed and frequency.	Jenkins ; Gradle ; GitLab ; GoCD ; Buddy ; Argo CD ;

	Continuous deployment (CD), for adding the deployment of the code to the continuous delivery activities.	
--	--	--

4.3. Specificities within the Tools4LEAs project

For EACTDA's delivery process, two methods (repositories) have been defined and implemented. The first one consists on uploading the release package to EACTDA's repository of tools (see Annex III). Each project developed under the Tools4LEAs v2 projects will contain an entry inside that repository where everything related to the project will be added, including the zip file with the release package. The internal structure of the package has already been described during the "Packaging of tools" section.

Every authorised end-user that requests access to the release package will receive an email with instructions on how to connect to a VPN. This will allow the user to access EACTDA's repository of tool. For that purpose, the user will have to specify its email and send its public PGP key, so that the instructions can be sent via an encrypted message.

The second method involves the use of a SFTP server. SFTP (Secure File Transfer Protocol) is a file transfer protocol that leverages a set of utilities that provide secure access to a remote computer to provide secure communications. It is considered by many to be the optimal method for secure file transfer. It makes use of SSH (Secure Socket Shell or Secure Shell) and is often also referred to as 'Secure Shell File Transfer Protocol'.

Therefore, the release package will be uploaded to the SFTP server by the system administrator. The end-user will only have to connect to the server and they will automatically be redirected to the location of the release package (zip file). To connect to the SFTP server, the end-user will have to install and connect to a VPN server (the credentials will be provided by EACTDA). Once that step is fulfilled, he/she will have to the enter the user credentials to connect to SFTP. Each time a new version of a tool has been released, the tool in question will be uploaded to the SFTP server with its newest version.

5. Summary

5.1. Conclusion

In this document we have introduced the concepts of packaging, releasing and delivering software packages, and for each of them we have presented the processes (steps) that are to be conducted in the Tools4LEAs project. When defining each process, a description of the process, the typical roles and responsibilities of the persons affected, inputs and outputs, benefits, exchange data formats, taxonomies and ontologies, standards, techniques, and tools (categories and some relevant example open-source tools) are presented with three points of view: DevOps, Security, and Fighting Cybercrime specific considerations.

This document therefore serves as the handbook on these matters that is to be used in the Tools4LEAs project.

5.2. Evaluation

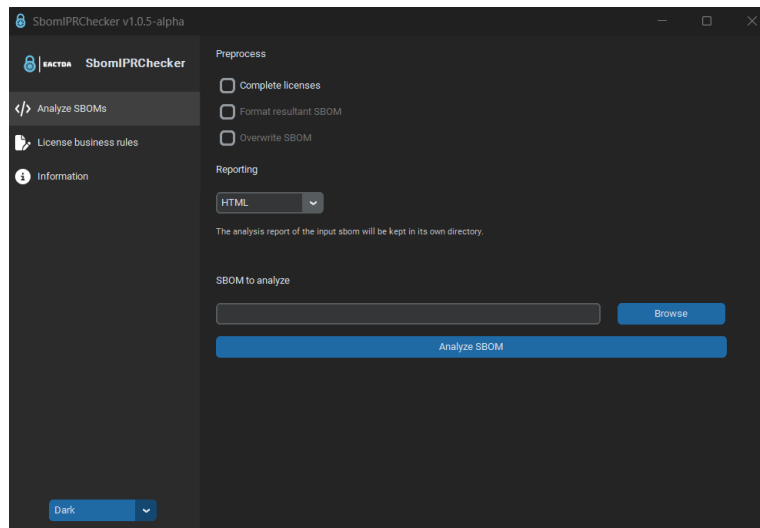
Though the document will be maintained and updated on a regular basis (at least yearly), it is considered to be ready to serve as the handbook on packaging, releasing, and delivering software products for the Tools4LEAs-v2 project.

5.3. Future work

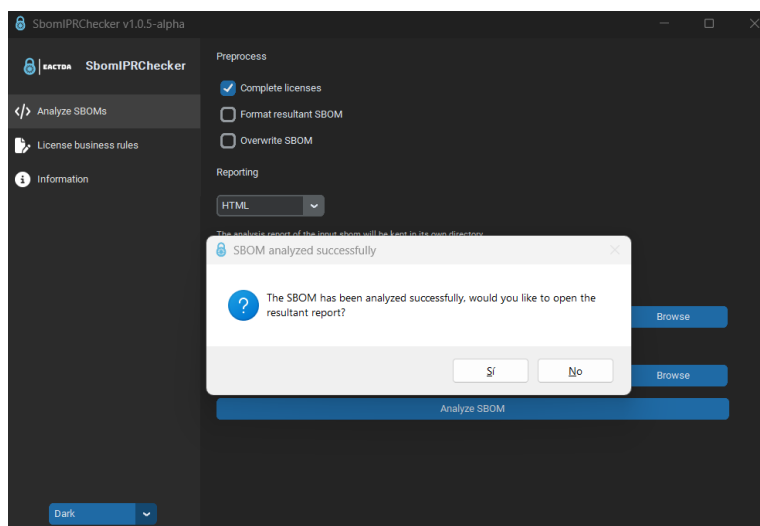
As previously stated, this document will be updated on a regular basis (at least yearly), though the updates will be for internal use and are not planned to be submit

ANNEX I – Overview of the S bomIPRChecker

S bomIPRChecker is a software internally developed which gathers all the licensing information related to the third-party components that conform a software product using a SBOM file according to the CycloneDX format. This internal tool was developed according to the need for a tool that would acquire licences for components that did not have them, making an IPR analysis in the process to verify if some component presents licensing problems.



To generate a SBOM report using the tool, it should run the application, go to the “Analyze SBOMs pane”, check the “Complete licenses” checkbox, select an input SBOM file and then press on the “Analyze SBOM” button, by default the reporting format will be HTML (preferred option by EACTDA). The analysis process time can vary depending the physical size of the SBOM file and how many components do not present a license, the “Complete licenses” checkbox increases the analysis time due to API calls made to retrieve component licenses.



Once the SBOM report is generated, the next step will be to print as a PDF the resultant HTML. This can be achieved by open the SBOM HTML report and click on the print button located at the bottom of the document. This way to get SBOM reports can vary depending on the version of this tool, but this task will be always performed by this tool in a similar way.

ANNEX II – Software signing

Depending on the operative system in which it is executed an executable file, it is required to sign the software in order to satisfy the security rules established by the OS to be able to execute software. For desktop applications, the type of certificate depends on the operating system being used, as described below:

Windows OS certificates

There are two kind of code signing certificates to sign a software on Windows, an individual/organization certificate and an extended validation certificate. An individual/organization certificate consists on a certificate file which it is used along with a password to sign software, this certificate gives standard validation, verifying the rules established by Microsoft Authenticode to be trusted.

On the other hand, an Extended Validation certificate gives full trust to any software signed with this type of certificate, verifying the rules established by Microsoft Authenticode and SmartScreen. This is the preferred option used by EACTDA since it does not return any kind of warning from the OS, it consists of a certificate file that it is stored on a physical USB key where it occurs the signing of a software along with a password.

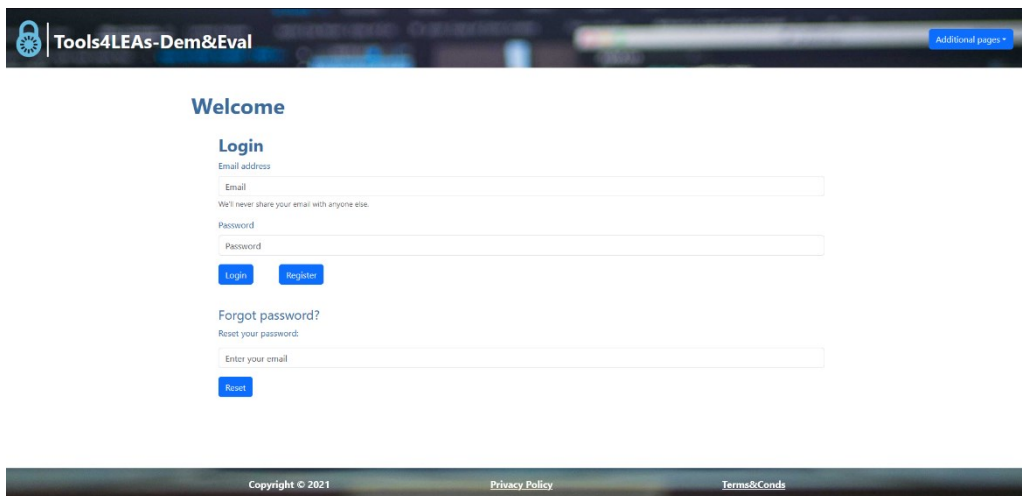
Mac OS certificates

For a Mac desktop application, it is necessary to obtain a code signing certificate through a Mac Developer Account, which it consists on a normal Mac account subscribed to the Apple Developer Program. This subscription allows to create any kind of certificate to sign applications depending on how they are distributed by different channels. This will prevent a warning from the MacOS gatekeeper, a program which verifies if a software is from a trusted developer when it is executed software on Mac.

On the other hand, regarding web applications, it is important to remark the usage of SSL certificates to secure the data transmission between the server and the client. If it is the case of a web application that runs over a private computers network, it should be created self-signed certificates and if it is the case of a web application running over a public computers network, it should be obtained a certificate issued by a recognized certificate authority.

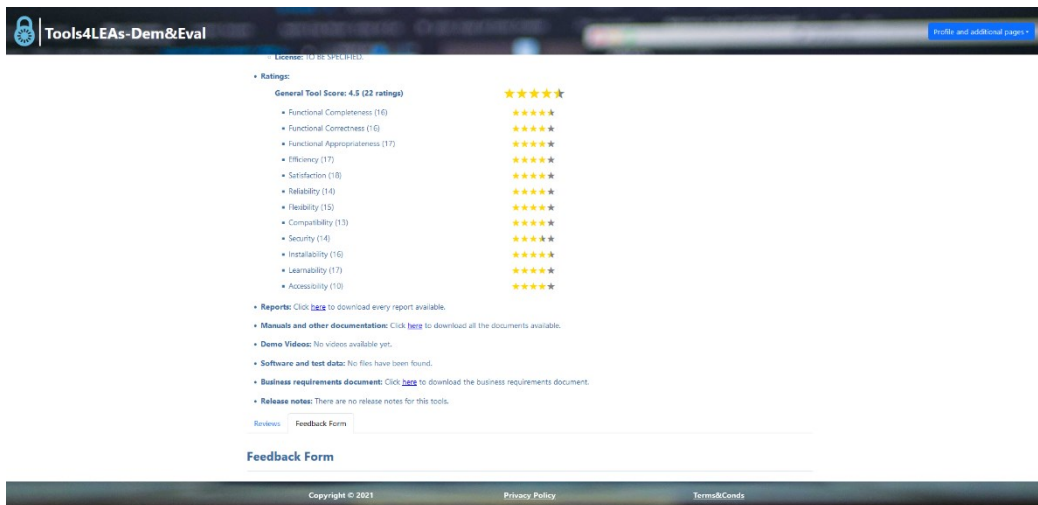
ANNEX III – Overview of EACTDA’s repository of tools

The EACTDA’s repository of tools consists on a web platform which main purpose is to get feedback from the end-users about any tool.



The screenshot shows the login page of the Tools4LEAs-Dem&Eval platform. At the top, there is a navigation bar with the logo and a link for 'Additional pages'. The main content area is titled 'Welcome' and contains a 'Login' section with input fields for 'Email address' and 'Password', and buttons for 'Login' and 'Register'. Below this is a 'Forgot password?' section with a 'Reset' button. At the bottom, there is a footer with 'Copyright © 2021', 'Privacy Policy', and 'Terms&Conds'.

In this platform, each tool already evaluated or to evaluate counts with a page in which the tool can be downloaded along with other auxiliar content such as its installation manual, its user guide, etc. The content which conforms the entire package of a tool.



The screenshot shows the evaluation page for a tool in the Tools4LEAs-Dem&Eval platform. The page features a 'Ratings' section with a 'General Tool Score: 4.5 (22 ratings)' and a list of 12 categories, each with a star rating. Below the ratings, there are sections for 'Reports', 'Manuals and other documentation', 'Demo Videos', 'Software and test data', 'Business requirements document', and 'Release notes'. At the bottom, there is a 'Feedback Form' section. The footer contains 'Copyright © 2021', 'Privacy Policy', and 'Terms&Conds'.

Category	Rating
General Tool Score	4.5 (22 ratings)
Functional Completeness (16)	★★★★★
Functional Correctness (16)	★★★★★
Functional Appropriateness (17)	★★★★★
Efficiency (17)	★★★★★
Satisfaction (18)	★★★★★
Reliability (14)	★★★★★
Flexibility (15)	★★★★★
Compatibility (13)	★★★★★
Security (14)	★★★★★
Instability (16)	★★★★★
Learnability (17)	★★★★★
Accessibility (10)	★★★★★