DG DIGIT
Unit B1

Project-End Report

EU-FOSSA Pilot Project

Date: 10/11/2016
Doc. Version: Public
Template Version: 2.5

This template is based on PM² v2.5

For the latest version of this template please visit the PM² Wiki

Disclaimer

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the Commission. The content, conclusions and recommendations set out in this publication are elaborated in the specific context of the EU – FOSSA project.

The Commission does not guarantee the accuracy of the data included in this study. All representations, warranties, undertakings and guarantees relating to the report are excluded, particularly concerning – but not limited to – the qualities of the assessed projects and products. Neither the Commission nor any person acting on the Commission's behalf may be held responsible for the use that may be made of the information contained herein.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

Introduction

The Free and Open Source Software Auditing (EU-FOSSA) is a pilot project financed by the European Parliament and implemented by the Directorate for Informatics of the European Commission (DIGIT). The project was proposed in 2014, after the discovery of the "Heartbleed" software bug, which shocked the entire IT community and affected a very widely used open source security library, OpenSSL.

The current OSS strategy of the European institutions puts a special emphasis on procurement, contribution to open source software projects and providing more of the software developed within the Commission as open source. Consequently, any possible disruption in open source software may have significant impacts on the operations of the European institutions and the connected organisations.

Identifying key building blocks and performing a systematic review of source code should help mitigate the risks of serious security vulnerabilities.

The EU-FOSSA pilot project offers a systematic approach for the EU institutions to contribute to maintaining integrity and security of widely used critical open source software.

1. Main Findings

EU-FOSSA has focused on three main areas:

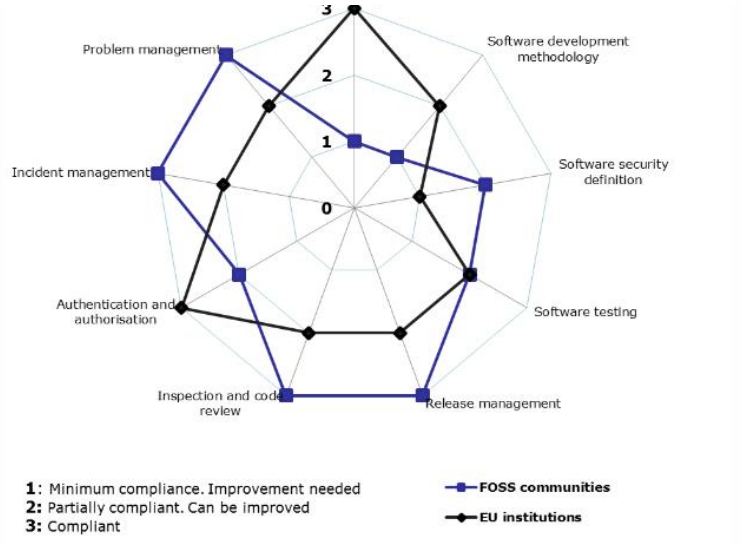
- contribution to the assessment of OSS project sustainability;
- inventory and criticality / sustainability assessment of the OSS installed at the European institutions;
- definition and pilot application of a methodology for code review.

For each one of these areas, EU-FOSSA has highlighted the main findings listed in the three paragraphs below.

1.1. Software security and sustainability is a common engagement with margins of improvement for EU institutions and FOSS communities

Representatives of 14 communities and 14 OSS projects of the European institutions[1] were contacted to gather information on their software development and security assurance practices. Additionalinformation on the subject was also collected from trusted sites such as the FOSS Communities websites, recognised wikis and forums, to analyse and compare how European institutions and FOSS communities develop their software and conduct the tasks

Figure 1 - Comparison of software development practices - EU institutions vs FOSS communities



[1] The analysed OSS communities were: Apache Tomcat, Bitergia, Debian, Drupal. Eclipse, Jenkins, LibreOffice, OWASP, OwnCloud, OpenSSL, OpenStack, Piwik, Red Hat, Spring.

The analysed EU projects were: Decide, Piwik, Statistical Atlas, OCM, CITnet, ECAS, UMM, Next Europe, Multisite, Sysper, Valor for the European Commission and Finord, ePortal, Glassroom for the European Parliament.

related to software security and security maintenance.

The study has identified on this basis the best software security practices used by both sides in the software development life cycle (SDLC) phases (see Figure 1).

Best practices for the FOSS communities are, for example:

- Frequent inclusion of **security objectives in project roadmaps**;
- **Publicity and submission of code reviews to multiple contributors**, several of which are security experts, are in the nature of the OSS communities;
- **Efficient and transparent incident management**;
- Their **capability to easily detect potential flaws**, as software contributions are analysed before being accepted, and FOSS software is run (and tested) in several different production environments (FOSS users' environments).

Best practices of the OSS projects managed by the European institutions, on the other hand, are:

- A more formal, **structured project management** approach;
- A **stronger trend towards standard authentication procedures and tools**.

The main **recommendations** for the two sides, inspired by the comparison of their respective best practices, may therefore be:

for the **FOSS communities**:

- use formal methodologies for project management and software development;
- promote project management contributions in the community;

for the **European institutions**:

- improve software security (definition, review, response) for all their projects;
- improve their user notification process and incident plans with special processes for critical incidents.

Both sides, finally, show margins of improvement in the area of Security Testing (where testing regularity and frequency is a common opportunity for practice enhancement).

Exchanges with the FOSS community have indeed dealt with the compatibility of formal project management techniques, since such community uses mechanisms as self-identification to distribute tasks and depend on voluntary contributions of their participants.[2]

In general, however, open source software development projects and the European institutions have similar concerns about software support. The two also share an approach to classify software requirements.

EU-FOSSA also approached another area of assessment of OSS projects: project sustainability, for which it has developed 34 metrics, divided in six categories:

- Community Activity;
- Performance;
- Quality and Security;
- Demographics and Diversity;
- Governance;
- FOSS support.

Each metric (defined through the following attributes: description, unit of measurement, method of measurement, frequency, responsible for the measurement), measures one of the factors

[2] https://creative-destruction.me/2016/07/11/eu-EU-FOSSA-needs-your-help-a-free-software-community-call-to-action/

contributing to ensure the project sustainability. The collective evaluation of such metrics provides therefore a view of the OSS project stability in the mid-long term.

1.2. The European institutions make significant use of open source software and standards

The European institutions make significant use of open source tools. For example, the European Commission and the European Parliament generally use open source tools and methods for software development. The institutions project management tools make room for agile, collaborative development cycles.

Commonly used open source tools include solutions for continuous integration and deployment of software, Jenkins and CruiseControl. These solutions let software developers manage updates of their code, which in turn helps to prevent errors.

Software developers working for the European Commission and the European Parliament also use open source code management tools Apache Subversion and GIT. The EC and EP software developers also commonly use Eclipse as integrated development environment, and Apache Maven as a build tool. For automated testing, they turn to Selenium.

In addition, software libraries related to security, such as OpenSSL and Spring Security, are standard development tools used in both institutions.

The significant use of Open Source Software has been somehow confirmed by the inventory performed within EU-FOSSA. The pilot inventory, performed on the IT assets managed by the Directorate General for Informatics (DG DIGIT)[3] of the European Commission, has identified 46243 software items, out of which 8226 (about 18%) are OSS (see Figure 2 on the right).

In terms of number of installations and deployments, OSS counts on 3.037.716 instances, out of 19.120.013 instances for all software installed or deployed, representing 16% of all instances.

The analysis of OSS by software types has shown that the vast majority of OSS, both in terms of items and in terms of instances, belongs to the "Library-utility" type, followed by the "Operating system" type, as per Figure 3 below.

Figure 2 - Inventoried software items and instances - OSS vs other software

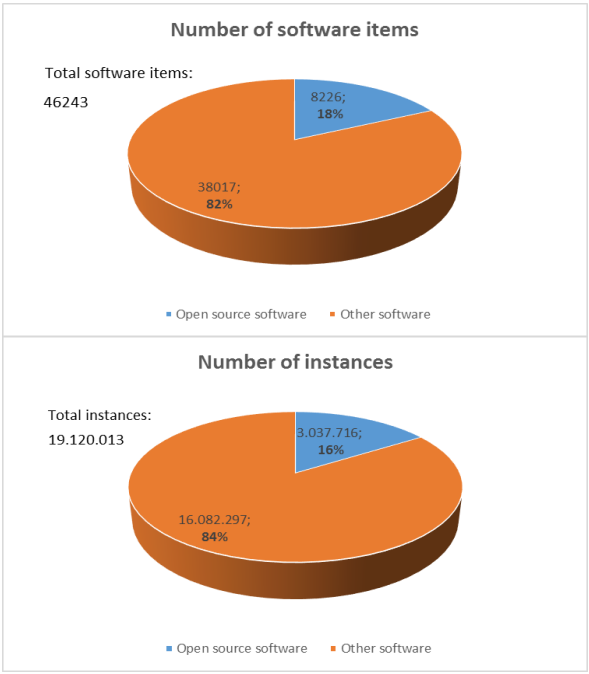
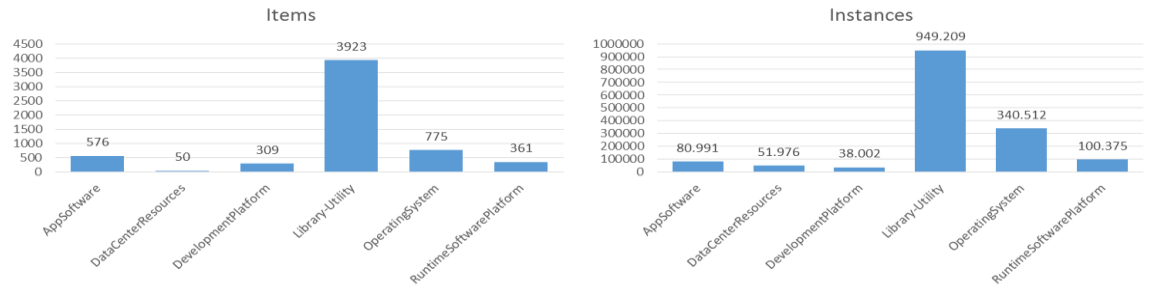


Figure 3 - OSS by software type



[3] With the exclusion of Datacenter and Desktop infrastructure, Mobile Devices and housed Datacenter applications

OSS is also widely used at the European Commission for a variety of purposes, as shown in the top-10 list of the most frequent items (see Table 1): from browsers (Firefox), to utilities (data compression: Info-ZIP, bzip2, 7-Zip), to media players (VLC), to E-book managers (Calibre), to libraries (OpenSSL), to tools commonly used for development purposes (Notepad++).

Table 1 - Top 10 inventoried OSS

Software Name	Total number of instances
Firefox	**43324**
Info-ZIP	**42890**
Firefox utilities	**42033**
VLC	**40434**
Calibre	**25506**
Calibre utilities	**12067**
bzip2	**6087**
7-Zip	**5143**
OpenSSL	**4918**
Notepad++	**4953**

Figure 4 - Critical software shortlist

Obtaining raw inventory data is only the first step to enhance the security of OSS components. EU-FOSSA has therefore submitted the inventoried OSS to a double assessment:

a) Evaluation of its criticality, by an index based on its use at the European institutions (both in terms of spread and exposure to end users) and its impact on security; this has led to the definition of a shortlist of critical software (see Figure 4);

b) Evaluation of the sustainability of the critical software shortlisted as per point a), based on the above mentioned 34 sustainability metrics.

This second analysis has shown that **the information necessary to calculate the various metrics is in several cases missing**, as OSS projects do not regularly track or document the drivers behind several of them, or the pertinent information must be searched through huge volumes of data with a massive effort. This is particularly the case for most Performance metrics, as the information is not usually made available (e.g. for the time spent on code reviews) or is often not available in a structured format (e.g. time to solve tickets).

With this caveat, the sustainability of the critical software measured as average of all metrics categories ranges from 20% (very low) for OpenLDAP to almost 80% (high) for Firefox, as per Figure 5 below.

Figure 5 - Overall sustainability of shortlisted OSS projects at the EC

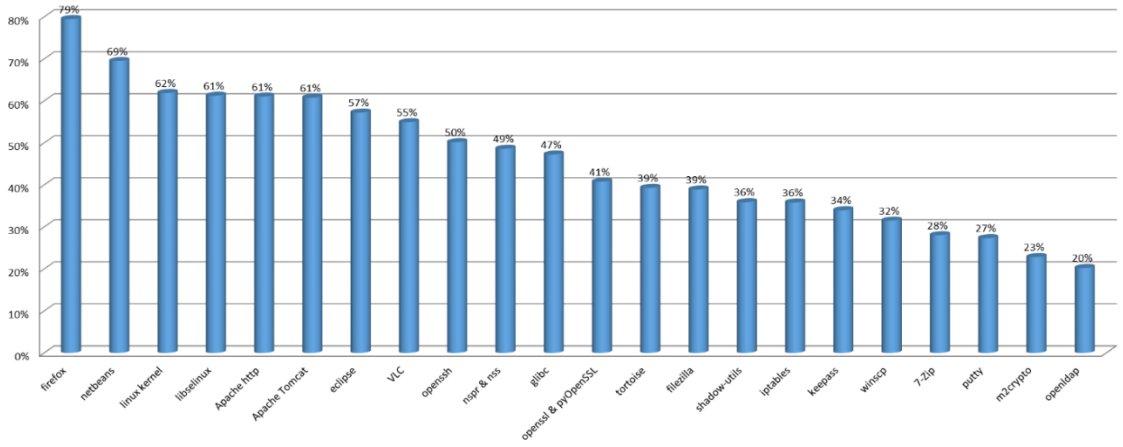
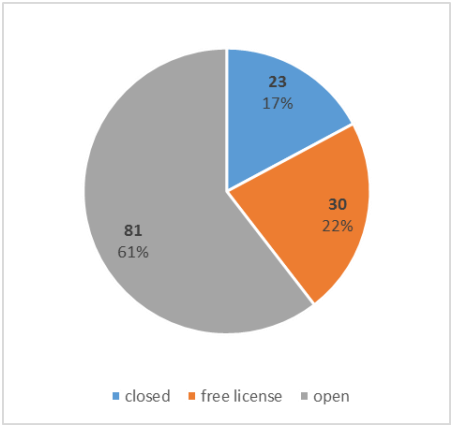


Figure 6 - Inventoried standards at the EC - by license type

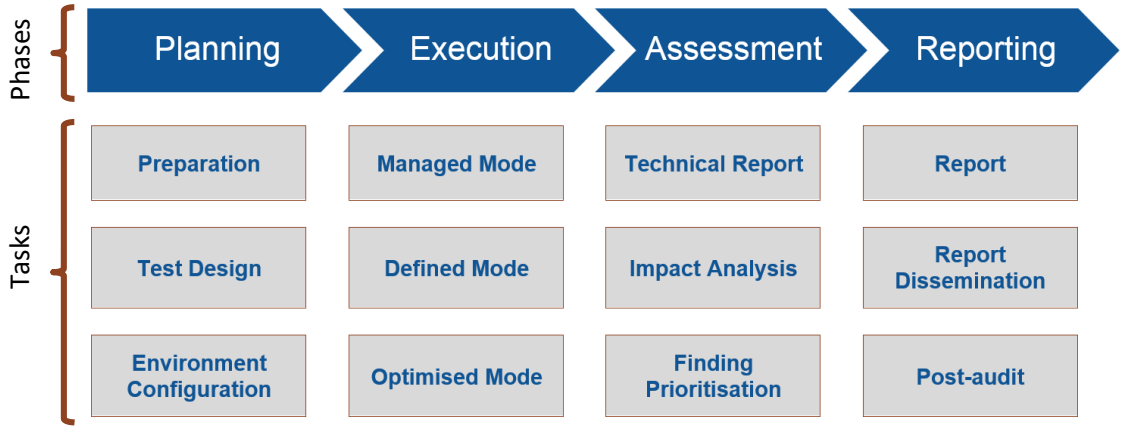


EU-FOSSA has also inventoried the standards in use at the European institutions and applied to the shortlist of critical OSS. 116 standards have been listed, mostly software standards (applied to software development and software operations, including operating systems, middleware, applications, and development environments: 46 occurrences) and information standards (relating to the representation, management, and interchange of information, both within and beyond the enterprise: 39 occurrences)[4]. Most of such standards (see Figure 6) are themselves open (in 61% of the cases), but a significant part of them are also patented under free license (22%) or closed (17%). In this latter case, the documentation and specification for such standards are not available to the public, enabling its developer to sell and license the code to manage their data format to other interested software developers. This aspect must be taken in consideration when aiming to foster the use of fully open software.

1.3. Definition and pilot application of a methodology for code review

EU-FOSSA has developed a Code Review Methodology covering 4 phases, shown in Figure 7:

Figure 7 - Phases of the code review methodology



[4] The taxonomy of standards has been based on "IT Governance: Managing Portfolios, Projects, Processes, and People", Butler Group, April 2007; p. 88

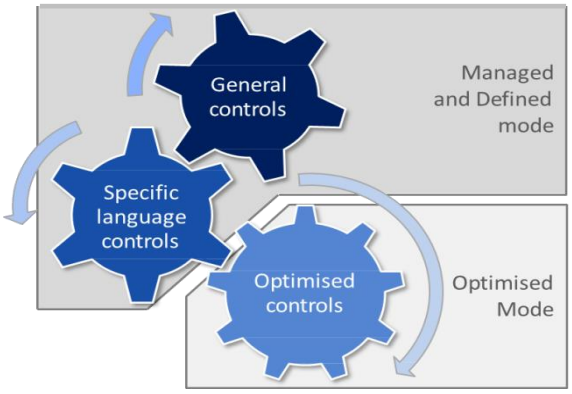
In particular, the **execution phase** covers three consecutive tasks:

- **Managed Mode**: use appropriately selected automated tools to perform general controls, providing initial results as a starting point for the other more manual-oriented tasks;
- **Defined Mode**: manual review of general controls;
- **Optimised Mode**: tests focused on the software architecture and unique peculiarities of the code.

Each mode provides further information and scope refining, allowing for a more in-depth analysis of the most critical sections of the code, as well as allowing the review of the vulnerabilities and weaknesses found.

The process followed depends on several sets of **controls**. These are grouped into three main ones (see Figure 8):

Figure 8 - Code review modes and controls



- **General controls**: those that apply to any code review, including architecture controls, regardless of the language;
- **Specific Language Controls**: those tests that only apply to specific languages, depending on their functions, characteristics, etc.;
- **Optimised Mode controls**: specific checks to test advanced features and review complex results.

The categories of controls performed in the various sets, particularly in the general and specific language ones, refer to:

- Data/Input Management;
- Authentication;
- Session Management;
- Authorisation Management;
- Cryptography;
- Error Handling/Information Leakage;
- Logging/Auditing;
- Secure Code Design;
- Specific programming language.

To engage the stakeholders, a workshop was conducted to review and validate the methodology while JoinUp[5] was the platform used to request feedback from the FOSS communities.

Before engaging in the code review of Open Source Software projects, EU-FOSSA suggests however to run a feasibility study providing a mechanism to analyse the viability of a code review project, based on the project requirements and the context of a specific code review.

The **project requirements** are gathered and classified in 5 aspects:

a. technical: evaluates the factors of knowledge and tools required to execute the project;
b. economic: evaluates the cost, cost limit and benefit factors;
c. legal: evaluates the tool license factor;

[5] https://joinup.ec.europa.eu/community/eu-fossa/description

d. operational: evaluates the project staff required, the stakeholders, the code, the code documentation, the methodology and the scope factors;

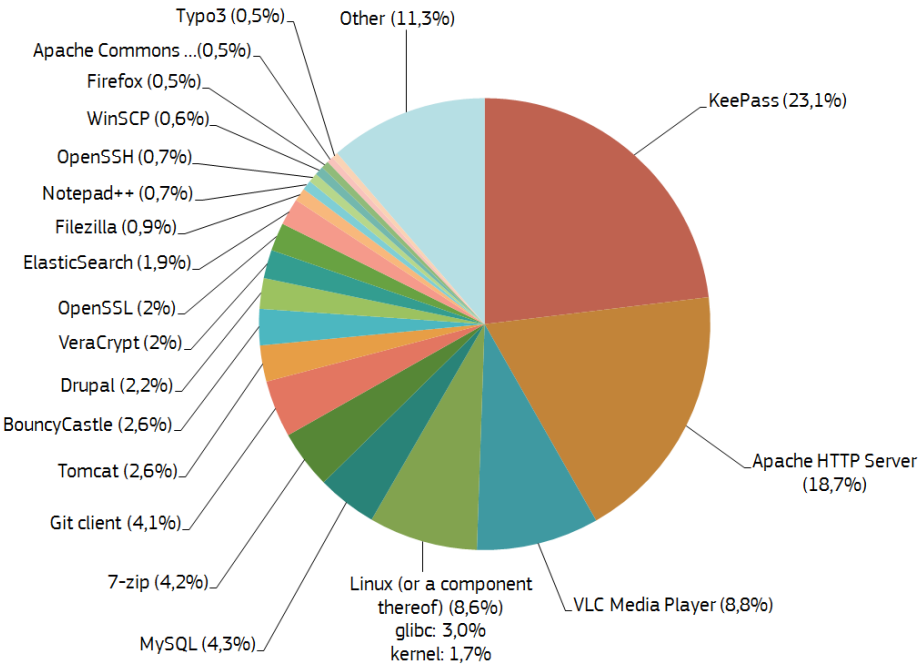
e. scheduling: evaluates the time and time limit factors.

Those requirements are evaluated vs. the project limitations, such as scope, time and cost. Such evaluation aims at determining whether the project is feasible or not. In the case that the result is negative, some of the requirements may be revised.

EU-FOSSA has tested the code review methodology on two code sections selected through a crossed evaluation of two inputs:

1. A **survey** submitted to the general public, asking the participants to provide their recommendations on the most critical software to scan for security issues[6]. Almost 3.300 answers were received, with KeePass and Apache HTTP Server making more than 40% of the total preferences (see Figure 9).

Figure 9 - Survey on critical OSS - results



2. **the shortlist of critical OSS** drawn on the basis of the inventory performed within the EU-FOSSA project, selecting the software based on its use at the European institutions (both in terms of spread and exposure to end users) and its impact on security.

A crossed analysis of the two inputs (see Table 2 below) led to the choice of **Apache HTTP Server** (and in particular, of its core and APR modules) and **KeePass** as candidates for the code review.

[6] The survey results, along with the criteria used for the selection of open source software codes to be reviewed, are published at https://joinup.ec.europa.eu/community/eu-EU-FOSSA/news/results-eu-EU-FOSSA-survey

Table 2 - Criteria for the selection of OSS components to review

In brackets: percentage of survey votes		Use at the EC and EP	
		Not widely used (80%)	Widely used (<=80% in a given domain in the EC or EP)
Security	Security-related (handling critical infrastructure or access thereto)	**KeePass (23.1%)** **MySQL (4.3%)** **git client (4.1%)** ElasticSearch (1.9%) Filezilla (0.9%) WinSCP (0.6%) OpenSSH (0.6%)	**Apache HTTP Server (18.7%)** **Linux (ora a part of it) (8.6%)** **(in particular glibc: 3.0; kernel: 1.7%)** **Tomcat (2.6%)** **BouncyCastle (2.6%)** **Drupal (2.2%)** **OpenSSL (2%)**
	Not directly security related	Notepad++ (0.7%)	**VLC media player (8.8%)** **7-zip (4.3%)** Firefox (0.5%)

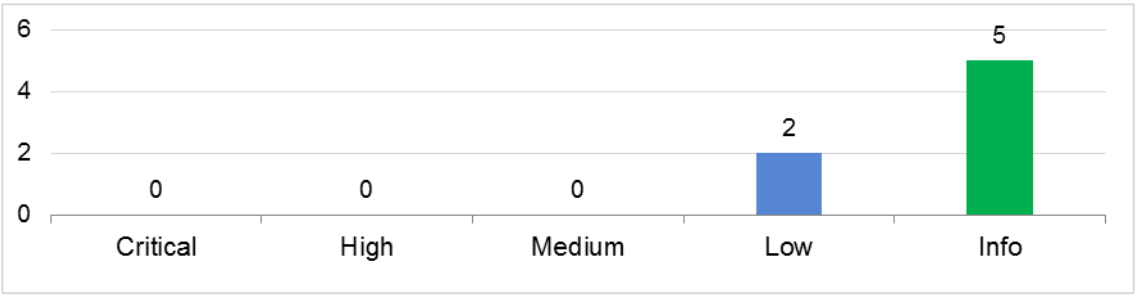
red: more than 5% of survey votes
bold: more than 2% of survey votes

In particular, the main rationales for the choice of Apache HTTP Server were the wide usage both in the EU institutions and externally, on top of the high level of preferences in the survey. The challenge linked to the complexity of the software was mitigated by focusing on its core and APR modules. As for KeePass, the elements in favour of its choice were mostly its top position in the preferences among the survey takers and its criticality from the security point of view.

The **Apache HTTP Server code review** was conducted on 61.286 lines from the Apache HTTP server core and the Apache Portable Runtime (about 20% of the total Lines of Code - LoC). Those modules were selected because they are responsible for interacting with the operating system (while most tests focus on the interaction with the outside) and they are critical from a security point of view as many machine resources are accessed in this part of the software. If this code is not secure, a potential attacker might gain access to the machine.

Only 7 findings were detected out of 160 controls reviewed, within the categories of Secure Code Design and, mostly, the Language-specific controls, with no findings in the other categories. **No critical, high-risk or medium-risk findings were detected**, therefore **Apache HTTP Server shows a good security level**.

Figure 10 - Findings of Apache HTTP Server review



The fact that most findings were related to the Language-specific controls is particularly linked to the fact that C, the programming language used in this code, is a complex language from a security point of view. The C language allows direct access to memory, and the lack of exception support means that errors and exceptions have to be controlled manually in the code.

The classification of findings by criticality also determines the timeline for their solution: short-term for the medium risk one, mid-term for the low risk and long term (without any pre-defined priority) for the purely informative ones.

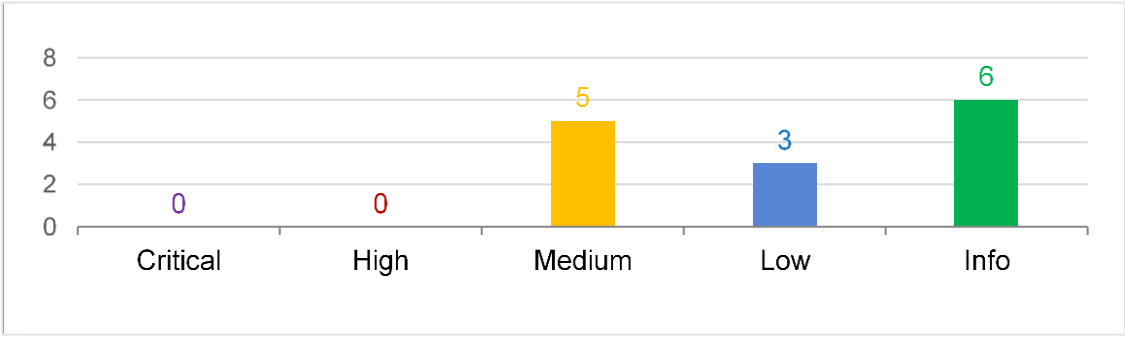
It is advisable for future actions to analyse the rest of Apache HTTP Server, as well as the rest of the libraries that it uses. This software is widely used as a web server and proxy server, and it should be analysed according to that fact.

The **KeePass code review** was conducted on all of 84.622 LoC of the application.

Also in this case, the total number of findings (14) can be considered low when compared to the 218 controls reviewed, with **no critical or high risk issues**. The concerned control categories were:

- Error Handling / Information Leakage
- Logging / Auditing
- Secure Code Design
- Specific language controls (C / C++)

Figure 11 - Findings of KeePass code review



The number of affected language-specific controls was, also for KeePass, the majority (10 out of 14). Again, this is particularly linked to the use of C (and C++) as programming language. It is also necessary to underline, for this specific application that is used for the management and storage of passwords, the relevance of controls related to the kind of randomisation used for cryptography. In general terms, **KeePass shows a good security level**.

All in all, around 2/3 of the controls were performed in the optimised (mostly manual) mode, through which almost 80% of the failed controls were discovered. This highlights the weight of the manual controls over those performed with the support of automated tools.

2. COMMUNICATION AND COOPERATION WITH THE OSS COMMUNITIES

Due to the nature of the subject treated by the EU-FOSSA pilot project, the cooperation and communication with the OSS communities has been a point of attention throughout the project leadtime.

In practical terms, this has been realised through a series of actions such as:

- The publication of all project deliverables on the EU-FOSSA community in Joinup[7]. Furthermore, in order to facilitate the exchange and discussion with the communities on the project outcomes, a direct communication channel has been established with the communities initially contacted for the review of best practices, proposing them to participate to the review of the deliverables via a dedicated forum focused on specific items of the deliverables;
- A workshop to present the code review methodology designed by the project team to representatives of the European institutions and of the communities and to get their feedback and contribution to the finalisation of the methodology;
- The survey that has given the general public, and in particular the members of OSS communities, the opportunity to select the candidate projects for the code review: not by chance, the two selected projects were the top preferences of the participants to the survey;

[7] https://joinup.ec.europa.eu/community/eu-fossa/og_page/project-deliveries

- The direct exchange on the results of the code review with the involved communities (Apache and KeePass).

Furthermore, a stream of news was published on the Joinup community[8] in order to maintain momentum on the project activities, informing the public on its main achievements.

3. WHAT HAS EU-FOSSA ACHIEVED?

In order to assess the achievements of the EU-FOSSA pilot project, one must go back to the user needs and success criteria identified at the project launch.

From this point of view, EU-FOSSA has contributed to the **assessment of the current security of OSS components in use at the European institutions** in three ways:

- Ensuring that the OSS in use is properly detected and inventoried through a consolidated inventory methodology;
- Implementing a code review methodology ensuring that the software they use is thoroughly checked against security weaknesses and vulnerabilities;
- Assessing the OSS in use under both a business criticality and a sustainability point of view.

Communities will greatly benefit from the results of the code reviews carried out by the European institutions regarding their software, receiving detailed vulnerability and weakness information.

As seen above, the assessment of current OSS security has been made possible by the **implementation of appropriate metrics**, to measure both the OSS project criticality (from a business point of view) and its sustainability (from the point of view of the capability of the community to maintain a continuous support to the projects).

As for the **contribution to the OSS communities**, EU-FOSSA was launched in a spirit of cooperation with them. In practical terms, while all communication actions mostly targeted this approach, the main concrete opportunity for a direct exchange and cooperation with the communities has been the direct exchange on the results of the code review with the involved communities (Apache and KeePass).

In fact, as it has already been emphasised, successful industry leaders or project managers in the open source community frequently employ peer review techniques as criteria for quality control in their development cycle[9].

This last point is particularly critical, as the nature of open source software development often makes standardized code appraisal difficult to achieve. As such, EU-FOSSA has provided its contribution by proposing a **comprehensive code review approach** that is flexible enough to be adapted to various projects and communities.

EU-FOSSA has also dealt with process and organizational matters. For example, roles and responsibilities for the code review activities were outlined. Another example is the definition of the inventory process, suggesting the creation of an “inventory management” role, governing the process especially for all “pull” activities.

As a pilot project, EU-FOSSA has identified the **best practices** applicable in the various areas of analysis, for software development, inventory management and code review activities.

The project methodologies (code review and inventory management) are also meant to be repeatable and to be applied not only at fixed dates, but also – and even more effectively – when a new project is launched or a new OSS component is installed.

[8] https://joinup.ec.europa.eu/community/eu-fossa/communications/all

[9] https://opensource.com/business/14/12/importance-peer-review-code

4. CHALLENGES

The main challenges emerged in the course of the project can be grouped into four areas: data quality, data availability, effort and communication / cooperation.

Data quality:

- Some inventory information is treated differently among the various input databases. Non homogeneous data have therefore been partially normalised by mass treatment based on hardcoded "where" clauses, but also manually (e.g. SoftwareName). In some cases, where it was not possible to complete the missing data or normalise them due to lack of information (e.g. OrganisationName), the pertinent field was disregarded for the pilot project.

Data availability:

- Information from the European Parliament has not been sufficient to proceed with the inventory of OSS software and standards at that institution. This is mostly due to the manual content of the data collection process and the lack of homogeneous inputs for the provision of the data.
- When applying the sustainability metrics (defined by WP1) on the inventoried software shortlist, information was not available on various projects, or was not consolidated and structured. In this case, either the missing information was evaluated based on qualitative parameters instead of the original quantitative ones, or it was disregarded, at least within the pilot project.
- Due to the nature of the FOSS Communities, it was difficult to gather information on their software development practices directly from their representatives. This issue was mitigated by obtaining information from trusted sites such as the FOSS Communities websites, recognised wikis and forums.

Effort

- For the inventory, large amounts of data had to be treated manually due to their unstructured nature and the impossibility of developing a more automated solution within the pilot project;
- The workload linked to the review of very large codes was unmanageable with a traditional sequential approach. The code was therefore divided into modules (code libraries, etc.) and then in "batches" (sets of files related to each other) on which multiple reviewer may work in parallel.

Communication / cooperation

- Notwithstanding the efforts to involve the OSS communities more actively, at least in the first part of the project mostly through the communication actions, one of the main challenges was to obtain an active involvement and cooperation of the open source communities. Additional communication actions, not included in the original communication plan (e.g. increase the frequency of news, open a forum and a code review log, publish the deliverables by sections) have been put in place;
- The difference in the working methods of the corporate world and of the EU institutions vs. those of the OSS communities (e.g. the use of articulated, detailed and long reports vs. the preference for shorter, more compact messages in the OSS communities) has also somehow hindered the cooperation and exchanges.

5. LESSONS LEARNED

As a pilot project, EU-FOSSA has been an opportunity to test on a relatively limited scope a new approach for the European institutions to contribute to the security of its ecosystem and of OSS in general. It has therefore offered an opportunity to capitalise on the experience gained in view of possible future continuations, in a series of areas that can be summarised as follows.

Code review methodology and scope:

The code review methodology developed by EU-FOSSA is meant to offer a high degree of flexibility and scalability, covering any code review scenario that could be required by the European Commission, DIGIT or any other European Institution. However, its use in the test code reviews has shown the need for further refinements of the general security controls, to improve their applicability to a wider range of OSS codes, aimed at:

- Improving their specificity, so that they provide enough detail for code reviewers to check them, and that they are applicable regardless of the language in which the code is reviewed;
- Widening their focus, from mostly interconnected applications for desktop software to other applications such as mobile and server solutions;
- Implementing controls on sockets.

Furthermore, the application of the methodology has shown the need to optimise the efficiency of controls, both by filtering the ones that are most relevant for a specific code and by making a flexible use of the control modes also by mixing them (e.g. with a joint use of manual aided by automated checks).

An additional reflection concerns the selection of the project for code review, which should be based on the community size, as larger communities may already have their own code review processes and more resources to commit to this activity.

Code review execution:

The pilot code review has shown the need to improve the efficiency of the code review operations, in:

- the collection and treatment of data for the subsequent analysis (where a dedicated application may be developed for this purpose);
- the execution of manual controls (by parallelising working queues, supporting them with automated controls, modularising the code review in a structured way to allow the parallel action of several analysts, including the analysis of functions calling to external libraries or functions).

The parallelisation of working queues has been considered particularly to cope with the additional time needed to discover all occurrences of a certain finding, information in which the communities are also interested.

In general, an effective planning phase is essential for the success of the code review, in order to adapt the methodology to the type of software analysed, organising differently the various working queues and allocating the available resources among them, based on the requested expertise and the time available.

Additionally, the experience in the pilot project has provided parameters to assess the workload of the code review operations in a way that shall be useful to plan and estimate the effort for similar future exercises.

Code review results analysis and presentation:

The report structures for the presentation of results may be further refined so that no relevant information risks being missed (and that, on the other hand, only relevant information is shown), reducing the effort of a subsequent Quality Assurance step.

Additionally, the communication with the communities may be facilitated by presenting the results of the code review in a more compact way, focusing the attention directly on the findings and including the considerations and recommendations in the same document containing such results rather than in separate reports.

Selection of tools for the software inventory:

When leaving to the project owner the possibility to choose with a wide liberty the tools to perform a certain activity (as in this case the inventory building), it is necessary to take into

account for the sake of project planning the learning curve necessary for the service provider to familiarise with the use of the selected tools.

Software inventory:

Considering the significant manual workload in the elaboration of inventory data, it is recommended to complement the data model underlying the inventory process to include attributes facilitating the automated labelling and grouping of software items. This, however, depends heavily on the availability of more complete and reliable input data.

One major value added of inventories is to allow comparison between situations in different moments, by achieving a level of detail and of information content comparable with the past inventories. It is recommended, for future similar exercises, to take those past inventories as a basis in order to define in a timely manner the information requirements and the approach to data treatment.

Cooperation and communication with OSS communities:

Under several respects, a more direct involvement of representatives of the OSS community, if not as full members of the project team but at least as an external support, as liaison officer with the OSS world, may have facilitated some tasks.

More specifically, while some communities require to interact for the submission of vulnerabilities (also discovered through similar code review exercises) through their mailing lists, it has been suggested to establish key contact points in the concerned communities to facilitate the direct discussion with the European institutions on the code review results.

6. WHAT'S NEXT?

EU-FOSSA, once tested the validity of its security assurance and open collaboration concept, has shown a number of opportunities for future continuation of its activity, by widening its scope, or by improving the solutions applied in the course of the project itself. A first indicative list of such opportunities, for both the code review and the inventory activities, may include the following items.

Code review:

The code review methodology is, alongside software, in **continuous evolution** in order to meet the ever-changing challenges brought by newer software, new programming languages and frameworks, and the new and varied threats that continuously challenge their security. It will be necessary, both in a project continuation and in the ongoing deployment of the methodology, to further refine and adapt the methodology to different software types and potential different new threats. Furthermore, the iterative nature of the process makes so that extending the exercise to other codes should improve the methodology by identifying potential new techniques, processes and tools or refining those already used.

Part of such evolution may be to expand the scope of the security tests to **include the solution of the security issues detected** during those tests.

Furthermore, the information acquired through the code review may be **integrated in the software development practices** to improve the security of EU institutions systems and applications. This may be done, for example, by creating security development guides and study material for software developers, to improve the security of the European systems and applications.

The code review must however be considered as a part of wider security framework, along with **other kinds of security audits** (e.g. penetration testing), in order to ensure a more complete detection of possible security issues in any software.

An **application may be developed** for the purpose of collecting evidences during the code review, allow the concurrent use of the same database, as well as providing an easy-to-use interface, help and guidelines and an easy way of providing assessment. From the point of view of the report,

this tool would also automatise the process of generating and providing a final assessment, graphs and indicators (such as CVE/CWE export data).

Inventory management:

A **comprehensive data quality approach**, rationalising and normalising all data sources, shall significantly benefit both the efficiency of the inventory process and its effectiveness, thus obtaining inherently reliable outputs, less exposed to errors due to manual operations.

The IT assets management practices developed for the EU-FOSSA project **may be extended to other units of DIGIT** that might be interested in the improvement of the current inventory approach, in order to build a consistent solution with a potentially wider coverage.

Even if maintaining the same organisational scope, there is still room for **extending the inventory coverage** by including other DIGIT infrastructure, including the webapp layer, the Datacenter physical layer, the mobile software...

The inventory process implemented in the pilot project is a "pull" process, calling for data when the inventory is run and therefore after a certain modification in the inventory has been made (e.g. adding a software item), so that a potential threat may be discovered only at a later stage. During the project it was not possible, indeed, to install agents in the servers to search automatically for the data needed. Future follow-up projects may consider **setting up an automatic, "push" approach** for the provision of inventory data, possibly by installing agents in the data servers, implementing a consolidated CMDB and recurring to large libraries to gather more information on the inventoried software.

Communication and cooperation

Joinup is a central and recognised source of info on the project promoted by the EU institutions. However, FOSS communities use to communicate other platforms (or direct channels, such as mailing lists, for specific purposes such as the submission of verified security issues). Different communication channels with the communities may therefore be considered for a potential project continuation, such as, for example, the participation to recognised forums, mailing lists, wikis or online publications of the FOSS world, in addition to Joinup. Traffic in Joinup related to future initiatives continuing the experience of EU-FOSSA may indeed be incremented thanks to the synergy with those channels.

Furthermore, the project is the starting point of a new model of cooperation among European institutions, the FOSS communities and the corporate world, which is *per se* a significant value added. In the longer term, opportunities for cooperation with the FOSS communities may arise on the side of the European institutions by:

- Collaborating in the FOSS development, either in an official way or by allowing the developers of the European institutions to contribute to FOSS communities;
- Creating documentation and guides for FOSS communities to improve software security;
- Developing subject matter experts (SMEs) in areas such as secure software and security in the software lifecycle;
- Creating open wikis of security experts to exchange information and best practices in software development;
- Creating webinars for developers about secure coding or about security in essential technologies used in software development (HTTP, etc.);
- Promoting the sponsorship of FOSS software regarded as a key part of IT infrastructure.

Last but not least, EU-FOSSA shall be an input of the OSS strategy of the European Commission. Thanks to an improved understanding of the use and security of OSS in the European institutions, it will be possible to promote in a more effective way the FOSS software within the European institutions by increasing its usage where appropriate, by contributing to the development of FOSS software or by helping with its dissemination.

APPENDIX 1: REFERENCES AND RELATED DOCUMENTS

ID	Reference or Related Document	Source or Link/Location
1	*Project deliverables – public releases*	*https://joinup.ec.europa.eu/community/eu-EU-FOSSA/og_page/project-deliveries*
2	*Project news*	*https://joinup.ec.europa.eu/community/eu-EU-FOSSA/communications/all*