WP6

DIGIT B1 - EP Pilot Project 645

Deliverable 1: Code Review Results Report

Apache Core & APR

Specific contract n°226 under Framework Contract n° DI/07172 – ABCIII

September 2016

DIGIT Fossa WP6 – Governance and Quality of Software Code – Auditing of Free and Open Source Software.

Deliverable 1: Apache Core & APR Code Review Results

Author:

Disclaimer

DIGIT Fossa WP6 – Governance and Quality of Software Code – Auditing of Free and Open Source Software.

Deliverable 1: Apache Core & APR Code Review Results

Report Summary

Title	**Apache Core & APR**		
Project Owner	Apache Community		
DIGIT Sponsor	EU-FOSSA project		
Author	DIGIT		
Type	Public		
Version	V 0.5	**Version date**	10/10/2016
Reviewed by	EU-FOSSA Team	**Revision date**	31/10/2016
Approved by	European Commission - Directorate-General for Informatics (DIGIT)	**Approval date**	To be approved
		Nº Pages	**36**

Distribution list

Name and surname	Area	Copies
IT contacts	To be identified	To be identified
FOSS Communities	Apache security Team	1

Contents

List of Tables

List of Figures

Acronyms and Abbreviations

API	Application Programming Interface
APR	Apache Portable Runtime
DG	Directorate General
EC	European Commission
FOSS	Free and Open Source Software
FOSSA	Free and open Source Software Auditing
GUI	Graphic User Interface
IDE	Integrated Development Environment
WP	Work Package

1 INTRODUCTION

1.1. Context

The security of the applications used nowadays has become a major concern for organisations, companies and citizens in general. Applications are becoming a more common part of our daily lives, and are being used for business and leisure purposes alike. The information managed by these applications has become the most essential asset to protect, as it includes personal information, internal data, industrial property, etc.

From a security point of view, this new scenario presents many new challenges that need to be addressed in order to protect the integrity and confidentiality of the data managed by the applications and their users.

Furthermore, the exposure of the applications to the Internet has turned them into a prime target, due to the value that this private and internal information has.

One of the advantages of Free and Open-Source Software (FOSS) is that its source code is readily available for review by anyone, and therefore it virtually enables any user to check and provide new features and fixes, including security ones. Also, from a more professional point of view, it allows organisations to review the code completely and to find the weaknesses that it presents, allowing for a refinement of their security and ending up in a safer experience for all the users of the applications.

1.2. Objective

The objective of this document is to provide the results of the code review of **Apache Core & APR** software. This review was carried out within the EU-FOSSA (Free and Open-Source Software Auditing) project, focusing on the security aspects of the software.

The objective of this code review is to examine the **Apache Core & APR** software, focusing mainly on its security aspects, the risk that they pose to its users and the integrity and confidentiality of the data contained within.

Apache HTTP Server is one the most used HTTP and proxy servers and it is a FOSS. It is a mature FOSS project running since 1995 and many security flaws have been detected and corrected since its conception.

1.3. Scope

The scope of the project is as follows:

Application name	**Apache Core & APR**			**Review start**	25/07/2016
Code review owner	European Commission - Directorate-General for Informatics (DIGIT)			**Review end**	22/08/2016
Objective	Security Code Review				
Num. Lines	61 286	**Version**	Apache 2.4.23 APR 1.5.2	**Programming language**	**C**
Code Review Mode	✓ 1-Managed	✓ 2-Defined	✓ 3-Optimised		
Libraries	• Apache Core (version 2.4.23) • Apache Portable Runtime (APR, version v1.5.2)				
Extensions/plugins	N/A				
Services required	N/A				
Result visibility	✓ Internal	✓ Restricted	✓ Public		
Critical notification	During assessment	Apache Security Group			

Categories	Data/Input Management	✓	Error Handling / Information Leakage	✓	Specific C controls	✓
	Authentication Controls	✓	Software Communications	✓	Specific C++ controls	X
	Session Management	✓	Logging/Auditing	✓	Specific JAVA controls	X
	Authorisation Management	✓	Secure Code Design	✓	Specific PHP controls	X
	Cryptography	✓	Optimised Mode Controls	✓		
Comments	The code review of the Apache Server includes: 1. Apache Core module 2. APR library					

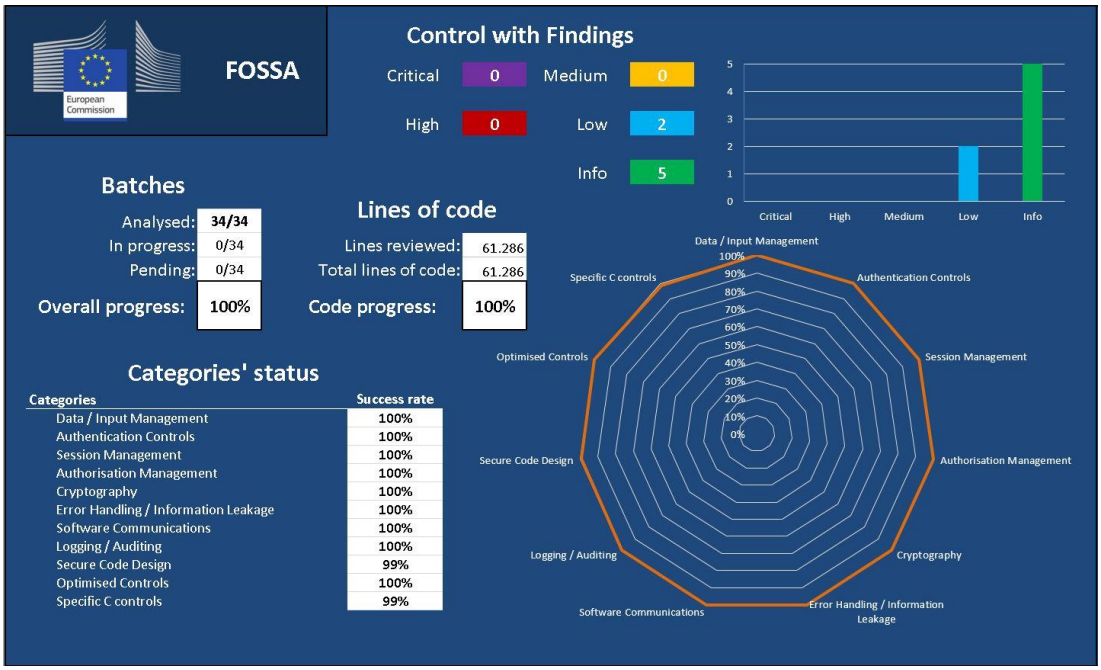
1.4. Deliverables

1 *WP2 - Deliverable 11: Design of the methods for performing the code reviews List of methods for communicating the results of code reviews*

2 EXECUTIVE SUMMARY

The results obtained from the controls evaluated during this code review provided a number of relevant findings regarding potential weaknesses of the application reviewed. A general overview of the results is depicted in **Error! Reference source not found.**, which shows a summary of the code review, listing the findings, their severity and the categories affected by them. As it can be observed, the number of findings within each category is small compared to the total number of controls reviewed, thus resulting in the graph shown on the bottom right section of the figure.

Figure 1: General overview



The Apache HTTP Server is composed of multiple components and relies heavily on the use of extensions and modules to include additional features (e.g. secure communications, encryption, proxy functionality, etc.). This code review, as it is a pilot included of the EU-FOSSA project, has focused primarily on the core sections of Apache, excluding at the moment any external module or extension.

The modules selected are the Apache HTTP Core and the Apache Portable Runtime (APR), which comprises a total of **61 286** lines of code reviewed, approximately 20% of the total lines from the Apache HTTP Server and common modules. The Lines of Code (LoC) were grouped into 34 sets (or 'batches') of code in order to optimise the process, allowing a distributed process among the EU-FOSSA project code review teams.

Furthermore, during the grouping process, only code pertaining to Windows and UNIX host systems was considered (other operating systems were discarded for this pilot, which are candidates to review if the complete Apache server code is reviewed in the future). These operating systems are the ones mostly used in production environments, and therefore are critical

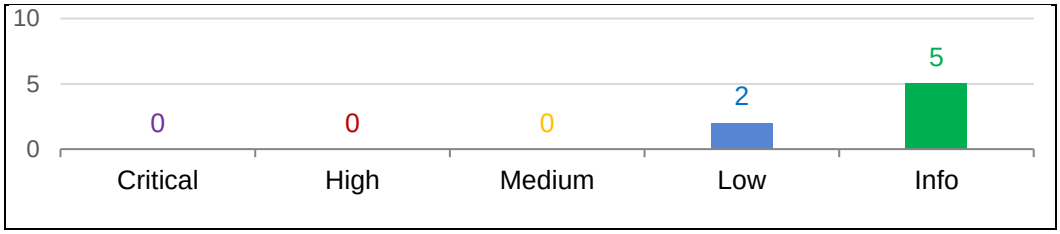
from a security point of view. If any part of this code is not secure, it could provide a potential entry point for an attacker in order to gain access to the machine/server. On the other hand, it is necessary to take into account the fact that the Apache HTTP Server is a world-wide used and highly regarded software which is continuously being tested and evaluated, mostly the front-end section (external access) via audits and pentesting analysis. However, the backbone core code is not tested as often, and being the part that interacts with the operating system it is of considerable interest to evaluate its current security standing.

Moving on to the code review, and in relation to the control categories, it is noteworthy the fact that several findings were discovered on the code, all of them within the **Secure Code Design** and **Specific C controls** categories.

A total of **7 controls**, from the **160** controls reviewed, had at least one finding detected in them, which can be considered a low percentage overall. The remaining categories passed successfully with no relevant findings.

A summary of the findings is depicted in Figure 2, comparing the failed controls found and indicating their distribution within the different risk levels.

Figure 2: Risk Level



After a detailed and careful evaluation of these findings, it was concluded that none of them were **critical;** in fact nor were them of **high** or **medium** risk level**.** The findings were determined to be: 2 low-risks and 5 of an informative nature. Furthermore, 4 of these affect libraries used exclusively during the compilation support libraries, and should be considered, although the risk they pose is not direct. Overall the results were:

- ✓ **Critical findings**

  No critical weaknesses were found in this code review.
- ✓ **High risk findings**

  No high risk weaknesses were found in this code review.
- ✓ **Medium risk findings**

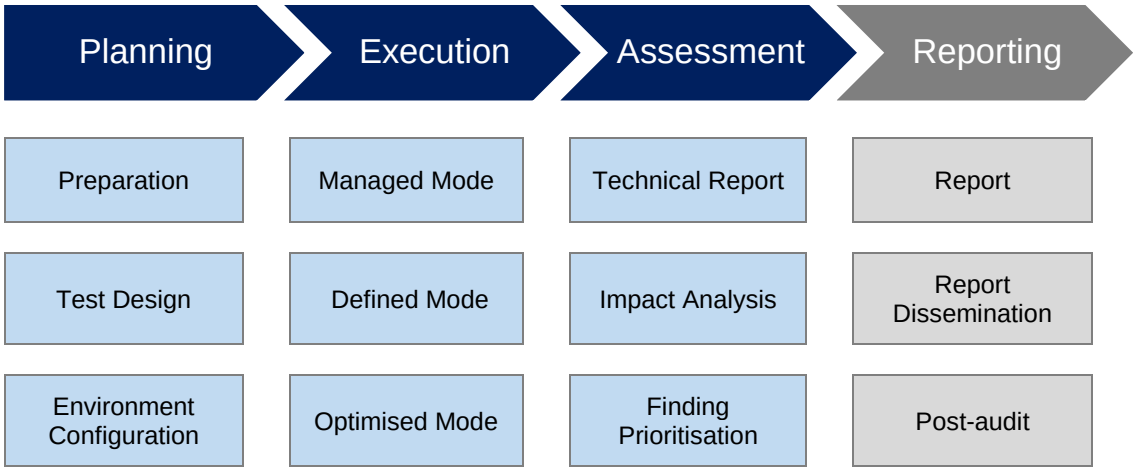
  No medium risk weaknesses were found in this code review.

Therefore the findings were classified as either low or informative, and while they are still relevant and should be eventually fixed, their impact does not justify a short-term or emergency fix process.

3 METHODOLOGY

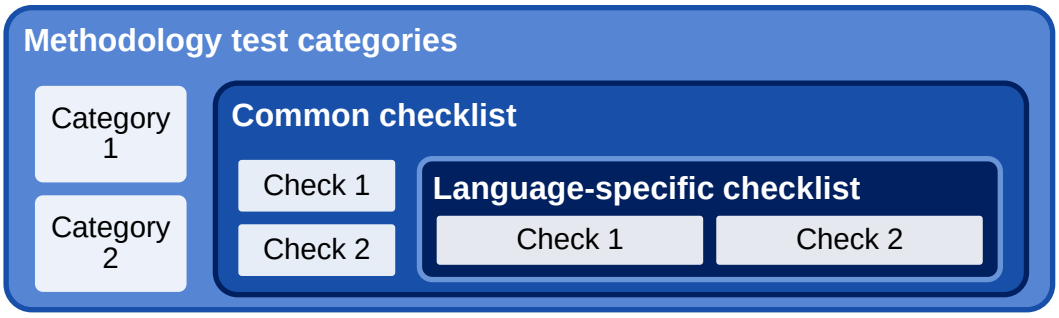
The methodology followed to carry out the code review is summarised in Figure 3. This methodology covers from the initial planning phase to an optional post-audit support phase. Furthermore, each of these phases is divided into several major tasks.

Figure 3: Methodology phases



During the execution phase, a set of controls is checked by the code reviewers in order to properly verify the security and stability of the code. These controls and checks are grouped in a checklist presented in *Section 4.3. Overview of Results*, to facilitate the reading of the findings.

Figure 4: Test category levels



As seen in Figure 4, there are two main groups of controls: the common ones (applicable regardless of the language of the code) and language-specific controls (for C, C++, JAVA or PHP). A combination of both should be used in any code review to ensure the most accurate results (explained in WP2 - Deliverable 11: *Design of the methods for performing the code reviews*).

3.1. Planning

The first phase of the methodology covers the information gathering activities needed in order to properly plan and carry out the code review. This includes the compilation of basic information about the code to be reviewed, an analysis of the applicable test cases and the preparation of the testing environment if any specific requirements are demanded by the particularities of the code.

This information was obtained from the stakeholders requesting the code review and from the developers or IT maintainers where applicable. Once this phase is finished, all needs should have been met in order to start the test cases.

To further organise this phase, three main activities have been defined:

- **Preparation**: this activity comprises all the interviews, meetings and information gathering activities needed to properly define the scope, objectives and needs of the code to be reviewed.

- **Test Design**: once the scope, objectives and custom needs of the code have been identified, the next logical step is to establish the test cases that are going to be considered in order to achieve the objectives that have been set. This is reflected in the checklist, indicating those cases that are not applicable.

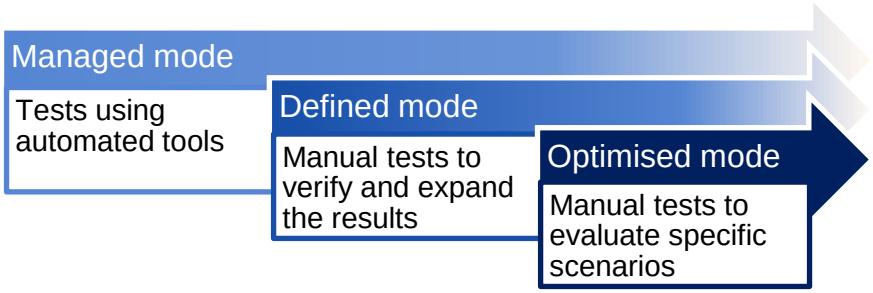
- **Environment Preparation**: before starting the next phase, it is necessary to ensure that the testing environment is prepared to carry out the tests selected during the previous activity. This includes the installation and configuration of the tools.

3.2. Execution

The next phase covers the execution of the test cases selected for the code review in the previous phase, taking into consideration the scope, objectives and constraints set.

The execution process was divided into three sequential phases, each providing data as input for the next one, as depicted in Figure 5. All of them were carried out by the code review team, using both automated and manual tools.

Figure 5: Code review execution order



To further organise this phase, three main activities have been defined:

- **Managed mode**: this activity covers the execution of the automated tools selected for the analysis of the code. The following categories were analysed:
  - *Data/Input Management (DIM)*: The data entry points of an application, service or library are one of the weak points that must be controlled against unexpected values. The subcategories covered are as follows:
    - File Input / Output Management (FIM)
    - Data stream management (DSM)
    - Character encoding management (CEM)
    - Input validation and sanitisation (IVS)
    - Sensitive Data Management (SDM)
    - Entry point validation (EPV)
    - XML schema validation (XSV)
  - *Authentication Controls (AUT)*: It covers any aspect related to the process during which the solution reviews and verifies the identity of another entity, such as a user. The subcategories covered are as follows:
    - Authentication verification (AUV)
    - Password policy usage (PPU)
    - Credential storage security (CST)
    - User account protection (UAP)
    - Password recovery process (PRP)
  - *Session Management (SMG)*: It covers all parts of the protection and management of user sessions once they are authenticated against the solution. The subcategories covered are as follows:
    - Session creation (SCP)
    - Session ID management (SID)
    - Session lifecycle (SLC)
    - Session logout (LGP)

- *Authorisation Management (ATS)*: This process is designed to ensure that when a user or entity correctly authenticates against the application, s/he gets the proper privileges assigned to it. The subcategories covered are as follows:
  - Access control system (ACS)
  - Privilege revision (PRV)
- *Cryptography (CPT)*: Covers all aspects related to the protection via encryption of the information and data in transit and at rest. The subcategories covered are as follows:
  - Credential protection at rest (CPR)
  - Cryptographic configuration (CRC)
- *Error Handling/Information Leakage (EHI)*: The information provided by the application errors, page metadata and sample content must be filtered to avoid a leakage of sensitive information. The subcategories covered are as follows:
  - Information leakage (INL)
  - Sample files (SFL)
  - Error handling (EHD)
- *Software communications (COM)*: it comprises those functions that manage and control network connections, including sockets and protocol functions. The subcategories covered are as follows:
  - HTTP Secure Management (HSM)
- *Logging/Auditing (LOG)*: The logs generated by an application are a superb source of information about its contents, workings and potential weaknesses. The subcategories covered are as follows:
  - Log configuration management (CFG)
  - Log generation (GEN)
  - Log sensitive information (LSI)
- *Secure Code Design*: There are several aspects related to the application itself and the technologies and frameworks used for its implementation. The subcategories are as follows:
  - Framework requirements (FWK)
  - Variable types / operations (VTY)
  - Expressions/Methods (EXM)

- **Defined Mode**: once the managed mode activity is finished, the code review team will have a set of results generated from the automated tools. These results, together with the manual tests needed, are checked in order to fill the controls and checks that will provide the final results.

- **Optimised Mode**: the final part of the execution phase focuses on those sections of the application that are found to be most at risk, alongside several more specific tests that require further evaluation. They are divided into the following subcategories:
    - Concurrency (CCR)
    - Denial of Service (DOS)
    - Memory and resource management (MRM)
    - Code Structure (COS)
    - Role-privilege matrix (RPM)

  The optimised mode covers the set of language-specific (C, C++, JAVA and PHP) controls, and other controls related to the code unique particularities. The language specific controls for C (CBC) are divided into the following subcategories:

    - Pre-Processor (PRE)
    - Variable Management (VMG)
    - Memory Management (MEM)
    - File I/O Management (FIO)
    - Environment (ENV)
    - Signal and Error Handling (SEH)
    - Concurrency (CON)
    - Miscellaneous (MSC)

3.3. Assessment

This phase covers the analysis and evaluation of the findings identified in the previous phase, with the objective of validating and assessing their real risk, considering their *Threat*, *Vulnerability* and *Impact* risk scores. Once these scores have been calculated, a prioritisation process is carried out to identify those findings that should be fixed in a timely manner. Finally, if the vulnerability is unknown and has not been reported before, the project owners might consider reporting it in a CVE, CWE, CVSS or similar system.

To further organise this phase, three main activities have been defined:

- **Technical Report Analysis**: review of the results from the previous phase, validating the findings and removing any incomplete, incorrect or false-positive results. As part of this task, the findings are classified based on their category.

- **Impact Analysis**: Once the findings have been properly validated and classified, the next step is to determine their *Threat*, *Vulnerability* and *Impact* risk scores:
    - **Threat factors**: skill required opportunity and dimension.
    - **Vulnerability factors**: ease of discovery, ease of exploitation and awareness.
    - **Impact factors**: confidentiality, integrity and availability.

From the average result of these factors considered for the score, one of the following scores is given to the Threat, Vulnerability and Impact risks, based on the numeric result:

0 to 3: Low **3 to 6: Medium** **6 to 9: High**

Finally, the checklist is completed by adding the global risk posed by the controls, which is calculated from the individual results (Threat, Vulnerability and Impact). Table 1 shows how to calculate the global risk taking into consideration the Impact and the Probability (Average value of both Threat and Vulnerability results).

Table 1: Global risk evaluation

Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info	Low	Medium
		Low	**Medium**	**High**

Probability (Avg. Threat & Vulnerability)

The possible values are ***Critical***, ***High***, ***Medium***, ***Low*** or ***Info.*** If a control fails, it is marked with an **X** (it is a finding); if it passes, it is marked with a ✓**;** and if the control is not applicable, it is marked with N/A.

- **Finding Prioritisation:** The prioritisation of the findings is based on their criticality, and the results are communicated as established in the initial phases of the project.

4 CODE REVIEW DETAILS

4.1. Initial Considerations

The application to review contained several particularities that needed to be identified in order to ensure the proper analysis of the code. This included characteristics such as frameworks or libraries implemented, and the different aspects of the modules in use.

The main focus of this code review is on the Apache Core Module. However, during the initial preparations it was found that it depends heavily on the Apache Portable Runtime (APR) library. Therefore, APR was considered of high relevance to include it in the analysis.

The code to review was divided in the following 'modules': (1) Apache Core and (2) APR. Each of them was in turn divided into smaller code sets (defined as 'batches'). This helped to carry out the code review in parallel using the code review procedure explained in the WP2 - *Deliverable 11: Design of the methods for performing the code reviews.*

The distribution of software files and batches can be found in the following excel file:

Batches_and_files.xlsx

Apache Core (modules/core)

Batch	Files	Lines
Mod_macro	1	956
Mod_so	2	479
Mod_watchog	2	934

Apache Portable Runtime (APR)

Batch	Files	Lines	Batch	Files	Lines
APR_misc (omitted)	4	493	**APR_file_io_unix**	16	3.525
APR_dso (omitted)	5	1.190	**APR_file_io_win32**	11	4.822
APR_atomic (omitted)	2	212	**APR_threadproc_unix**	5	1.765
APR_network_io (omitted)	11	1.078	**APR_threadproc_win32**	4	1.599

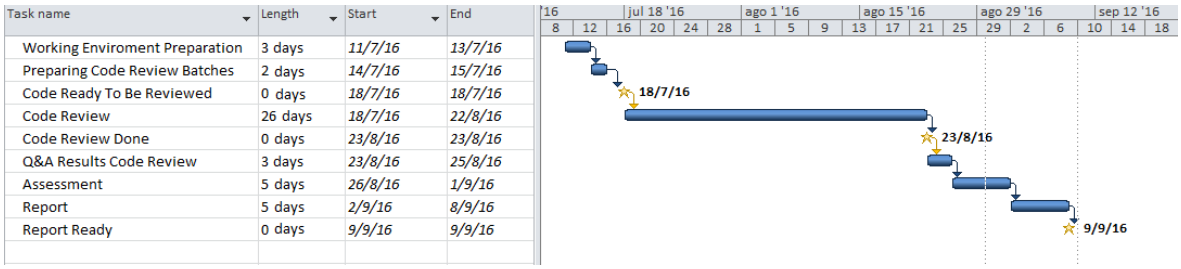
Batch	Files	Lines	Batch	Files	Lines
APR_locks (omitted)	12	1.701	**APR_network_io_unix**	8	4.044
APR_time (omitted)	0	0	**APR_network_io_win32**	3	1.297
APR_tools	1	115	**APR_dso_unix_win32**	2	418
APR_memory	1	2.655	**APR_misc_unix**	8	1.518
APR_shmem (omitted)	2	344	**APR_misc_win32**	8	1.256
APR_file_io (omitted)	25	2.860	**APR_atomic_unix_win32**	7	967
APR_threadproc (omitted)	14	3.019	**APR_time_unix**	2	502
APR_build	2	2.806	**APR_time_win32**	2	553
APR_user (omitted)	2	111	**APR_locks_unix**	5	1.643
APR_passwd	1	256	**APR_locks_win32**	4	697
APR_tables	3	2.569	**APR_shmem_unix_win32**	2	1.140
APR_include (omitted)	30	1.638	**APR_user_unix**	2	233
APR_mmap	3	376	**APR_user_win32**	2	380
APR_support	1	123	**APR_poll_unix**	8	3.777
APR_encoding	1	1.183	**APR_include_common**	38	11.927
APR_poll (omitted)	2	335	**APR_include_unix**	15	1.219
APR_random	4	962	**APR_include_win32**	14	1.715
APR_strings	6	2.875			

In order to better focus the efforts on the code review, only code related to the most used operating systems (‘Win32’ and ‘UNIX’) has been analysed, leaving other code parts related to ‘NetWare’ and ‘OS/2’ out of the scope in this particular case. Also, any code use for testing or debugging purposes has also been omitted.

4.2. Planning

The code review performed followed the planning defined at the beginning of the project, which takes into consideration the tests selected and the size/complexity of the application to review. The final planning is detailed in Figure 6, including the dates and time required for each step.

Figure 6: Code review planning



Task name	Length	Start	End
Working Enviroment Preparation	3 days	11/7/16	13/7/16
Preparing Code Review Batches	2 days	14/7/16	15/7/16
Code Ready To Be Reviewed	0 days	18/7/16	18/7/16
Code Review	26 days	18/7/16	22/8/16
Code Review Done	0 days	23/8/16	23/8/16
Q&A Results Code Review	3 days	23/8/16	25/8/16
Assessment	5 days	26/8/16	1/9/16
Report	5 days	2/9/16	8/9/16
Report Ready	0 days	9/9/16	9/9/16

4.3. Overview of Results

The controls used in the code review were generated as part of the EU-FOSSA project, and taking as a main source the information provided by two authorities regarding software security. These are the following:

- Application Security Verification Standard from OWASP.
- Secure coding standards from the Carnegie Mellon Software Engineering Institute (SEI).

As the full control set designed as part of the EU-FOSSA project covers a wide range of aspects and functionalities, it is important to be aware that not all of them will apply to every code review.

Therefore, the controls to include as part of this code review will depend on the features and characteristics of the code to review (as an example, authentication controls do not apply to a code that does not contain such functionality).

Each control has a unique identifier, following this template:

[CAT]-[SUB]-[###]

Legend:

[CAT] → Control category.
[SUB] → Control subcategory.
[###] → Control number.

4.3.1. General Findings

Table 2 shows a summary of all the general controls available on the code review methodology. It includes a result column indicating which of these controls are applicable, if they were successful or if they failed, including their severity.

Important: the results indicate the controls affected, not the number of findings in each one of them. Therefore, if a control has multiple findings in them, it will appear only once in this table. Further details of these controls, including the individual findings in each one, can be found in Section 4.4.

Table 2: Checklist general controls

ID	Control	Result	
DIM-FIM-001	Deletion of temporary files	✓	
DIM-FIM-002	File permissions at creation	✓	
DIM-FIM-003	Ensure that all files are closed after use	✓	
DIM-FIM-004	Usage of canonical path of files	✓	
DIM-FIM-005	Always check EOF on streams I/O operations	✓	
DIM-FIM-006	Updated file management	✓	
DIM-DSM-001	All data streams have to be closed after use	✓	
DIM-DSM-002	Get all valid data contained in a data stream	✓	
DIM-CEM-001	Correct format exchange of binary to string data	N/A	
DIM-CEM-002	Normalise all string inputs	N/A	
DIM-IVS-001	Data input validation	✓	
DIM-IVS-002	Data output validation	✓	
DIM-XSV-001	Review the XML schema, or DTD, used and its structure	N/A	
DIM-XSV-002	Data is sanitised before constructing and sending it in XML format	N/A	
AUT-AUV-001	The application uses a robust authentication verification process	✓	
AUT-PPU-001	The application makes use of a complex password policy	N/A	
AUT-PPU-002	Password history is maintained	N/A	
AUT-PPU-003	Passwords must expire after a set amount of time	N/A	
AUT-CST-001	Protection of the password at rest	N/A	
AUT-UAP-001	Number of login attempts is limited	N/A	
AUT-UAP-002	Connections from uncommon locations are restricted	✓	
AUT-PRP-001	A password recovery process is defined	N/A	
AUT-PRP-002	Password recovery process requires additional validation steps	N/A	
AUT-PRP-003	User is warned of any password recovery attempts	N/A	
SMG-SCP-001	Review controls in place to assign user privileges	✓	
SMG-SCP-002	Server keeps a list of all active identifiers and their data	✓	
SMG-SCP-003	Session cookies are protected and do not have sensitive data	N/A	
SMG-SID-001	A unique ID is assigned to each individual user session	N/A	
SMG-SID-002	Control active sessions at any time	N/A	
SMG-SLC-001	Session timeouts are implemented	N/A	
SMG-SLC-002	Privilege management	✓	

ID	Control	Result	
SMG-LGP-001	ID, assignations, privileges and resources are discarded on logout	✓	
SMG-LGP-002	Logout functionality should terminate the session and connection	✓	
ATZ-ACS-001	Use only trusted system objects for access authorisation decisions	N/A	
ATZ-ACS-002	Authorisation rules and process	N/A	
ATZ-PRV-001	Privileges and roles	✓	
ATZ-PRV-002	Privilege modification	N/A	
CPT-CPR-001	Sensitive information is stored securely using encryption	✓	
CPT-CPR-002	Information stored is hashed to preserve its integrity	✓	
CPT-CRC-001	Review cryptographic configuration parameters	✓	
CPT-CRC-002	Management cryptographic keys	N/A	
EHI-INL-001	Metadata leakage on any files accessible by the users	✓	
EHI-INL-002	Comments accessible in any client-side code files	N/A	
EHI-INL-003	Internal routes and paths must not be shown as default routes	N/A	
EHI-SFL-001	Sample files must be removed or filtered by the server	N/A	
EHI-EHD-001	Application errors must be controlled in the GUI	✓	
EHI-EHD-002	Try-catch-finally block	N/A	
EHI-EHD-003	Correct Exception and Error Management	✓	
EHI-EHD-004	Object is restored to a previous state after an error or failure	N/A	
EHI-EHD-005	Third-party services and libraries errors are controlled locally	✓	
COM-HSM-001	Avoid HTTP Response Splitting	N/A	
COM-HSM-002	Prevent Directory Traversal	✓	
COM-HSM-003	HTTP Strict Transport Security	N/A	
COM-HSM-004	Avoidance of redirects and forwards in webpages	N/A	
LOG-CFG-001	Logs are properly configured	✓	
LOG-CFG-002	Logs register only the information needed for their purpose	✓	
LOG-CFG-003	Debug Logging	✓	
LOG-CFG-004	Logging exceptions	N/A	
LOG-GEN-001	Log generation must continue after a log system exception	N/A	
LOG-LSI-001	Logs must not contain sensitive information, or else use hashes	✓	
LOG-LSI-002	User passwords and tokens must be omitted from logs	✓	
SCD-FWK-001	All frameworks and third party components are up-to-date	X	Info
SCD-VTY-001	Review operation on numeric values and bit collections	✓	
SCD-VTY-002	On division operations, check that the divisor does not equal zero	✓	
SCD-VTY-003	Direct comparisons with NaN must not be carried out	✓	
SCD-VTY-004	Do not use floating-point variables as loop counters	✓	
SCD-EXM-001	Function return values are parsed and evaluated	✓	
SCD-EXM-002	Method arguments must fall within the established bounds	✓	
OPT-CCR-001	Ensure that instance locks are controlled	✓	
OPT-CCR-002	Do not use unsafe operations, expressions or methods in Threads	✓	
OPT-CCR-003	Thread pools must be controlled	✓	
OPT-DOS-001	Check DoS vulnerabilities on the application	N/A	

ID	Control	Result	
OPT-MRM-001	Review the memory management process	✓	
OPT-MRM-002	Review resource management process	N/A	
OPT-COS-001	Evaluate processes that call back to the code multiple times	N/A	
OPT-COS-002	There is a clear separation between the application layers	N/A	
OPT-RPM-001	Analyse role-privilege matrix used on the application	N/A	

4.3.2. Language-Specific Findings

Table 3 contains a summary of all the language-specific controls available on the methodology for the **C programming language** (controls are also available for JAVA, PHP and C++ but have been omitted as none of them apply in this case). As in the previous table, only control results are listed, and not each individual finding within them. Detailed results are, as well, available in Section 4.4.

Table 3: Check-list language-specific controls

ID	Control	Result	
CBC-PRE-001	Do not create a universal character name through concatenation	✓	
CBC-PRE-002	Avoid side effects in arguments to unsafe macros	✓	
CBC-PRE-003	Do not use pre-processor directives in invocations of function like macros	✓	
CBC-VMG-001	Declare objects with appropriate storage durations	✓	
CBC-VMG-002	Declare identifiers before using them	✓	
CBC-VMG-003	Do not declare and identifier with conflicting linkage	✓	
CBC-VMG-004	Do not declare or define a reserved identifier	X	Info
CBC-VMG-005	Use the correct syntax when declaring a flexible array member	✓	
CBC-VMG-006	Do not create incompatible declarations of the same function or object	✓	
CBC-VMG-007	Do not declare variables inside a switch statement before the first case label	✓	
CBC-VMG-008	Ensure that floating-point conversions are within range of new type	✓	
CBC-VMG-009	Preserve precision when converting integral values to floating-point type	N/A	
CBC-VMG-010	Do not use object representations to compare floating-point values	✓	
CBC-VMG-011	Do not form or use out-of-bounds pointers or array subscripts	X	Info
CBC-VMG-012	Ensure size arguments for variable length arrays are in a valid range	✓	
CBC-VMG-013	Do not subtract or compare two pointers that do not refer to the same array	✓	
CBC-VMG-014	Do not add or subtract an integer to a pointer to a non-array object	✓	
CBC-VMG-015	Guarantee that library functions do not form invalid pointers	✓	
CBC-VMG-016	Do not add or subtract a scaled integer to a pointer	✓	
CBC-VMG-017	Do not attempt to modify string literals	✓	
CBC-VMG-018	Guarantee that string storage has sufficient space for character data and the null terminator	✓	
CBC-VMG-019	Do not pass a non-null-terminated character sequence to a library function that expects a string	✓	
CBC-VMG-020	Cast characters to unsigned char before converting to larger	N/A	

ID	Control	Result	
CBC-VMG-021	Do not confuse narrow and wide character strings and functions	✓	
CBC-VMG-022	Do not read uninitialized memory	N/A	
CBC-VMG-023	Do not dereference null pointers	✓	
CBC-VMG-024	Do not dereference null pointers	✓	
CBC-VMG-025	Variables must not be accessed using an incompatible type pointer	✓	
CBC-VMG-026	Prevent undefined behaviour when restrict-qualified pointers are used	N/A	
CBC-VMG-027	Do not apply operands within the sizeof, _Alignof or _Generic functions	✓	
CBC-VMG-028	Ensure that unsigned and signed integer operations are managed correctly	✓	
CBC-MEM-001	Do not access freed memory	X	Low
CBC-MEM-002	Free dynamically allocated memory when no longer needed	✓	
CBC-MEM-003	Allocate and copy structures containing a flexible array member dynamically	✓	
CBC-MEM-004	Only memory allocated dynamically should be freed	✓	
CBC-MEM-005	Allocate sufficient memory for an object	X	Info
CBC-MEM-006	Do not modify the alignment of objects by calling realloc()	N/A	
CBC-FIO-001	Exclude user input from format strings	X	Low
CBC-FIO-002	Do not perform operations on devices that are only appropriate for files	N/A	
CBC-FIO-003	Do not assume that fgets() or fgetws() returns a nonempty string when successful	N/A	
CBC-FIO-004	Do not copy a FILE object	N/A	
CBC-FIO-005	Do not alternately input and output from a stream without an intervening flush or positioning call	N/A	
CBC-FIO-006	Reset strings or fgets() or fgetws() failure	N/A	
CBC-FIO-007	Do not call getc(), putc(), getwc(), or putwc() with a stream argument that has side effects	N/A	
CBC-FIO-008	Only use values for fsetpos() that are returned from fgetpos()	N/A	
CBC-FIO-009	Avoid TOCTOU race conditions while accessing files	✓	
CBC-FIO-010	Do not access a closed file	✓	
CBC-ENV-001	Do not modify the object referenced by the return value of certain functions	✓	
CBC-ENV-002	Do not rely on an environment pointer following an operation that may invalidate it	✓	
CBC-ENV-003	All exit handlers must return normally	✓	
CBC-ENV-004	Do not call system()	✓	
CBC-ENV-005	Do not store pointers returned by certain functions	✓	
CBC-ENV-006	Ensure proper usage of the readlink() function	N/A	
CBC-ENV-007	Do not call putenv() with a pointer to an automatic variable as the argument	✓	
CBC-ENV-008	Proper privilege revocation and relinquish process must be defined	✓	
CBC-SEH-001	Call only asynchronous-safe functions within signal handlers	N/A	
CBC-SEH-002	Do not access shared objects in signal handlers	✓	
CBC-SEH-003	Do not call signal() from within interruptible signal handlers	✓	
CBC-SEH-004	Do not return from a computational exception signal handler	N/A	
CBC-SEH-005	Set errno to zero before calling a library function known to set errno, and check errno only after the functions returns a value indicating failure	✓	

ID	Control	Result	
CBC-SEH-006	Do not rely on indeterminate values of errno	N/A	
CBC-SEH-007	Detect and handle standard library errors	X	Info
CBC-SEH-008	Detect errors when converting a string to a number	✓	
CBC-CON-001	Clean up thread-specific storage	✓	
CBC-CON-002	Do not destroy a mutex while it is locked	✓	
CBC-CON-003	Prevent data races when accessing bit-fields from multiple threads	✓	
CBC-CON-004	Avoid race conditions when using library functions and files	✓	
CBC-CON-005	Declare objects shared between threads with appropriate storage durations	N/A	
CBC-CON-006	Avoid deadlock by locking in a predefined order	✓	
CBC-CON-007	Wrap functions that can spuriously wake up in a loop	N/A	
CBC-CON-008	Do not call signal() in a multithreaded program	✓	
CBC-CON-009	Do not join or detach a thread that was previously joined or detached	N/A	
CBC-CON-010	Do not refer to an atomic variable twice in an expressions	✓	
CBC-CON-011	Wrap functions that can fail within a loop	N/A	
CBC-CON-012	Do not use the vfork() function	✓	
CBC-CON-013	Do not use signals to terminate threads	N/A	
CBC-MSC-001	Do not use the rand() function for generating pseudorandom numbers	✓	
CBC-MSC-002	Properly seed pseudorandom number generators	✓	
CBC-MSC-003	Do not pass invalid data to asctime() function	N/A	
CBC-MSC-004	Ensure that control never reaches the end of a non-void function	✓	
CBC-MSC-005	Do not treat a predefined identifier as an object if it might only be implemented as a macro	N/A	
CBC-MSC-006	Do not call va_arg() on a va_list that has an indeterminate value	N/A	
CBC-MSC-007	Do not violate constraints	N/A	

4.4. Detailed Results

This section contains a detailed view of each control reviewed that has at least one finding in them. It provides a detailed explanation of the control, the checks that were carried out and the results and evidences backing up the finding. Furthermore, these controls have been assessed considering three scores: *Threat*, *Vulnerability* and *Impact*.

The term 'Vulnerability' is used in the context of this study as a 'Weakness', it is not necessarily a security vulnerability that can be exploited. Threat, Vulnerability and Impact indicators are used to assess the global risk, as explained on sub-section **3.3 Assessment**.

There were findings in 7 controls. These controls are:

- **Secure Code Design**
  - *Framework Requirement*: SCD-FWK-001 (**info**)
- **Specific C Controls**
  - *Variable Management*: CBC-VMG-004 (**info**), CBC-VMG-011 (**info**)
  - *Memory Management*: CBC-MEM-001 (**low**), CBC-MEM-005 (**info**)
  - *File I/O Management*: CBC-FIO-001 (**low**)
  - *Signal and Error Handling*: CBC-SHE-007 (**info**)

These controls, and their findings, are described in detail in the following sub-sections.

4.4.1. Specific C Controls

4.4.1.1. Variable Management

Table 4: CBC-VMG-004 findings

CBC-VMG-004	Do not declare or define a reserved identifier		**Info**
Description	All identifiers have to avoid variable declarations with the same value as reserved values, such as with an uppercase letter.	**Threat**	Low
		Vulnerability	Low
		Impact	Low
Checks	**1**	Check the name of file scope objects declared in the class does not begin with underscore.	✓
	2	Avoid the use of underscore in declaration of header guards.	✓
	3	Avoid declarations which contains name that begin with 'INT' or ends '_MAX', in order to prevent conflicts with reserved macros.	X
	4	Check that any C standard functions which are included in header definition such as 'malloc', 'realloc', 'aligned_alloc', 'calloc', are not redefined in the class.	✓
	5	If the program declares an identifier 'errno', it is mandatory to change it. This identifier has to be in the header <errno.h>. Include this sentence #include <errno.h> in the code.	✓
Results	The usage of the _MAX suffix in names of variables can lead to a conflict with reserved macros. The usage of this suffix varies from one OS to another, so additional controls to ensure proper transition of the fix for this finding may be needed.		
Evidence	**%APR%\shmem\unix\shm.c** (line 32) NAME_MAX		
Recommendation / Specific Solution	Ensure that there are no common variables defined making uses of the _MAX suffix, and replace any uses identified. If needed, add controls to ensure that the change does not impact in the code functions that make use of that variable/s.		

4.4.1.2. Memory Management

The findings identified within this control are not considered vulnerabilities, as they affect legacy systems not officially supported by Microsoft nor the Apache HTTP project.

Table 5: CBC-MEM-001 findings

CBC-MEM-001	Do not access freed memory		**Low**
Description	Using a pointer that directs to memory that has been already freed (and therefore can be used by another process), can lead to unexpected behaviour an instability.	**Threat**	Low
		Vulnerability	Medium
		Impact	Low
Checks	**1**	Verify that pointers are destroyed when memory is freed.	✓
	2	Ensure that memory has been freed before writing data on it.	X
	3	Verify that the memory-freeing process is done only once.	✓
Results	**Legacy Finding**: the following finding is mentioned to create awareness among users that keep running Apache servers on older OS (Windows XP, Windows Server 2003…), but it does not have to be fixed as those systems are not supported (**by neither Microsoft nor httpd**). It does not impact on the Control assessment results. • ***InitializeCriticalSection****: Exceptions can be thrown in low-memory situations. Use InitializeCriticalSectionAndSpinCount instead.*		
Evidence	**LEGACY FINDINGS *(for reference only)*** *%APR%\threadproc\win32\proc.c* (line 430) InitializeCriticalSection(&proc_lock); *%APR%\misc\win32\misc.c* (line 223) (InitializeCriticalSection)(&cs); *%APR%\locks\win32\thread_cond.c* (line 52) InitializeCriticalSection(&cv->csection); *%APR%\locks\win32\thread_mutex.c* (line 64) InitializeCriticalSection(&(*mutex)->section);		
Recommendation / Specific Solution	These findings only affect implementations of the Apache Server in older operating systems. However, these operating systems are no longer supported by Apache or Microsoft. Furthermore, adding fixes to these legacy findings would introduce complexity to the code and, as it is no longer supported, it is discouraged. **Specific Solution:** Although it is discouraged to use Apache in older operating systems, and taking into consideration that this should not be fixed by the Apache Foundation, the following information is provided for any older user of legacy OS: ○ Replace **`InitializeCriticalSection`** with **`InitializeCriticalSectionAndSpinCount.`**		

4.4.1.3. File I/O Management

Table 6: CBC-FIO-001 findings

CBC-FIO-001	Exclude user input from format strings		**Low**
Description	Never call a formatted I/O function with a format string containing a non-sanitised value.	**Threat**	Low
		Vulnerability	Medium
		Impact	Low
Checks	**1**	Ensure that user input values are not used in a formatted I/O functions, such as 'snprintf()'+'fprintf()' or 'snprintf()'+'syslog()'. Non-sanitised input will not be concatenated in a string that will be used in file operations.	**X**
Results	sprintf: Does not check for buffer overflows (CWE-120). Use sprintf_s, snprintf, or vsnprintf.		
Evidence	**%APR%\misc\win32\misc.c** (line 228) (sprintf)(sbuf, "%p %08x %08x %s() %s:%d\n", ha, (unsigned int)seq, (unsigned int)GetCurrentThreadId(), fn, fl, ln);		
Recommendation / Specific Solution	The use of weak vulnerable functions should be avoided whenever possible as to increase the robustness of the code and prevent related risks as well. **Specific Solution:** For the case of **`sprint`**, it should not be used but replaced with **`sprintf_s`**, **`snprintf`**, or **`vsnprintf`**.		

4.4.2. Build Tool (build folder)

These findings are related to the compilation support libraries that are part of the APR library but take no part on the final executable code generated. The purpose of this library is to assist compilation, therefore the findings are not directly related to the running APR, but to the compilation process. They are included here to serve as a reference for future upgrades and development on them.

Important: these findings do not have a direct impact on the security of the runtime code or on the execution of the server, as they are part of a separate block (build tool) used exclusively during compilation time.

Before deciding to change them, one must take into account the risk of adding more complexity to the code.

4.4.2.1. Variable Management

Table 7: CBC-VMG-011 findings

CBC-VMG-011	Do not form or use out-of-bounds pointers or array subscripts		**Info**
Description	The addition or subtraction of a pointer into/from, or just beyond, an array object and an integer type produces a result that does not point into, or just beyond the same array object.	**Threat**	Low
		Vulnerability	Low
		Impact	Low
Checks	**1**	Ensure that there are no negative index values.	X
	2	Use an unsigned type of index to avoid negative index values.	X
	3	Use L' \0' to terminate a loop with any array to prevent buffer overflow.	✓
	4	Initialising matrix elements in the same row-major as multidimensional objects (go through the matrix first by rows and then by columns).	N/A
	5	Avoid incrementing the pointers to a loop condition.	✓
	6	Validate the index by using relational operators to prevent overflows. It is mandatory to #define a SIZE_MAX and to make sure, in each iteration, that the loop does not overflow.	✓
Results	**Arrays:** Several times specific positions of arrays are accessed without checking if the position exists. Possible negative positions, or larger ones than the array limit can be accessed freely. For instance, in the first evidence, the argument 'position', in line 353, is declared as int, instead of unsigned int. The value of the 'position' variable is not checked, in other words, there is no control inside the function to ensure that the value received by the 'position' argument is a positive value, something worthy due to the int declaration. If there is a large negative value as the value received by the 'position'		

<table>
<tr><td></td><td>argument, many memory positions could be compromised.
The lack of control over an int variable which will be used as some kind of ‘pointer’ makes this a finding.</td></tr>
<tr><td>Evidence</td><td>%APR%\build\jlibtool.c (line 353)

void insert_count_chars(count_chars *cc, const char *newval, int position)
{
int i;

for (i = cc->num; i > position; i--) {
cc->vals[i] = cc->vals[i-1];
}

cc->vals[position] = newval;
cc->num++;
}

%APR%\build\jlibtool.c (line 341)

cc->vals[cc->num++] = newval;

%APR%\build\jlibtool.c (line 531)

var[equal_pos - arg] = 0;</td></tr>
<tr><td>Recommendation / Specific Solution</td><td>This finding does not have a direct impact on the security of the runtime code, as it is part of a separate block (build tool) used exclusively during compilation time.
Before deciding to change it, one must take into account the risk of adding more complexity to the code.

Recommendation:
Implement control functionality to check the value of the loop limit variable in order to ensure that it is a valid positive number and larger than zero.
Any access to arrays (especially within structures) should be done after checking the bounds of that array.</td></tr>
</table>

4.4.2.2. Memory Management

Table 8: CBC-MEM-005 findings

CBC-MEM-005	Allocate sufficient memory for an object		**Info**
Description	It is necessary to guarantee that storage for strings has sufficient space available for character data and consequently allocate sufficient memory for an object.	**Threat**	Low
		Vulnerability	Low
		Impact	Low
Checks	**1**	The length of string storage arrays must not equal zero.	✓
	2	Validate string operations to ensure that they are controlled and cannot result in an overflow.	X
	3	Arguments passed to functions must match the expected format and size.	✓
Results	**Strcpy, strcat**: Modules do not check for buffer overflow (CWE-120) and format strings manipulation. Consider using strcpy_s, strcat_s and a constant for the format specifications. For instance, in the second evidence there are no controls within the function to ensure that all the characters of 'name' can be allocated in 'newarg'. **Memory operations**: Several times memory operations, done using **memcpy**, are used without checking the size of source and destiny.		

Evidence	**%APR%\build\aplibtool.c** (line 157)
	strcpy(value, equal_pos + 1);
	%APR%\build\aplibtool.c (line 272)
	strcat(newarg, name);
	%APR%\build\jlibtool.c (line 850)
	memcpy(newarg, arg, arglen);		

Recommendation/ Specific Solution	This finding does not have a direct impact on the security of the runtime code, as it is part of a separate block (build tool) used exclusively during compilation time. Before deciding to change it, one must take into account the risk of adding more complexity to the code. **Specific Solution:** Put in place controls to ensure that the source can be allocated into the destination or: ○ Replace all instances of **`strcpy`** with **`strcpy_s`**. ○ Replace all instances of **`strcat`** with **`strcat_s`**. **Recommendation:** The use of memcpy should only be considered after checking the size of the destination memory position against the source, to avoid an overflow.		

4.4.2.3. Signal and Error Handling

Table 9: CBC-SEH-007 findings

CBC-SEH-007	Detect and handle standard library errors		**Info**
Description	The majority of the standard library functions return a valid value or a value of the correct return type that indicates an error, but it is necessary that the programmer detect and appropriately handle all errors in accordance with error-handling policy.	**Threat**	Low
		Vulnerability	Low
		Impact	Low
Checks	**1**	Verify and check the Standard C Library Errors list.	**X**
Results	**malloc**: Lack of error checking functionality. If an error happens in 'cc-> vals', this variable will be NULL. There should be a NULL check after line 352. **remove**: Lack of error checking functionality, if an error happens in that line. There should be a 0 check after line 606 to ensure the success of the operation. **fgets**: Lack of error checking functionality, if an error happens in that line. There should be a NULL check after line 969 to ensure that the path has received a value.		

Evidence	**%APR%\build\jlibtool.c** (line 325)
	`void init_count_chars(count_chars *cc)`
	`{`
	`  cc->vals = (const char**)malloc(PATH_MAX*sizeof(char*));`
	`  cc->num = 0;`
	`}`
	%APR%\build\aplibtool.c (line 606)
	`remove(fullname);`
	%APR%\build\jlibtool.c (line 969)
	`fgets(path, PATH_MAX, f);`		

Recommendation/ Specific Solution	This finding does not have a direct impact on the security of the runtime code, as it is part of a separate block (build tool) used exclusively during compilation time. Before deciding to change it, one must take into account the risk of adding more complexity to the code. **Recommendation:** ○ A 'NULL' check should be used after the buffer creation to detect possible errors and handle it properly. ○ A '0' check should be done after using the **remove** function in order to detect possible errors. ○ A 'NULL' check should be used after using **fgets** to detect possible errors and handle it properly.		

4.4.3. Findings controlled programmatically

During the code review, there were several findings that were identified that are a consequence of the use of weak or deprecated functions. After a detailed review, it was determined that these findings are controlled within the code in the current iteration of Apache. For this reason, the findings were moved to a separate section, as the risk of using these functions has been mitigated.

Before deciding to change them, one must take into account the risk of adding more complexity to the code, and ensure that the mitigation of the risk that is provided via the code is maintained.

4.4.3.1. Framework Requirements

Table 10: SCD-FWK-001 findings

SCD-FWK-001	All frameworks and third party components are up-to-date		**Info**
Description	All frameworks and components used are kept up-to-date including all existing patches and security hotfixes. The latest version is not needed but must be at least patched.	**Threat**	Low
		Vulnerability	Medium
		Impact	Low
Checks	**1**	Framework components are kept up-to-date.	**X**
	2	Third-party components are kept up-to-date.	**N/A**
Results	**_alloca**: In the finding detected in the code, the use of this function in the version evaluated is controlled by ensuring that the parameter is not large enough to cause instability in its use. Taking into account that this function is considered deprecated according to MSDN (for Windows systems) due to the free-up memory controls it provides, it is recommended to consider updating it to use the **_malloca** function alternative. **getpass**: In the finding detected in the code, the use of this function in the version evaluated is controlled by ensuring that the function will not be used under Operating Systems in which this function could represent a security flaw. Nevertheless this function is obsolete and not portable. This finding is highlighted in order to keep it in mind for future developments. It is something that adds risk to the code and should be mitigated whenever possible. It is a bad practise to have deprecated or legacy code, as it leads to instability and weaker security, even if it is controlled in its current version. Later versions may override this and raise the finding again. Before deciding to change it, one must take into account the risk of adding more complexity to the code.		

Evidence	**%APR%\network_io\win32\sendrecv.c** (line 118) pWsaBuf = (nvec <= WSABUF_ON_STACK) ? _alloca(sizeof(WSABUF) * (nvec)) **%APR%\password\apr_getpass.c** (line 242) char *pw_got = getpass(prompt);
Recommendation/ Specific Solution	Despite that this finding is controlled within the code it is included under this section to keep them in mind for future development. Before deciding to change it, one must take into account the risk of adding more complexity to the code. **Recommendation:** The **getpass** function is obsolete due to its high insecurity. It should never be used; instead, the functionality should be defined manually in the code to ensure the proper processing of the information according to the needs of the application. **Specific Solution:** The **_alloca** function allocates memory on the stack in a Windows system. This function is deprecated because a more secure version is available. The recommendation is to use the new version: **_malloca**

5 TECHNICAL CONCLUSIONS

The most relevant output to consider, and the first that stands out, is the nature of the findings.

Most of them are language-specific, instead of common, general ones. This is mostly due to the nature of the software section reviewed, as it is the core and it does not contain as many business logic and functionalities as other parts. Furthermore, it is important to highlight that this software is **actively maintained and upgraded**, with periodic bug-fixes and patches provided not only for the latest versions but also for legacy ones.

The focus of the code review was on the core backend part of Apache, specifically on modules/core and APR. This is a critical section from a security point of view, even if it is not as 'visible' as other front-end modules that are usually reviewed during pentesting and vulnerability assessments.

Another interesting aspect to highlight is the programming language used in this code. It is written in C, which is a complex language from a security point of view. It provides a very high level of flexibility and customisation, especially when compared with other modern languages used for software development.

This leads to one of the areas to focus on: its memory management features or, to be more precise, the lack of exception management in those features. The main focus of a code review in C must always include an in-depth review of the use of memory, as C allows direct access to it without providing exception support.

In conclusion, the code review carried out confirmed the fact that the code of both Apache Core and APR **have a good level from a security point of view**, with only a few controls with findings, none of them being of high severity.

As a final note, it is fundamental to take into account that these findings cannot be directly considered security flaws that can be exploited, since 'Security' is a set of layers and therefore several risky findings are necessary to compromise the software.

