Tuijin Jishu/Journal of Propulsion Technology
ISSN: 1001-4055
Vol. 44 No. 4 (2023)

Source Code Analysis on Asterisk VOIP IP PBX: An Open-Source Communication Toolkit

Pradeep Mullanagi[1], Kannan Shanmugam[2*], Kalasamudram Maheswari Devi[3], Kannan Paramasivan[4], M. Deena Babu[5], Cyril Mathew O[6], J. Jaganpradeep[7], Rajeswaran Nagalingam[8]

[1]Department of ECE, Shri Vishnu Engineering College for Women, Bhimavaram, Andhrapradesh, India
[2*]School of Computing Science and Engineering, VIT Bhopal University, Bhopal-Indore Highway, Kothrikalan, Sehore, Madhya Pradesh, India.
[3]Department of ECE, Malla Reddy Engineering College, Secunderabad, Telangana State, India
[4]Department of Electronics and Communication Engineering, Francis Xavier Engineering College, Tirunelveli. Tamil Nadu, India.
[5] Department of IT, Malla Reddy Engineering College, Secunderabad, Telangana State, India
[6]Department of Electronics and Communication Engineering. Al Ameen Engineering College, (Autonomous), Erode, Tamilnadu, India.
[7]Department of ECE, SSM College of Engineering,Komarapalayam, Tamilnadu, India
[8]Department of Electrical and Electronics Engineering, Malla Reddy College of Engineering, Secunderabad, Telangana State, India.

[*]Corresponding Author: Kannan Shanmugam[2*]

[2*]School of Computing Science and Engineering, VIT Bhopal University, Bhopal-Indore Highway, Kothrikalan, Sehore, Madhya Pradesh, India.

Abstract: Analysis of source code has risen to the top of the list of priorities as a result of the persistent danger posed by security breaches to the communication infrastructure that is now interconnected throughout the globe. Studying the source code may help find vulnerabilities in the security coding that might be exploited by hackers to get access to vital information. These vulnerabilities can be found before the hackers have a chance to use them.Either a static analysis or a dynamic analysis may be performed on source code.The difference between static and dynamic analysis is that the latter executes the code in a runtime environment to check for security problems, while the former does not.The purpose of this work is to use both free and commercial tools to analyze the source code of the 'c' language-based Asterisk VOICE IP-PBX for security vulnerabilities.Security coding standards like MISRA C, CERT C, and others were applied to the Asterisk 1.8 VOIP IP PBX (Open source) source code. This report details the vulnerabilities discovered in the codebase using both free and paid security analysis tools.

Keywords: VoIP, Asterisk, Vulnerability, Source Code Analysis, Flaw Finder Andsplint.

1. SECURE CODING STANDARDS

The examination is performed by means of source code analysis tools, which compare the code to predefined safe coding standards. To guarantee that the code is safe against Security flaws, these standards provide a standard set of guidelines and tenets that must be followed [1,2].

These are the most popular safe coding practises:

1.1 MISRA -MOTOR INDUSTRY SOFTWARE RELIABILITY ASSOCIATION

MISRALeading developers across industries now use MISRA as a paradigm for best practises. This includes the aerospace, telecom, medical device, military, railway, and other industries. MISRA is a group that has created software coding standards for languages like C and C++.The goal of MISRA is to make it easier for developers of embedded systems written in ISO C to create code that is secure, portable, and reliable. The MISRA C language standard is closed. Users or those who implement must pay for the requisite guidelines. The MISRA C++ guidelines are similar in this respect [2].

1.2 CERT-COMPUTER EMERGENCY RESPONSE TEAM

Through a collaborative effort involving developers and security experts, the CERT publishes best practises and standards for popular computer coding languages such as C, C++, Java, and Perl. Guidelines for preventing both implementation and low-level design problems are provided in the CERT Security coding standards. The purpose of these guidelines is to construct secure, trustworthy, and safe systems.

1.3 CWE-Common Weakness Enumeration

MITRE, with funding from the United States Department of Homeland Security (DHS), compiled a formal list of software vulnerabilities called the Common Weakness Enumeration (CWE). The Common Weakness Enumeration (CWE) is a shared vocabulary for characterizing vulnerabilities in the architecture, design, or coding of software. It provides a standardized reference point from which software security tools may identify vulnerabilities.

2. TOOLS FOR STATIC ANALYSIS

The original code of ASTERISK 1.8 VOICE IP PBX Solution was analyzed with the help of the five open-source static analysis tools listed below [3,4].

2.1 Open Source Tools

The different open-source tools are given in Table 1.

Table 1- Open-source tools

S. No	Free / Open Source Tools	Language	Standards
1.	Flaw Finder	C/C++	CERT
2.	Rough Auditing Tools for Security (RATS)	C/C++	CWE
3.	Splint	C/C++	ISO C99
4.	Cppcheck-1.40	C/C++	C89,C90
5.	Cqual	C/C++	CERT

2.1.1 Flaw Finder

The flaw detector was one of the first tools developed for lexical analysis. This instrument was created by David A. Wheeler's company. Python is the platform that is being used to create the tool. It examines the system for a variety of security flaws, including buffer overflow, difficulties with format strings, Meta character dangers, race situations, and inadequate random number generation, among others. The flaw finder starts by comparing the names of the variables in the source code to the values in the built-in database. It then generates a list of "hits" that may or may not constitute possible security flaws. The flaw finder scans through the C/C++ source code in search of probable vulnerabilities in the security [5].

2.1.2 RATS

The Rough Auditing Tool for Security, or RATS, is a free piece of software that Security Software offers for the purpose of doing static analysis. It offers the lexical scanning for the C programming language, as well as C++, C#, and Java. Additionally, it supports scripting languages such as Perl, PHP, and Python. It is able to identify buffer overflow issues, race circumstances between Time of Check and Time of Use, and other similar issues. It is compatible with ODBC and SQL databases [6].

2.1.3 SPLINT

A tool for doing static analysis is known as the splint tool. The National Science Foundation provided funding in order to support the development of splints. Splint performs a scan of the C source code to check for possible security problems such as type mismatch, leaks of memory, null dereference, usage of Un-initialized formal arguments, unauthorised use of globally accessible variables, overflows of buffers, and so on [7].

2.1.4 CPPCHECK

CPPcheck is a tool for analysing code written in C and C++. It may be used in a variety of contexts. Daniel Marjamaki is the one who came up with the idea and is also the primary developer. The GNU General Public Licence allows for the programme to be distributed without cost. Out of bounds checking, inspecting the source code for each class, verifying the safety of exceptions, checking for memory leaks, warning if outdated functions are used, checking for incorrect use of STL, checking for unused variables and functions, and warning if outdated functions are used are some of its capabilities. CPPcheck is designed to be as accurate as possible when reporting problems since it places a greater emphasis on reducing the number of false positives. However, it is unable to identify a great number of problems[8].

2.1.5 CQUAL

Lightweight and practical technique for describing and testing characteristics of C programmes, Cqual is a type-based analysis tool. Central Qualifications is shortened to CQUAL. A graphical user interface displays the analysis findings, allowing the developer to go through the inferred qualifiers and their respective flow pathways [9].

2.2 Non-Open Source Tool

Professor Michael Hennell of the University of Liverpool launched Liverpool Data Research Associates (LDRA) in 1975 to commercialise a software test-bed he developed to evaluate the quality of mathematical libraries used in his research on nuclear physics. Static analysis may be performed on code in C, C++, JAVA, ADA, and Assembly with the help of LDRA Test bed, an exclusive software analysis tool. Code, quality, and design evaluations are all part of the service.Its primary use is in safety-critical electronics and aviation, where software must be dependable, tough, and error-free [10,11].

3. EVALUATION OF ASTERISK SOURCE CODE

Asterisk is a free and open source C++ IP PBX system for voice and data communications. Because of Asterisk, any computer may act as a server for voice and video calls. IP PBXs, VoIP gateways, conferencing servers, and other bespoke solutions are all powered by Asterisk [12,13,14,15]. In this study, the complete asterisk code base has been analysed using the aforementioned five open source tools, and an expensive source code analysis tool, LDRA, has been used to analyse a single file of asterisk code (chan_sip.c).

4. RESULTS & EVALUATION

All of the aforementioned free static analysis tools were fed the full set of Asterisk's 27 modules of source code, and the results are shown below Table 2:

Table 2. List of modules in asterisk -1.8.20.2

Addons	Agi	Apps	Autoconf
Bridges	Build_tools	Cdr	Cel
Channels	Codecs	Configs	**Contrib**
Doc	Formats	Funcs	Images
Include	Keys	Main	Menuselect
Pbx	Phoneprov	Res	Sounds
Static-http	Tests	Utils	

4.1 Comparison of open-source tools against the Security flaws

The following Table 3, details the most common security flaws and whether or not they are detected by the various open-source tools. If a specific security flaw is found by a tool, it will be noted with a "YES" entry; otherwise, it will be marked with a "NO" entry [16,17,18].

Table 1- Comparison of Open-source tools against Flaws

S.NO	**SECURITY VULNERABILITY**	**OPEN SOURCE STATIC ANALYSIS TOOLS**				
		Flaw Finder	**Splint**	**CPPcheck**	**RATS**	**Cqual**
1.	Missing check against NULL	YES	YES	YES	NO	YES
2.	Unchecked Return Value	NO	NO	NO	YES	NO
3.	Dead Code	NO	YES	YES	NO	NO
4.	Double Free	NO	YES	NO	NO	NO
5.	Infinite Loop	NO	YES	NO	NO	NO
6.	NULL Dereference	YES	YES	YES	YES	YES
7.	Divide by Zero	YES	NO	YES	YES	YES
8.	Uninitialized Variable	YES	YES	YES	YES	YES
9.	Unreleased Resource	NO	NO	NO	NO	NO
10.	Incorrect Block Delimitation	NO	YES	NO	NO	NO
11.	Buffer Overflow(stack & Heap)	YES	YES	YES	YES	YES
12.	Buffer Under run	YES	YES	YES	YES	NO
13.	Command Injection	YES	YES	NO	YES	YES

14.	Format String	YES	NO	YES	YES	YES
15.	Illegal Pointer Value	NO	YES	NO	NO	YES
16.	Integer Overflow	YES	YES	NO	YES	YES
17.	Log Forging	YES	NO	YES	NO	NO
18.	String Termination Error	YES	NO	YES	YES	YES
19.	In Security Randomness	YES	NO	NO	YES	NO
20.	Time of check & Time of Use	YES	NO	NO	YES	YES
21.	Constant Inferences	NO	NO	NO	YES	YES
22.	Use After Free	YES	NO	NO	YES	NO
23.	Buffer over read & write	YES	YES	NO	YES	YES
24.	Conversion Error(sign to unsigned,& unsigned to sign)	YES	YES	NO	YES	YES
25.	Numeric Truncation Error	YES	YES	NO	NO	YES
26.	Files Manipulation Errors	YES	NO	NO	NO	NO
27.	Global variable analysis	YES	YES	NO	YES	NO
28.	Local Variable Analysis	YES	YES	NO	YES	YES
29.	Assertion Checking	NO	NO	YES	NO	NO
30.	Directive Vulnerabilities	NO	YES	NO	YES	YES
31.	Random Number Generation failure	YES	NO	YES	NO	YES
32.	Preprocessor Mismatch	NO	YES	NO	YES	YES
33.	various cut & paste issues, missing breaks	YES	YES	YES	YES	YES
34.	priority of operators issues	YES	NO	YES	YES	NO
35.	stray semicolon after if	YES	YES	NO	NO	YES
36.	Missing asterisks in pointer operations	YES	NO	NO	YES	YES

4.2 Detailed Comparison of All Tools

The following Table 4, details the types of Security vulnerabilities found by the selected source code analysis tool, as well as the faulty functions or variables that led to those vulnerabilities:

Table 2- Detail comparison of All Tools

Tools Name	API / Variable Name	Security Vulnerability Name	Class of Vulnerability
Flaw Finder[19]	fgets,	Buffer overflow(stack & heap)	Boundary Problems / Buffer Handling
	Strcpy	Buffer under run	
	Read	Buffer over read & write	
	Atoi	Conversion error(sign to unsigned & unsigned to sign)	
	Char name[32]	Integer Overflow	
	Getenv	Global , local & Environment variable analysis	Uninitialized / Unused Variables
	Char last	Uninitialized variable	
	If((ifileid=fopen(argv*2+,”rb”))==NULL	Null dereference	Memory Safety
	Usleep	Obsolete Error	
	Astresp =’\0’	Missing check against null	
	Asn1PD_H235 Random Val	Random number generation failure	Random number Issue
	Pvalue->random	InSecurity randomness	
	Chmod ,fopen	Time of check & time of use	Race Condition
	System(cmd)	Command injection	String Manipulation Problem
	attribute((format(printf))	Format string	
	Vsnprintf	String termination error	
	While	Various cut & paste issues, missing breaks	Human Issues
	Else if	Stray semicolon after if	
	Mkstemp	Temporary file vulnerability	File Handling
Splint	Snprintf	Missing check against null	Memory Safety
	OPENSSL_free	Use after free & Double free	
	If((id==Null))	Null dereference	

	If then else	Incorrect block delimitation	
	Char last	Uninitialized variable	Unused Variable
	Getenv	Global, local & environment variable analysis	
	Fgets,strcpy	Buffer overflow(stack & heap)	Boundary Problem
	Strncpy	Buffer under run	
	Read	Buffer over read & write	
	Unsigned to signed	Conversion error(sign to unsigned, & unsigned to sign)	
	Char set_name[255]	Integer overflow	
	Load Library	Directive vulnerabilities	Preprocessor Issues
	Include files	Preprocessor mismatch	
	While	Various cut & paste issues, missing breaks	Coded by Human Issues
	if then else	Stray semicolon after if	
RATS[20]	Strcpy, fgets	Buffer Overflow(stack & Heap)	Boundary Condition
	Strncpy	Buffer Under run	
	Read command	Buffer over read & write	
	Unsigned char buf[]	Conversion error(sign to unsigned, & unsigned to sign)	
	Char last	Uninitialized Variable	Unused Variables
	Char rfcomm_buff[256], getenv	Global, Local & Environment variable analysis	
	Snprintf(s,sizeof(s),”%d”,id)	Missing check against NULL	Memory Safety
	Memcpy	NULL Dereference	
	Vfork()	Dead Code	
	OPENSSL_free	Double Free & Use After Free	
	Realloc	Memory allocation Error	

	Res=random();	Random Number Generation failure	Random Number Issues
	Srand	InSecurity Randomness	
	Access	Access error	Authentication and Access Handling
	If(!stat), lstat	Time of check & Time of Use	Race condition
	Fd=open(dev_filename,O_RD WR))<0)	Race condition	
	Sprint, Err_error_string	Format String, Error String	String Manipulation Errors
	Getopt, syslog	Truncate input string error	
	Load Library	Directive Vulnerabilities &Preprocessor Mismatch	Preprocessor Errors
	Gethostbyname	DNS error	Miscellaneous
	Umask	Unsafe Error	
	Tmpfile	Temporary Files	
	Signal(SIGHUP,SIG_DFL)	Signal Handler error	
	While()	various cut & paste issues, missing breaks	Code by Human Errors
	if(!stat)	stray semicolon after if	
	System(cmd)	Command Injection	Command Injection
CPPcheck	Snprintf()	Missing check against NULL	Boundary problem
	Fgets, char data[1],strcat	Buffer Overflow(stack & Heap)	
	Strncpy	Buffer Under run	
	Unsigned int to int *	Conversion Error(sign to unsigned, & unsigned to sign)	
	Char cmd[32]	Integer Overflow	
	Read	Buffer over read & write	
	Return	Unchecked Return Value	Memory Safety
	Targetcall_pvt	NULL Pointer Dereference	

	Usleep	Obsolete function error	
	OPENSSL_free	Double free & Use After Free	
	Start variable	Uninitialized Variable	Unused Variables
	Getenv	Assertion variable & Environment variable analysis	
	Random	Random Number Generation failure	Random Number Generation
	Srand	InSecurity Randomness	
	Sscanf ,	Format String	String Manipulation Errors
	Snprintf	String Termination Error	
	Unlink(newfn)	Time of check & Time of Use	Race condition
Cqual[21]	Snprintf	Missing check against null	Memory Safety
	OPENSSL_free	Use after free & Double free	
	If((id==Null))	Null dereference	
	If then else	Incorrect block delimitation	
	Char last	Uninitialized variable	Unused Variable
	Getenv	Global, local & environment variable analysis	
	Fgets,strcpy	Buffer overflow(stack & heap)	Boundary Problem
	Strncpy	Buffer under run	
	Read	Buffer over read & write	
	Unsigned to signed	Conversion error(sign to unsigned, & unsigned to sign)	
	Char sresult[10];	Integer overflow	
	Load Library	Directive vulnerabilities	Preprocessor Issues
	Include files	Preprocessor mismatch	

4.3 Results of the commercial tool LDRA

The LDRA programme was fed the Asterisk source code file chan_sip.c, and it performed an analysis of the file in comparison to the MISRA and CERT security standards:

4.3.1 LDRA Test bed Code Detailed Review Report

System: ASterisk_1.8

While compliance with checking standards is generally expected, it is possible to override them in certain cases by annotating the code. Standards that are optionally enforced nonetheless give extra quality requirements. Additional details on some breaches of norms are included in results Tables 5.Take note that macro expansions are not shown in the source code. If a standard is marked as "Off" in the Summary tables, it is disabled in the pen.The "MR" dat file in the Summary tables denotes a MISRA Limited standard that requires a MISRA licence for verification[22].

Table 5: A summary is given of the pass/fail result of each program component

Components which pass all standards are marked:	pass
Components which fail only Optional standards are marked:	Conditional Pass
Components which fail on insufficient comments only are marked:	Comment FAIL
Components which violate Rule standards are marked:	FAIL

The violation density of a function is the fraction of reformed lines that include a violation. It's a measure of how well the code was written. Smaller functions with a high number of violations tend to have a high violation density. A lower density may represent less-violated, broader functions. Multiply (nViols / nRefLines) by 100 wherenViols is the total number of deviations from the norm of the function. additionally: nRefLines = total lines reorganised in a function. An indicator of the diversity of standards breaches in a module is provided by the unique requirements failure ratio. Report Configuration and Analysisis given in Table 6. A high ratio indicates that the function violates many criteria, whereas a low ratio indicates that it violates just one [23].

Table 6 - Report Configuration and Analysis

Report Production	Report Configuration	Analysis phases
1.C/C++ LDRA Testbed Version: 9.4.2 2.Config. File: /home/prashant/ldra_toolsuite/c/creport.dat 3.Produced On: Wed Jan 22 2014 at 17:26:00 4.Penalty File: /home/prashant/ldra_toolsuite/c/cpen.dat	1.Report Level: Summary Report 2.Procedures Reported: Fails Only 3.Programming Standards Model: CWE 4.Line Numbers refer to: Original Source File 5.Violation Details: Violations Only 6.Reporting Scope: Source file and associated header	1.Static: Yes 2.Complexity: No 3.Static Data Flow: No 4.Information Flow: No 5.Cross Reference: No

4.3.2 Overall Code Review Summary

Totals of Violations for Selected Code Review Standards

A minus sign ('-') means that data from the Analysis Phase is currently unavailable.

A value of "Off" in the Penalty File (lang>pen.dat) indicates that the criterion has been disabled.

The letter 'MR' denotes a Misra Restricted criterion. The details result of LDRA is tabulated in Table 7.

Table 7- Detail Result of LDRA

Number of Violations	LDRA Code	Rule Standards	CWE Code
9	5 S	Empty then clause.	CWE 480
0	20 S	Parameter not declared explicitly.	CWE 628
46	21 S	No expected match found for the parameter	CWE 234,235,685
0	23 S	No explicit function call on the analysed code	CWE 561
8	26 S	Infinite loop used.	CWE 835
0	27 S	Void procedure with return statement.	CWE 398
185	35 S	No explicit function call of static on analysed code	CWE 561
4	36 S	Function has no return statement.	CWE 758
0	39 S	Unsuitable type for loop variable.	CWE 739
3	41 S	Ellipsis utilized in function parameter list.	CWE 628
152	44 S	Use of restrictedmethods or variable.	CWE 78,187,242,338,365,475,676
2	47 S	Array bound exceeded.	CWE 121,122,124,126,127,129,193,466,758,823
45	48 S	No default case in switch statement.	CWE 478
1	51 S	Shifting value too far.	CWE 758
0	52 S	Unsigned expression negated.	CWE 192,197
0	54 S	Sizeof operator with side effects.	CWE 737
0	56 S	Equality comparison of floating point.	CWE 682
290	57 S	Statement with no side effect.	CWE 398,482
71	58 S	Null statement found.	CWE 747
17	64 S	Void procedure used in expression.	CWE 758
0	65 S	Void variable passed as parameter.	CWE 758
0	66 S	Function with empty return expression.	CWE 758
32	71 S	Pointer assignment to wider scope.	CWE 562,672,758,825
0	73 S	Bit field not signed or unsigned int.	CWE 758
0	76 S	More than one of # or ## in a macro.	CWE 758
6	77 S	Macro replacement list needs parentheses.	CWE 735
10	78 S	Macro parameter not in brackets.	CWE 735
0	81 S	Use of trigraph.	CWE 735

0	86 S	Attempt to define reserved word.	CWE 746
1065	93 S	Value assigned to a variable is in improper type	CWE 192,197,758
24	98 S	Actual and formal parameters inconsistent (MR).	CWE 685
1	99 S	Function not called	CWE 480
0	100 S	#include filename is non conformant.	CWE 758
7	101 S	Function return type inconsistent.	CWE 192,197
21	105 S	Initialisation brace { } fault.	CWE 758
5	107 S	Type mismatch in ternary expression.	CWE 192
0	113 S	Non standard character in source.	CWE 747
0	115 S	String incorrectly terminated.	CWE 741
0	118 S	main must be int (void) or int (int,char*[]).	CWE 744
6	119 S	Nested comment found.	CWE 747
30	127 S	Array has no bounds specified.	CWE 665,758
80	131 S	Name reused in inner scope.	CWE 736
0	134 S	Volatile variable in complex expression.	CWE 758
0	135 S	Parameter list is KR.	CWE 628
0	136 S	Bit operator with boolean operand.	CWE 480
50	139 S	Construct leads to infeasible code.	CWE 561,570,571
8	140 S	Infeasible loop condition found.	CWE 561,570,571
0	148 S	No return type for function/procedure.	CWE 628
500	157 S	Modification of string literal.	CWE 758
0	176 S	Non standard escape sequence in source.	CWE 176,758
306	203 S	Cast on a constant value.	CWE 704
0	296 S	Function declared at block scope.	CWE 758
0	299 S	Pointer to function declared without typedef.	CWE 736
32	302 S	Comment possibly contains code.	CWE 747
0	326 S	Declaration is missing type.	CWE 628
257	329 S	Operation not appropriate to plain char.	CWE 327,682
0	330 S	Implicit conversion of underlying type.	CWE 192
0	335 S	Operator defined contains illegal items.	CWE 758
0	336 S	#if expansion contains define operator.	CWE 758
0	339 S	#include directive with illegal items.	CWE 758

2	340 S	Use of function like macro.	CWE 735
0	341 S	Preprocessor construct as macro parameter.	CWE 758
0	344 S	Cast on volatile value.	CWE 704
0	355 S	Variables not unique within 64 characters.	CWE 736
0	376 S	Use of octal escape sequence.	CWE 176
27	381 S	Enum, struct or union not typedeffed.	CWE 736
0	389 S	Bool value incremented/decremented.	CWE 136
29	397 S	Array initialisation has insufficient items.	CWE 740
0	402 S	Comparison of booleans.	CWE 136
0	403 S	Negative (or potentially negative) shift.	CWE 190,680,681
2	404 S	Array initialisation has too many items.	CWE 665
13	407 S	Free used on string.	CWE 590,758,762
0	408 S	Volatile variable accessed on RHS of && or ||.	CWE 664,682,737,768
2	411 S	Inappropriate value assigned to enum.	CWE 192
12	433 S	Type conversion without cast.	CWE 192,197
310	434 S	Signed/unsigned conversion without cast.	CWE 192,195,196,197
0	435 S	Float/integer conversion without cast.	CWE 192
0	437 S	<><= >= used on different object pointers.	CWE 758
0	438 S	Pointer subtraction not addressing one array.	CWE 468,469,758
0	439 S	Cast from pointer to integral type.	CWE 681
302	440 S	Cast from integral type to pointer.	CWE 587,588,681
0	441 S	Float cast to non-float.	CWE 758
0	442 S	Signed integral type cast to unsigned.	CWE 192,195,197
0	443 S	Unsigned integral type cast to signed.	CWE 192,196,197
0	444 S	Integral type cast to non-integral.	CWE 681
16	446 S	Narrower int conversion without cast.	CWE 192,194,197
0	450 S	Wide string and string concatenated.	CWE 758
0	452 S	No cast for widening complex int expression.	CWE 192
0	456 S	Implicit float widening for function return.	CWE 704
0	457 S	Implicit int widening for function return.	CWE 192
2068	458 S	Implicit conversion: actual to formal param.	CWE 192,197
0	465 S	Struct/union not completely specified.	CWE 758

0	479 S	Right shift loses all bits.	CWE 189
1	480 S	String function params access same variable.	CWE 758
0	482 S	Incomplete structure referenced.	CWE 758
12	483 S	free parameter is not heap item.	CWE 590,758
40	484 S	Attempt to use already freed object.	CWE 415,416,758
1	486 S	Incorrect number of formats in output function.	CWE 134,685,686,688,758
0	487 S	Insufficient space allocated.	CWE 131,190,680,758
4	488 S	Value outside range of underlying type.	CWE 190
0	489 S	Insufficient space for operation.	CWE 121,122,758
2	491 S	No cast for widening int parameter.	CWE 192
0	493 S	Numeric overflow.	CWE 128,190,192,197,680,758
0	494 S	Numeric underflow.	CWE 128,190,191,192,197,680,758
3440	496 S	Function call with no prior declaration.	CWE 628
0	503 S	Function returns local resources.	CWE 562,825
0	545 S	Assignment of overlapping storage.	CWE 758,843
11	553 S	Function and proto should both be static.	CWE 736
0	565 S	Assignment to wider scope.	CWE 758
75	567 S	Pointer arithmetic is not on array.	CWE 188,468,469
0	568 S	#include "filename" uses standard library name.	CWE 735
0	573 S	Macro concatenation of uni char names.	CWE 735,758
0	575 S	Linkage differs from previous declaration.	CWE 758
82	576 S	Function pointer is of wrong type.	CWE 468,686,688,758
16	577 S	Sizeof argument is a pointer.	CWE 467
154	579 S	More than one variable per declaration.	CWE 736
1135	582 S	const object reassigned.	CWE 758
19	585 S	Bitwise and arith operations on same data.	CWE 738
0	586 S	Format is not %j for user defined type.	CWE 738
0	587 S	Const local variable not immediately initialised.	CWE 758
0	588 S	Use of system function.	CWE 78,88
7	589 S	Format is not appropriate type.	CWE 686,688,758
0	590 S	Mode fault in fopen.	CWE 758

0	591 S	Inappropriate use of file pointer.	CWE 743
0	592 S	Use of filename based functions.	CWE 367,676
0	593 S	Use fseek() rather than rewind().	CWE 743
0	594 S	Use setvbuf() rather than setbuf().	CWE 743
0	600 S	Argument of strlen is unterminated.	CWE 170
1	606 S	Cast involving function pointer.	CWE 737
0	617 S	Function call may return unexpected value.	CWE 758
0	618 S	Use of memcmp between structures.	CWE 188
41	623 S	String assigned to non const object.	CWE 741
1	629 S	Divide by zero found.	CWE 369
0	630 S	Duplicated enumeration value.	CWE 738
0	631 S	Declaration not reachable.	CWE 457
0	633 S	Use of a broken or risky cryptography algorithm.	CWE 327
-	5 C	Procedure contains infinite loop.	CWE 835
-	2 D	Function does not return a value on all paths.	CWE 758
-	6 D	Recursion in procedure calls found.	CWE 674
-	8 D	DD data flow anomalies found.	CWE 563
-	15 D	Unused procedural parameter.	CWE 398,747
-	17 D	Identifier not unique within 31 characters.	CWE 758
-	20 D	No declaration for variable found before use.	CWE 398
-	27 D	Variable should be declared static.	CWE 736
-	28 D	Potentially infinite loop found.	CWE 835
-	35 D	Expression has side effects.	CWE 737,758,768
-	41 D	Procedure call has no prototype declared.	CWE 628
-	43 D	Divide by 0 found.	CWE 129,190,369,680,758
-	48 D	Attempt to write to unopened file.	CWE 399,910
-	53 D	Attempt to use uninitialised pointer.	CWE 457,468,758
-	57 D	Global not initialised at declaration.	CWE 456
-	61 D	Procedure should be declared static.	CWE 736
-	62 D	Pointer parameter should be declared const.	CWE 736

-	63 D	No definition in system for prototyped procedure.	CWE 758
-	74 D	Potential side effect from repeated function call.	CWE 758
-	78 D	Global variable should be declared const.	CWE 736
-	80 D	Potentially unused function-modified value.	CWE 252,273,391
-	83 D	Potentially repeated call to ungetc.	CWE 675,758
-	85 D	Filename not verified before fopen.	CWE 22,37,38,39,41,59
-	86 D	User input not checked before use.	CWE 134
-	91 D	Function return value potentially unused.	CWE 252,273,391
-	93 D	Local variable should be declared const.	CWE 736
-	94 D	Named variable declared but not used in code.	CWE 563
-	97 D	Signal called from within signal handler.	CWE 479
-	105 D	DU anomaly dead code, variable value is unused on all paths.	CWE 14,563
-	61 X	Identifier match in 31 chars.	CWE 758
-	62 X	Function prototype/defn return type mismatch (MR).	CWE 758
-	63 X	Function prototype/defn param type mismatch (MR).	CWE 758
-	64 X	Array bound exceeded at call.	CWE 121,122,124,126,127,129,193,466,758,823
-	66 X	Insufficient array space at call.	CWE 121,122,758
-	67 X	Identifier is typographically ambiguous.	CWE 736
-	68 X	Parameter indexing array too big at call.	CWE 121,122,124,126,127,129,193,466,758,823
-	69 X	Global array bound exceeded at use.	CWE 121,122,124,126,127,129,193,466,758,823
-	70 X	Array has insufficient space.	CWE 121,122,758
-	71 X	Insufficient space for copy.	CWE 121,122,758
0	1 Q	Call has execution order dependant side effects.	CWE 737,758,768
0	1 U	Inter-file recursion found.	CWE 561,674
7	1 J	Unreachable Code found.	CWE 561,570,571
1850	49 S	Logical conjunctions need brackets.	CWE 783
380	59 S	Else alternative missing in if.	CWE 697

1564	94 S	Casting operation on a pointer.	CWE 737
676	95 S	Casting operation to a pointer.	CWE 737
200	96 S	Use of mixed mode arithmetic.	CWE 192
58	132 S	Assignment operator in Boolean expression.	CWE 481
969	331 S	Literal value requires a U suffix.	CWE 195
24	361 S	Expression needs brackets.	CWE 783
-	69 D	Procedure contains UR data flow anomalies.	CWE 457,665,758,824,908

According to the output above, LDRA found 688 unique Security vulnerabilities in the Asterisk code base (chan_sip.c), and many of these vulnerabilities were found in the code several times, resulting in a "FAIL" result [24,25].

5. Conclusion

Static code analysis proved to be a potent and efficient method of checking code for vulnerabilities.Many Security flaws were found in the Asterisk 1.8 source code, and all five open-source static analysis tools and the commercial programme LDRA found them. In contrast to open-source alternatives, the commercial programme LDRA found more Security vulnerabilities and provided a more thorough analysis report. In addition, LDRA encourages a wide variety of programming languages and security standards, while most open-source tools only support a single language and a single security standard (such as C, C++, PHP, JAVA, ADA, or NETRINO). As a consequence of this effort, we can now use Static code analysis to examine a particular C/C++ source file.

Compliance with Ethical Standards

Fundings: No Funds received.

Conflict of interest: The authors declare that they have no conflict of interest.

Ethical approval: This article does not contain any studies with human participants performed by any of the authors.

References

[1] R. Rosenberg, S. Bique, M. Koop, K. Andersen and C. Kung, "Exploring Best Practices for the DSRCs with Benchmarking," *2010 DoD High Performance Computing Modernization Program Users Group Conference*, Schaumburg, IL, USA, 2010, pp. 498-507.

[2] T. Ball and S. G. Eick, "Visualizing program slices," *Proceedings of 1994 IEEE Symposium on Visual Languages*, St. Louis, MO, USA, 1994, pp. 288-295.

[3] T. Ball and J. R. Larus, "Efficient path profiling," *Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture. MICRO 29*, Paris, France, 1996, pp. 46-57.

[4] F. Balmas, "Using dependence graphs as a support to document programs," *Proceedings. Second IEEE International Workshop on Source Code Analysis and Manipulation*, Montreal, QC, Canada, 2002, pp. 145-154.

[5] D. Binkley, "Semantics guided regression test cost reduction," in *IEEE Transactions on Software Engineering*, vol. 23, no. 8, pp. 498-516, Aug. 1997.

[6] D. Binkley and M. Harman, "Results from a large-scale study of performance optimization techniques for source code analyses based on graph reachability algorithms," *Proceedings Third IEEE International Workshop on Source Code Analysis and Manipulation*, Amsterdam, Netherlands, 2003, pp. 203-212.

[7] D. Binkley and M. Harman, "Analysis and visualization of predicate dependence on formal parameters and global variables," in *IEEE Transactions on Software Engineering*, vol. 30, no. 11, pp. 715-735, Nov. 2004.

[8] D. Binkley, M. Harman and J. Krinke, "Characterising, Explaining, and Exploiting the Approximate Nature of Static Analysis through Animation," *2006 Sixth IEEE International Workshop on Source Code Analysis and Manipulation*, Philadelphia, PA, USA, 2006, pp. 43-52.

[9] G. Canfora and M. Di Penta, "New Frontiers of Reverse Engineering," *Future of Software Engineering (FOSE '07)*, Minneapolis, MN, USA, 2007, pp. 326-341.

[10] H. Cleve and A. Zeller, "Locating causes of program failures," *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, St. Louis, MO, USA, 2005, pp. 342-351.

[11] J. M. Cobleigh, L. A. Clarke and L. J. Osterweil, "FLAVERS: A finite state verification technique for software systems," in *IBM Systems Journal*, vol. 41, no. 1, pp. 140-165, 2002.

[12] A. De Lucia, A. R. Fasolino and M. Munro, "Understanding function behaviors through program slicing," *WPC '96. 4th Workshop on Program Comprehension*, Berlin, Germany, 1996, pp. 9-18.

[13] M. B. Dwyer, J. Hatcliff, R. Robby, C. S. Pasareanu and W. Visser, "Formal Software Analysis Emerging Trends in Software Model Checking," *Future of Software Engineering (FOSE '07)*, Minneapolis, MN, USA, 2007, pp. 120-136.

[14] K. B. Gallagher and J. R. Lyle, "Using program slicing in software maintenance," in *IEEE Transactions on Software Engineering*, vol. 17, no. 8, pp. 751-761, Aug. 1991.

[15] R. Gupta, M. J. Harrold and M. L. Soffa, "An approach to regression testing using slicing," *Proceedings Conference on Software Maintenance 1992*, Orlando, FL, USA, 1992, pp. 299-308.

[16] M. Harman, C. Fox, R. Hierons, D. Binkley and S. Danicic, "Program simplification as a means of approximating undecidable propositions," *Proceedings Seventh International Workshop on Program Comprehension*, Pittsburgh, PA, USA, 1999, pp. 208-217.

[17] M. Harman, N. Gold, R. Hierons and D. Binkley, "Code extraction algorithms which unify slicing and concept assignment," *Ninth Working Conference on Reverse Engineering, 2002. Proceedings.*, Richmond, VA, USA, 2002, pp. 11-20.

[18] M. Harman, B. Korel and P. K. Linos, "Guest Editorial: Special Issue on Software Maintenance and Evolution," in *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 801-803, Oct. 2005.

[19] M. P. E. Heimdahl, "Safety and Software Intensive Systems: Challenges Old and New," *Future of Software Engineering (FOSE '07)*, Minneapolis, MN, USA, 2007, pp. 137-152.

[20] D. Heuzeroth, T. Holl, G. Hogstrom and W. Lowe, "Automatic design pattern detection," *11th IEEE International Workshop on Program Comprehension, 2003.*, Portland, OR, USA, 2003, pp. 94-103.

[21] S. Horwitz and T. Reps, "The use of program dependence graphs in software engineering," *International Conference on Software Engineering*, Melbourne, VIC, Australia, 1992, pp. 392-411.

[22] V. Issarny, M. Caporuscio and N. Georgantas, "A Perspective on the Future of Middleware-based Software Engineering," *Future of Software Engineering (FOSE '07)*, Minneapolis, MN, USA, 2007, pp. 244-258.

[23] B. Korel and J. Rilling, "Dynamic program slicing in understanding of program execution," *Proceedings Fifth International Workshop on Program Comprehension. IWPC'97*, Dearborn, MI, USA, 1997, pp. 80-89.

[24] A. Lakhotia and E. U. Kumar, "Abstracting stack to detect obfuscated calls in binaries," *Source Code Analysis and Manipulation, Fourth IEEE International Workshop on*, Chicago, IL, USA, 2004, pp. 17-26.

[25] A. Lakhotia, Junwei Li, A. Walenstein and Yun Yang, "Towards a clone detection benchmark suite and results archive," *11th IEEE International Workshop on Program Comprehension, 2003.*, Portland, OR, USA, 2003, pp. 285-286.