XXIII Brazilian Symposium on Information and Computational Systems Security
Juiz de Fora/Mg - 2023

LSVerifier: A BMC Approach to Identify Security Vulnerabilities in C Open-Source Software Projects

Janislley Oliveira de Sousa, Bruno Carvalho de Farias, Thales Araujo da Silva, Eddie Batista de Lima Filho, Lucas Carvalho Cordeiro

Email: janislley.sousa@sidia.com

SBSeg|23
JUIZ DE FORA

XXIII Brazilian Symposium on Information
and Computational Systems Security
Juiz de Fora/Mg - 2023

LSVerifier Team:

Mr. Janislley Oliveir
(UFAM/SIDIA)

Bruno Farias
(U. Manchester)

M.Sc. Thales Silva
(UFAM)

Dr. Eddie Batista
(TPV/UFAM)

Dr. Lucas Cordeiro
(U. Manchester/UFAM)

ESBMC

UFAM

MANCHESTER 1824
The University of Manchester

SAMSUNG SIDIA

Motivation

Overview of research area challenges.

- **Airbus** discovered a software vulnerability in the **A400M aircraft,** leading to a crash in 2015.
- **Security researchers** could remotely exploit a vulnerability in the **Jeep Cherokee's Uconnect infotainment system.**
- **Samsung** fixes bug that allowed **Galaxy Smartphones** to be hacked since 2014. The security flaw allowed attackers to have easy access to **Skia, Android's graphics library**.

Motivation

Overview of research area challenges.

- **Software validation and verification techniques** are essential tools for developing robust systems with **high dependability** and **reliability requirements**.

- **Memory errors** in C software written in **unsafe programming languages** represent one of the main problems in **computer security**.

- The **Common Weakness Enumeration (CWE)** community identified a lot of vulnerabilities regarding the **C programming** language in **third-party libraries** used on the **open-source projects**.

The CWE Top 13

#	ID	Name
1	**CWE-787**	**Out-of-bounds Write**
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
4	CWE-20	Improper Input Validation
5	**CWE-125**	**Out-of-bounds Read**
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
7	**CWE-416**	**Use After Free**
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
9	CWE-352	Cross-Site Request Forgery (CSRF)
10	CWE-434	Unrestricted Upload of File with Dangerous Type
11	**CWE-476**	**NULL Pointer Dereference**
12	CWE-502	Deserialization of Untrusted Data
13	**CWE-190**	**Integer Overflow or Wraparound**

More info: https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html

The problem

Challenges and motivations.

- The **C programming language** is widely used to develop **critical software**, such as operating systems, drivers, and encryption libraries. However, it **lacks protection mechanisms**, leaving memory and resource management's responsibility in the **developers' hands**.

- **Large software systems** are frequently composed of a myriad of elements declared in **several source files**, usually divided into **various directories**.

- To **handle large pieces of software** with many files, a scenario typically found in open-source applications, it is necessary to **verify each one** at once and then change the current entry point when required.

Objective

What will be done?

- Propose a new approach using a **bounded model checker** to **exploit security bugs** for **C open-source** software projects.
- An **in-depth evaluation** of our approach over a **dataset of large open-source applications** in order to find security vulnerabilities.

LSVerifier

Open-Source Tool

Apache 2.0

https://github.com/janislley/LSVerifier

A novel verification tool combining **input-code analysis** and **BMC technique** to detect **software vulnerabilities** for **Open-Source C software** projects.

Verification Algorithm

Description of the proposed approach.

Algorithm 1 The proposed verification approach.

Require: Program P, Directory D, Configuration C,
Ensure: Verification Outcome V

$configs \leftarrow get\_configs(C)$
$files \leftarrow list\_files(P, D)$
$num\_files \leftarrow length(files)$
$k \leftarrow 1$
while $k \leq num\_files$ **do**
  $functions \leftarrow list\_function(files[k])$
  $log \leftarrow ESBMC\_Check(files[k], functions[k], configs)$
  $k \leftarrow k + 1$
end while
$V \leftarrow spreedsheat(log)$
return V

SBSeg|23 JUIZ DE FORA

Verification Algorithm

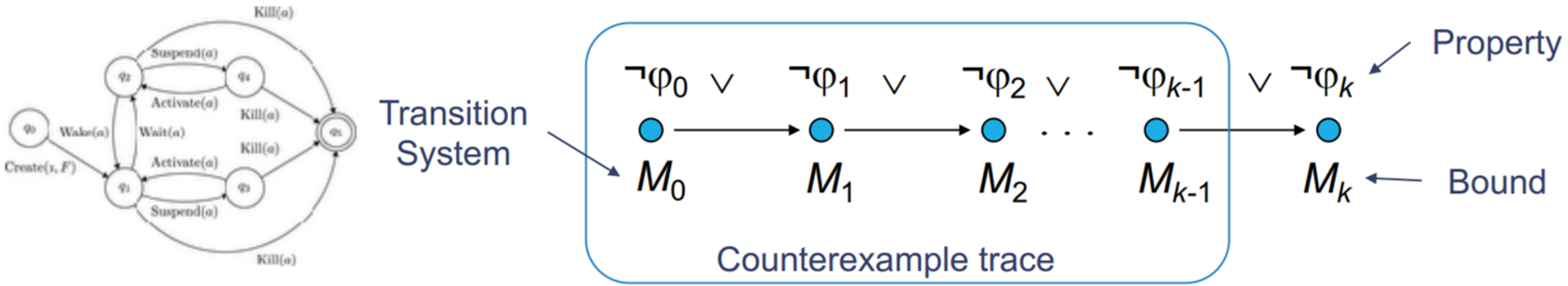
Description of the proposed approach.

- CTAGS is used to list all functions, variables, marcos, etc. in a C file.
- Generate a AST (Abstract Syntax Tree) with all classified data.
- Generate a CFG (Control Flot Chat).
- ESBMC uses Boolector as SMT solver by default when none is specified in command line.
- ESBMC's module is used to convert C programs into GOTO ones.
- A State Machine is used to analyze all processed data.
- LSVerfier is implemented in Python, and ESBMC module is implemented in C/C++ .
- The ESBMC module is used as binary.

Bounded Model Checking (BMC) Approach

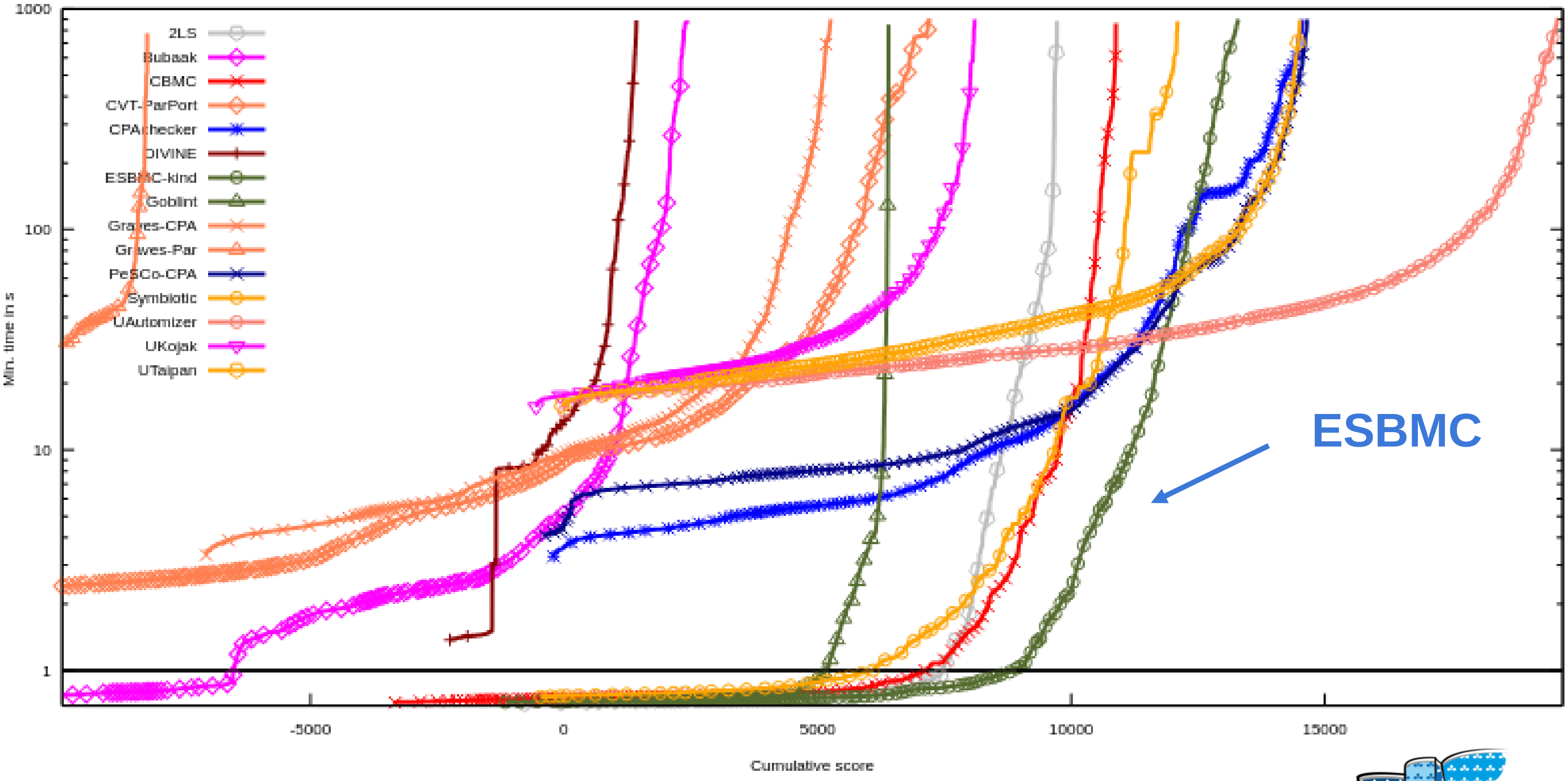
- **Basic Idea:** given a transition system **M**, check negation of a given property **φ** up to given depth **k**.

$$\mathrm{BMC}_{\Phi}(k) = I(s_1) \wedge \left(\bigwedge_{i=1}^{k-1} T(s_i, s_{i+1}) \right) \wedge \left(\bigvee_{i=1}^{k} \neg\phi(s_i) \right)$$

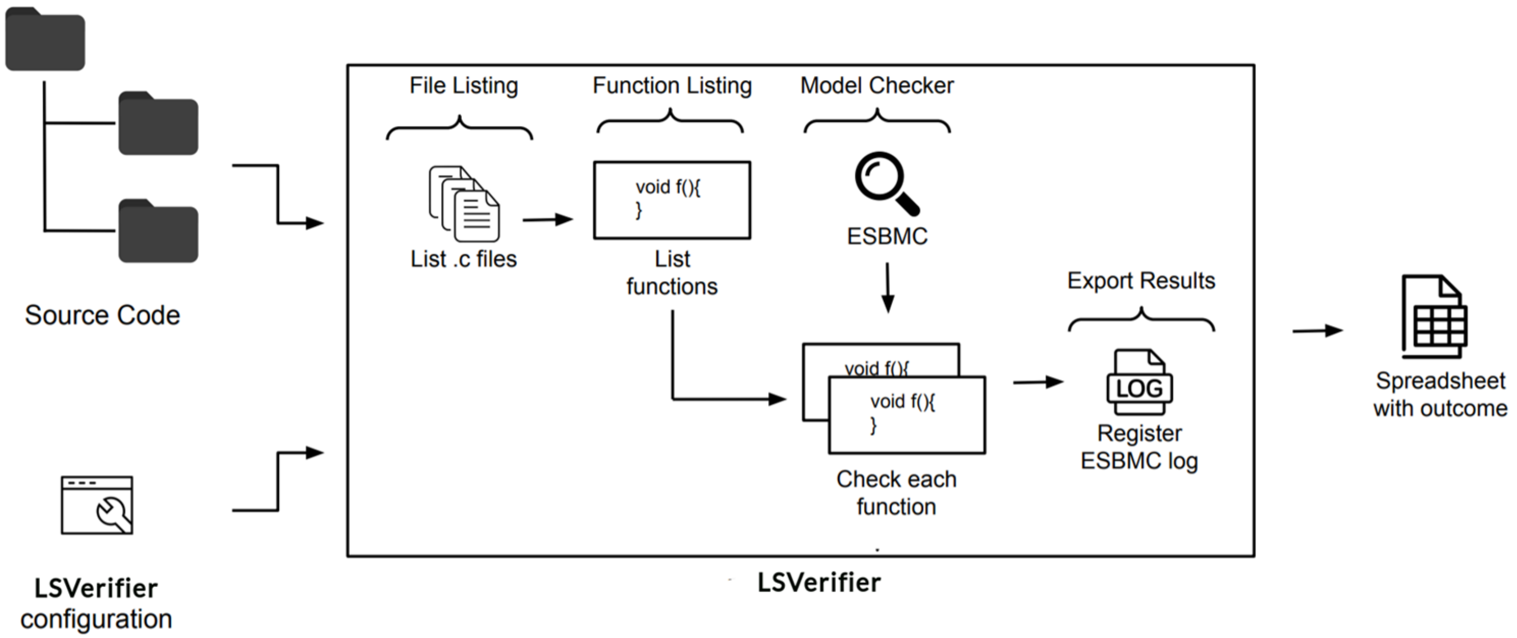


- Bounded model checkers “slice” the state space in depth.
- It is aimed to find bugs and can only prove correctness if all states are reachable within the bound.
- Exhaustively explores all executions.
- Can be bounded to limit number of iterations and context-switch.
- Report errors as traces.

SV-COMP 2023 - ESBMC module



Large Systems Verifier (LSVerifier) Architecture



- The Tool takes a **source-code directory**, a **software**, and **dependencies configuration** as inputs. It then lists all .c files and iterates through them to verify each function.
- The verification outcomes are compiled into a **report (logs, CSV files)**, which is returned as final output.

LSVerifier: Property Verification Process

- Configuration parameters are divided into the following groups:
  - File listing;
  - Function verification;
  - Outcome display;
  - ESBMC module options;

- **LSVerifier tool options** to code verification:

```
parser = argparse.ArgumentParser(description="Input Options", formatter_class=NewLineHelpFormatter)
parser.add_argument("-l", "--libraries", help="Path to the file that describes the libraries dependencies", default=False)
parser.add_argument("-p", "--properties",)
parser.add_argument("-f", "--functions", help="Enable Functions Verification", action="store_true", default=False)
parser.add_argument("-fp", "--function-prioritized", help="Enable Prioritized Functions Verification", action="store_true", default=False)
parser.add_argument("-fl", "--file", help="File to be verified", default=False)
parser.add_argument("-v", "--verbose", help="Enable Verbose Output", action="store_true", default=False)
parser.add_argument("-r", "--recursive", help="Enable Recursive Verification", action="store_true", default=False)
parser.add_argument("-d", "--directory", help="Set the directory to be verified", default=False)
parser.add_argument("-dp", "--disable-pointer-check", help="Disable invalid pointer verification", action="store_true", default=False)
parser.add_argument("-e", "--esbmc-parameter", help="Use ESBMC parameter")
```

- Tool repository: https://github.com/janislley/LSVerifier

LSVerifier – Property Verification Process

More artefacts:

```
/usr/include/glib-2.0/
/usr/lib/x86_64-linux-gnu/glib-2.0/include/
extcap/
plugins/epan/ethercat/
plugins/epan/falco_bridge/
plugins/epan/wimaxmacphy/
randpkt_core/
writecap/
epan/crypt/
...
```

Project dependencies example (dep.txt).

```
void vector_func(char c[]) {
    c[2] = 'a';
    int i[1];
    i[2] = 1;
}
```

re with bug.

```
void vector_func(char c[]) {
    c[2] = 'a';
    int i[1];
    i[0] = 1;
}
```

ked.

```
Counterexample:

State 2 file main.c line 51 function vector_func thread 0
----------------------------------------------------
Violated property:
file main.c line 51 function vector_func
array bounds violated: array `i' upper bound
```

Property violation log.

LSVerifier: Property Verification Process

Vulnerability type	CWE numbers
Buffer overflow	CWE-20, CWE-120, CWE-121, CWE-125, CWE-129, CWE-131, CWE-676, CWE-628, CWE-754, CWE-788
Arithmetic overflow	CWE-190, CWE-191, CWE-754, CWE-680, CWE-681, CWE-682
Array bounds violated	CWE-119, CWE-125, CWE-129, CWE-131, CWE-193, CWE-755, CWE-787, CWE-788
NULL pointer	CWE-391, CWE-476
Invalid pointer	CWE-416, CWE-476, CWE-690, CWE-822, CWE-824, CWE-825
Double free	CWE-415
Division by zero	CWE-369
Memory Leak	CWE-401
Other vulnerabilities	CWE-119, CWE-125, CWE-158, CWE-362, CWE-389, CWE-459, CWE-416, CWE-469, CWE-590, CWE-617, CWE-664, CWE-662, CWE-674, CWE-685, CWE-704, CWE-761, CWE-787, CWE-789, CWE-823, CWE-825, CWE-843, CWE-908

- **LSVerifier tool has support** to exploit the following properties violations:
  - Out-of-bounds array access;
  - Illegal pointer dereferences (null dereferencing, out-of-bounds dereferencing, double free, and misaligned memory access);
  - Arithmetic overflow;
  - Buffer overflow;
  - Not a number (NaN) occurrences in floating-point;
  - Division by zero;
  - Memory leak;
  - Dynamic memory allocation;
  - Data races;
  - Deadlock;
  - Atomicity violations at visible assignments.

LSVerifier – Property Verification Process

- Property verification for an entire project:

```
$ lsverifier -v -r -f -e "--unwind 1 --no-unwinding-assertions" -l dep.txt
```

- Property verification for specific .c files:

```
$ lsverifier -v -r -f -fl main.c
```

- Property verification for a specific path:

```
$ lsverifier -v -r -f -l dep.txt -d project-root/
```

- Property verification by specific class of vulnerability:

```
$ lsverifier -v -r -f -p memory-leak-check,overflow-check,deadlock-check,data-races-check
```

More details: **https://github.com/janislley/LSVerifier/blob/main/README.md**

Experimental Methodology

Inputs and definitions for the proposed approach validation.

- Experimental Tests:
  - CPU consumption.
  - Memory usage.
- Use the standard: **Confidentiality**, **Integrity**, and **Availability** (**CIA**) triad.
- Security management standard **ISO/IEC 27001.**
- As benchmarking, we prepared a **dataset** consisting of five commonly used software modules based on the C language: **RUFUS**, **OpenSSH**, **CMake**, **Wireshark**, and **PuTTY**.

Experimental Results

Data collected and analyzed.

- With verification logs report (counterexample traces) provided by tool, we reported the **critical issues** found on the dataset to the respective software owners.
- Issues were identified in **2 open-source projects** and fixed.
- Most of the issues were related with **third-party libraries**.
- The LSVerifier tool maintained **low peak memory usage,** which significantly differs from other recent verification tools based on model checking.

Experimental Results: Properties violated and CWE categories

Table 1. Dataset analysis using LSVerifier tool.

Software	Property violations	Files analyzed	Functions verified	Overall time	Peak memory usage
VIM	5	184	8804	406.02 s	36.46 MB
RUFUS	186	142	1575	101.59 s	32.6 MB
OpenSSH	337	286	3033	490.33 s	15.32 MB
Wireshark	122	2194	108824	39413.97 s	119.52 MB
PuTTY	2019	244	4575	91448.89 s	53.79 MB

- Dataset: https://github.com/janislley/LSVerifier_Benchmarks
- Issues reported and fixed:
    - RUFUS: https://github.com/pbatard/rufus/issues/1856 (CWE-119)
    - WIreshark: https://gitlab.com/wireshark/wireshark/-/issues/17897 (CWE-416)
- RUFUS presented property violations such as array out of bounds, with 3 issues opened and 1 fixed regarding imported libraries.
- The Wireshark's property violations, which are related to array out of bounds and invalid pointers, were due to errors in the NPL third-party library.

Experimental Results: Properties violated on RUFUS

re.c (8 changes)

```diff
@@ -251,7 +251,7 @@ re_t re_compile(const char* pattern)

 void re_print(regex_t* pattern)
 {
-  const char* types[] = { "UNUSED", "DOT", "BEGIN", "END", "QUESTIONMARK", "STAR", "PLUS", "CHAR", "CHAR_CLASS", "INV_CHAR_CLASS", "DIGIT", "NOT_DIGIT", "ALPHA", "NOT_ALPHA", "WHITESPACE", "NOT_WHITESPACE", "BRANCH" };
+  const char* types[] = { "UNUSED", "DOT", "BEGIN", "END", "QUESTIONMARK", "STAR", "PLUS", "CHAR", "CHAR_CLASS", "INV_CHAR_CLASS", "DIGIT", "NOT_DIGIT", "ALPHA", "NOT_ALPHA", "WHITESPACE", "NOT_WHITESPACE" /*, "BRANCH" */ };

   int i;
   int j;
@@ -263,7 +263,11 @@ void re_print(regex_t* pattern)
         break;
       }

-    printf("type: %s", types[pattern[i].type]);
+    if (pattern[i].type <= NOT_WHITESPACE)
+      printf("type: %s", types[pattern[i].type]);
+    else
+      printf("invalid type: %d", pattern[i].type);
+
```

Array bounds violated: array `types' upper bound fix (**CWE-119**).
This issue was **fixed** and others **9 fixes** were provide for another parts in re.c
in **tyne-regex third-party library**. More details: https://github.com/kokke/tiny-regex-c/pull/78

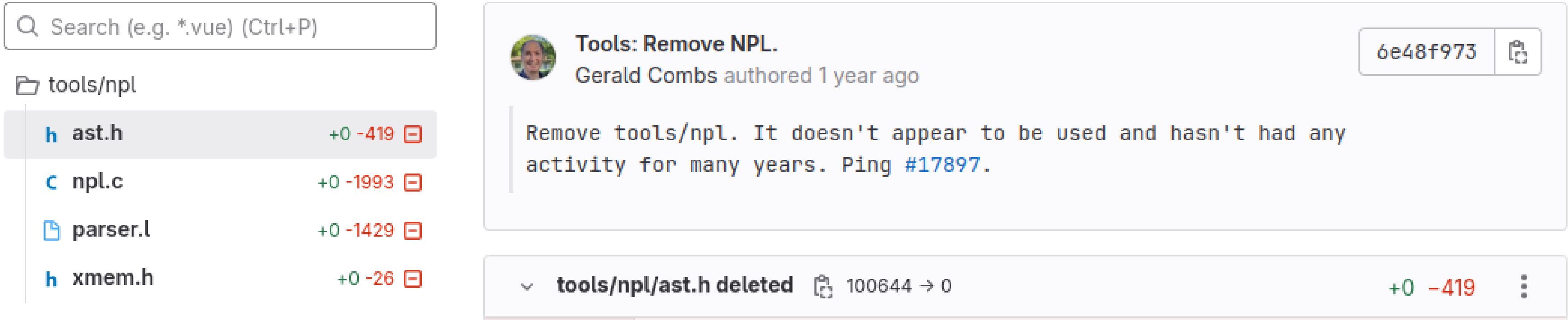
Experimental Results: Properties violated on RUFUS

```diff
7 re.c
@@ -296,15 +296,15 @@ void re_print(regex_t* pattern)
 /* Private functions: */
 static int matchdigit(char c)
 {
-  return isdigit(c);
+  return isdigit((unsigned char)c);
 }
 static int matchalpha(char c)
 {
-  return isalpha(c);
+  return isalpha((unsigned char)c);
 }
```

```diff
-      printf("type: %s", types[pattern[i].type]);
+      if (pattern[i].type <= NOT_WHITESPACE)
+        printf("type: %s", types[pattern[i].type]);
+      else
+        printf("invalid type: %d", pattern[i].type);
+
```

Array bounds violated: array `types' upper bound fix (**CWE-119**).
This issue was **fixed** and others **9 fixes** were provide for another parts in re.c
in **tyne-regex third-party library**. More details: https://github.com/kokke/tiny-regex-c/pull/78

Experimental Results: Properties violated on Wireshark



- **Dereference failure (invalid pointer and Null pointer)** issues (**CWE-416**) were found in the **NPL third-party library**.
- The fix involved removing this library, as it is no longer used in Wireshark.
- More details: https://gitlab.com/wireshark/wireshark/-/merge_requests/6021

Conclusion and Future Work

- The **LSVerifier tool** is **released (v0.3.0)** as **Apache License 2.0** open-source software.
- The proposed tool was possible to check the files that make up the application one by one, identifying the functions listed in each file. It then executes the ESBMC verification, thus **identifying vulnerabilities and generating an output report summarizing all software weaknesses found**.
- While its design opens the door to directions like whole-system exploitation, LSVerifier tool implementation is **mature enough to handle large and complex open-source software** like **Wireshark** and **RUFUS**.
- The results show that the approach of vulnerability analysis is feasible and can be helpful to the **open-source software community**. It proved to be an important tool to check the **security of third-party libraries**.
- For future work, we aim to enhance the counterexample verification method by incorporating machine learning techniques to automatically suggest solutions for property violations.

XXIII Brazilian Symposium on Information and Computational Systems Security
Juiz de Fora/Mg - 2023

LSVerifier: A BMC Approach to Identify Security Vulnerabilities in C Open-Source Software Projects

Thank You!

Email: janislley.sousa@sidia.com Tool: https://github.com/janislley/LSVerifier