SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Verifysoft TECHNOLOGY

Was ist eigentlich „Statische Codeanalyse“?

Royd Lüdtke
Verifysoft Technology GmbH
luedtke@verifysoft.com
+49 781 127 8118-0

1

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Agenda

Verifysoft TECHNOLOGY

- Dynamische vs. statische Analyse
- Allegorie Hausbau
- Relative Kosten zur Fehlerbeseitigung
- „Manuelle“ statische Analyse
- Wie funktioniert ein Werkzeug zur statischen Analyse
- Funktionale Sicherheit
- Aufdeckung von Sicherheitsschwachstellen
- Vermeidung von Performance-Problemen
- Überprüfung auf Einhaltung von Programmierrichtlinien / Standards
- Kann man Softwarequalität messen?
- Spezielle Anwendung: Statische Analyse von Binärcode

2

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Statische vs. dynamische Analyse

Verifysoft TECHNOLOGY

Quellcode

```
void Compliant::init() {

  Rules rules;

  std::ifstream csvread(cs::csurf_info::install_path()
  + "/codesonar/nameconv.set");
  if (!csvread.is_open()) {
                std::cerr << "Can't open file!" << std::endl;
  }
  else {
     for (ReadSet rs; read_in(csvread, rs); ) {

        if (!rs.starts.empty()
            || !rs.contains.empty()
            || !rs.ends.empty()) {
           // Populate rules struct
           rules.starts = rs.starts;
           rules.contains = rs.contains;
           rules.ends = rs.ends;
           // Populate rulesmap
           rulesmap[rs.type] = rules;
        }
     }

  }

}
```

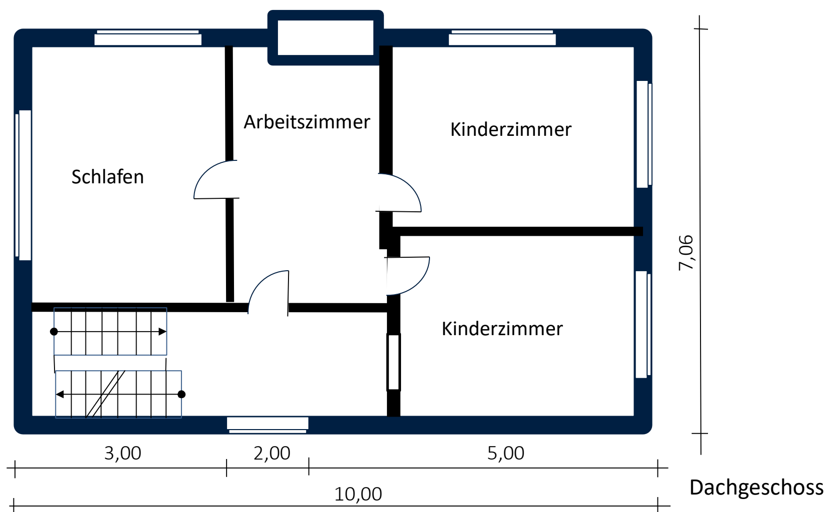
```
010011101001110100010110110100
010101011001001011110000110011
001010101100100100010010001000
1011111101100101010010010011001
0101010001010010001001010001001
0010010011111101001010101010101
0100100100101111110110011010100
1111110101001010101101001010101
0101010010100101010001010101010
1010101001010101000101010001010
1010010101001011000101010010100
1010101001001001010101010001010
1010010101010101010100111001010
0101010010101001010101010101001
0110010101101010101001100110011
0010101001101010010101001010101
0100110101010100010100101010100
0101010101001010101010100101001
1
```

Binärdatei

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Allegorie Hausbau (Quellcode)

Verifysoft TECHNOLOGY



Dachgeschoss

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Allegorie Hausbau (Compiler)

Verifysoft TECHNOLOGY

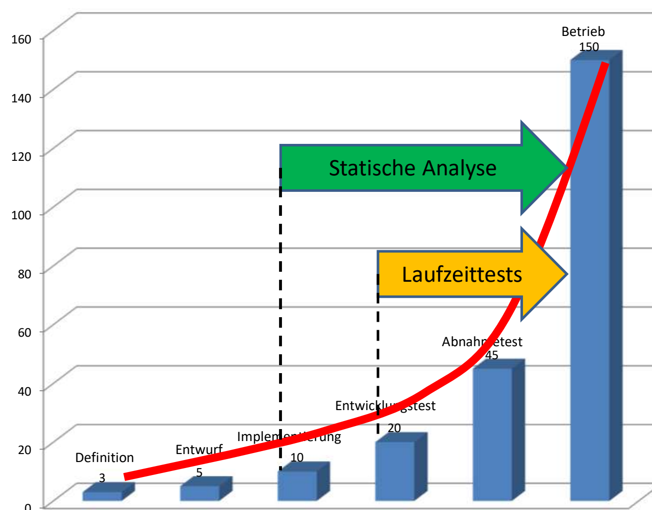
Foto: www.shutterstock.com

www.verifysoft.com

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Relative Kosten zur Fehlerbeseitigung (nach Barry W. Boehm)

Verifysoft TECHNOLOGY



www.verifysoft.com

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

„Manuelle“ statische Analyse

Verifysoft TECHNOLOGY

Seit er bei uns den Walk-through durchführt, ist unsere Software nahezu fehlerfrei!

Foto: www.shutterstock.com

www.verifysoft.com 7

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

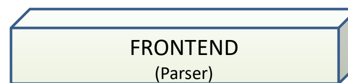
Wie funktioniert ein Werkzeug zur statischen Analyse?

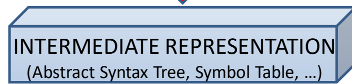
Verifysoft TECHNOLOGY

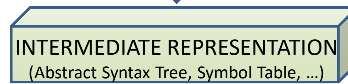
```
int main(void){
        printf("Hello World!\n");
        return 0;
}
```

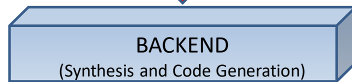
Compiler

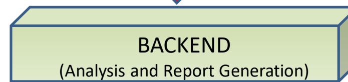
Static Analysis Tool

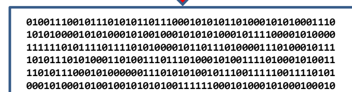


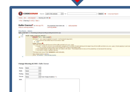








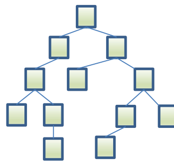




www.verifysoft.com 8

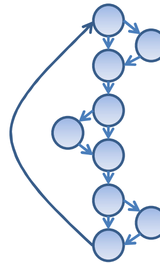
Quellcodepräsentation

Symbol Table



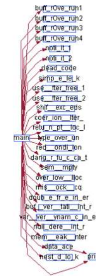
Name	Kind	Location
calc_sub	function	induct.c:137
i	variable	init.c:25

Abstract Syntax Tree (AST)



Control Flow Graph (CFG)

Call Graph



SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Verifysoft TECHNOLOGY

Funktionale Sicherheit

Foto: www.shutterstock.com

SAD 2021 Static Analysis Days

Reviews und Analysen vom Quellcode nach DO-178C

Verifysoft TECHNOLOGY

Überprüfungen	DAL				
	A	B	C	D	E
Quellcode ist akkurat und vollständig, enthält keine Implementierung nicht dokumentierter Funktionen	++	++	+	-	-
Quellcode stimmt mit den Architekturvorgaben hinsichtlich Datenfluss und Kontrollfluss überein	++	+	+	-	-
Quellcode ist ohne Modifikation verifizierbar	+	+	-	-	-
Quellcode ist standardkonform (z.B. geringe Komplexität)	+	+	+	-	-
Quellcode ist nachvollziehbar im Hinblick auf die „low-level" Vorgaben	+	+	+	-	-
Quellcode ist akkurat und korrekt (z.B. keine nicht initialisierten Variablen, keine Speicherlecks etc.)	++	+	+	-	-

++ muss unabhängig erfüllt sein + muss erfüllt sein - im Ermessen

Quelle: DO-178C

SAD 2021 Static Analysis Days

Methoden zur Verifikation von Design und Implementierung nach ISO 26262

Verifysoft TECHNOLOGY

Methoden	ASIL			
	A	B	C	D
Walk-through	++	+	o	o
Inspection	+	++	++	++
Semi-formal verification	+	+	++	++
Formal verification	o	o	+	+
Control flow analysis	+	+	++	++
Data flow analysis	+	+	++	++
Static code analysis	+	++	++	++
Semantic code analysis	+	+	+	+

+ empfohlen ++ dringend empfohlen

Quelle: ISO/FDIS 26262-6:2011

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Exemplarische Auswahl an Prüfungen

Verifysoft TECHNOLOGY

- Speicherüberläufe
- Null-Pointer Dereferenzierung
- Division durch 0
- Nicht initialisierte Variablen
- Nicht erreichbarer Code
- Speicherfreigabe von Non-Heap Variablen
- Zugriff auf bereits freigegebenen Speicher
- Freigabe von bereits freigegebenem Speicher
- Rückgabe von Zeigern auf lokale Variablen
- Speicherlecks
- Copy and Paste Fehler
- Fehlendes Return Statement

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Verifysoft TECHNOLOGY

Aufdeckung von

Sicherheitsschwachstellen

Foto: www.shutterstock.com

SAD 2021 Static Analysis Days Online 5.-6. Mai 2021

Exemplarische Auswahl an Prüfungen

Verifysoft TECHNOLOGY

- Speicherüberläufe
- Nicht abgeschlossener C String
- Nicht vertrauenswürdiger Format String
- Additionsüberlauf bei vorgegebener Speichergrösse
- Multiplikationsüberlauf bei vorgegebener Speichergrösse
- Subtraktionsüberlauf bei vorgegebener Speichergrösse
- Fest programmierter DNS Name
- Speicherung eines Passwortes im Klartext
- Infizierte Netzwerkadresse
- Verwendung von „system()“

www.verifysoft.com 15

15

SAD 2021 Static Analysis Days Online 5.-6. Mai 2021

Möglicher Speicherüberlauf

Verifysoft TECHNOLOGY

```
#include <stdio.h>


struct {
        char name[20];
        int alter;
} person;


void main(void)
{
        scanf("%s", person.name);
}
```

www.verifysoft.com 16

16

SAD 2021 Static Analysis Days Online 5.-6. Mai 2021

Code Injection durch Speicherüberlauf

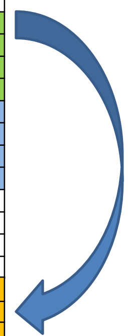
Verifysoft TECHNOLOGY

char buffer[4];

int i;

Rücksprungadresse

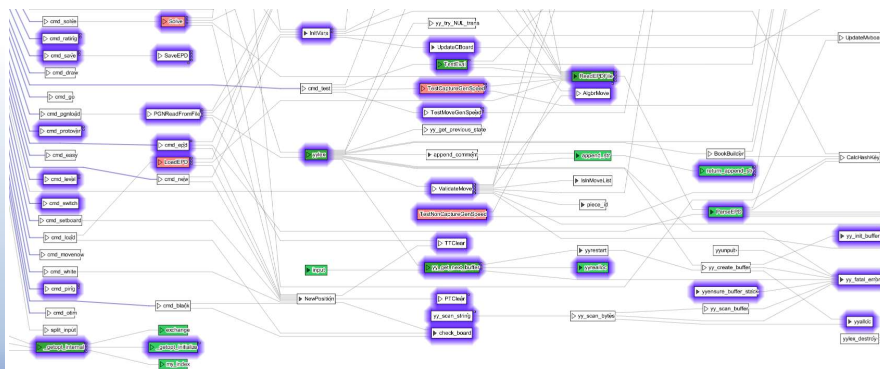
00007FF0h	
00007FF1h	
00007FF2h	
00007FF3h	
00007FF4h	#
00007FF5h	#
00007FF6h	#
00007FF7h	#
00007FF8h	#
00007FF9h	#
00007FFAh	#
00007FFBh	#
00007FFCh	#
00007FFDh	#
00007FFEh	#
00007FFFh	#
00008000h	54h / 03h
00008001h	7Fh / 77h
00008002h	00h / 05h
00008003h	00h / 00h
00008004h	3
00008005h	´B´
00008006h	6
00008007h	0



SAD 2021 Static Analysis Days Online 5.-6. Mai 2021

Taint Data Tracking

Verifysoft TECHNOLOGY



SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Verifysoft TECHNOLOGY

Foto: www.shutterstock.com

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Performance Problem: Partieller Datenzugriff

Verifysoft TECHNOLOGY

Performance Bottleneck

```
struct Test_Struct
{
        int a;
        int b;
        int c;
        int d;
        int e;
} ;
struct Test_Struct *precord;
…

for (i = 0; i < 1024 * 1024 * 60; i++)
{
        precord[i].b++;
}
```

Optimierung: Aufspalten der Struktur

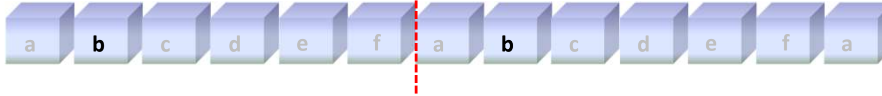
```
struct Test_Struct
{
        int a;
        int c;
        int d;
        int e;
} ;
struct Test_Struct *precord;

int *b;
…

for (i = 0; i < 1024 * 1024 * 60; i++)
{
        b[i]++;
}
```

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

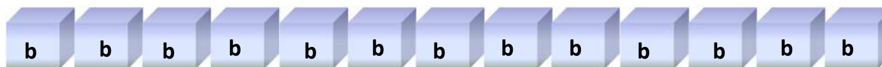
Performance Problem: Partieller Datenzugriff

Verifysoft TECHNOLOGY

Schlechte Cache Line Ausnutzung durch partiellen Zugriff auf eine Struktur:



Verbesserte Performance durch optimierte Cache Line Ausnutzung:



www.verifysoft.com 21

21

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Verifysoft TECHNOLOGY

Überprüfung auf Einhaltung von Programmierrichtlinien / Standards

Foto: www.fotolia.de

www.verifysoft.com 22

22

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Überprüfung auf Einhaltung von Programmierrichtlinien / Standards

Verifysoft TECHNOLOGY

- **MISRA C 2012**
- **MISRA C++ 2008**
- **AUTOSAR C++ Coding Guidelines**
- **JPL**
- **Power of Ten**
- **SEI CERT**
- **Eigene Regeln**

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Überprüfung auf Einhaltung von Programmierrichtlinien / Standards

Verifysoft TECHNOLOGY

Namenskoventionen als Bestandteil von Programmierrichtlinien

Beispiel für früher übliche ungarische Notation:

```
int intCounter;      /* Variable vom Typ int */
long lngMilliTime; /* Variable vom Typ long */
double dblQuote; /* Variable vom Typ double */
```

Heute werden Variablen und Symbole vermehrt nach Clean-Code-Richtlinien benannt.
Beispiel für „sprechenden“ Symbolnamen: upperLimit

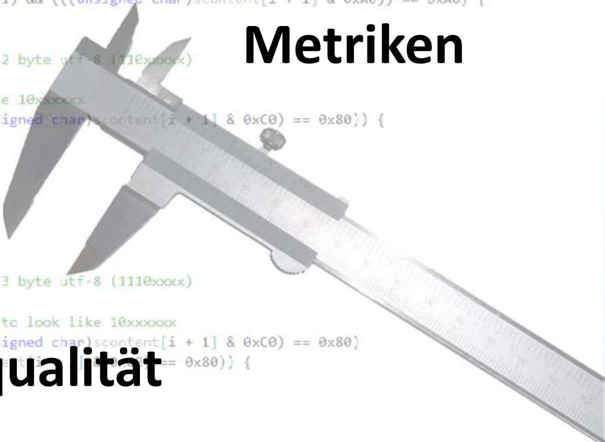
Programmierrichtlinien können zudem Formatierungsvorgaben enthalten.

Beispiel:
Einheitliche Einrückung
Einheitliche Position von {}-Blöcken

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Verifysoft TECHNOLOGY

Metriken



Kann man Softwarequalität messen?

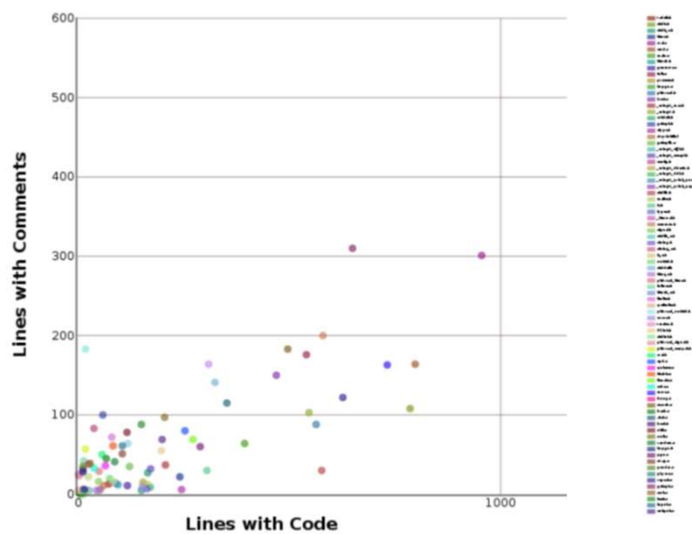
www.verifysoft.com 25

25

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Metriken - Kommentardichte

Verifysoft TECHNOLOGY



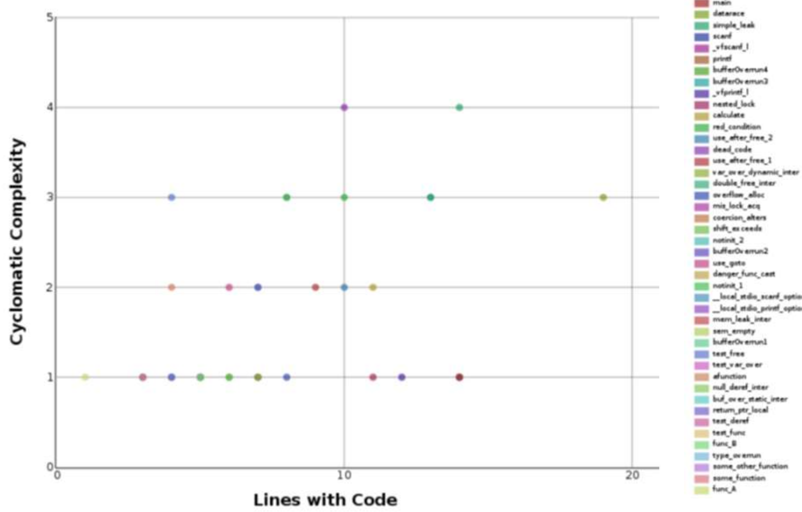
www.verifysoft.com 26

26

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Metriken - Zyklomatische Komplexität

Verifysoft TECHNOLOGY



SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Verifysoft TECHNOLOGY

Spezielle Anwendung:

Statische Analyse von Binärcode

Foto: www.fotolia.de

SAD 2021 Static Analysis Days Online 5.-6. Mai 2021

Aufdeckung von Fehlern und Sicherheitsschwachstellen im Binärcode

Verifysoft TECHNOLOGY

```
544  _text:00000000004014F0    push   rbp
545  _text:00000000004014F1    mov    rbp,rsp
546  _text:00000000004014F4    sub    rsp,0x30
547  _text:00000000004014F8    mov    ecx,5
```

⚠ Event 1: `rcx` is set to 5.
- This determines the capacity of the buffer that will be overrun later.

▼ hide

```
548  _text:00000000004014FD    call   __thunk_malloc
```

⚠ Event 3: `malloc()` returns the address of a new object.
- This points to the buffer that will be overrun later.

▲ ▼ hide

```
549  _text:0000000000401502    mov    qword [rbp+var_8],rax
550  _text:0000000000401506    mov    rax,qword [rbp+var_8]
551  _text:000000000040150A    mov    byte [rax],0x30
552  _text:000000000040150D    mov    rax,qword [rbp+var_8]
553  _text:0000000000401511    add    rax,1
554  _text:0000000000401515    mov    byte [rax],0x31
555  _text:0000000000401518    mov    rax,qword [rbp+var_8]
556  _text:000000000040151C    add    rax,2
557  _text:0000000000401520    mov    byte [rax],0x32
558  _text:0000000000401523    mov    rax,qword [rbp+var_8]
559  _text:0000000000401527    add    rax,3
560  _text:000000000040152B    mov    byte [rax],0x33
561  _text:000000000040152E    mov    rax,qword [rbp+var_8]
562  _text:0000000000401532    add    rax,4
563  _text:0000000000401536    mov    byte [rax],0x34
564  _text:0000000000401539    mov    rax,qword [rbp+var_8]
565  _text:000000000040153D    add    rax,5
566  _text:0000000000401541    mov    byte [rax],0x35
```

Buffer Overrun

This code writes past the end of the buffer pointed to by `rax`.
- `rax` evaluates to `malloc()` a.exe:548 + 5.
- The byte written is at offset 5 from the beginning of the buffer pointed to by `rax`, whose capacity is 5 bytes.
  - The offset exceeds the capacity.
- The overrun occurs in heap memory.

The issue can occur if the highlighted code executes.

See related events 4 and 8.

Show: All events | Only primary events

SAD 2021 Static Analysis Days Online 5.-6. Mai 2021

Komponentenliste (SBOM)

Verifysoft TECHNOLOGY

CODESENTRY customer.discover.grammatech.com

Dashboard
MyProject
MyApp
090120P
090120A
090120P1
090120A1
090920P
090920A

5 Components | 1 Scan | 6 Vulnerabilities

+ NEW SCAN

090920P > 090920A > libxul.so: Scan Complete, Scan Depth: Average. File size: 120.27 MB

Bill of Materials | Vulnerabilities | Scan Tree

Search

Pass/Fail	Component	Version	Match	CVSS Distribution	Artifact
✗	libjpeg	1.4.2	High	2 / 0 / 0 / 0 / 0	libxul.so
✓	libxml	2.9.1	Medium	0 / 0 / 0 / 0 / 0	libxul.so
✓	libvpx	1.7.0	Medium	0 / 0 / 0 / 0 / 0	libxul.so
✗	libxslt	1.1.3	Low	4 / 0 / 0 / 0 / 0	libxul.so
✓	jsoncpp	1.7.2	Low	0 / 0 / 0 / 0 / 0	libxul.so

Items per page: 25 1 – 5 of 5

Settings

https://customer.discover.grammatech.com

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Ausweisen bekannter Sicherheitsschwachstellen

Verifysoft TECHNOLOGY

CODESENTRY customer.discover.grammatech.com

Dashboard
MyProject
MyApp
090120P
090120A
090120P1
090120A1
090920P
090920A
MarksTest
puttygen
MarkProject2
QtSGUI
Settings

5 Components | 1 Scan | 6 Vulnerabilities | + NEW SCAN

090920P > 090920A > libxul.so: Scan Complete, Scan Depth: Average, File size: 120.27 MB

Bill of Materials | Vulnerabilities | Scan Tree

Filter by severity: Severity: All | Search

Severity	Component	Version	Match	Vulnerability
⊖	libjpeg	1.4.2	High	CVE-2020-14152
⊖	libjpeg	1.4.2	High	CVE-2020-14153
⊖	libxslt	1.1.3	Low	CVE-2012-6139
⊖	libxslt	1.1.3	Low	CVE-2013-4520
⊖	libxslt	1.1.3	Low	CVE-2019-11068
⊖	libxslt	1.1.3	Low	CVE-2019-5815

Items per page: 25 | 1 – 6 of 6

SAD 2021
Static Analysis Days
Online 5.-6. Mai 2021

Verifysoft TECHNOLOGY

Moderne Werkzeuge zur statischen Codeanalyse können eine große Bandbreite von Überprüfungen wie z.B.

- Überprüfung auf syntaktische Richtigkeit
- Überprüfung auf semantische Sinnhaftigkeit
- Überprüfung auf funktionale Sicherheit
- Überprüfung auf Sicherheitsschwachstellen
- Überprüfung auf Wartbarkeit

durchführen und damit die Softwarequalität entscheidend verbessern. Ihr frühzeitiger Einsatz zu Beginn der Implementierungsphase verringert Risiken und senkt Kosten.

Verifysoft TECHNOLOGY

VIELEN DANK !

Royd Lüdtke
Verifysoft Technology GmbH
luedtke@verifysoft.com
+49 781 127 8118-8