Recommending security fixes for weak open-source code with AI

Moritz Mock

Jorge Melegati

Barbara Russo

10.11.2023 – 23th SFScon

Agenda

unibz

- Problem statement
- Background
- Existing approach
- Generation of a dataset
- Proposed solution with CodeBERT

Problem statement

unibz

- Developers release code that is not perfect
  - they admit it is not perfect, but they do not know that it could also not be secure

Problem statement

unibz

- Developers release code that is not perfect
  - they admit it is not perfect, but they do not know that it could also not be secure

- Propose a solution that detects not-perfect code that works, but it is not secure (weak)

Background

- Self-admitted technical debt (SATD)
- Vulnerabilities / Common Weakness Enumeration (CWE)
- ML for code detection - CodeBERT

```
1  /* FIXME: this code assumes that sigmask is an even multiple of
       the size of a long integer. */
2
3  unsigned long *src = (unsigned long const *) set;
4  unsigned long *dest = (unsigned long *) &(thread.p->sigmask);
5
6  switch (how)
7  {
8      case SIG_BLOCK:
9      for (i = 0; i < (sizeof (sigset_t) / sizeof (unsigned long)
           ); i++)
10     {
11       /* OR the bit field longword-wise. */
12       *dest++ |= *src++;
13     }
14     break;
15     case SIG_UNBLOCK:
16     for (i = 0; i < (sizeof (sigset_t) / sizeof (unsigned long)
           ); i++)
17     {
18       /* XOR the bitfield longword-wise. */
19       *dest++ ^= *src++;
20     }
21     case SIG_SETMASK:
22     /* Replace the whole sigmask. */
23     memcpy (&(thread.p->sigmask), set, sizeof (sigset_t));
24     break;
25 }
```

Background

- Self-admitted technical debt (SATD)
- Vulnerabilities / Common Weakness Enumeration (CWE)
- ML for code detection - CodeBERT

```
1  /* FIXME: this code assumes that sigmask is an even multiple of
      the size of a long integer. */
2
3  unsigned long *src = (unsigned long const *) set;
4  unsigned long *dest = (unsigned long *) &(thread.p->sigmask);
5
6  switch (how)
7  {
8      case SIG_BLOCK:
9      for (i = 0; i < (sizeof (sigset_t) / sizeof (unsigned long)
          ); i++)
10     {
11       /* OR the bit field longword-wise. */
12       *dest++ |= *src++;
13     }
14     break;
15     case SIG_UNBLOCK:
16     for (i = 0; i < (sizeof (sigset_t) / sizeof (unsigned long)
          ); i++)
17     {
18       /* XOR the bitfield longword-wise. */
19       *dest++ ^= *src++;
20     }
21     case SIG_SETMASK:
22     /* Replace the whole sigmask. */
23     memcpy (&(thread.p->sigmask), set, sizeof (sigset_t));
24     break;
25 }
```

Existing approach

WeakSATD

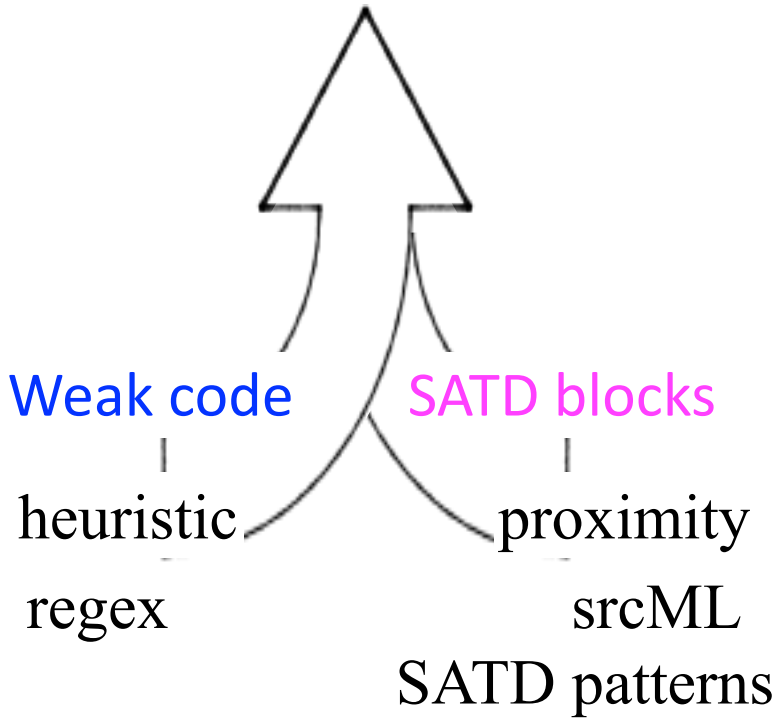
unibz

Common Weakness Enumeration

Title	**CWE-242: Use of Inherently Dangerous Function**
Description	The program calls a function that can never be guaranteed to work safely
Extended Description	Certain functions behave in dangerous ways regardless of how they are used. ... The gets() function is unsafe because does not perform bounds checking on the size of its input. An attacker can easily send arbitrarily-sized input to gets() and overflow the destination buffer. ...
Relationships	Relevant to the view "Software Development" (CWE-699) *Nature* *Type* *ID* *Name* MemberOf C 1228 API/Function Errors
Modes of Introduction:	Phase: Implementation
Applicable Platform	Languages: C, C++
Common Consequences	Technical Impact: Varies by *Context*
Likelihood Of Exploit	High

Demonstrative Examples	The code below calls gets() to read information into a buffer.
	Example. Language: C
	`char buf[BUFSIZE];`
	`gets(buf);`

Potential mitigations	Phases: Implementation; Requirements ban the use of dangerous functions. Use their safe equivalent. Phase: Testing; Use grep or static analysis tools to spot usage of dangerous functions.

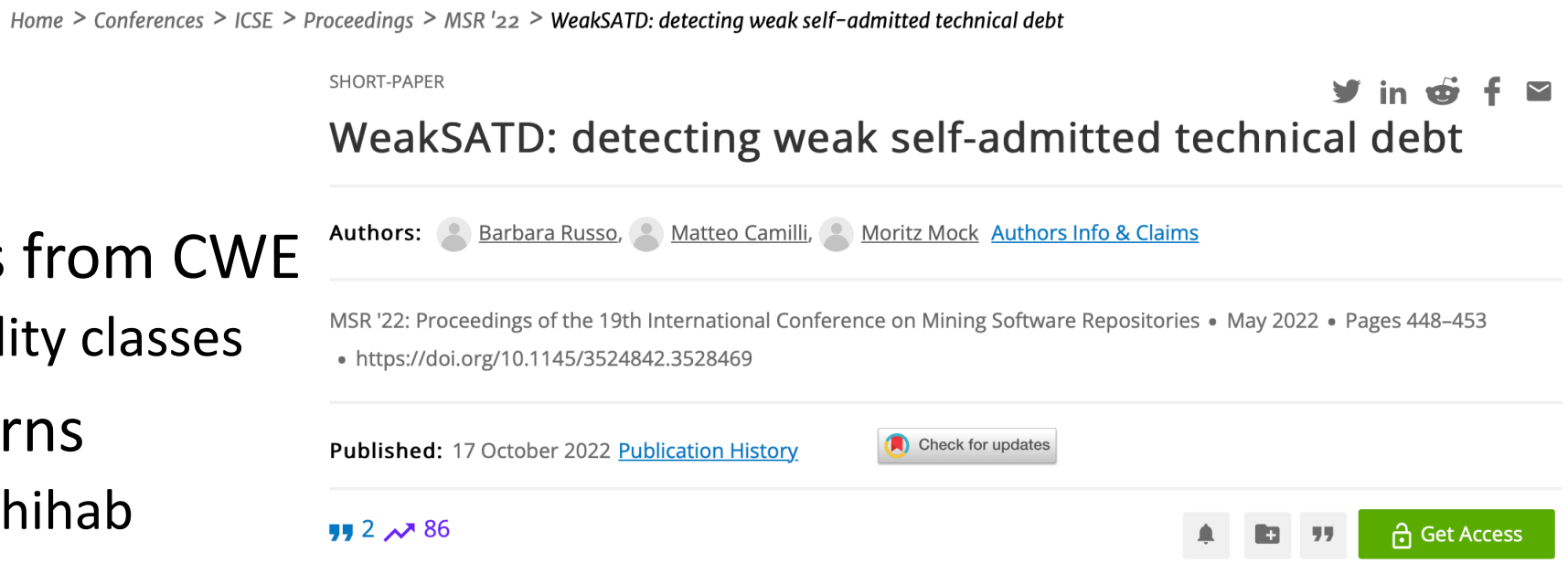


Self-admitted technical debt


```
1  /* FIXME: this code assumes that sigmask is an
       even multiple of the size of a long
       integer. */
2
3  unsigned long *src = (unsigned long const *)
       set;
4  unsigned long *dest = (unsigned long *) &(
       thread.p->sigmask);
5
6  switch (how)
7  {
8      case SIG_BLOCK:
9      for (i = 0; i < (sizeof (sigset_t) /
           sizeof (unsigned long)); i++)
10     {
11       /* OR the bit field longword-wise. */
12       *dest++ |= *src++;
13     }
14     break;
15     case SIG_UNBLOCK:
16     for (i = 0; i < (sizeof (sigset_t) /
           sizeof (unsigned long)); i++)
17     {
18       /* XOR the bitfield longword-wise. */
19       *dest++ ^= *src++;
20     }
21     case SIG_SETMASK:
22     /* Replace the whole sigmask. */
23     memcpy (&(thread.p->sigmask), set, sizeof
           (sigset_t));
24     break;
25 }
```

WeakSATD (Russo et al., 2022)

- Vulnerabilities from CWE
  - 34 vulnerability classes
- 61 SATD patterns
  - Potdar and Shihab
- Line-level
- Leveraging on Chromium



Home > Conferences > ICSE > Proceedings > MSR '22 > WeakSATD: detecting weak self-admitted technical debt

SHORT-PAPER

WeakSATD: detecting weak self-admitted technical debt

Authors: Barbara Russo, Matteo Camilli, Moritz Mock Authors Info & Claims

MSR '22: Proceedings of the 19th International Conference on Mining Software Repositories • May 2022 • Pages 448–453
• https://doi.org/10.1145/3524842.3528469

Published: 17 October 2022 Publication History

Check for updates

2 86

Get Access

WeakSATD: highlights

unibz

File:test.c — Filelength: 26

```
1.  /* FIXME: this code assumes that sigmask is an even multiple of the size of a long integer. */
2.
3.  unsigned long *src = (unsigned long const *) set;
4.  unsigned long *dest = (unsigned long *) &(thread.p->sigmask);
5.
6.  switch (how)
7.  {
8.      case SIG_BLOCK:
9.      for (i = 0; i < (sizeof (sigset_t) / sizeof (unsigned long)); i++)
10.     {
11.       /* OR the bit field longword-wise. */
12.       *dest++ |= *src++;
13.     }
14.     break;
15.     case SIG_UNBLOCK:
16.     for (i = 0; i < (sizeof (sigset_t) / sizeof (unsigned long)); i++)
17.     {
18.       /* XOR the bitfield longword-wise. */
19.       *dest++ ^= *src++;
20.     }
21.     case SIG_SETMASK:
22.     /* Replace the whole sigmask. */
23.     memcpy (&(thread.p->sigmask), set, sizeof (sigset_t));
24.     break;
25. }
26.
```

WeakSATD: recommendation

- Leveraging CWE recommendation

- Easy to understand

- Widely accepted by practitioners

3. CWE-Issue-Number 484
Implementation:

Omitting a break statement so that one may fall through is often indisti
have clearly documented this within the switch statement, and ensure t
Implementation:

The functionality of omitting a break statement could be clarified with

In the following lines (switch statements) where found one or more err

Generation of a dataset

Generation of a dataset

unibz

- Tangled changes
- No dataset annotated for:
  - SATD comments
  - Vulnerabilities

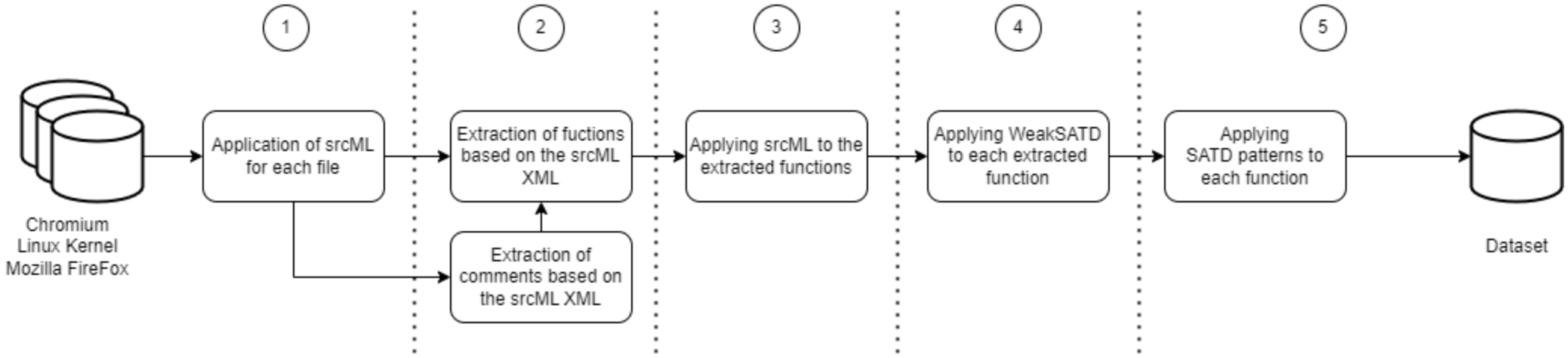
Generation of a dataset

unibz

- Tangled changes
- No dataset annotated for:
  - SATD comments
  - Vulnerabilities

- Function level
- Annotation for SATD and Vulnerabilities
- Based on Chromium, Mozilla Firefox, and Linux Kernel
  - Extension from our dataset on Chromium

Process of dataset generation



Obtained datasets (functions)

unibz

Classes	Chromium	Linux Kernel	Firefox	Total
SATD and Vuln.	134	4808	104	5046
SATD and no Vuln.	17	787	12	816
no SATD and Vuln.	9790	393937	7606	411333
no SATD and no Vuln.	10121	253270	7665	271056
Total	20062	652802	15387	688251

Obtained datasets (functions)

unibz

Classes	Chromium	Linux Kernel	Firefox	Total
SATD and Vuln.	134	4808	104	5046
SATD and no Vuln.	17	787	12	816
no SATD and Vuln.	9790	393937	7606	411333
no SATD and no Vuln.	10121	253270	7665	271056
Total	20062	652802	15387	688251

Proposed solution with CodeBERT

CodeBERT

unibz

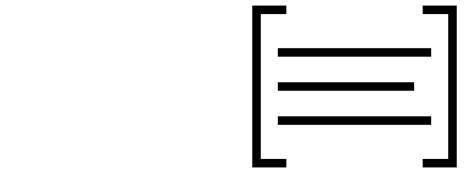
- Pre-trained model on a dataset with over 8 million data points.
- Based on the SOTA architecture BERT
- CodeBERT has been used in source code related tasks

WeakSATD

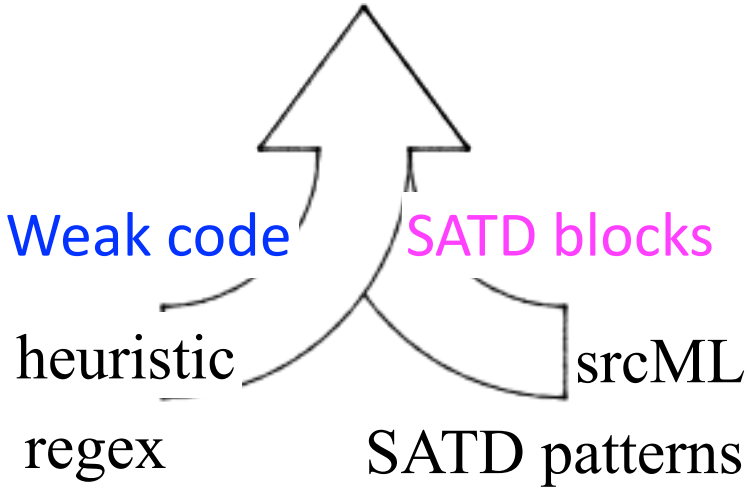
unibz

Common Weakness Enumeration

Title	**CWE-242: Use of Inherently Dangerous Function**
Description	The program calls a function that can never be guaranteed to work safely
Extended Description	Certain functions behave in dangerous ways regardless of how they are used. ... The gets() function is unsafe because does not perform bounds checking on the size of its input. An attacker can easily send arbitrarily-sized input to gets() and overflow the destination buffer. ...
Relationships	Relevant to the view "Software Development" (CWE-699)
	Nature *Type* *ID* *Name* MemberOf C 1228 API/Function Errors
Modes of Introduction:	Phase: Implementation
Applicable Platform	Languages: C, C++
Common Consequences	Technical Impact: Varies by *Context*
Likelihood Of Exploit	High
Demonstrative Examples	The code below calls gets() to read information into a buffer. Example. Language: C
	`char buf[BUFSIZE];` `gets(buf);`
Potential mitigations	Phases: Implementation; Requirements ban the use of dangerous functions. Use their safe equivalent. Phase: Testing; Use grep or static analysis tools to spot usage of dangerous functions.



Machine Learner w. CodeBERT



Self-admitted technical debt


```c
/* FIXME: this code assumes that sigmask is an
      even multiple of the size of a long
      integer. */

unsigned long *src = (unsigned long const *)
    set;
unsigned long *dest = (unsigned long *) &(
    thread.p->sigmask);

switch (how)
{
    case SIG_BLOCK:
    for (i = 0; i < (sizeof (sigset_t) /
        sizeof (unsigned long)); i++)
    {
      /* OR the bit field longword-wise. */
      *dest++ |= *src++;
    }
    break;
    case SIG_UNBLOCK:
    for (i = 0; i < (sizeof (sigset_t) /
        sizeof (unsigned long)); i++)
    {
      /* XOR the bitfield longword-wise. */
      *dest++ ^= *src++;
    }
    case SIG_SETMASK:
    /* Replace the whole sigmask. */
    memcpy (&(thread.p->sigmask), set, sizeof
        (sigset_t));
    break;
}
```

CodeBERT - Results

unibz

	Accuracy	Precision	Recall	F1	Step
Single-task SATD	97.61%	97.53%	94.23%	95.85	train
Single-task SATD	95.99%	97.35%	94.23%	95.76	validation
Single-task SATD	92.59%	90.75%	95.15%	92.89	prediction
Single-task Vulnerability	99.61%	99.55%	99.7%	99.62	train
Single-task Vulnerability	94.44%	93.15%	94.44%	93.79	validation
Single-task Vulnerability	91.35%	91.72%	89.26%	90.47	prediction
Single-task WeakSATD	98.15%	97.74%	98.63%	98.18	train
Single-task WeakSATD	88.27%	84.52%	92.21%	88.19	validation
Single-task WeakSATD	90.12%	90.54%	88.15%	89.3	prediction
Multitask SATD	100%	100%	100%	100	train
Multitask SATD	94.75%	93.17%	96.15%	94.63	validation
Multitask SATD	94.44%	92.48%	96.96%	94.67	prediction
Multitask Vulnerability	100%	100%	100%	100	train
Multitask Vulnerability	94.14%	93.1%	93.75%	93.42	validation
Multitask Vulnerability	91.04%	88.96%	90.84%	89.89	prediction
Multitask WeakSATD	87.96%	72.5%	77.3%	74.83	prediction

CodeBERT - Results

unibz

	Accuracy	Precision	Recall	F1	Step
Single-task SATD	97.61%	97.53%	94.23%	95.85	train
Single-task SATD	95.99%	97.35%	94.23%	95.76	validation
Single-task SATD	92.59%	90.75%	95.15%	92.89	prediction
Single-task Vulnerability	99.61%	99.55%	99.7%	99.62	train
Single-task Vulnerability	94.44%	93.15%	94.44%	93.79	validation
Single-task Vulnerability	91.35%	91.72%	89.26%	90.47	prediction
Single-task WeakSATD	98.15%	97.74%	98.63%	98.18	train
Single-task WeakSATD	88.27%	84.52%	92.21%	88.19	validation
Single-task WeakSATD	90.12%	90.54%	88.15%	89.3	prediction
Multitask SATD	100%	100%	100%	100	train
Multitask SATD	94.75%	93.17%	96.15%	94.63	validation
Multitask SATD	94.44%	92.48%	96.96%	94.67	prediction
Multitask Vulnerability	100%	100%	100%	100	train
Multitask Vulnerability	94.14%	93.1%	93.75%	93.42	validation
Multitask Vulnerability	91.04%	88.96%	90.84%	89.89	prediction
Multitask WeakSATD	87.96%	72.5%	77.3%	74.83	prediction

CodeBERT: predicted as vulnerable (TP)

- CVE-2016-1621
- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer

```
long long Block::GetTimeCode(const Cluster* pCluster) const
{
    if (pCluster == 0)
        return m_timecode;

    const long long tc0 = pCluster->GetTimeCode();
    assert(tc0 >= 0);

    const long long tc = tc0 + m_timecode;
    return tc;  //unscaled timecode units
}
```

Conclusion

unibz

- SATD blocks contains vulnerabilities (Russo et. Al 2022)
- SATD should be remove fast (Bavota and Russo 2016)
- Creating awareness for developers
- Creation of a more usable tool for practitioners

unibz

Thank you for your attention