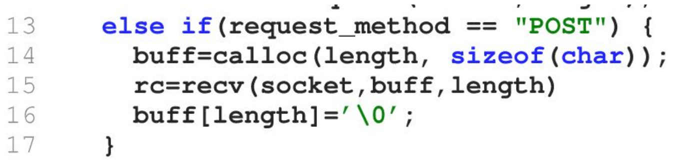
Automated Patching

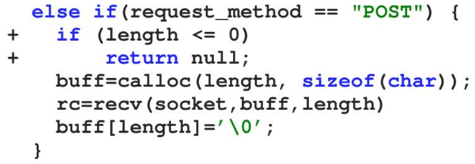
Holistic Software Security

Aravind Machiry

Fixing code automatically!



```
else if(request_method == "POST") {
  buff=calloc(length, sizeof(char));
  rc=recv(socket,buff,length)
  buff[length]='\0';
}
```



```
  else if(request_method == "POST") {
+   if (length <= 0)
+       return null;
    buff=calloc(length, sizeof(char));
    rc=recv(socket,buff,length)
    buff[length]='\0';
  }
```

What is it?

Patching a defect (bug or vulnerability) automatically, also known as Automated Program Repair:

- Where and how to fix?
- How to specify the defect?

What is it?

Patching a defect (bug or vulnerability) automatically:

- Where and how to fix? => **On source code, by making source level changes (i.e., editing code statements).**
- How to specify the defect? => **Failing Test cases.**

What is it?

Patching a defect (bug or vulnerability) automatically:

- Where and how to fix? => **On source code, by making source level changes (i.e., editing code statements).**
  - Alternatives:
    - On binaries by doing binary rewriting.
    - Runtime by avoiding error behavior (error recovery).
- How to specify the defect? => **Failing Test cases.**
  - Alternatives:
    - High level specification: All memory errors.

What is it?

Patching a defect (bug or vulnerability) automatically:

- Where and how to fix? => **On source code, by making source level changes (i.e., editing code statements).**
  - Alternatives:
    - ~~On binaries by doing binary rewriting.~~
    - ~~Runtime by avoiding error behavior (error recovery).~~
- How to specify the defect? => **Failing Test cases.**
  - Alternatives:
    - ~~High level specification: All memory errors.~~

Not in this course.

Clarifications

Bug => Root cause and Symptom.

- Root cause => Uninitialized variable, out of bounds access, etc.
  - Fixing Root cause => Program Repair or Automated Patching.

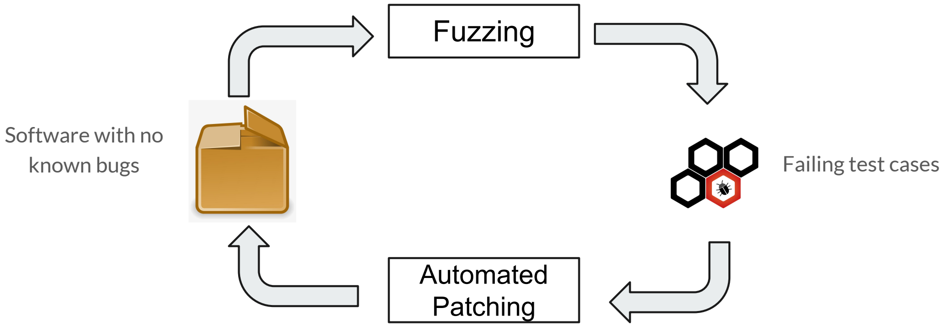
- Symptom => SIGSEGV, Failing test case, etc.
  - Fixing Symptom => Error recovery.

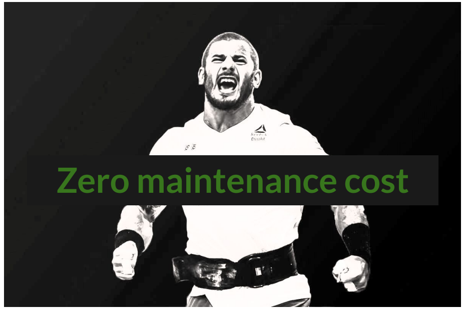
Why is it needed: Automated and continuous software maintenance.

google/**clusterfuzz**

Scalable fuzzing infrastructure.

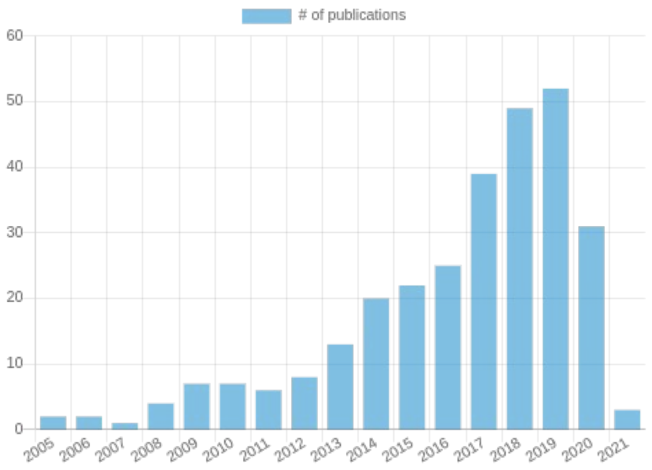
43 Contributors · 2 Used by · 5k Stars · 446 Forks



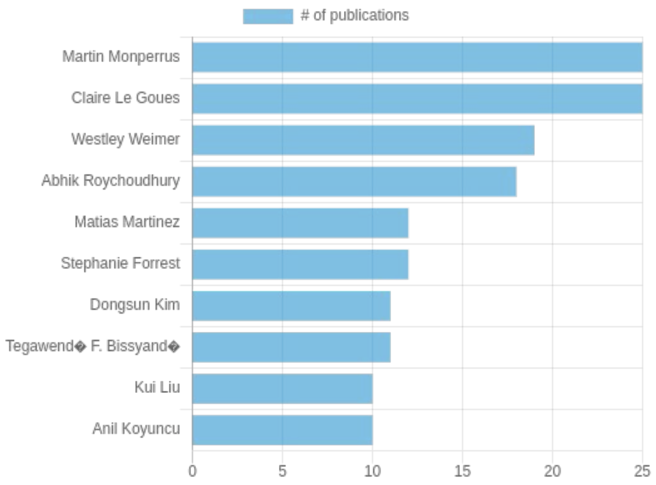


“What one would like ideally [...] is the automatic detection and correction of bugs” R. J. Abbott, 1990

Very active research area



Leaders



https://program-repair.org/index.html

Approaches: Overview

- Genetic Programming: GenProg and family.
- Program Analysis: Senx, Talos, SAVER, SPR, etc.
- Machine Learning: Prophet, DeepFix, etc.

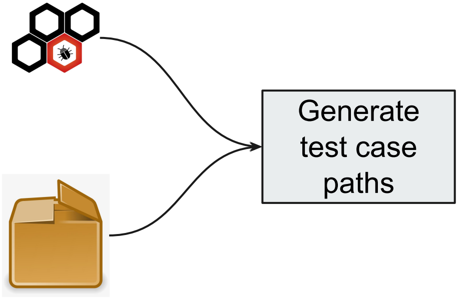
Approaches: Overview

- Genetic Programming: **GenProg** and family.
- Program Analysis: Senx, **Talos**, **SAVER**, SPR, etc.
- Machine Learning: Prophet, DeepFix, etc.

GenProg: Fixing by genetic programming

- Intuition: "The fix for a bug is most likely already present somewhere in the program."
- The developer might have written mostly bug-free code except for a few cases where the bug might have crept in.

GenProg: Generate paths

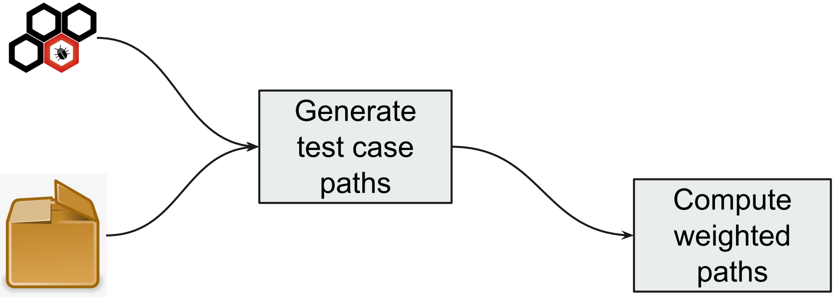


GenProg: Test case paths

For each test case:

- Get the path, i.e., sequence of statements executed.
- Remove duplicate statements, i.e., statements in loops.

GenProg: Weighted paths

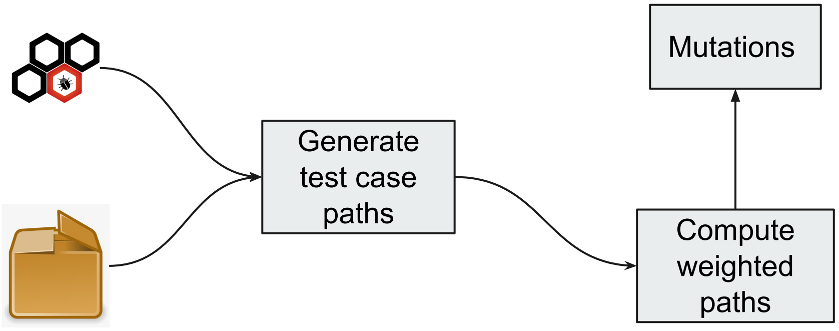


GenProg: Weighted paths

For each path:

- Assign a weight for each statement:
    - Statement executed only in failure test case, Weight = 1.
    - Statement executed in successful test case, Weight = 0.01.

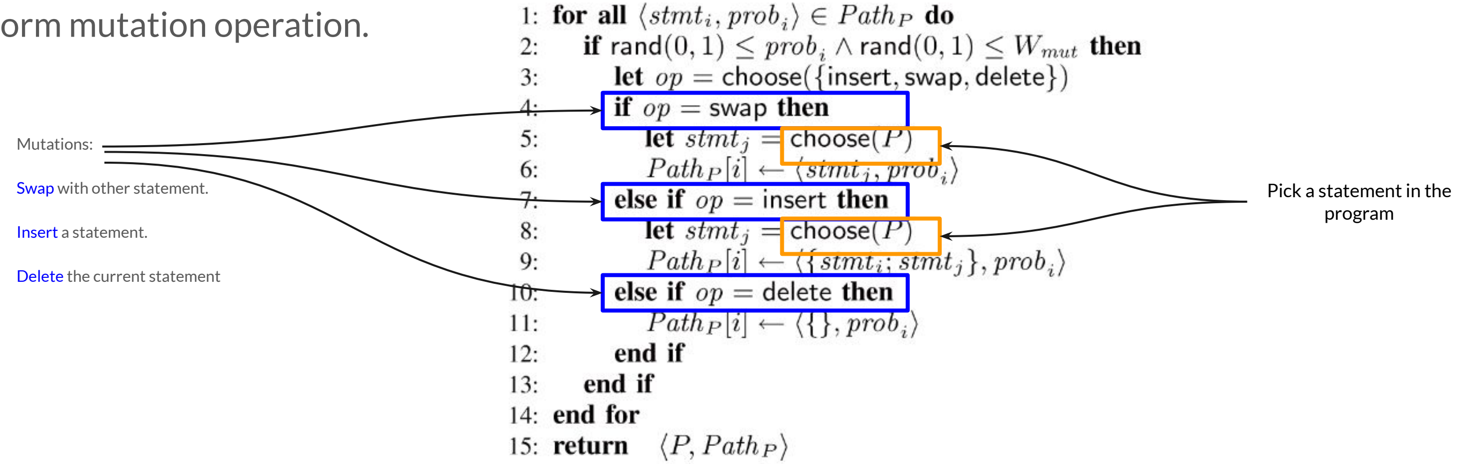
GenProg: Mutations



GenProg: Mutations

For each path:

- Pick a statement: Higher weight => Higher probability of picking.
- Perform mutation operation.



```
for all $\langle stmt_i, prob_i \rangle \in Path_P$ do
    if rand(0, 1) $\leq prob_i \land$ rand(0, 1) $\leq W_{mut}$ then
        let $op$ = choose({insert, swap, delete})
        if $op$ = swap then
            let $stmt_j$ = choose($P$)
            $Path_P[i] \leftarrow \langle stmt_j, prob_i \rangle$
        else if $op$ = insert then
            let $stmt_j$ = choose($P$)
            $Path_P[i] \leftarrow \langle \{stmt_i; stmt_j\}, prob_i \rangle$
        else if $op$ = delete then
            $Path_P[i] \leftarrow \langle \{\}, prob_i \rangle$
        end if
    end if
end for
return $\langle P, Path_P \rangle$
```

Mutations:

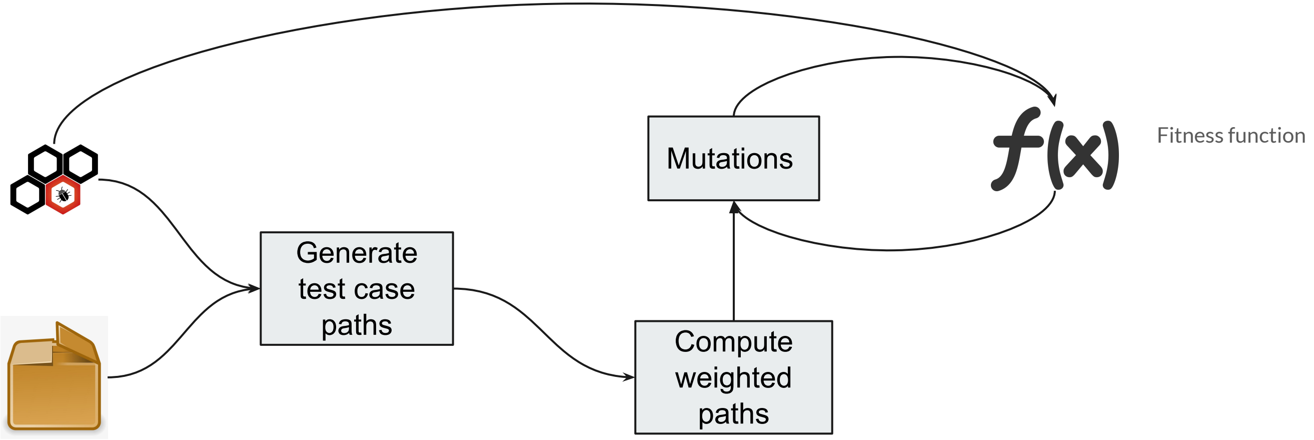
Swap with other statement.

Insert a statement.

Delete the current statement

Pick a statement in the program

GenProg: Fitness function



Generate test case paths

Compute weighted paths

Mutations

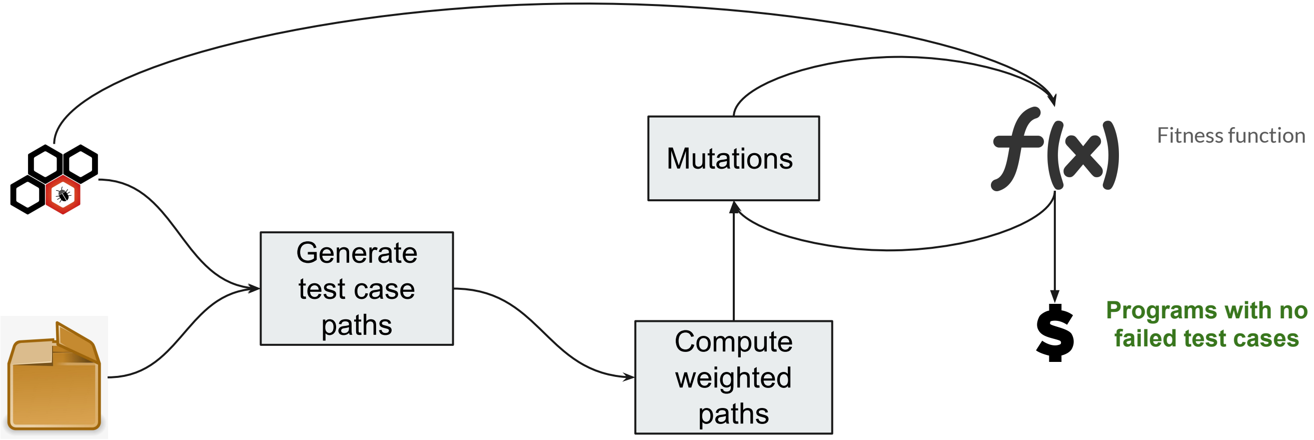
Fitness function

GenProg: Fitness function

Higher score => Passes most of the positive test cases and fails least of the test cases.

$$\begin{aligned}\text{fitness}(P) = {} & W_{PosT} \times |\{t \in PosT \mid P \text{ passes } t\}| \\ & + W_{NegT} \times |\{t \in NegT \mid P \text{ passes } t\}|.\end{aligned}$$

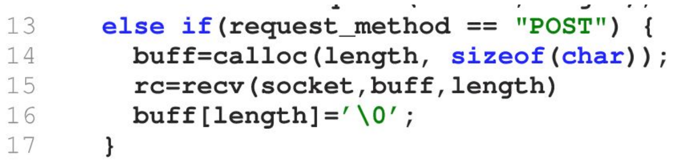
GenProg: Post processing



GenProg: Post processing

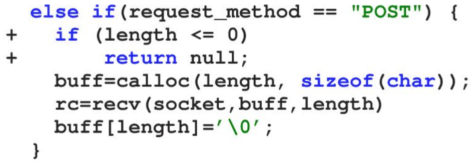
- Minimize the patched program:
  - Delta debugging : Iteratively remove statements unless there is a failed test case.

GenProg: Results



```
else if(request_method == "POST") {
  buff=calloc(length, sizeof(char));
  rc=recv(socket,buff,length)
  buff[length]='\0';
}
```

→



```
  else if(request_method == "POST") {
+   if (length <= 0)
+       return null;
    buff=calloc(length, sizeof(char));
    rc=recv(socket,buff,length)
    buff[length]='\0';
  }
```

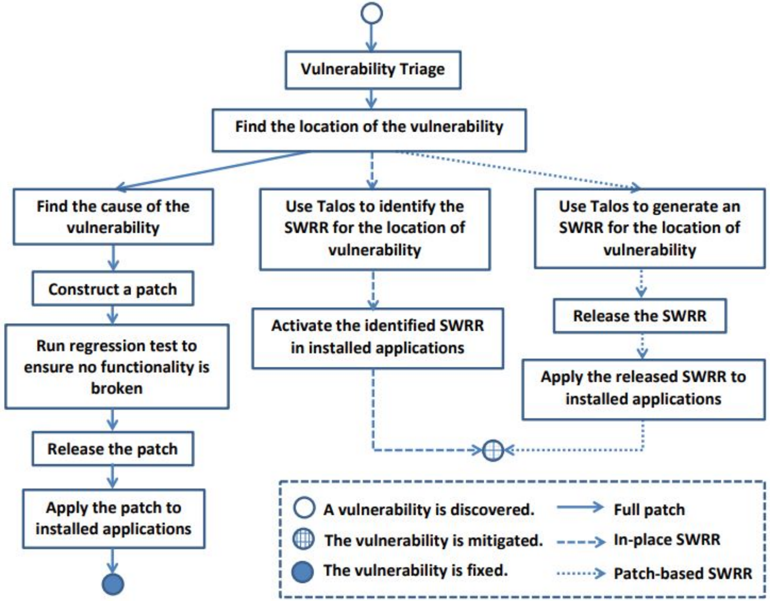
GenProg: Improvements

- Improved search: Randomized Search

Defect Specific Techniques

- Workarounds => Talos: Instead of fixing, avoid the bug
- Buffer overflow, Integer overflow, Bad casts => Senx
- Temporal heap errors => SAVER

Security Workarounds



Vulnerability Triage

Find the location of the vulnerability

Find the cause of the vulnerability

Construct a patch

Run regression test to ensure no functionality is broken

Release the patch

Apply the patch to installed applications

Use Talos to identify the SWRR for the location of vulnerability

Activate the identified SWRR in installed applications

Use Talos to generate an SWRR for the location of vulnerability

Release the SWRR

Apply the released SWRR to installed applications

○ A vulnerability is discovered.

⊕ The vulnerability is mitigated.

● The vulnerability is fixed.

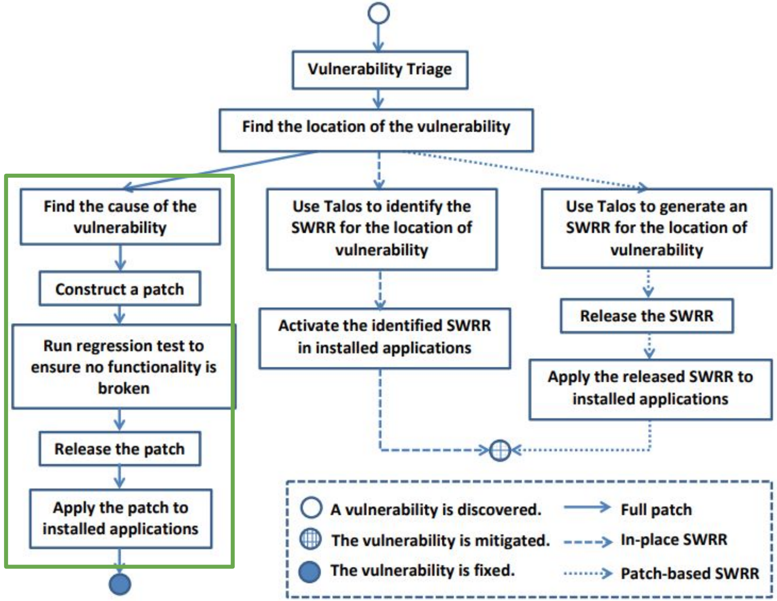
→ Full patch

- - -> In-place SWRR

·······> Patch-based SWRR

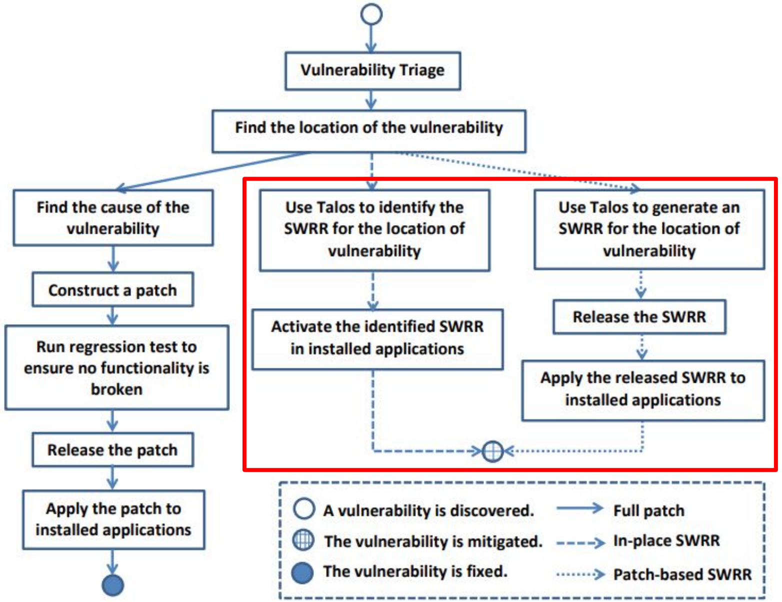
Security Workarounds

Regular flow:
Patching vulnerability



Vulnerability Triage
Find the location of the vulnerability
Find the cause of the vulnerability
Construct a patch
Run regression test to ensure no functionality is broken
Release the patch
Apply the patch to installed applications
Use Talos to identify the SWRR for the location of vulnerability
Activate the identified SWRR in installed applications
Use Talos to generate an SWRR for the location of vulnerability
Release the SWRR
Apply the released SWRR to installed applications
A vulnerability is discovered.
The vulnerability is mitigated.
The vulnerability is fixed.
Full patch
In-place SWRR
Patch-based SWRR

Security Workarounds


Vulnerability Triage
Find the location of the vulnerability
Find the cause of the vulnerability
Construct a patch
Run regression test to ensure no functionality is broken
Release the patch
Apply the patch to installed applications
Use Talos to identify the SWRR for the location of vulnerability
Activate the identified SWRR in installed applications
Use Talos to generate an SWRR for the location of vulnerability
Release the SWRR
Apply the released SWRR to installed applications
A vulnerability is discovered.
The vulnerability is mitigated.
The vulnerability is fixed.
Full patch
In-place SWRR
Patch-based SWRR

Vulnerability Mitigation:
Security Workarounds

Talos: Security Workarounds

Basic Idea: Selectively disable execution of certain (i.e., vulnerable) functions.

Instrument appropriate functions and disable execution of those functions.

Novelty: Correctly disabling functions without affecting “major” functionality of the application.

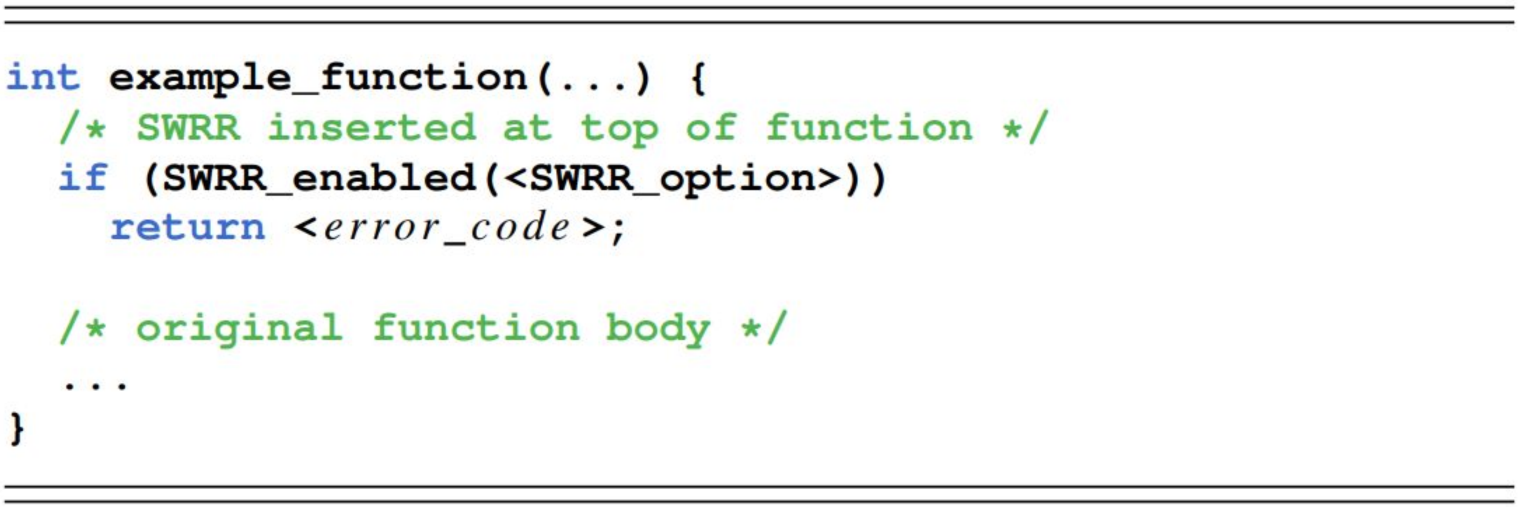
Talos: Disabling functions

Find error handling behavior of each function:

- `return error_code/NULL.`
- `log error message.`
- `Other heuristics.`

Instrument function to have error handling behavior.

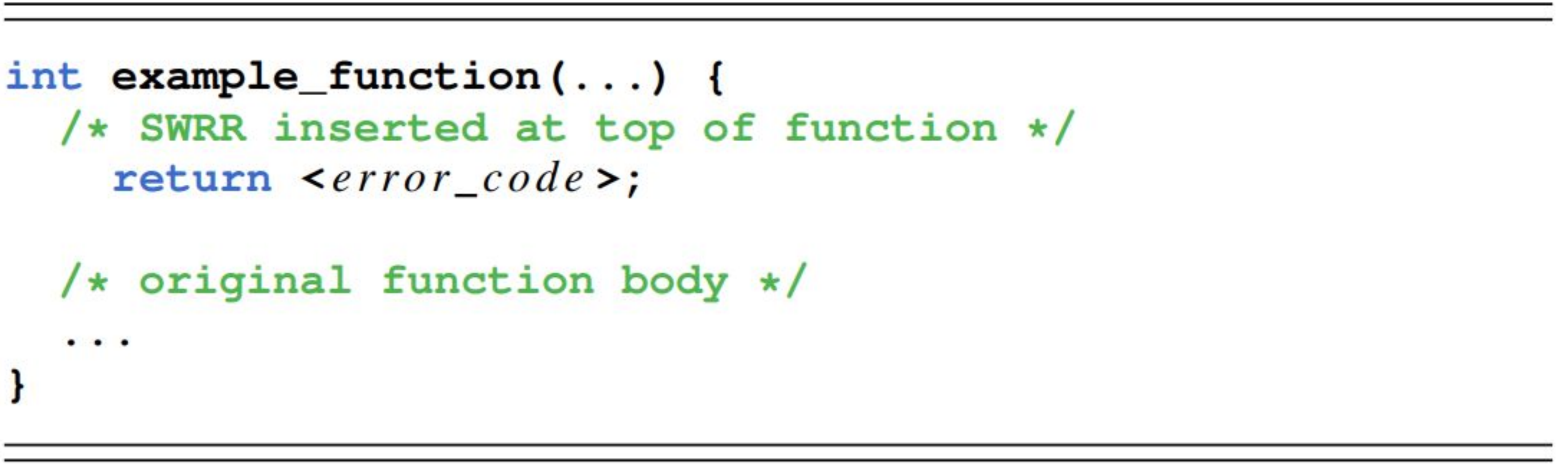
Talos: Disabling functions



```
int example_function(...) {
  /* SWRR inserted at top of function */
  if (SWRR_enabled(<SWRR_option>))
    return <error_code>;

  /* original function body */
  ...
}
```

Talos: Disabling functions

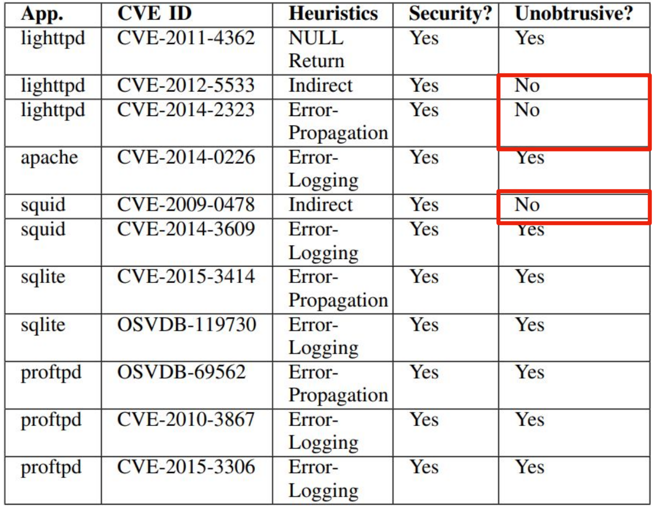


```
int example_function(...) {
  /* SWRR inserted at top of function */
    return <error_code>;

  /* original function body */
  ...
}
```

If the vulnerability is known then just disable the function.

Talos: Results



App.	CVE ID	Heuristics	Security?	Unobtrusive?
lighttpd	CVE-2011-4362	NULL Return	Yes	Yes
lighttpd	CVE-2012-5533	Indirect	Yes	No
lighttpd	CVE-2014-2323	Error-Propagation	Yes	No
apache	CVE-2014-0226	Error-Logging	Yes	Yes
squid	CVE-2009-0478	Indirect	Yes	No
squid	CVE-2014-3609	Error-Logging	Yes	Yes
sqlite	CVE-2015-3414	Error-Propagation	Yes	Yes
sqlite	OSVDB-119730	Error-Logging	Yes	Yes
proftpd	OSVDB-69562	Error-Propagation	Yes	Yes
proftpd	CVE-2010-3867	Error-Logging	Yes	Yes
proftpd	CVE-2015-3306	Error-Logging	Yes	Yes

Affected major functionality of the application

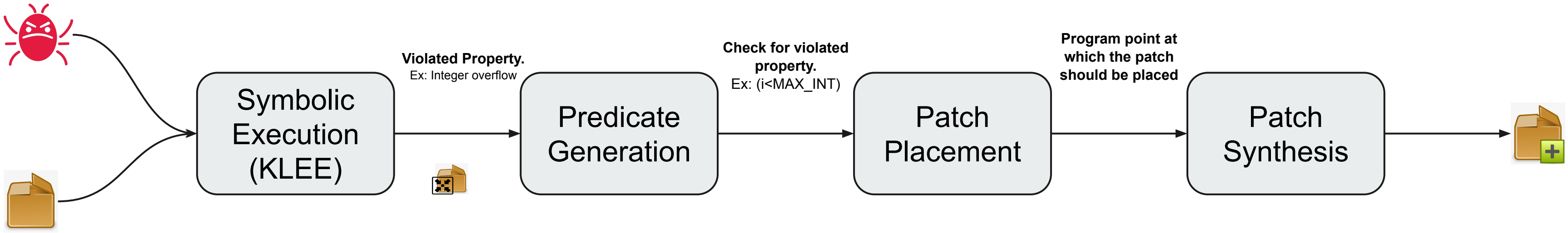
Senx: Vulnerability Specific Patches

Given a vulnerability triggering input => Create a patch that avoids the vulnerability.

Vulnerability types:

- Buffer overflow.
- Bad-cast.
- Integer overflow.

Senx: Overview



Senx: Symbolic Execution

Given program and vulnerability triggering input:

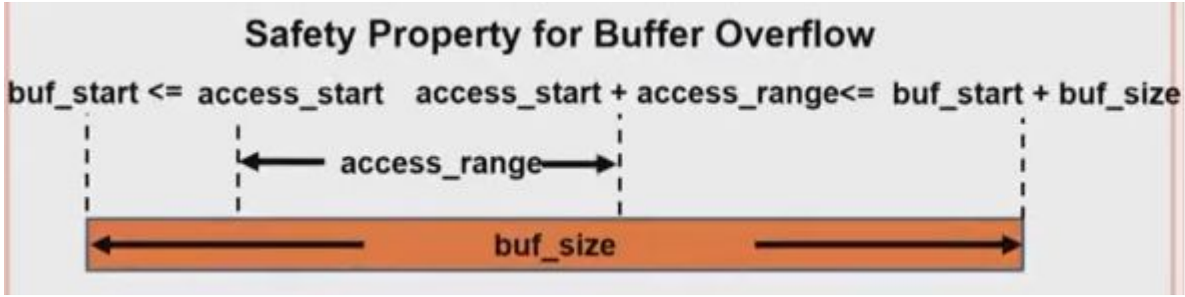
Symbolically trace the program with pre-constraining the input.

At each program point, check for vulnerability condition:

- Out of memory access.
- Integer overflow
- Bad casts.

Vulnerability point: Program point at which the vulnerability condition (security property violation) occurs

Senx: Safety property

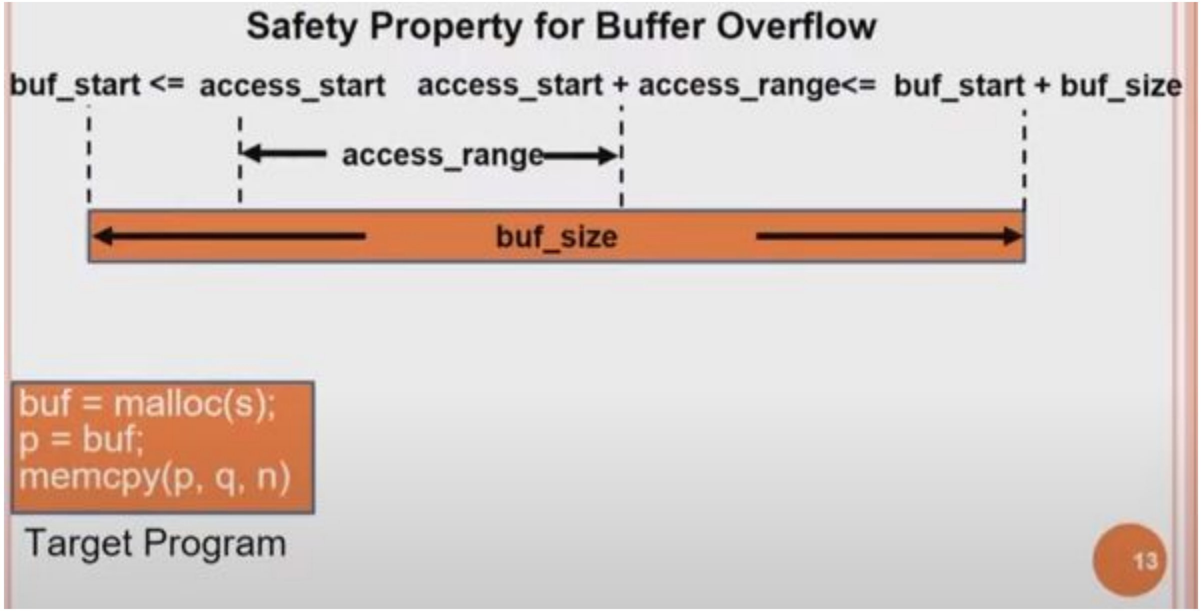

Safety Property for Buffer Overflow
buf_start <= access_start access_start + access_range<= buf_start + buf_size
access_range
buf_size

https://www.youtube.com/watch?v=rHf_teG6iQc

Senx: Predicate Generation

Generate a condition that prevents the vulnerable condition.

Senx: Predicate Generation



Safety Property for Buffer Overflow

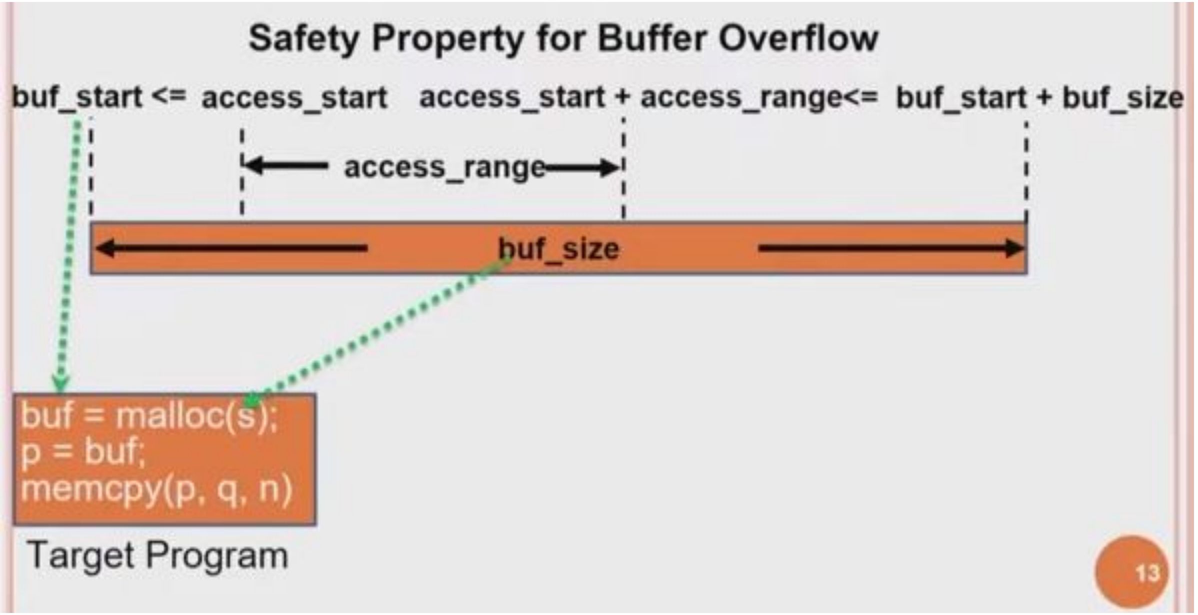
buf_start <= access_start access_start + access_range<= buf_start + buf_size

```
buf = malloc(s);
p = buf;
memcpy(p, q, n)
```

Target Program

https://www.youtube.com/watch?v=rHf_teG6iQc

Senx: Predicate Generation



Safety Property for Buffer Overflow

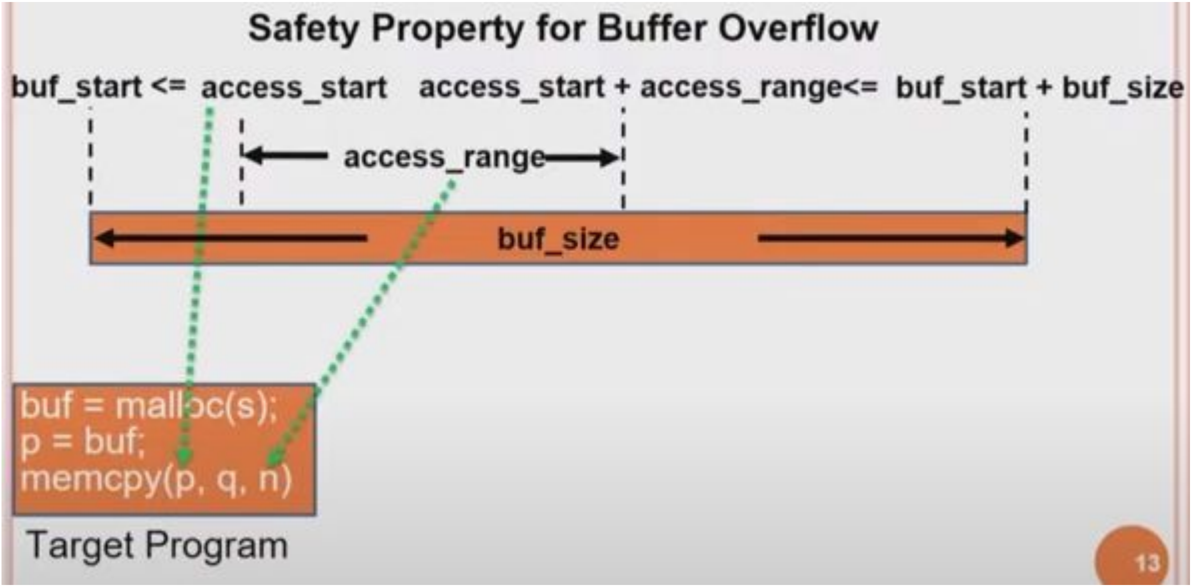
buf_start <= access_start access_start + access_range<= buf_start + buf_size

```
buf = malloc(s);
p = buf;
memcpy(p, q, n)
```

Target Program

https://www.youtube.com/watch?v=rHf_teG6iQc

Senx: Predicate Generation



Safety Property for Buffer Overflow

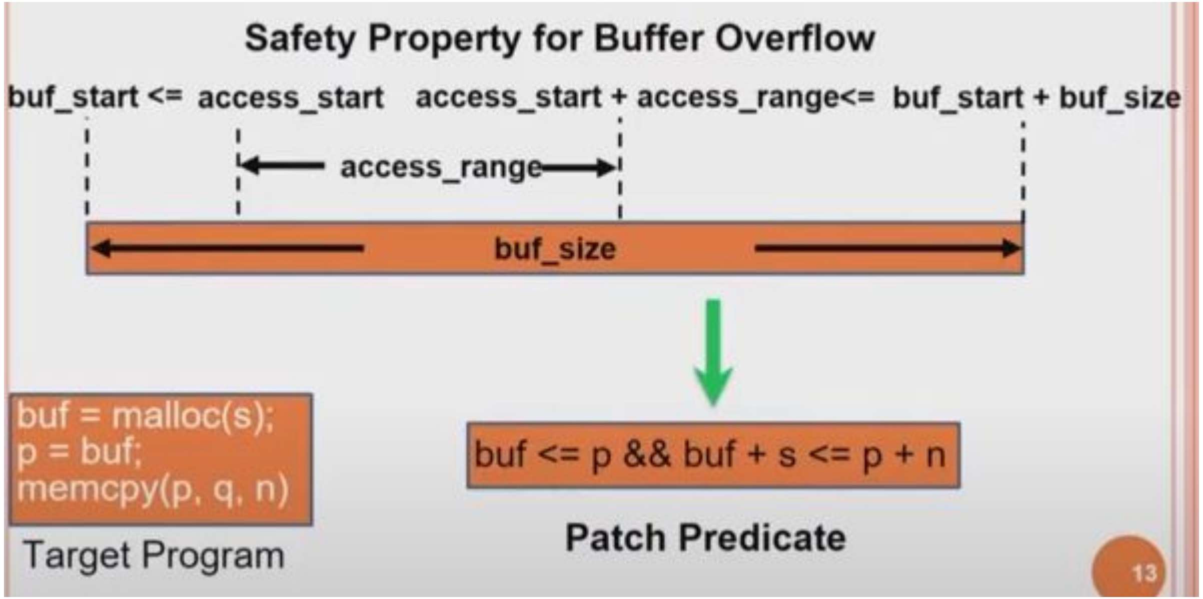
buf_start <= access_start   access_start + access_range<= buf_start + buf_size

```
buf = malloc(s);
p = buf;
memcpy(p, q, n)
```

Target Program

https://www.youtube.com/watch?v=rHf_teG6iQc

Senx: Predicate Generation



Safety Property for Buffer Overflow

buf_start <= access_start access_start + access_range<= buf_start + buf_size
access_range
buf_size

```
buf = malloc(s);
p = buf;
memcpy(p, q, n)
```

Target Program

buf <= p && buf + s <= p + n

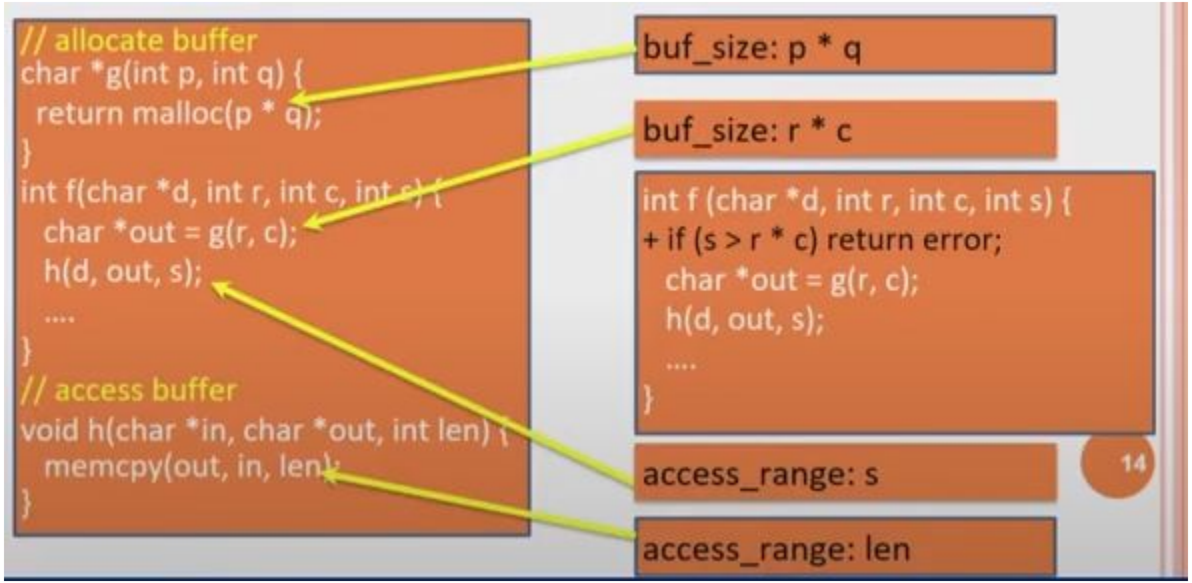
Patch Predicate

https://www.youtube.com/watch?v=rHf_teG6iQc

Senx: Patch Placement

Place the patch at the **highest point in the call-graph** where all the **variables needed for the predicate are available**.

Senx: Patch Placement



```
// allocate buffer
char *g(int p, int q) {
  return malloc(p * q);
}
int f(char *d, int r, int c, int s) {
  char *out = g(r, c);
  h(d, out, s);
  ....
}
// access buffer
void h(char *in, char *out, int len) {
  memcpy(out, in, len);
}
```

buf_size: p * q

buf_size: r * c

```
int f (char *d, int r, int c, int s) {
+ if (s > r * c) return error;
  char *out = g(r, c);
  h(d, out, s);
  ....
}
```

access_range: s

access_range: len

https://www.youtube.com/watch?v=rHf_teG6iQc

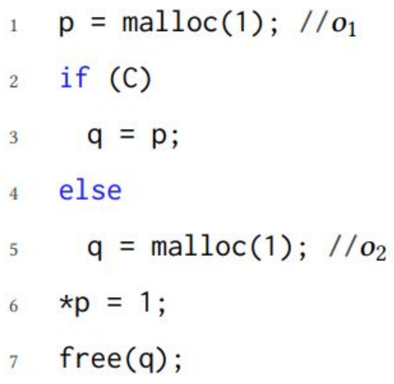
SAVER: Memory Error Repair

Fixes temporal memory errors using static analysis warning.

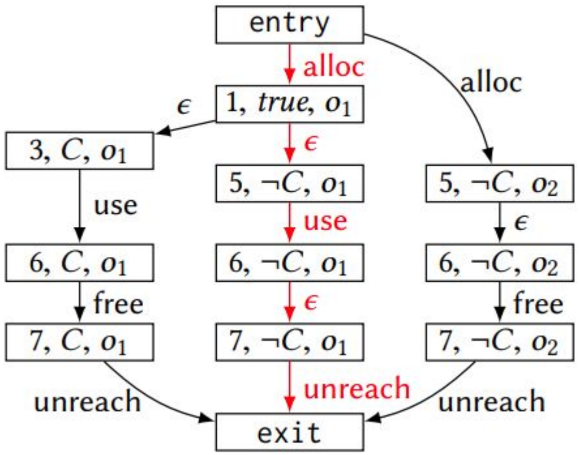
Run Infer (static analysis tool) to find temporal memory errors, i.e., use-after-free, memory leak, double free.

SAVER: Object flow graph

Construct Object flow graph from Infer warning: “*Object allocated at 1 is unreachable at 7*”



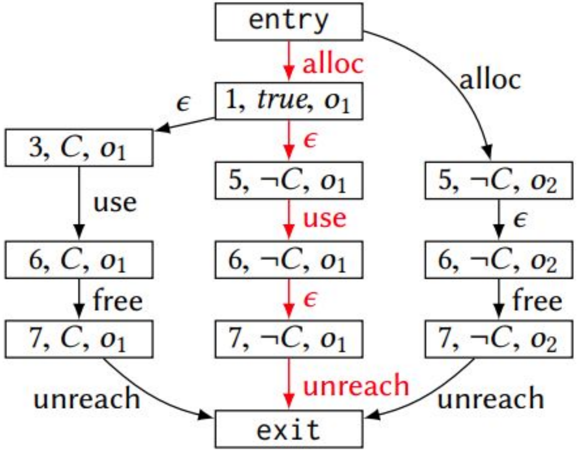
```
1  p = malloc(1); //o₁
2  if (C)
3    q = p;
4  else
5    q = malloc(1); //o₂
6  *p = 1;
7  free(q);
```



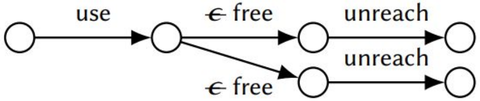
SAVER: Buggy Paths

We need to fix paths containing invalid event sequences by inserting appropriate memory allocation/deallocation operations.

$$\text{alloc} \cdot \epsilon \cdot \text{use} \cdot \epsilon \cdot \text{unreach}$$

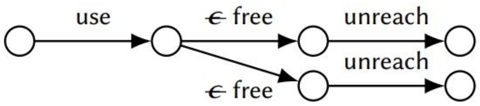


SAVER: Fixing strategies

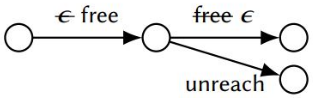


(a) Inserting free

SAVER: Fixing strategies

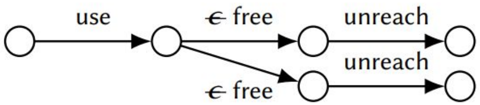


(a) Inserting free

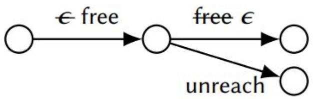


(b) Relocating free

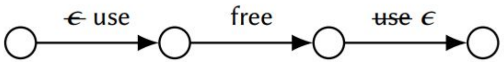
SAVER: Fixing strategies



(a) Inserting free

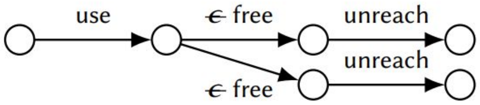


(b) Relocating free

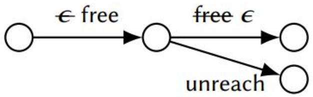


(c) Relocating use (dereference)

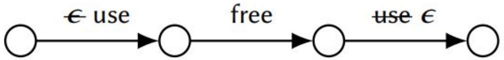
SAVER: Fixing strategies



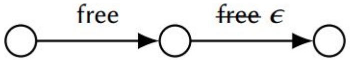
(a) Inserting free



(b) Relocating free

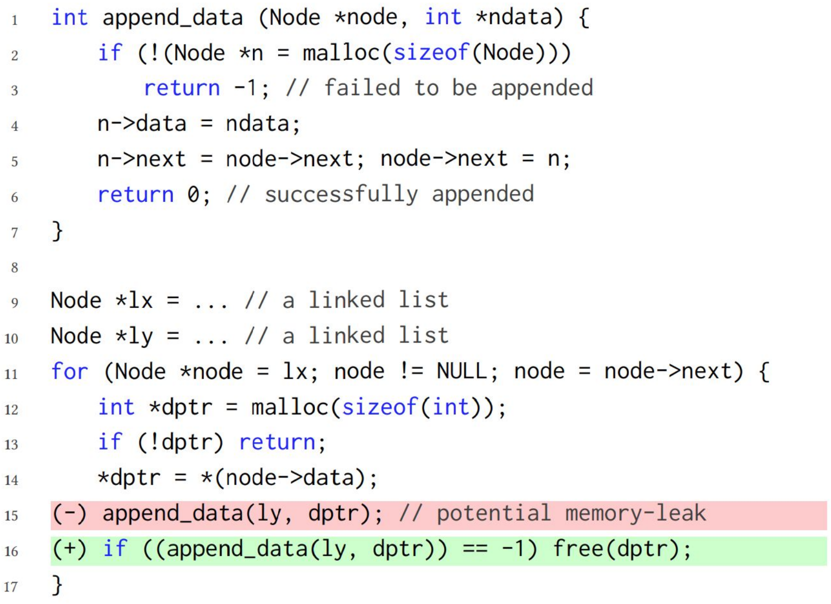


(c) Relocating use (dereference)



(d) Deleting free

SAVER: Results



```
int append_data (Node *node, int *ndata) {
    if (!(Node *n = malloc(sizeof(Node)))
        return -1; // failed to be appended
    n->data = ndata;
    n->next = node->next; node->next = n;
    return 0; // successfully appended
}

Node *lx = ... // a linked list
Node *ly = ... // a linked list
for (Node *node = lx; node != NULL; node = node->next) {
    int *dptr = malloc(sizeof(int));
    if (!dptr) return;
    *dptr = *(node->data);
(-) append_data(ly, dptr); // potential memory-leak
(+) if ((append_data(ly, dptr)) == -1) free(dptr);
}
```

Automated Patching: Final Thoughts

- Defect specific techniques and ML techniques are on rise.
- Should explore interactive patching strategies => Active learning for patching strategies!!?
- Can we ask developer for some input that would make the patching easier and more precise!?
- Keep an eye on: https://program-repair.org/index.html