

Semantic Crash Bucketing

Rijnard van Tonder, John Kotheimer, and
Claire Le Goues



The Problem: Duplicate Crashes

- Large-scale automated testing
 - Fuzzing
 - Symbolic execution

The Problem: Duplicate Crashes

- Large-scale automated testing
 - Fuzzing
 - Symbolic execution
- Heuristic deduplication techniques
 - Call stack hash
 - Branch sequence

Example: SQLite Bug

```
select e.*,0 from(s,(L))e;
```

Example: SQLite Bug

```
select e.*,0from(s,(L))e;
```

Example: SQLite Bug

```
select e.*,0 from(s,(L))e;
```

```
--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
-   sqlite3WalkSelect(pWalker, pSel);
+   if( sqlite3WalkSelect(pWalker, pSel) ) return WRC_Abort;
    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;
```

```
--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
-   sqlite3WalkSelect(pWalker, pSel);
+   if( sqlite3WalkSelect(pWalker, pSel) ) return WRC_Abort;
    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;
```

Developer Fix


```
--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
-   sqlite3WalkSelect(pWalker, pSel);
+   if( sqlite3WalkSelect(pWalker, pSel) ) return WRC_Abort;
    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;
```



Developer Fix

```
--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
            return 0;
        }
    }
}
```

```

--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
-   sqlite3WalkSelect(pWalker, pSel);
+   if( sqlite3WalkSelect(pWalker, pSel) ) return WRC_Abort;
    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;

```

Prevents null
dereference here

Developer Fix

```

--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ static int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zDb, const char *zDb){
    int n;

    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
            return 0;
        }
    }

```

```
--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
    sqlite3WalkSelect(pWalker, pSel);

    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;
```



```
--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

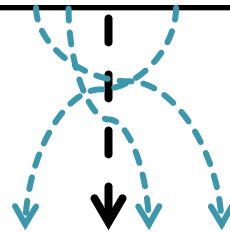
    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
    if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
        return 0;
```

```

--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
    sqlite3WalkSelect(pWalker, pSel);

    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;

```



```

--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
            return 0;
        }
    }

```

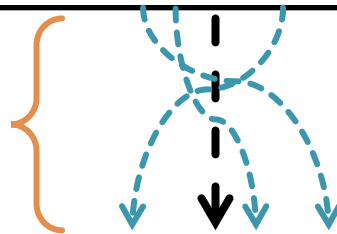
```

--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
    sqlite3WalkSelect(pWalker, pSel);

    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;

```

select e.*,0 from(s,(L))e;



```

--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
            return 0;
        }
    }

```

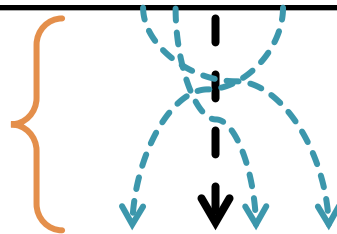
```

--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
    sqlite3WalkSelect(pWalker, pSel);

    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;

```

select e.*,0 from(s,(x))e;



```

--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
            return 0;
        }
    }

```

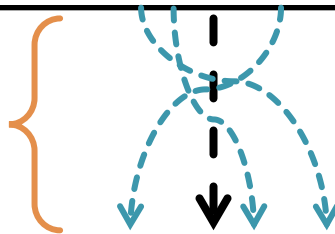
```

--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
    sqlite3WalkSelect(pWalker, pSel);

    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;

```

select e.*,**55** from(s,(x))e;



```

--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
            return 0;
        }
    }

```

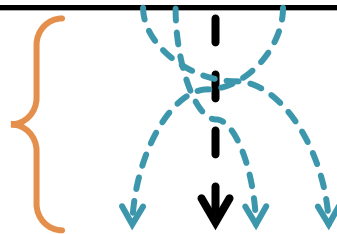
```

--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
    sqlite3WalkSelect(pWalker, pSel);

    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;

```

Source of imprecision



```

--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
            return 0;
        }
    }

```



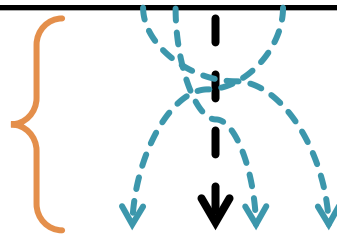
```

--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
    sqlite3WalkSelect(pWalker, pSel);

    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;

```

326 variants



```

--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
            return 0;
        }
    }

```

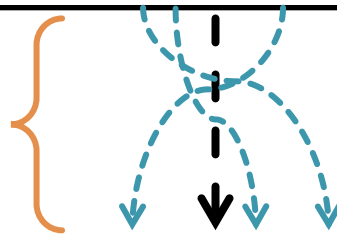
```

--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
    sqlite3WalkSelect(pWalker, pSel);

    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;

```

48 duplicates



```

--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
            return 0;
        }
    }

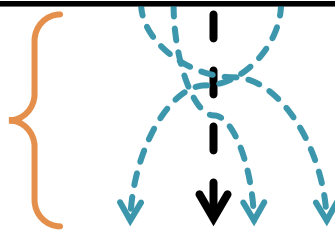
```

```

--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
-   sqlite3WalkSelect(pWalker, pSel);
+   if( sqlite3WalkSelect(pWalker, pSel) ) return WRC_Abort;
    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;

```

48 duplicates



Catches all input
variants

```

--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
            return 0;
        }
    }

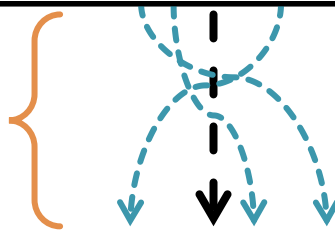
```

```

--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
-   sqlite3WalkSelect(pWalker, pSel);
+   if( sqlite3WalkSelect(pWalker, pSel) ) return WRC_Abort;
    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;

```

48 duplicates



0 duplicates

```

--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
            return 0;
        }
    }

```

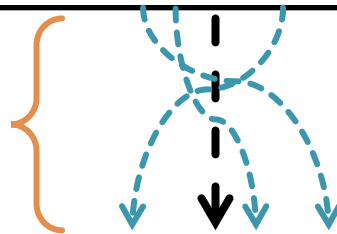
```

--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
    sqlite3WalkSelect(pWalker, pSel);

    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;

```

48 duplicates



Approximate Fix

```

--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanTime(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

+   if(zSpan == NULL) {
+       exit(101);
+   }
    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
            return 0;

```

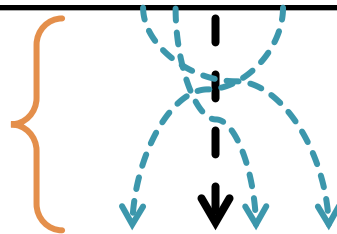
```

--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
    sqlite3WalkSelect(pWalker, pSel);

    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;

```

48 duplicates



Catches the same input variants!

```

--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

+   if(zSpan == NULL) {
+       exit(101);
+   }
    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
            return 0;

```

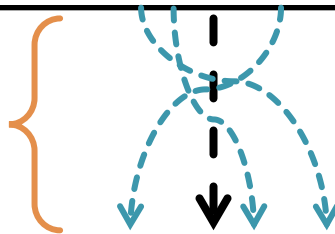
```

--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
    sqlite3WalkSelect(pWalker, pSel);

    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;

```

48 duplicates



Bug specific!

```

--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

+   if(zSpan == NULL) {
+       exit(101);
+   }
    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
            return 0;

```

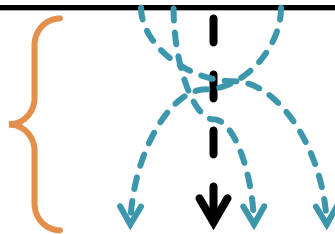
```

--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
    sqlite3WalkSelect(pWalker, pSel);

    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;

```

48 duplicates



0 duplicates

```

--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

+  if(zSpan == NULL) {
+    exit(101);
+  }
    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
            return 0;

```


Suppose we have a bug

**An ideal way to remove duplicate
crashes:**

**An ideal way to remove duplicate
crashes:**

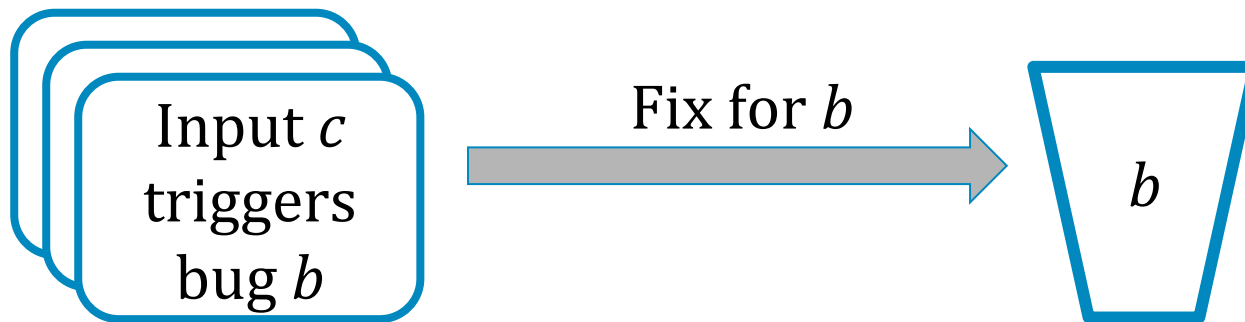
Fix the bug!

An Ideal Solution: No Duplicates

- Intuition: a fix maps all crashing inputs of that bug to non-crashing state.

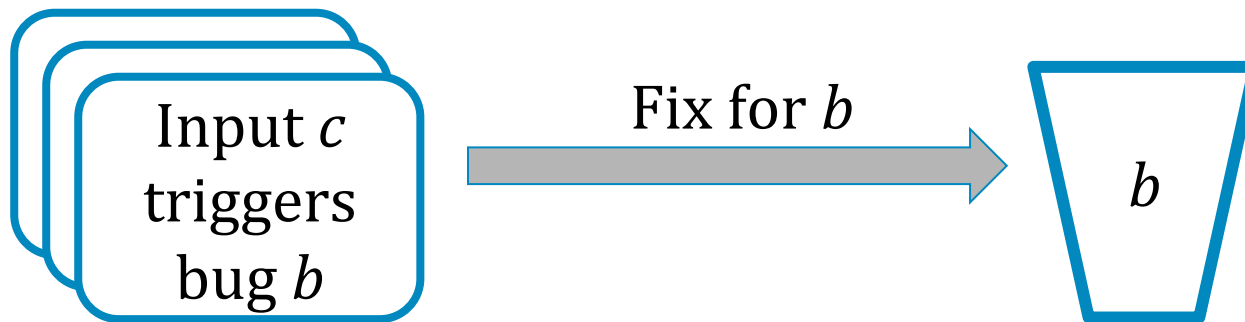
An Ideal Solution: No Duplicates

- Intuition: a fix maps all crashing inputs of that bug to non-crashing state.



An Ideal Solution: No Duplicates

- Intuition: a fix maps all crashing inputs of that bug to non-crashing state.

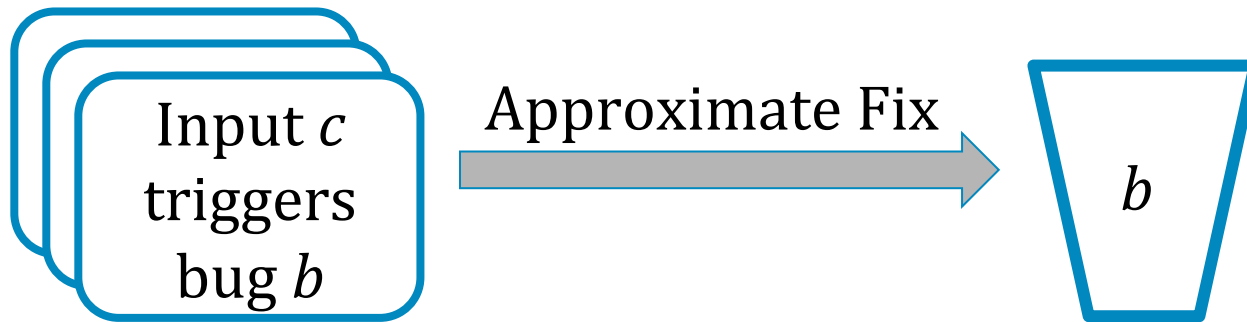


- Needs to be a “correct” developer fix
 - Expensive
 - Hard to automate

Our Key Insight: ***Approximate* Real Bug Fixes**

Our Key Insight: *Approximate Real Bug Fixes*

- Maps crashing inputs as a function of program transformation (semantic delta).



Semantic Crash Bucketing

- Rule-based approach
 - Fix templates (per bug class)

Semantic Crash Bucketing

- Rule-based approach
 - Fix templates (per bug class)

- 
- Null dereferences
 - Buffer overflows

Semantic Crash Bucketing

- Rule-based approach
 - Fix templates (per bug class)

Patch templates

(what to change)

Semantic Crash Bucketing

- Rule-based approach
 - Fix templates (per bug class)
 - Semantic feedback from dynamic execution

Patch templates

(what to change)

Semantic Crash Bucketing

- Rule-based approach
 - Fix templates (per bug class)
 - Semantic feedback from dynamic execution

Patch templates

(what to change)

Bug-specific
semantic cues

Semantic Crash Bucketing

- Rule-based approach
 - Fix templates (per bug class)
 - Semantic feedback from dynamic execution

Patch templates

(what to change)

Bug-specific
semantic cues

(when to apply)

Approximate Fixes for Null Derefs

Approximate Fixes for Null Derefs

```
if (%%%PVAR%%% == null) {  
    exit(101);  
}
```


Approximate Fixes for Null Derefs

```
if (%%%PVAR%%% == null) {  
    exit(101);  
}
```

Run Crashing Input

Approximate Fixes for Null Derefs

```
if (%%%PVAR%%% == null) {  
    exit(101);  
}
```

Run Crashing Input

```
--- a/src/resolve.c  
+++ b/src/resolve.c  
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char  
*zCol, const char *zTab, const char *zDb){  
    int n;  
  
    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){  
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){  
            return 0;  
        }  
    }  
}
```

Approximate Fixes for Null Derefs

```
if (%%%PVAR%%% == null) {  
    exit(101);  
}
```

Run Crashing Input

Check for null variables

```
--- a/src/resolve.c  
+++ b/src/resolve.c  
@@ -164,6 +164,9 @@ int sqlite3MatchSpan(const char *zSpan, const char  
*zCol, const char *zTab, const char *zDb, const char *zSql, int n){  
    int n;  
  
    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){  
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){  
            return 0;  
        }  
    }  
}
```

Approximate Fixes for Null Derefs

```
if (zSpan == null) {  
    exit(101);  
}
```

Run Crashing Input

Generate candidate patch

```
--- a/src/resolve.c  
+++ b/src/resolve.c  
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char  
*zCol, const char *zTab, const char *zDb){  
    int n;  
  
    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){  
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){  
            return 0;  
        }  
    }  
}
```

Approximate Fixes for Null Derefs

```
if (zSpan == null) {  
    exit(101);  
}
```

Run Crashing Input

Validate

```
--- a/src/resolve.c  
+++ b/src/resolve.c  
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char  
*zCol, const char *zTab, const char *zDb){  
    int n;  
  
+   if(zSpan == NULL) {  
+       exit(101);  
+   }  
    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){  
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
```

Evaluating Approximate Fixes

Evaluating Approximate Fixes

1. Compare to bucketing obtained by ground truth developer fixes

Evaluating Approximate Fixes

1. Compare to bucketing obtained by ground truth developer fixes
 - Do approximate fixes "overfit" or hide other bugs?

Evaluating Approximate Fixes

1. Compare to bucketing obtained by ground truth developer fixes
 - Do approximate fixes "overfit" or hide other bugs?
2. Compare bucketing to state-of-art fuzzer deduplication

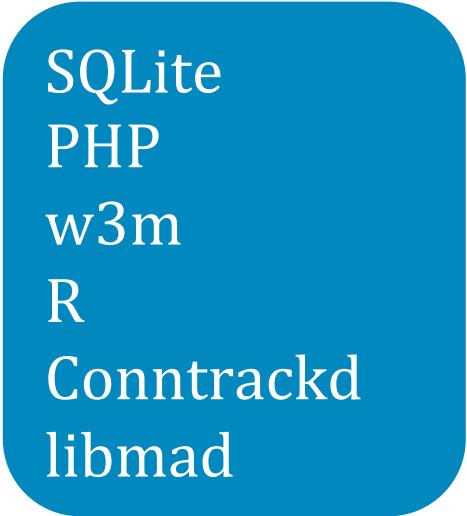
Experimental Setup: Bugs with Ground Truth

Experimental Setup: Bugs with Ground Truth

- Collect developer fixes
 - 18 null dereference bugs
 - 3 buffer overflow bugs
 - 6 real world projects

Experimental Setup: Bugs with Ground Truth

- Collect developer fixes
 - 18 null dereference bugs
 - 3 buffer overflow bugs
 - 6 real world projects



SQLite
PHP
w3m
R
Contrackd
libmad

Experimental Setup: Crash Corpus Generation

Experimental Setup: Crash Corpus Generation

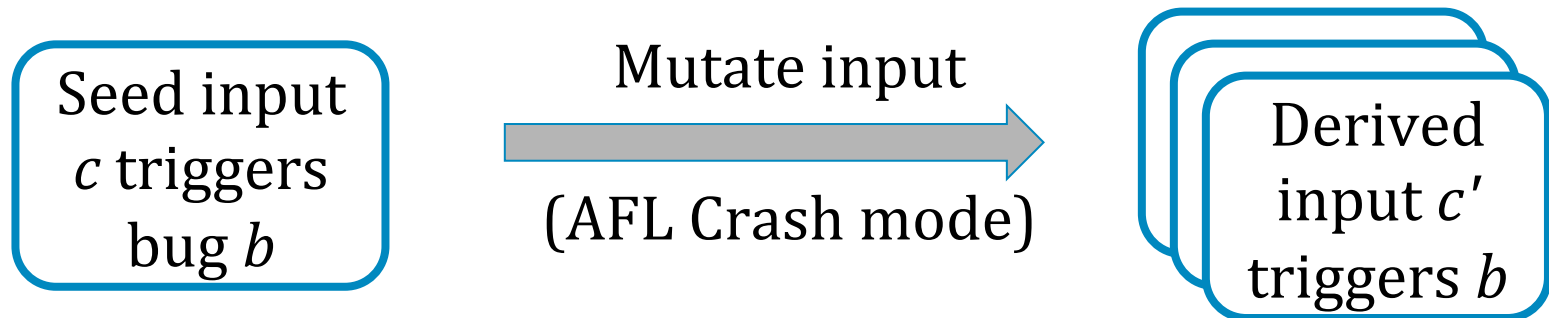
- Organic fuzzing campaigns nondeterministic and expensive

Experimental Setup: Crash Corpus Generation

- Organic fuzzing campaigns nondeterministic and expensive
- Instead: mutate initial seed crashing input to generate derived crashing inputs

Experimental Setup: Crash Corpus Generation

- Organic fuzzing campaigns nondeterministic and expensive
- Instead: mutate initial seed crashing input to generate derived crashing inputs



Experimental Setup: Comparing Fuzzers

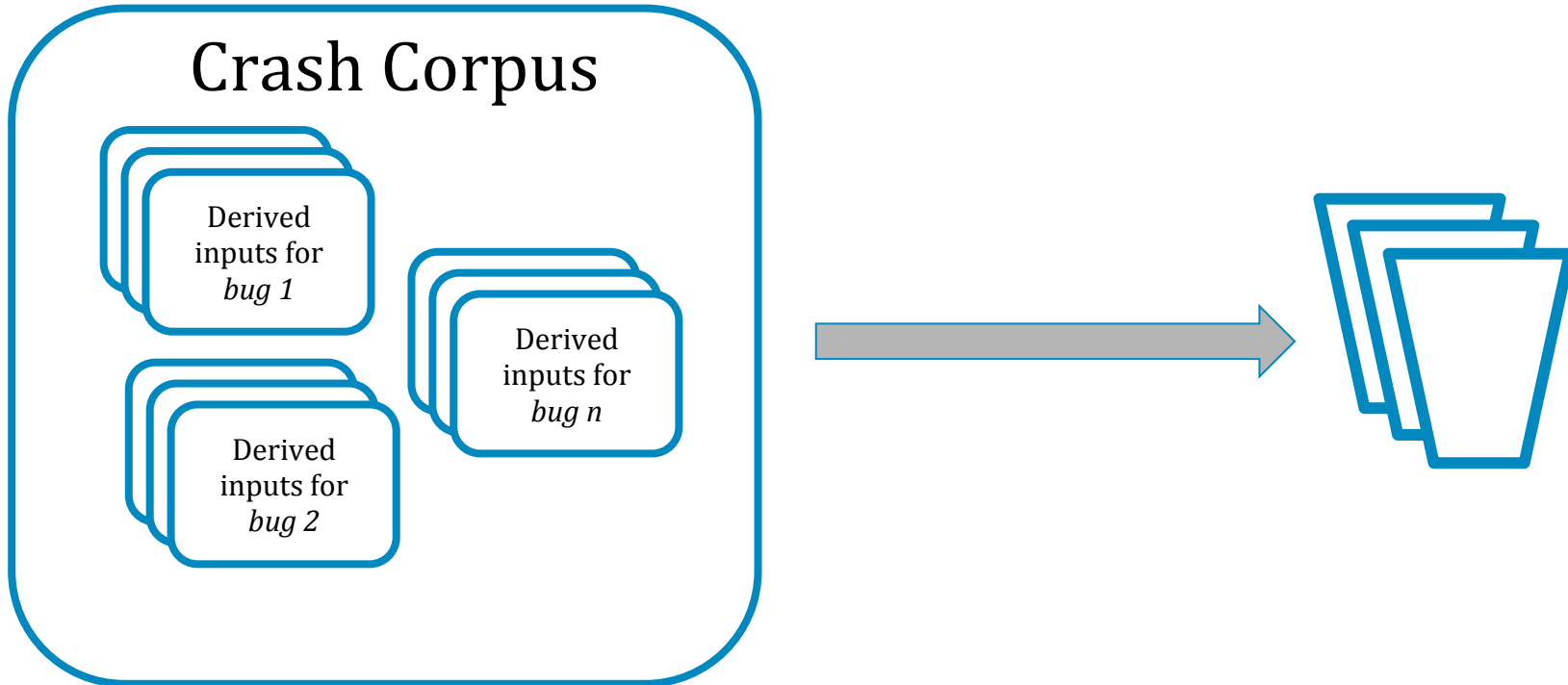
Crash Corpus

Derived
inputs for
bug 1

Derived
inputs for
bug 2

Derived
inputs for
bug n

Experimental Setup: Comparing Fuzzers



Experimental Setup: Comparing Fuzzers

Crash Corpus

Derived
inputs for
bug 1

Derived
inputs for
bug n

Derived
inputs for
bug 2



- AFL-Fuzz
- CERT Basic Fuzzing Framework
- Honggfuzz

Experimental Setup: Comparing Fuzzers

Crash Corpus

Derived
inputs for
bug 1

Derived
inputs for
bug 2

Derived
inputs for
bug n

1 hour campaign

(per fuzzer)



Experimental Setup: Comparing Fuzzers

Crash Corpus

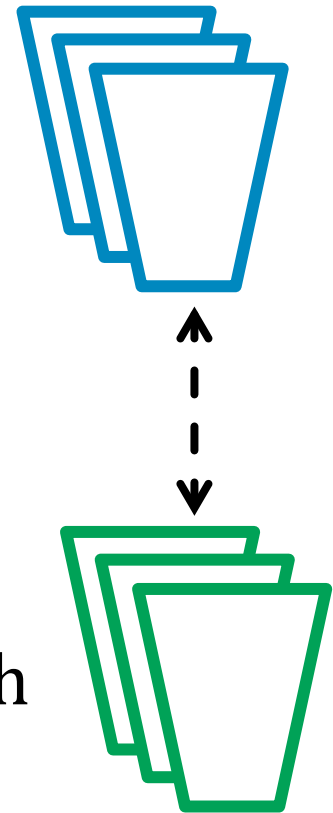
Derived
inputs for
bug 1

Derived
inputs for
bug n

Derived
inputs for
bug 2

1 hour campaign
→
(per fuzzer)

Compare Ground Truth



Experimental Setup: Comparing Fuzzers

Crash Corpus

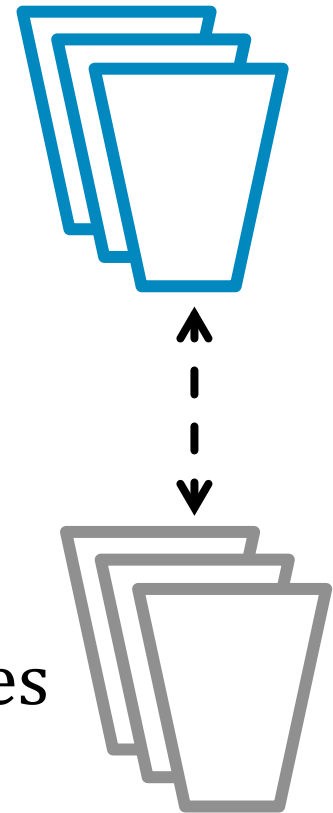
Derived
inputs for
bug 1

Derived
inputs for
bug n

Derived
inputs for
bug 2

1 hour campaign
→
(per fuzzer)

Compare Approximate Fixes



Key Result

- Approximate fixes equal to ground truth for 19 out of 21 bugs

Key Result

- Approximate fixes equal to ground truth for 19 out of 21 bugs
- Just 3 duplicates total

Key Result

- Approximate fixes equal to ground truth for 19 out of 21 bugs
- Just 3 duplicates total
 - AFL-Fuzz: 754
 - BFF: 41
 - Honggfuzz: 1,037

SQLite Results

SQLite Results

(lower is better)

Project	Type	Crash Corpus	SCB	AFL	BFF	HFuzz
SQLite	Null Deref	191				
		482				
		153				
		326				
		139				
		66				
		97				
		235				
		389				
		270				
		167				
		108				

SQLite Results

(lower is better)

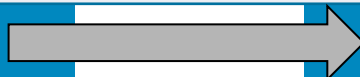
Project	Type	Crash Corpus	SCB	AFL	BFF	HFuzz
SQLite	Null Deref	191	1			
		482	0			
		153	0			
		326	0			
		139	0			
		66	0			
		97	0			
		235	0			
		389	0			
		270	0			
		167	2			
		108	0			

SQLite Results

(lower is better)

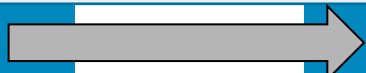
Project	Type	Crash Corpus	SCB	AFL	BFF	HFuzz
SQLite	Null Deref	191	1	25	2	10
		482	0	85	2	4
		153	0	38	6	16
		326	0	48	0	1
		139	0	34	0	0
		66	0	21	0	0
		97	0	20	0	0
		235	0	82	1	3
		389	0	29	1	1
		270	0	65	0	1
		167	2	45	0	4
		108	0	36	0	0

Duplication sensitivity to bug type

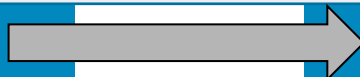

Project	Type	Crash Corpus	SCB	AFL	BFF	HFuzz	
SQLite	Null Deref	191	1	25	2	10	
		482	0	85			4
		153	0	38	6	16	
		326	0	48	0	1	
		139	0	34	0	0	
		66	0	21	0	0	
		97	0	20	0	0	
		235	0	82	1	3	
		389	0	29	1	1	
		270	0	65	0	1	
		167	2	45	0	4	
		108	0	36	0	0	

Duplication sensitivity to bug type

Project	Type	Crash Corpus	SCB	AFL	BFF	HFuzz
SQLite	Null Deref	191	1	25	2	10
		482	0	85		4
		153	0	38	6	16
		326	0	48	0	1
		139	0	34	0	0
		66	0	21	0	0
		97	0	20	0	0
		235	0	82	1	3
		389	0	29	1	1
		270	0	65	0	1
		167	2	45	0	4



Duplication sensitivity to bug type

Project	Type	Crash Corpus	SCB	AFL	BFF	HFuzz	
SQLite	Null Deref	191	1	25	2	10	
		482	0	85			4
		153	0	38	6	16	
		326	0	48	0	1	
		139	0	34	0	0	
		66	0	21	0	0	
		97	0	20	0	0	
		235	0	82	1	3	
		389	0	29	1	1	
		270	0	65	0	1	
		167	2	45	0	4	
R	Overflow	7	0	5			145

SCB does uniformly better

Project	Type	Crash Corpus	SCB	AFL	BFF	HFuzz
SQLite	Null Deref	191	1	25	2	10
		482	0	85		4
		153	0	38	6	16
		326	0	48	0	1
		139	0	34	0	0
		66	0	21	0	0
		97	0	20	0	0
		235	0	82	1	3
		389	0	29	1	1
		270	0	65	0	1
		167	2	45	0	4
R	Overflow	7	0	5		145

Summary

Approximate fixes remove imprecision

```
--- a/src/select.c
+++ b/src/select.c
@@ -4153,7 +4153,7 @@ static int selectExpander(Walker *pWalker, Select *p)
{
    /* A sub-query in the FROM clause of a SELECT */
    assert( pSel!=0 );
    assert( pFrom->pTab==0 );
    sqlite3WalkSelect(pWalker, pSel);

    pFrom->pTab = pTab = sqlite3DbMallocZero(db, sizeof(Table));
    if( pTab==0 ) return WRC_Abort;
```

Source of imprecision

Catches the same input variants!

```
--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

+   if(zSpan == NULL) {
+       exit(101);
+   }
    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
            return 0;
```

Rule-based patch template application

```
if (zSpan[n] == null) {
    exit(101);
}
```

Run Crashing Input

Apply and validate

```
--- a/src/resolve.c
+++ b/src/resolve.c
@@ -164,6 +164,9 @@ int sqlite3MatchSpanName(const char *zSpan, const char
*zCol, const char *zTab, const char *zDb){
    int n;

+   if(zSpan == NULL) {
+       exit(101);
+   }
    for(n=0; ALWAYS(zSpan[n]) && zSpan[n]!='.'; n++){
        if( zDb && (sqlite3StrNICmp(zSpan, zDb, n)!=0 || zDb[n]!=0) ){
```

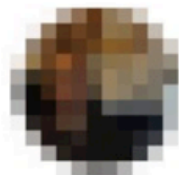
Fewer duplicates than state of art

- Approximate fixes equal to ground truth for 19 out of 21 bugs
- Just 3 duplicates total
 - AFL-Fuzz: 754
 - BFF: 41
 - Honggfuzz: 1,037

SCB does uniformly better

Project	Type	Crash Corpus	SCB	AFL	BFF	HFuzz
SQLite	Null Deref	191	1	25	2	10
		482	0	85	4	4
		153	0	38	6	16
		326	0	48	0	1
		139	0	34	0	0
		66	0	21	0	0
		97	0	20	0	0
		235	0	82	1	3
		389	0	29	1	1
		270	0	65	0	1
		167	2	45	0	4
R	Overflow	7	0	5	145	145

<https://github.com/squaresLab/SemanticCrashBucketing>



Andrew J. ...
...

Follow



Anybody know of good tools for reducing "qualitatively identical" crashes from fuzzing? Example: memcpy crash with invalid pointers can hit the aligned or unaligned path but are both essentially the same bug if the call stack up to that point is identical

8:15 PM - 22 Feb 2018

Approximate Fixes for Overflows

```
size_t angelic_length = 1;  
strncpy(%%%DST%%%,%%%SRC%%%,angelic_length);
```

Run Crashing Input

Scan trace for potentially
unsafe library calls

```
if (GetNextItem(fp, buf, 0, &state)) { fclose(fp); return 0;} /* [ */  
    for(i = 0; i < 256; i++) {  
        if (GetNextItem(fp, buf, i, &state)) { fclose(fp); return 0; }  
        strcpy(encnames[i].cname, buf+1); // overflow  
        ...  
        if (!isPDF) strcat(enccode,"]\n");  
        return 1; // segfault triggered (maybe)  
    }
```

Approximate Fixes for Overflows

```
size_t angelic_length = 1;  
strncpy(encnames[i].cname, buf+1, angelic_length);
```

Run Crashing Input

Actual fix

```
if (GetNextItem(fp, buf, 0, &state)) { fclose(fp); return 0;} /* [ */  
    for(i = 0; i < 256; i++) {  
        if (GetNextItem(fp, buf, i, &state)) { fclose(fp); return 0; }  
+   strncpy(encnames[i].cname, buf+1, 39);  
    ...  
    if (!isPDF) strcat(enccode, "]\n");  
    return 1;  
}
```