Angelix

PRESENTER NAME(S) REMOVED FOR FERPA CONSIDERATIONS

Research questions

- **Can semantic analysis based bug fix method be scaled to repair large-scale real world programs?**
- Previous semantic analysis repair methods (**DirectFix**):
  - promise in quality of repairs
  - the scalability has been a concern

Research questions

- **Can scalable semantic analysis based bug fix method be used to repair multi-location bugs?**
- Previous scalable methods applicable to single location bug fixes (**SemFix**).
- search-based repair method in **GenProg**
  - can change multiple locations
  - BUT the repair is often functionally equivalent to the single line modification.

Research questions

- **How does the repairability and quality of synthesized patches compare to the repairability and quality of patches found by search-based methods, such as SPR?**
- **Is the quality of synthesized patches comparable to the developer-provided patch?**
- **How often do functionality-deleting patches occur?**
  - SPR: frequent functionality-deleting patches

Contributions

- **Angelic forest**:
  - light-weight repair constraint representation
  - program size independent semantic information
- **Angelix:**
  - extension of semantic analysis based bug fix methods
  - both scalability and multi-location bug fixes
  - lower rate of functionality-deleting bug fixes

Why Angelix?

Why Angelix?

- GenProg, SPR, SemiFix, DirectFix, are some good already existing repair programs.

Why Angelix?

- But, scalability of semantic based analysis has always been a concern.

So, How does it work ?

Consider the Following Buggy Code.

```
1  - if (max_range_endpoint < eol_range_start)
2  -    max_range_endpoint = eol_range_start;
3
4  - printable_field = xzalloc(max_range_endpoint/CHAR_BIT+1);
```

First,

- Add if conditionals before each unguarded assignment statement.

First,

```
1   if (max_range_endpoint < eol_range_start)
2       max_range_endpoint = eol_range_start;
3
4   if (1)
5       printable_field = xzalloc(max_range_endpoint/CHAR_BIT+1);
```

Second,

- The repair algorithm replaces user configured n most “suspicious expressions” with symbolic variables.

Second,

```
1    if (α)
2        max_range_endpoint = β;
3
4    if (γ)
5        printable_field = xzalloc(max_range_endpoint/CHAR_BIT+1);
```

Second,

User can configure the number and kinds of expressions that can be converted to symbols.

Lastly,

- Algorithm proceeds to run symbolic execution over the program we obtained above with provided tests to collect the semantic information necessary to synthesize a solution.

Lastly,

```
1    if (0)
2        max_range_endpoint = eol_range_start;
3
4    if (! (max_range_endpoint == 0))
5        printable_field = xzalloc(max_range_endpoint/CHAR_BIT+1);
```

Wait, what does that mean ?

Wait, what does that mean ?

- First, the program tries to detect if there exists some “path” through which the given tests pass.
- If no such path exists then we make the next n suspicious expressions “symbolic” and repeat step 1.
- If such a path exists, solve for values of symbols.
- And lastly, the program state needs to be known.

Semantic Signature of Our example.

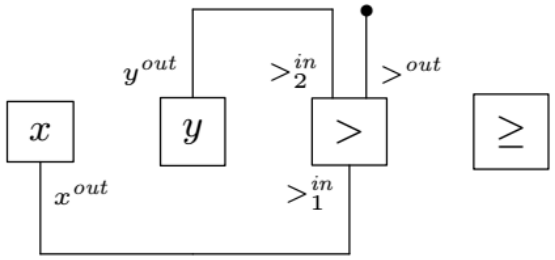
$$
\begin{aligned}
t_1 : \{&\pi_1 : \langle(\alpha, False, \sigma_1), (\gamma, False, \sigma_2)\rangle, \\
&\pi_2 : \langle(\alpha, True, \sigma_3), (\beta, 0, \sigma_4), (\gamma, False, \sigma_5)\rangle\} \\
t_2 : \{&\pi_3 : \langle(\alpha, False, \sigma_6), (\gamma, True, \sigma_7)\rangle, \\
&\pi_4 : \langle(\alpha, True, \sigma_8), (\beta, 3, \sigma_9), (\gamma, True, \sigma_{10})\rangle\},
\end{aligned}
$$

where t_i referes to a test, π_i denotes a test-passing path, and $\sigma_i : Variables \rightarrow Values$ denotes an angelic state.

Patch Synthesis

- Angelic forest serves as input for repair synthesizer
- Repair follows one of the paths in AF for each test
- Each repaired expression returns angelic value
- CBRS:
  - Program = circuit of components (variables, constants, operators)
  - Original program + components + specification (AF) → connect components (satisfy specification, minimize difference from the original)

Patch Synthesis



Input		Output
x	y	
2	1	True
2	2	False
1	2	False

Path	k	$x[\![e^k]\!]$	$y[\![e^k]\!]$	Angelic
1	1	2	1	True
	2	2	2	False
2	1	1	2	False

- $\big((x[e^1] = 2 \wedge y[e^1] = 1 \wedge e^1 = True) \wedge (x[e^2] = 2 \wedge y[e^2] = 2 \wedge e^2 = False)\big) \vee (x[e^1] = 1 \wedge y[e^1] = 2 \wedge e^1 = False)$

Patch Synthesis

- Solve constrains using MaxSMT solver
- Minimal patch:
  - Easier to validate
  - Less likely to change the correct behavior
- Multiple suspicious expressions – fault localization (which expressions to modify + how to modify)

The Test

- Competitors: GenProg, AE, SPR
- Test Subjects: Wireshark (2814 Kloc), PHP (1046 KLoC), Gzip, gmp, libtiff (Software selected from GenProg Evaluation)
  - Some more multi-line defects from CoREBench

The Evaluation

- Were the tools able to synthesize patches that were functionally equivalent to the Developer-provided patches?

The Results

Table 2: Experimental results

Subject	LoC	Tests	Versions	W/I Our Defect Class	Fixed Defects				Equiv. to Developer Fixes				Time (min)
					Angelix	SPR	GenProg	AE	Angelix	SPR	GenProg	AE	
wireshark	2814K	63	7	4	4	4	1	4	0	0	0	0	23
php	1046K	8471	44	12	10	18	5	7	4	8	1	2	62
gzip	491K	12	5	2	2	2	1	2	1	1	0	0	4
gmp	145K	146	2	2	2	2	1	1	2	1	0	0	14
libtiff	77K	78	24	12	10	5	3	5	3	1	0	0	14
Overall			82	32	28	31	11	19	10	11	1	2	32

- What does this mean?

Angelix VS. SPR

- Repairability:
  - Higher in one
  - Lower in another
  - The same in remaining three
- Uniqueness:
  - Angelix produced more unique repairs than any other tool.
    - Multi-line fixes

Subject	Angelix	SPR	GenProg	AE
wireshark	0	0	0	0
php	0	4	0	0
gzip	1	0	0	0
gmp	0	0	0	0
libtiff	5	0	0	0
Overall	6	4	0	0

Correctness of Patch

- Angelix maintains the functionality of the code more than any other tool out there.
- Percentage of functionality-deleting repairs:
  - SPR overall: 42%
  - Angelix: 21%
  - Other tools similarly high

Table 4: The number of functionality-deleting repairs

Subject	Angelix			SPR		
	Fixes	Del	Per	Fixes	Del	Per
wireshark	4	1	25%	4	1	25%
php	10	3	30%	18	7	39%
gzip	2	0	0%	2	1	50%
gmp	2	0	0%	2	0	0%
libtiff	10	2	20%	5	4	80%
Overall	28	6	21%	31	13	42%

Concerns (Future Work)

- While Angelix was proven, still some weaknesses to consider
  - Still unable to create perfectly functionally equivalent patches to developer.
    - Result of fixes outside of its defect scope, e.g. creating a new variable, function call.
  - Angelix developers wish for further research in this area to improve the tool

Conclusion

- Angelix, while not completely perfect, was proven to be effective against leading state-of-the-art software at
  - Implementing repairs in large-scale subjects and in multi-location defects
  - Create repairs that other tools cannot due to its unique patch creation technique
  - Low rate of function-deleting patches to maintain patch and code quality
    - Was able to create a functionally-equivalent patch to fix the HeartBleed vulnerability.

Some Questions:

- What types of bugs or defects does Angelix seem best equipped to fix?

Some Questions:

- Do you believe the Angelic Forest conceptual model is the right direction to pursue with synthesized patching? Could another approach be potentially more effective?

Some Questions:

- The Angelix Forest seems to be a powerful tool for determining an effective solution from a multitude of choices given the structure of local data. Can the Angelic Forest model have other potential applications in other fields?

Some Questions:

- When checking whether the test subject defects were within Angelix' defect class, the developers discounted any developer-provided patches that were outside the scope of Angelix' defect class. Do you think this is a good idea? Should they have still evaluated Angelix against patches that seem to be outside of Angelix' apparent scope?

Some Questions:

- The paper makes a note of how Angelix was able to patch the Heartbleed bug. However, this is the only bug used as evidence of Angelix' ability to patch such vulnerabilities. What other types of vulnerabilities could be tested to better enforce this claim?

Thank You