Automatic Error Elimination by Horizontal Code Transfer Across Multiple Applications

Stelios Sidiroglou-Douskos Eric Lahtinen Fan Long Martin Rinard

slide author names omitted for FERPA compliance

Motivation

- Common runtime errors:
    - Integer overflow
    - Out of bounds access
    - Divide by zero
- Many existing programs already protect against these errors.
- You may not anticipate these errors, but someone did.
- Automatically grab the proper checks from existing programs to protect against runtime errors above.

Research Questions

1) Can errors in software applications be eliminated by generating fixes based solely off of the binaries of different applications that protect against these errors?
2) Is it enough to compare only the inputs (as opposed to the functionality) of two different applications in order to correct errors?
3) Can an error in an older version of a software application be resolved by a targeted update without the disruption often associated with a full upgrade?

Contributions

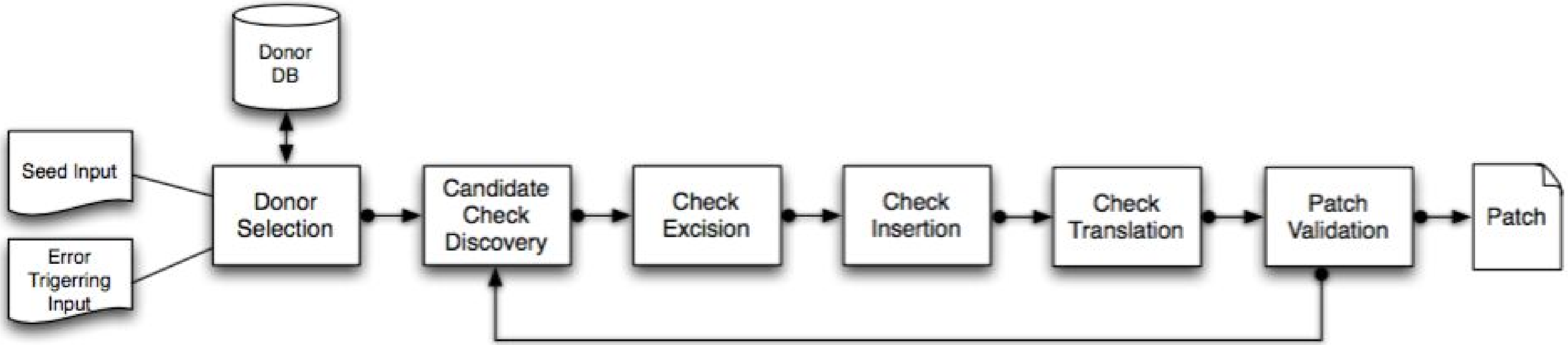
- **Horizontal code transfer** - The novel concept of transferring code from a donor application to a recipient application.

- **Code Phage (CP)** - A system that realizes horizontal code transfer using only the binaries of donor applications in order to fix runtime errors in recipients.

Key Idea: Definitions

- Recipient: The application containing a runtime error which needs to be fixed.
- Donor: The application that protects against the same runtime error.
- Seed input: An input that is successfully processed by the recipient application.
- Error-triggering input: An input that triggers a runtime error in the recipient but not the donor.

Key Idea: High-level

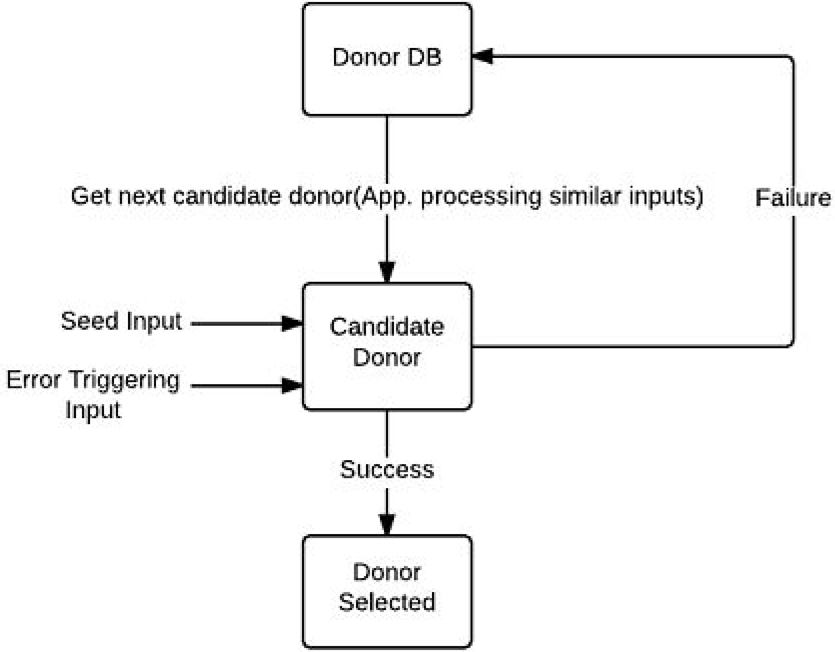


Technique - Error Discovery

- Run DIODE (automatic error discovery tool) on recipient application to identify seed and error triggering input
- Example : CWebP - Converts image to WebP format
  - DIODE identifies overflow error for height = 62848 and width = 23200

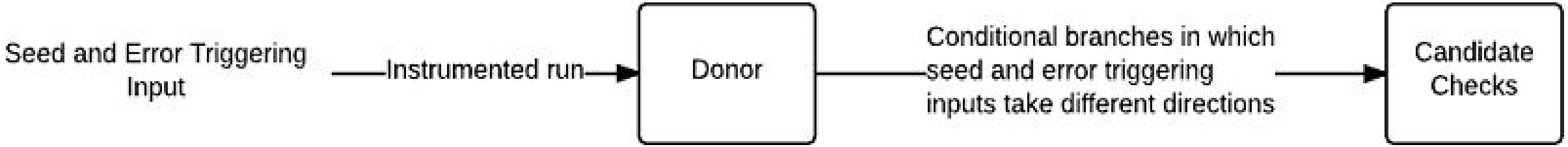
```
int ReadJPEG(...) {
     width = dinfo.output_width;
     height = dinfo.output_height;
     stride = dinfo.output_width * dinfo.output_components * sizeof(*rgb);
     /* the overflow error */
     rgb = (uint8_t*)malloc(stride * height);
     if (rgb == NULL) {
          goto End;
     }
}
```

Technique - Donor Selection



Example : FEH - Image Viewer is identified as a donor for CWebP

Technique - Candidate Check Discovery



- Run instrumented version of donor application on seed and error triggering inputs
- Records the conditional branches influenced by the relevant input bytes
- Records the direction taken by the seed input and the error triggering input
- Candidate Check: Check at which the inputs take two different directions

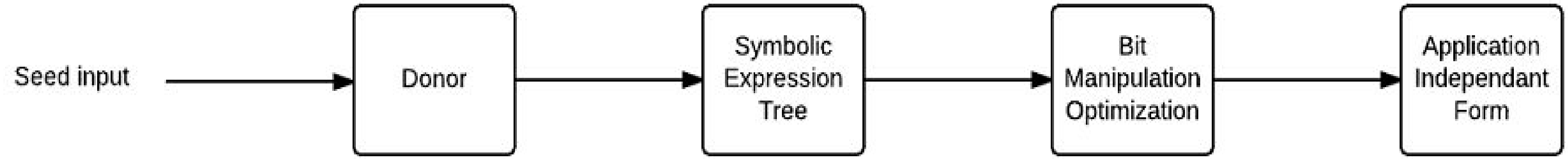
Technique - Candidate Check Discovery

Example : FEH Image Viewer

```
# define IMAGE_DIMENSIONS_OK(w, h) ( ((w) > 0) && ((h) > 0) &&  ((unsigned long long)(w) * \
  (unsigned long long)(h) <= (1ULL << 29) - 1) )

char load(...) {
   int w, h;
   struct jpeg_decompress_struct cinfo; struct ImLib_JPEG_error_mgr jerr; FILE *f;
   if (...) { ...
        im->w = w = cinfo.output_width;
        im->h = h = cinfo.output_height;
      /* Candidate check condition */
        if ((cinfo.rec_outbuf_height > 16) || (cinfo.output_components <= 0) ||
     !IMAGE_DIMENSIONS_OK(w, h)) {
            return 0;
      }
   }
}
```

Technique - Candidate Check Excision



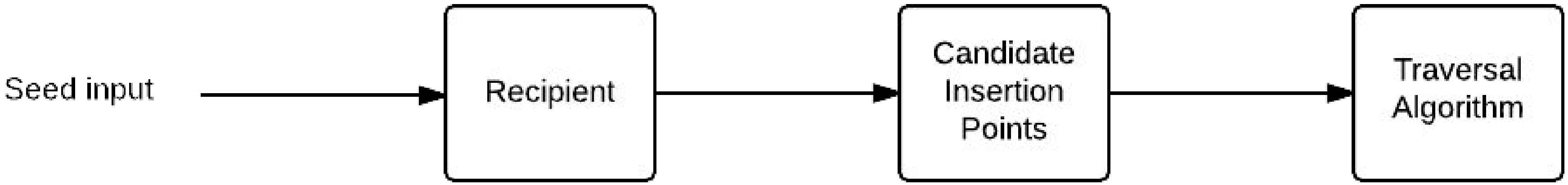
- Reruns the donor with additional instrumentation to generate the full symbolic expression tree for candidate checks
- Symbolic Expression Tree : Records how the conditions in the check were computed by tracking the flow of input bytes
- Bit Manipulation Optimization to reduce the size of the symbolic expression tree

Technique - Candidate Check Excision

Example : Code Phage generates the following application-independent symbolic expression from the FEH Image Viewer

ULessEqual(32,Shrink(32,Mul(64,Shrink(32,Div(32,BvOr(64,Shl(64, ToSize(64,SShr(32,Sub(32,Add(32,Constant(8),Shl(32,Add(32,Shl (32,ToSize (32,BvAnd(16,HachField(16,’/start_frame/content/height’), Constant(0xFF))),Constant(8)),ToSize(32,UShr(32,BvAnd(16,HachField(16, ’ /start_frame/content/height’),Constant(0xFF00)),Constant(8)))), Constant(3))),Constant(1)),Constant(31))),Constant(32)),ToSize(64, Sub(32,Add(32, Constant(8),Shl(32,Add(32,Shl(32,ToSize(32,BvAnd(16, HachField(16,’/start_frame/content/height’),Constant(0xFF))),Constant(8)), ToSize(32,UShr (32,BvAnd(16,HachField(16,’/start_frame/content/height’), Constant(0xFF00)),Constant(8)))),Constant(3))),Constant(1)))),Constant(8))), Shrink(32, Div(32,BvOr(64,Shl(64,ToSize(64,SShr(32,Sub(32,Add(32, Constant(8),Shl(32,Add(32,Shl(32,ToSize(32,BvAnd(16,HachField(16, ’ /start_frame/content/width’),Constant(0xFF))),Constant(8)), ToSize(32,UShr(32,BvAnd(16,HachField(16,’/start_frame/content/width’), Constant (0xFF00)),Constant(8)))),Constant(3))),Constant(1)), Constant(31))),Constant(32)),ToSize(64,Sub(32,Add(32,Constant(8), Shl(32,Add(32,Shl(32, ToSize(32,BvAnd(16,HachField(16, ’/start_frame/content/width’),Constant(0xFF))),Constant(8)), ToSize(32,UShr(32,BvAnd(16,HachField(16,’ /start_frame/content/width’), Constant(0xFF00)),Constant(8)))),Constant(3))),Constant(1)))), Constant(8))))),Constant(536870911))

Technique - Check Insertion



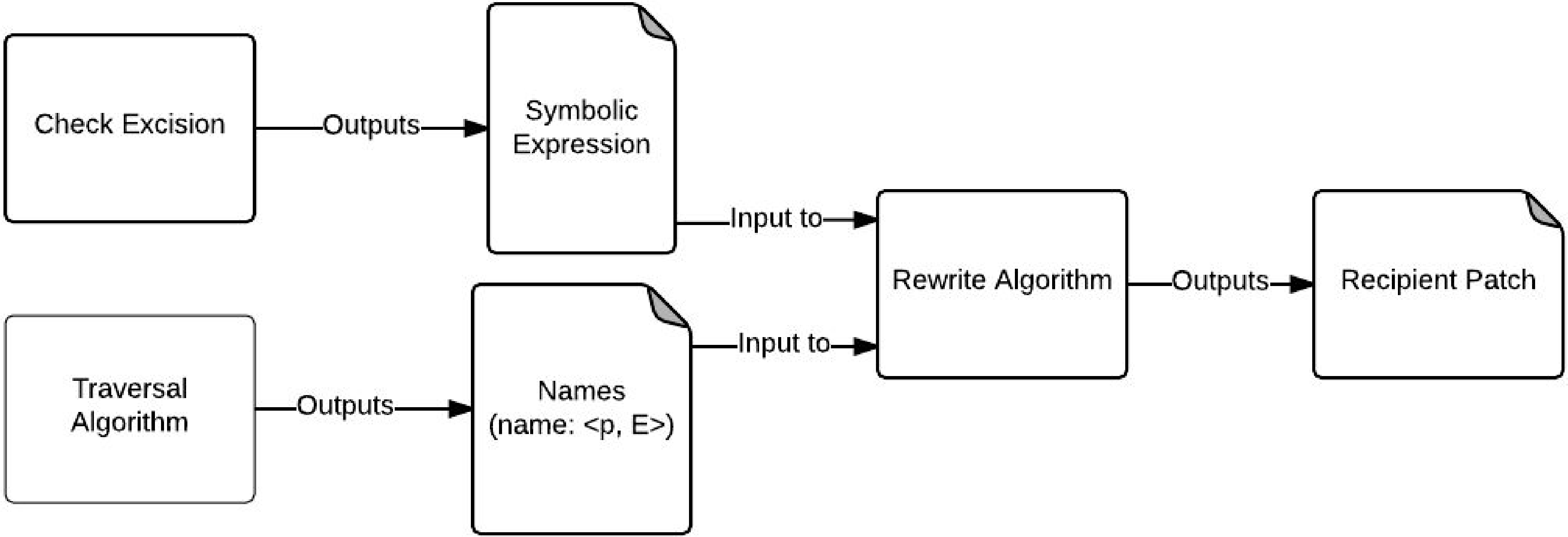
- Run the instrumented version of the recipient
- Locate candidate points - points at which relevant input bytes are available as program expressions in the recipient
- Remove unstable points (points which execute different values when invoked from different parts of the application), to reduce the risks of inducing irrelevant errors
- Obtain local and global variables at insertion points and feed it to the traversal algorithm
- Traversal algorithm - gives the Names of those variables that leads to a reachable relevant input variable
    - outputs a set of pairs where each pair has the form <p,E>, where p is the path leading to a reachable relevant variable and E is the symbolic expression

Technique - Check Insertion

Example : Insertion point at CWebP after line 2

```
int ReadJPEG(...) {
1    width = dinfo.output_width;
2    height = dinfo.output_height;
3    stride = dinfo.output_width * dinfo.output_components * sizeof(*rgb);
4    /* the overflow error */
5    rgb = (uint8_t*)malloc(stride * height);
6    if (rgb == NULL) {
7         goto End;
8    }
9}
```

Technique - Check Translation

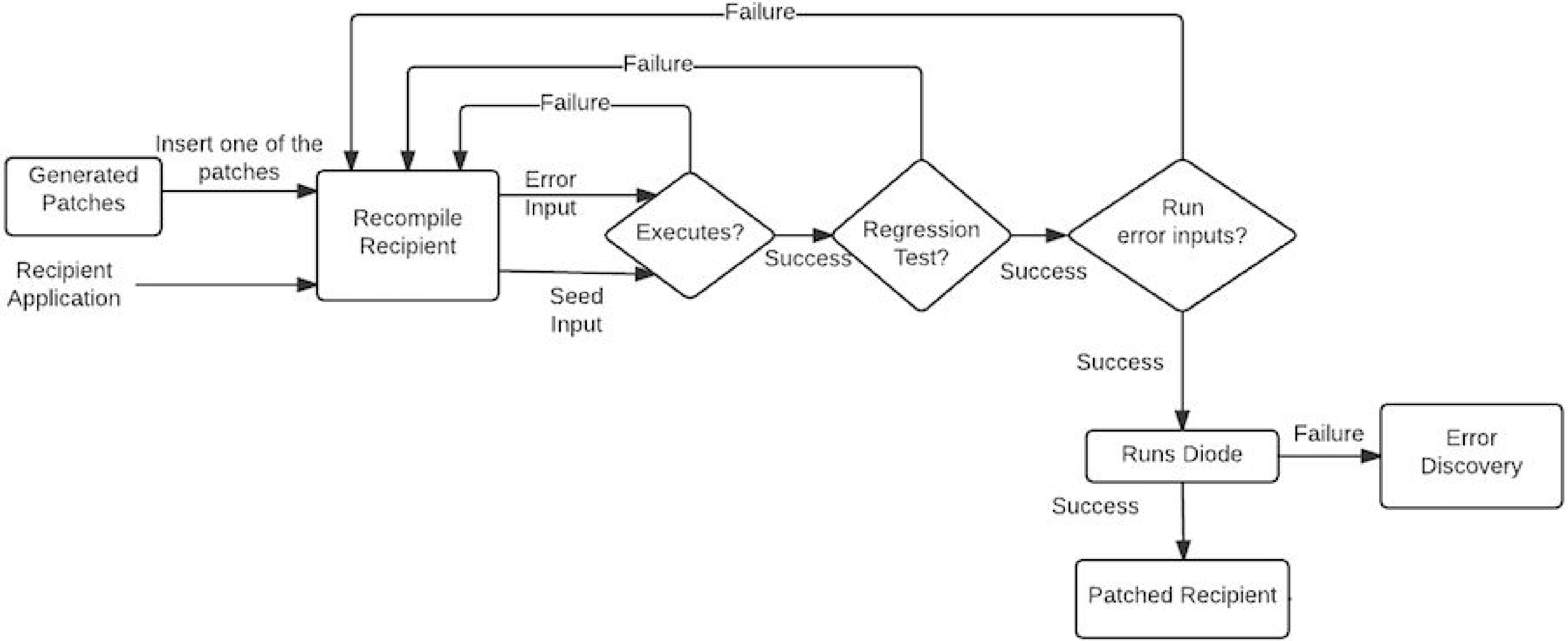


Technique - Check Translation

Example : Generated Patch for CWebP

```
if (!((unsigned long long)dinfo.output_height * (unsigned
long long)dinfo.output_width)<=536870911)) {
    exit(-1);
}
```

Technique - Patch Validation



Evaluation

- Code Phage was evaluated on three errors
  - Integer Overflow
  - Out of bounds
  - Divide by zero
- Recipients selected - 7
- Donors selected - 8

Results

Recipient	Target	Donor	Generation Time	# Relevant Branches	# Flipped Branches	# Used Checks	# Candidate Insertion Pts	Check Size
CWebP 0.3.1	jpegdec.c:248	feh-2.9.3	4m	157	5	1	38 - 2 - 31 = 5	57 → 4
CWebP 0.3.1	jpegdec.c:248	mtpaint-3.40	4m	94	5	1	38 - 2 - 30 = 6	28 → 2
CWebP 0.3.1	jpegdec.c:248	viewnior-1.4	1m	137	1	1	38 - 2 - 31 = 5	111 → 12
Dillo 2.1	png.c@203	mtpaint-3.40	3m	29	[1,1]	2	16 - 1 - 8 = 7 16 - 1 - 9 = 6	[(18 → 1),(18 → 1)]
Dillo 2.1	png.c@203	feh-2.9.3	3m	120	[4,1]	2	16 - 1 - 9 = 6 16 - 1 - 9 = 6	[(76 → 8), (37 → 3)]
Dillo 2.1	png.c@203	viewnior-1.4	18m	117	1	1	16 - 1 - 9 = 6	79 → 12
Dillo 2.1	fltkimagebuf.cc@39	mtpaint-3.40	13m	29	[1,1]	2	22 - 1 - 10 = 11 22 - 1 - 11 = 10	[(18 → 1),(18 → 1)]
Dillo 2.1	fltkimagebuf.cc@39	feh-2.9.3	2m	120	4	1	22 - 1 - 11 = 10	76 → 9
Dillo 2.1	fltkimagebuf.cc@39	viewnior-1.4	9m	117	1	1	22 - 1 - 11 = 10	79 → 12
Display 6.5.2	xwindow.c@5619	viewnior-1.4	4m	142	6	1	74 - 5 - 60 = 9	55 → 14
Display 6.5.2	xwindow.c@5619	feh-2.9.3	4m	147	6	1	74 - 7 - 58 = 9	17 → 4
Display 6.5.2	display.c@4393	viewnior-1.4	4m	142	6	1	49 - 2 - 45 = 2	55 → 14
Display 6.5.2	display.c@4393	feh-2.9.3	4m	147	6	1	49 - 2 - 45 = 2	17 → 4
SwfPlay 0.5.5	jpeg_rgb_decoder.c@253	gnash	12m	264	7	1	43 - 3 - 35 = 5	53 → 12
SwfPlay 0.5.5	jpeg.c@192	gnash	18m	264	[1,1,3,3]	4	38 - 2 - 34 = 2 38 - 2 - 34 = 2 38 - 0 - 37 = 1 38 - 0 - 37 = 1	[(5 →1),(5 →1),(4 →1),(3 →1)]
JasPer 1.9	jpg_dec.c:492	OpenJpeg 1.5.2	1m	63	19	1	18 - 1 - 16 = 1	188 → 3
gif2tiff 4.0.3	gif2tiff.c:355	Display 6.5.2-9	9m	9	2	1	2 - 1 - 0 = 1	3 → 3
Wireshark 1.4.14	packet-dcp-etsi.c:258	Wireshark 1.8.6	4m	101	2	1	40 - 5 - 15 = 20	6 → 2

Results: Patch Generation Time

- Minimum: 1 minute
- Maximum: 18 minutes
- Average: 6.5 minutes
- Mode: 4 minutes

Blue: CWebP example; **Red:** Key points

Recipient	Target	Donor	Generation Time
CWebP 0.3.1	jpegdec.c:248	feh-2.9.3	4m
CWebP 0.3.1	jpegdec.c:248	mtpaint-3.40	4m
CWebP 0.3.1	jpegdec.c:248	viewnior-1.4	1m
Dillo 2.1	png.c@203	mtpaint-3.40	3m
Dillo 2.1	png.c@203	feh-2.9.3	3m
Dillo 2.1	png.c@203	viewnior-1.4	18m
Dillo 2.1	fltkimagebuf.cc@39	mtpaint-3.40	13m
Dillo 2.1	fltkimagebuf.cc@39	feh-2.9.3	2m
Dillo 2.1	fltkimagebuf.cc@39	viewnior-1.4	9m
Display 6.5.2	xwindow.c@5619	viewnior-1.4	4m
Display 6.5.2	xwindow.c@5619	feh-2.9.3	4m
Display 6.5.2	display.c@4393	viewnior-1.4	4m
Display 6.5.2	display.c@4393	feh-2.9.3	4m
SwfPlay 0.5.5	jpeg_rgb_decoder.c@253	gnash	12m
SwfPlay 0.5.5	jpeg.c@192	gnash	18m
JasPer 1.9	jpg_dec.c:492	OpenJpeg 1.5.2	1m
gif2tiff 4.0.3	gif2tiff.c:355	Display 6.5.2-9	9m
Wireshark 1.4.14	packet-dcp-etsi.c:258	Wireshark 1.8.6	4m

Results : Candidate Insertion Points

- **X-Y-Z = W**

- X: # of Candidate Insertion Points
- Y: # of Unstable Points
- Z: # of Insertion Points where no Patch was generated

- W : # of points where successful patch was inserted

Blue: CWebP example; **Red:** Key points

Recipient	Target	Donor	# Candidate Insertion Pts
CWebP 0.3.1	jpegdec.c:248	feh-2.9.3	38 - 2 - 31 = 5
CWebP 0.3.1	jpegdec.c:248	mtpaint-3.40	38 - 2 - 30 = 6
CWebP 0.3.1	jpegdec.c:248	viewnior-1.4	38 - 2 - 31 = 5
Dillo 2.1	png.c@203	mtpaint-3.40	16 - 1 - 8 = 7 16 - 1 - 9 = 6
Dillo 2.1	png.c@203	feh-2.9.3	16 - 1 - 9 = 6 16 - 1 - 9 = 6
Dillo 2.1	png.c@203	viewnior-1.4	16 - 1 - 9 = 6
Dillo 2.1	fltkimagebuf.cc@39	mtpaint-3.40	22 - 1 - 10 = 11 22 - 1 - 11 = 10
Dillo 2.1	fltkimagebuf.cc@39	feh-2.9.3	22 - 1 - 11 = 10
Dillo 2.1	fltkimagebuf.cc@39	viewnior-1.4	22 - 1 - 11 = 10
Display 6.5.2	xwindow.c@5619	viewnior-1.4	74 - 5 - 60 = 9
Display 6.5.2	xwindow.c@5619	feh-2.9.3	74 - 7 - 58 = 9
Display 6.5.2	display.c@4393	viewnior-1.4	49 - 2 - 45 = 2
Display 6.5.2	display.c@4393	feh-2.9.3	49 - 2 - 45 = 2
SwfPlay 0.5.5	jpeg_rgb_decoder.c@253	gnash	43 - 3 - 35 = 5
SwfPlay 0.5.5	jpeg.c@192	gnash	38 - 2 - 34 = 2 38 - 2 - 34 = 2 38 - 0 - 37 = 1 38 - 0 - 37 = 1
JasPer 1.9	jpg_dec.c:492	OpenJpeg 1.5.2	18 - 1 - 16 = 1
gif2tiff 4.0.3	gif2tiff.c:355	Display 6.5.2-9	2 - 1 - 0 = 1
Wireshark 1.4.14	packet-dcp-etsi.c:258	Wireshark 1.8.6	40 - 5 - 15 = 20

Results: Check Size

- X → Y
- X : # of operations in the application-independent representation of the check
- Y: # of operations in the translated check inserted in the recipient

- Minimum: 14
- Maximum: 2

Blue: CWebP example; Red: Key points

Recipient	Target	Donor	Check Size
CWebP 0.3.1	jpegdec.c:248	feh-2.9.3	57 → 4
CWebP 0.3.1	jpegdec.c:248	mtpaint-3.40	28 → 2
CWebP 0.3.1	jpegdec.c:248	viewnior-1.4	111 → 12
Dillo 2.1	png.c@203	mtpaint-3.40	[(18 → 1),(18 → 1)]
Dillo 2.1	png.c@203	feh-2.9.3	[(76 → 8), (37 → 3)]
Dillo 2.1	png.c@203	viewnior-1.4	79 → 12
Dillo 2.1	fltkimagebuf.cc@39	mtpaint-3.40	[(18 → 1),(18 → 1)]
Dillo 2.1	fltkimagebuf.cc@39	feh-2.9.3	76 → 9
Dillo 2.1	fltkimagebuf.cc@39	viewnior-1.4	79 → 12
Display 6.5.2	xwindow.c@5619	viewnior-1.4	55 → 14
Display 6.5.2	xwindow.c@5619	feh-2.9.3	17 → 4
Display 6.5.2	display.c@4393	viewnior-1.4	55 → 14
Display 6.5.2	display.c@4393	feh-2.9.3	17 → 4
SwfPlay 0.5.5	jpeg_rgb_decoder.c@253	gnash	53 → 12
SwfPlay 0.5.5	jpeg.c@192	gnash	[(5 →1),(5 →1),(4 →1),(3 →1)]
JasPer 1.9	jpg_dec.c:492	OpenJpeg 1.5.2	188 → 3
gif2tiff 4.0.3	gif2tiff.c:355	Display 6.5.2-9	3 → 3
Wireshark 1.4.14	packet-dcp-etsi.c:258	Wireshark 1.8.6	6 → 2

Results: Main Takeaways

- Maximum time to generate a patch was 18 minutes and minimum was 1 minute
- Maximum check size = 14 and minimum check size = 2
- Successfully generated correct patches for all of the recipient/donor pairs
- Success highlight CP’s effective techniques
  - Check Identification Technique
  - Insertion Point Location algorithm
  - Rewrite Algorithm

Research questions (Revisited)

1) Can errors in software applications be eliminated by generating fixes based solely off of the binaries of different applications that protect against these errors?
2) Is it enough to compare only the inputs (as opposed to the functionality) of two different applications in order to correct errors?
3) Can an error in an older version of a software application be resolved by a targeted update without the disruption often associated with a full upgrade?

RQ1 : Binary Donors & RQ2: Divergent Functionality

Runtime Error	Number of Errors Found	Number of Errors Resolved	Recipients	Donors
Integer Overflow	7	7	CWebP 0.31 Dillo 2.1 swfplay 0.55 Display 6.5.2-8	FEH-2.9.3 mtpaint 3.4 ViewNoir 1.4 ViewNoir 0.8.11
Out of Bounds Access	2	2	JarPer 1.9 gif2tiff 4.0.3	OpenJPEG Display 6.5.2-9

RQ1 and RQ2 : Results

- For each of the donors, CP had access only to their binaries and not the source code.
- Each of the recipient-donor pairs process the same input, but had different functionalities.
- Code Phage was able to successfully generate patches for all of the recipient applications that eliminated errors.

RQ3: Multi Version Code Transfer

Runtime Error	Number of Errors Found	Number of Errors Resolved	Recipients	Donors
Divide By Zero	2	2	Wireshark-1.4.14	Wireshark-1.8.6

RQ3 : Results

- Obtained a targeted updated by resolving error in Wireshark-1.4.14 without performing a full upgrade to Wireshark-1.8.6
- Implemented an alternative strategy to return 0 rather than exiting when divide by zero error is encountered. This enabled the application to continue to execute productively

Discussion question #1

We see that CP can fix three different errors (integer overflow, out of bounds access, divide by zero). Does it seem that CP could work on other errors?

Discussion question #2

Do you think CP could be extended to allow custom code to be executed within the generated patch to handle specific errors?

Discussion question #3

The experimental results reflect a 100% success rate, but for a very small set of applications. Do these results make you think CP is reliable?

Discussion question #4

The research paper was rather ambiguous regarding how the set of possible donors was constructed. How would you obtain a list of applications that could be candidates for the donor selection process? Do you think this affects the success rate of CP?

Discussion question #5

Could Code Phage be used to maliciously reverse engineer specific algorithms of closed-source projects?

REFERENCE

Sidiroglou--Douskos, Stelios, et al. "Automatic error elimination by horizontal code transfer across multiple applications." (2015).