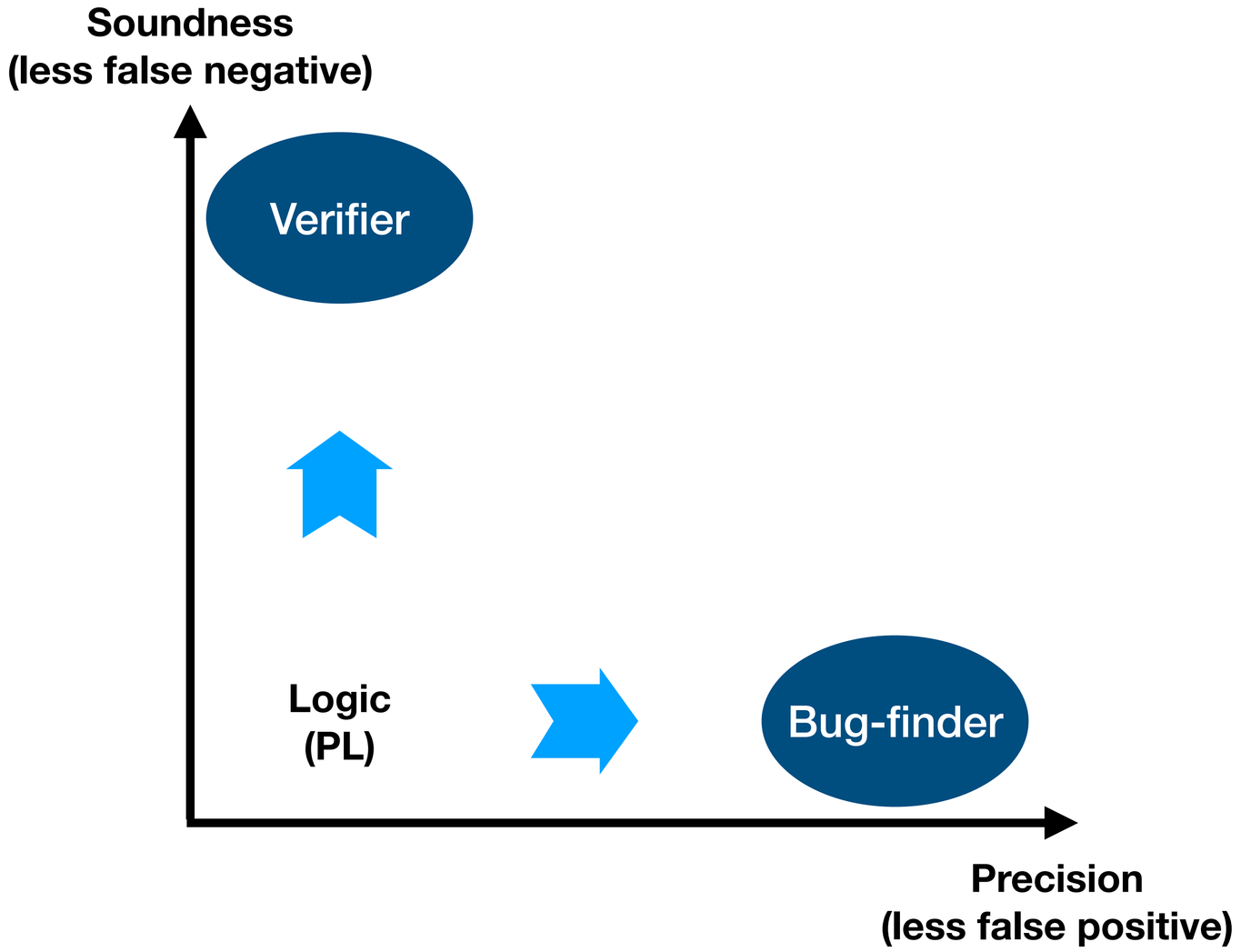
AI 기반 프로그램 오류 분석

허기홍
KAIST 전산학부

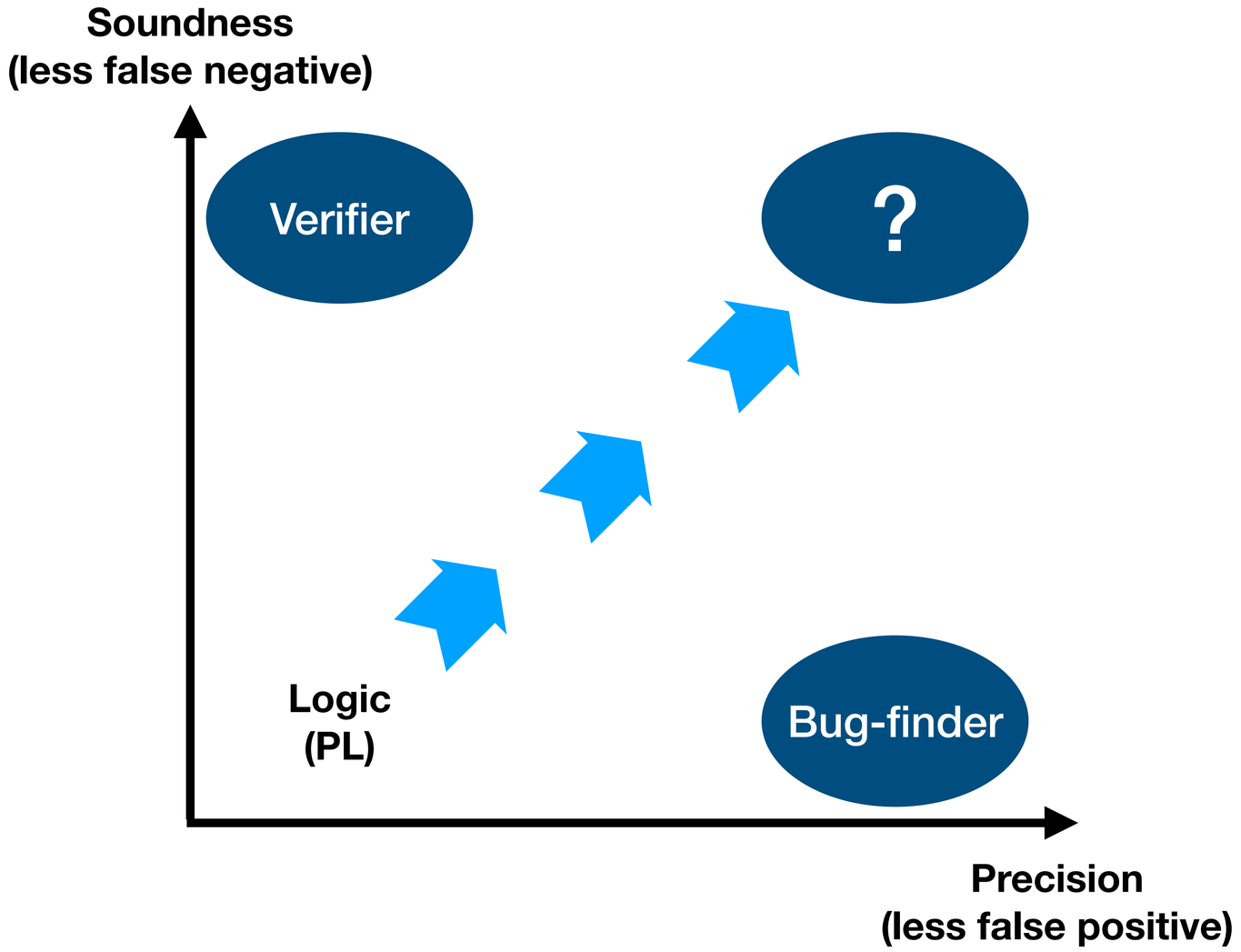
KAIST

Programming Systems Laboratory

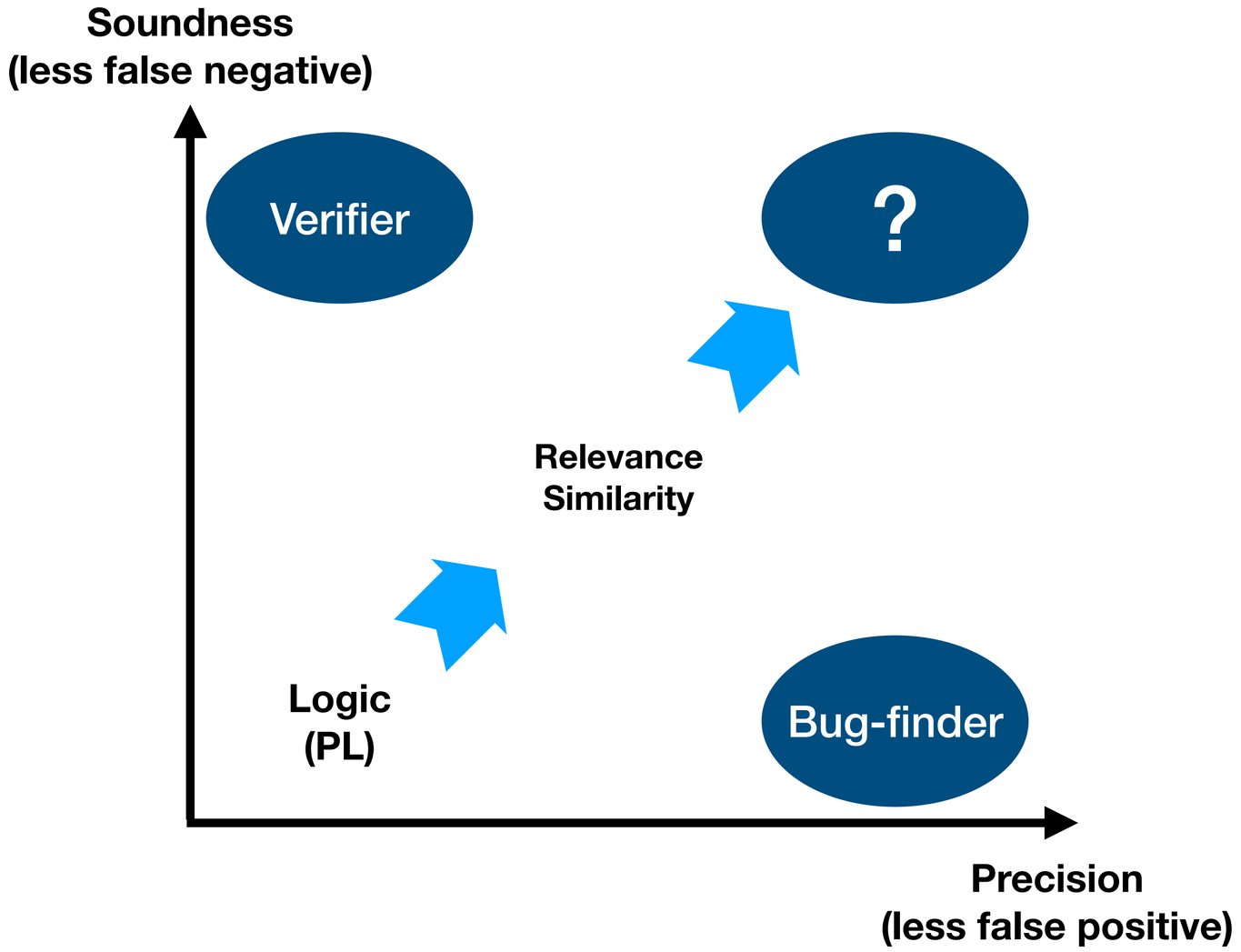
Conventional Program Analysis



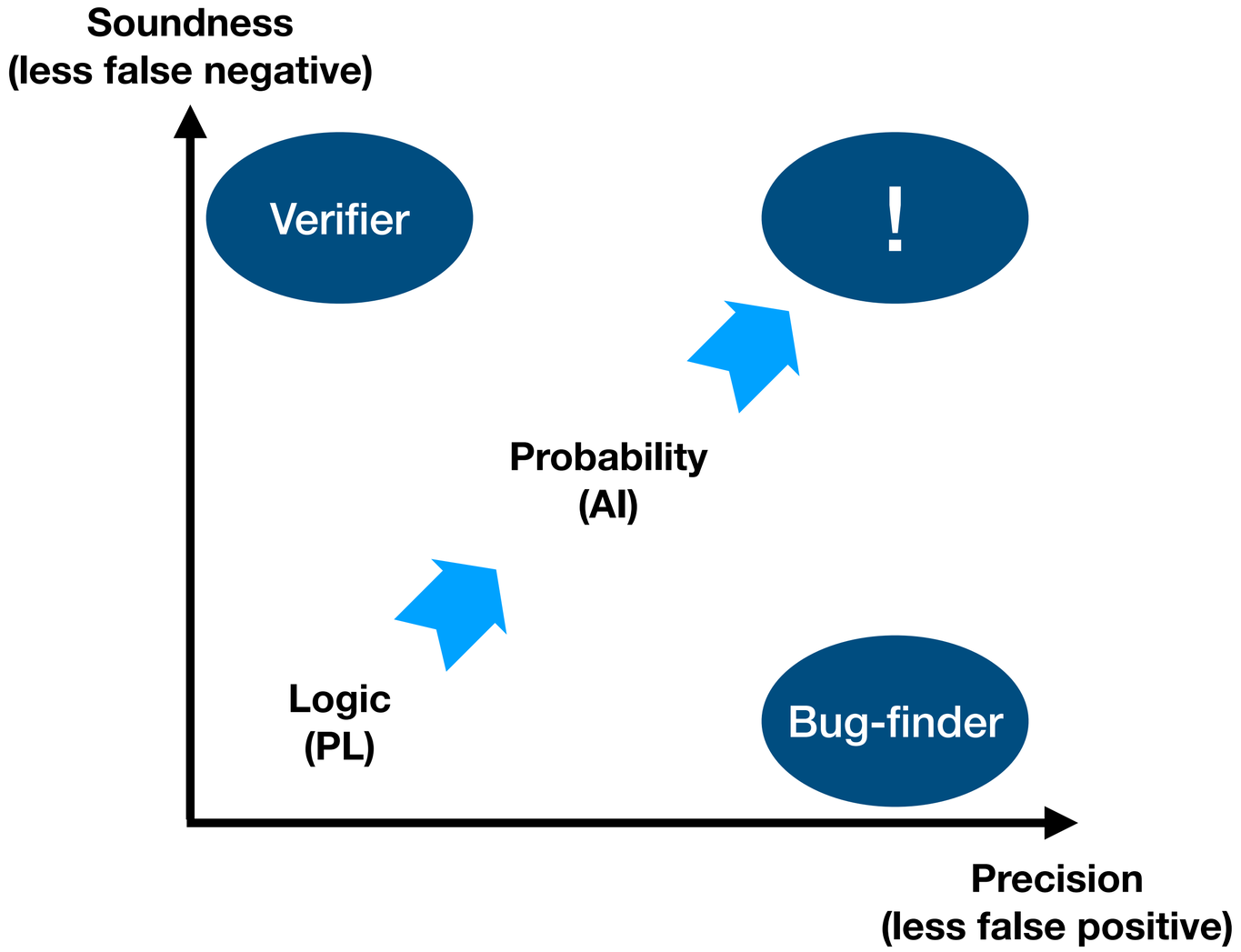
Conventional Program Analysis



Our Goal


Soundness
(less false negative)
Verifier
?
Relevance
Similarity
Logic
(PL)
Bug-finder
Precision
(less false positive)

Our Goal


Soundness
(less false negative)
Verifier
!
Probability
(AI)
Logic
(PL)
Bug-finder
Precision
(less false positive)

Static Analysis for SW Error Detection

	Verifier	Bug-finder
Methodology	Property-based	Pattern-based
What	General correctness properties	Specific bug patterns

```
void f() {
  char buf[16];
  int i = 0;
  while (i < 16) i++;
  // buffer overrun
  buf[i] = 0;
}
```

```
void g() {
  char buf[16];
  char *p = input();
  // buffer overrun
  strcpy(buf, p);
}
```

```
void read_bmp(File *f) {
  int w = read_bmp_width(f);
  int h = read_bmp_height(f);
  // integer overflow
  int area = w * h;
  char *buf = malloc(area);
  ...
}
```

```
void read_jpg(File *f) {
  int w = read_jpg_width(f);
  int h = read_jpg_height(f);
  // integer overflow
  int area = w * h;
  char *buf = malloc(area);
  ...
}
```

Static Analysis for SW Error Detection

	Verifier	Bug-finder
Methodology	Property-based	Pattern-based
What	General correctness properties	Specific bug patterns
How	Checking for logical properties	Searching for bug patterns

idx < size?

area < INT_MAX?

p != NULL?

1. **Read two numbers from a file**
2. **Multiplies the numbers**
3. **Use it as an argument of malloc**

Static Analysis for SW Error Detection

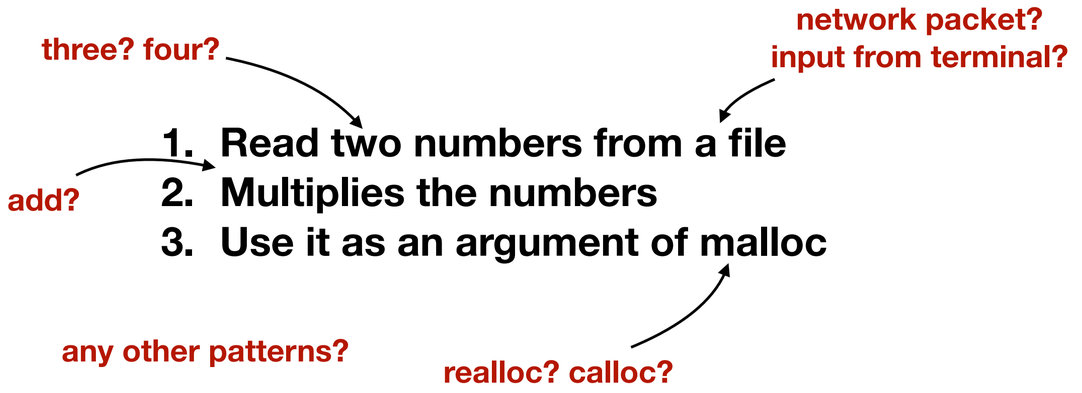
	Verifier	Bug-finder
Methodology	Property-based	Pattern-based
What	General correctness properties	Specific bug patterns
How	Checking for logical properties	Searching for bug patterns
Challenge	False positives	False negatives

pointer?

loop?

external libraries?

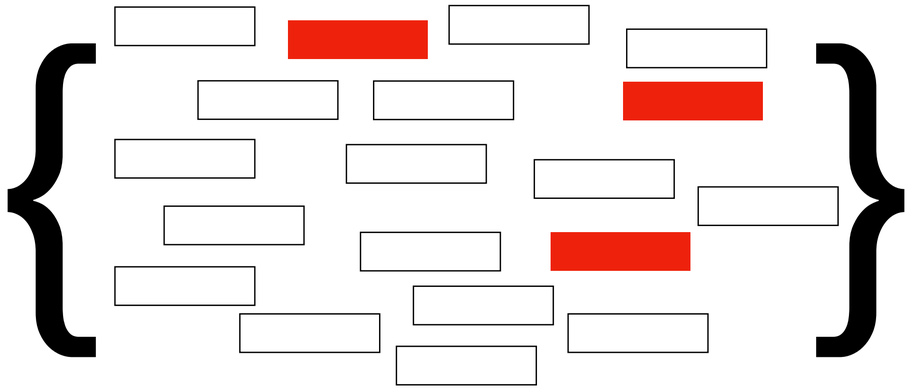
function calls?



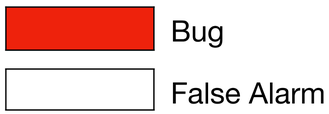
Static Analysis for SW Error Detection

	Verifier	Bug-finder
Methodology	Property-based	Pattern-based
What	General correctness properties	Specific bug patterns
How	Checking for logical properties	Searching for bug patterns
Challenge	False positives	False negatives
Solution	Relevance of alarms	Similarity of alarms
Example	**Bayesian alarm ranking system** [PLDI'18,PLDI'19,FSE'21,ICSE'22]	**Signature-based static analysis** [CCS'22]

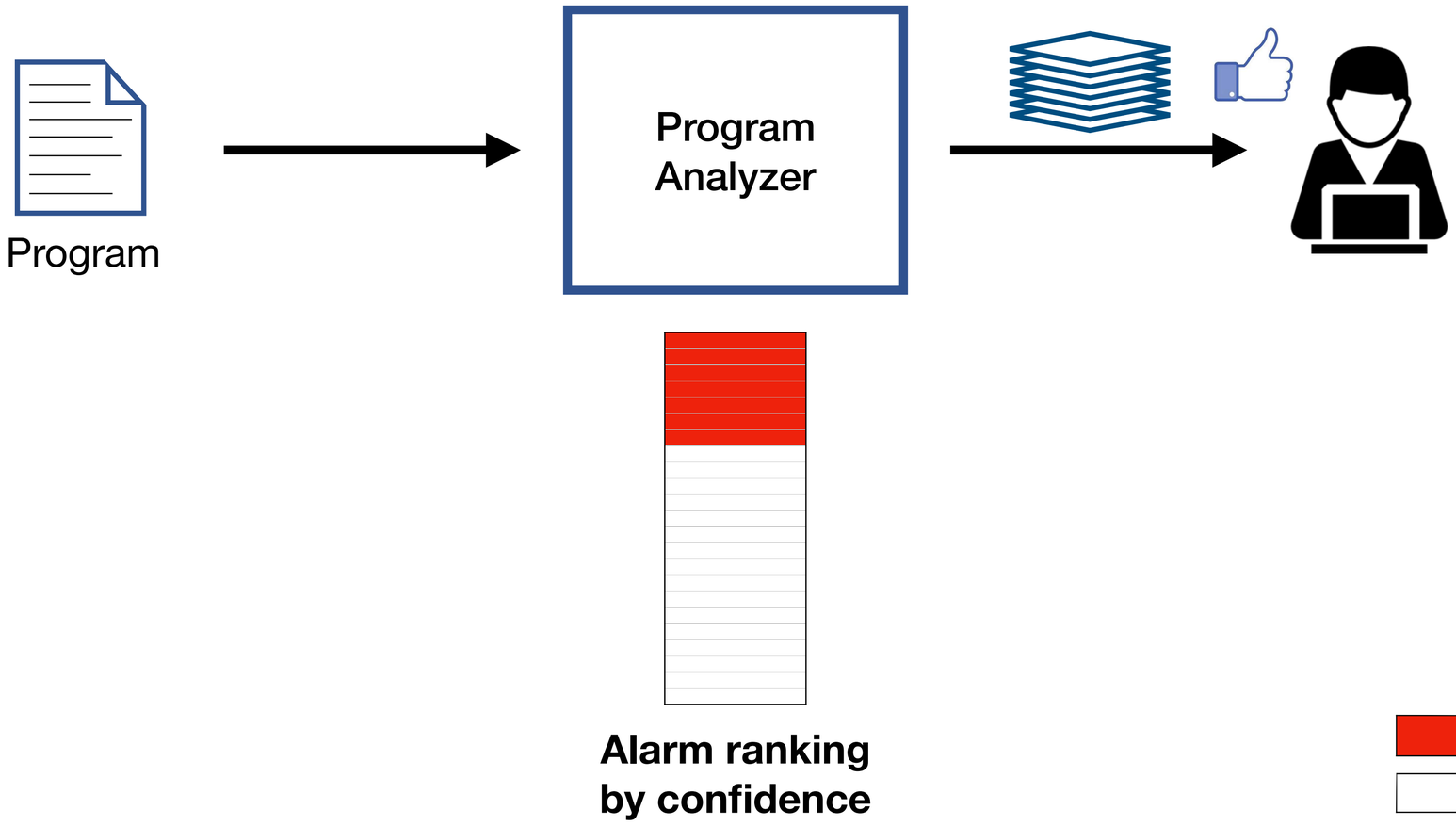
Problem: Alarm Report



Unordered set of alarms

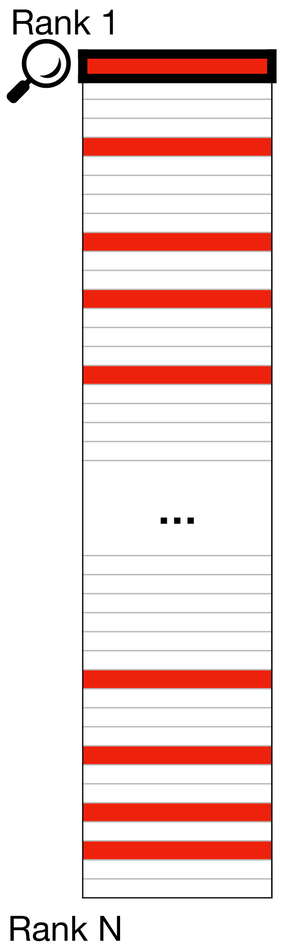


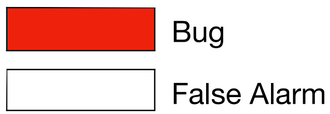
Goal


Program
Program Analyzer
Alarm ranking by confidence
Bug
False Alarm

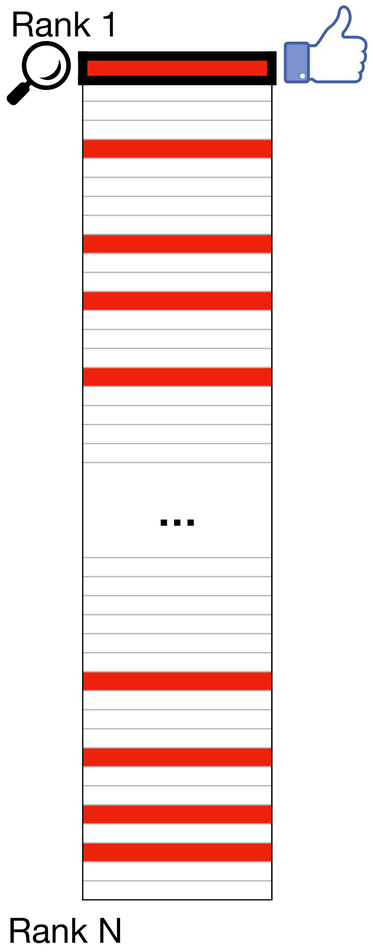
Interactive Alarm Ranking System

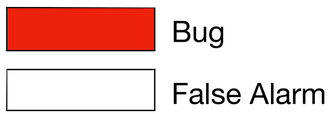
Interactive Alarm Ranking System



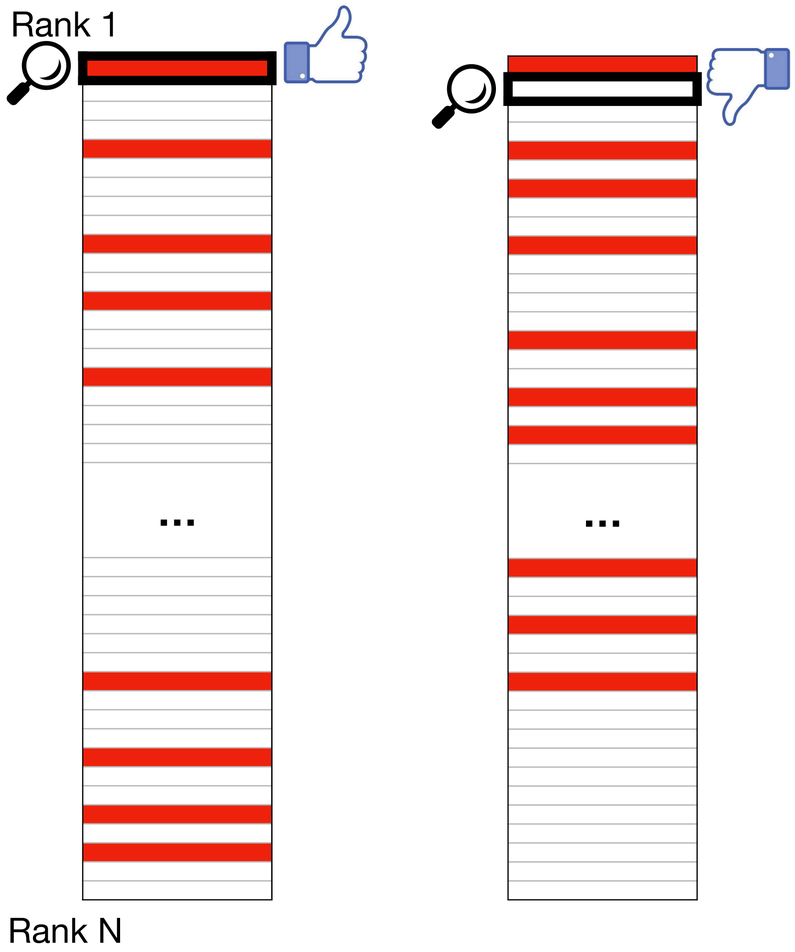


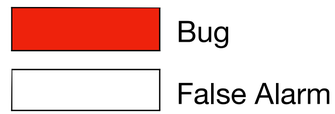
Interactive Alarm Ranking System



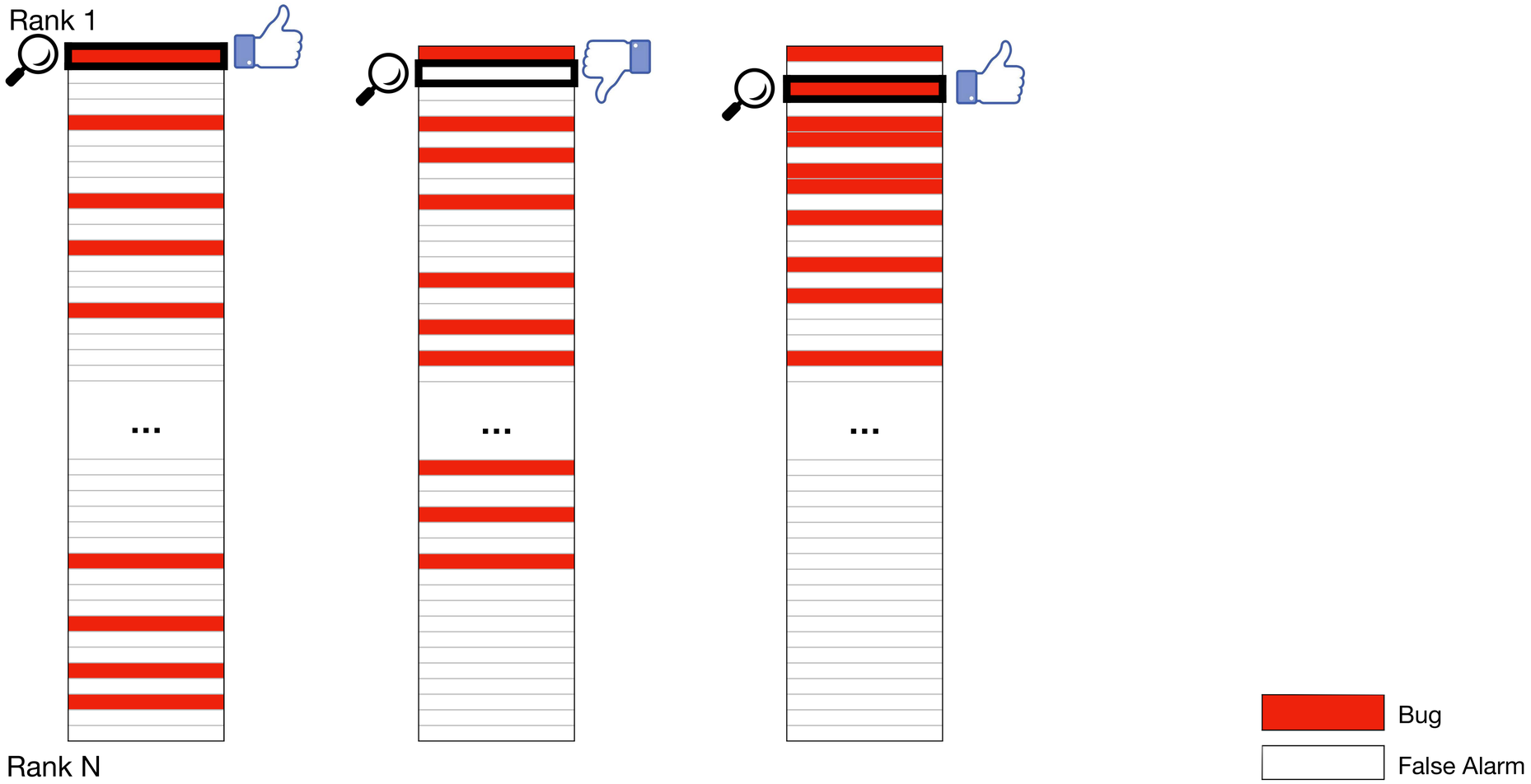


Interactive Alarm Ranking System

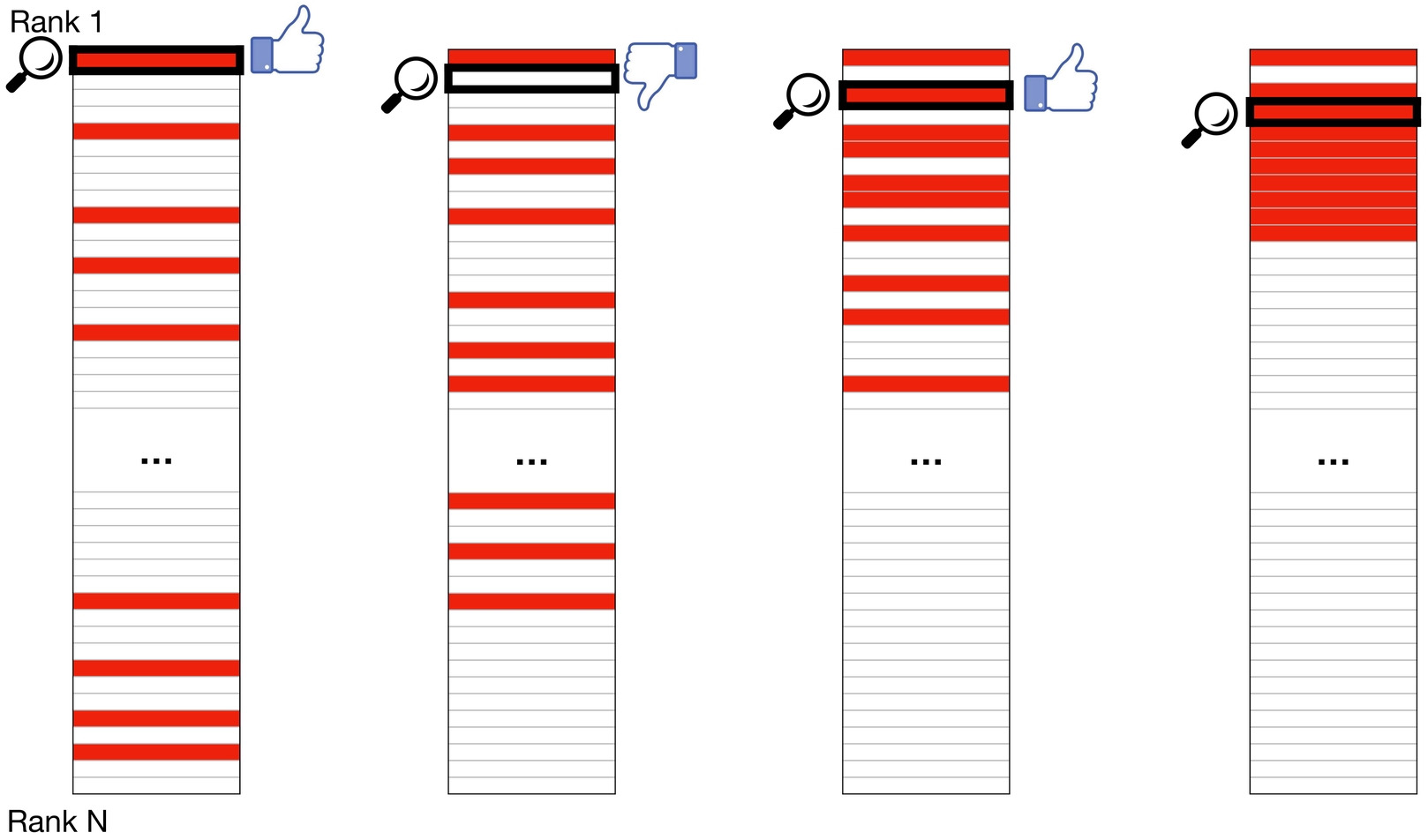




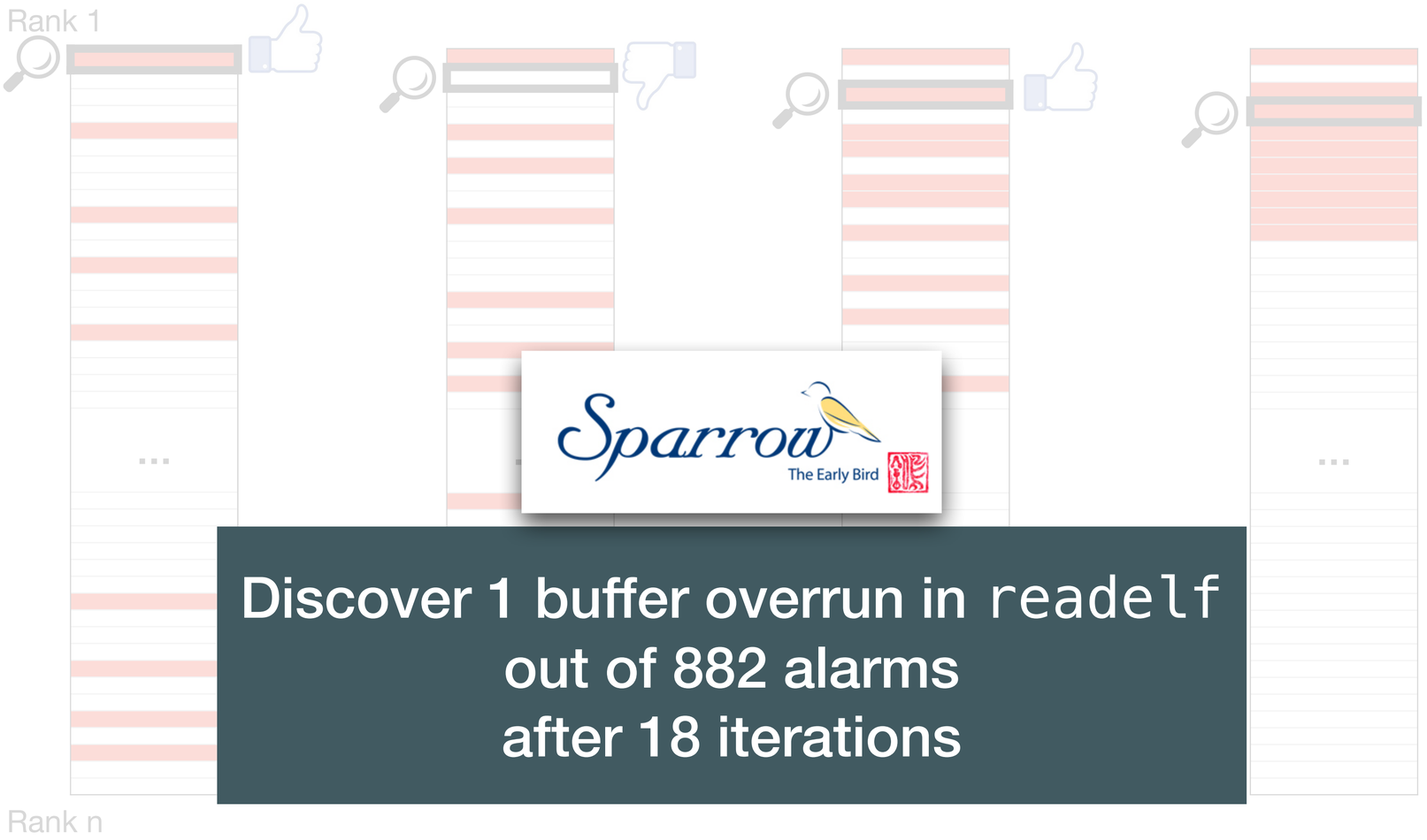
Interactive Alarm Ranking System



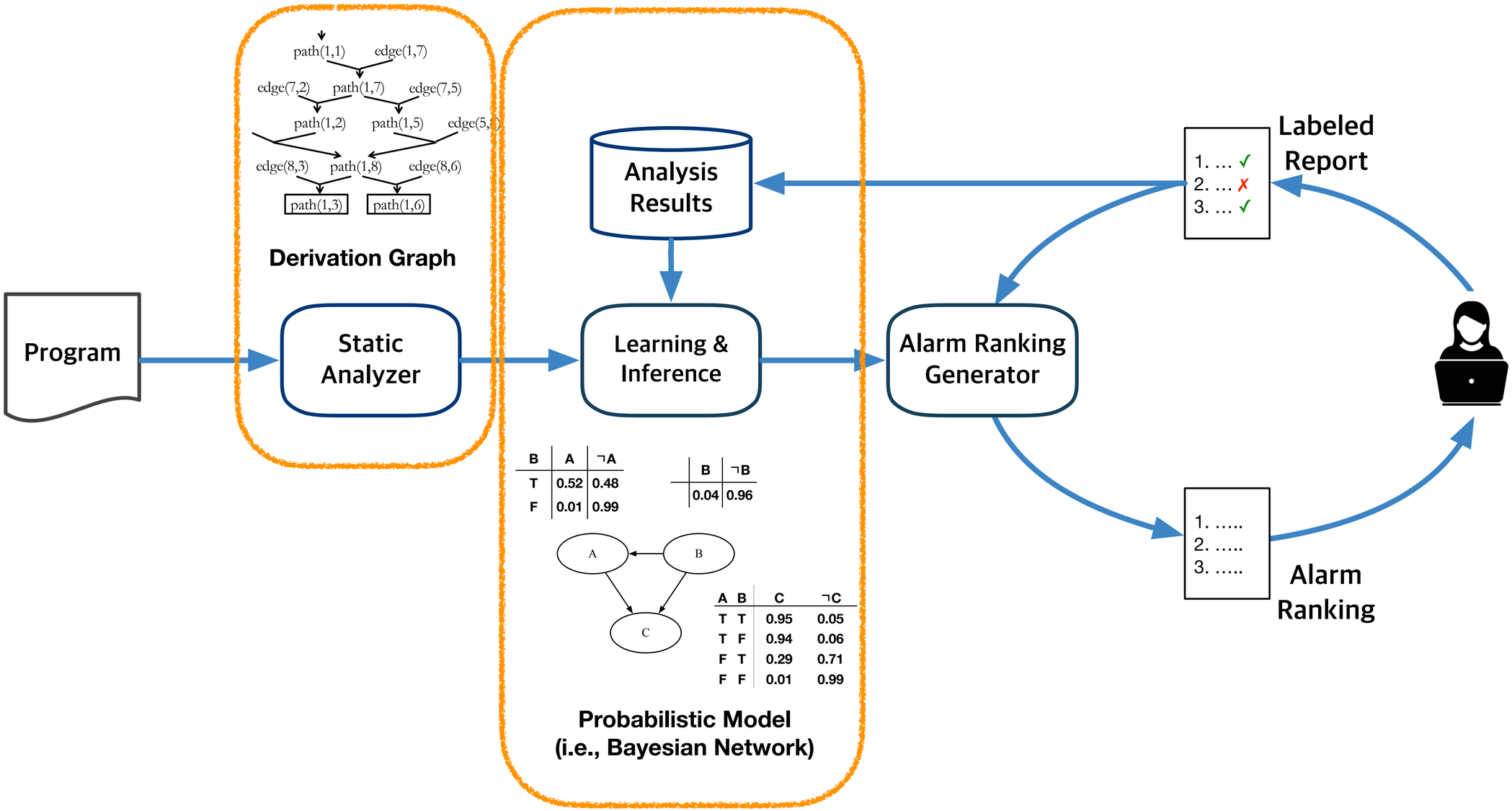
Interactive Alarm Ranking System



Interactive Alarm Ranking System



Our Solution: Bayesian Alarm Ranking System



(Approximated) Static Analysis Rules

- Static analysis as logical rules
  - Use the original rules as they are (e.g. Datalog-based analysis), or
  - Design rules to approximately describe the original analysis semantics
- Example: buffer overflow analysis approximated using def-use relations

```
1: char *line = input();
2: char *tok = strtok(line, " ");
3: char *p = tok + strlen(tok);
4: while (p > tok) {
5:   if (is_digit(*p)) break; // false alarm
6:   if (is_alpha(*p)) break; // false alarm
7:   p--;
8: }
```

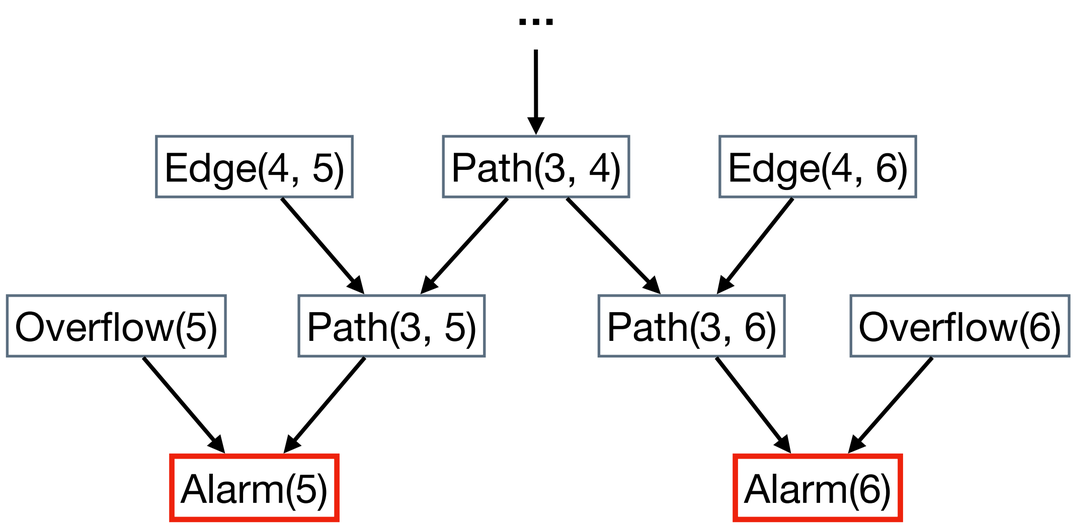
$Edge(a, b)$: immediate dataflow edge b/w a and b
$Path(a, b)$: transitive dataflow path from a to b
$Overflow(a)$: possible buffer overflow operation at a
$Alarm(a)$: alarm indicating possible buffer overflow at a

r_1: $Path(a, b)$:- $Edge(a, b)$.
r_2: $Path(a, c)$:- $Path(a, b), Edge(b, c)$.
r_3: $Alarm(b)$:- $Path(a, b), Overflow(b)$.

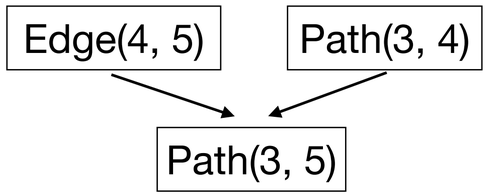
Derivation Graph

- Logical reasoning chains
  - “How is this alarm derived by the analyzer?”
  - “How is this alarm related to other alarms?”
- Example: buffer overflow analysis approximated using def-use relations

```
1: char *line = input();
2: char *tok = strtok(line, “ ”);
3: char *p = tok + strlen(tok);
4: while (p > tok) {
5:   if (is_digit(*p)) break; // false alarm
6:   if (is_alpha(*p)) break; // false alarm
7:   p--;
8: }
```



Derivation Graph to Bayesian Network



Logical Rule

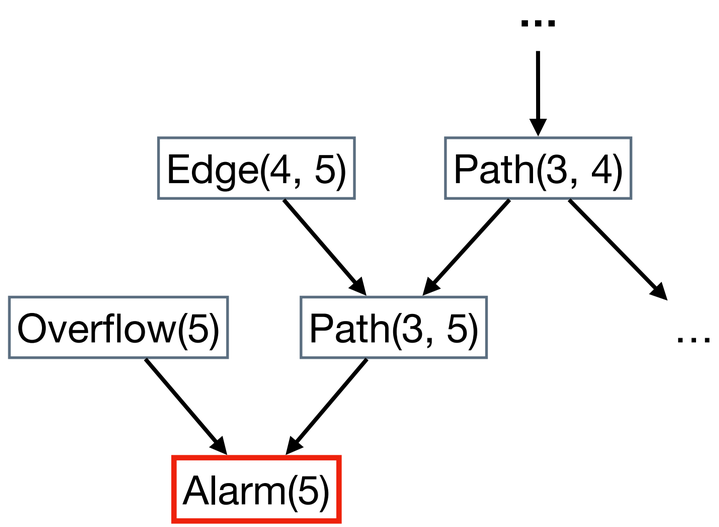
r_1: $Path(a, b)$:- $Edge(a, b)$.
r_2: $Path(a, c)$:- $Path(a, b), Edge(b, c)$.
r_3: $Alarm(b)$:- $Path(a, b), Overflow(b)$.

Probabilistic Rule

Edge(4,5)	Path(3,4)	*Pr*(Path(3,5) | …)
TRUE	TRUE	0.95*
TRUE	TRUE	0
…		
FALSE	FALSE	0

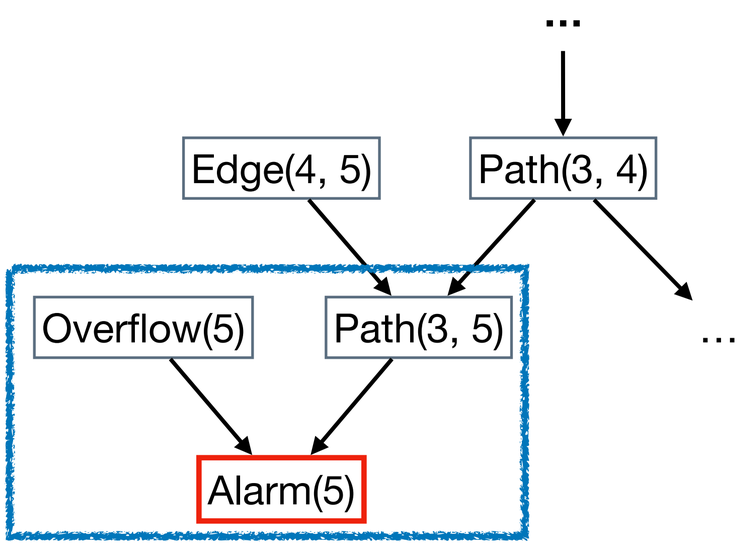
*computed by an offline learning

Probability of Alarm



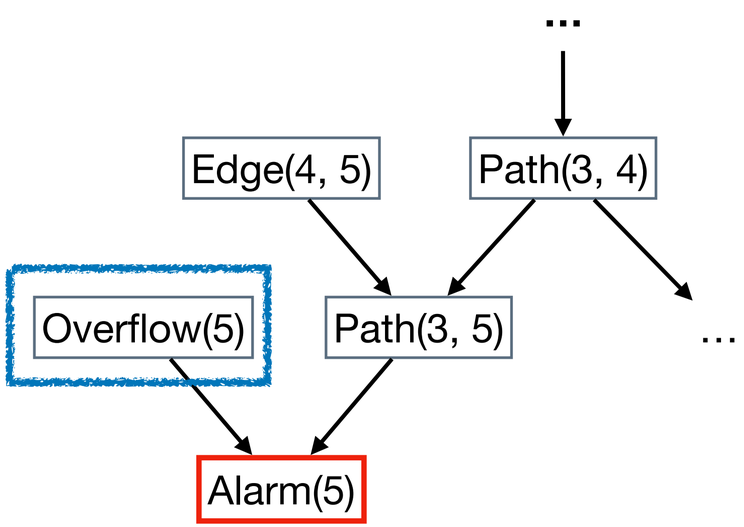
Pr(Alarm(5)) =

Probability of Alarm



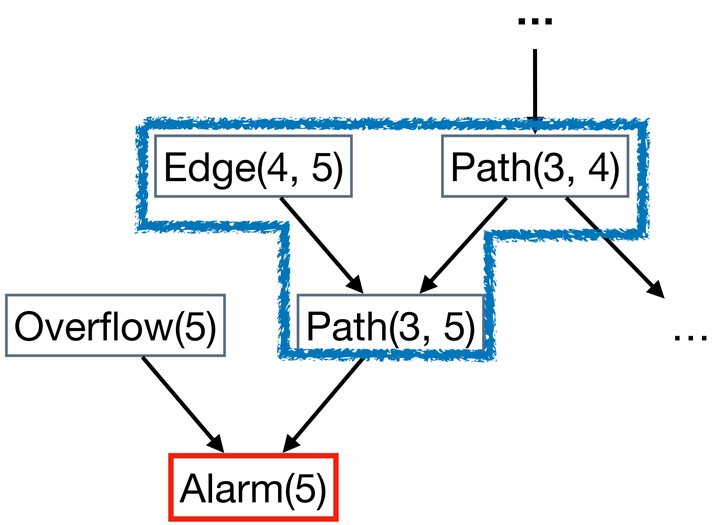
$Pr(\text{Alarm}(5)) = Pr(\text{Alarm}(5) \mid \text{Path}(3,5), \text{Overflow}(5))$ x

Probability of Alarm



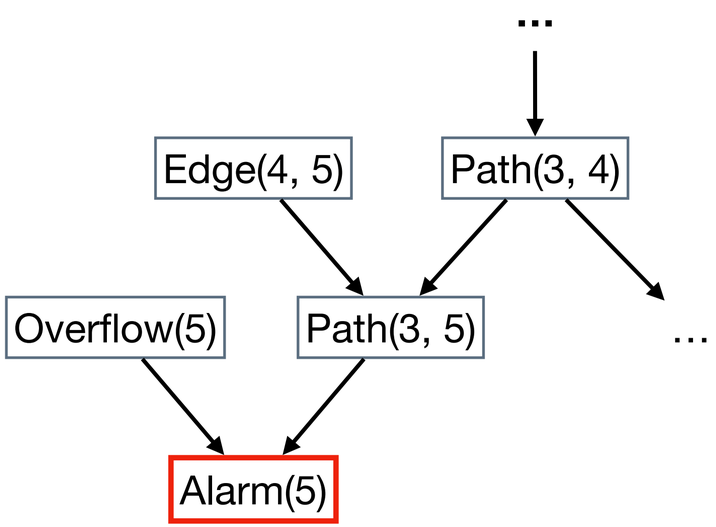
Pr(Alarm(5)) = *Pr*(Alarm(5) | Path(3,5), Overflow(5))
x *Pr*(Overflow(5))
x

Probability of Alarm



$Pr(\text{Alarm}(5)) = Pr(\text{Alarm}(5) \mid \text{Path}(3,5), \text{Overflow}(5))$
$\times Pr(\text{Overflow}(5))$
$\times Pr(\text{Path}(3,5) \mid \ldots)$

Probability of Alarm



Pr(Alarm(5)) = *Pr*(Alarm(5) | Path(3,5), Overflow(5))
x *Pr*(Overflow(5))
x *Pr*(Path(3,5) | …)
…
= 0.96

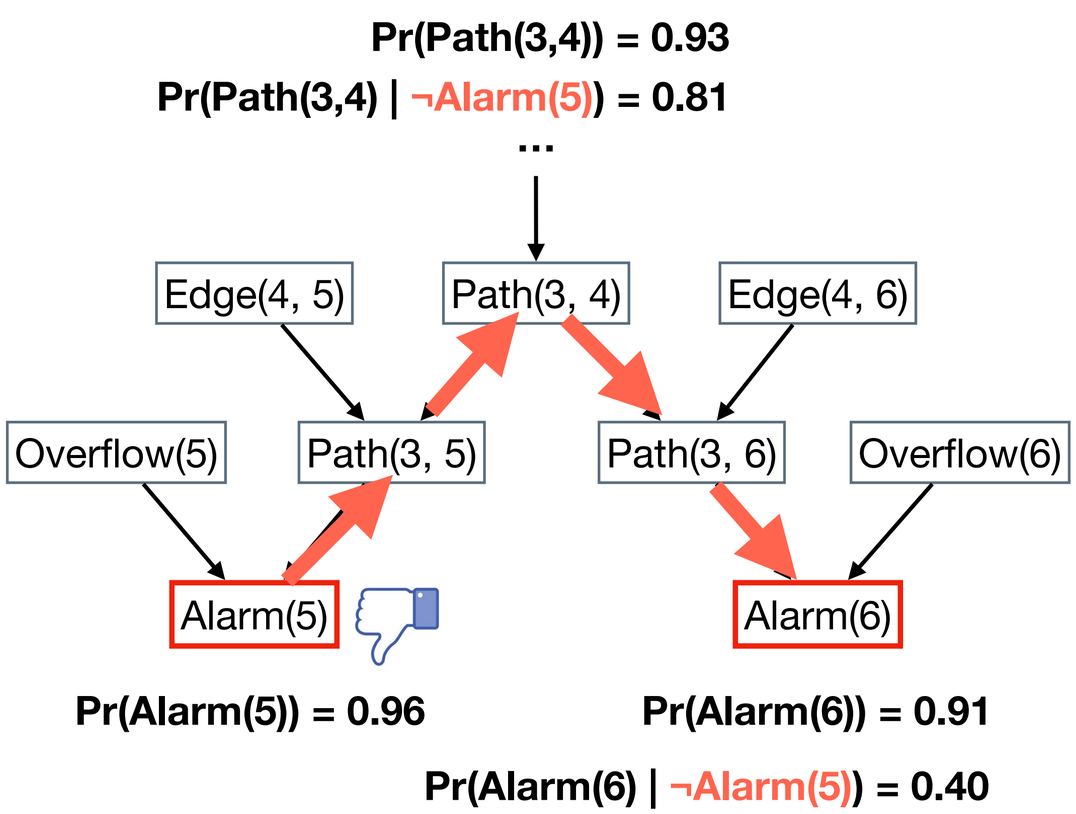
Initial Alarm Ranking

```
1: char *line = input();
2: char *tok = strtok(line, " ");
3: char *p = tok + strlen(tok);
4: while (p > tok) {
5:   if (is_digit(*p)) break; // false alarm
6:   if (is_alpha(*p)) break; // false alarm
7:   p--;
8: }
```

Ranking	Alarm	Probability
1	Alarm(5)	0.96
2	Alarm(6)	0.91
3	Alarm(24)	0.78
	…	

Q: What are the probabilities of the other alarms when Alarm(5) is false?

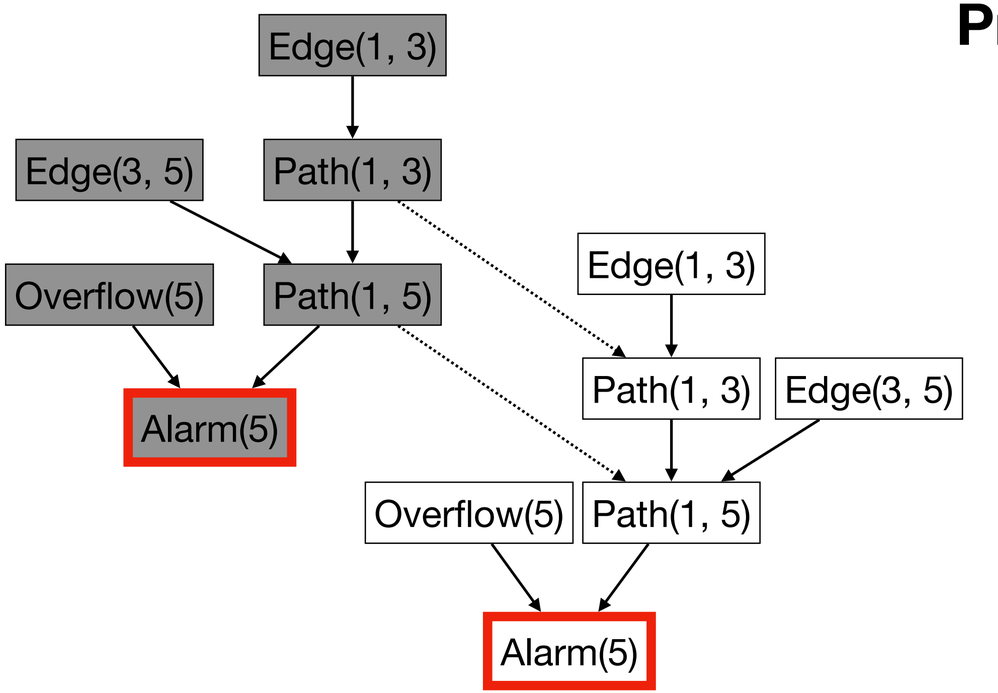
Bayesian Inference



Ranking	a	Pr(a)
1	Alarm(5)	0.96
2	Alarm(6)	0.91
3	Alarm(24)	0.78
	...	

Ranking	a	Pr(a | ¬Alarm(5))
1	Alarm(24)	0.78
	...	
54	Alarm(6)	0.40
	...	
-	Alarm(5)	0

Bayesian Framework for Static Analysis



: semantic fact from new version

: semantic fact from old version

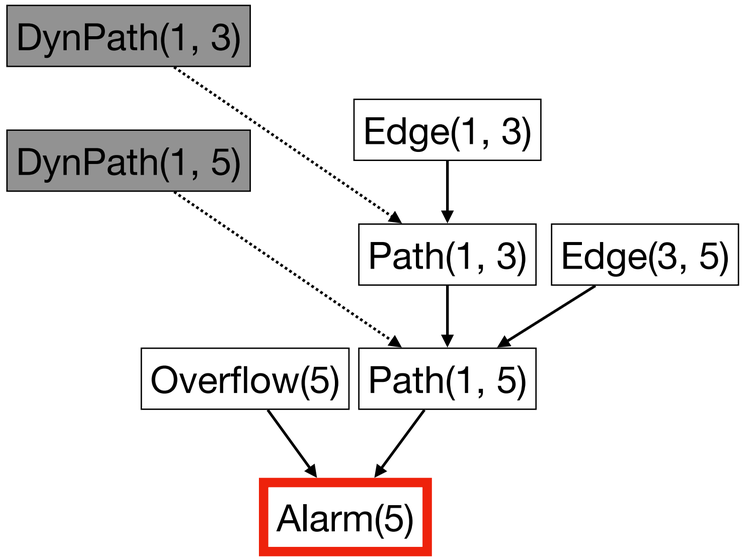
Pr(a | e)

Relevance to	Alarm Inspection Burden
User feedback [PLDI'18]	44% ↓
Code change [PLDI'19]	65% ↓

"How relevant is each alarm to this code change?"

Bayesian Framework for Static Analysis

Pr(a | e)



: semantic fact from static analysis

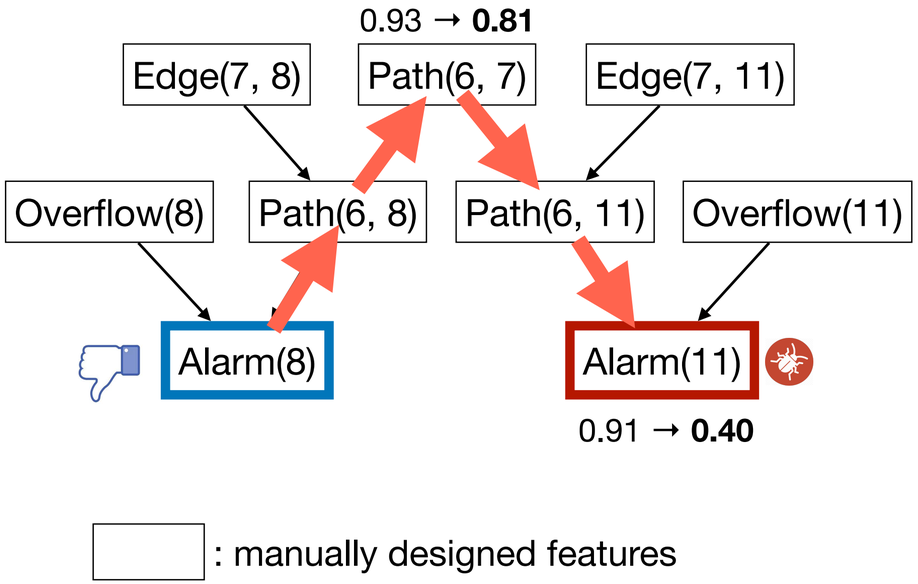
: semantic fact from dynamic analysis

Relevance to	Alarm Inspection Burden
User feedback [PLDI'18]	44% ↓
Code change [PLDI'19]	65% ↓
Dynamic analysis [FSE'21]	35% ↓

"How likely is each alarm to be true given program executions?"

Bayesian Framework for Static Analysis

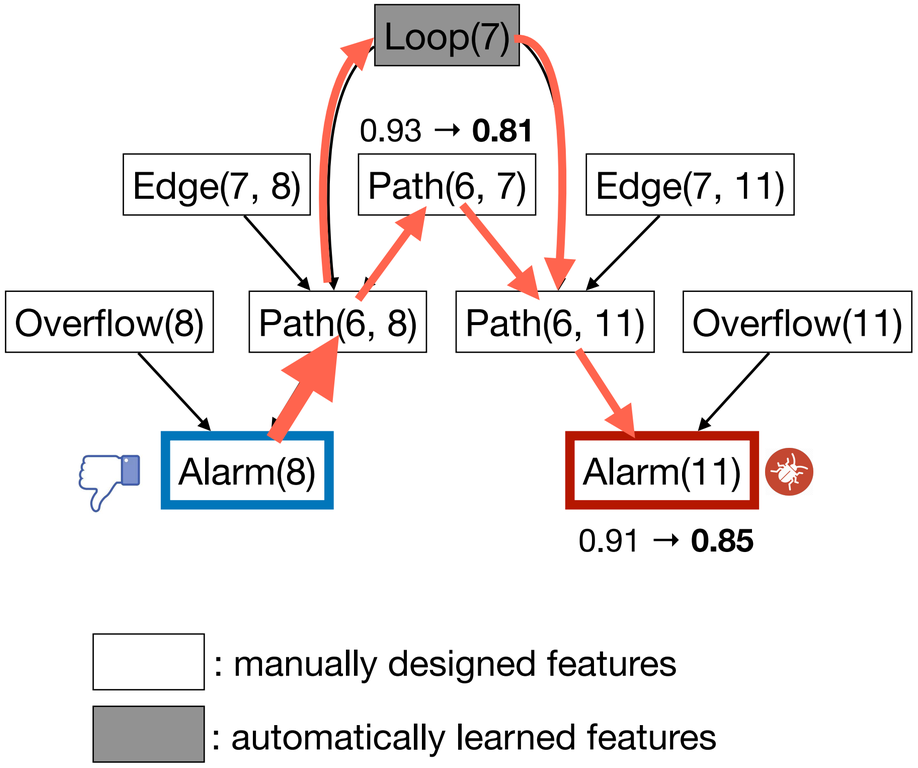
Pr(a | e)



	Alarm Inspection Burden	
Relevance to	**Manual**	**Learned [ICSE'22]**
User feedback [PLDI'18]	44% ↓	22% further ↓
Code change [PLDI'19]	65% ↓	54% further ↓
Dynamic analysis [FSE'21]	35% ↓	20% further ↓

Bayesian Framework for Static Analysis

Pr(a | e)



	Alarm Inspection Burden	
Relevance to	**Manual**	**Learned [ICSE'22]**
User feedback [PLDI'18]	44% ↓	22% further ↓
Code change [PLDI'19]	65% ↓	54% further ↓
Dynamic analysis [FSE'21]	35% ↓	20% further ↓

Static Analysis for SW Error Detection

	Verifier	Bug-finder
Methodology	Property-based	Pattern-based
What	General correctness properties	Specific bug patterns
How	Checking logical properties	Searching for bug patterns
Challenge	False positives	False negatives
Solution	Relevance of alarms	Similarity of alarms
Example	**Bayesian alarm ranking system** [PLDI'18,PLDI'19,FSE'21,ICSE'22]	**Signature-based static analysis** [CCS'22]

Recurring SW Vulnerabilities

```
long ToL (char *pbuffer) { return (puffer[0] | puffer[1]<<8 | puffer[2]<<16 | puffer[3]<<24); }

short ToS (char *pbuffer) { return ((short)(puffer[0] | puffer[1]<<8)); }

gint32 ReadBMP (gchar *name, GError **error) {
    if (fread(buffer, Bitmap_File_Head.biSize - 4, fd) != 0)
        FATALP ("BMP: Error reading BMP file header #3");
    Bitmap_Head.biWidth = ToL (&buffer[0x00]);
    Bitmap_Head.biBitCnt = ToS (&buffer[0x0A]);

    rowbytes = ((Bitmap_Head.biWidth * Bitmap_Head.biBitCnt - 1) / 32) * 4 + 4;
    image_ID = ReadImage (rowbytes);
    ...
}

gint32 ReadImage (int rowbytes) {
    buffer = malloc(rowbytes);      // malloc with overflowed size
    ...
}
```

gimp-2.6.7 (CVE-2009-1570)

Recurring SW Vulnerabilities

```
long ToL (char *pbuffer) { return (puffer[0] | puffer[1]<<8 | puffer[2]<<16 | puffer[3]<<24); }

short ToS (char *pbuffer) { return ((short)(puffer[0] | puffer[1]<<8)); }

gint32 ReadBMP (gchar *name, GError **error) {
    if (fread(buffer, Bitmap_File_Head.biSize - 4, fd) != 0)
        FATALP
    Bitmap_Hea
    Bitmap_Hea

    rowbytes =
    image_ID =
    ...
}

gint32 ReadIma
    buffer = m
    ...
}
```

```
long ToL (char *pbuffer) { return (puffer[0] | puffer[1]<<8 | puffer[2]<<16 | puffer[3]<<24); }

short ToS (char *pbuffer) { return ((short)(puffer[0] | puffer[1]<<8)); }

bitmap_type bmp_load_image (FILE* filename) {
    if (fread(buffer, Bitmap_File_Head.biSize - 4, fd) != 0)
        FATALP ("BMP: Error reading BMP file header #3");
    Bitmap_Head.biWidth = ToL (&buffer[0x00]);
    Bitmap_Head.biBitCnt = ToS (&buffer[0x0A]);

    rowbytes = ((Bitmap_Head.biWidth * Bitmap_Head.biBitCnt - 1) / 32) * 4 + 4;
    image.bitmap = ReadImage (rowbytes);
    ...
}

unsigned char* ReadImage (int rowbytes) {
    unsigned char *buffer = (unsigned char*) new char[rowbytes];    // malloc with overflowed size
    ...
}
```

sam2p-0.49.4 (CVE-2017-1570)

Recurring SW Vulnerabilities

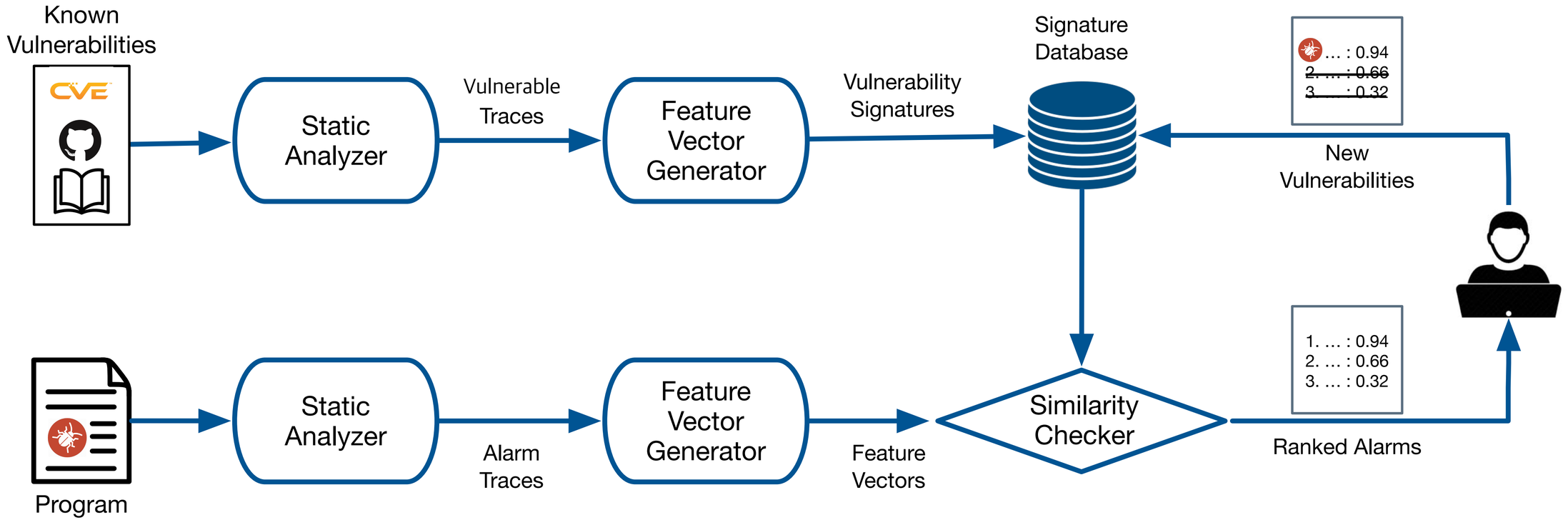
```
long ToL (char *pbuffer) { return (puffer[0] | puffer[1]<<8 | puffer[2]<<16 | puffer[3]<<24); }

short ToS (char *pbuffer) { return ((short)(puffer[0] | puffer[1]<<8)); }

gint32 ReadBMP (gchar *name, GError **error) {
    if (fread(
        FATALP
    Bitmap_Hea
    Bitmap_Hea

    rowbytes =
    image_ID =
    ...
}

gint32 ReadIma
    buffer = m
    ...
}
```

```
long ToL (char *pbuff

short ToS (char *pbuf

bitmap_type bmp_load_
    if (fread(buffer,
        FATALP ("BMP:
    Bitmap_Head.biWid
    Bitmap_Head.biBit

    rowbytes = ((Bitma
    image.bitmap = Rea
    ...
}

unsigned char* ReadIma
    unsigned char *bu
    ...
}
```

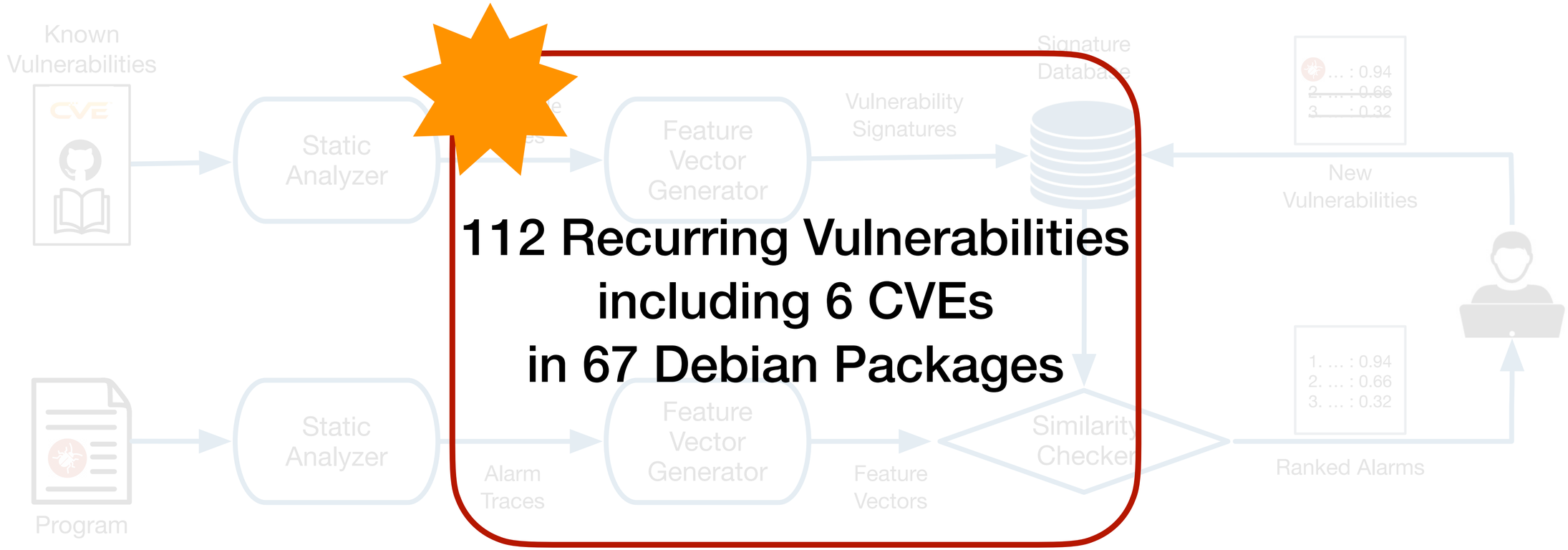
```
static XcursorBool _XcursorReadUInt (XcursorFile *file, XcursorUInt *u) {
  unsigned char bytes[4];
  if ((*file->read)(file, bytes, 4) != 4) return XcursorFalse;
  *u = ((bytes[0] << 0) | (bytes[1] << 8) | (bytes[2] << 16) | (bytes[3] << 24));
  return XcursorTrue;
}

_XcursorReadImage (XcursorFile *file, XcursorFileHeader    *fileHeader, int toc) {
  XcursorChunkHeader  chunkHeader;
  XcursorImage head;

  ...
 if (!_XcursorReadUInt (file, &head.width))
    return NULL;
  if (!_XcursorReadUInt (file, &head.height))
    return NULL;
  image = XcursorImageCreate(head.width, head.height);
  ....
}

XcursorImage *XcursorImageCreate (int width, int height) {
  image = malloc (sizeof (XcursorImage) + width * height * sizeof (XcursorPixel));
  ...
}
```

libXcursor-1.1.14 (CVE-2017-16612)

Our Solution: Signature-based Static Analysis



Our Solution: Signature-based Static Analysis

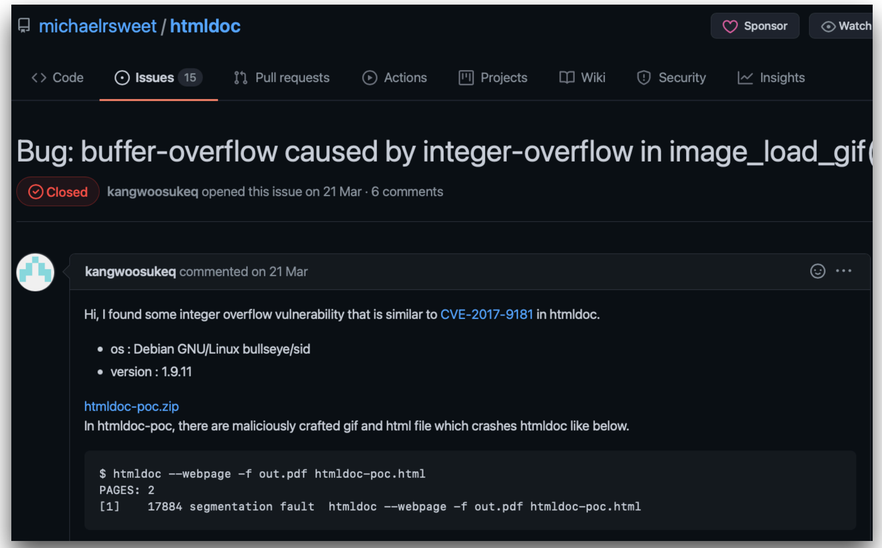


Example

Similarity to CVE-2017-9181 (score: 0.92)

CVE

```
static int image_load_gif(…) {
    ...
    fread(buf, 13, 1, fp);
    while (1) {
        img->width = (buf[5] << 8) | buf[4];
        img->height = (buf[7] << 8) | buf[6];
        img->depth = gray ? 1 : 3;
        img->pixels = (uchar *)malloc((size_t)(img->width * img->height * img->depth));
        ...
     }
}
```



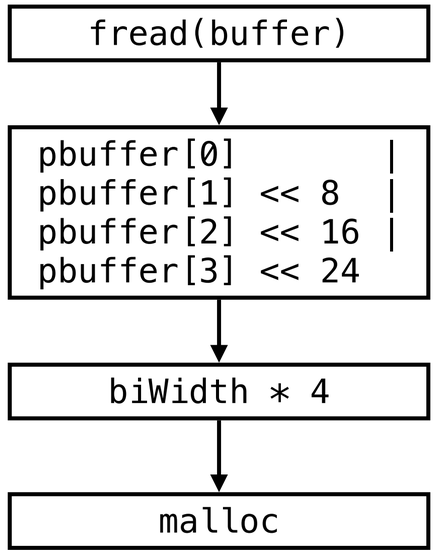
Static Analysis

- Taint analysis: tracking flows of malicious data
  - Source points: user inputs (e.g., fread, gets)
  - Sink points: security-sensitive points (e.g., malloc, printf)
- 7 checkers on top of Facebook Infer
  - Custom: int overflow, int underflow, buffer overflow, format string bug, cmd injection
  - Out-of-the-box: use-after-free, double free

Extracting Vulnerable Traces

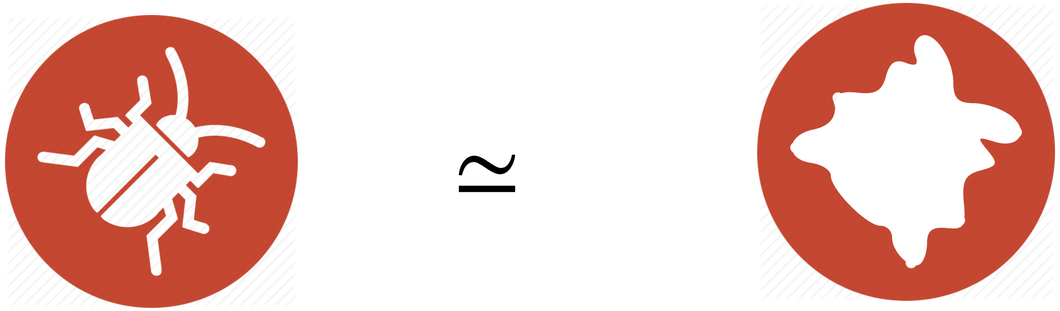
- Vulnerable trace: a list of commands related to the vulnerabilities
- Trace on data dependency extracted by static analysis

```
long ToL (char *pbuffer) {
    return (puffer[0] | puffer[1]<<8 | puffer[2]<<16 | puffer[3]<<24);
}

gint32 ReadBMP (gchar *name, GError **error) {
    if (fread(buffer, Bitmap_File_Head.biSize - 4, fd) != 0)
        FATALP ("BMP: Error reading BMP file header #3");
    biWidth = ToL (&buffer[0x00]);
    ...
    rowbytes = biWidth * 4;
    image_ID = ReadImage (rowbytes);
    ...
}

gint32 ReadImage (int rowbytes) {
    buffer = malloc(rowbytes);
    ...
}
```



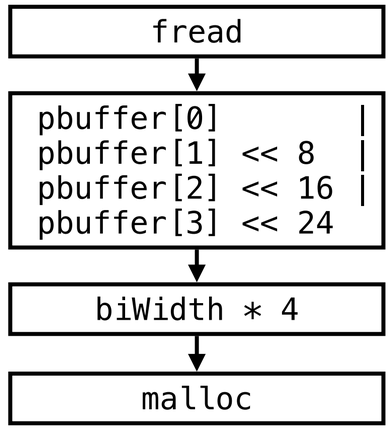
Similarity

- Comparison between known vulnerability traces and potential vulnerability traces
- Each trace is encoded as a feature vector
- Similarity of two vulnerabilities = Similarity of two vectors



Feature Vector

- Two categories
  - low-level: # basic operators and libraries on the trace
  - high-level: # common sanity-checking code idioms



Feat	#
fread	1
<<	3
|	3
*	1
malloc	1

Checking upper bound

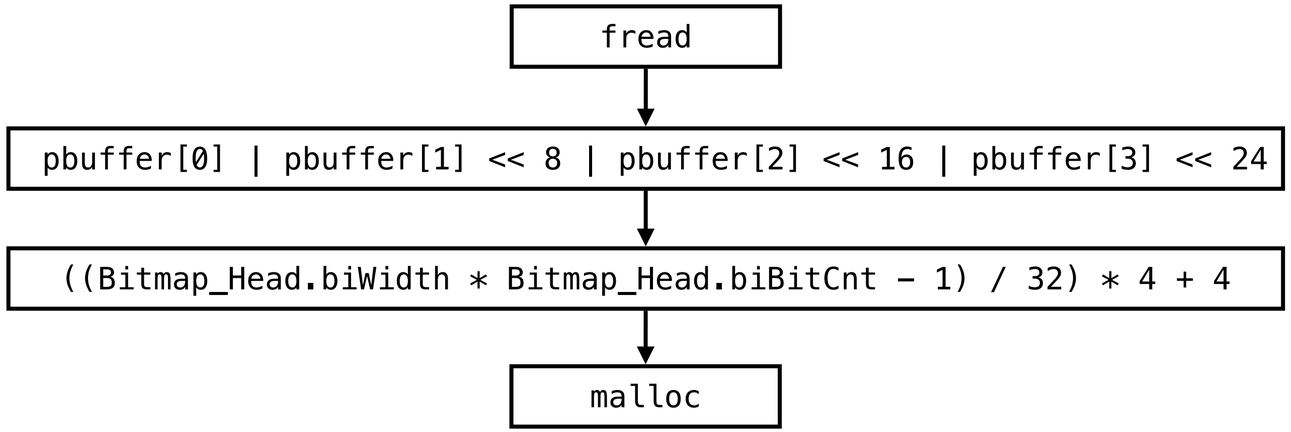
```
if (n > UPPER_BOUND) {
    …
}
```

Checking format string

```
if (ch == ‘%’) {
    …
}
```

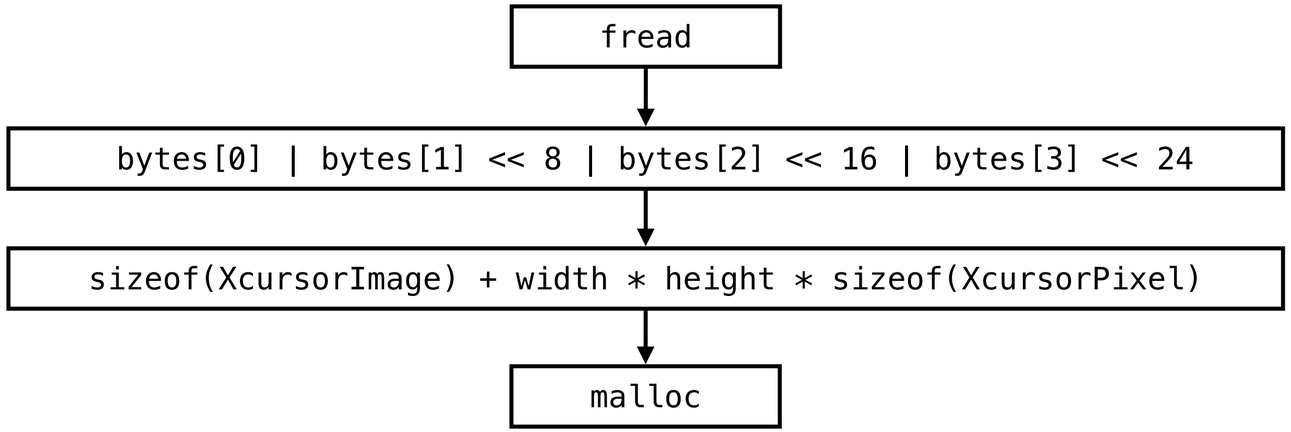
Example: gimp-2.6.7

```
long ToL (char *pbuffer) {
  return (puffer[0] | puffer[1]<<8 | puffer[2]<<16 | puffer[3]<<24);
}

short ToS (char *pbuffer) { return ((short)(puffer[0] | puffer[1]<<8)); }

gint32 ReadBMP (gchar *name, GError **error) {
  if (fread(buffer, Bitmap_File_Head.biSize - 4, fd) != 0)
    FATALP ("BMP: Error reading BMP file header #3");
  Bitmap_Head.biWidth = ToL (&buffer[0x00]);
  Bitmap_Head.biBitCnt = ToS (&buffer[0x0A]);

  rowbytes = ((Bitmap_Head.biWidth * Bitmap_Head.biBitCnt - 1) / 32) * 4 + 4;
  image_ID = ReadImage (rowbytes);
  ...
}

gint32 ReadImage (int rowbytes) {
  buffer = malloc(rowbytes);      // malloc with overflowed size
  ...
}
```



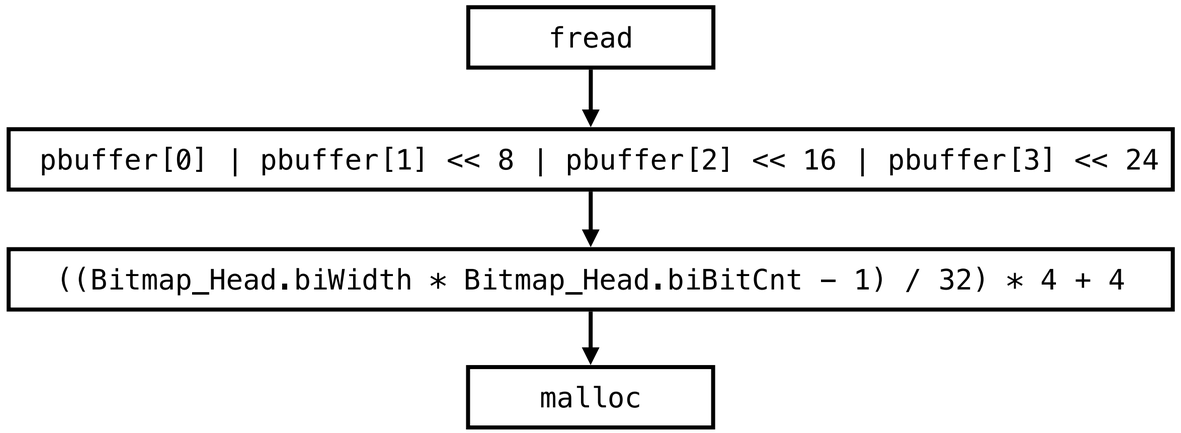
Example: libXcursor-1.1.14

```
static XcursorBool _XcursorReadUInt (XcursorFile *file, XcursorUInt *u) {
  unsigned char bytes[4];
  if ((*file->read)(file, bytes, 4) != 4) return XcursorFalse;
  *u = ((bytes[0] << 0) | (bytes[1] << 8) | (bytes[2] << 16) | (bytes[3] << 24));
  return XcursorTrue;
}

_XcursorReadImage (XcursorFile *file, XcursorFileHeader    *fileHeader, int toc) {
  XcursorChunkHeader  chunkHeader;
  XcursorImage head;

  ...
 if (!_XcursorReadUInt (file, &head.width))
    return NULL;
  if (!_XcursorReadUInt (file, &head.height))
    return NULL;
  image = XcursorImageCreate(head.width, head.height);
  ....
}

XcursorImage *XcursorImageCreate (int width, int height) {
  image = malloc (sizeof (XcursorImage) + width * height * sizeof (XcursorPixel));
  ...
}
```



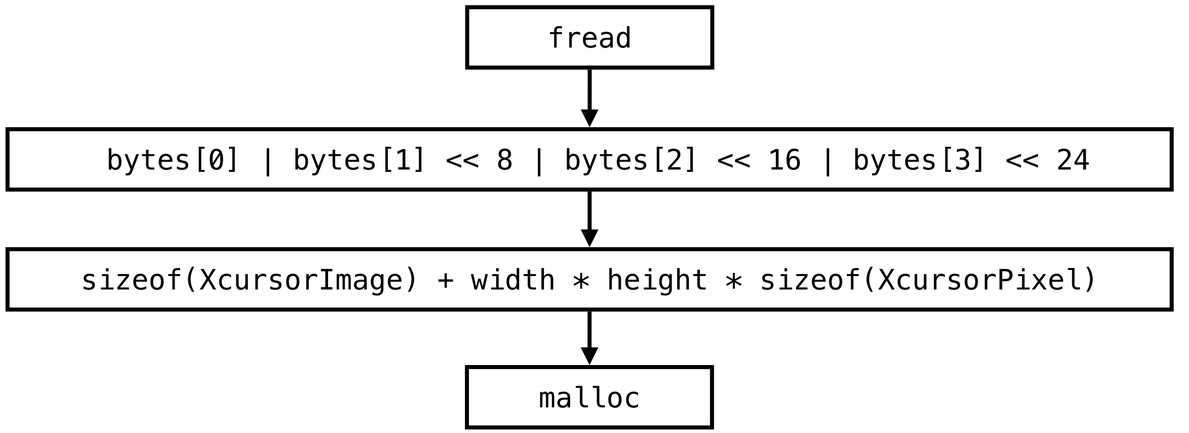
Vector Representation

gimp



fread	|	<<	*	-	/	+	malloc
1	3	3	2	1	1	1	1

libXcursor



fread	|	<<	*	-	/	+	malloc
1	3	3	2	0	0	1	1

Similarity Comparison

- Cosine similarity of two vectors

$$sim(A, B) = \frac{A \cdot B}{\| A \| \| B \|}$$

Signature vulnerability

fread	|	<<	*	-	/	+	malloc
1	3	3	2	1	1	1	1

Potential vulnerability

fread	|	<<	*	-	/	+	malloc
1	3	3	2	0	0	1	1

$$\frac{1 \times 1 + 3 \times 3 + 3 \times 3 + 2 \times 2 + 1 \times 1 + 1 \times 1}{\sqrt{1^2 + 3^2 + 3^2 + 2^2 + 1^2 + 1^2 + 1^2 + 1^2}\sqrt{1^2 + 3^2 + 3^2 + 2^2 + 1^2 + 1^2}} \cong 0.96$$

Evaluation

- 273 Debian C/C++ packages
- 7 bug types
  - cmd injection, int over/underflow, format string, buffer overflow, UAF, double free
- Signature DB
  - 16 publicly known vulnerabilities
  - 5,383 examples in Juliet test suite (collection of buggy programs by NSA)
  - 5 examples in OWASP (secure coding guidelines)
- Implemented on top of Facebook Infer

Performance

112 new vulnerabilities including 6 CVEs in 67 packages

Score	Precision
>0.95	87.5%
>0.90	85.7%
>0.85	78.1%
<0.85	37.0%

```
void CWE190_Integer_Overflow__int_64_t_fscanf_square_01_bad() {
  int64_t data;
  data = 0LL;
  fscanf(stdin, "%" SCNd64, &data);
  // potential integer overflow
  int64_t result = data * data;
  char *p = malloc(result);
}
```

Signature: Juliet test case

Similarity score by Tracer: 1.0

```
static DiaObject *fig_read_polyline(FILE *file, DiaContext *ctx) {
  fscanf(file, "%d %d %d %d %d %d %d %lf %d %d %d %d %d %d\n", ...,
         &npoints);
  newobj = create_standard_polyline(npoints, ...);
  ...;
}

DiaObject *create_standard_polyline(int num_points, ...) {
  pcd.num_points = num_points;
  new_obj = otype->ops->create(NULL, &pcd, &h1, &h2);
  ...;
}

static DiaObject *polyline_create(Point *startpoint, void *user_data,
                                  Handle **handle1, Handle **handle2) {
  MultipointCreateData *pcd = (MultipointCreateData *)user_data;
  polyconn_init(poly, pcd->num_points);
  ...;
}

void polyconn_init(PolyConn *poly, int num_points) {
  // potential integer overflow
  poly->points = g_malloc(num_points * sizeof(Point));
  ...;
}
```

Target: dia-0.97.3

Conclusion

- AI-based program analysis through relevance and similarity
- Relevance of alarms to user feedback, code change, dynamic analysis, etc
  - Reduce user's alarm insertion burden
  - Enabled by Bayesian alarm ranking framework
- Similarity of alarms to known bugs
  - Reduce user's pattern encoding burden
  - Enabled by static analysis, vulnerability DB and similarity checking
- Future work: more challenges in static analysis using LLM