TRACER:
Signature-based Static Analysis for Detecting Recurring Vulnerabilities

Wooseok Kang, Byoungho Son, Kihong Heo

SOFTWARE VULNERABILITIES RECUR...

Deja vu-lnerability (Google Project Zero)

The most notable fact is that 25% of the 0-days detected in 2020

are closely related to previously publicly disclosed vulnerabilities.

2021/02/03

The 0-days we saw in 2021 generally followed the same bug patterns,

attack surfaces, and exploit “shapes” previously seen in public research.

2022/04/19

gimp

(CVE-2009-1570)

8 years later...

```
gint32 ReadBMP(const gchar *name, ...) {
    ...
    filename = name;
    fd = fopen(filename, "rb");

1   if (!(fread(buffer, Bitmap_File_Head.biSize - 4, 1, fd) != 0)) {
        return -1;
    }

2   Bitmap_Head.biWidth = ToL(&buffer[0x00]);
    Bitmap_Head.biBitCnt = ToS(&buffer[0x0A]);

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

4   image_ID = ReadImage(..., rowbytes, ...);
    ...
}
```

➜

```
static gint32 ToL(const guchar *puffer) {
    return (puffer[0] | puffer[1] << 8 | puffer[2] << 16 | puffer[3] << 24);
}
```

➜

```
static gint16 ToS(const guchar *puffer) { return (puffer[0] | puffer[1] << 8); }
```

➜

```
static gint32 ReadImage(..., gint rowbytes, ...) {
    ...
    buffer = malloc(rowbytes);
    ...
}
```

sam2p

(CVE-2017-1663)

```
bitmap_type bmp_load_image(at_string filename) {
    ...
    fd = fopen(filename, "rb");

1   if (!(fread(buffer, Bitmap_File_Head.biSize - 4, 1, fd) != 0))
        FATALP("BMP: Error reading BMP file header #3");

2   Bitmap_Head.biWidth = ToL(&buffer[0x00]);
    Bitmap_Head.biBitCnt = ToS(&buffer[0x0A]);

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

4   image.bitmap = ReadImage(..., rowbytes, ...);
    ...
}

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

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

static unsigned char *ReadImage(..., int rowbytes, ...) {
    ...
    buffer = (unsigned char *)new char[rowbytes];
    ...
}
```

```
gint32 ReadBMP(const gchar *name, ...) {
    ...
    filename = name;
    fd = fopen(filename, "rb");

    if (!(fread(buffer, Bitmap_File_Head.biSize - 4, 1, fd) != 0)) {
        return -1;
    }

    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, ...);
    ...
}


static gint32 ToL(const guchar *puffer) {
    return (puffer[0] | puffer[1] << 8 | puffer[2] << 16 | puffer[3] << 24);
}


static gint16 ToS(const guchar *puffer) { return (puffer[0] | puffer[1] << 8); }


static gint32 ReadImage(..., gint rowbytes, ...) {
    ...
    buffer = malloc(rowbytes);
    ...
}
```

gimp (CVE-2009-1570)

```
bitmap_type bmp_load_image(at_string filename) {
    ...
    fd = fopen(filename, "rb");

    if (!(fread(buffer, Bitmap_File_Head.biSize - 4, 1, 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, ...);
    ...
}


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


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


static unsigned char *ReadImage(..., int rowbytes, ...) {
    ...
    buffer = (unsigned char *)new char[rowbytes];
    ...
}
```

sam2p (CVE-2017-1663)

libXcursor

(CVE-2017-16612)

```
static XcursorImage *_XcursorReadImage(XcursorFile *file, ...) {
    XcursorImage head;

1   if (!_XcursorReadUInt(file, &head.width)) return NULL;
    if (!_XcursorReadUInt(file, &head.height)) return NULL;

3   image = XcursorImageCreate(head.width, head.height);
    ...
}


static XcursorBool _XcursorReadUInt(XcursorFile *file, XcursorUInt *u) {
    unsigned char bytes[4];

2   if (fread(bytes, 1, 4, file) != 4) return XcursorFalse;

    *u = ((bytes[0] << 0) | (bytes[1] << 8) | (bytes[2] << 16) |
          (bytes[3] << 24));
    return XcursorTrue;
}


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

Why?

1. Source code copy & paste

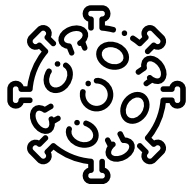
   ↪ **Syntactic Similarity**

2. Similar mistakes when implementing the same logic

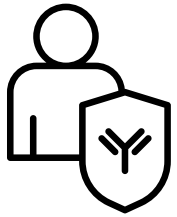
   ↪ **Semantic Similarity**

Goal: Build a Software Immune System

Immune System



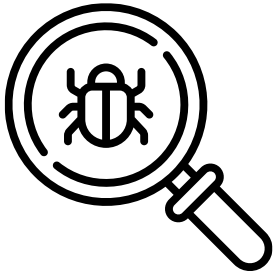
virus penetration



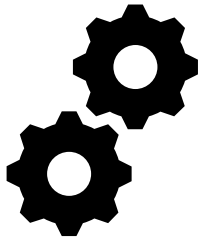
antibody production

protection against the same virus

Software Immune System

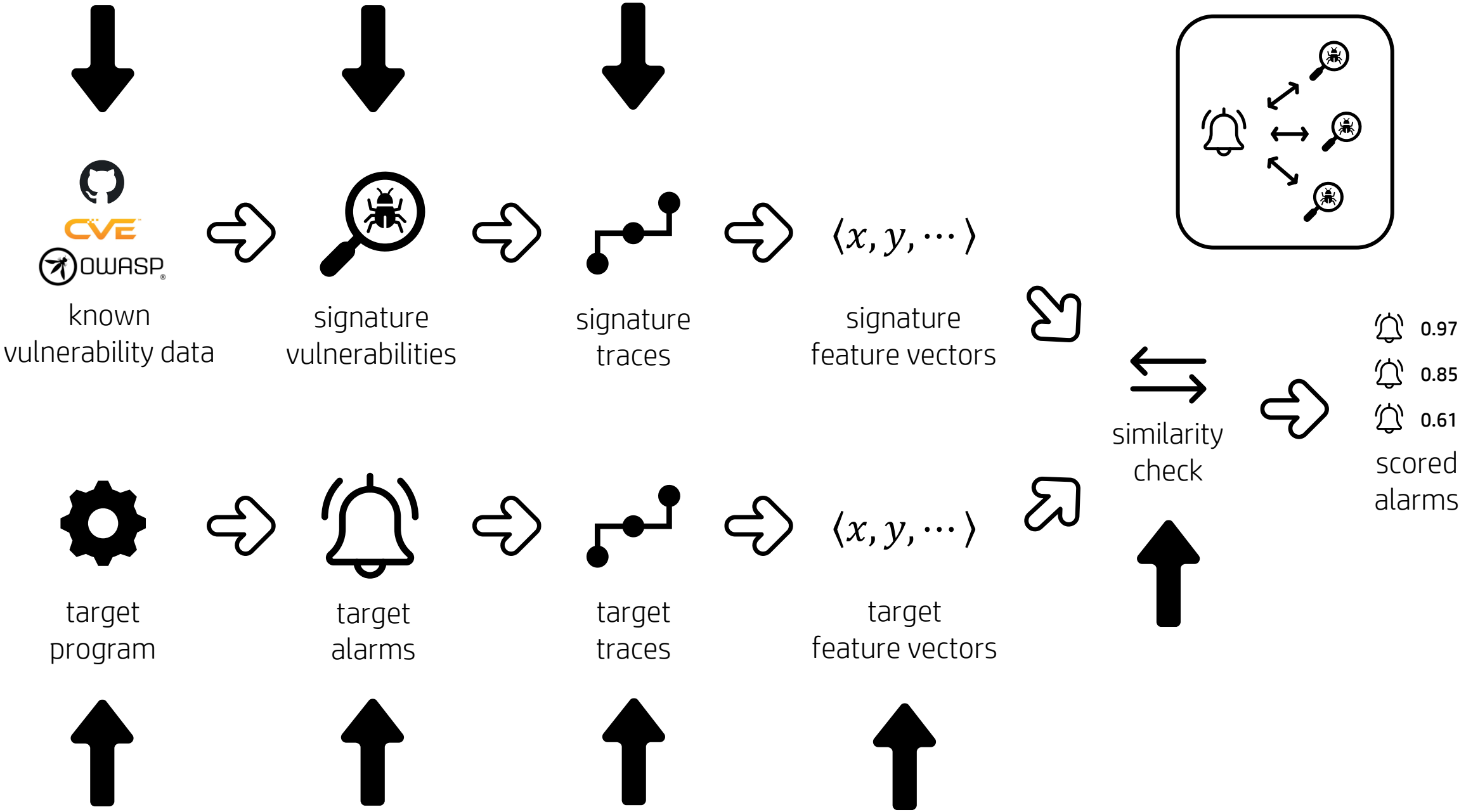


known vulnerability



analysis system

no recurring vulnerability


known
vulnerability data
signature
vulnerabilities
signature
traces
$\langle x, y, \cdots \rangle$
signature
feature vectors
similarity
check
0.97
0.85
0.61
scored
alarms
target
program
target
alarms
target
traces
$\langle x, y, \cdots \rangle$
target
feature vectors

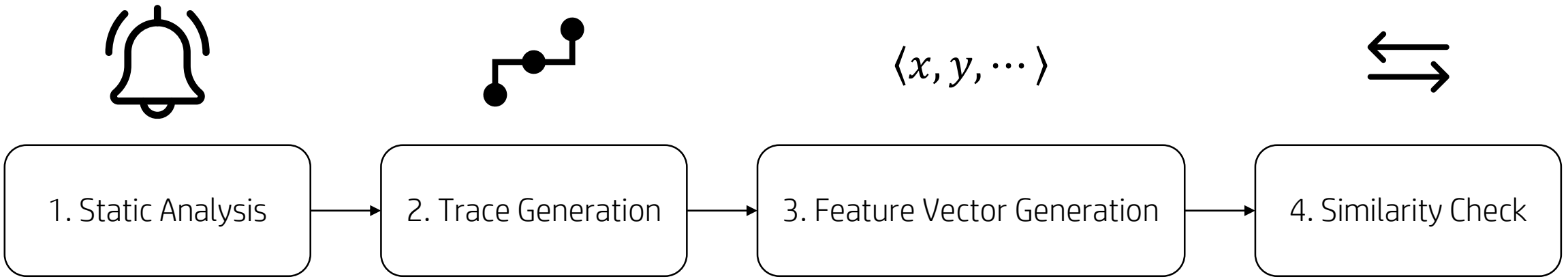
Effectiveness

273 Debian packages

112 new vulnerabilities

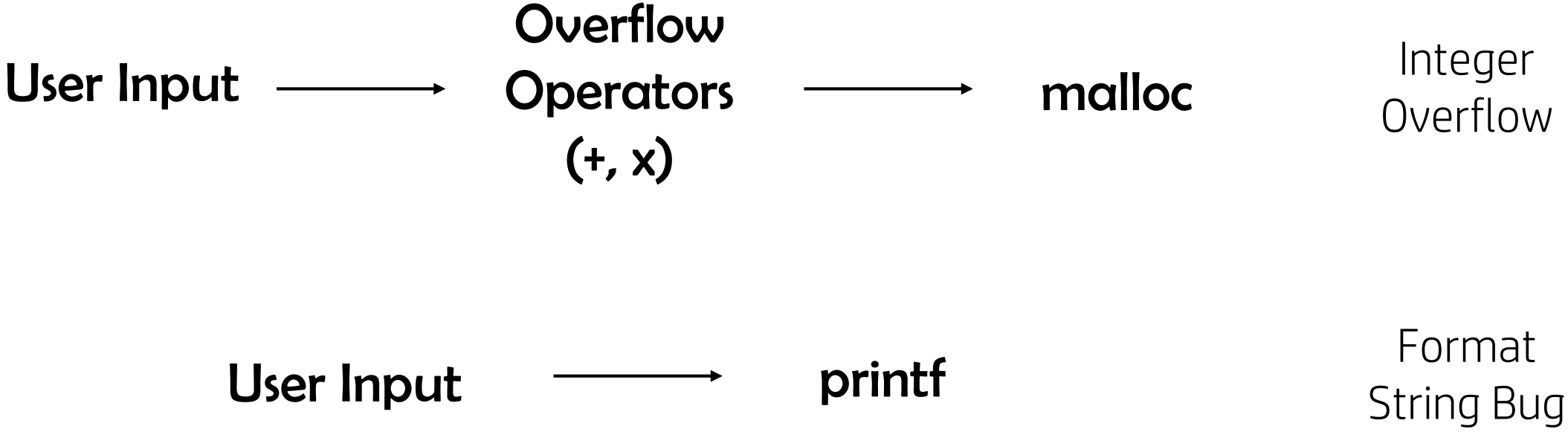
including 6 CVEs

Overall Process



Static Analysis

- Taint analysis : track the flow of malicious data



Integer Overflow

Integer Underflow

Buffer Overflow

Format String Bug

Command Injection

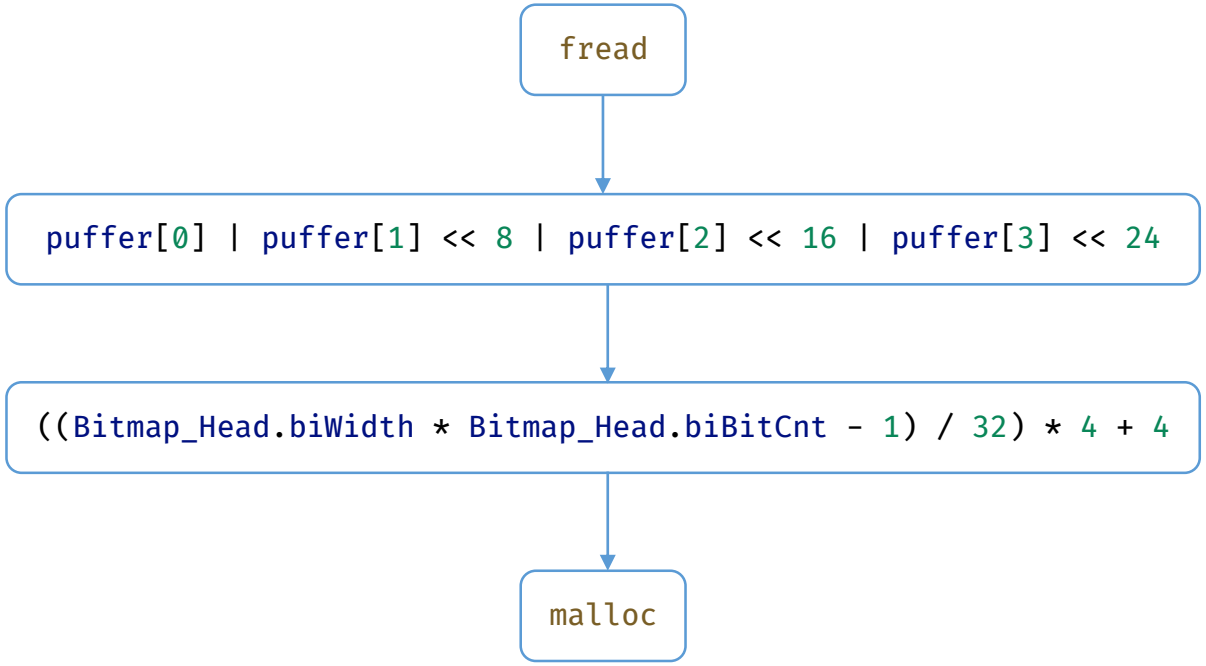
Use After Free

Double Free

Infer framework

```
gint32 ReadBMP(const gchar *name, ...) {
    ...
    filename = name;
    fd = fopen(filename, "rb");

1   if (!(fread(buffer, Bitmap_File_Head.biSize - 4, 1, fd) != 0)) {
        return -1;
    }

    Bitmap_Head.biWidth = ToL(&buffer[0x00]);
    Bitmap_Head.biBitCnt = ToS(&buffer[0x0A]);

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

    image_ID = ReadImage(..., rowbytes, ...);
    ...
}


static gint32 ToL(const guchar *puffer) {
2   return (puffer[0] | puffer[1] << 8 | puffer[2] << 16 | puffer[3] << 24);
}


static gint16 ToS(const guchar *puffer) { return (puffer[0] | puffer[1] << 8); }


static gint32 ReadImage(..., gint rowbytes, ...) {
    ...
4   buffer = malloc(rowbytes);
    ...
}
```

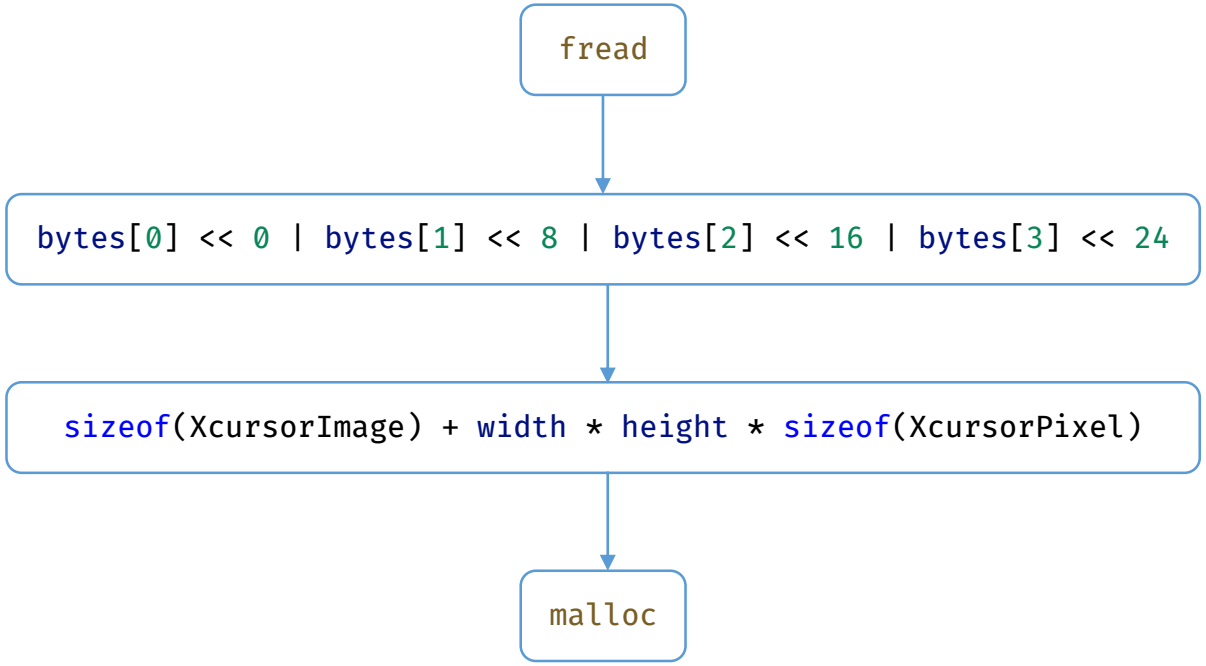
gimp (CVE-2009-1570)



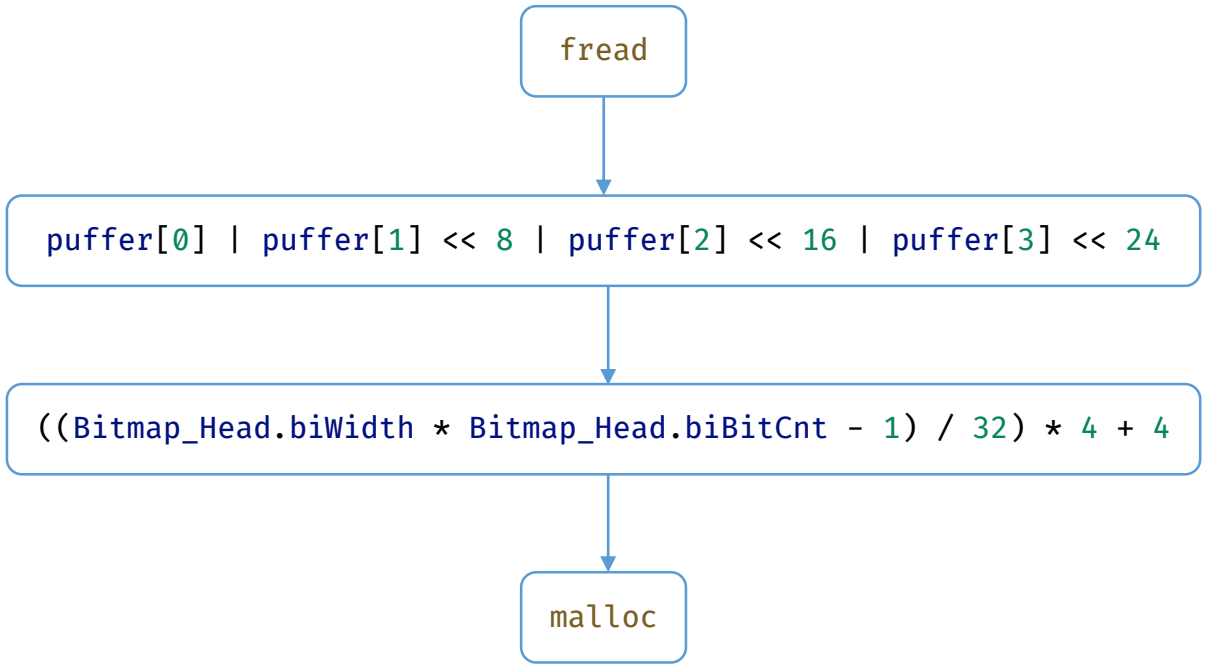
Trace of gimp

```
static XcursorBool _XcursorReadUInt(XcursorFile *file, XcursorUInt *u) {
    unsigned char bytes[4];

2   if (fread(bytes, 1, 4, file) != 4) return XcursorFalse;

3   *u = ((bytes[0] << 0) | (bytes[1] << 8) | (bytes[2] << 16) |
          (bytes[3] << 24));
    return XcursorTrue;
}


static XcursorImage *_XcursorReadImage(XcursorFile *file, ...) {
    XcursorImage head;
1   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 =
4       malloc(sizeof(XcursorImage) + width * height * sizeof(XcursorPixel));
    ...
}
```

libXcursor (CVE-2017-16612)

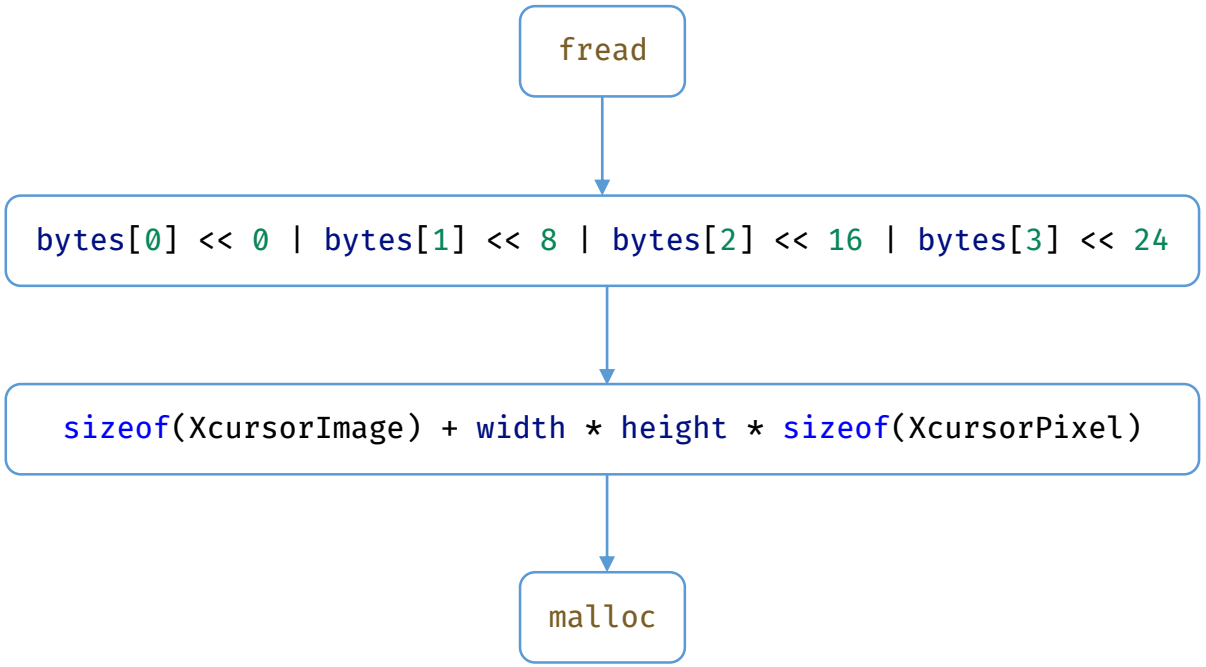


Trace of libXcursor



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

Feature vector of gimp



fread	malloc	|	<<	+	-	*	/
1	1	3	4	1	0	2	0

Feature vector of libXcursor

Similarity Check

- Cosine Similarity

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

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

fread	malloc	|	<<	+	-	*	/
1	1	3	4	1	0	2	0

$$\frac{1 \times 1 + 1 \times 1 + 3 \times 3 + 3 \times 4 + 1 \times 1 + 1 \times 0 + 2 \times 2 + 1 \times 0}{\sqrt{1^2 + 1^2 + 3^2 + 3^2 + 1^2 + 1^2 + 2^2 + 1^2} \times \sqrt{1^2 + 1^2 + 3^2 + 4^2 + 1^2 + 0^2 + 2^2 + 0^2}} = 0.95$$

Experiment

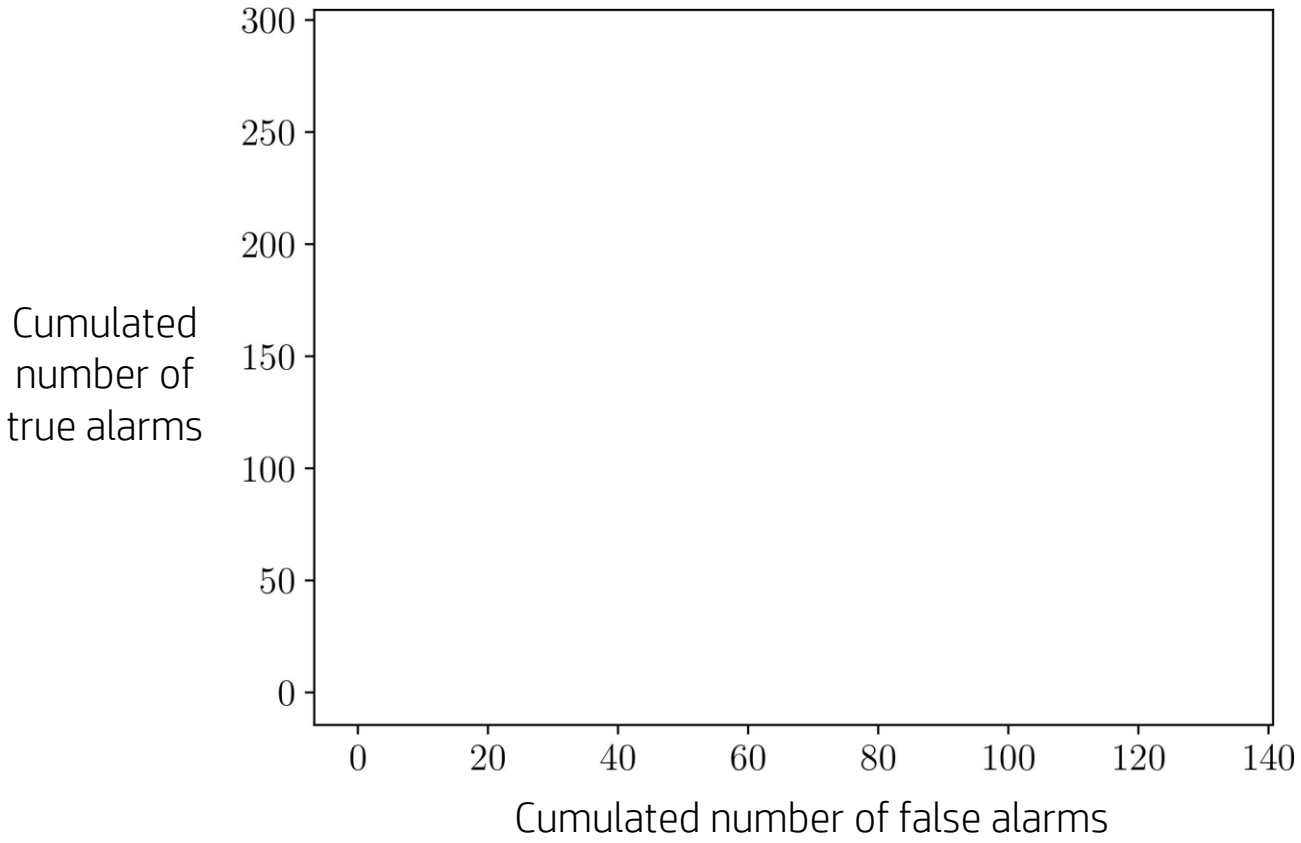
- Target : 273 Debian packages
- Signature : 16 publicly known vulnerabilities

  + 5383 examples in Juliet test suite (collection of test cases developed by NSA)

  + 5 OWASP bug examples

- Inspection : all 324 alarms with >0.85 + randomly sampled 100 alarms with <0.85

score	precision
>0.95	87.5%
>0.90	85.7%
>0.85	78.1%
<0.85	37.0%



1. Starts with (0, 0)
2. Check alarms from top scores
3. If it is true go up, otherwise go right

Juliet Test Suite

```
void CWE190_Integer_Overflow__int64_t_fscanf_square_01_bad() {
    int64_t data;
    data = 0LL;
    fscanf(stdin, "%" SCNd64, &data);
    int64_t result = data * data;
    char *p = malloc(result);
}
```

Juliet test suite

TRACER: Similarity 1.0

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


DiaObject *create_standard_polyline(int num_points, ...) {
    pcd.num_points = num_points;
2   new_obj = polyline_create(NULL, &pcd, ...);
    ...
}


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


void polyconn_init(..., int num_points) {
4   poly->points = malloc(num_points * sizeof(Point));
    ...
}
```

dia-0.97.3

Official website of Tracer →

SOFTWARE VULNERABILITIES RECUR...

no more :)

Backup Slides

Integer Overflow to Buffer Overflow

1. Overflowed value goes into the malloc
2. Returns a smaller memory area than the user expected
3. The user writes a value to that memory area
4. It may end up writing a value to an unallocated area

Alarm inspection

- We manually inspected whether the vulnerability could occur based on the source code of the program

- There is no PoC for every bug that we found

Vulnerability Report

- Tracer found 112 vulnerabilities in 67 packages.
- 30 vulnerabilities have been confirmed by the developers.
- 6 CVEs have been assigned.

Actual Format of Trace

- Trace is built from IR that Infer uses
- It has instructions like load, store, call, etc
- It is enough to generate features that we want

Ignore Ordering

- First, we chose the most intuitive and straightforward way
- We did not experiment with a feature vector considering the order
- It may have its advantages and disadvantages

Other Similarity Calculation Metrics

- We tested using Euclidian distance to calculate the similarity
- However, it did not show a better result
- Cosine similarity is good because the result is always 0~1

Why Debian Packages?

- There are some old and unmaintained packages
- But they can be installed on the user system for dependency
- If there are security vulnerabilities, it can be a potential threat

Sink vs Root Cause

- Sink : distinguish by sink points

- Default option for our checkers
- Fair comparison with other benchmarks

- Root Cause : remove duplicate sources

- Provide a more realistic number of bugs

Sink vs Root Cause

	Root Cause	True Positive	False Positive
>0.95	58	154	22
>0.90	69	192	32
>0.85	87	253	71

Use After Free, Double Free

- Early research only targeted five types of vulnerabilities
- Checkers for heap-related bugs are experimental
- However, it still catches two true alarms for >0.85