Getting started with vulnerability discovery using Machine Learning

Gustavo Grieco

Hack In The Box Lab 2016

CIFASIS - CONICET / VERIMAG

Motivation

What if we had the best team of security researchers .. ?

program + input $\rightarrow$ security issue?

.. but

They are **expen$ive** and we want to discover **more vulnerabilities**, using less resources (time/money).

Program Behaviors

We should focus on programs and inputs that could do something "bad".

.. but

They are **expen$ive** and we want to discover **more vulnerabilities**, using less resources (time/money).

Program Behaviors

We should focus on programs and inputs that could do something "bad".

Overview and Applications

How?

program and inputs $\rightarrow$ **traces** $\rightarrow$ **machine learning** $\rightarrow$ **program behaviors**

Why?

Vulnerability Detection: $\rightarrow$ **extrapolation and prediction** of vulnerable inputs.

Seed selection: $\rightarrow$ **reduction** of the set of inputs to "cover" all the program behaviors.

Programs, traces and behaviors

Let's start with..

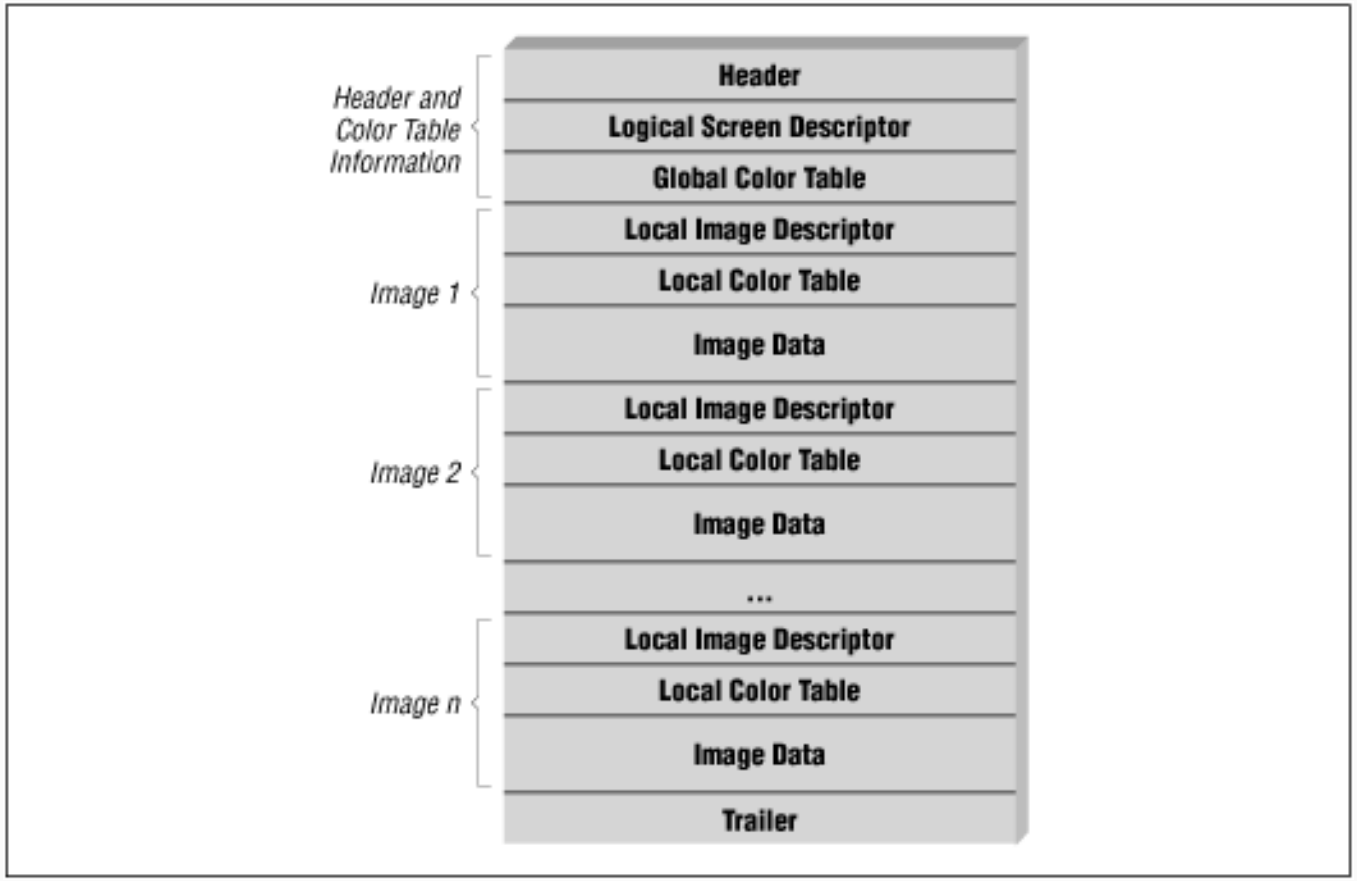
1. A binary program: **gifflip**:

   A program to flip (mirror) GIF file along X or Y axes, or rotate the GIF file 90 degrees to the left or to the right.

2. A large number of inputs: hundreds or thousands gif files.

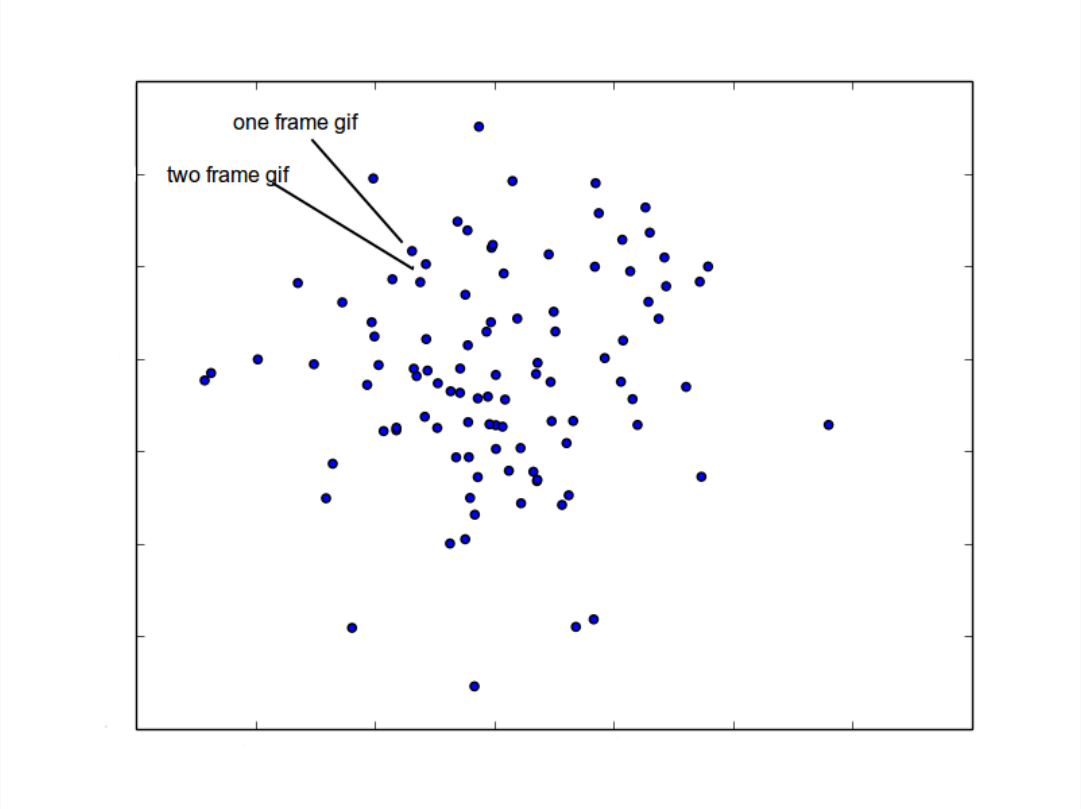
Graphics Interchange Format

The input space of **gifflip** can be specified using the following structure:



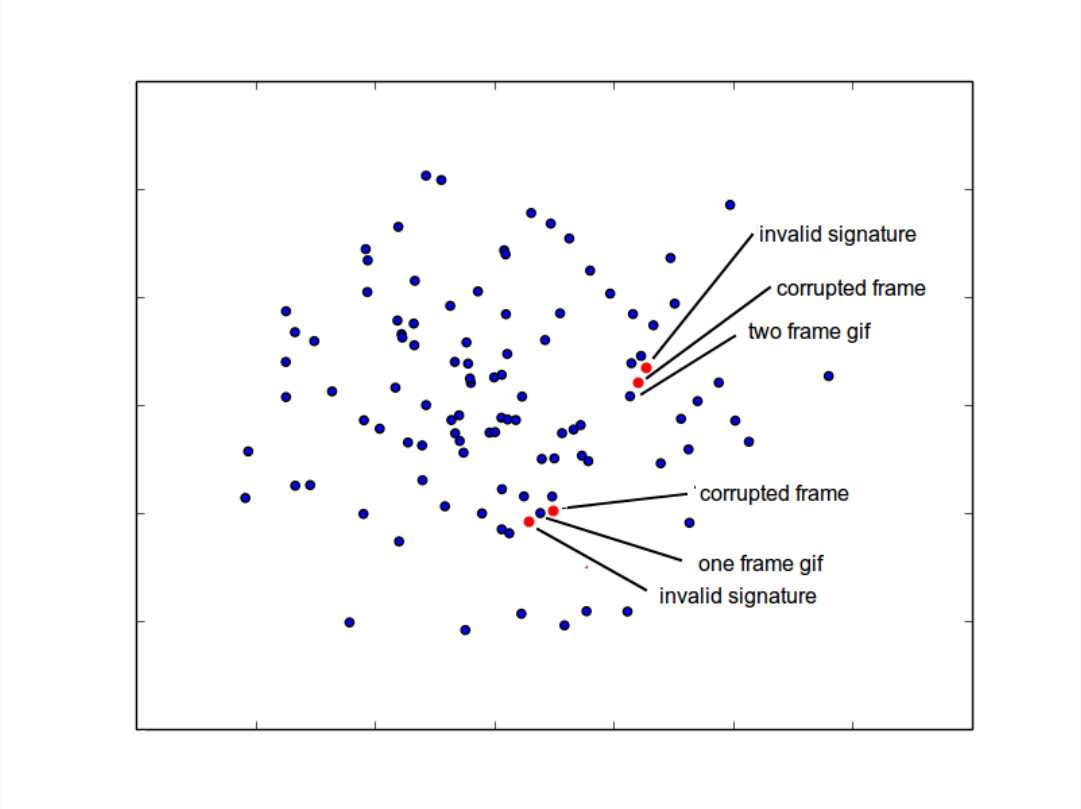
Extracting this information using the binary and some inputs is a **very challenging** task!

Input Specification Space



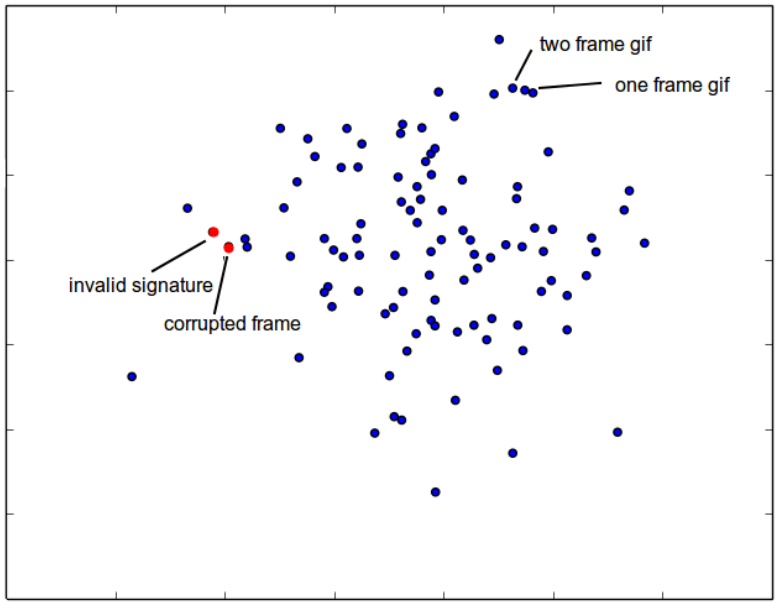
where similar gif structures are close together.

Input File Space



where similar files are close together.

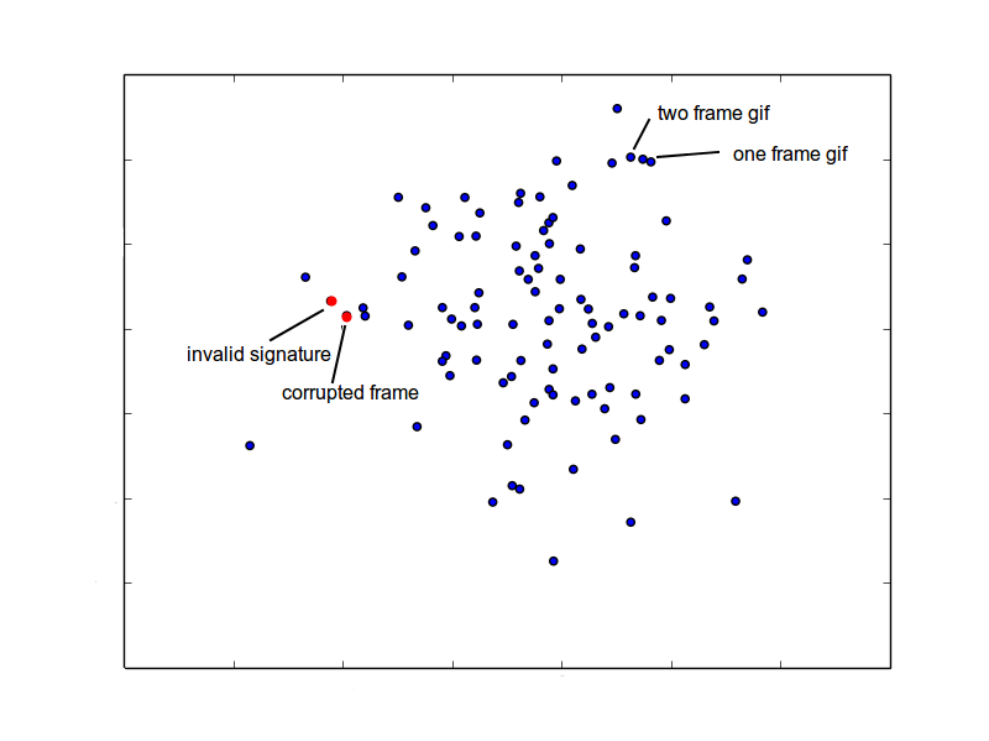
Trace Space



where similar traces are close together.

Clusters of traces represent a **program behavior**

Trace Space



where similar traces are close together.

Clusters of traces represent a **program behavior**

What are traces anyway?

PIN

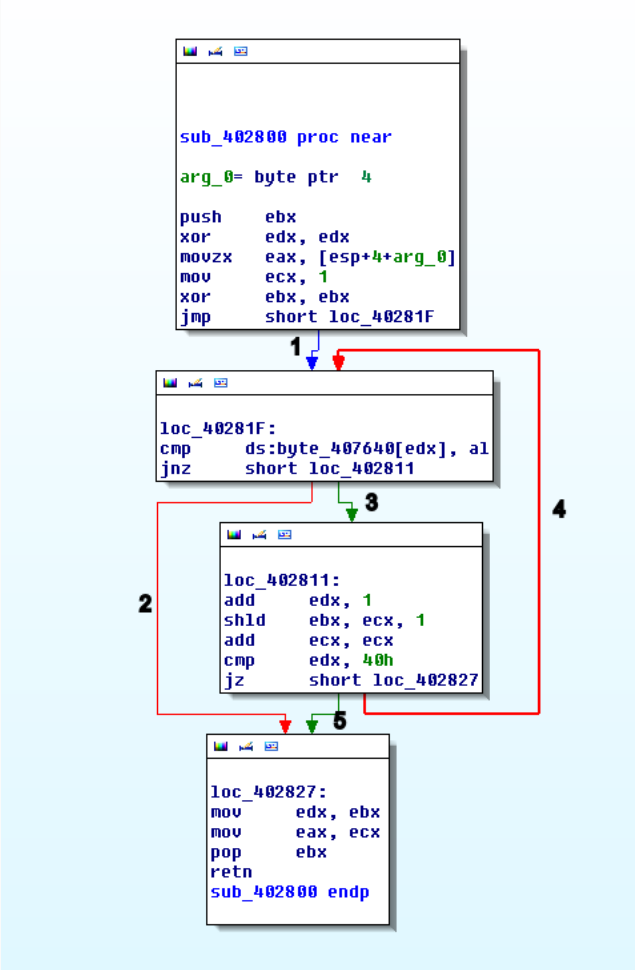
```
0x8048e4b mov [0x809a100], eax S@809a100[4]=0xffffc98a R[eax]=ffffc98a R[ds]=2b
0x8048e50 mov eax, [0x809a100] W[eax]=ffffc98a L@809a100[4]=0xffffc98a R[ds]=2b
0x8048e55 test eax, eax W[eflags]=282 R[eax]=ffffc98a R[eax]=ffffc98a
0x8048e57 jz 0x8048e68 W[eip]=8048e59 R[OF]=0 R[CF]=0 R[ZF]=0 R[SF]=1 R[DF]=0 R[PF]=0

...
```

- Developed by Intel and used in many projects.
- Every instruction and its operands are recorded.
- Traces are sequences of instructions with all its operands values.

American Fuzzy Lop

- Developed by Google but only used in AFL.
- Every jump in a binary is instrumented to have a label using afl-gcc/g++ or QEMU.
- Traces are sequences of labels representing transitions between basic blocks.
- For instance:

  $1 - 3 - 4 - 3 - 4 - 2$

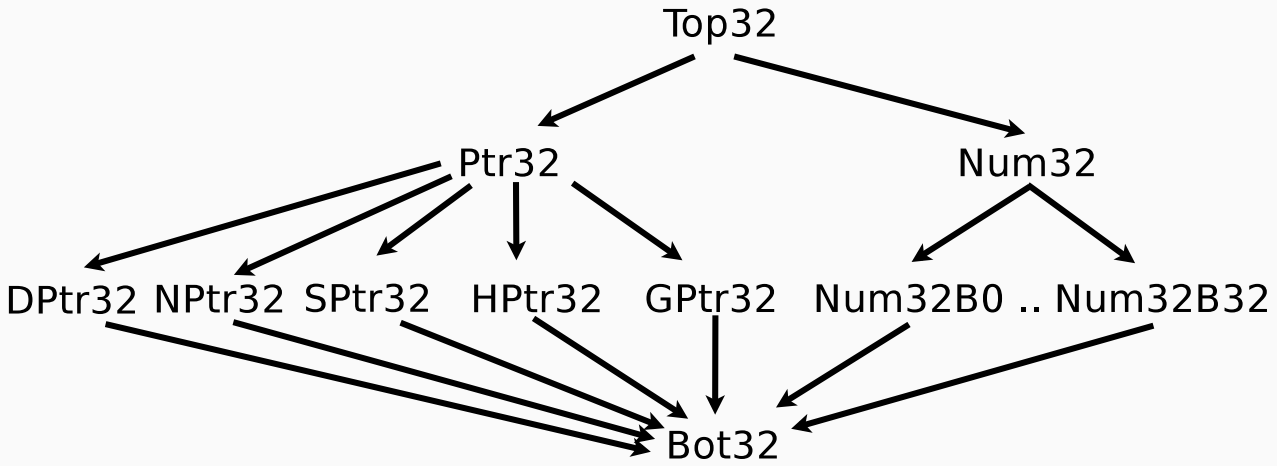


VDiscover

Itrace	VDiscover
getenv('XAINPUT')	getenv(GPtr32)
strcpy('', 'input')	strcpy(SPtr32,HPtr32)
strtok('input', ',')	strtok(HPtr32,GPtr32)

- Every call to the standard C library is captured and augmented with dynamic information of its arguments using ptrace.
- Traces are sequences of events corresponding to such calls.

Dynamic processing of values



Remember:

Machine Learning algorithms cannot deals with values like string, pointers, integers, that why replace them with meaningful labels.

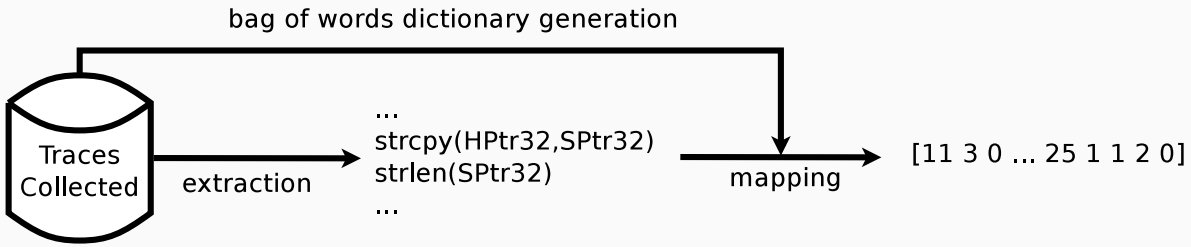
Traces Representations

Unfortunately..

Traces needs to be normalized since longer traces are likely to contain more information than short ones.

- Bag of words: a trace is represented as the bag (multiset) of its events, disregarding grammar and even event order but keeping multiplicity.
- Subtraces of maximum length: a trace is represented as the set of subtraces sampled from the original (long) trace.

For instance



Remember:

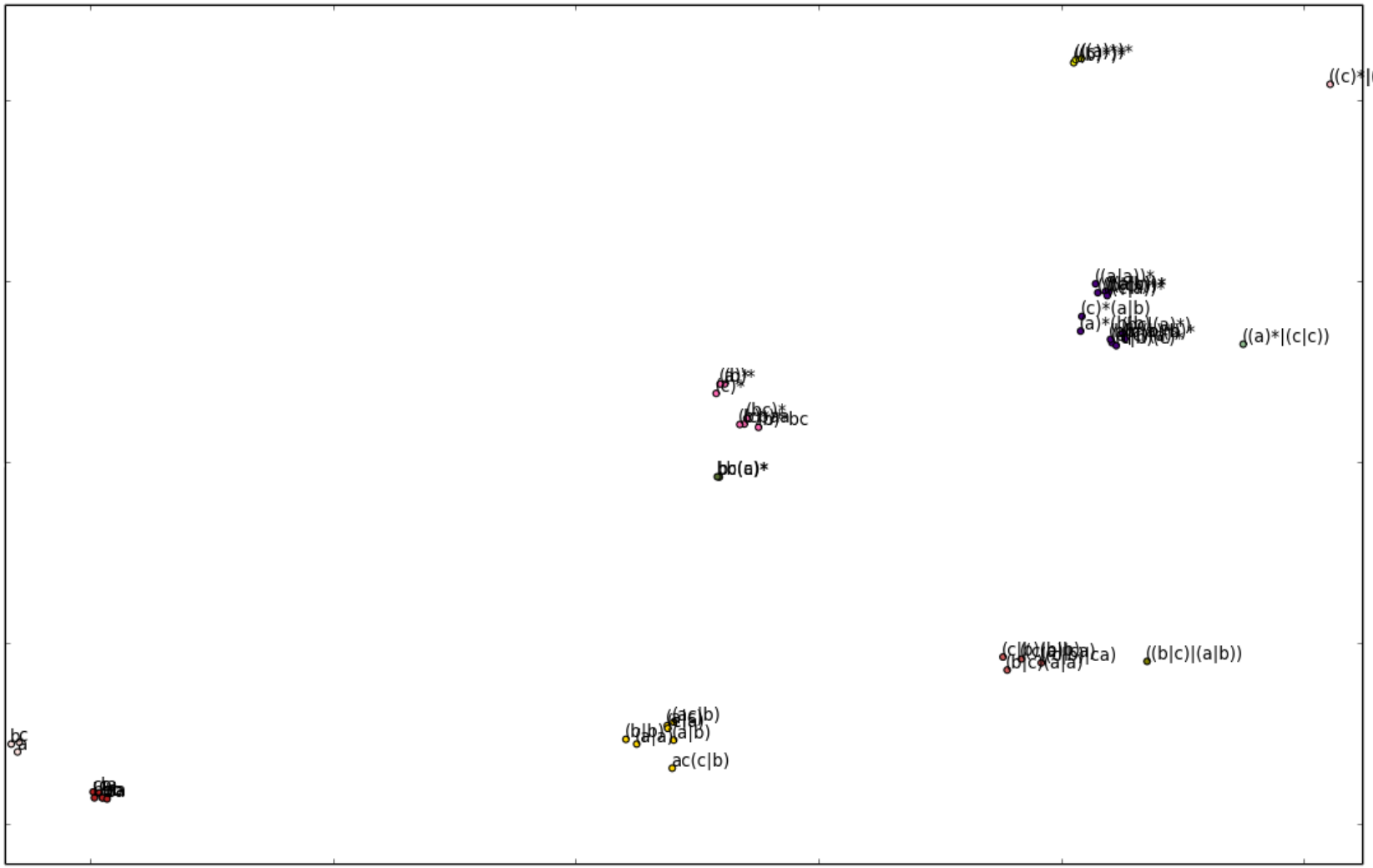
A trace and its representation can be completely different things.

Visual Explorations of Trace Space

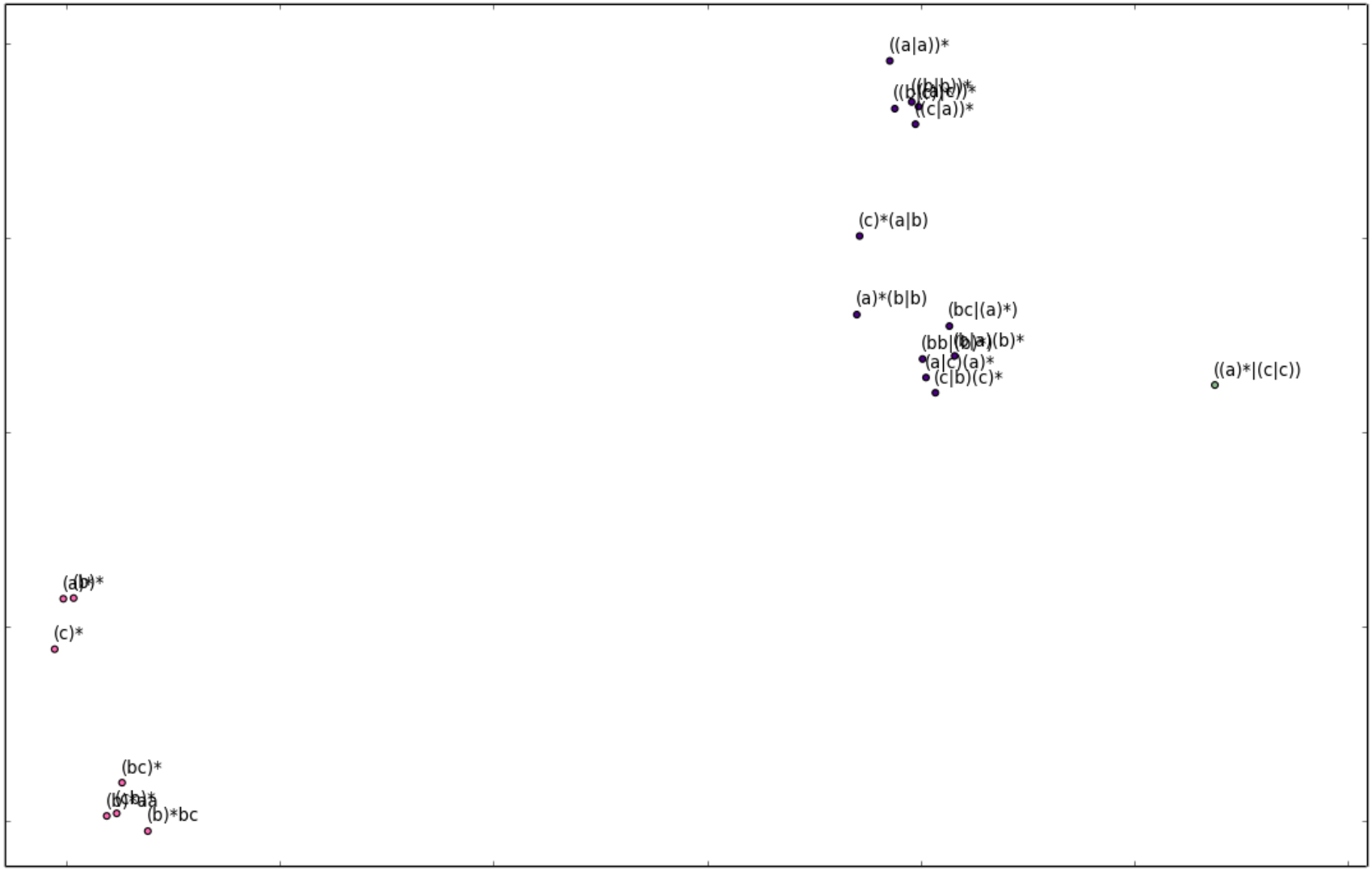
Inputs and programs traced

- Parsing of simple regex expressions (pcre).
- Detection of file types using file (libmagic).
- Display of information of PNG files from pnginfo (libpng 1.2)

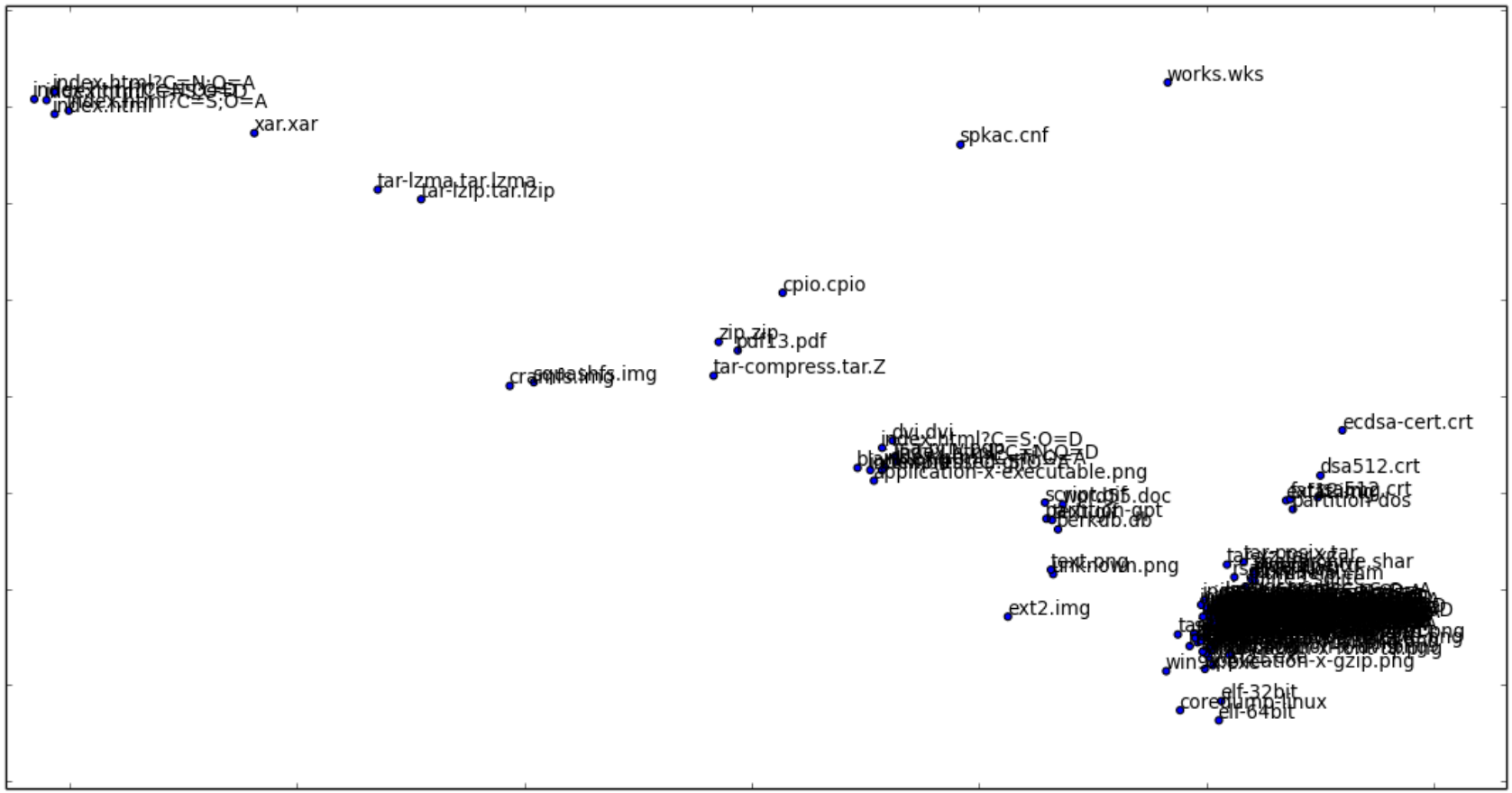
regex (pcre) - AFL - BOW



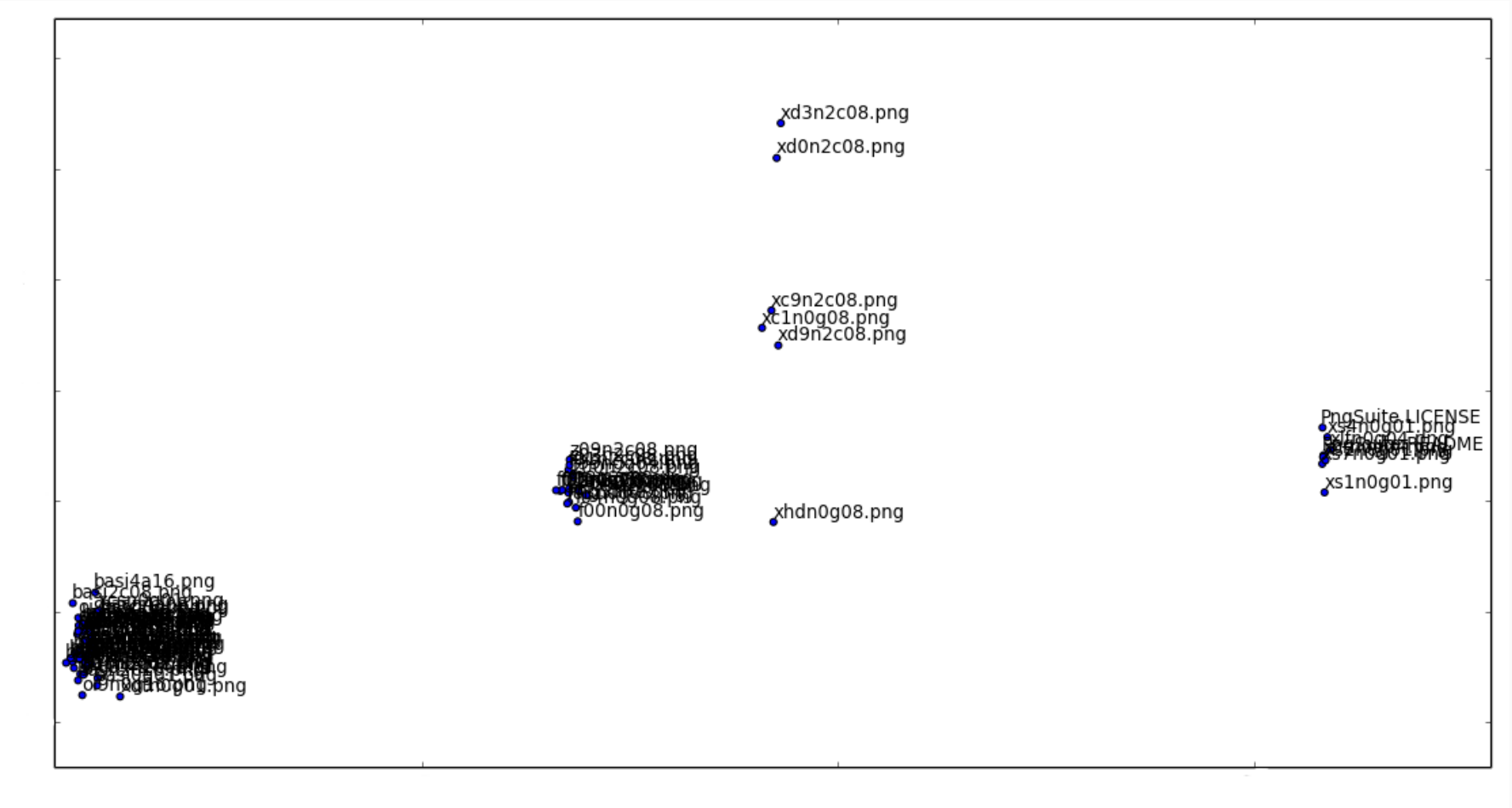
regex (pcre) - AFL - BOW



file (libmagic) - VD - BOW

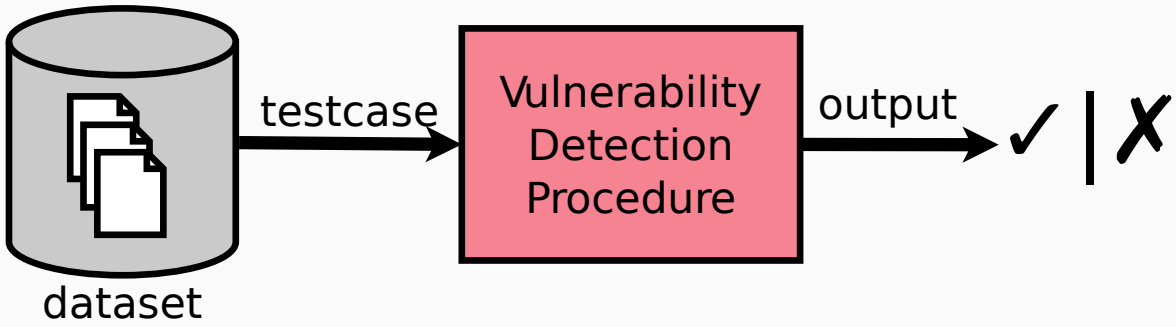


png (libpng12) - VD - BOW

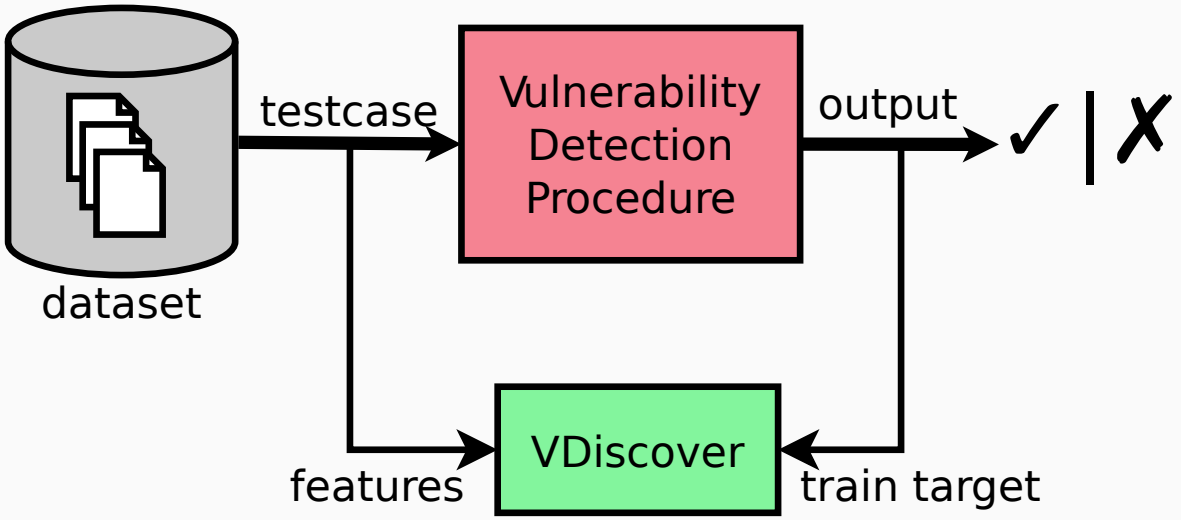


Vulnerability Prediction

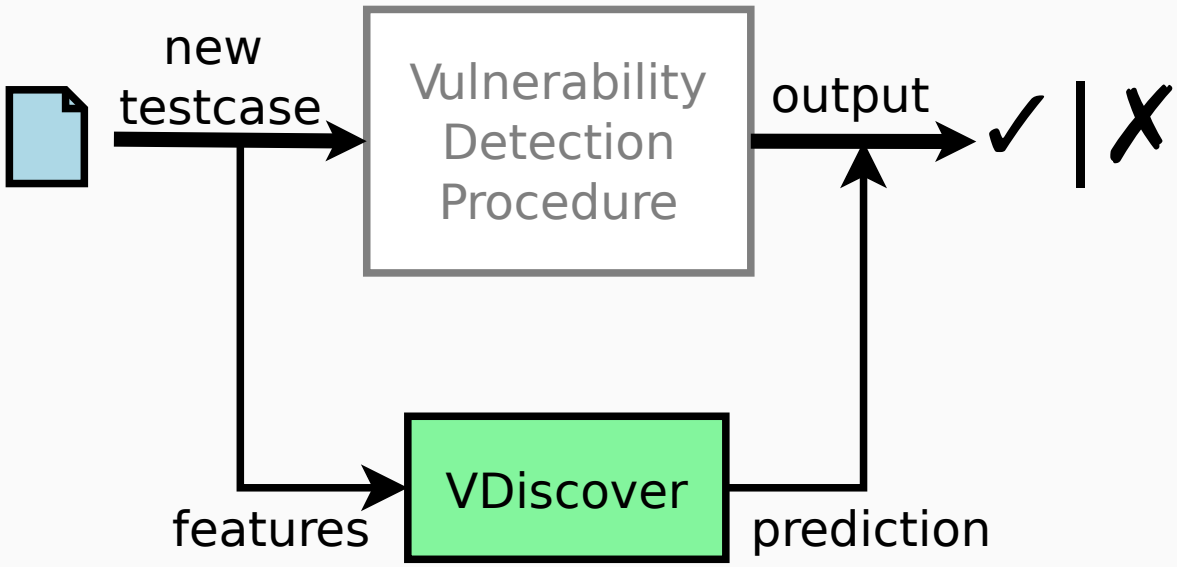
Overview


testcase
Vulnerability
Detection
Procedure
output
✓
✗
dataset

Overview


testcase
Vulnerability
Detection
Procedure
output
✓ | ✗
dataset
VDiscover
features
train target

Overview


new
testcase
Vulnerability
Detection
Procedure
output
✓ | ✗
VDiscover
features
prediction

Key Principles of VDiscover

1. **No source-code required**: Our features are extracted using static and dynamic analysis for binaries programs, allowing our technique to be used in proprietary operating systems.
2. **Automation**: No human intervention is need to select features to predict, we focused only on feature sets that can be extracted and selected automatically, given a large enough dataset.
3. **Scalability**: Since we want to focus on scalable techniques, we only use lightweight static and dynamic analysis. Costly operations like instruction per instruction reasoning are avoided by design.

A harmless crash?

xa is a small cross-assembler for the 65xx series of 8-bit processors (i.e. Commodore 64). We can easily crash it:

```
$ gdb --args env -i /usr/bin/xa '\bo@e\0' '@o' '-o'
...
Program received signal SIGSEGV, Segmentation fault.
(gdb) x/i
$eip => 0x8049788:  movzbl (%ecx),%eax
(gdb) info registers
eax 0x0 0
ecx 0x0 0
...
```

Question:

It is just a NULL pointer dereference, should we spend our resources trying to fuzz this test case?

Smashing the stack..

```
$ gdb --args env -i /usr/bin/xa '\bo@e\0' '@o' 'AAAA...AAAA-o'
```

```
Copyright (C) 1989-2009 Andre Fachat, Jolse Maginnis, David Weinehall
o@e:line 1:  1000:Syntax error
and Cameron Kaiser.
o@e:line 2:  1000:Syntax error
Couldn't open source file '@o'!
o@e:line 3:  1000:Syntax error
Couldn't open source file 'o@'!
*** buffer overflow detected ***:  /usr/bin/xa terminated
...
```

vulnerability detection procedure

We used a simple fuzzer producing 10,000 mutation for each test case.

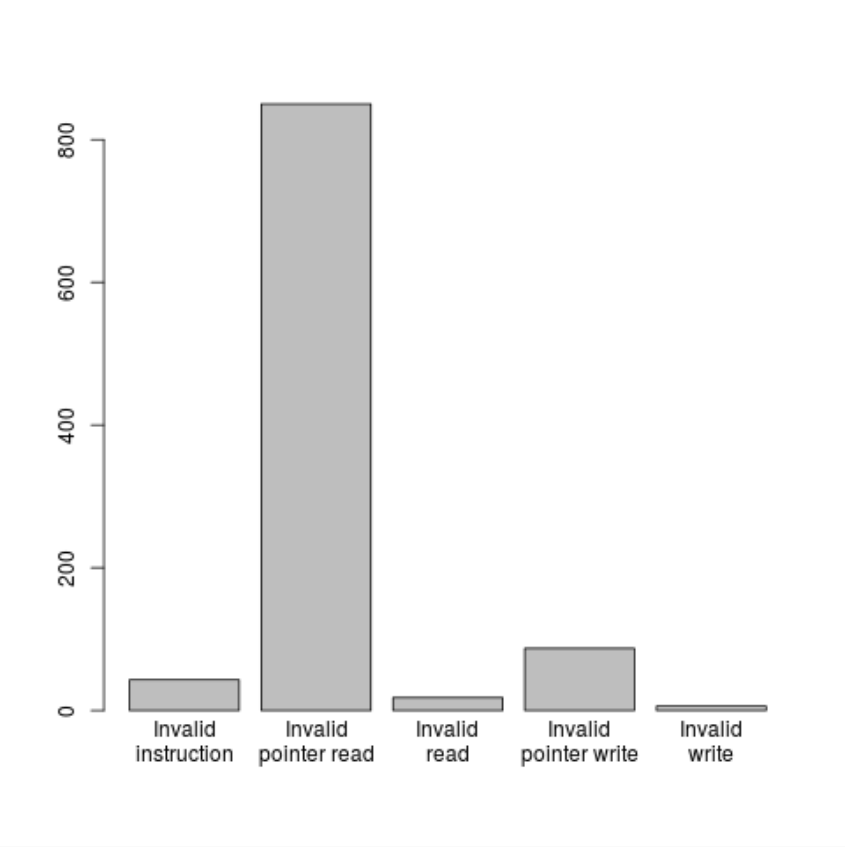
Smashing the stack..

```
$ gdb --args env -i /usr/bin/xa '\bo@e\0' '@o' 'AAAA...AAAA-o'


Copyright (C) 1989-2009 Andre Fachat, Jolse Maginnis, David Weinehall
o@e:line 1:  1000:Syntax error
and Cameron Kaiser.
o@e:line 2:  1000:Syntax error
Couldn't open source file '@o'!
o@e:line 3:  1000:Syntax error
Couldn't open source file 'o@'!
*** buffer overflow detected ***:  /usr/bin/xa terminated

...
```

vulnerability detection procedure

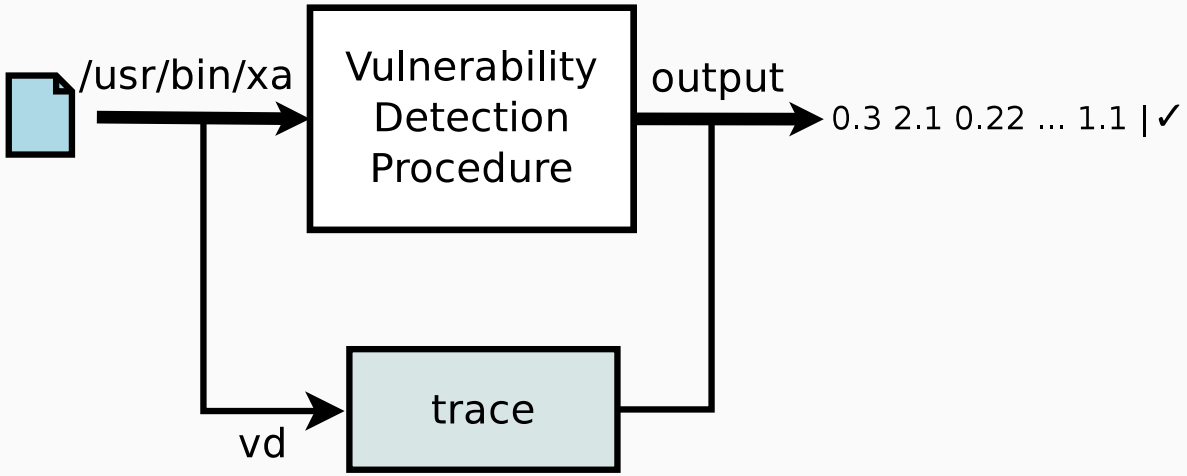
We used a simple fuzzer producing 10,000 mutation for each test case.

Debian bug reports from Mayhem



- A total of 1039 bugs in 496 packages.
- Every bug is packed with a crash report and the required inputs to reproduce it.

For instance

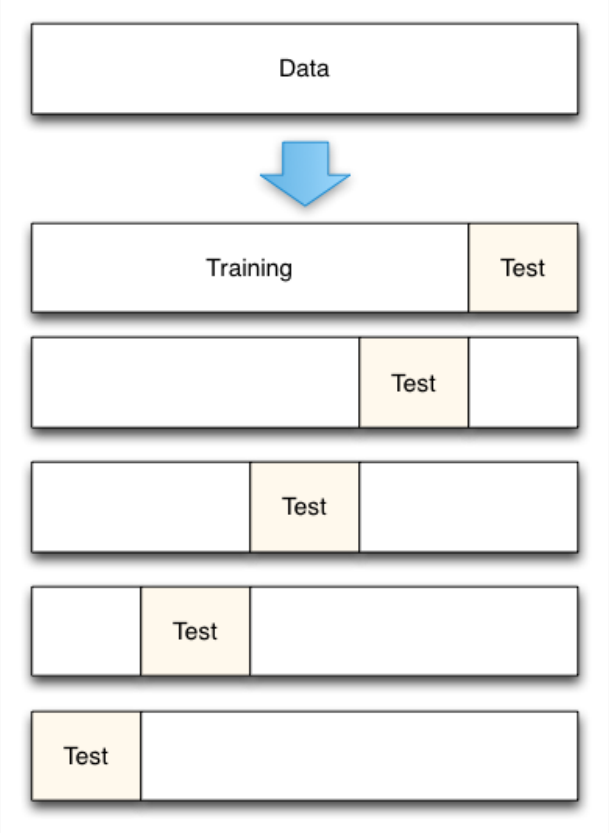


vulnerability detection procedure

Around 8% was found vulnerable to interesting memory corruptions.

Model training/inference

Training and Testing



Prediction accuracy (best predictor)

	Flagged	**Not Flagged**
Flagged	**55%**	17%
Not Flagged	45%	**83%**

These results are obtained using Random Forest (scikit-learn) in 1-3 grams representation.

Not flagged cases are slower, because the fuzzer will not find vulnerabilities.

Prediction accuracy (best predictor)

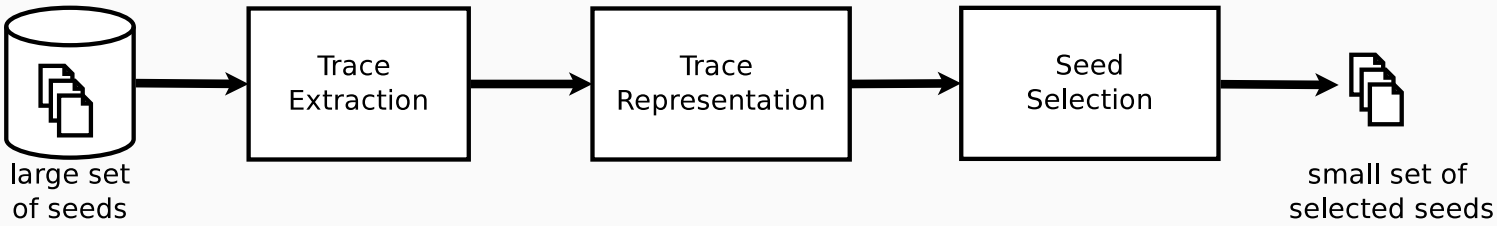
	Flagged	**Not Flagged**
Flagged	**55**%	17%
Not Flagged	45%	**83**%

These results are obtained using Random Forest (scikit-learn) in 1-3 grams representation.

Not flagged cases are slower, because the fuzzer will not find vulnerabilities.

Seed Selection for fuzzing [WIP]

Overview



- Seed selection in mutational fuzzing for a program P:
  1. Collect a very large number of input files (seeds).
  2. Select a subset of seeds according to some criteria.
  3. Start fuzzing with selected seeds checking if P fails.

Observation:

Seed selection should avoid redundancy in the initial selection.

Collecting seeds

```
...  conceptdraw.html ichannels.html nanrenwo.html skionline.html
xooit.html confused.html ifc.html naukrinama.html sltrib.html
xpartner.html congtyinanquangcao.html iflscience.html naunet.html
smartertravel.html xxl-sale.html contracostatimes.html igri-2012.html
nbcsandiego.html smartsms.html xxxvideoo.html cookingforgirlz.html
ihc.html nbnews.html smartwebads.html yanstat.html cooltext.html ...
```

- HTML and CSS files obtained randomly sampling from the first 10k most visited pages (Alexa)
- Files are randomly cut in fragments of certain max sizes (128b, 1k)
- All kinds of languages, encoding and types of websites were retrieved!

Targets

- libxml2 (2.7.2): “xmllint –html @@”
- w3m (0.5.3): “w3m -dump -T text/html @@”
- gumbo-parser (0.9.0): “clean_text @@”
- html2text (1.3.2a): “html2text @@”
- htmlcxx (0.85): “htmlcxx @@”
- htmldoc (1.8.27): “htmldoc @@”
- html-xml-utils (6.5): “hxnormalize @@”
- tidy (20091223cvs): “tidy @@”

All these programs were recompiled using ASAN in order to detect invalid memory reads/writes.

Targets

- libxml2 (2.7.2): “xmllint –html @@”
- w3m (0.5.3): “w3m -dump -T text/html @@”
- gumbo-parser (0.9.0): “clean_text @@”
- ~~html2text (1.3.2a): “html2text @@”~~
- ~~htmlcxx (0.85): “htmlcxx @@”~~
- htmldoc (1.8.27): “htmldoc @@”
- html-xml-utils (6.5): “hxnormalize @@”
- tidy (20091223cvs): “tidy @@”

All these programs were recompiled using ASAN in order to detect invalid memory reads/writes.

Fuzzing time!

General settings:

- AFL 1.94b was used instrumenting the target programs (recompiled using afl-gcc/g++).
- For each experiment, we fuzzed at least 48hs in a dedicated core using "quick and dirty" mode (-d).

Selecting seeds:

- AFL includes its own seed selection (called corpus minimization) based on afl-traces and implemented in afl-cmin.
- VDiscover includes a pattern based seed selection algorithm.

Fuzzing time!

General settings:

- AFL 1.94b was used instrumenting the target programs (recompiled using afl-gcc/g++).
- For each experiment, we fuzzed at least 48hs in a dedicated core using "quick and dirty" mode (-d).

Selecting seeds:

- AFL includes its own seed selection (called corpus minimization) based on afl-traces and implemented in afl-cmin.
- VDiscover includes a pattern based seed selection algorithm.

From traces to vectors

trace extraction

```
$ vd -i seeds -o program.traces -c "./program @@"
```

complete trace

... read(Num32B8,HPtr32,Num32B24) free(HPtr32) calloc(Num32B8,Num32B24) ...

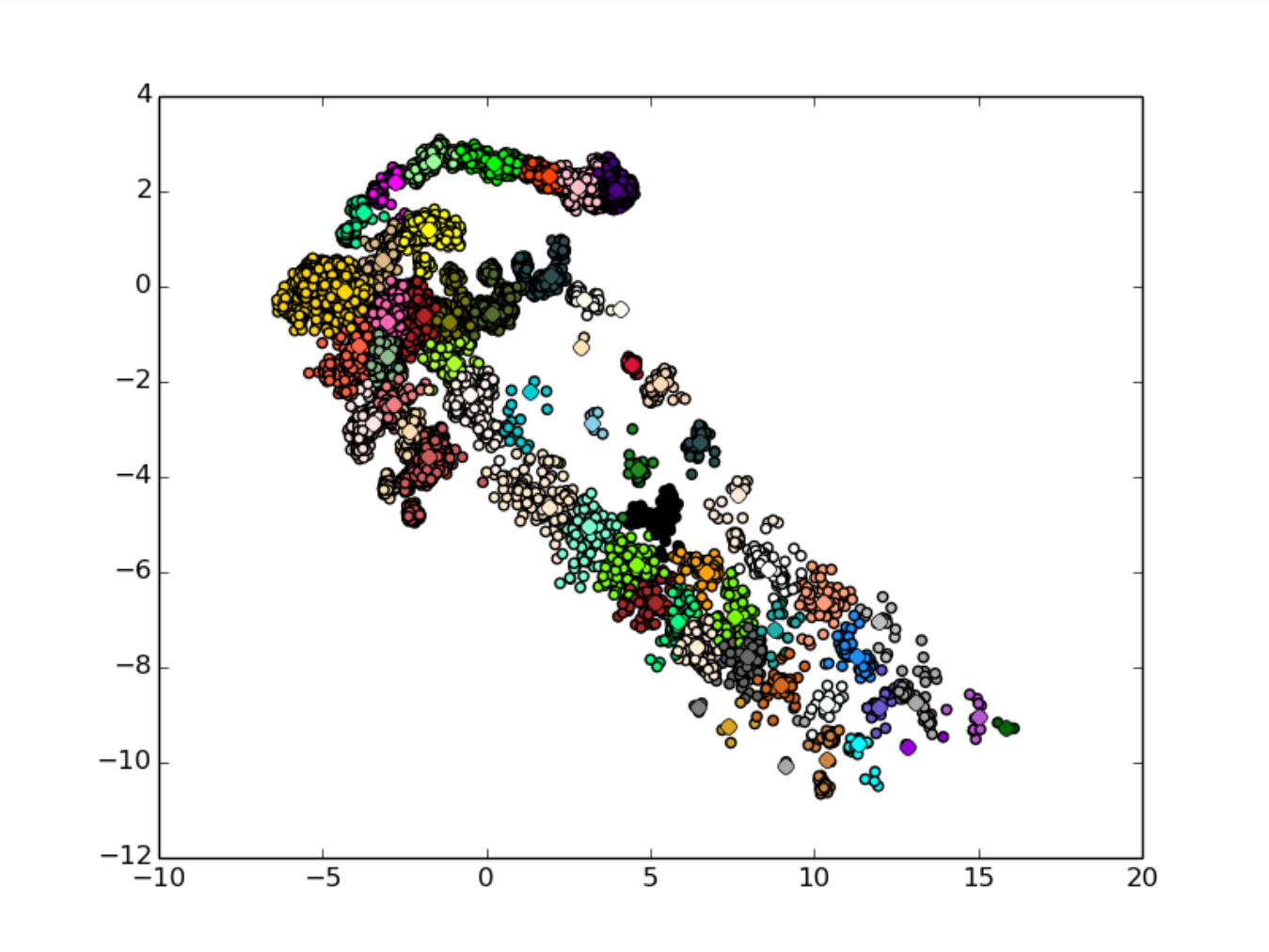
fixed size subtrace

read(Num32B8,HPtr32,Num32B24)	free(HPtr32)	calloc(Num32B8,Num32B24)

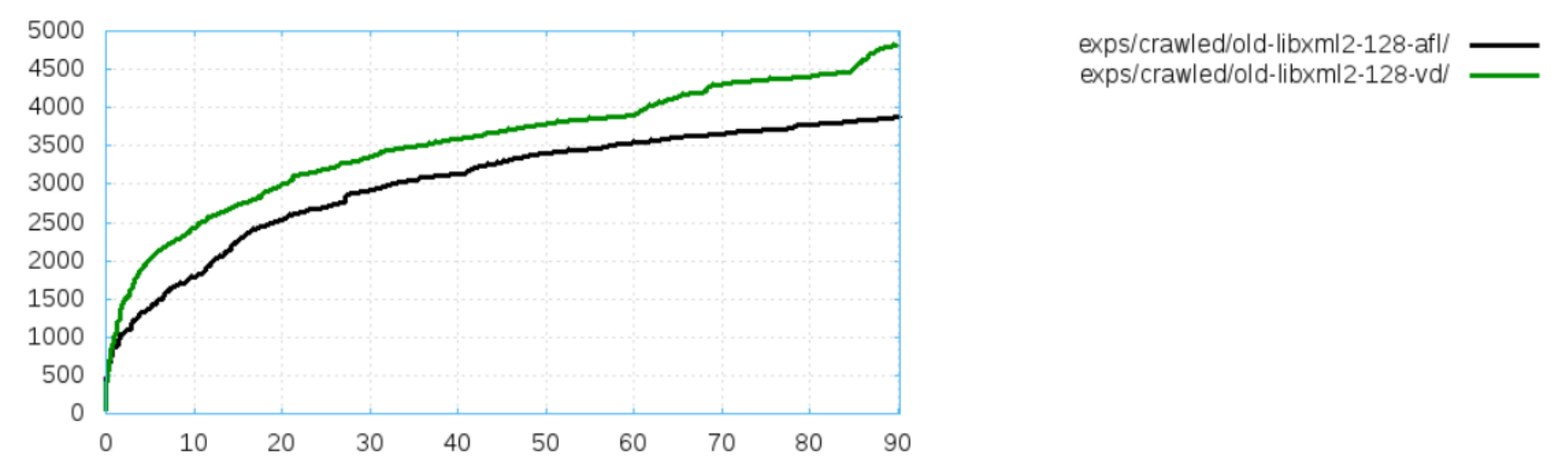
fixed size real vector

0.12	0.31	0.06	0.91	0.42

libxml2 traces and results

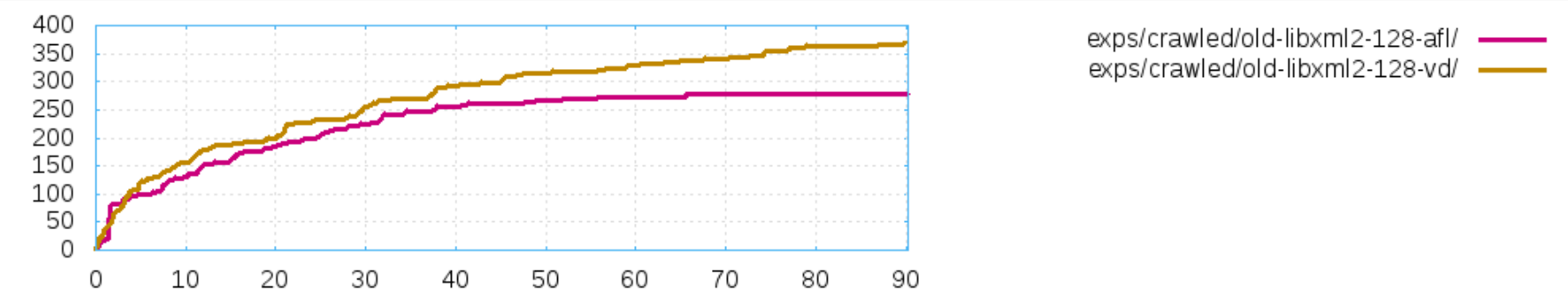


libxml2 traces and results



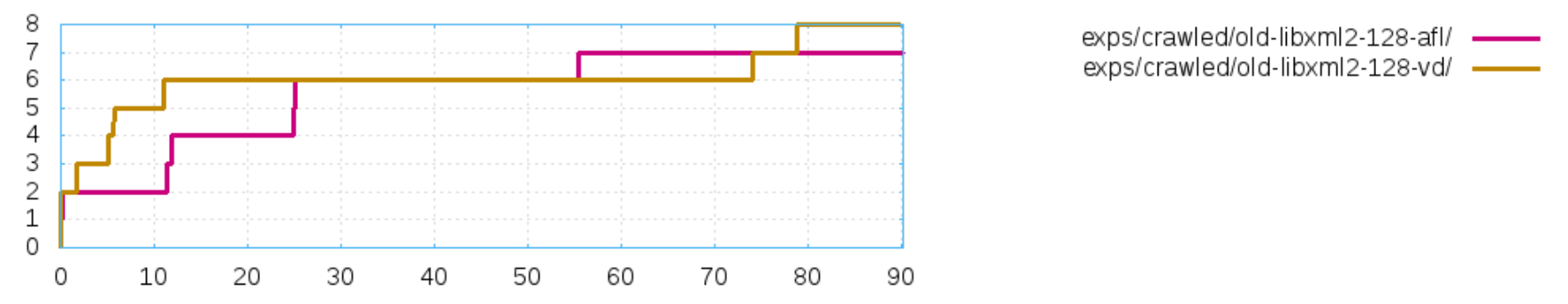
Paths explored using AFL

libxml2 traces and results



Crashes discovered using AFL

libxml2 traces and results



Unique crashes discovered using AFL

Give me a break!

Workshop Time!

Overview

1. Installing VDiscover.
2. Creating test cases and extracting traces.
3. Trace visualization and seed selection.
4. Training and predicting with ZZUF dataset.

Installing VDiscover

Make sure you install a recent version, not the ancient version from the Ubuntu repositories (you can download packages here)

1. Setup a VM:

```
vagrant init ubuntu/trusty32
vagrant up --provider virtualbox
vagrant ssh -- -X
```

2. Take some minutes to update and install basic stuff (git, python-setuptools, python-matplotlib, python-scipy ..)

```
git clone https://github.com/CIFASIS/vdiscover-workshop
git clone https://github.com/CIFASIS/VDiscover
cd VDiscover
./setup.py install --user
```

(don't forget to append "PATH=$PATH:~/.local/bin" to your .bashrc)

VDiscover

- Open source (GPL3) and available here:
  `http://www.vdiscover.org/`
- Written in Python 2:
  - python-ptrace
  - scikit-learn (and dependencies)
- Composed by:
  - tcreator: test case creation
  - fextractor: feature extraction
  - vpredictor: trainer and predictor
  - vd: a high level script to save time extracting data
- Trace should be collected in x86 (because i'm lazy!)

Setting up a test case

```
$ printf '<b>Hello!'  > test.html
$ tcreator --name test-html --cmd "/usr/bin/html2text
file:$(pwd)/test.html" out
```

Workshop Time!

Experiment adding and removing arguments and files to check how test cases are created.

Setting up a test case

```
$ printf '<b>Hello!'  > test.html
$ tcreator --name test-html --cmd "/usr/bin/html2text
file:$(pwd)/test.html" out
```

Workshop Time!

Experiment adding and removing arguments and files to check how test cases are created.

Collecting my first trace (1)

```
$ fextractor --dynamic out/test-html/ > trace1.csv
$ cat trace1.csv
out/test-html/ strcmp:0=GxPtr32 strcmp:1=GxPtr32 strcmp:0=GxPtr32
strcmp:1=GxPtr32 strcmp:0=GxPtr32 strcmp:1=GxPtr32
strcmp:0=GxPtr32 strcmp:1=GxPtr32 strcmp:0=GxPtr32
strcmp:1=GxPtr32 strcmp:0=GxPtr32 strcmp:1=GxPtr32
strcmp:0=GxPtr32 strcmp:1=GxPtr32 ..
```

Workshop Time!

Take a few minutes to extract traces from other programs and how to include/exclude events from different modules (–inc-mods/–ign-mods)

Collecting my first trace (1)

```
$ fextractor --dynamic out/test-html/ > trace1.csv
$ cat trace1.csv
out/test-html/ strcmp:0=GxPtr32 strcmp:1=GxPtr32 strcmp:0=GxPtr32
strcmp:1=GxPtr32 strcmp:0=GxPtr32 strcmp:1=GxPtr32
strcmp:0=GxPtr32 strcmp:1=GxPtr32 strcmp:0=GxPtr32
strcmp:1=GxPtr32 strcmp:0=GxPtr32 strcmp:1=GxPtr32
strcmp:0=GxPtr32 strcmp:1=GxPtr32 ..
```

Workshop Time!

Take a few minutes to extract traces from other programs and how to include/exclude events from different modules (–inc-mods/–ign-mods)

Collecting my first trace (2)

```
$ printf '<baaa>Bye!'  > test.html
$ fextractor --dynamic out/test-html/ > trace2.csv
$ cat trace2.csv
out/test-html/ strcmp:0=GxPtr32 strcmp:1=GxPtr32 strcmp:0=GxPtr32
strcmp:1=GxPtr32 strcmp:0=GxPtr32 strcmp:1=GxPtr32
strcmp:0=GxPtr32 strcmp:1=GxPtr32 strcmp:0=GxPtr32
strcmp:1=GxPtr32 strcmp:0=GxPtr32 strcmp:1=GxPtr32
strcmp:0=GxPtr32 strcmp:1=GxPtr32 ..
```

It looks exactly the same!!

.. but in fact, they are not. Later, we are going to show how to **easily** visualize traces..

Visualizing test cases

- Collecting data:

  ```
  $ tar -xf bmpsuite-2.4.tar.gz
  ```

  ```
  $ vd -m netpbm -i bmps "/usr/bin/bmptopnm @@" -o
  ```

  ```
  bmptopnm-traces.csv
  ```
- Clustering using bag of words and display:

  ```
  $ vpredictor --cluster-bow --dynamic bmptopnm-traces.csv
  ```
- After the clustering, a file (bmptopnm-traces.csv.clusters) will be written.

Exercise:

Using the source code of bmptopnm, try to understand why test cases are clusterized like this.

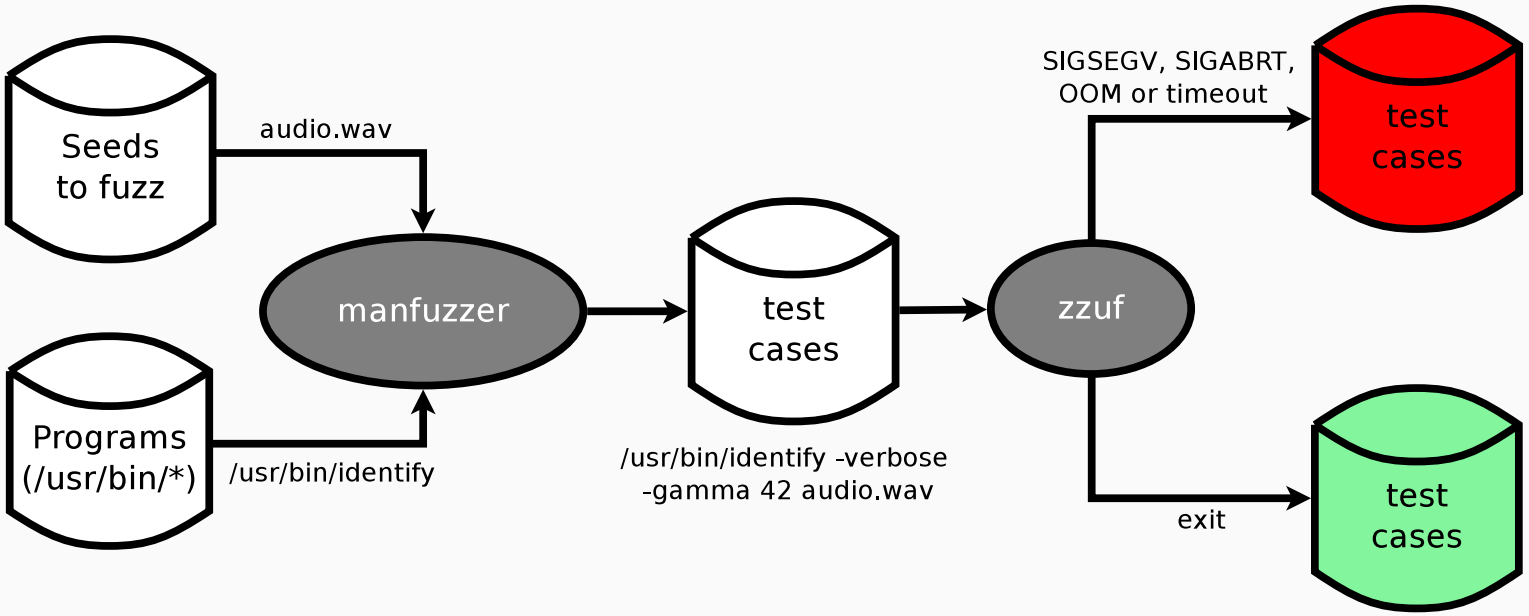
Seed Selection

```
$ tseeder bmptopnm-traces.csv.clusters seeds
Copying seeds..
bmps/badbitcount.bmp
bmps/pal4gs.bmp
bmps/rgba32-61754.bmp
bmps/pal4.bmp
bmps/shortfile.bmp
bmps/baddens2.bmp
```

Question

You can adjust how many test cases per cluster are selected using -n.

ZZUF dataset (1)



A detailed explanation of this dataset is available here:

`http://www.vdiscover.org/OS-fuzzing.html`

ZZUF dataset (2)

- cmds.csv.gz: 64k command-line to fuzz
- traces.csv.gz: sampled and balanced traces ready to be trained and tested
- zzuf.csv.gz: output from zzuf after fuzzing

To split the data in train and test sets:

```
$ ./split.py dataset/traces.csv.gz 42
```

Training and testing a bug predictor

- Training:

```
$ vpredictor --dynamic --train-rf data/42/train.csv --out-file
model.pklz
```

- Testing:

```
$ vpredictor --test --dynamic --model model.pklz data/42/test.csv
--out-file predicted.out
...
Accuracy per class:  0.72 0.78
Average accuracy:  0.75
```