

The good 0(ld) days

Finding old bits of code in binaries in the hope of finding 0day

Thomas Dullien (“Halvar Flake”) -- Google Project Zero
thomasdullien@google.com

Software Supply Chains are complicated

- Including a third-party software component under liberal licenses is “free”
- Unprecedented economics:
 - You want to build X.
 - Many input parts for X are available, and for free!
 - Of course you will use them!
- Explosive innovation in IT in part driven by the vast quantities of high-quality libraries under liberal licenses.
- “Software is eating the world, and a good part of it is open-source.”

Software Supply Chains are complicated

- Dependencies have further dependencies.
- Your third-party library may contain code from another third-party library.
- Enumerating dependencies is nontrivial.

3rd party library security is hard

- How do you monitor 3rd-party libraries for security fixes?
- Does the 3rd-party library even distinguish between security and non-security fixes? If not, can you?
- How quickly can you update if a security flaw in a 3rd-party library is identified?

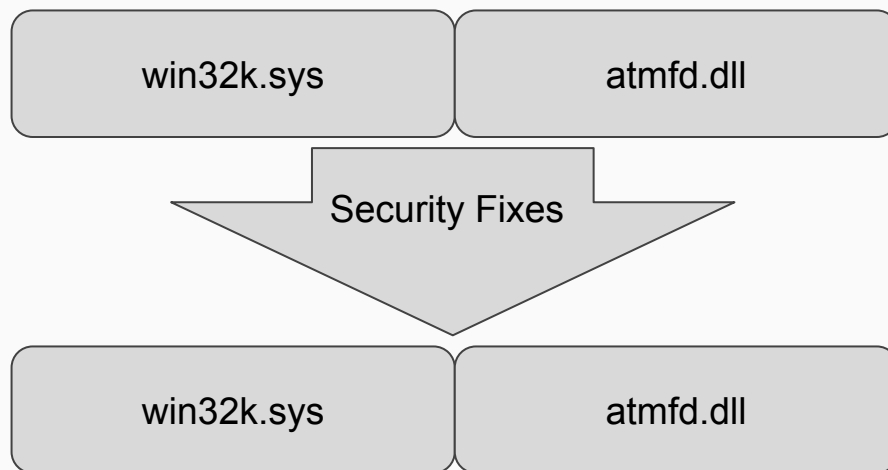
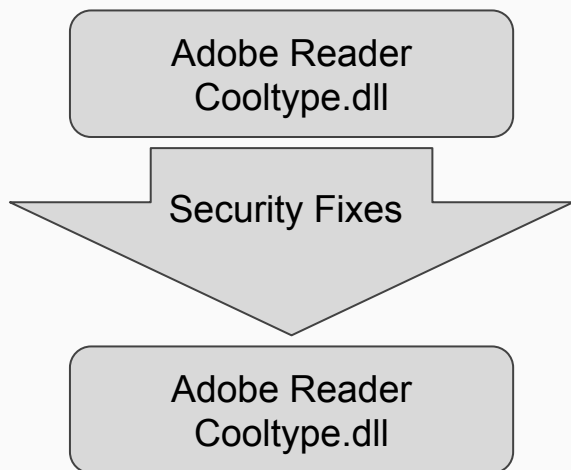
Centralized libraries are high-value targets

- A flaw in a single, well-chosen open-source library can affect dozens of products.
- Gold standard: Zlib (historical example: CVE-2005-2096 - crash in zlib through decompressing a PNG)
- Other libraries with high centrality: Unrar (one bug gives compromise of almost all antivirus engines), libtiff (PDF rendering, browsers, thumbnailers), compression libraries (Brotli) etc. etc.

Examples

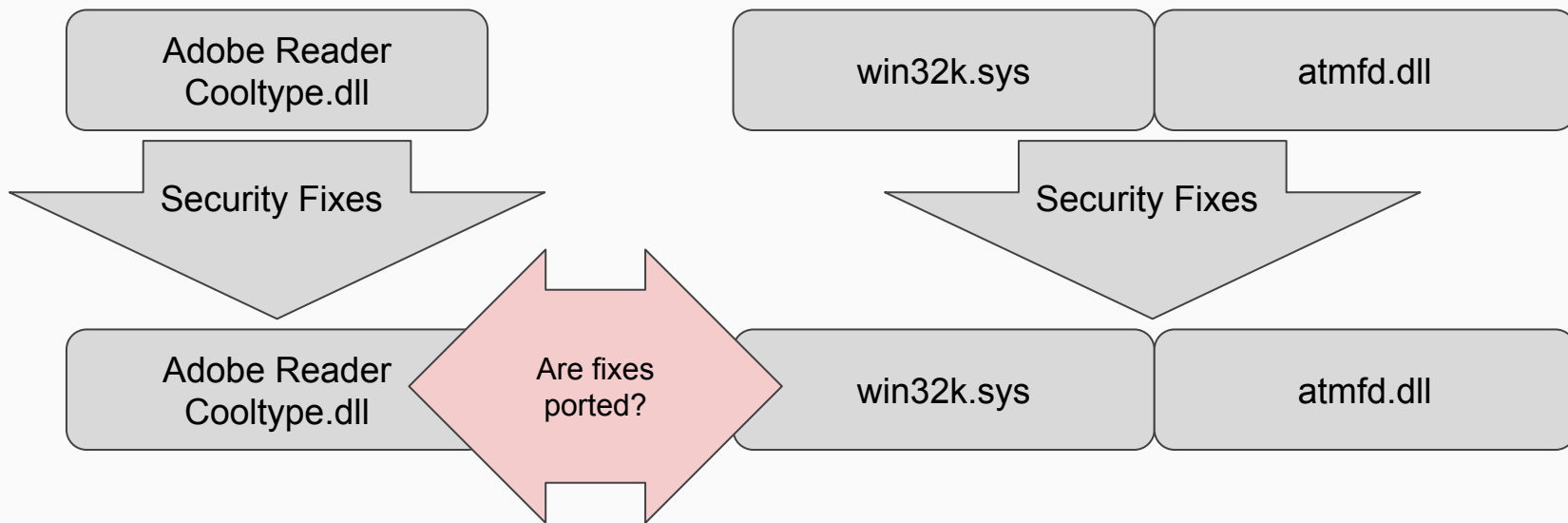
Example: Font parser codebase ancestry

Unknown Codebase,
likely at Apple?

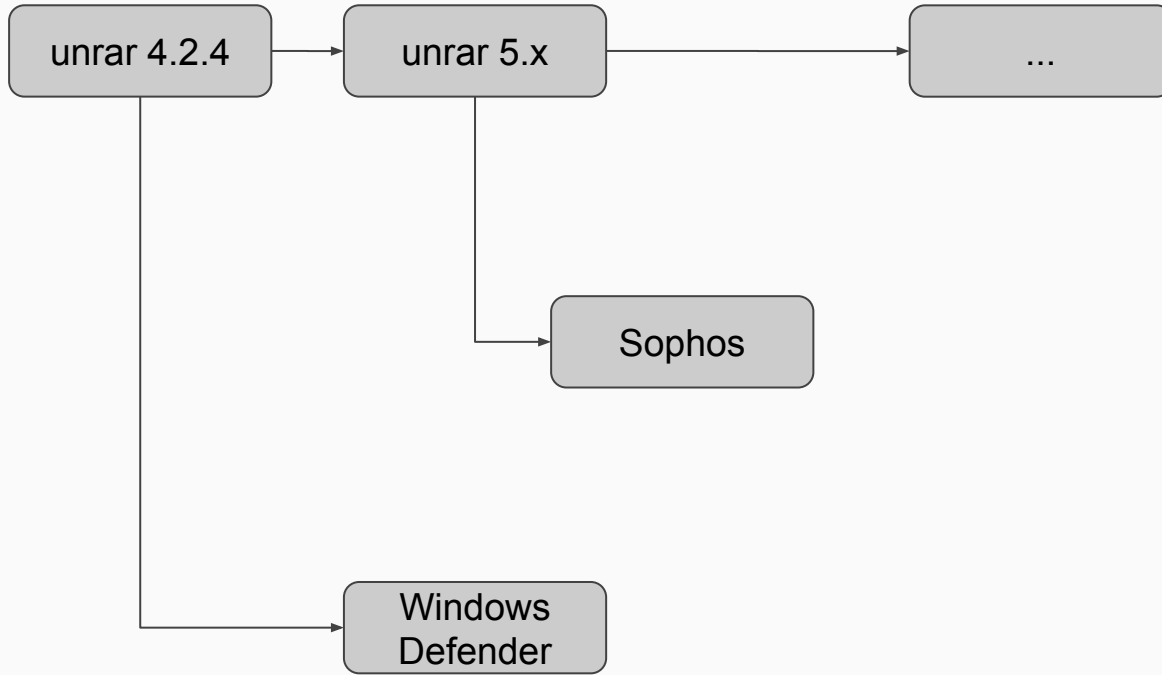


Example: Font parser codebase ancestry

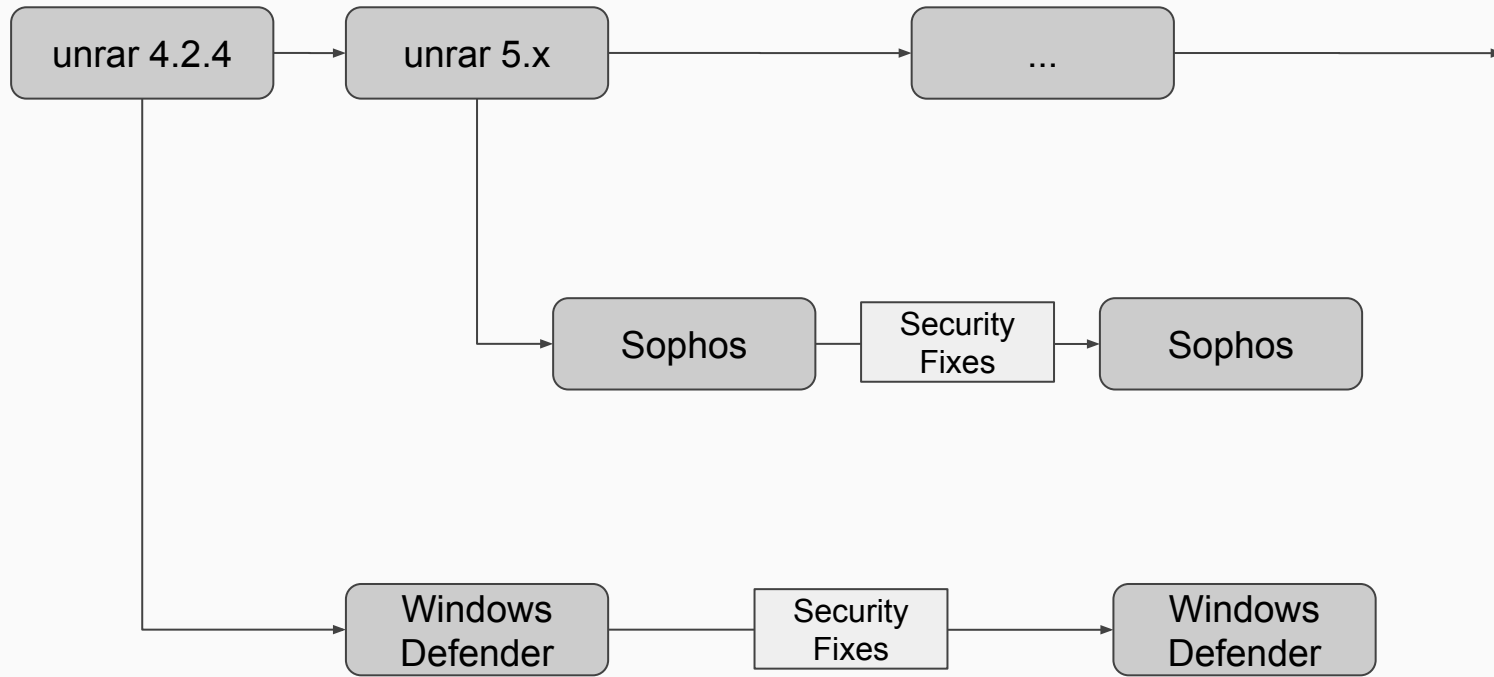
Unknown Codebase,
likely at Apple?



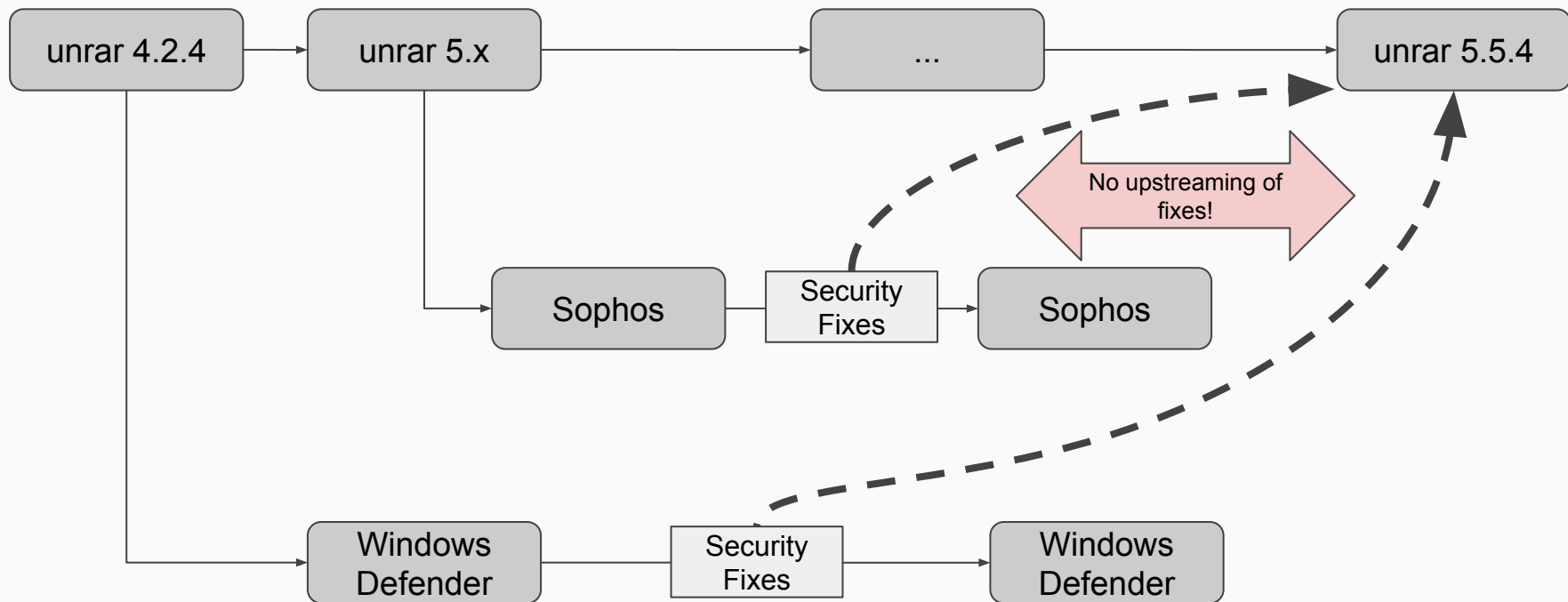
Example: UnRAR



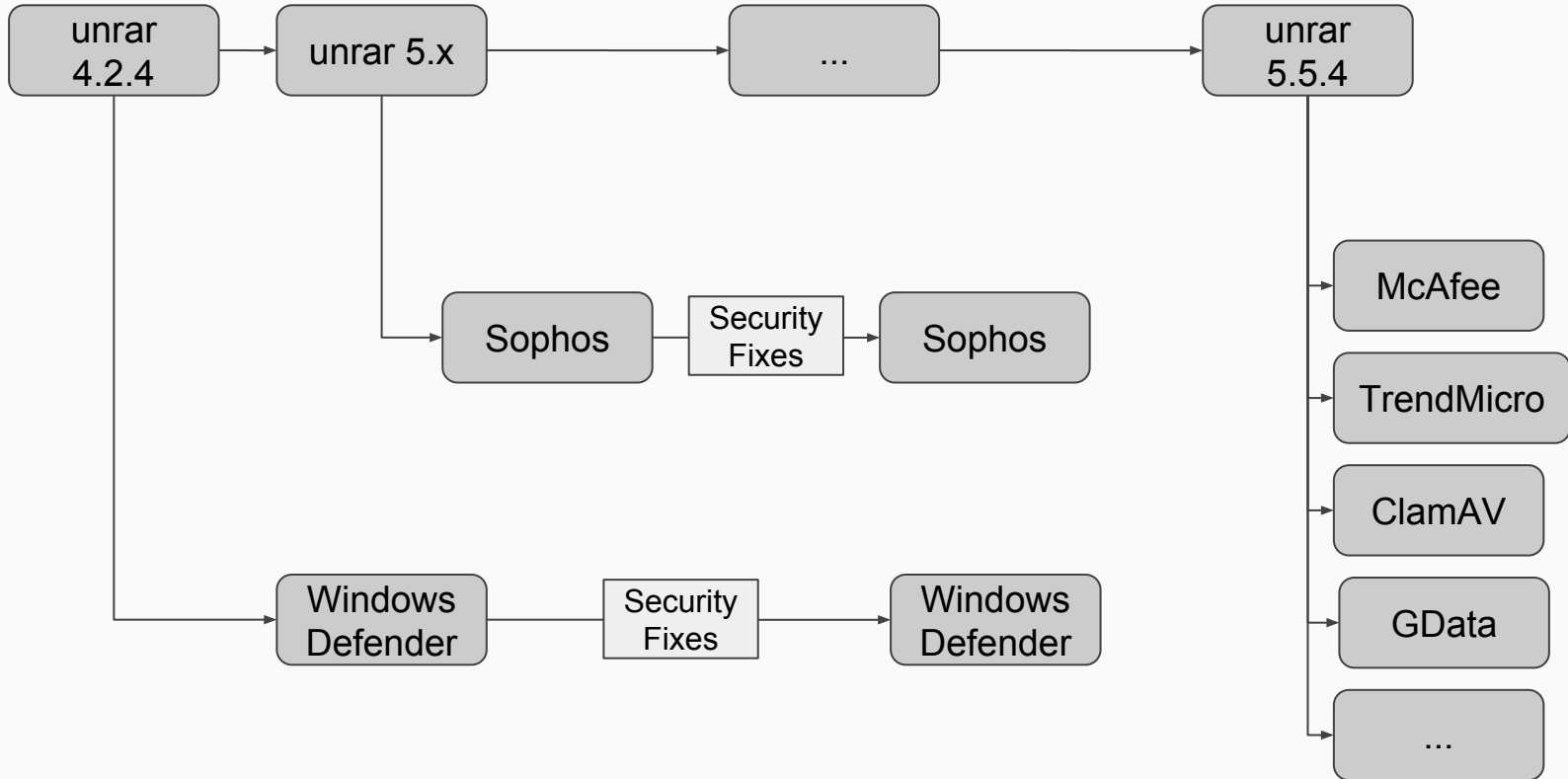
Example: UnRAR



Example: UnRAR



Example: UnRAR



Detection in binaries

Detection of libraries in binaries is difficult

- Compilers change over time
- The libraries themselves change over time
- Only “precise” way of detection: Common strings?

- Control-flow-graph & disassembly can change drastically ...
 - due to compiler changes
 - due to compiler setting changes
 - due to library code changes
 - due to interaction of library code changes with compiler changes & settings changes

Related work

IDA's FLIRT

- Byte signatures with wildcards & extra stack information.

MACHOKE

- Serialize CFG into a canonical string.
- Apply Murmurhash3 on this string.
- No concept of “similar but not equal”.
- Tries to achieve fuzzyness by not taking instructions into account at all.

GENIUS [[Paper](#)]

- Operates on “ACFG” (CFGs annotated with extra data like # of calls, string constants etc.)
- Selects a “codebook” of representative graphs from groups of training graphs (e.g. implementations of the same function) via spectral clustering.
- Associates each codebook-graph with a high-dimensional vector.

GENIUS [[Paper](#)]

- Incoming graph is compared to each representative codebook graph (via calculating a bipartite graph matching), result vector in \mathbb{R}^n is constructed from this.
- Similarity-search is performed using locality-sensitive hashing for approximate nearest neighbors using the \mathbb{R}^n vector.

GEMINI [[Paper](#)]

- GENIUS suffers from ...
 - expensive pre-clustering (for the codebook generation)
 - relatively expensive search (many bipartite graph matchings against codebook)
- GEMINI addresses these problems using Deep Learning.
- Uses “structure2vec” method to compute an R^n embedding from a graph.

GEMINI [[Paper](#)]

- Learn embedding from ACFGs to R^n using “structure2vec” (general graph-to-embedding model)
- “Siamese architecture” (2 parallel runs of the embedding, minimize / maximize resulting distance)
- Given input graph, calculating embedding into R^n is quick, lookup using locality-sensitive hashing for ANN.

GEMINI [Paper]

- Looks powerful.
- Not available publicly.

Practical
considerations.

Motivation

- Reading MPEngine.dll, I recognized unrar code.
- Extensive experience with Adobe's font parser & it's heritage.
- "Can I build something that helps me find third-party libraries for my practical day-to-day-work?"

Design considerations

Automatically recognize & suggest libraries

- Given a function, I should be able to ask: “Does this look similar to anything in my library of 3rd-party-libs?”
- Search should be resilient to small changes in both graph and assembly.

Single-machine setup

- The system should not require extensive setup.
- Should run on a single machine without requiring big databases or distributed key-value stores in the background.
- The system should still allow quick lookup of $O(\text{million})$ stored library functions.

Easy integration with other RE tools

- Tools need to be integrated with RE workflows.
- Vulnerability researchers have heterogeneous setups: IDA, Binary Ninja, Radare2, Miasm, Hopper etc.
- Highly divergent extension APIs, philosophies etc.
- Solution: Compile to Python extension, should be easily accessible from all tools that have a Python Interpreter.

Learning from data

- “Machine Learning” (e.g. automated use of statistical estimation) can help extract useful information from lots of unstructured data.
- Both GENIUS and GEMINI use heavy-duty machine-learning algorithms.
- System under consideration should also allow improvement from data (“learning”).

Inspectability of learnt results

- When the ML algorithm learns something, the results should be “inspectable by an expert”.
- At least initially, I am quite happy to sacrifice accuracy in the search for interpretability of results.
- As confidence in the system grows, I am happy to move to more complicated/powerful ML models.

Easy sharing of results

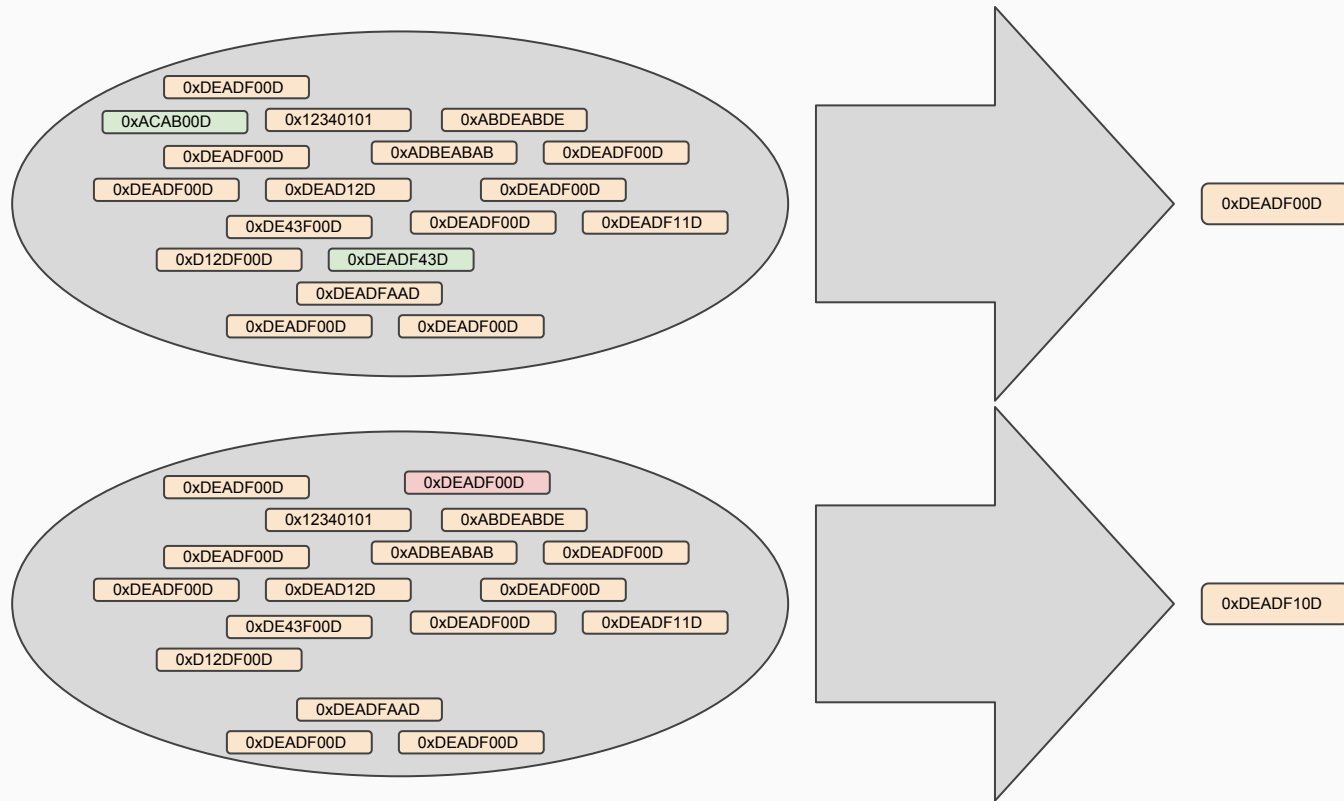
- The primary means of communication is still “email” or “chat”.
- It should be easy to send a friend the “fingerprint” of a given function via email or chat.

Implementation details

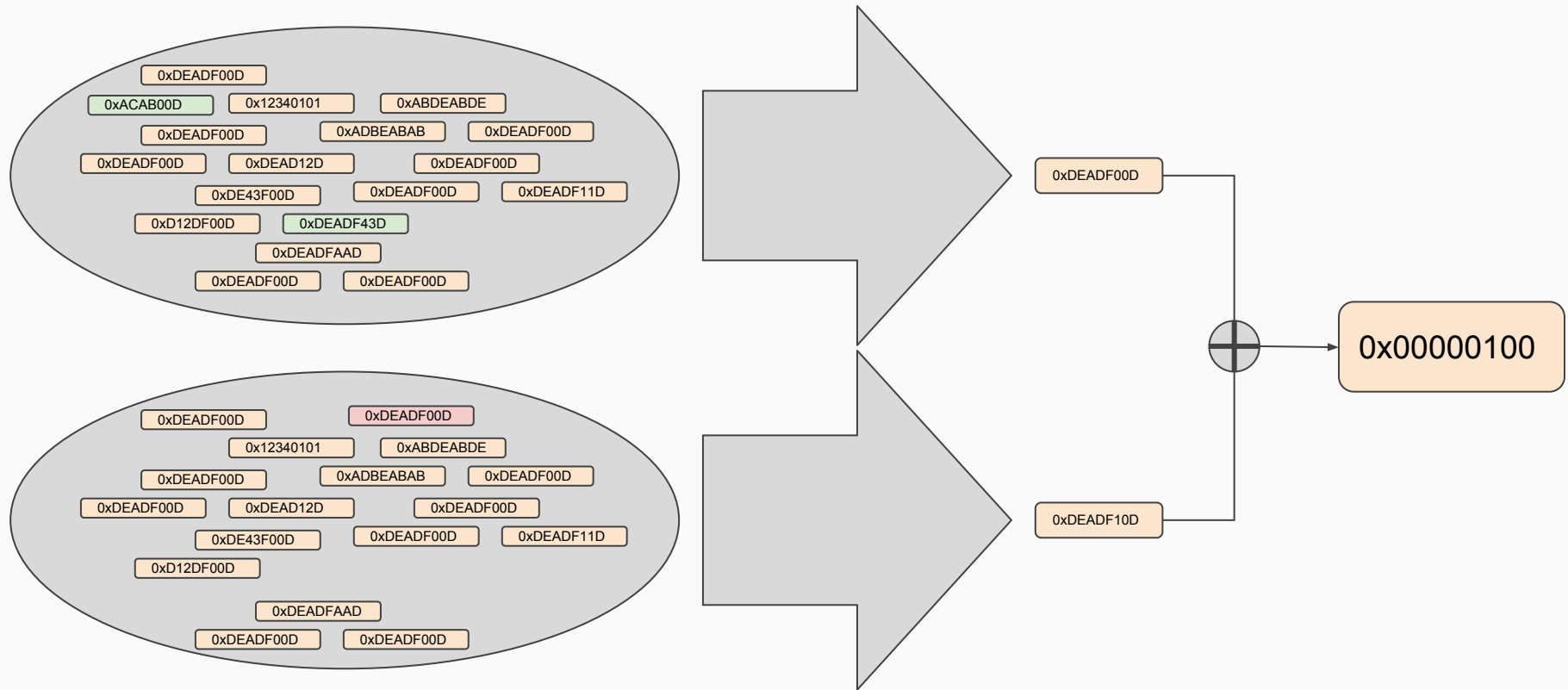
SimHash to obtain compact hashes

- [SimHash](#) provides a method to calculate a “similarity-preserving” hash from a set of feature hashes.
- Given two sets of features extracted from two functions, the SimHashes calculated from the two sets will have low hamming distance if the set similarity was high.
- Very nice properties: Our search index can be 128-bit hashes (compact!)

SimHash to obtain compact hashes



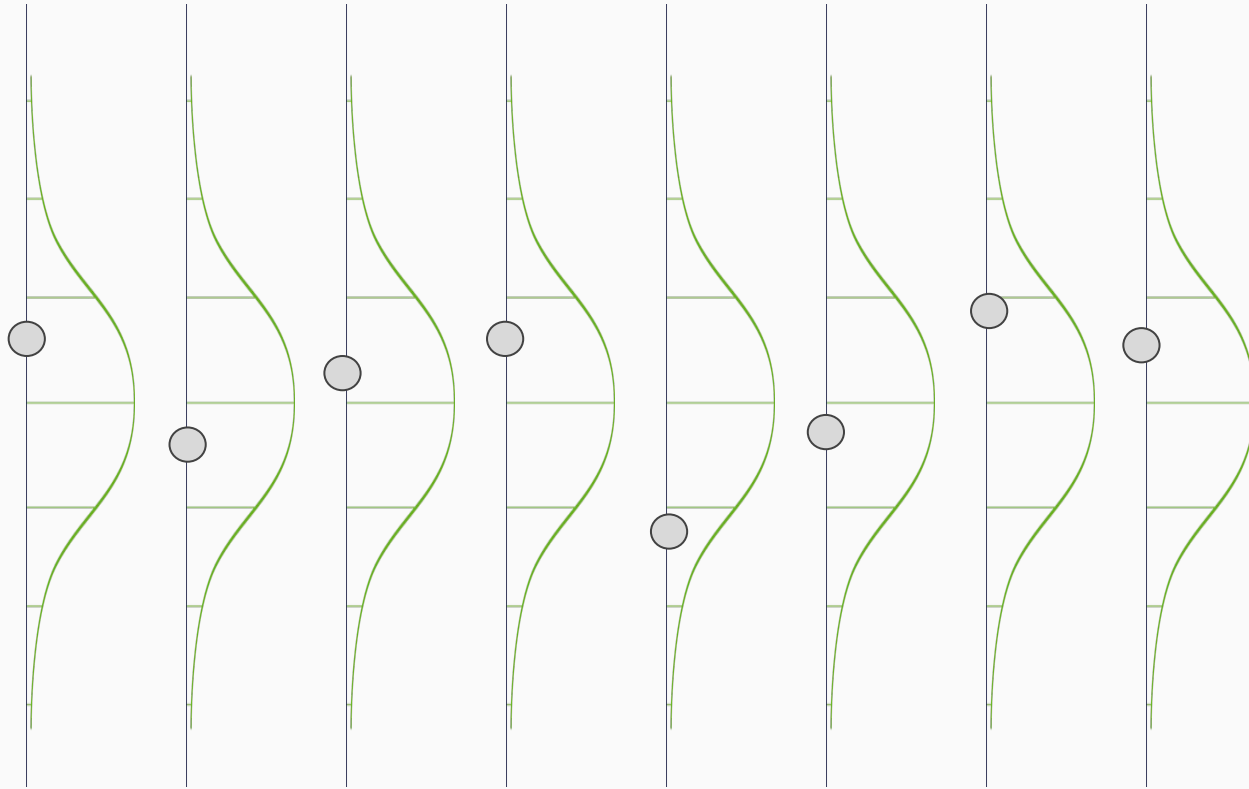
SimHash to obtain compact hashes



SimHash algorithm sketch

- Allocate array of N floats (if your input hashes have N bits)
- For every input hash, do:
 - If the bit `input_hash[k] == 1`, **increment** the `float[k]` by 1.0
 - If the bit `input_hash[k] == 1`, **decrement** the `float[k]` by 1.0
- Convert floats to bits again by assigning positive floats to “1” bits and negative floats to “0” bits.

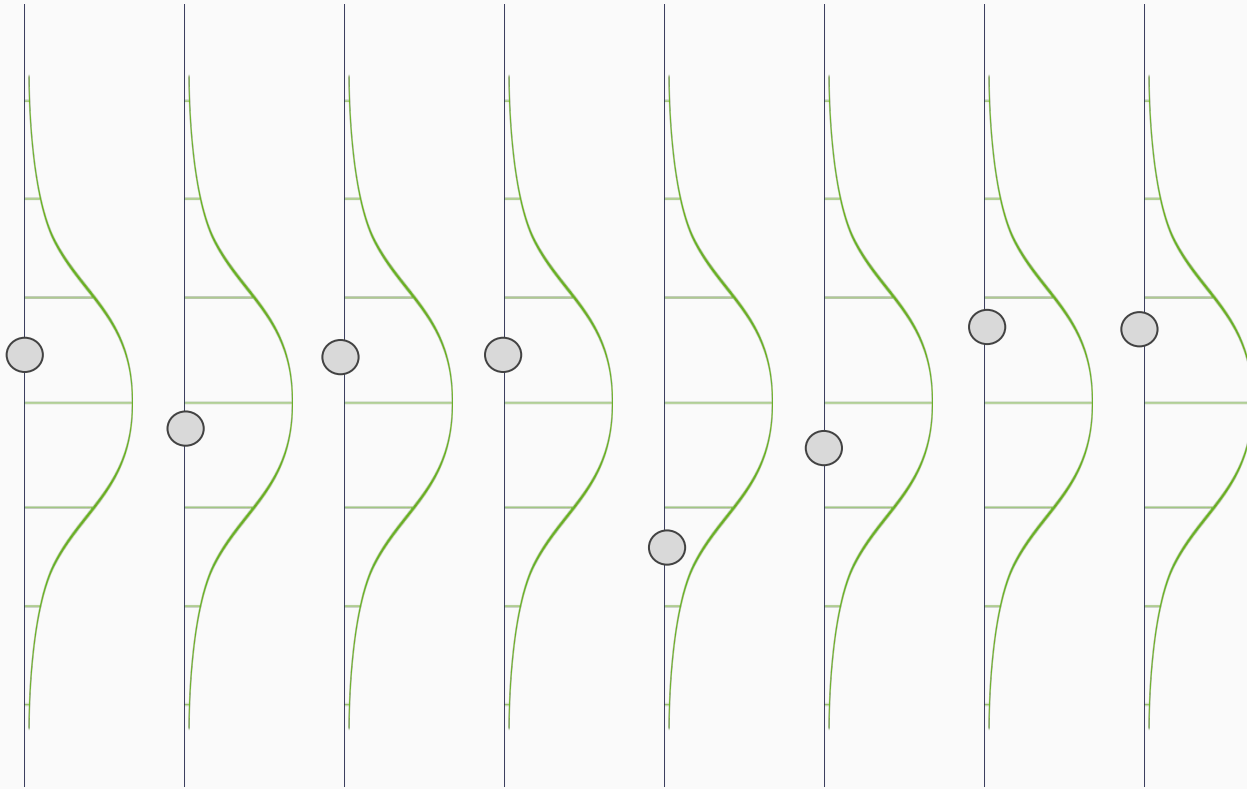
SimHash Illustration: Internal state after K steps



Floats are normally distributed around zero.

With every additional processed feature, they will wobble up or down a little bit.

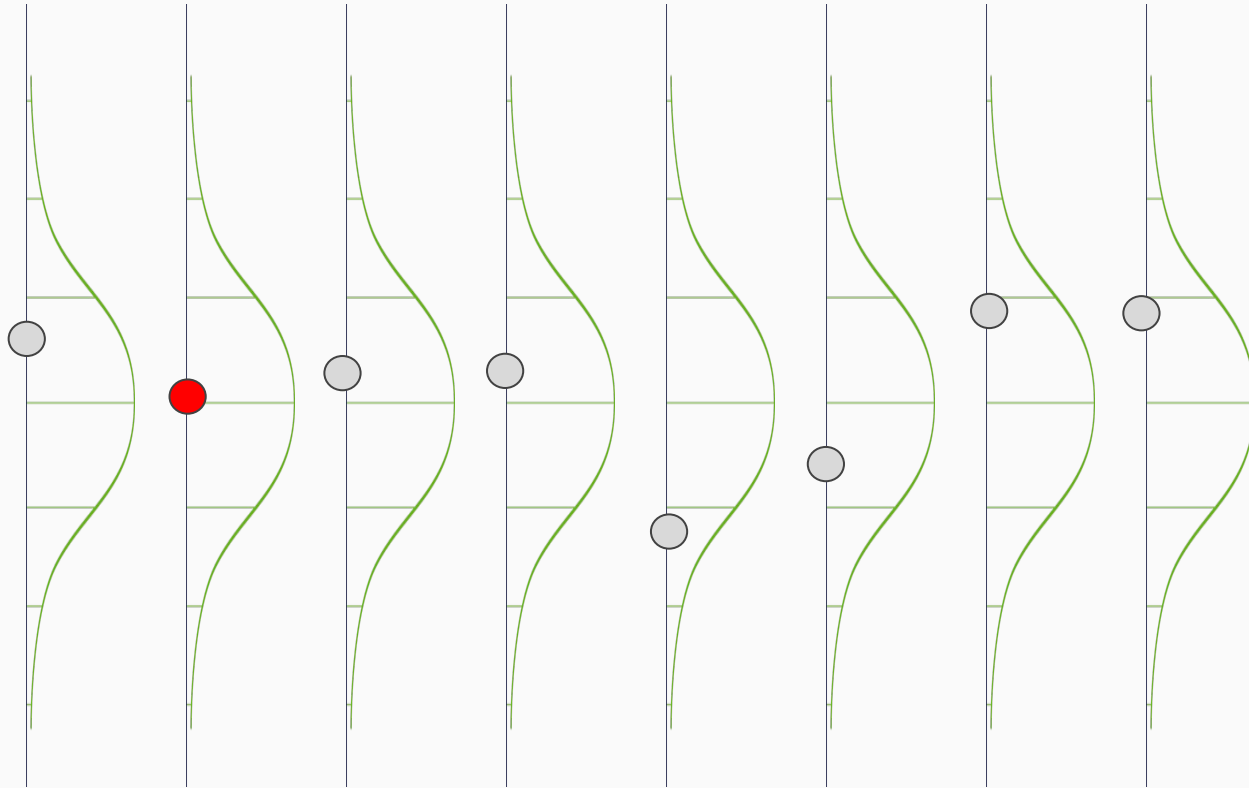
SimHash Illustration: Internal state after K steps



Floats are normally distributed around zero.

With every additional processed feature, they will wobble up or down a little bit.

SimHash Illustration: Internal state after K steps



Floats are normally distributed around zero.

With every additional processed feature, they will wobble up or down a little bit.

Only those that cross the “zero” line will change the resulting hash.

Locality-sensitive hashes via permutation

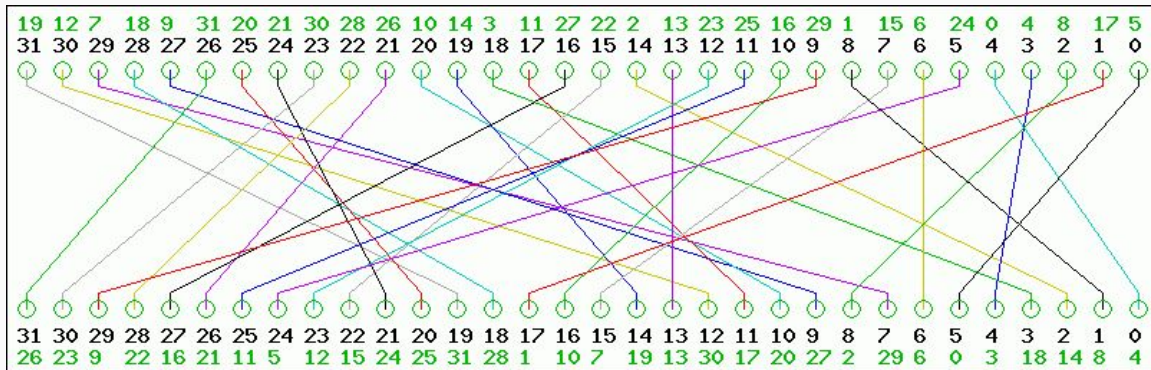
- Similarity search in $O(\text{million})$ of hashes is doable, but $O(\text{million})$ is slow.
- Approximate nearest-neighbor search can be achieved using locality-sensitive hashing.
- LSH: A family of hash functions where nearby points have a higher probability of landing in the same hash bucket than far-away points.

Locality-sensitive hashes via permutation

- Easy for hashes (\sim bit-vectors): Simply take a random subset of bits.
- Pick random permutation. Permute all the bits, then take first N bits as hash.
- Permute & take first N bits again for the next hash.

Locality-sensitive hashes via permutation

- 128-bitwise permutation can be had for ~65 cycles. (I used a cool generator that generates C code for a given bitwise permutation: <http://programming.sirrida.de/calcpERM.php>)



Learning good
weights from
example data.

SimHash with learnt weights

- SimHash uses +1 and -1 as weights.
- Not every feature has the same relevance - function prologues etc.
- How can we best “learn” good weights from examples?
- First, generate labeled examples: Lots of versions of the same function.

SimHash with learnt weights

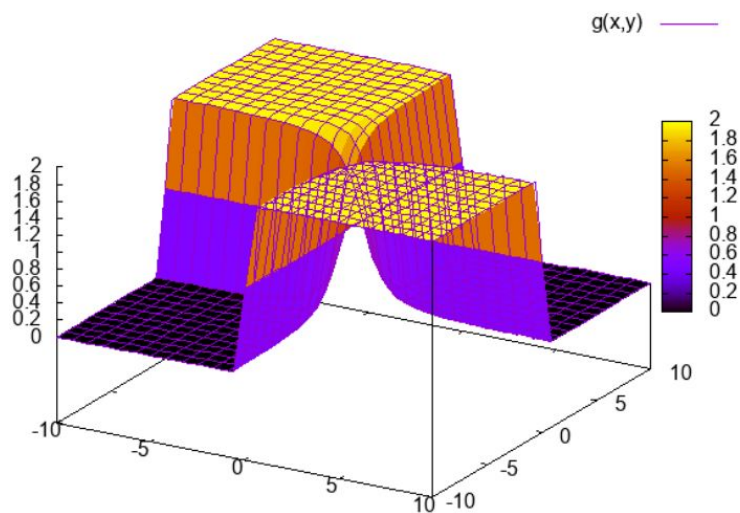
- Can we optimize weights so that ...
 - ... pairs of similar functions get closer together and
 - ... pairs of dissimilar functions get moved to be further apart?
- SimHash distance is Hamming Distance, which is discrete.
- Supervised Machine Learning usually means running optimization algorithms on a differentiable loss function.
- We need something continuous to differentiate so we can “learn weights”.

SimHash with learnt weights

- Before we convert floats to bits again, we have float values for every k
- We can take two vectors of floats and run them component-wise through a function that punishes “same sign” or “different sign”.

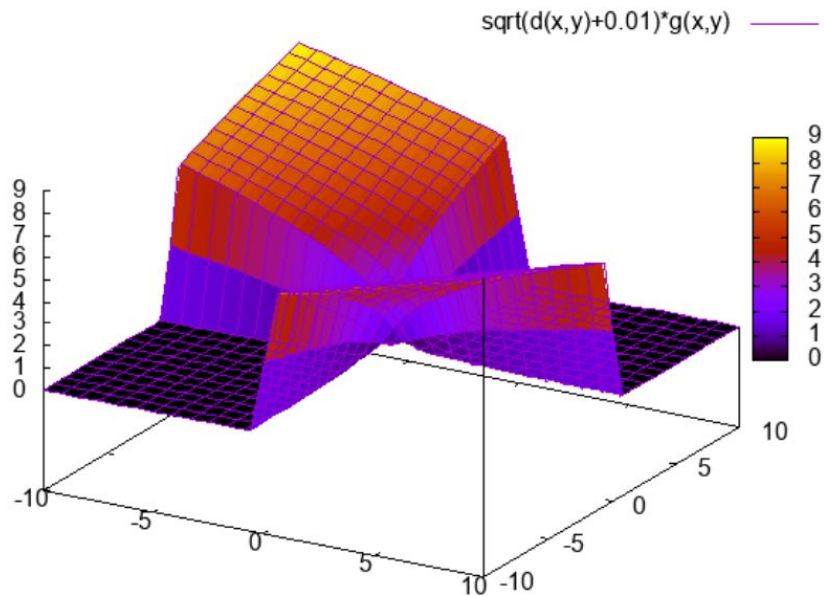
SimHash with learnt weights

$$g(x,y) := -\frac{xy}{\sqrt{x^2y^2+1.0}} + 1.0$$



Smooth step function: No gradient in the flats.

$$d(x,y) := \sqrt{(x-y)^2 + 0.01}$$



Multiply with $d(x,y)$ to slope the flat sections.

Automatic differentiation & minimization

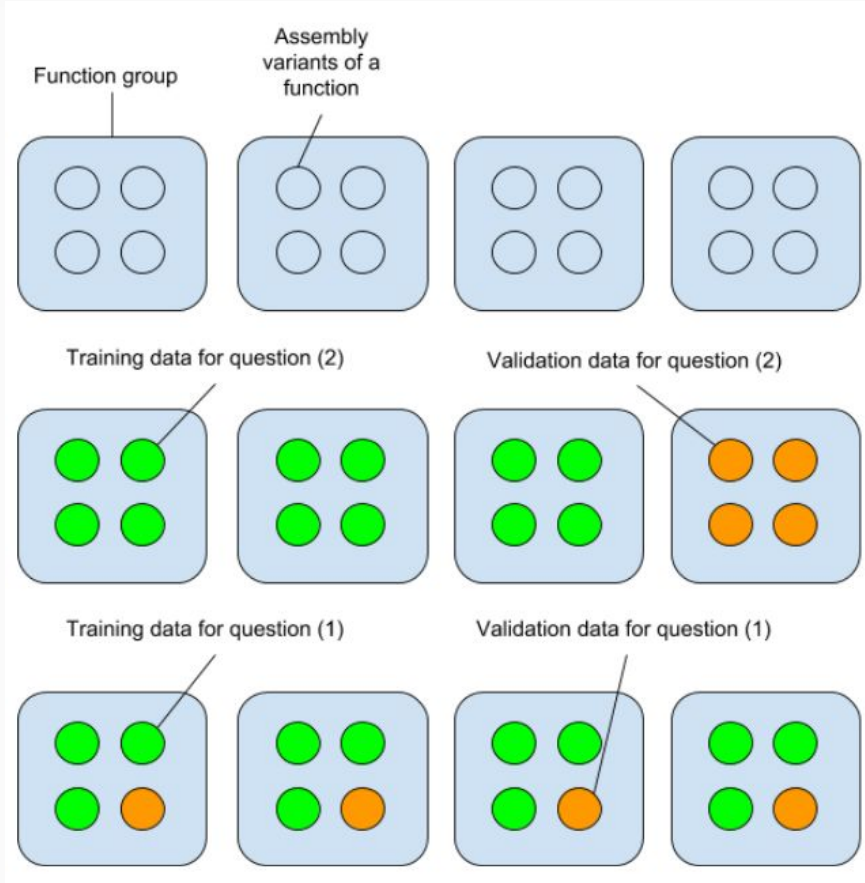
- Many libraries exist to perform automatic differentiation & minimization.
- I wanted a pure C++ codebase, so instead of Python/TensorFlow or Python/Keras or Julia I used a C++ library (SP11) for it.
- This makes using GPUs for training hard, so was probably not a great decision.

Evaluating the
training results.

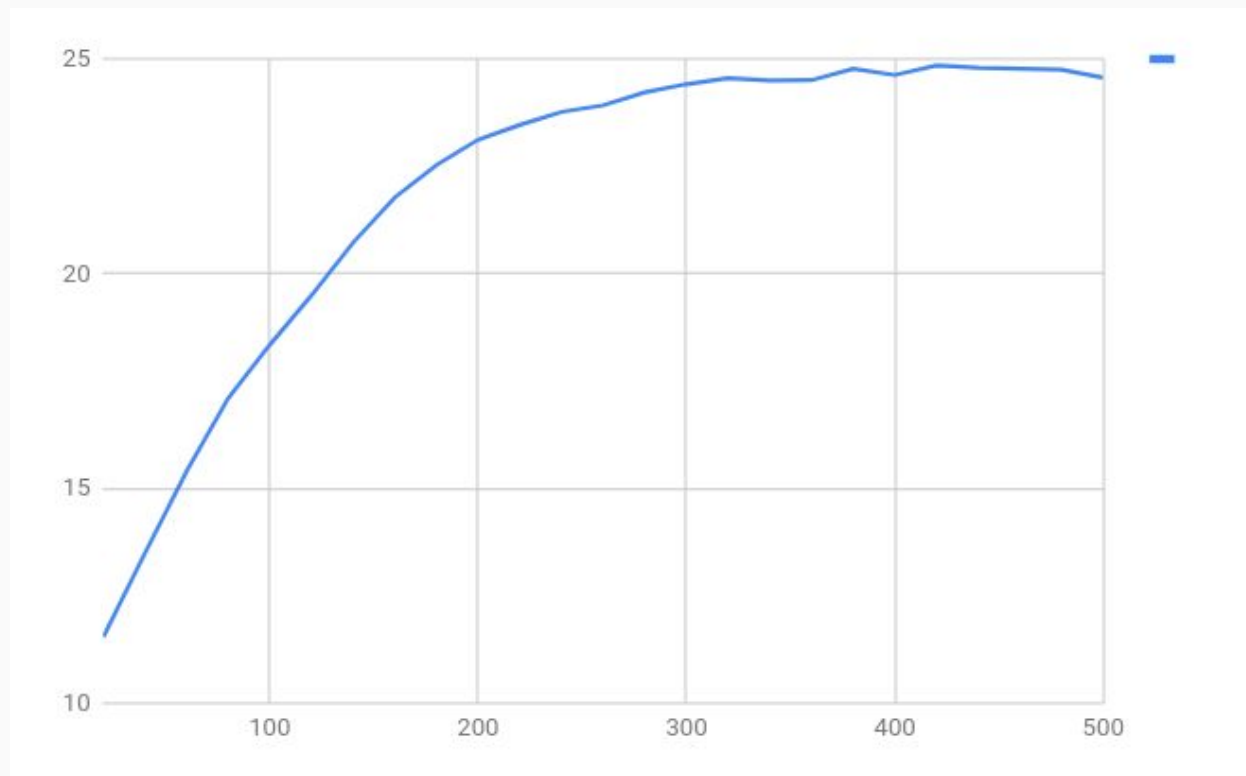
Questions we have:

1. Does the learning process improve our ability to detect variants of a function we have trained on?
2. Does the learning process improve our ability to detect variants of a function **even if we have not seen** a variant of it before?
3. Are the results we get for (1) or (2) practically useful yet?

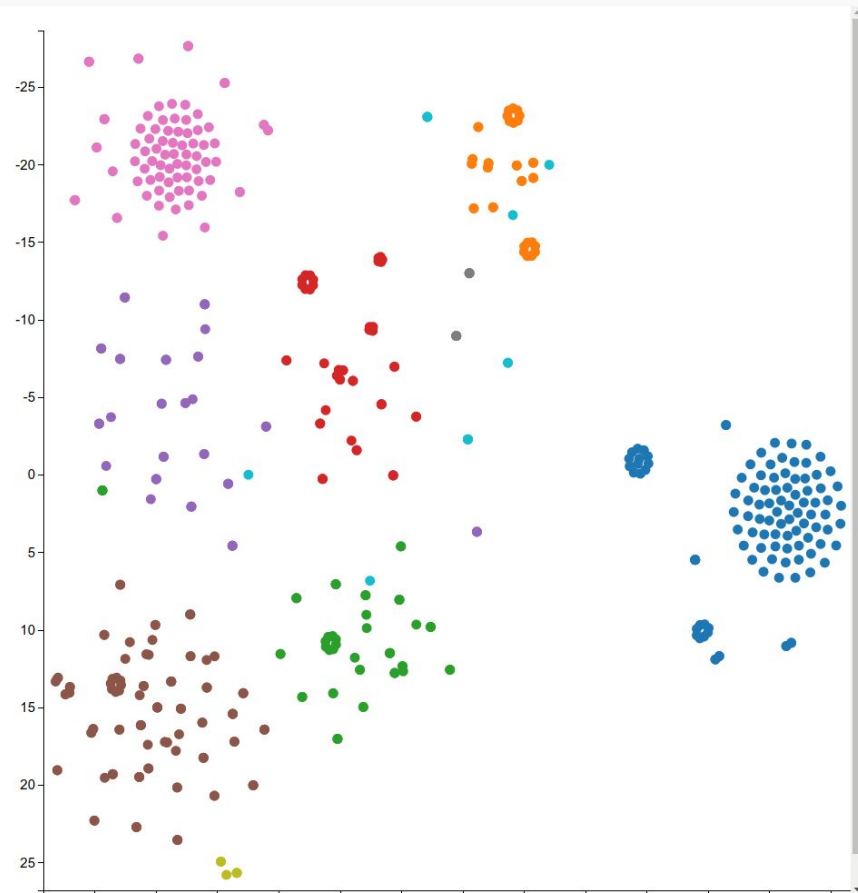
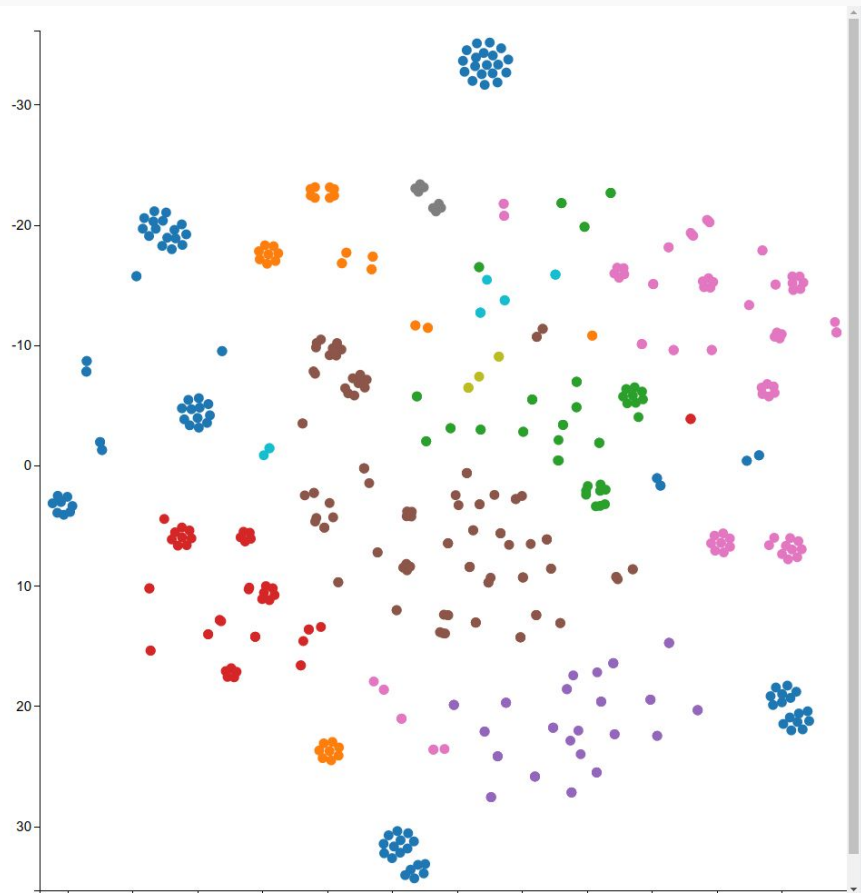
Different ways of splitting data for question 1 and 2



Difference in average distance for between similar and dissimilar pairs after N steps



T-SNE of untrained (left) and trained (right) hash distances



Current state

1. Difference-between-good-and-bad-pairs goes up significantly for functions that we had variants for -- see graph (better separation).
2. Difference-between-good-and-bad-pairs goes up very slightly for functions that we had no variants for (very slightly better separation).

Interfacing with Python, IDA, Binja

Python Interface is very simple

```
git clone https://github.com/googleprojectzero/functionsimsearch
# install only the python bindings
cd functionsimsearch
python ./setup.py install --user
```

Python

```
>>> import functionsimsearch
>>> fg = functionsimsearch.FlowgraphWithInstructions()
>>> fsh = functionsimsearch.SimHasher()
>>> fg.add_node(0x401000)
>>> fg.add_instructions(0x401000, (("push", ("ebp", "")), ("mov", ("ebp", "esp")), ("sub", ("esp", "0x20"))))
>>> fg.to_json()
u'{"edges":[],"name":"CFG","nodes":[{"address":4198400,"instructions":[{"mnemonic":"push","operands":["ebp",""]},{"mnemonic":"mov","operands":["ebp","esp"]},{"mnemonic":"sub","operands":["esp","0x20"]}]}]}'
>>> fsh.calculate_hash(fg)
(7763007128511167962L, 7763007128511167962L)
```


Python Interface is very simple

```
>>> index = functionsimsearch.SimHashSearchIndex("/home/thomasdullien/searchindex", False)
>>> # Add a function with a given SimHash to the index.
>>> index.add_function(hash[0], hash[1], file_id, address)

>>> # Query for the best 5 matches for a given function.
>>> results = index.query_top_N(hash[0], hash[1], 5)
>>> for r in results:
>>>     number_of_identical_bits = r[0]
>>>     file_id_of_result = r[1]
>>>     address_of_result = r[2]
```

Experimental plugins for IDA and Binja exist

- Proof-of-concepts
- Only allow saving & search
- Only text output as UI at the moment

Problems & Challenges

False positive requirements

- Scanning a single large binary can easily involve 60k+ queries.
- False positives are very wasteful of vuln-researcher time.
- False-discovery-rate needs to be somewhere below 0.0001 or even 0.00001.
- We are **not** there yet.

Small graphs

- Method fails spectacularly on small graphs.
- If a change in the graph is only 4 edges away from most other nodes, the graphlets all change and mess up results.
- Different methods will be needed for small functions (context?)

Answer we have so far:

1. Does the learning process improve our ability to detect variants of a function we have trained on? **Yes, measurably.**
2. Does the learning process improve our ability to detect variants of a function **even if we have not seen** a variant of it before? **Yes, but only slightly. So not yet in practice.**
3. Are the results we get for (1) or (2) practically useful yet? **Not yet, due to extremely strict false-positive / false-discovery requirements.**

Some example
functions and
their distances.

Future directions

Reimplement the “learning” code

- Current learning code is implemented in C++.
- Auto-parallelization and GPU offload is made complicated.
- Great libraries for learning exist. This code needs to be rewritten.
 - TensorFlow
 - Keras
 - Most likely: Julia v1.0 (because I like the language)

More powerful ML models

- Our model is extremely simple.
- Features are never considered in their “interaction”, simple linear weights for all features.
- Much stronger models exist:
 - DNN approach from Gemini.
 - Graph NN’s to learn graph structure.
 - RNN’s to learn better embeddings for instruction sequences.

Better features to go into the hash

- Current feature input is subgraphs (“graphlets”) and mnemonic tuples.
- Operands are discarded, string references too.

- Both should be included (they carry a lot of signal !)
- Only obstacle: A clean, cross-disassembler way of parsing constants out of operands.

Questions?

<https://github.com/googleprojectzero/functionsimsearch>