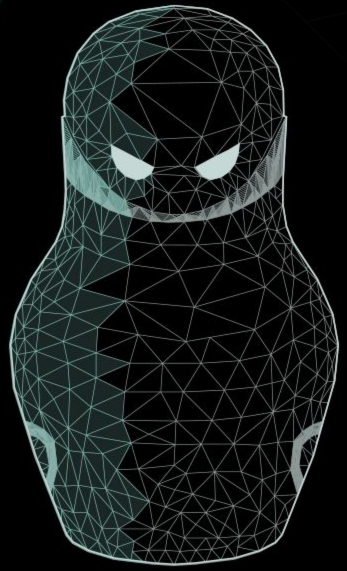


CENSUS
IT Security Works

ZERONIGHTS 2015

Introducing Choronzon: An approach at knowledge-based evolutionary fuzzing

Zisis Sialveras (zisis@census-labs.com)
Nikolaos Naziridis (nullsem@census-labs.com)

$ whoami;

Nikolaos Naziridis

- researcher at CENSUS S.A.
  - Vulnerability research, reverse engineering, exploit development
- fuzzing enthusiast
- aspiring CS graduate student from A.U.Th.
- hates feta cheese

Zisis Sialveras

- trying to graduate from Electrical and Computer Engineer Department in A. U.Th
- works as 9 to 5 reverser and vulnerability researcher at CENSUS S.A
- loves feta cheese

Outline of presentation

- Introduction and motivation
- Related Work
- State of the art fuzzers
- A walk through Choronzon
- Comparison with other fuzzers
- Conclusion / Future Work

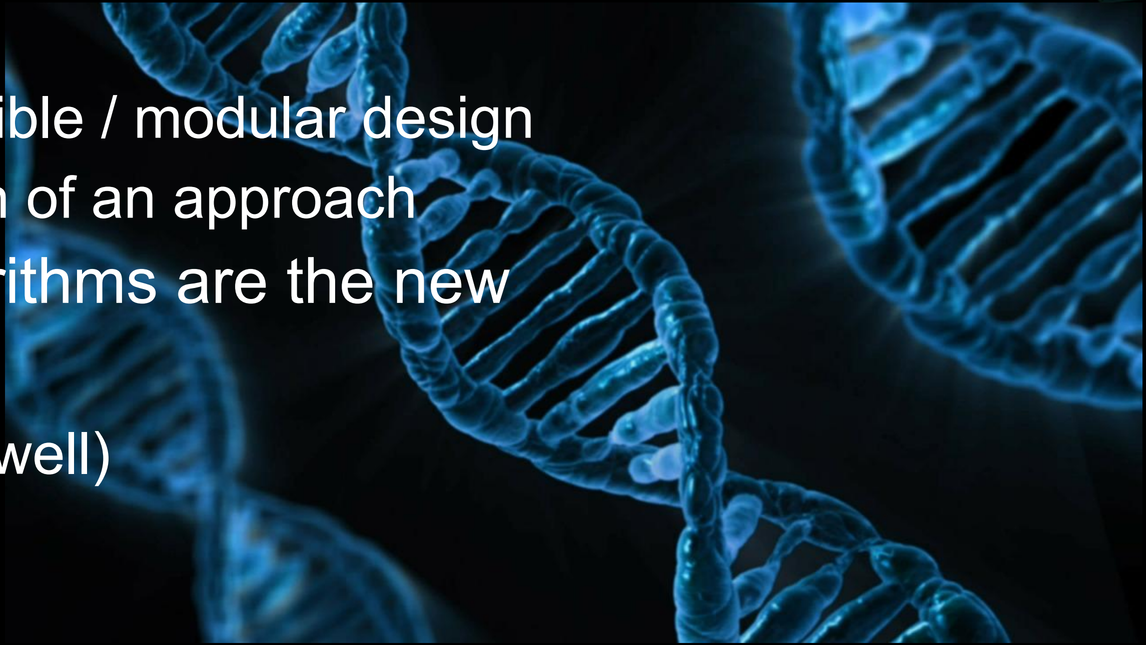
★ WARNING: there are fuzzed images with *Choronzon* throughout the presentation

Motivation 1/2

- Stop re-inventing the wheel
  - “same, same, but different! but still same!” ~ James Franco
  - re-use code and tools developed for other targets
- Targets across many platforms, need cross-platform fuzzers
- Need to attack binary targets
- Insight about a target file format makes a fuzzer smarter than a brick

Motivation 2/2

- Different methods apply better to different targets
  - need for extensible / modular design
  - quick evaluation of an approach
- Evolutionary algorithms are the new hip thing to do
  - (and they work well)



What's evolutionary fuzzing ?

- Paradigm of natural selection applies also to fuzzing
  - genes are mutated randomly
  - new genes are produced
  - only the fittest survive
  - nature explores the possible combinations of genes in a chromosome
- How does this apply to fuzzing ?

Notable prior work 1/2

- Sidewinder:An Evolutionary Guidance System For Malicious Input Crafting/Embletton, Sparks, Cunningham @ BH-USA 2006
  - representation of the file format using Context Free Grammars (CFG)
  - calculates the fitness of a seed file using a statistical model (Markov Process)
  - instrumentation done by setting up breakpoints to each basic block
- Revolutionizing the Field of Gray-Box Fuzzing Attack Surface using Evolutionary Algorithms/DeMott @ BH-USA & DefCon 2007
  - designed to work specially for network protocols
  - introduced interesting approaches to mutation of seed files

Notable prior work 2/2

- Automated Whitebox Fuzz Testing -- Godefroid, Levin, Molnar @ Microsoft Research 2008
  - symbolic execution
  - instruction instrumentation using iDNA framework
  - presents very interesting results and observations
- Many more academic (and not) papers can be found on the Internet

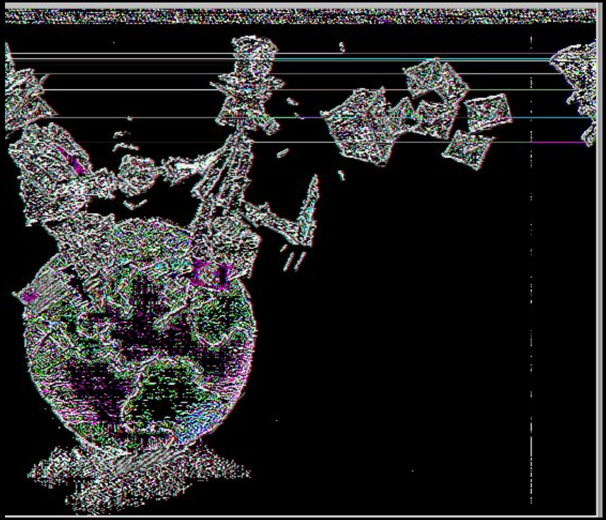
State-of-the-art fuzzers

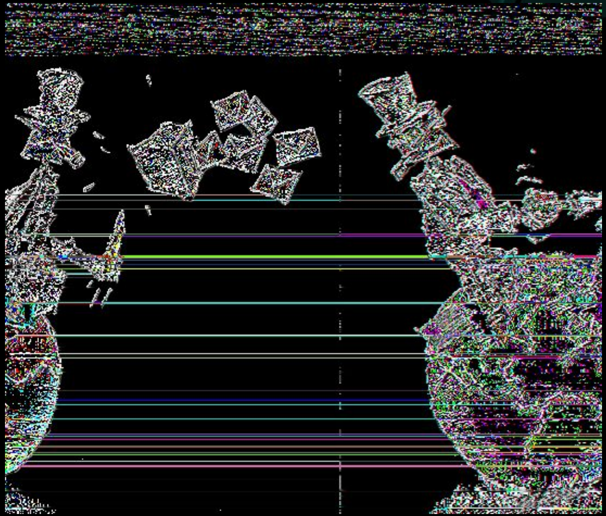
- Asking who's the best fuzzer out there is like asking which is the best programming language (it's C)
  - obviously, the major concern is to produce crashes
  - however, can the fuzzer handle all possible situations ?
  - how about performance, robustness and ease of use ?
- Quick overview of modern open-source fuzzers
  - advantages and disadvantages
  - unique features

State-of-the-art fuzzers

	FOE	Peach	honggfuzz	AFL
Windows Support	✓	✓		
Feedback driven			✓	✓
Source Code Agnostic	✓	✓	✓	
Inter-file mutation				✓
Aware of file format		✓		

Introducing Choronzon





Introducing Choronzon

- A few buzzwords:
  - modular
  - distributed
  - cross-platform
  - source-code agnostic
  - knowledge based
- Target range:
  - Intel architecture
  - Linux, Windows, OSX, Android
  - binary targets

Architecture overview

- Central driver for fuzzing seems nice, but it’s really not
  - database solutions are VERY slow
- Dump interesting files every now and again
- Watch-dog process updates interesting files pool regularly from the dump directory
  - watch-dog monitors remote share for communication among multiple instances of Choronzon
  - rudimentary file locking to avoid race conditions

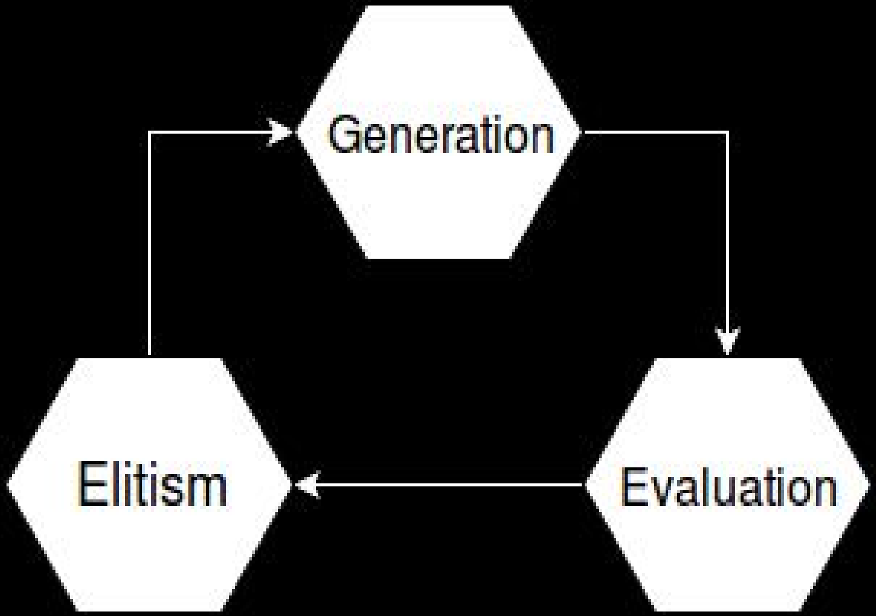
Terminology

- **Chromosome** == seed/test file
- **Gene** == basic structural component of a file
- **Generation** == a collection of mutated files
- **Elitism** == the process of selecting the best test files
- **Population** == the current and previous generation
  - an elite generation and its derived generation
- **Fitness** == scalar score of each chromo after evaluation
- **Metric** == fitness is calculated using various metrics

A round of fuzzing 1/2

- A round of fuzzing with Choronzon
    1. evaluation of initial collection of seed files
    2. best seed selection based on execution metrics
    3. generation of new test files from the best seeds
    4. evaluation of new generation
    5. check for crashes ;)
    6. goto (2)

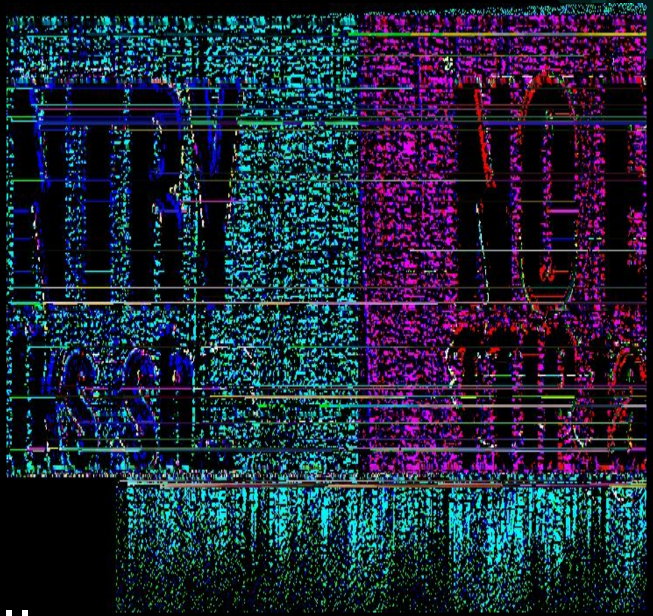
A round of fuzzing 2/2



File format API

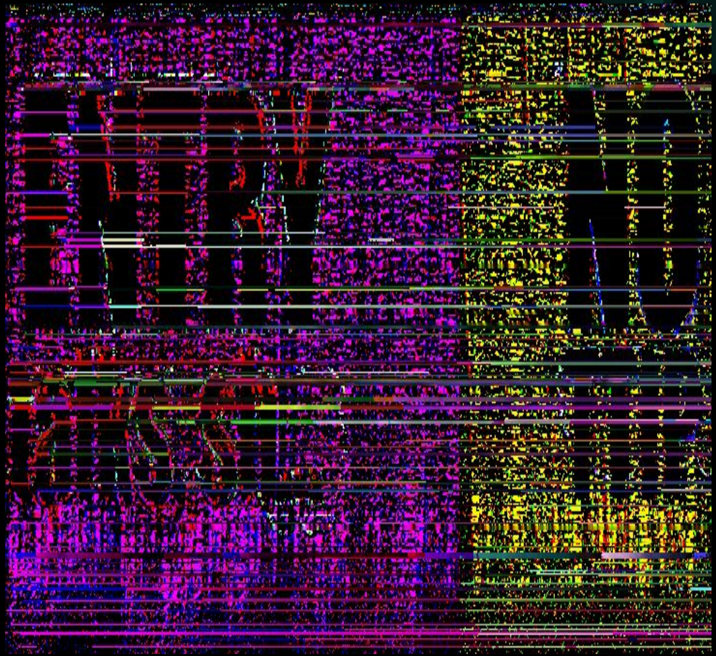
File format API overview

- Converts files to chromosomes
- Focus on flexibility
- API available in python
- Hooks for every step of the parsing
    - custom de-/serializers
    - selection of fuzzable data
    - pre-/postprocessing of test cases
        - checksum fixing, compression, ...



A taxonomy of file formats

- Chunk-based (PNG, SVG)
  - order of chunks matters
- Hierarchical (XML, MP4)
  - parent/child relationship matters
- Index-based (ELF, FAT32)
  - index tables with positions
- Container (ZIP, DOCX)
  - data encoded in other formats



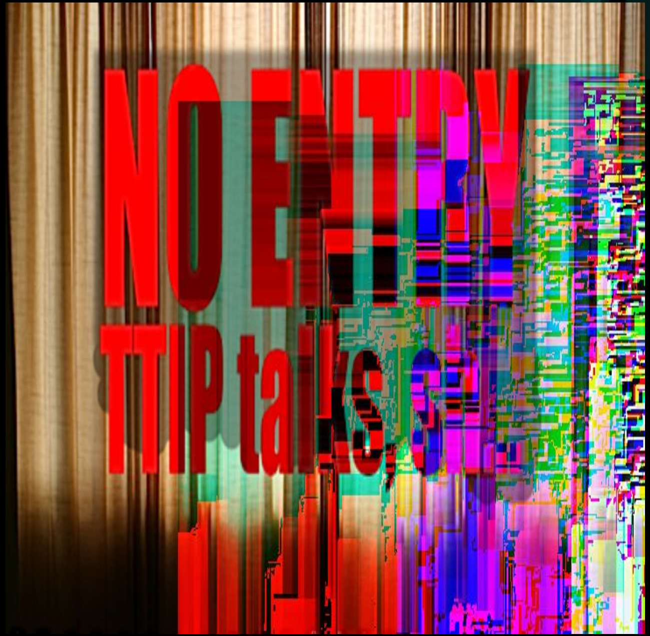
Supported format categories

- Choronzon supports
  - chunk-based formats
  - hierarchical formats
  - container formats
    - can be stripped away
    - or fuzzed liked the other two
- Indexed not supported yet
  - quite difficult to generalize

CENSUS
IT Security Works

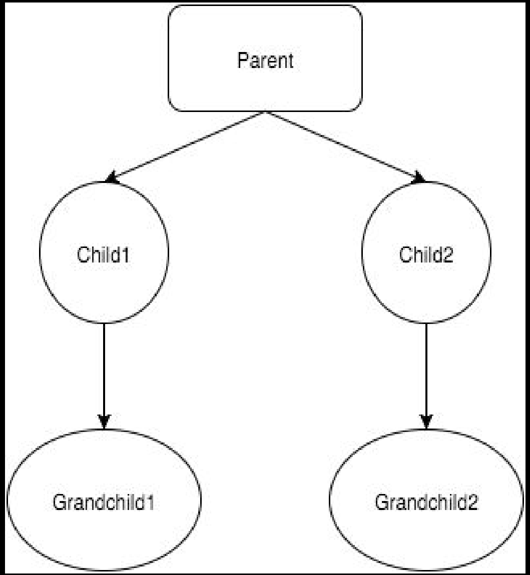
Memory representation

- Genes are distinct conceptual entities
- Input file becomes a tree of genes
    - serialize() : file -> tree of genes
    - deserialize() : tree of genes -> file
- A chromosome is the tree of genes, the serializer and the deserializer
- Fuzzers only deal with chromosomes



A simple gene tree

```
<parent>
    <child1>
        <grandchild1>
        </grandchild1>
    </child1>
    <child2>
        <grandchild2>
        </grandchild2>
    </child2>
<parent>
```



Let’s look at a real-world example

PNG file structure

8-byte signature

Length	Name	Data	CRC
4 bytes	4 bytes	*Length* bytes	4 bytes

...

Length	Name	Data	CRC
4 bytes	4 bytes	*Length* bytes	4 bytes

Example step 1: Gene

```
class PNGGene(gene.AbstractGene):
    '''
        The PNGGene represent a png chunk.
    '''

    def __init__(self, chunk):
        super(PNGGene, self).__init__()
        self.length = chunk['length']
        self.name = chunk['name']
        self.data = chunk['data']
        self.crc = chunk['crc']
```

Example step 2: Gene *serialize()*

```
def serialize(self):
    // dump chunk
    bytestring = ''
    data = self.get_data()
    bytestring += struct.pack('>I', len(data))
    bytestring += struct.pack('>I', self.name)
    bytestring += data
    bytestring += struct.pack('>I', self.crc)

    // post-process
    return self.fix_crc(bytestring)
```

Example step 3: Deserializer

```
class PNGDeserializer(deserializer.BaseDeserializer):
    '''
        A parser for PNG files.
    '''
    fsize = None
    fstream = None
    chunks = None

    def __init__(self):
        super(PNGDeserializer, self).__init__()
        self.fsize = 0
        self.fstream = None
        self.chunks = list()
```

Example step 4: *deserialize()*

```
def deserialize(self, filename):
    # open PNG file
    genes = list()
    self.__open_file(filename)
    self.__parse_signature()

    // parse PNG
    while self.fsize > self.fstream.tell():
        chunk = dict()
        chunk['length'], = struct.unpack('>I', self.fstream.read(4))
        chunk['name'], = struct.unpack('>I', self.fstream.read(4))
        chunk['data'] = self.fstream.read(chunk['length'])
        chunk['crc'], = struct.unpack('>I', self.fstream.read(4))
        self.chunks.append(chunk)
```

Example step 4.5: *deserialize()*

```
# pre-processing
self.__inflate_IDAT_chunks(self.chunks)

# initialize PNG genes
for chunk in self.chunks:
    genes.append(PNGGene(chunk))
return genes
```

Example step 5 - Serializer

```
def serialize(self, genes):
    bytestring = PNG_SIGNATURE
    # pre-processing
    deflated_genes = self.__deflate_IDAT_chunks(genes)
    for gene in deflated_genes:
        bytestring += gene.serialize()
    return bytestring
```

Seed evaluation

Evaluation overview

- Evaluate test files using execution statistics
    1. disassemble executables into basic blocks
    2. trace the execution of target executable images
    3. use the traced basic blocks and the blocks of the images to calculate metrics, like basic block coverage
    4. use metrics to calculate fitness, a scalar value
- Elitism is just sorting the generation by chromosome fitness

Collecting execution traces

- Alternative solutions to execution tracing
  - binary instrumentation, e.g.: Intel PIN, Dynamorio
  - binary patching, e.g.: libdyninst
  - source-code instrumentation, e.g.: AFL approach
  - kernel provided statistics, e.g.: IPT/BTS
  - plain ol' debugging
- All have advantages and disadvantages
- Binary instrumentation is the most generic technique

Instrumentation in Choronzon

- Binary Instrumentation with Intel PIN
    - pros
        - most robust solution
        - works on all major platforms
        - easy to setup and use
    - cons
        - staggering performance overhead (at least 30%)
        - works only on Intel architectures

Disassembly in Choronzon

- Disassembling with IDA Pro
  - pros
    - works on most major platforms / architectures
    - supports wide range of formats
    - easy integration in our workflow
  - cons
    - it’s heavy
    - it’s buggy

Fitness and metrics 1/2

- Choronzon should walk as many unique paths in an executable image as possible
  - If a test file discovers a new basic block, it’s elite
  - elite chromosomes survive to the next generation
- If two chromosomes have discovered the same path, compare their fitness to resolve the conflict

Fitness and metrics 2/2

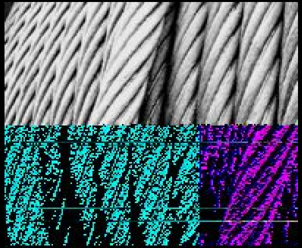
- Metrics used in Choronzon
  - *basic block coverage* (unique trace bbls / total image bbls)
  - *code commonality* (unique trace bbls / total trace bbls)
- Fitness is the combination of coverage and commonality
  - metrics are not equally important
    - preliminary normalization
    - use weights on normalized metrics to calculate fitness

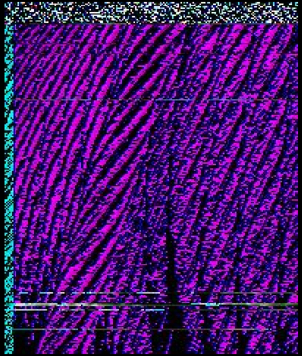
Fuzzing

Ben Nagy
@rantyben

Follow

Watching image fuzzers run is like sprinting through an abstract art exhibition. On acid. Forever.





ZERO NIGHTS 2015

Fuzzing Methodology

- Fuzzing is similar to random walking inside a search space
  - aggressive mutations search for global maxima
  - smalls changes explore local maxima
- The two basic concepts of fuzzing in Choronzon are
  - Mutators
  - Recombinators
- “Bogus files find few bugs” ~ Godefroid

Mutators gonna mutate

- Genes are responsible to pass the fuzzable data to the mutators.
- Choronzon implements a wide range of mutators.
  - bytestring and line-ended string specific mutators
  - add/swap/remove/duplicate bytes, words, dwords and qwords
  - set or unset high bits
  - regular expression based mutators
  - and the powerful random byte mutator!

Recombinators 1/2

- Recombination is not a common feature among fuzzers
  - it seems to work well though
- In Choronzon, recombinations can occur between two chromosomes
  - add a gene from one chromosome to the other
  - swap similar genes between two chromosomes

Recombinators 2/2

- Or it can mean the restructuring of a single chromosome
  - adding/duplicating/swapping/removing genes from a chromosome
  - changing the position of a single gene in the chromosome
  - shuffling of two or more genes in the chromosome
- Hierarchical recombination
  - smart recombination for chromosomes with complex gene trees

Recombination Example

```
<parent>
    <child1>
        <grandchild1>
        </grandchild1>
    </child1>
    <child2>
    </child2>
</parent>
```

```
<PARENT>
    <CHILD1>
    </CHILD1>
    <CHILD2>
        <GRANDCHILD2>
        </GRANDCHILD2>
    </CHILD2>
</PARENT>
```

Two chromosome recombination

```
<parent>
    <child1>
        <GRANDCHILD2>
        </GRANDCHILD2>
    </child1>
    <child2>
    </child2>
</parent>
```

```
<PARENT>
    <CHILD1>
    </CHILD1>
    <CHILD2>
        <grandchild1>
        </grandchild1>
    </CHILD2>
</PARENT>
```

Two chromosome recombination

```
<parent>
    <child1>
        <grandchild1>
            <CHILD2>
                <GRANDCHILD2>
                </GRANDCHILD2>
            </CHILD2>
        </grandchild1>
    </child1>
    ...
```

```
<PARENT>
    <CHILD1>
    </CHILD1>
</PARENT>
```

Single chromosome recombination

<parent>
<child1>
<grandchild1>
</grandchild1>
</child1>
<child2>
</child2>
</parent>

<parent>
<child2>
</child2>
<child1>
<grandchild1>
</grandchild1>
</child1>
</parent>

Hierarchical recombination

```
<parent>
    <child1>
        <grandchild1>
        </grandchild1>
    </child1>
    <child2>
    </child2>
</parent>
```

→

```
<parent>
    <child1>
    </child1>
    <grandchild1>
    </grandchild1>
    <child2>
    </child2>
</parent>
```

Radical hierarchical recombination

```
<parent>
    <child1>
        <grandchild1>
        </grandchild1>
    </child1>
    <child2>
    </child2>
</parent>
```

```
<parent>
    <grandchild>
        <child1>
        </child1>
    </grandchild1>
    <child2>
    </child2>
</parent>
```

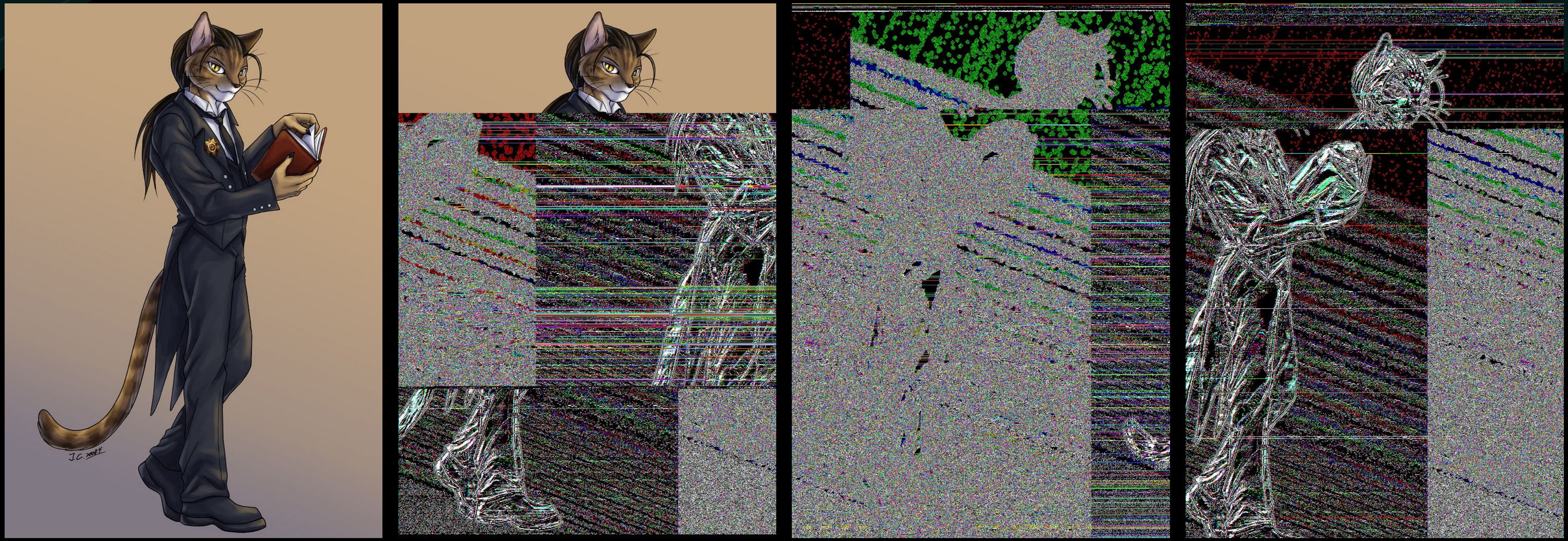
Dynamic fuzzer evaluation

- Different pairs of mutators and recombinators apply better to different file formats
- Choronzon uses a lottery "scheduling" algorithm
  - instead of a time share, a pair is allowed to generate a file
  - if it's elite, the pair is more likely to be selected again
- Evaluation feedback drives the generation of new chromosomes

Fuzzer evaluation samples

Recombinator - Mutator	score(PNG)	score(DOCX)
AdditiveSimilarGene - RandomByte	6	1
RandomGeneInsert - SwapDword	5	1
SimilarGeneInsert - SetHighBit	5	0
DuplicateGene - AddRandomData	1	4
SimilarGeneSwap - SwapLines	0	4
AdditiveSimilarGene - RemoveLines	1	5

Choronzon vs AFL vs Honggfuzz



Comparison of the fuzzers

- Comparison of implementation details
  - performance
  - seed file evaluation
  - fuzzing techniques
- There's no best fuzzer
  - no quantitative comparison
  - discussion of different approaches
  - they all have found real world bugs

Performance - AFL

- **American Fuzzy Lop**
  - all about performance
  - compile time instrumentation
  - introduced fork server
    - avoid time consuming process initialization
  - prefers small files
  - memory and time restrictions

Performance - Choronzon, honggfuzz

- **Honggfuzz**
  - uses BTS for instrumentation
    - although it's a hardware feature, decoding takes too much time
  - No optimizations for performance
- **Choronzon**
  - Python & PIN give high overhead
  - No optimizations for performance

Seed file evaluation - hongfuzz

- **Honggfuzz**
  - each thread of honggfuzz grabs a file from the initial corpus.
  - is its coverage better ?
    - no, drop it
    - yes, keep it as the best

Seed file evaluation - AFL

- **American Fuzzy Lop**
  - maintains a global map of bbl transitions seen so far
  - discovered a new transition ?
    - yes, add it to the queue
    - no, discard it
  - exec_time * file_size
  - how many times each edge was visited

Seed file evaluation - Choronzon

- **Choronzon**
  - keeps a global map of each basic block for every instrumented image
  - hit new basic block ?
    - yes, keep it
    - no, find better by checking fitness & hit count of bbl

Fuzzing techniques - Honggfuzz

- **Honggfuzz**
  - various mutators
    - random byte mutation
    - move blobs of data to a different position in file
    - random file truncation
    - etc.
  - use seeds generated by another tool

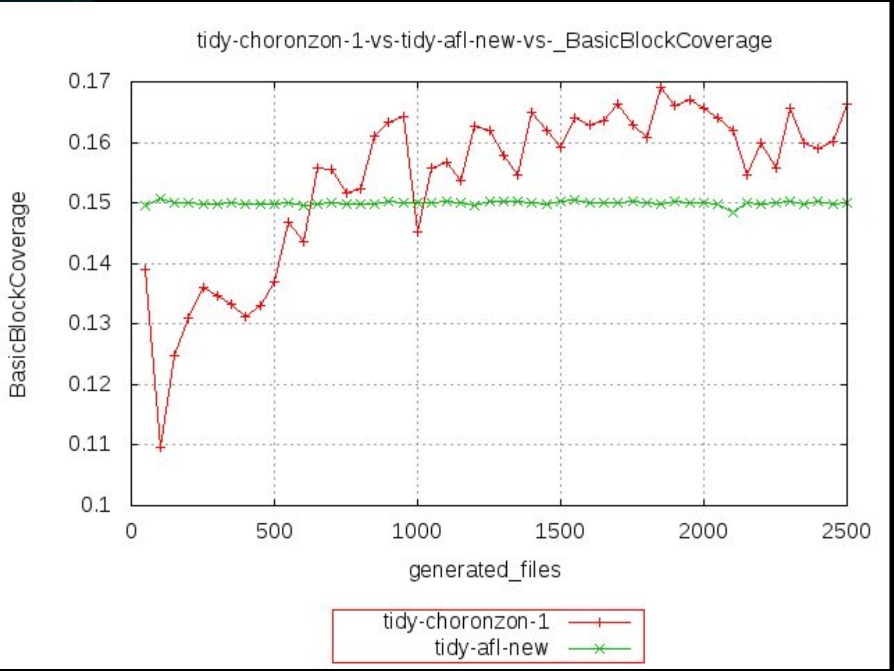
Fuzzing techniques - AFL

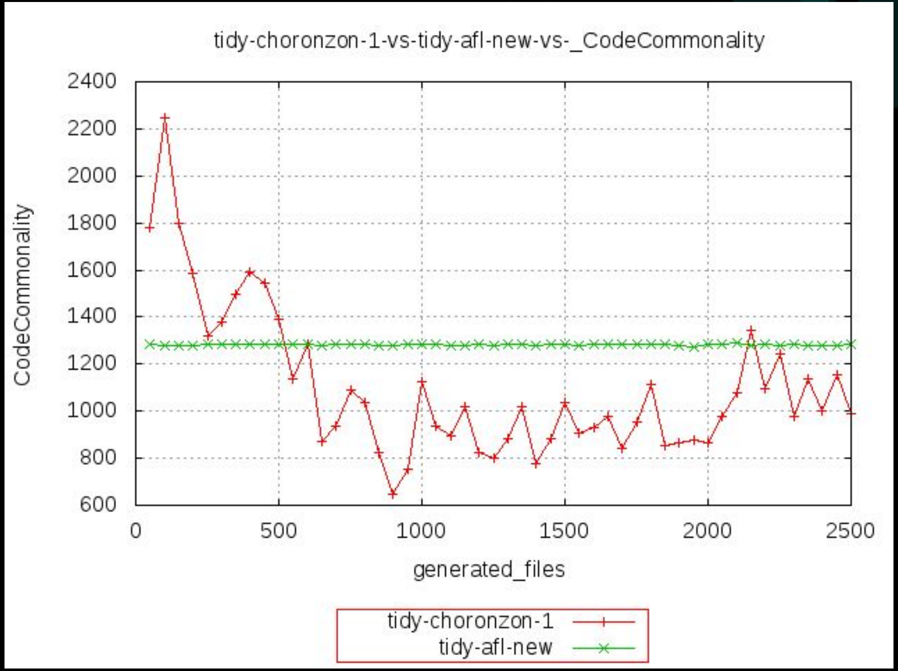
- **American Fuzzy Lop**
  - deterministic selection of mutators
  - input file trimming
  - read custom dictionary of fuzzing vectors
  - file splicing
  - custom post handler

Fuzzing techniques - Choronzon

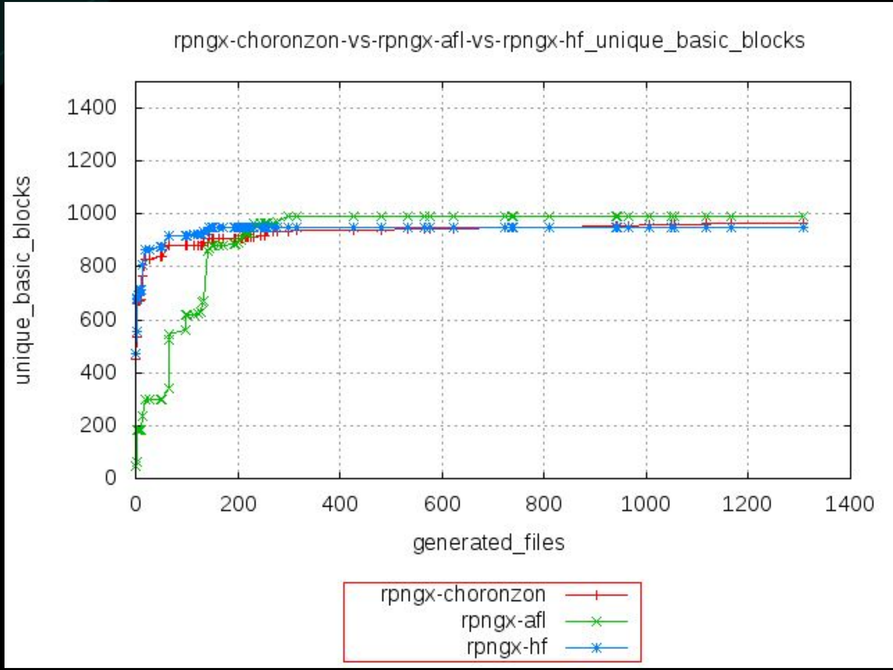
- **Choronzon**
  - wide range of mutators and recombinators
  - file format knowledge
    - efficient fuzzing of complex file formats
    - focus on interesting parts of a format
  - dynamic evaluation of fuzzers
    - fuzzers adapt to the target file format
- aims to generate mostly sane test files

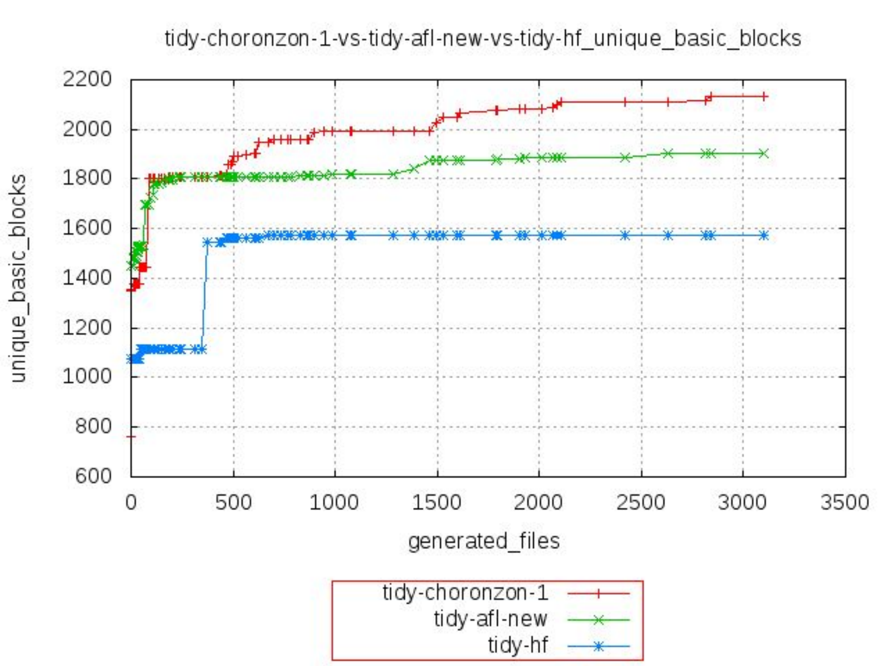
XML - fuzzer evaluation kicks in





Unique basic block sample





State-of-the-art fuzzers

	FOE	Peach	honggfuzz	AFL	Choronzon
Windows Support	✔	✔			✔
Feedback driven			✔	✔	✔
Source Code Agnostic	✔	✔	✔		✔
Inter-file mutation				✔	✔
Aware of file format		✔			✔

Conclusion

Lessons learned

- Performance is very important!
- Knowledge about the file format makes a difference
  - saves time by ignoring non-relevant parts
  - makes fuzzing more flexible
- Dynamic evaluation of fuzzers reduces the number of less interesting test files
- Recombination seems to reach parts of executables, traditional mutators don’t

Future Plans

- Increase Choronzon’s performance
  - test multiple instrumentation backends
- Implement support for index-based file formats
- Release the source code

Questions