BINFUZZ.JS

A Binary Fuzzer in JavaScript

http://dinaburg.org/binfuzz

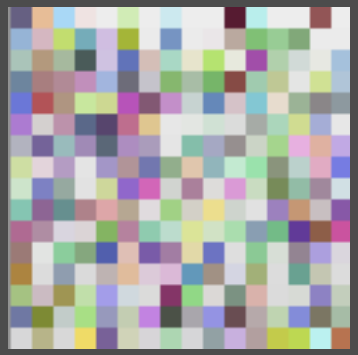
Artem Dinaburg

artem@dinaburg.org

Blackhat Arsenal 2013

Binfuzz.js

- A Binary Data Fuzzer



- Written in JavaScript

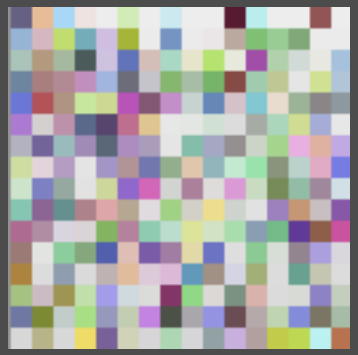
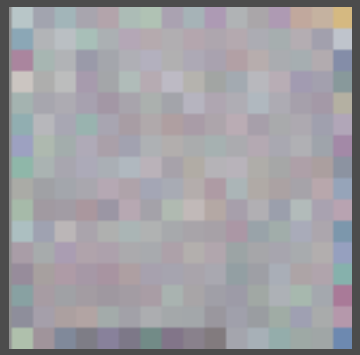
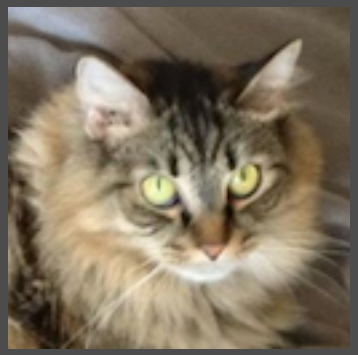
Binary Data

- A sequence of bytes not intended to be interpreted as text
- Can be easily defined as a C structure

```
struct PDU {
    uint32_t      magic;
    uint8_t       type;
    uint16_t      size;
    char          data[1];
};
```

Fuzzer

- Randomized Input Testing
- Impossible to test every program input
- Assumption: testing a random subset of every possible input will find bugs


JavaScript

- Dynamically typed object-based client-side scripting language
- Power the client side web
- Atwood's Law: any application that can be written in JavaScript, will eventually be written in JavaScript.

Binfuzz.js

- Randomized Input Testing
  - … with some brains
- In JavaScript
  - … because it can be
- Makes icons
  - … I needed a good demo!

Binfuzz.js: Challenges

- Too many combinations!
  - 4 Bytes => 4294967296 combinations
  - 8 Bytes => $1.84467440737096 * 10^{19}$ combinations
- A typical icon is ~4096 bytes.
- We have to be smarter

Need Brains

“more brains please” © flickr user abbamouse

Binfuzz.js: Challenges

- Use semantic knowledge of data to reduce combinations.

```
enum {
  PDU_DATA      = 0,
  PDU_CONTROL   = 1
} PDUTYPE;

struct PDU {
  uint32_t magic;
  uint32_t type;
  uint32_t length;
};
```

PDU is 12 bytes, but generates only **two** valid combinations

Binfuzz.js: Challenges

- Not all fuzz changes useful!
  - Checksums
  - Magic Numbers
  - Enumerations
- Use semantic knowledge to create ‘correct-enough’ files.

Binfuzz.js: Features

- Fuzz by defining semantic relationships

```
struct PDU {
  uint32_t magic;
  uint8_t  type;
  uint16_t size;
  char     data[1];
};
```

```
var PDU = new Container({'name': 'PDU'});
PDU.addChild( new UInt32({
   'root': PDU, 'name': 'magic' }));
PDU.addChild( new UInt8({
   'root': PDU, 'name': 'type' }));
PDU.addChild( new IntSize({
   'root': PDU, 'name': 'size',
   'bytesize': 2, 'target': 'PDU.data' }));
PDU.addChild(new Blob({
   'root': PDU, 'name': 'data',
   'generator': makeRandomString }));
```

Binfuzz.js: Features

- Nested Structures

```
struct inner {
  uint16_t foo;
  char bar[12]
};

struct outer {
  uint32_t magic;
  struct inner in;
};
```

```
var outer = new Container(
  {'name': 'outer'});
var inner = new Container(
  {'root': outer, 'name': 'in'});
inner.addChild( new UInt16(
  {'root': outer, 'name': 'foo'} ));
inner.addChild( new Blob(
  {'root': outer, 'name': 'bar',
   'length': 12}));
outer.addChild( new UInt32 (
  {'root': outer, 'name': 'magic',
   'constant': 0xDEADBEEF} ));
outer.addChild( inner );
```

Binfuzz.js: Features

- Length Counted Fields

```
struct big_blob {
  uint32_t bsize;
  char b[1];
};
// bsize is the
// total size of
// big_blob
```

```
var big_blob = new Container(
  {'name':'bigblob'});
big_blob.addChild( new IntSize({
  'root': big_blob, 'name': 'bsize',
  'bytesize': 4,
  'target': big_blob}));
big_blob.addChild( new Blob({
  'root': big_blob, 'name': 'b',
  'generator': makeRandomString,
  'length':
  function (seed, parent, root)
    {return
getRandomNumber(Math.pow(2,32))}}));
```

Binfuzz.js: Features

- Counted Arrays

```
struct counted {
  uint16_t num_foo;
  uint16_t foo[1];
};
// num_foo counts
// how many
// elements of
// ‘foo’ will
// actually exist
```

```
var counted = new Container({'name':
'counted'});
counted.addChild( new UInt16(
  {'root': counted,
   'name': 'num_foo'}));
var foo_array = new ArrayContainer(
  {'root': counted,
   'name': 'foo',
   'count': 'counted.num_foos'});
foo_array.addChild(new UInt16({
   'root': counted,
   'name': 'dummy'}));
counted.addChild(foo_array);
```

Binfuzz.js: Features

- Combinatorics
  - Determine number of tests a-priori
  - Select tests at random

```
var cbtest = doDemo4(0);
Log('Demo4: Combinatorics');
Log('Demo4 Combos: ' + cbtest.Combos());
Log("Demo4 Size of Combo 0" + ": " + cbtest.Size());
cbtest = doDemo4(6);
Log("Demo4 Size of Combo 6" + ": " + cbtest.Size());
```

```
Demo4: Combinatorics
Demo4 Combos: 64
Demo4 Size of Combo 0: 2
Demo4 Size of Combo 8: 131072
```

Binfuzz.js: Features

- File Offsets
  - Calculates offsets for this specific fuzz case

```
... Add counted array ...
offsets.addChild(new IntOffset({
  'root': offsets,
  'name': 'f_offset',
  'bytesize': 4,
  'target': 'offsets.f_offset'}); // self referencing!
```

```
Demo5: File Offsets
Demo5 Offset for Combo 1: 4
Demo5 Offset for Combo 4: 65536
```

Binfuzz.js: Features

- Custom Population Functions
  - Can reference other dynamic elements

```
new Blob( {'root': ICON, 'name': 'icXOR',
'generator': makeRandomString,
'length': function (seed, parent, root) {
  var bih = parent.getChild('tagBITMAPINFOHEADER');
  var width = bih.getChild('biWidth').NativeValue();
  var height = bih.getChild('biHeight').NativeValue()/2;
  var bpp = bih.getChild('biBitCount').NativeValue();
  var planes = bih.getChild('biPlanes').NativeValue();
  var stride = (( width * planes * bpp + 31) & ~31) / 8;
  var thislen = stride * height;
  return thislen; }}));
```

Binfuzz.js: Features

- Open Source
- MIT Licensed
- http://github.com/artemdinaburg/binfuzz

The Demo

- http://dinaburg.org/binfuzz
- Fuzz Windows Icons
  - Surprisingly complex!
  - Arrays
  - Nested Structures
  - Length Counters
  - Fields calculated from multiple dependencies
  - File offsets

Future Work

- Common Format Libraries
- Permutation of Blobs with generic fuzzers
- Memory Usage

Get The Code

- http://dinaburg.org/binfuzz
- http://github.com/artemdinaburg/binfuzz