Shawn Rasheed
`https://conf.researchr.org/profile/conf/shawnrasheed`
Lecturer in Computer Science and Software Engineering
Massey University
Program analysis, security

Current projects:

- Fuzzing Java EE applications
- NSC project on test flakiness

Static and Hybrid Program Analyses for Detecting Parser Vulnerabilities.

State of security

Computer security incidents common and costly.

- 7,809 incidents in 2020, a 65% increase from 2019.
- Costs banks an average NZD 104m annually.
- Ransomware attacks (e.g. Waikato DHB).

Response : early detection of vulnerabilities, policies/processes to prevent attacks.

Security threats 1/2

Denial-of-service attacks

Vulnerabilities:

- Algorithmic-complexity vulnerabilities for DoS
  Small inputs → worst-case space/time behaviour
- ReDos, HashDoS
  `/(a+)+b/.test('aaaa')`
- Crafted object graphs

Exploitation:

- Resource exhaustion →
  denial-of-service or degradation-of-service

Security threats 2/2

Code injection attacks

Vulnerabilities:

- Processing invalid/unsanitised user input
- Low-level attacks

Exploitation:

- Information leaks
- Attackers execute malicious code

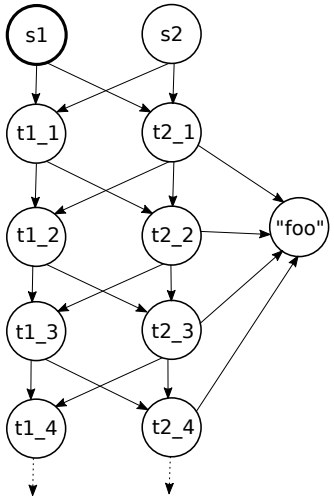
Billion Laughs

```
<?xml version="1.0"?>

<!DOCTYPE doc [
 <!ENTITY l "lol">
 <!ELEMENT doc (#PCDATA)>
 <!ENTITY l1 "&l;&l;&l;&l;&l;&l;&l;&l;&l;&l;">
 <!ENTITY l2 "&l1;&l1;&l1;&l1;&l1;&l1;&l1;&l1;&l1;&l1;">
 -- ENTITY l3...l8
 <!ENTITY l9 "&l8;&l8;&l8;&l8;&l8;&l8;&l8;&l8;&l8;&l8;">

]>

<doc>
&l9;
</doc>
```

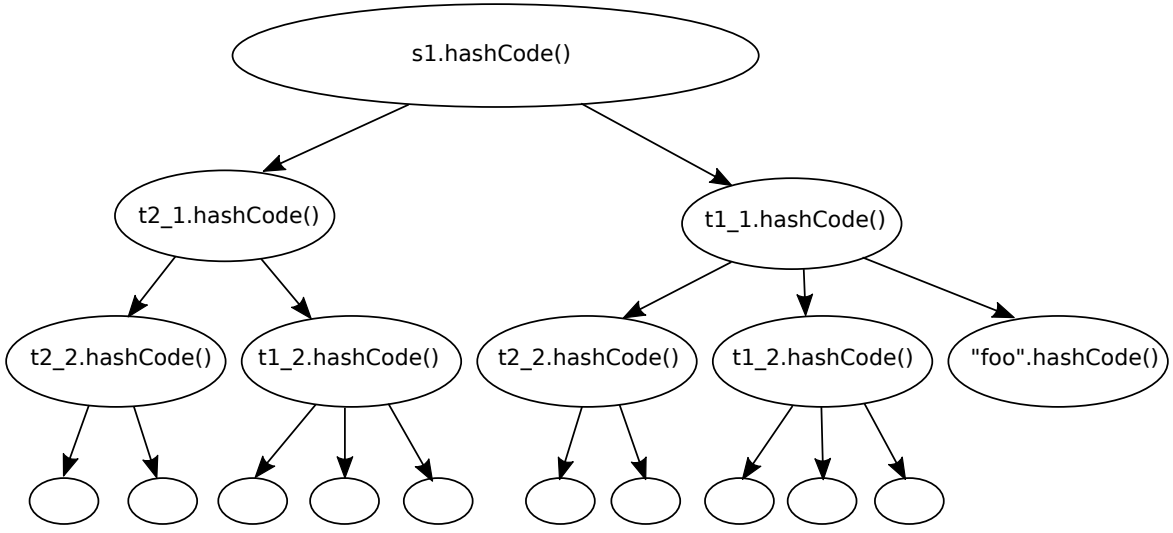
SerialDoS

```
import java.util.HashSet;
...
Set root = new HashSet();
Set s1 = root;
Set s2 = new HashSet();
for (int i = 0; i < depthN; i++) {
    Set child1 = new HashSet();
    Set child2 = new HashSet();
    child1.add("foo");
    s1.add(child1); s1.add(child2);
    s2.add(child1); s2.add(child2);
    s1 = child1; s2 = child2;
}
root.hashCode();
```

Internals of SerialDoS



Object graph



Call tree

Properties of vulnerability

- Types that allow many-to-many references
- Child-recursive methods
- Resource-monotonic methods
- Attack vectors (e.g. deserialisation or file format parsers)

Approach

Detect TTT patterns using static analysis.

Construct payload to verify true positive

Use payload for non-Java parser library

Topology 1/4

Composite

Design pattern for tree-like data structures

Models 1:M relationships between container and component objects

Recursive: containers are also components

Variants

- Relationship modelling:
  - parent $\rightarrow$ children
  - children $\rightarrow$ parent
  - parent $\longleftrightarrow$ children
- Nominal composite and leaf types, or structural types

Topology 2/4

- Generic collection types: `List<T>`, `Set<T>`, `Map<K,V>`, ...
- Arrays: `T[]`
- Specialised containers:
  `org.w3c.dom.Nodelist`, `java.beans.PropertyChangeSupport`
- Tuples: `org.apache.commons.lang3.tuple.Pair<L,R>`

Topology 3/4

Aliasing often allows using data structures intended for 1:M to model M:M

e.g. add same object to different collections

Topology 4/4

Invariants can prevent M:M structures

```
// The components in this container.

private java.util.List<Component> component = ... ;

protected void addImpl(Component comp, ...) {
    // Reparent the component and tidy up the tree's state
    if (comp.parent != null) {
        comp.parent.remove(comp);
        ...
```

Traversal

Direct

- parent.foo() has call site this.child.foo()
- indirect references to children
- child reference in parameter

Indirect

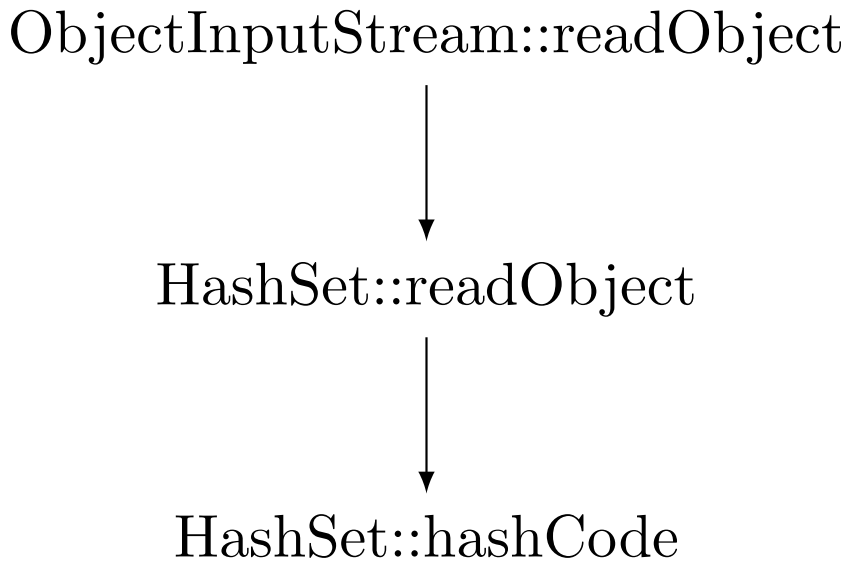
- visitors: mutual recursion between accept and visit

Modelling:

- Callgraph: edges to self
- Points-to graph: this/parameter at callsite points to child object

Trigger

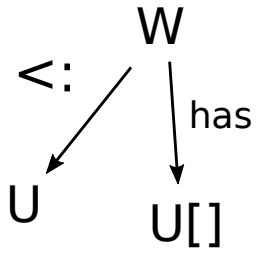
Path in callgraph to activate recursion over composite from entry method



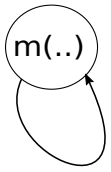
Implementation 1/2

```
class U {
 Object f;
}
class W extends U {
 U[] children;
 void m() {
   U x, y, z;
   y = new U();
   x = new U();
   y.f = x;
   z = y.f;
   m();
 }
}
```

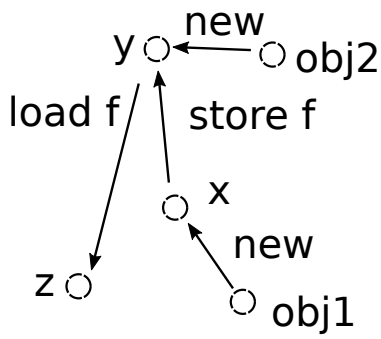
Type graph



Call graph



Points-to



Implementation 2/2

Represent TTT patterns in datalog using custom predicates

- Topology: types with many-to-many pattern
- Traversal: recursive callsites with flow constraints
- Trigger: Reachable method in call graph

Semantics defined by rules or facts

ASM-based fact extraction

Can be added to analyses with x-context-sensitivity

Dataset

- Serialisation languages (JSON, YAML) and file formats (e.g. SVG, PDF)
- Widely used parser libraries
- Some libraries require driver to invoke parsing

PDF results

Container:	`CosDictionary`
Component:	`CosBase`
Traversal:	`COSParser::checkPagesDictionary`
Trigger:	`PDFBox::main`

PDF Vulnerabilities 2/3

PDF can be constructed from detected pattern to verify vulnerability for:

- PDFBox
- PDFxStream
- Ghostscript
- PDFtk

PDF Vulnerabilities 3/3

`checkPagesDictionary` does terminate at lower depths

```
PDDocument doc = PDDocument.load(new File("foo.pdf"));
System.out.println(doc.getNumberOfPages());
// outputs   134,217,728
```

can exhaust disk space with `PDFToImage`

YAML Vulnerabilities 1/3

Container:	`MappingNode`
Component:	`NodeTuple`
Traversal:	`mergeNode`
Trigger:	`org.yaml.snakeyaml.Yaml::load`

YAML Vulnerabilities 2/3

```
? - &t2a
    - &t3a [lol]
    - &t3b [lol]
  - &t2b
    - *t3a
    - *t3b
: value
--
{ << { << { key: value} } }
```

YAML Vulnerabilities 3/3

- Manually constructed several payloads to investigate YAML parser libraries
- Recursive structure in list, and map (as key and value)
- Studied 14 libraries (Java, JS, Python, PHP, Perl, Ruby, Rust, Swift, C# and Dart)
- Found seven bugs / vulnerabilities
  - js-yaml
    https://www.npmjs.com/advisories/788
  - PHP Yaml-Extensions
    https://bugs.php.net/bug.php?id=77720

SVG Vulnerabilities 1/3

Container:	`batik.anim.dom.Node`
Component:	`batik.anim.dom.Node`
Traversal:	`SVGOMElement::getCascadedXMLBase`
Trigger:	`batik.apps.rasterizer.Main::main`

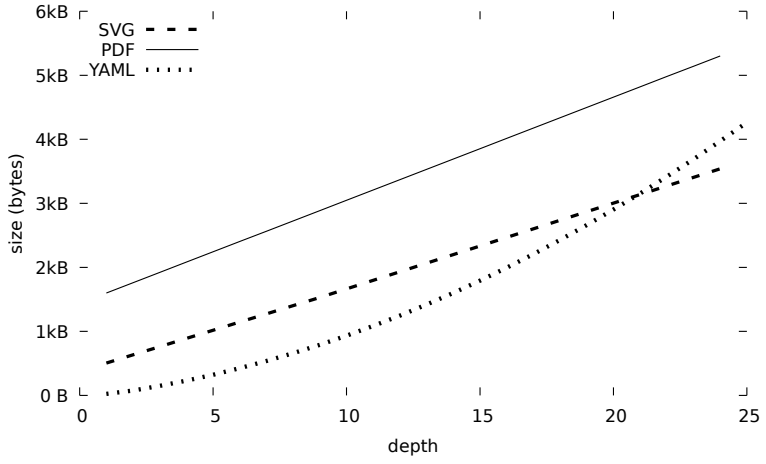
SVG Vulnerabilities 2/3

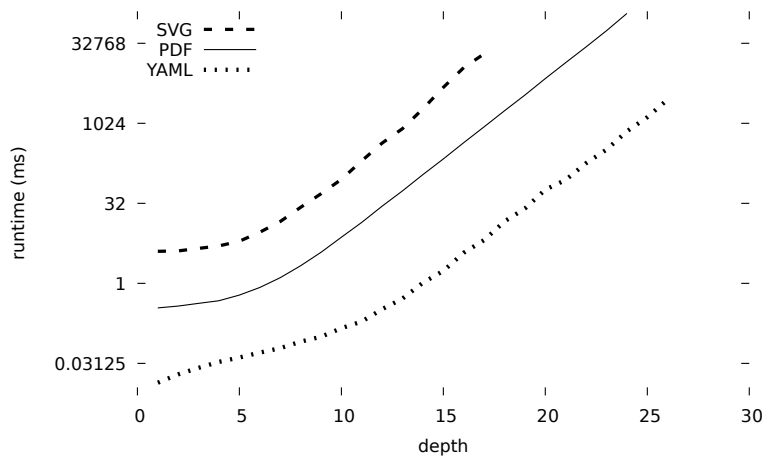
```
<g id="t0a">
<use xlink:href="#t1a"/>
<use xlink:href="#t1b"/>
</g>

<g id="t0b">
<use xlink:href="#t1a"/>
<use xlink:href="#t1b"/>
</g>
```

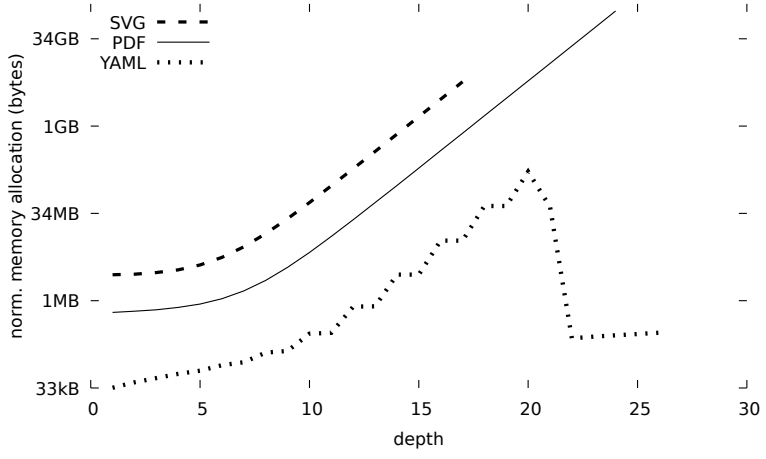
SVG Vulnerabilities 3/3

- Browsers: firefox, chrome, edge, ..
- Libraries: librsvg, cariosvg, svg-sanitizer
- Markup: ODT, GitHub, GitLab, bitbucket, Stack Overflow
- Illustration packages: Inkscape, Adobe Illustrator

Benchmark


SVG
PDF
YAML
size (bytes)
6kB
5kB
4kB
3kB
2kB
1kB
0 B
0
5
10
15
20
25
depth


SVG
PDF
YAML
runtime (ms)
32768
1024
32
1
0.03125
0
5
10
15
20
25
30
depth


SVG
PDF
YAML
norm. memory allocation (bytes)
34GB
1GB
34MB
1MB
33kB
0
5
10
15
20
25
30
depth

Precision and Soundness

- Precision:
  - Not all patterns can be turned into actual vulnerabilities
  - Problems are invariants
  - It might be possible to convert more
  - Automation via fuzzing
  - More precise static analysis
- Recall:
  - Patterns not discovered by analysis
  - Hybrid analysis or support for dynamic language features to boost recall

Injection Vulnerabilities

Deserialising untrusted input

- Attacker controls input (serialised stream has data and types)
- Encapsulation and invariants do not hold
  Attacker crafts serialised object graph
- Control flow hijacked
- Security sensitive operations (sink)
  - Remote code execution
  - Loading arbitrary class

Approach

Static pre-analysis to improve efficiency of dynamic analysis (fuzzing).

- Pre-analysis
  - Open program points-to analysis
    Models heap allocations for serialised objects
  - Heap access path
    Sequence of field dereferences from root of object graph to sink
- Dynamic analysis
  - Mutational fuzzer
  - Generative fuzzer
  - Use classes on heap access path to direct fuzzing

Results

Library	Time (s)	Sinks	Paths	Objects
bsh-2.0b5	655	1	1	0
clojure-1.8.0	timeout	n/a	n/a	n/a
commons-beanutils-1.9.2	786	0	0	0
commons-collections-3.1	1611	1	1	1
commons-collections4-4.0	681	1	1	1
groovy-2.3.9	5204	3	3	0
hibernate	3397	2	3	0
jython-standalone-2.5.2	timeout	n/a	n/a	n/a
rome-1.0	390	1	0	0

- Dataset:
  ysoserial: public repository of deserialisation payloads
- Results:
  Two vulnerabilities confirmed
  Static pre-analysis improves fuzzing

Future work

- Straight-forward static analysis – fast and easy to implement
- Precision and recall issues but sufficient to find multiple issues
- Impact beyond Java: facilitates payload construction
- Manual last mile
- Future work:
  - More formats (office, etc)
  - Use as static pre-analysis for fuzzing campaign (SlowFuzz, Badger, Perrfuzz, ...)
  - Improve taint modelling for deserialisation