ODDFuzz: Hunting Java Deserialization Gadget Chains via Structure-Aware Directed Greybox Fuzzing

Speakers: Biao He, Haowen Mu

Contributors: Sicong Cao, Xiaobing Sun, Yu Ouyang, Chao Zhang, Xiaoxue Wu, Ting Su, Lili Bo, Bin Li, Chuanlei Ma, Jiajia Li, Tao Wei

YANGZHOU UNIVERSITY
ANT GROUP
Tsinghua University
EAST CHINA NORMAL UNIVERSITY

About Speakers

Biao He

- Security researcher @ Ant Security FG Lab
- Black Hat Europe 2022 Speaker
- @codeplutos

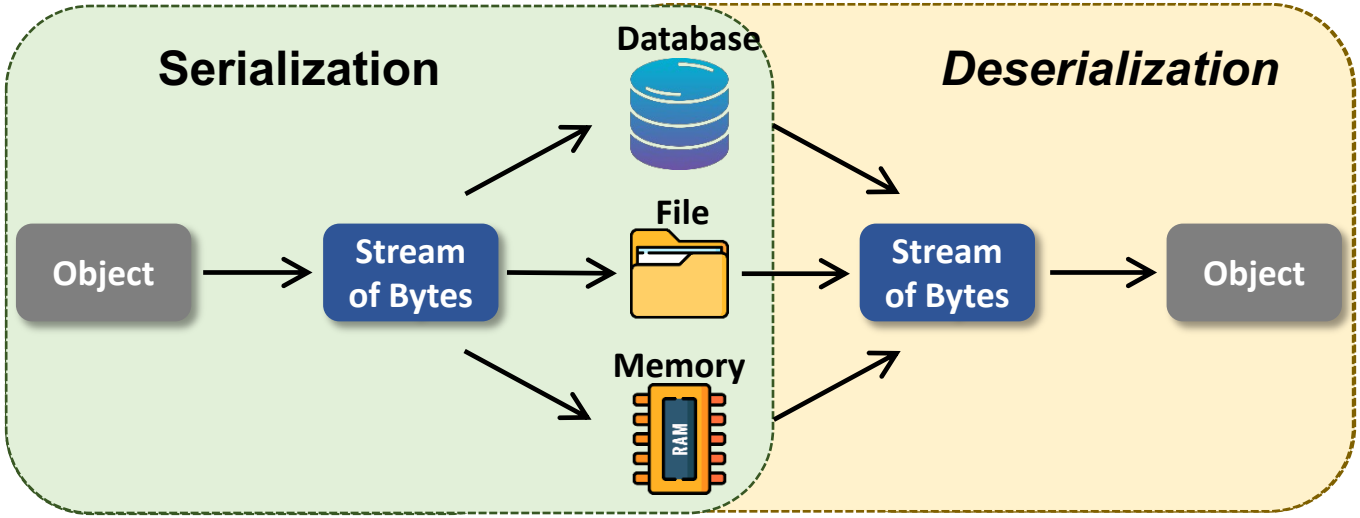
Haowen Mu

- Founder @ Cyberutopian
- CTF player at Nu1L Team
- @meizjm3i

Agenda

- Introduction
- Previous work & Remaining challenges
- ODDFuzz: A novel approach to hunting gadget chains
- Evaluation
- Conclusion & Takeaways

Java Deserialization



- Communication
- Caching
- Deep Copy
- Cross JVM Synchronization
- Persistence

Java Deserialization Vulnerability

Magic methods will be executed automatically when deserialization

```
ObjectInputStream ois = new ObjectInputStream(source.getInputStream());
User user =  (User) ois.readObject();
```

```
public class Evil{
    public void readObject(ObjectInputStream ois){
        Runtime.getRuntime().exec(ois.readUTF());
    }
}
```

- readObject
- readResolve
- ……

Java Deserialization Gadget Chain

```java
public void readObject(ObjectInputStream ois) {
    Comparator comparator = (Comparator) ois.readObject();
    comparator.compare(ois.readObject(),ois.readObject());
}
```

```java
public class ValueComparator implements Comparator{
    @Override
    public int compare(Value v1, Value v2) {
        return v1.getValue() - v2.getValue();
    }
}
```

```java
public class InvokeComparator implements Comparator {
    String methodName;
    String className;

    @Override
    public int compare(Object o1, Object o2) {
        Method method = Class.forName(className)
                            .getMethod(methodName);
        method.setAccessible(true);
        return (int) method.invoke(o1,o2);
    }
}
```

Exploit

A call chain starts with a magic method (*source method*) and ends with a security-sensitive method (*sink method*)

Why gadget chains are so significant

For defenders

- Deserialization is unavoidable
- Blacklist

For attackers

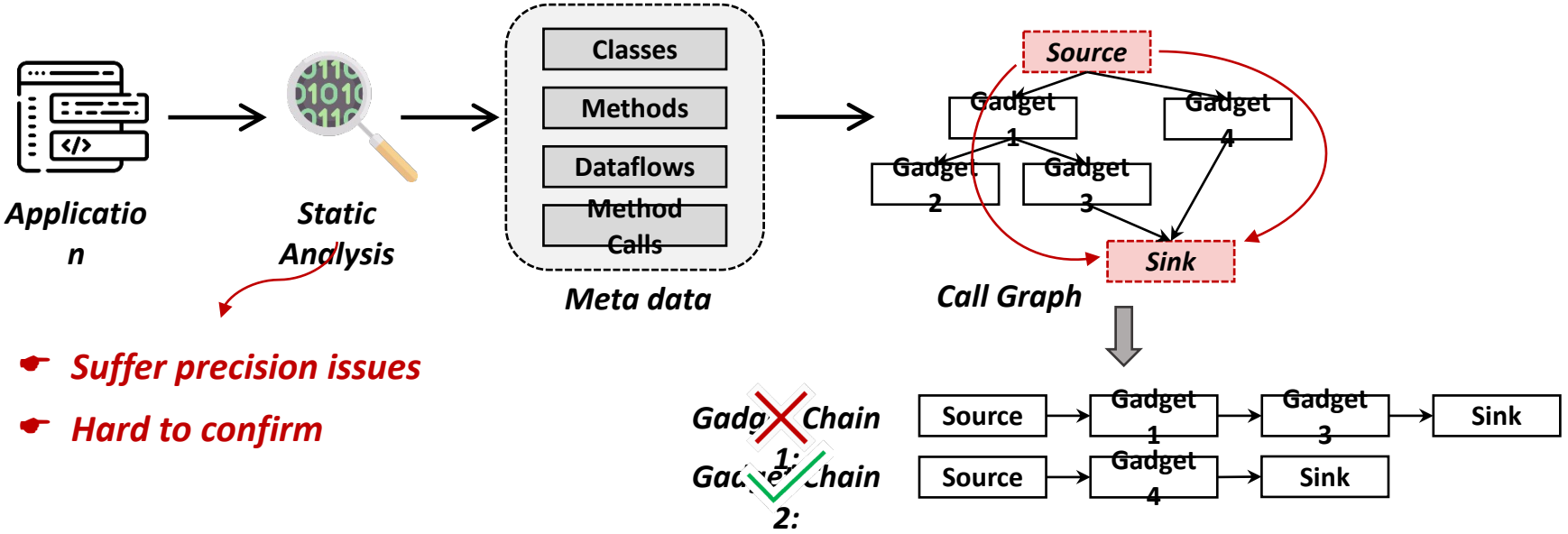
- Make Java deserialization vulnerability exploitable
- Bypass blacklist

Agenda

- Introduction
- Previous work & Remaining challenges
- ODDFuzz: A novel approach to hunting gadget chains
- Evaluation
- Conclusion & Takeaways

Existing Solutions

Gadget Inspector [BlackHat 2018]

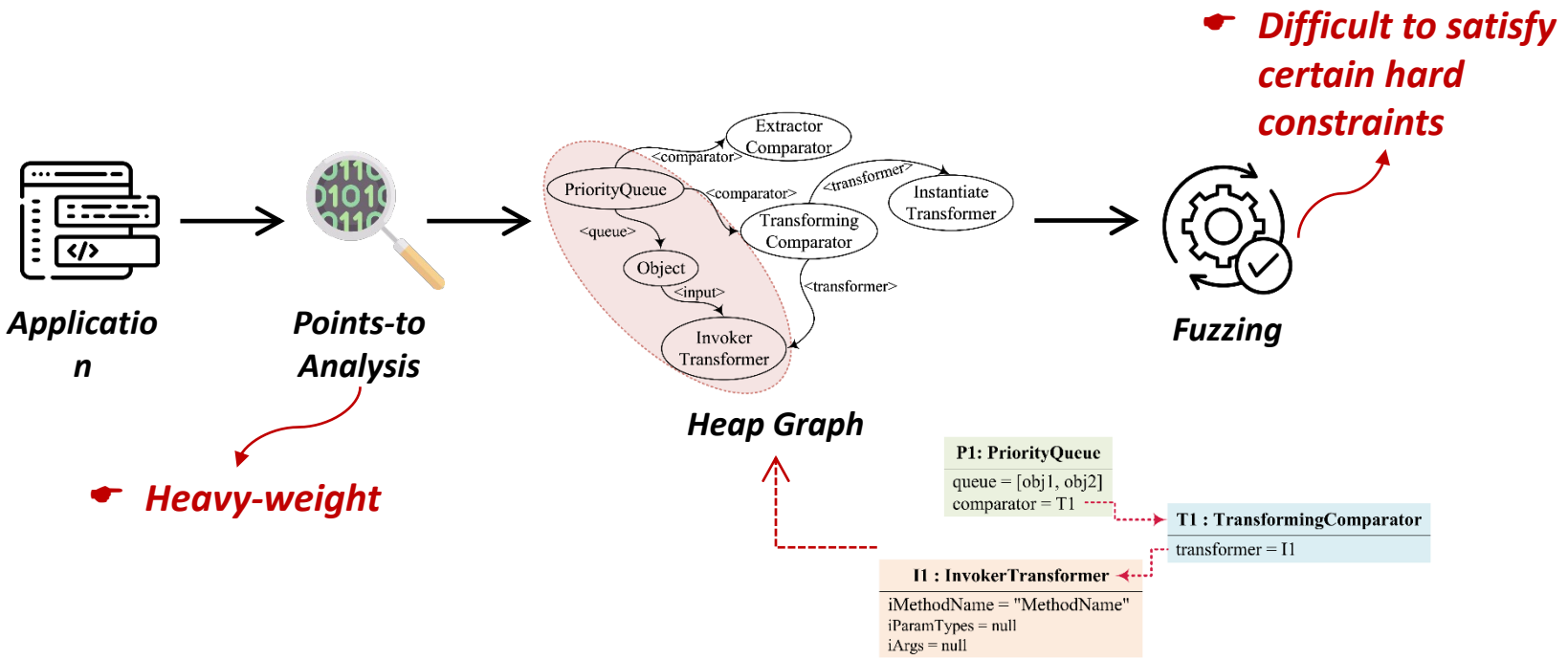


- *Suffer precision issues*
- *Hard to confirm*

[1]I. Haken, "Automated discovery of deserialization gadget chains," BlackHat USA, 2018.

Existing Solutions

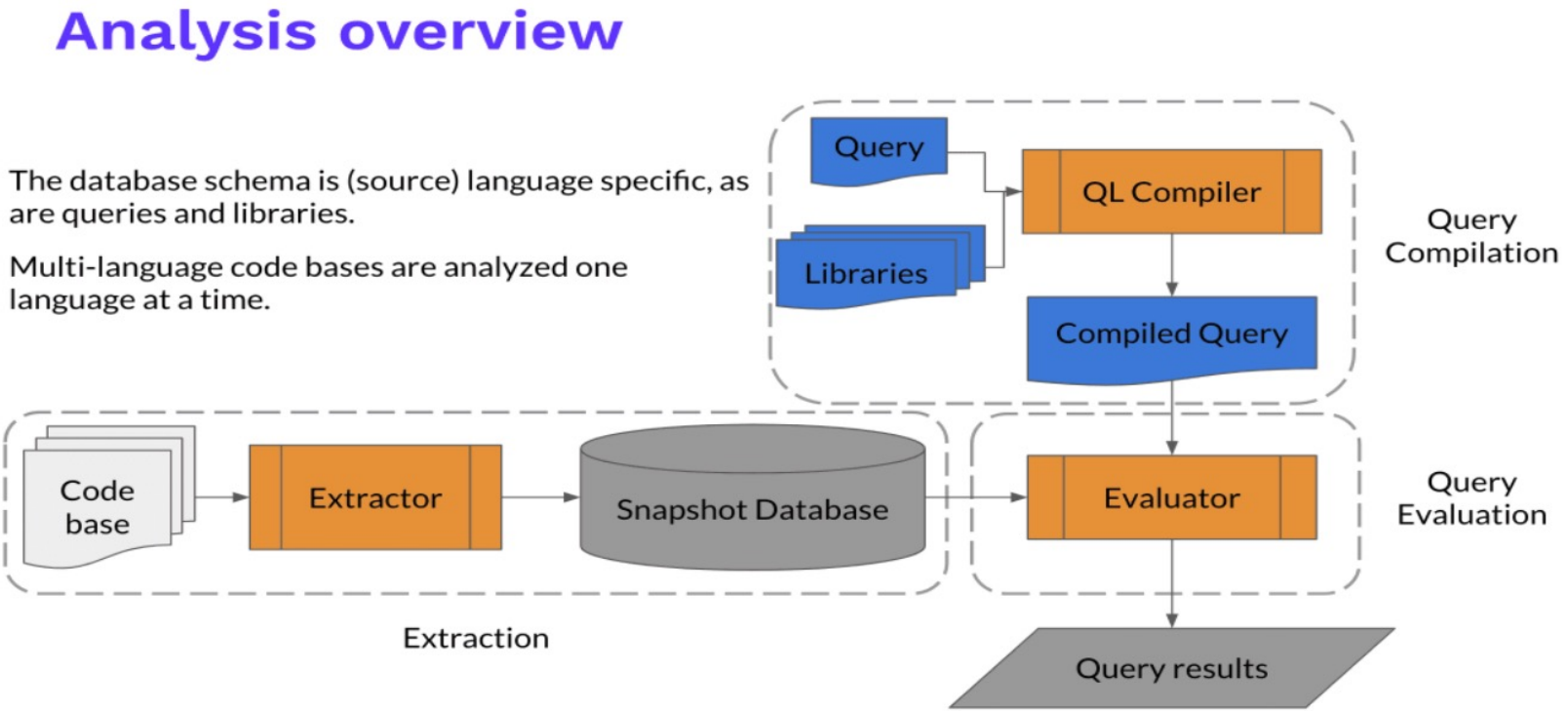
SerHybrid [ASE 2020]



[2]S. Rasheed and J. Dietrich, "A hybrid analysis to detect java serialization vulnerabilities," IEEE/ACM International Conference on Software Engineering, 2020.

Existing Solutions

CodeQL



Remaining Challenges

1. Runtime polymorphism and other dynamic language features make it difficult to make trade-offs between precision and recall.

2. Java deserialization gadget chains can be quite long, which causes huge computation space and amplifies the inaccuracy.

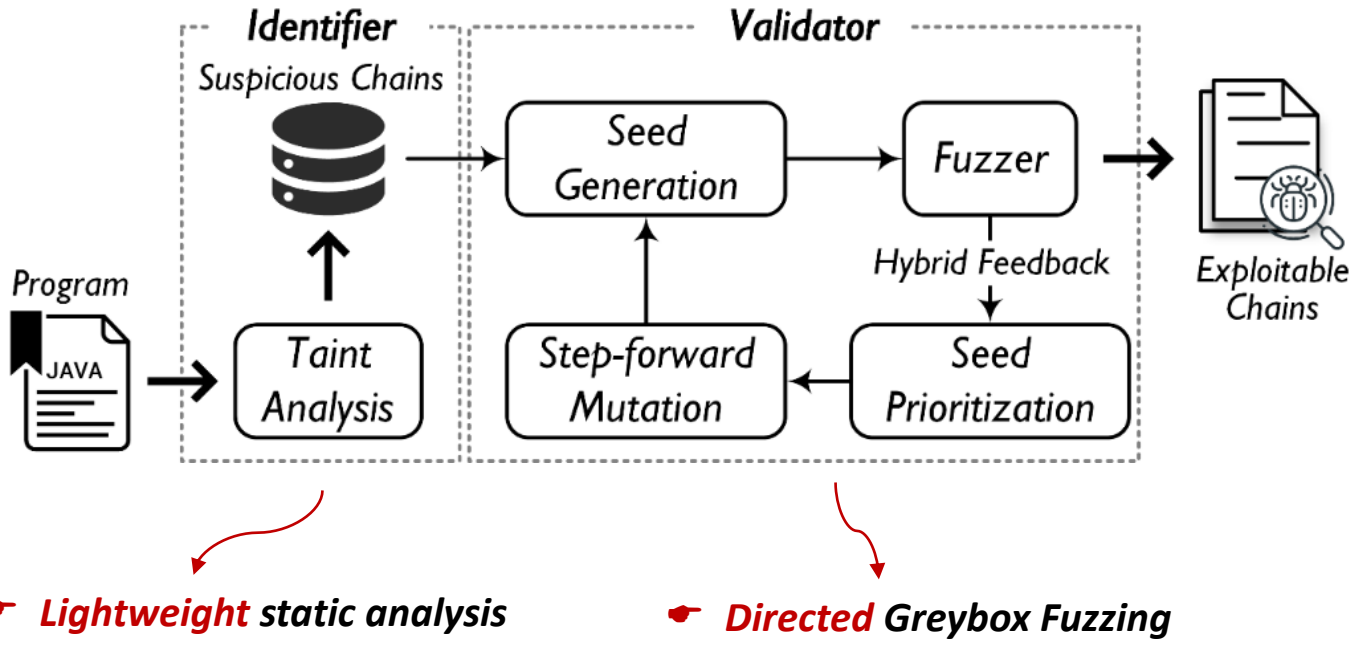
3. Existing tools are unable to validate candidate gadget chains, which require manual inspection and are time-consuming and error-prone.

Agenda

- Introduction
- Previous work & Remaining challenges
- ODDFuzz: A novel approach to hunting gadget chains
- Evaluation
- Conclusion & Takeaways

Our approach: ODDFuzz

Lightweight Taint **Analysis and** ***Directed*** **Greybox Fuzzing**



Lightweight static analysis

Class A

```java
public void readObject(ObjectInputStream ois) {
    //get comparator
    this.comparator = (Comparator) ois.readObject();
    Comparator comparator = threadLocal.get();
    if (comparator == null) {
        comparator = this.comparator;
    }
    //read values
    value1 = ois.readObject();
    value2 = ois.readObject();
    //compare and add to ordered arraylist
    if (comparator.compare(value1, value2) > 0) {
        orderedList.add(value1);
        orderedList.add(value2);
    } else {
        orderedList.add(value2);
        orderedList.add(value1);
    }
}
```

```java
public class ValueComparator implements Comparator{
    @Override
    public int compare(Value v1, Value v2) {
        return v1.getValue() - v2.getValue();
    }
}
```

```java
public class InvokeComparator implements Comparator {
    String methodName;
    String className;

    @Override
    public int compare(Object o1, Object o2) {
        Method method = Class.forName(className)
                             .getMethod(methodName);
        method.setAccessible(true);
        return (int) method.invoke(o1,o2);
    }
}
```

Exploit

Controlling Data Types
=> Controlling Code!

Lightweight static analysis

```
public void readObject(ObjectInputStream ois) {
    //get comparator
    this.comparator = (Comparator) ois.readObject();
    Comparator comparator = threadLocal.get();
    if (comparator == null) {
        comparator = this.comparator;
    }
    //read values
    value1 = ois.readObject();
    value2 = ois.readObject();
    //compare and add to ordered arraylist
    if (comparator.compare(value1, value2) > 0) {
        orderedList.add(value1);
        orderedList.add(value2);
    } else {
        orderedList.add(value2);
        orderedList.add(value1);
    }
}
```

New	`x = new T()`
Assign	`x = y`
Store	`x.f = y`
Load	`y = x.f`
Call	`r = x.k(a, …)`

Lightweight static analysis

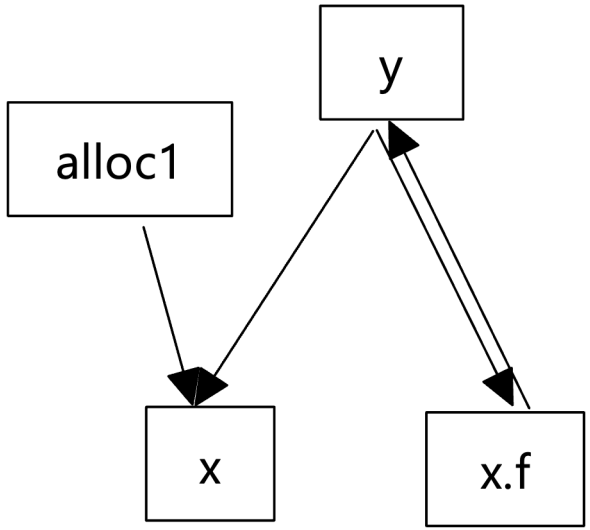
```
public void readObject(ObjectInputStream ois) {
    //get comparator
    this.comparator = (Comparator) ois.readObject();
    Comparator comparator = threadLocal.get();
    if (comparator == null) {
        comparator = this.comparator;
    }
    //read values
    value1 = ois.readObject();
    value2 = ois.readObject();
    //compare and add to ordered arraylist
    if (comparator.compare(value1, value2) > 0) {
        orderedList.add(value1);
        orderedList.add(value2);
    } else {
        orderedList.add(value2);
        orderedList.add(value1);
    }
}
```

New	`x = new T()`
Assign	`x = y`
Store	`x.f = y`
Load	`y = x.f`
Call	`r = x.k(a, …)`

SPAG (Simplified Pointer Assignment Graph) + Worklist algorithm

Lightweight static analysis

New	`x = new T()`
Assign	`x = y`
Store	`x.f = y`
Load	`y = x.f`
Call	`r = x.k(a, …)`



Construct SPAG (Simplified Pointer Assignment Graph)

Lightweight static analysis

```
public void readObject(ObjectInputStream ois) {
    //get comparator
    this.comparator = (Comparator) ois.readObject();
    Comparator comparator = threadLocal.get();
    if (comparator == null) {
        comparator = this.comparator;
    }
    //read values
    value1 = ois.readObject();
    value2 = ois.readObject();
    //compare and add to ordered arraylist
    if (comparator.compare(value1, value2) > 0) {
        orderedList.add(value1);
        orderedList.add(value2);
    } else {
        orderedList.add(value2);
        orderedList.add(value1);
    }
}
```

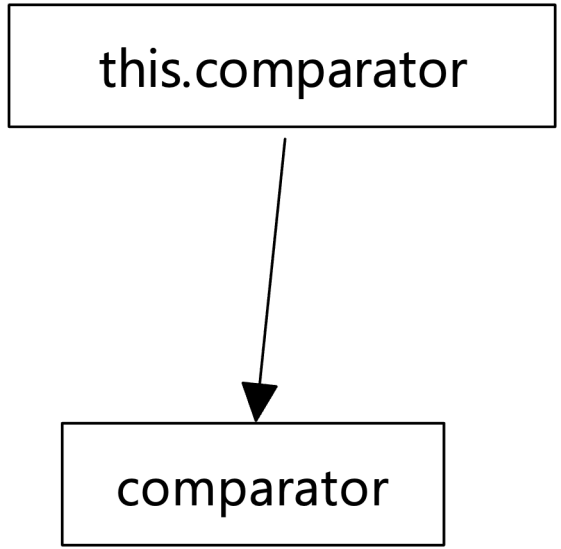
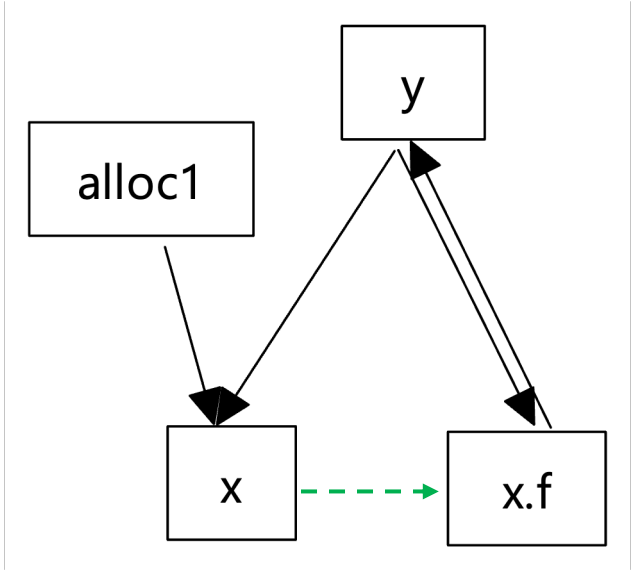


Fig. 1: SPAG

Construct SPAG (Simplified Pointer Assignment Graph)

Lightweight static analysis

New	`x = new T()`
Assign	`x = y`
Store	`x.f = y`
Load	`y = x.f`
Call	`r = x.k(a, …)`

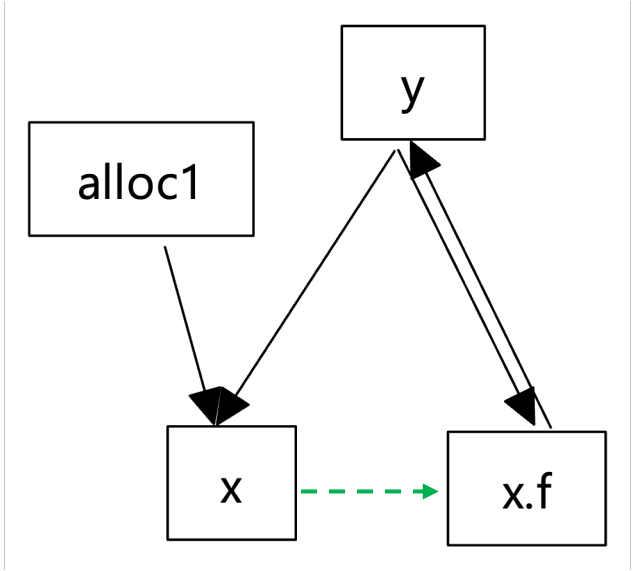


Worklist algorithm

Lightweight static analysis

New	`x = new T()`
Assign	`x = y`
Store	`x.f = y`
Load	`y = x.f`
Call	`r = x.k(a, …)`

- Static Call X.k(a, …)
- Special Call x.<init>(a, …) / super.k(a, …)
- Virtual Call x.k(a, …)



Worklist algorithm

Lightweight static analysis

New	`x = new T()`
Assign	`x = y`
Store	`x.f = y`
Load	`y = x.f`
Call	`r = x.k(a, …)`

- Static Call — X.k(a, …)
- Special Call — x.<init>(a, …) / super.k(a, …)
- Virtual Call — x.k(a, …)

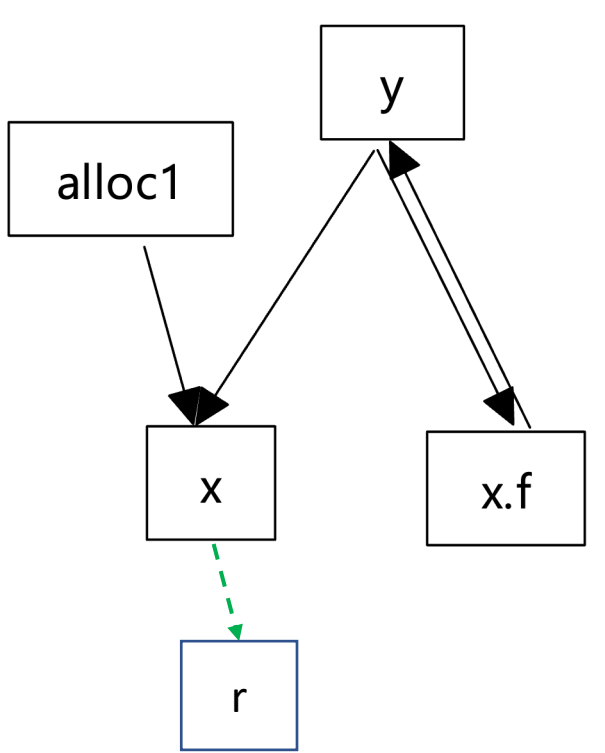
Call statement

- Construct call graph on-the-fly
- Perform Class Hierarchy Analysis (CHA) on the call statement *only* when the receive object is tainted
- Prior knowledge
  - Native method
  - Specific method (readObject)

Lightweight static analysis

New	`x = new T()`
Assign	`x = y`
Store	`x.f = y`
Load	`y = x.f`
Call	`r = x.k(a, …)`

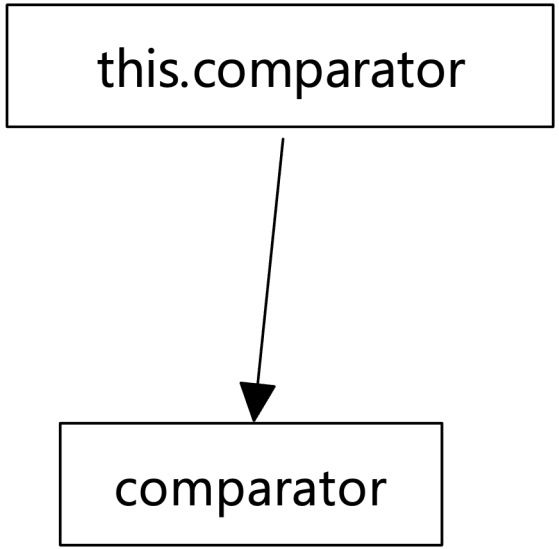
- Static Call X.k(a, …)
- Special Call x.<init>(a, …) / super.k(a, …)
- Virtual Call x.k(a, …)



Worklist algorithm

Lightweight static analysis

```java
public void readObject(ObjectInputStream ois) {
    //get comparator
    this.comparator = (Comparator) ois.readObject();
    Comparator comparator = threadLocal.get();
    if (comparator == null) {
        comparator = this.comparator;
    }
    //read values
    value1 = ois.readObject();
    value2 = ois.readObject();
    //compare and add to ordered arraylist
    if (comparator.compare(value1, value2) > 0) {
        orderedList.add(value1);
        orderedList.add(value2);
    } else {
        orderedList.add(value2);
        orderedList.add(value1);
    }
}
```



SPAG (Simplified Pointer Assignment Graph) + Worklist algorithm

Lightweight static analysis

Class A

```
public void readObject(ObjectInputStream ois) {
    //get comparator
    this.comparator = (Comparator) ois.readObject();
    Comparator comparator = threadLocal.get();
    if (comparator == null) {
        comparator = this.comparator;
    }
    //read values
    value1 = ois.readObject();
    value2 = ois.readObject();
    //compare and add to ordered arraylist
    if (comparator.compare(value1, value2) > 0) {
        orderedList.add(value1);
        orderedList.add(value2);
    } else {
        
    }
}
```

```
public class ValueComparator implements Comparator{
    @Override
    public int compare(Value v1, Value v2) {
        return v1.getValue() - v2.getValue();
    }
}
```

```
public class InvokeComparator implements Comparator {
    String methodName;
    String className;

    @Override
    public int compare(Object o1, Object o2) {
        Method method = Class.forName(className)
                            .getMethod(methodName);
        method.setAccessible(true);
        return (int) method.invoke(o1,o2);
```

Exploit

Simplified PAG (Pointer Assignment Graph) + Worklist algorithm

Lightweight static analysis - Highlight

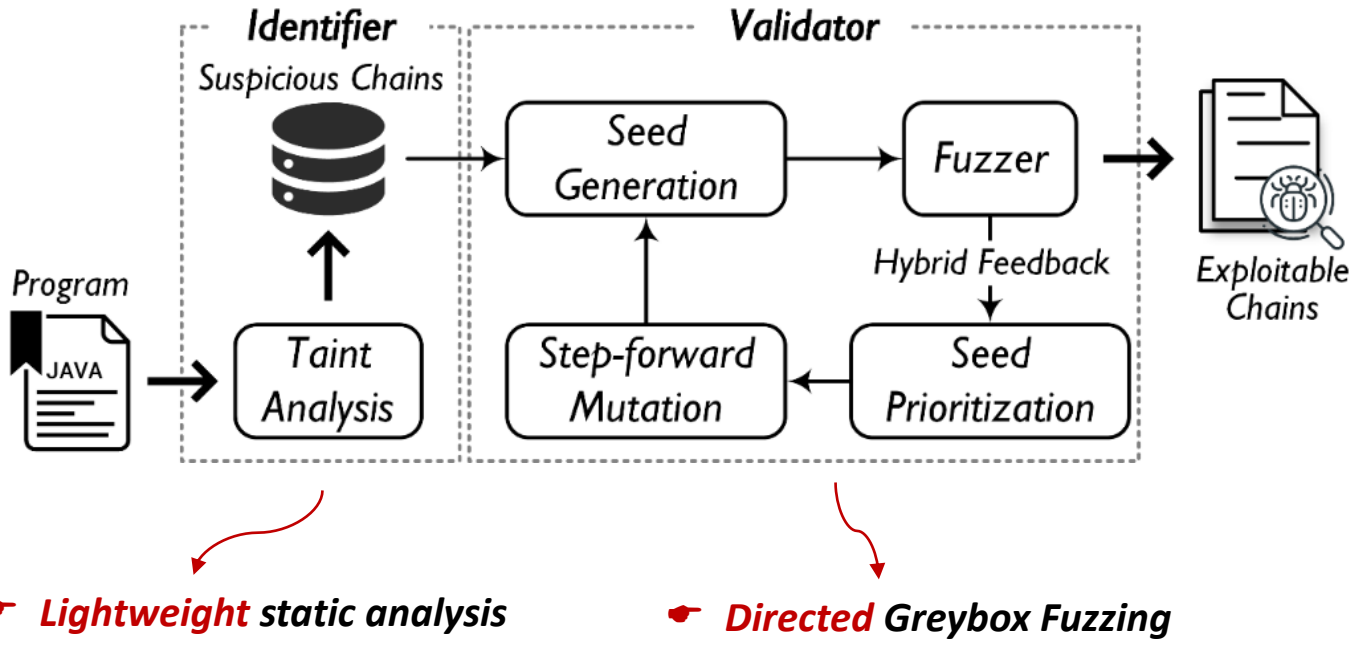
1. Goal: Locate all candidate gadget chains within limited computation time and resources

2. Solution: Andersen-style analysis.

3. Techniques: SPAG and On-demand CHA.

Directed greybox fuzzing

***Lightweight Taint* Analysis and *Directed* Greybox Fuzzing**



Directed greybox fuzzing – seed generation

How to generate seeds which satisfied serialization stream format?

- Construct object that will be serialized, rather than generate serialization stream.

```
00: ac ed 00 05 73 72 00 04 4c 69 73 74 69 c8 8a 15 >....sr..Listi...<
10: 40 16 ae 68 02 00 02 49 00 05 76 61 6c 75 65 4c >z......I..valueL<
20: 00 04 6e 65 78 74 74 00 06 4c 4c 69 73 74 3b 78 >..nextt..LList;x<
30: 70 00 00 00 11 73 71 00 7e 00 00 00 00 00 13 70 >p....sq.~......p<
40: 71 00 7e 00 03                                  >q.~..<
```

Fig. 1: The serialization stream sample[1]

[1] https://docs.oracle.com/javase/8/docs/platform/serialization/spec/protocol.html

Directed greybox fuzzing – seed generation

Class A

```
public void readObject(ObjectInputStream ois) {
    //get comparator
    this.comparator = (Comparator) ois.readObject();
    Comparator comparator = threadLocal.get();
    if (comparator == null) {
        comparator = this.comparator;
    }
    //read values
    value1 = ois.readObject();
    value2 = ois.readObject();
    //compare and add to ordered arraylist
    if (comparator.compare(value1, value2) > 0) {
        orderedList.add(value1);
        orderedList.add(value2);
    } else {
        orderedList.add(value2);
        orderedList.add(value1);
    }
}
```

Okay, but how to construct object that will be serialized?

- Use sun.misc.Unsafe to obtain an instance.
- Use reflection feature to set fields.

Directed greybox fuzzing – seed generation

```
public class A {
    Comparator comparator;
    transient ArrayList<Object> orderedList;
    Object value1;
    Object value2;

    public void readObject(ObjectInputStream ois){
        //...
    }
}
public class InvokeComparator implements Comparator {
    String methodName;
    String className;

    @Override
    public int compare(Object o1, Object o2) {
        Method method = Class.forName(className)
                        .getMethod(methodName);
        method.setAccessible(true);
        return (int) method.invoke(o1,o2);
    }
}
```

Fine, How to represent the multi-level struct of seeds?

- We adopt a hierarchical data structure called *property tree* to handle the complex forms of seeds.

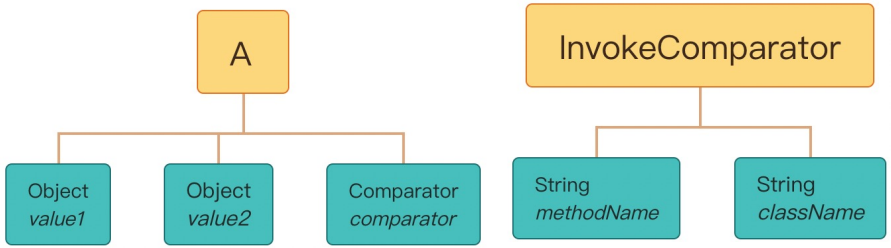


Fig. 1: property tree

Directed greybox fuzzing – seed mutation

```
A#readObject
    InvokeComparator#compare
        java.lang.reflect.Method#invoke
```

Fig. 1: candidate gadget chain

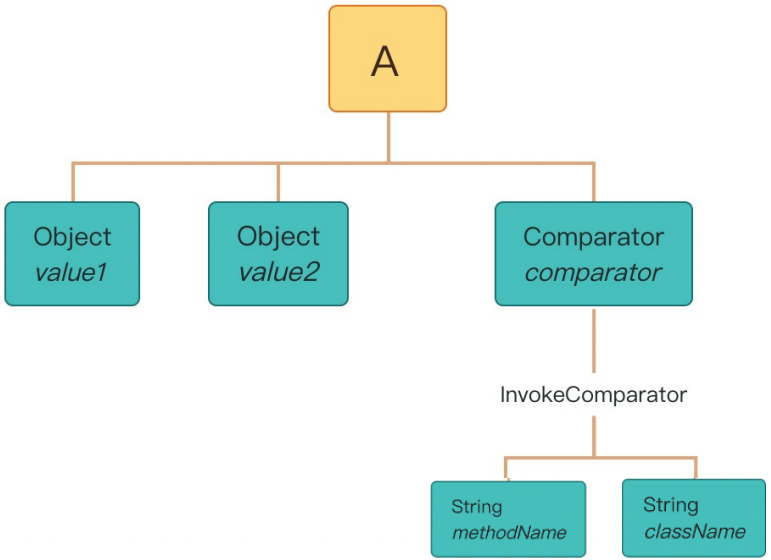


Fig. 2: property tree

Fine, what values will be assigned to the leaf nodes?

- Pre-generated values and Random generated values
- Pre-generated values are constructed according to the candidate gadget chain (Fig. 1)
- Pre-generated values have a higher priority when mutation

Directed greybox fuzzing – seed mutation

```
A#readObject
    InvokeComparator#compare
        java.lang.reflect.Method#invoke
```

Fig. 1: candidate gadget chain

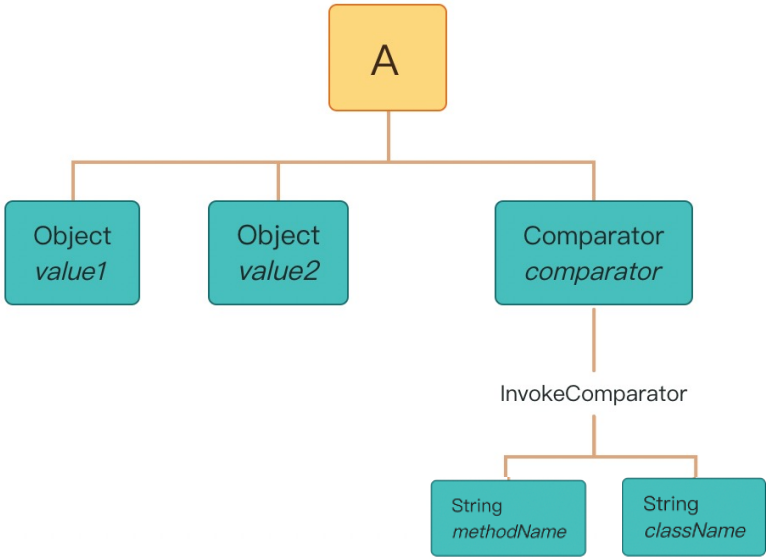
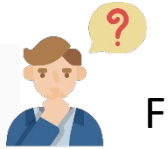


Fig. 2: property tree



Fine, what values will be assigned to the leaf nodes?

Type	Candidate values
Comparator	InvokeComparator
String	class names, method names, ‘f5a5a608’, random
Class	class involved in gadget chains, random
Boolean	true, false
Integer	-15~15, random
Object	above values

Directed greybox fuzzing – seed prioritization

```
A#readObject
    InvokeComparator#compare
        java.lang.reflect.Method#invoke
```

Fig. 1: candidate gadget chain

```
public class InvokeComparator implements Comparator {
    String methodName;
    String className;

    @Override
    public int compare(Object o1, Object o2) {
        Method method = Class.forName(className)
                        .getMethod(methodName);
        method.setAccessible(true);
        return (int) method.invoke(o1,o2);
    }
}
```

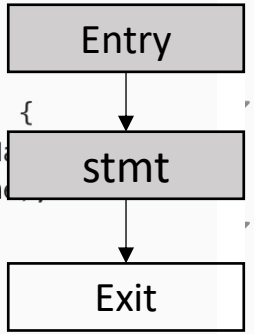


Fig. 2: CFG

Class A

```
public void readObject(ObjectInputStream ois) {
    //get comparator
    this.comparator = (Comparator) ois.readObject();
    Comparator comparator = threadLocal.get();
    if (comparator == null) {
        comparator = this.comparator;
    }
    //read values
    value1 = ois.readObject();
    value2 = ois.readObject();
    //compare and add to ordered arraylist
    if (comparator.compare(value1, value2) > 0) {
        orderedList.add(value1);
        orderedList.add(value2);
    } else {
        orderedList.add(value2);
        orderedList.add(value1);
    }
}
```

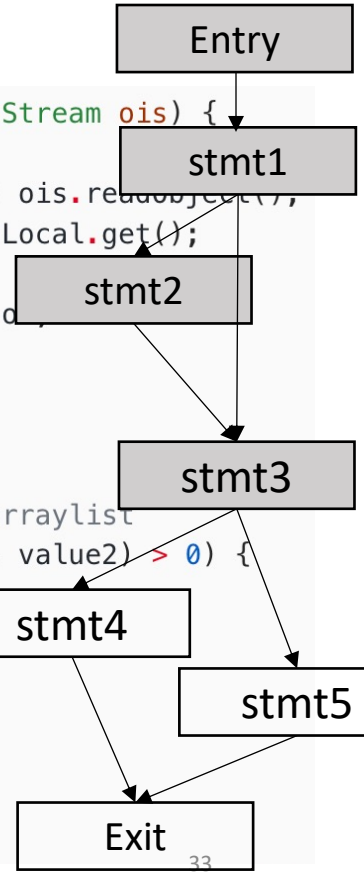


Fig. 3: CFG

Directed greybox fuzzing – seed prioritization

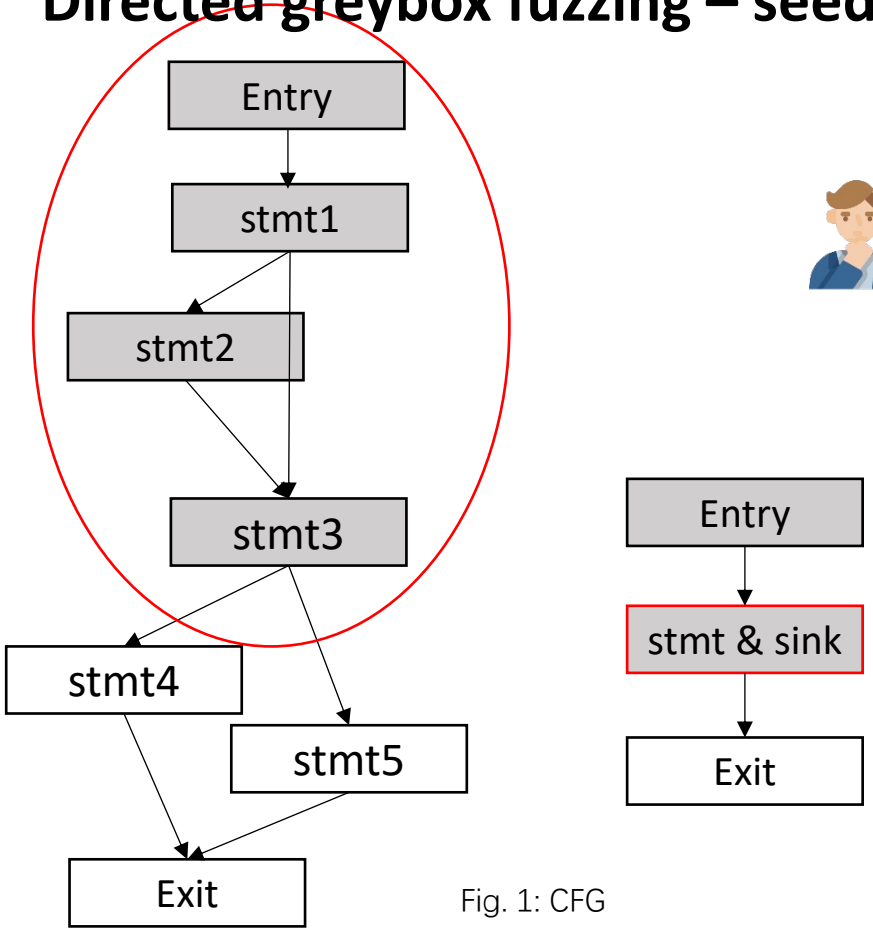


Fig. 1: CFG

How to efficiently select and schedule the seeds to trigger sensitive sinks?

- Distance + Coverage
- Distance: more likely to reach the sink method
- Coverage: exploring more paths

Closer to sink

$$p(s, T_b) = \varphi(s) \cdot (1 - \tilde{d}(s, T_b))$$

Trigger more branches in the gadget (exploring diverse execution paths)

Directed greybox fuzzing – Highlight

1. Goal: verify whether the candidate gadget chain is reachable and exploitable.

2. Solution: Directed greybox fuzzing.

3. Techniques:
   - Seed generation: Property tree
   - Seed mutation: Pre-generated values
   - Seed prioritization: Hybrid feedback

Agenda

- Introduction
- Previous work & Remaining challenges
- ODDFuzz: A novel approach to hunting gadget chains
- Evaluation
- Conclusion & Takeaways

Evaluation

Experimental setup

- **Target Applications**
  - Known gadget chains reproduction: 22 Java libraries (covering 34 chains) from ysoserial
  - Unknown gadget chains discovery: Well-known applications (including Oracle WebLogic Server, Sonatype Nexus, Apache Dubbo, protostuff)

- **Implementations**
  - Repeat each experiment 10 times and report the average statistical performance.
  - Set the threshold for each gadget chain to 15 gadgets.
  - Limit the fuzzing campaign to 120 seconds

Evaluation - baseline

Application	Version	LoC	Classes	Methods	Covered Sources	Covered Sinks	Known Chains	Identified Chains	Confirmed Chains	Analysis Time	Fuzzing Time
JDK	1.7	4.4M	38.5K	324.6K	7	4	4	9 (1)	1	1m51s	16m32s
AspectJWeaver	1.9.2	692.4K	7.1K	19.8K	4	2	1	9 (1)	0	1m56s	18m
BeanShell	2.0b5	44.8K	1.1K	17K	3	1	1	8 (0)	0	1m53s	16m
C3P0	0.9.5.2	30.3K	644	10.1K	6	3	1	13 (1)	1	1m50s	25m53s
Click	2.3.0	10.8K	73	8.5K	4	1	1	8 (1)	1	1m48s	15m26s
Clojure	1.8.0	58.4K	3.8K	25.7K	5	4	1	184 (1)	1	3m30s	6h7m34s
CommonsBeanutils	1.9.2	71.4K	504	7.8K	3	1	1	8 (1)	1	1m52s	14m25s
CommonsCollections	3.1	101K	798	9.7K	7	4	5	97 (5)	3	1m58s	3h10m53s
CommonsCollections4	4.0	101K	630	7.4K	5	2	2	112 (2)	2	1m55s	3h41m9s
FileUpload	1.3.1	10.5K	56	3.1K	3	1	1	8 (0)	0	1m55s	16m
Groovy	2.3.9	252.4K	4.2K	45.6K	4	1	1	13 (0)	0	2m8s	26m
Hibernate	4.3.11	855.7K	7.4K	42.7K	3	1	2	8 (2)	2	2m8s	14m7s
JBossInterceptors	2.0.0	24.2K	166	2.3K	2	1	1	8 (0)	0	1m51s	16m
JSON	2.4	28K	172	5.9K	3	2	1	9 (0)	0	1m52s	18m
JavassistWeld	3.12.1	60.4K	813	11.3K	2	1	1	8 (0)	0	1m58s	16m
Jython	2.5.2	271.9K	6.7K	66.4K	4	1	1	32 (1)	0	2m54s	1h4m
MozillaRhino	1.7R2	118.7K	329	8.2K	4	2	2	7 (2)	2	1m56s	12m10s
Myfaces	2.2.9	330.1K	1.8K	22.8K	2	1	2	7 (0)	0	2m1s	14m
ROME	1.0	94.5K	423	6.9K	2	1	1	5 (1)	1	1m48s	8m53s
Spring	4.1.4	904.3K	1.3K	14.5K	3	2	2	10 (0)	0	1m59s	20m
Vaadin	7.7.14	572.1K	4.5K	17.5K	4	1	1	13 (1)	1	1m54s	24m37s
Wicket	6.23.0	420.7K	3.2K	11.1K	2	1	1	7 (0)	0	1m50s	14m
Total		-	-	-	-	-	34	583 (20)	16	-	-

Evaluation - baseline

Application	Known Chains	GadgetInspector			SerHybrid			ODDFuzz			
		Identified Chains	Confirmed Chains	Analysis Time	Identified Chains	Confirmed Chains	Analysis Time	Identified Chains	Confirmed Chains	Analysis Time	Fuzzing Time
JDK	4	5	0	53s	N/A	N/A	N/A	9 (1)	1	1m51s	16m32s
AspectJWeaver	1	6	0	41s	N/A	N/A	N/A	9 (1)	0	1m56s	18m
BeanShell	1	2	0	49s	1	0	10m55s	8 (0)	0	1m53s	16m
C3P0	1	2	0	48s	N/A	N/A	N/A	13 (1)	1	1m50s	25m53s
Click	1	4	0	39s	N/A	N/A	N/A	8 (1)	1	1m48s	15m26s
Clojure	1	12	1	40s	N/A	N/A	Timeout	184 (1)	1	3m30s	6h7m34s
CommonsBeanutils	1	2	0	37s	0	0	13m6s	8 (1)	1	1m52s	14m25s
CommonsCollections	5	4	1	39s	1	1	26m51s	97 (5)	3	1m58s	3h10m53s
CommonsCollections4	2	4	0	38s	1	1	11m21s	112 (2)	2	1m55s	3h41m9s
FileUpload	1	3	0	38s	N/A	N/A	N/A	8 (0)	0	1m55s	16m
Groovy	1	4	0	47s	3	0	1h26m	13 (0)	0	2m8s	26m
Hibernate	2	3	0	41s	3	0	56m37s	8 (2)	2	2m8s	14m7s
JBossInterceptors	1	2	0	38s	N/A	N/A	N/A	8 (0)	0	1m51s	16m
JSON	1	2	0	39s	N/A	N/A	N/A	9 (0)	0	1m52s	18m
JavassistWeld	1	2	0	39s	N/A	N/A	N/A	8 (0)	0	1m58s	16m
Jython	1	42	1	50s	N/A	N/A	Timeout	32 (1)	0	2m54s	1h4m
MozillaRhino	2	3	0	40s	N/A	N/A	N/A	7 (2)	2	1m56s	12m10s
Myfaces	2	2	0	37s	N/A	N/A	N/A	7 (0)	0	2m1s	14m
ROME	1	2	0	36s	0	0	6m30s	5 (1)	1	1m48s	8m53s
Spring	2	2	0	38s	N/A	N/A	N/A	10 (0)	0	1m59s	20m
Vaadin	1	5	0	37s	N/A	N/A	N/A	13 (1)	1	1m54s	24m37s
Wicket	1	3	0	36s	N/A	N/A	N/A	7 (0)	0	1m50s	14m
Total	34	116	3	-	9	2	-	583 (20)	16	-	-

Evaluation - baseline

ID	Gadget Chain	Affected Version	# Gadgets	ODDFuzz		GadgetInspector	SerHybrid
				Identified	Validated		
1	*AspectJWeaver*	aspectjweaver-1.9.2	9	✓	✗	-	-
2	*BeanShell1*	bsh-2.0b5	6	-	-	-	-
3	*C3P0*	c3p0-0.9.5.2	6	✓	✓	-	-
4	*Click1*	click-nodeps-2.3.0	10	✓	✓	-	-
5	*Clojure*	clojure-1.8.0	10	✓	✓	✓	-
6	*CommonsBeanutils1*	commons-beanutils-1.9.2	5	✓	✓	-	-
7	*CommonsCollections1*	commons-collections-3.1	7	✓	✗	✓	-
8	*CommonsCollections2*	commons-collections4-4.0	13	✓	✓	-	✓
9	*CommonsCollections3*	commons-collections-3.1	13	✓	✗	-	-
10	*CommonsCollections4*	commons-collections4-4.0	15	✓	✓	-	-
11	*CommonsCollections5*	commons-collections-3.1	8	✓	✓	-	-
12	*CommonsCollections6*	commons-collections-3.1	10	✓	✓	-	✓
13	*CommonsCollections7*	commons-collections-3.1	9	✓	✓	-	-
14	*FileUpload1*	commons-fileupload-1.3.1	3	-	-	-	-
15	*Groovy1*	groovy-2.3.9	10	-	-	-	-
16	*Hibernate1*	hibernate-core-4.3.11.Final	7	✓	✓	-	-
17	*Hibernate2*	hibernate-core-4.3.11.Final	9	✓	✓	-	-
18	*JBossInterceptors1*	jboss-interceptor-core:2.0.0.Final	5	-	-	-	-
19	*JRMPClient*	JDK-1.7	13	-	-	-	-
20	*JRMPListener*	JDK-1.7	9	-	-	-	-
21	*JSON1*	json-lib:jar-jdk15:2.4	22	-	-	-	-
22	*JavassistWeld1*	javassist-3.12.1.GA	5	-	-	-	-
23	*Jdk7u21*	JDK-1.7	11	-	-	-	-
24	*Jython1*	jython-standalone-2.5.2	5	✓	✗	✓	-
25	*MozillaRhino1*	js-1.7R2	8	✓	✓	-	-
26	*MozillaRhino2*	js-1.7R2	12	✓	✓	-	-
27	*Myfaces1*	myfaces-impl-2.2.9	4	-	-	-	-
28	*Myfaces2*	myfaces-impl-2.2.9	6	-	-	-	-
29	*ROME*	rome-1.0	15	✓	✓	-	-
30	*Spring1*	spring-core:4.1.4.RELEASE	11	-	-	-	-
31	*Spring2*	spring-core:4.1.4.RELEASE	12	-	-	-	-
32	*URLDNS*	JDK	7	✓	✓	-	-
33	*Vaadin1*	vaadin-server-7.7.14	10	✓	✓	-	-
34	*Wicket1*	wicket-util-6.23.0	3	-	-	-	-

No.	Application	Version	Impact	Status	CVE-ID
1	WebLogic	12.2.1.4.0	RCE	Patched	CVE-2020-14756
2	WebLogic	12.2.1.4.0	RCE	Patched	CVE-2020-14825
3	WebLogic	12.2.1.4.0	RCE	Patched	CVE-2021-2135
4	Sonatype Nexus	3.25.0	RCE	Patched	CVE-2020-15871
5	Apache Dubbo	2.7.7	RCE	Patched	CVE-2020-11995
6	ProtoStuff	1.8.0	RCE	Reported	

Effectiveness

- ✓ Detect ***13*** unique gadget chains that cannot be found by baselines.
- ✓ Report ***6*** exploitable Java ODD vulnerabilities, ***5*** of them have been assigned CVE-IDs.

Evaluation – case study

WebLogic coherence RCE

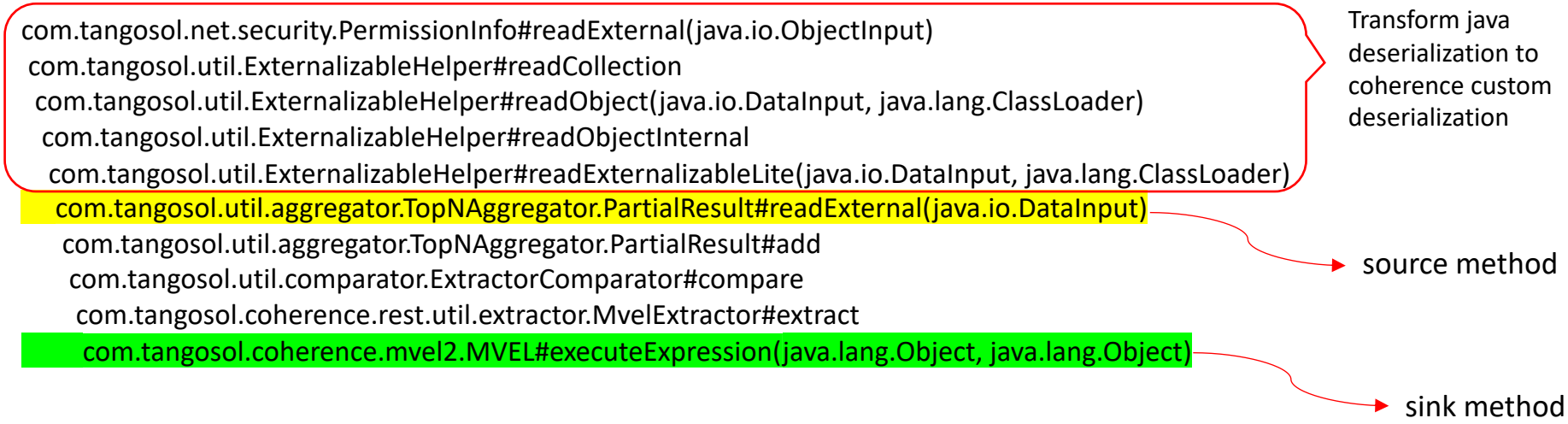


Fig. 1: candidate gadget chain reported by the lightweight static analyzer

Evaluation – case study

com.tangosol.util.aggregator.TopNAggregator.PartialResult#readExternal(java.io.DataInput)
com.tangosol.util.aggregator.TopNAggregator.PartialResult#add
com.tangosol.util.comparator.ExtractorComparator#compare
com.tangosol.coherence.rest.util.extractor.MvelExtractor#extract
com.tangosol.coherence.mvel2.MVEL#executeExpression(java.lang.Object, java.lang.Object)

```
public void readExternal(DataInput in) throws IOException {
    this.m_comparator = (Comparator)ExternalizableHelper.readObject(in);
    this.m_cMaxSize = ExternalizableHelper.readInt(in);
    this.m_map = this.
    int cElems = in.re

    for(int i = 0; i <
        this.add(Exter
    }

    this.m_comparator_
}
```

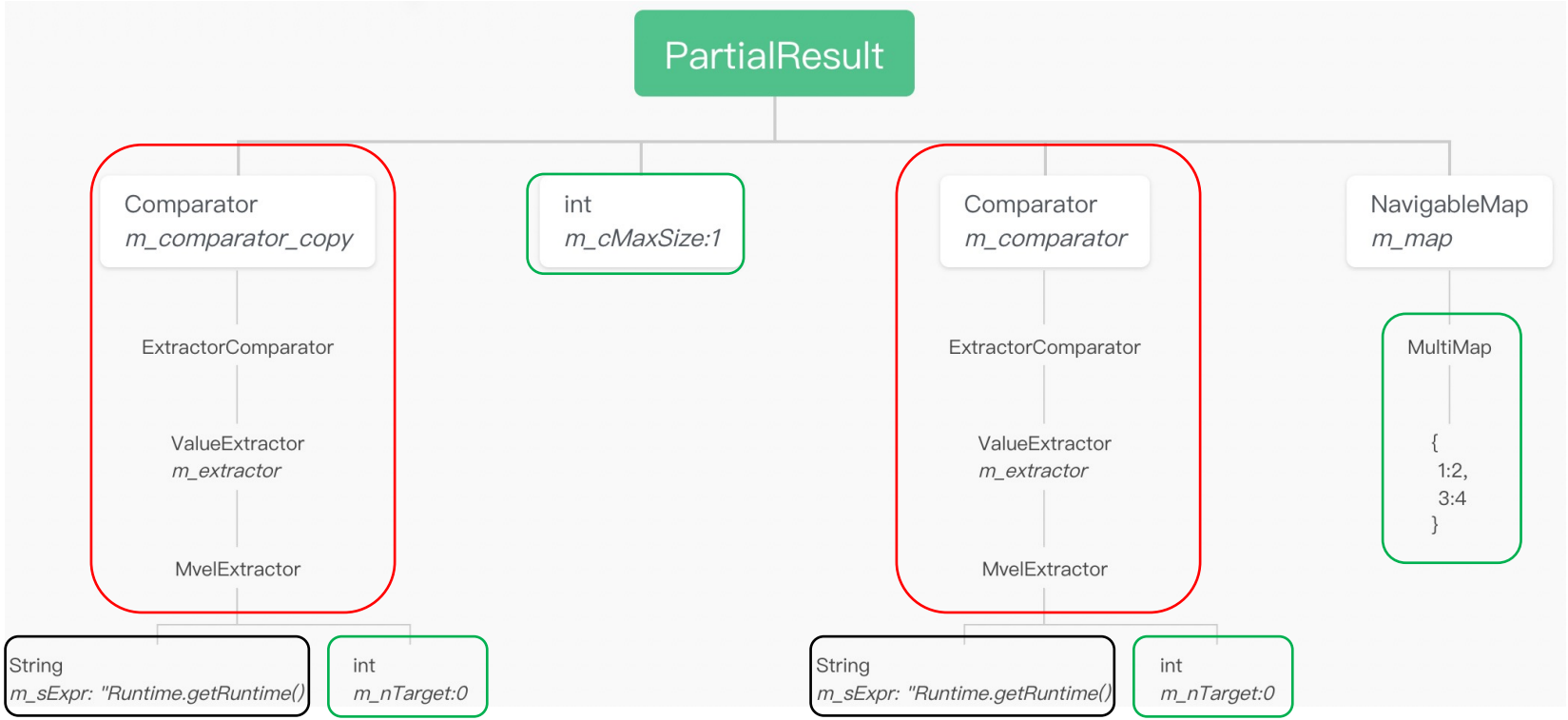
```
public boolean add(E value) {
    if (this.size() < this.m_cMaxSize) {
        return super.add(value);
    } else if (this.m_comparator.compare(value, this.first()) > 0) {
        this.removeFi
        super.add(val
        return true;
    } else {
        return false;
    }
}
```

```
public int compare(T o1, T o2) {
    Comparable a1 = o1 instanceof Entry ? (Comparable)((Entry)o1).extract
(this.m_extractor) : (Comparable)this.m_extractor.extract(o1);
    Comparable a2 = o2 instanceof Entry ? (Comparable)((Entry)o2).extract
(this.m_extractor) : (Comparable)this.m_extractor.extract(o2);

}
```

```
public Object extract(Object oTarget) {
    return oTarget == null ? null : MVEL.executeExpression(this.getCompile
dExpression(), oTarget);
}
```

Evaluation – case study



Pre-generated

Randomly generated

Hardcoded

Evaluation – case study

com.tangosol.util.aggregator.TopNAggregator.PartialResult#readExternal(java.io.DataInput)
com.tangosol.util.aggregator.TopNAggregator.PartialResult#add
com.tangosol.util.comparator.ExtractorComparator#compare
com.tangosol.coherence.rest.util.extractor.MvelExtractor#extract
com.tangosol.coherence.mvel2.MVEL#executeExpression(java.lang.Object, java.lang.Object)

Fig. 1: expected stack trace

com.tangosol.util.aggregator.TopNAggregator.PartialResult#readExternal(java.io.DataInput)
com.tangosol.util.aggregator.TopNAggregator.PartialResult#add
com.tangosol.util.SortedBag#add
java.util.TreeMap#put
java.util.TreeMap#compare
com.tangosol.util.SortedBag.WrapperComparator#compare
com.tangosol.util.comparator.ExtractorComparator#compare
com.tangosol.coherence.rest.util.extractor.MvelExtractor#extract
com.tangosol.coherence.mvel2.MVEL#executeExpression(java.lang.Object, java.lang.Object)

Fig. 2: actual stack trace

Evaluation – case study

```java
public void readExternal(DataInput in) throws IOException {
    this.m_comparator = (Comparator)ExternalizableHelper.readObject(in);
    this.m_cMaxSize = ExternalizableHelper.readInt(in);
    this.m_map = this.instantiateInternalMap(this.m_comparator);
    int cElems = in.readInt();

    for(int i = 0; i < cElems; ++i) {
        this.add(ExternalizableHelper.readObject(in));
    }

    this.m_comparator_copy = this.m_comparator;
}
```

```java
protected NavigableMap instantiateInternalMap(Comparator comparator) {
    return new TreeMap(new SortedBag.WrapperComparator(comparator));
}
```

Trigger the sink method earlier than expected, but sink method has been executed.

com.tangosol.util.aggregator.TopNAggregator.PartialResult#readExternal(java.io.DataInput)
com.tangosol.util.aggregator.TopNAggregator.PartialResult#add
com.tangosol.util.SortedBag#add
java.util.TreeMap#put
java.util.TreeMap#compare
com.tangosol.util.SortedBag.WrapperComparator#compare
com.tangosol.util.comparator.ExtractorComparator#compare
com.tangosol.coherence.rest.util.extractor.MvelExtractor#extract
com.tangosol.coherence.mvel2.MVEL#executeExpression(java.lang.Object, java.lang.Object)

Evaluation – discussion

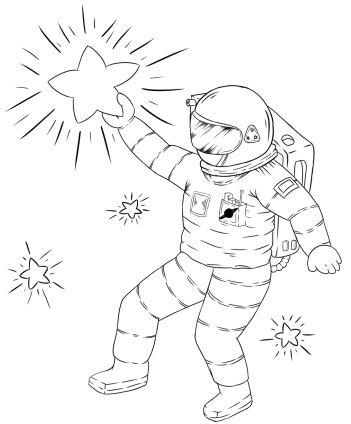
1. Runtime polymorphism and other dynamic language features make it difficult to make trade-offs between precision and recall.

2. Java deserialization gadget chains can be quite long, which causes huge computation space and amplifies the inaccuracy.

3. Existing tools are unable to validate candidate gadget chains, which require manual inspection and are time-consuming and error-prone.

Evaluation – future work

1. Better static analysis.
2. More dynamic features support.
3. Automatic exploit construction.



Agenda

- Introduction
- Previous work & Remaining challenges
- ODDFuzz: A novel approach to hunting gadget chains
- Evaluation
- Conclusion & Takeaways

Conclusion & Takeaways

1. Attendees will know the current state and remaining challenges of state-of-the-art gadget chain discovery tools.

2. Attendees will understand how ODDFuzz works and why it hunts gadget chains efficiently and precisely.

3. Attendees will learn how to optimize fuzzing when adapting it to an object-oriented language.

Thanks

@codeplutos

@meizjm3i