45th International Conference on Software Engineering

Improving Java Deserialization Gadget Chain Mining via Overriding-Guided Object Generation

Sicong Cao[1], Xiaobing Sun[1], Xiaoxue Wu[1], Lili Bo[1], Bin Li[1], Rongxin Wu[2], Wei Liu[1], Biao He[3], Yu Ouyang[3], and Jiajia Li[3]

[1]Yangzhou University
[2]Xiamen University
[3]Ant Group

Back to 2015

Marshalling Pickles

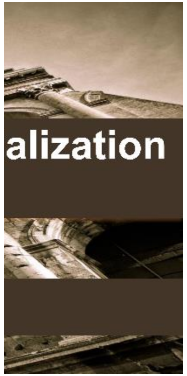
how deserializin

Gabriel Lawrence (@gebl) a

QUALCOMM

Defending against Java

OWASP TOP 10 – 2013	OWASP TOP 10 – 2017
A1 – Injection	A1 – Injection
A2 – Broken Authentication and Session Management	A2 – Broken Authentication
A3 – Cross-Site Scripting (XSS)	A3 – Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged + A7]	A4 – XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	A5 – Broken Access Control [MERGED]
A6 – Sensitive Data Exposure	A6 – Security Misconfiguration
A7 – Missing Function Level Access Control [Merged + A4]	A7 – Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	A8 – Insecure Deserialization [NEW, COMMUNITY]
A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	A10 – Insufficient Logging & Monitoring [NEW, COMMUNITY]



2015: *Chri*
their resea
ultimately
the biggest

ORACLE WEBLOGIC

JBoss by Red Hat

IBM WebSphere

What is Java Deserialization ?
Why is it so serious ?

Java Deserialization

Serialization

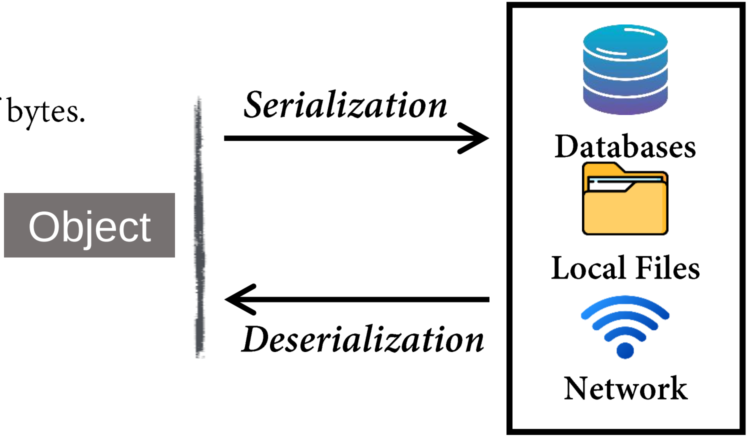
- The process of converting a Java object into stream of bytes.

Deserialization

- A reverse process of creating a Java object from stream of bytes.

Used for ?

- Remote method invocation.
- Transfer the object to remote system via network.
- Store the object in database or local files for reusing.



Controlling Data Types => Controlling Code !

```java
public static class Cat implements Animal,Serializable {
    @Override public void eat() {
          System.out.println("cat eat fish");
    }
}
public static class Dog implements Animal,Serializable {
    @Override
    public void eat() {
        try {
            Runtime.getRuntime().exec("calc");
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("dog eat bone");
    }
}
public static class Person implements Serializable {
    private Animal pet;
    public Person(Animal pet){
        this.pet = pet;
    }
    private void readObject(java.io.ObjectInputStream stream)
            throws IOException, ClassNotFoundException {
        pet = (Animal) stream.readObject();
        pet.eat();
    }
}
public static void main(String[] args) throws Exception {
    Animal animal = new Dog();
    Person person = new Person(animal);
    GeneratePayload(person,"test.ser");
    payloadTest("test.ser");
}
```

Controlling Data Types => Controlling Code !

```
public static class Cat implements Animal,Serializable {
    @Override public void eat() {
          System.out.println("cat eat fish");
    }
}
public static class Dog implements Animal,Serializable {
    @Override
    public void eat() {
        try {
            Runtime.getRuntime().exec("calc");
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("dog eat bone");
    }
}
public static class Person implements Serializable {
    private Animal pet;
    public Person(Animal pet){
        this.pet = pet;
    }
    private void readObject(java.io.ObjectInputStream stream)
            throws IOException, ClassNotFoundException {
        pet = (Animal) stream.readObject();
        pet.eat();
    }
}
public static void main(String[] args) throws Exception {
    Animal animal = new Dog();
    Person person = new Person(animal);
    GeneratePayload(person,"test.ser");
    payloadTest("test.ser");
}
```

Controlling Data Types => Controlling Code !

```java
public static class Cat implements Animal,Serializable {
    @Override public void eat() {
          System.out.println("cat eat fish");
    }
}
public static class Dog implements Animal,Serializable {
    @Override
    public void eat() {
        try {
            Runtime.getRuntime().exec("calc");
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("dog eat bone");
    }
}
public static class Person implements Serializable {
    private Animal pet;
    public Person(Animal pet){
        this.pet = pet;
    }
    private void readObject(java.io.ObjectInputStream stream)
            throws IOException, ClassNotFoundException {
        pet = (Animal) stream.readObject();
        pet.eat();
    }
}
public static void main(String[] args) throws Exception {
    Animal animal = new Dog();
    Person person = new Person(animal);
    GeneratePayload(person,"test.ser");
    payloadTest("test.ser");
}
```

Controlling Data Types => Controlling Code !

```java
public static class Cat implements Animal,Serializable {
    @Override public void eat() {
        System.out.println("cat eat fish");
    }
}
public static class Dog implements Animal,Serializable {
    @Override
    public void eat() {
        try {
            Runtime.getRuntime().exec("calc");
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("dog eat bone");
    }
}
public static class Person implements Serializable {
    private Animal pet;
    public Person(Animal pet){
        this.pet = pet;
    }
    private void readObject(java.io.ObjectInputStream stream)
            throws IOException, ClassNotFoundException {
        pet = (Animal) stream.readObject();
        pet.eat();
    }
}
public static void main(String[] args) throws Exception {
    Animal animal = new Dog();
    Person person = new Person(animal);
    GeneratePayload(person,"test.ser");
    payloadTest("test.ser");
}
```

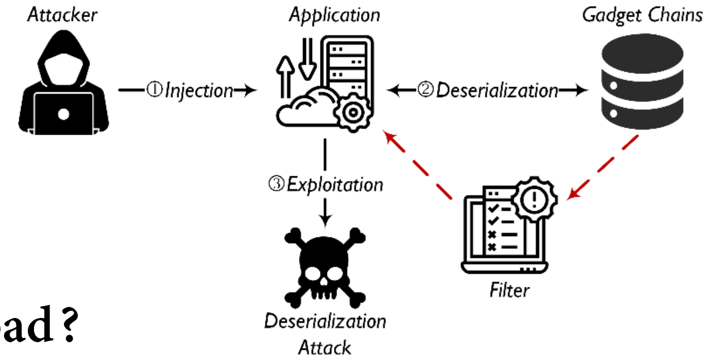
```java
public static class Person implements Serializable {
        private Animal pet = new cat();
        public Person(Animal pet){
            this.pet = pet;
        }
        private void readObject(java.io.ObjectInputStream stream)
                throws IOException, ClassNotFoundException {
            pet = (Animal) stream.readObject();
            pet.eat();
        }
    }
    public static void main(String[] args) throws Exception {
        Animal animal = new Dog();
        Person person = new Person(animal);

        Field field = person.getClass().getDeclaredField("pet");
        field.setAccessible(true);
        field.set(person, animal);

        GeneratePayload(person,"test.ser");
        payloadTest("test.ser");
    }
```

Gadget Chain:

readObject() -> eat() -> getRuntime().exec()

Attack Scenario

- A remote service accept untrusted data for deserializing.
- The classpath of the application includes serializable class.
- Dangerous function in the callback of serializable class.



Why are deserialization vulnerabilities so bad？

Magic methods get executed ***automatically*** by the deserializer, even before deserialization finishes !

Magic Method

- *Object.readObject()*
- *Object.readResolve()*
- *Object.finalize()*

……

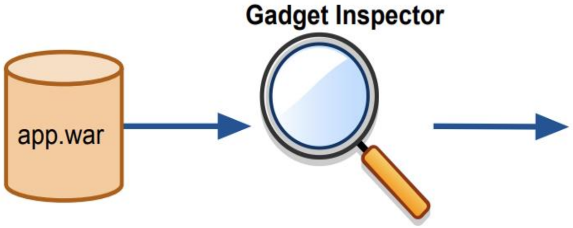
- *HashMap*
  - ✓ *Object.hashCode()*
  - ✓ *Object.equals()*
- *PriorityQueue*
  - ✓ *Comparator.compare()*
  - ✓ *Comparable.CompareTo()*

……

Existing Solutions

Gadget Inspector (BlackHat 2018)

Static Analysis + Symbolic Execution



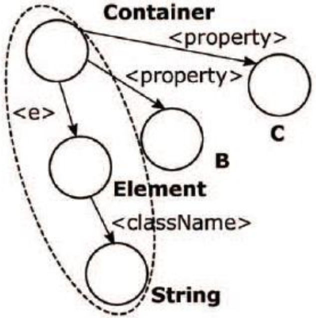
- CertificateRevokedException.readObject()
- Collections$CheckedMap.put()
- TreeMap.put()
- scala/math/Ordering$$anon$5.compare()
- scala/PartialFunction$OrElse.apply()
- ...

High false positive rate

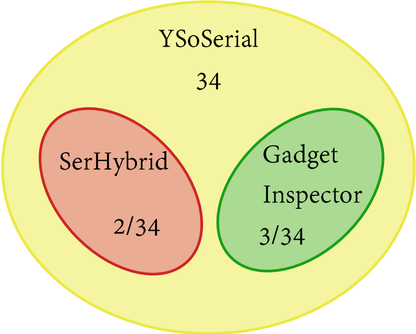
High false negative rate

SerHybrid (ASE 2022)

Points-to Analysis + Heap-based Fuzzing



High false negative rate



How to improve?
An Empirical Study

Research Questions

- **RQ1:** How are Java deserialization gadgets exploited?
- **RQ2:** How are gadget chains constructed?

TABLE I: Benchmark information.

Library	Affected Application	#Chain	Type
-	ysoserial	34	-
YAML	JBoss RESTEasy	1	RCE
	Apache Camel	2	
	Apache Brooklyn	1	
	Apache XBean	1	
JDK	Shiro	3	JNDIi
	Pippo	2	RCE
BlazeDS	Adobe Coldfusion	2	RCE
	VMWare VCenter	1	
Red5	Red5	1	RCE
Hessian	Hessian	5	RCE
XStream	XStream	14	RCE SRA
Others	Commons Collections	3	RCE
	Dubbo	2	RCE
	WebLogic	5	RCE JNDIi
	Emissary	3	SSRF
	Jenkins	2	RCE
	Apache OFBiz	3	RCE
	Spring	1	JNDIi
Total		**86**	-

- **Step 1:** Chose ysoserial repository, a famous project that provides 34 Java payloads with corresponding gadget chains exploited in publicly known deserialization attacks.
- **Step 2:** *Manually* collect public Java deserialization gadget chains from well-known vulnerability disclosure platforms such as NVD, CVE, Exploit-DB.
- **Step 3:** Filter out entries which do not 1) belong to open-source applications, 2) support deserialization operations, and 3) contain sufficient information for verification.

In total, we collect 86 exploitable gadget chains, covering 18 Java applications, 52 out of which are new.

Research Questions

- **RQ1:** How are Java deserialization gadgets exploited？
- RQ2: How are gadget chains constructed？

TABLE I: Benchmark information.

Library	Affected Application	#Chain	Type
-	ysoserial	34	-
YAML	JBoss RESTEasy	1	RCE
	Apache Camel	2	
	Apache Brooklyn	1	
	Apache XBean	1	
JDK	Shiro	3	JNDIi
	Pippo	2	RCE
BlazeDS	Adobe Coldfusion	2	RCE
	VMWare VCenter	1	
Red5	Red5	1	RCE
Hessian	Hessian	5	RCE
XStream	XStream	14	RCE SRA
Others	Commons Collections	3	RCE
	Dubbo	2	RCE
	WebLogic	5	RCE JNDIi
	Emissary	3	SSRF
	Jenkins	2	RCE
	Apache OFBiz	3	RCE
	Spring	1	JNDIi
Total		**86**	-

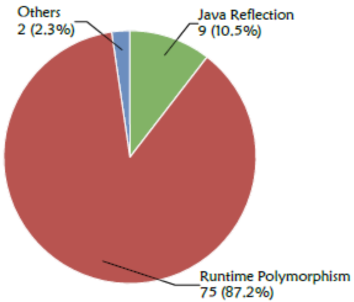


Fig. 2: Ways of exploiting available gadgets.

[Finding-1] *Java deserialization gadgets are commonly exploited by abusing advanced language features (e.g., runtime polymorphism), which enables attackers to reuse serializable overridden methods on the application's class-path.*

Research Questions

- RQ1: How are Java deserialization gadgets exploited？
- **RQ2:** How are gadget chains constructed？

TABLE I: Benchmark information.

Library	Affected Application	#Chain	Type
-	ysoserial	34	-
YAML	JBoss RESTEasy	1	RCE
	Apache Camel	2	
	Apache Brooklyn	1	
	Apache XBean	1	
JDK	Shiro	3	JNDIi
	Pippo	2	RCE
BlazeDS	Adobe Coldfusion	2	RCE
	VMWare VCenter	1	
Red5	Red5	1	RCE
Hessian	Hessian	5	RCE
XStream	XStream	14	RCE SRA
Others	Commons Collections	3	RCE
	Dubbo	2	RCE
	WebLogic	5	RCE JNDIi
	Emissary	3	SSRF
	Jenkins	2	RCE
	Apache OFBiz	3	RCE
	Spring	1	JNDIi
Total		**86**	-

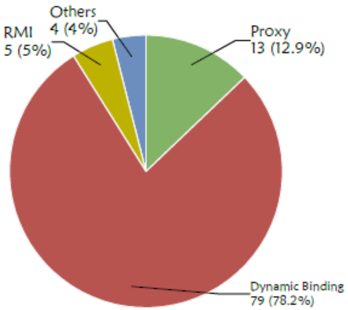
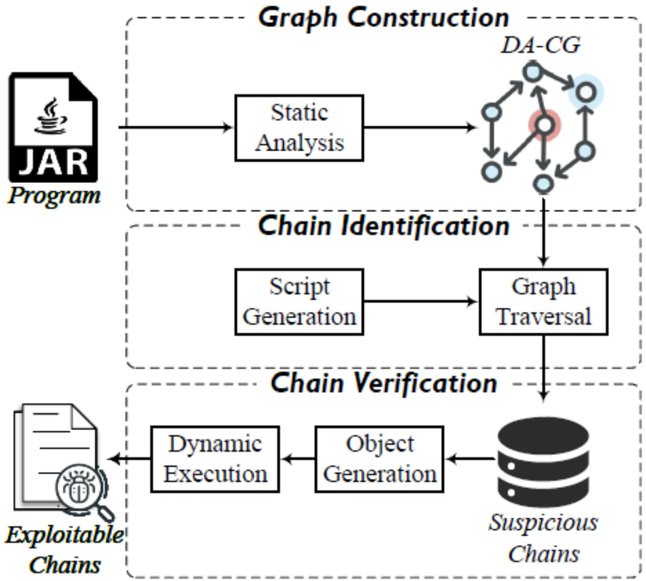


Fig. 3: Ways of gadget chain construction.

[Finding-2] *To construct exploitable gadget chains, attackers usually invoke exploitable overridden methods (gadgets) via dynamic binding to generate injection objects, which facilitate the malicious data flowing into dangerous sinks.*

Our Apporach: GCMiner

Workflow of GCMiner



Step 1: Graph Construction

- Constructing the *Deserialization-Aware Call Graph (DA-CG)* through static analysis to model both explicit and implicit method.

Step 2: Chain Identification

- Storing the DA-CG into the graph database and searches for suspicious gadget chains through graph traversal.

Step 3: Chain Verification

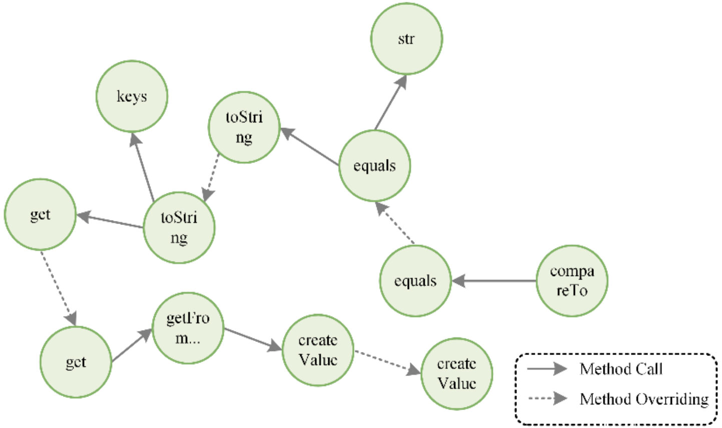
- Adopting an *overriding-guided object generation* approach to generate exploitable injection objects for fuzzing.

Step1: Graph Construction

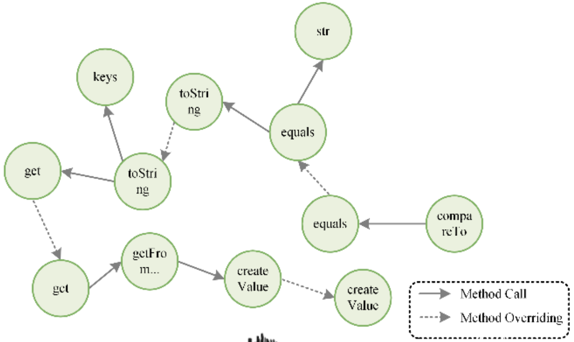
```
/*javax.naming.ldap.Rdn$RdnEntry.class*/
private Oject value;
public int compareTo(RdnEntry that) { /*Source or Magic Method*/
  if (value.equals(that.value)) {...}
                Overriding
/*com.sun.org.apache.xpath.internal.objects.XString.class*/
public boolean equals(Object obj2) { /*2nd gadget*/
  return str().equals(obj2.toString()); }
                              Overriding
/*javax.swing.MultiUIDefaults.class*/
public synchronized String toString() { /*3rd gadget*/
  Enumeration keys = keys();
  while (keys.hasMoreElements()) {
    Object key = keys.nextElement();
    buf.append(key + "=" + get(key) + ","); ...}
public Object get(Object key) { /*4th gadget*/
   Object value = super.get(key); ...}

/*javax.swing.UIDefaults.class*/
public Object get(Object key) { /*5th gadget*/
  Object value = getFromHashtable(key); ...}
private Object getFromHashtable(final Object key) { /*6th gadget*/
 if (value instanceof LazyValue) {
   try {
     value = ((LazyValue)value).createValue(this); ...}}
                         Overriding
/*sun.swing.SwingLazyValue.class*/
public Object createValue(final UIDefaults table) { /*7th gadget*/
  try {
    Class<?> c = class.forName(className, true, null);
    if (methodName != null) {
      Class[] types = getClassArray(args);
      Method m = c.getMethod(methodName, types);
      makeAccessible(m);
      return m.invoke(c, args); /*Sink or Security-Sensitive Call Site*/
```

Vulnerable Code

Deserialization-Aware Call Graph

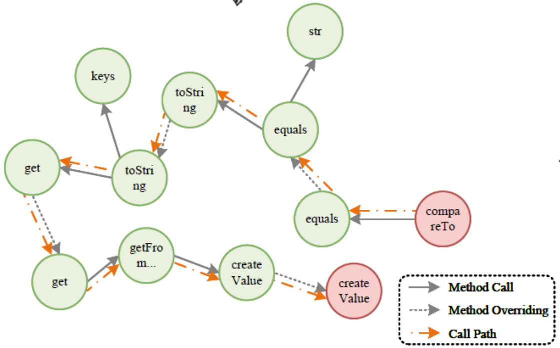


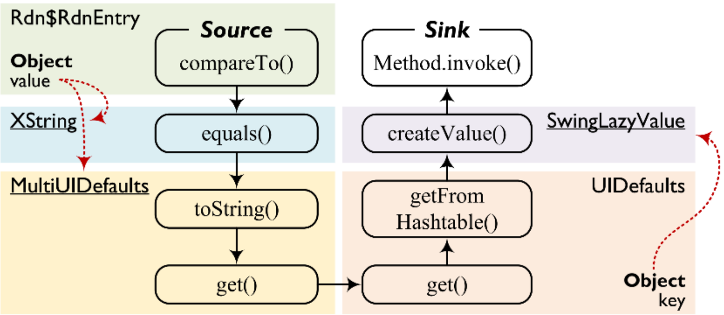
Step2: Chain Identification



Query Script

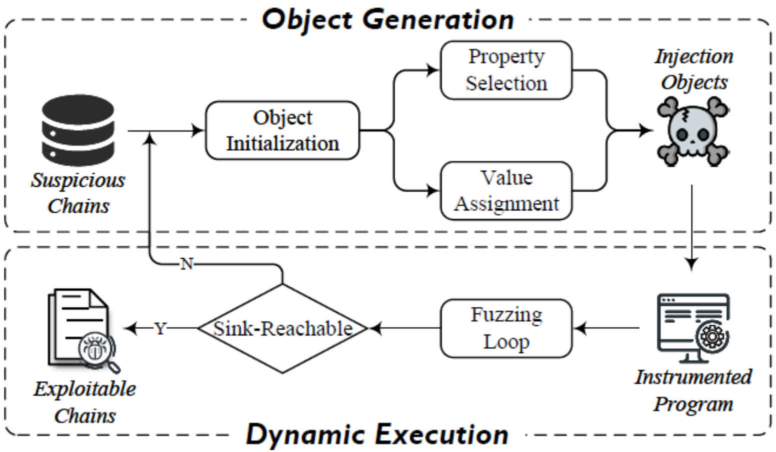
```
match (source: Method {NAME:"readObject"})
match (sink: Method {NAME:"invoke"})
call apoc.algo.allSimplePaths(sink, source, "<Call|Overriding>")
yield path
return path
```





Gadget Chain

Step3: Chain Verification



Overview

XString

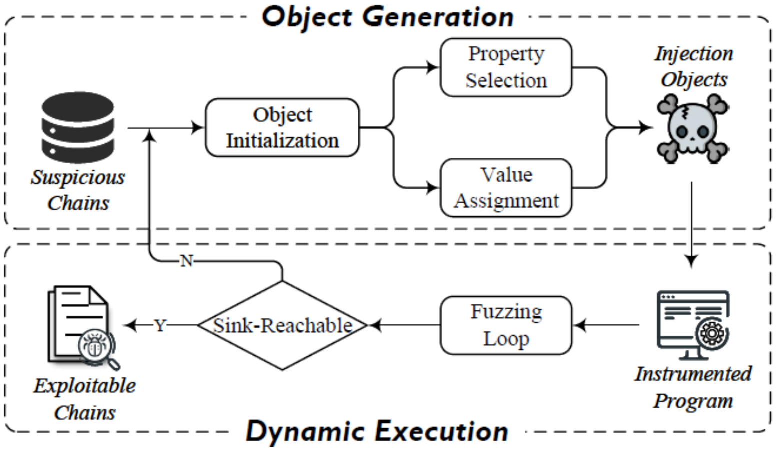
```
/*javax.naming.ldap.Rdn$RdnEntry.class*/
private Oject value;
public int compareTo(RdnEntry that) { /*Source or Magic Method*/
  if (value.equals(that.value)) {...}
                  Overriding
/*com.sun.org.apache.xpath.internal.objects.XString.class*/
public boolean equals(Object obj2) { /*2nd gadget*/
  return str().equals(obj2.toString()); }
                          Overriding
/*javax.swing.MultiUIDefaults.class*/
public synchronized String toString() { /*3rd gadget*/
  Enumeration keys = keys();
  while (keys.hasMoreElements()) {
    Object key = keys.nextElement();
    buf.append(key + "=" + get(key) + ","); ...}
public Object get(Object key) { /*4th gadget*/
   Object value = super.get(key); ...}

/*javax.swing.UIDefaults.class*/
public Object get(Object key) { /*5th gadget*/
  Object value = getFromHashtable(key); ...}
private Object getFromHashtable(final Object key) { /*6th gadget*/
 if (value instanceof LazyValue) {
   try {
     value = ((LazyValue)value).createValue(this); ...}}
                        Overriding
/*sun.swing.SwingLazyValue.class*/
public Object createValue(final UIDefaults table) { /*7th gadget*/
  try {
    Class<?> c = class.forName(className, true, null);
    if (methodName != null) {
      Class[] types = getClassArray(args);
      Method m = c.getMethod(methodName, types);
      makeAccessible(m);
      return m.invoke(c, args); /*Sink or Security-Sensitive Call Site*/
```

A. Object Generation

- Property Selection
- Value Assignment

B. Dynamic Execution

Step3: Chain Verification



Overview

A. Object Generation

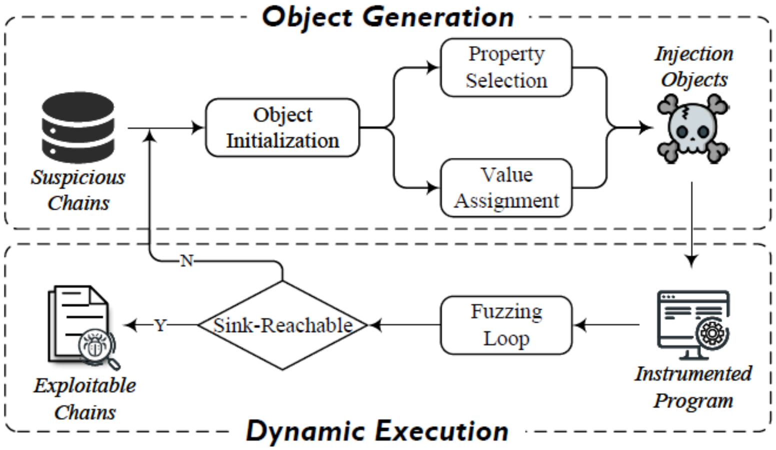
- Property Selection
- Value Assignment

B. Dynamic Execution

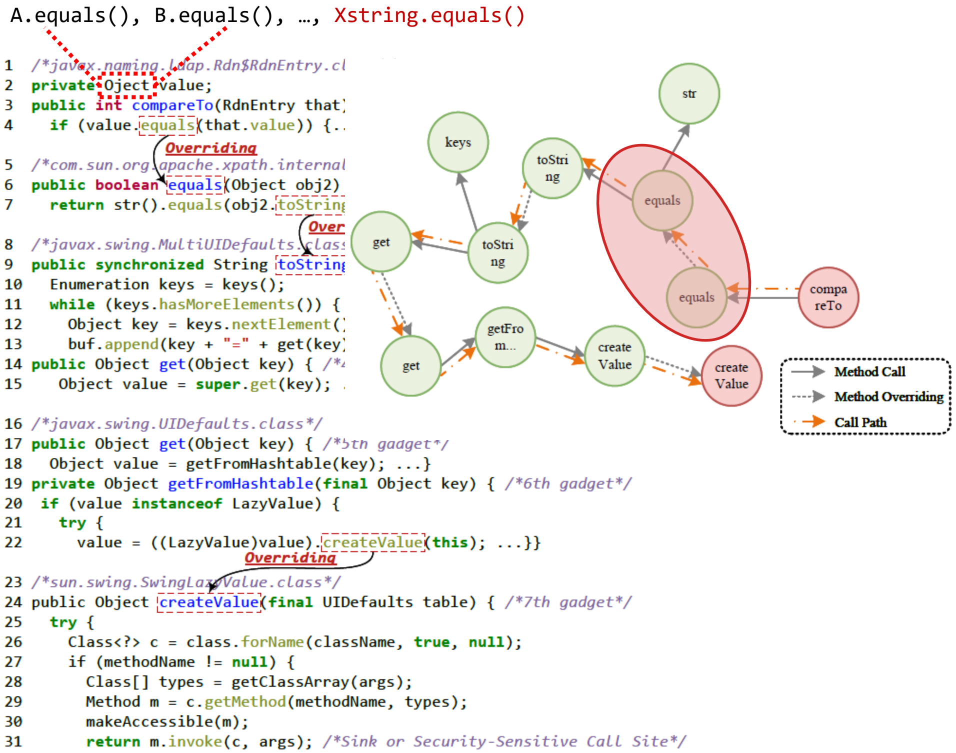
Whether this property can receive a class object?

```
/*javax.naming.ldap.Rdn$RdnEntry.class*/
private Oject value;
public int compareTo(RdnEntry that) { /*Source or Magic Method*/
  if (value.equals(that.value)) {...}
            Overriding
/*com.sun.org.apache.xpath.internal.objects.XString.class*/
public boolean equals(Object obj2) { /*2nd gadget*/
  return str().equals(obj2.toString()); }
                   Overriding
/*javax.swing.MultiUIDefaults.class*/
public synchronized String toString() { /*3rd gadget*/
  Enumeration keys = keys();
  while (keys.hasMoreElements()) {
    Object key = keys.nextElement();
    buf.append(key + "=" + get(key) + ","); ...}
public Object get(Object key) { /*4th gadget*/
   Object value = super.get(key); ...}

/*javax.swing.UIDefaults.class*/
public Object get(Object key) { /*5th gadget*/
  Object value = getFromHashtable(key); ...}
private Object getFromHashtable(final Object key) { /*6th gadget*/
 if (value instanceof LazyValue) {
   try {
     value = ((LazyValue)value).createValue(this); ...}}
                  Overriding
/*sun.swing.SwingLazyValue.class*/
public Object createValue(final UIDefaults table) { /*7th gadget*/
  try {
    Class<?> c = class.forName(className, true, null);
    if (methodName != null) {
      Class[] types = getClassArray(args);
      Method m = c.getMethod(methodName, types);
      makeAccessible(m);
      return m.invoke(c, args); /*Sink or Security-Sensitive Call Site*/
```

Step3: Chain Verification



Overview

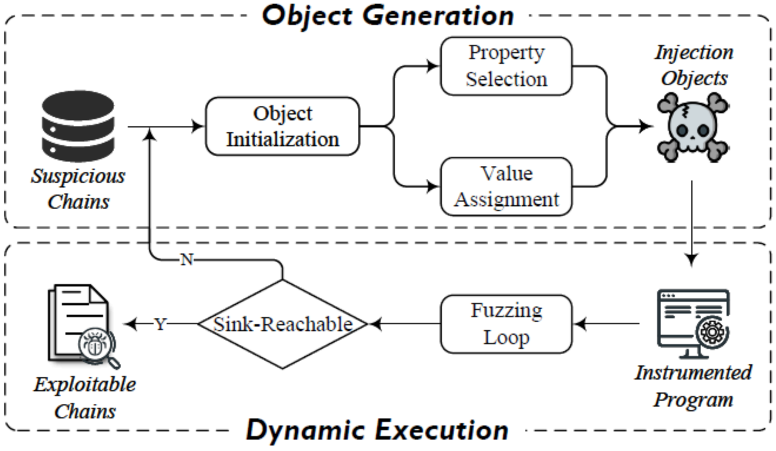


A. Object Generation

- Property Selection
- Value Assignment

B. Dynamic Execution

Step3: Chain Verification



Overview

A. Object Generation

- Property Selection
- Value Assignment

B. Dynamic Execution

Runtime Instrumentation

- **Only instrument classes to which gadgets belong on the application's classpath.**

Property-based Coverage-Guided Fuzzing

- For *primitive* data types (e.g., boolean, int), the fuzzer uses multiple pseudo-random methods built in JQF[1] to convert untyped bit parameters into random typed values.
- For *reference* data types, we tailor targeted templates for specific types. When the property type is *class*, the fuzzer will randomly select a class from the sub-classes of this property. For *array*, we randomly set up the array size and assigns random values based on the type of elements (i.e., instances that inherit the class type of the array) to the array.

1 https://github.com/rohanpadhye/JQF

So… Does GCMiner work?

Research Questions

- **RQ3:** Effectiveness of GCMiner.
- **RQ4:** Ablation study.
  - **RQ4a:** Impact of additional sources and sinks.
  - **RQ4b:** Impact of introducing method overriding.
  - **RQ4c:** Impact of overriding-guided object generation.

Evaluation Metrics

- **Known Gadget Chains (KGC)** is the number of the publicly known gadget chains in a target application.
- **Reported Gadget Chains (Rep)** computes the total number.
- **True Positives (TP)** is the number of truly exploitable gadget chains reported by each approach. In our experimental evaluation, TP counts how many known gadget chains in the benchmark are mined.
- **Precision (P)** is the fraction of truly exploitable gadget chains among the reported ones. It is calculated as: $P = TP/Rep$.
- **Recall (R)** is the fraction of known gadget chains that are identified by each approach. It is calculated as: $R = TP/KGC$.

RQ3: Effectiveness of GCMiner

Application	#KGC	*GCMiner*			*Gadget Inspector*		
		#TP/#Rep	P*	R	#TP/#Rep	P	R
ysoserial	34	21 / 29	1	0.618	3 / 116	0.026	0.088
JBoss RESTEasy	1	1 / 3	1	1	0 / 2	0	0
Apache Camel	2	2 / 2	1	1	0/ 2	0	0
Apache Brooklyn	1	1 / 1	1	1	0 / 2	0	0
Apache XBean	1	0 / 2	1	0	0 / 2	0	0
Shiro	3	1 / 2	1	0.333	0 / 2	0	0
Pippo	2	2 / 5	1	1	0 / 2	0	0
Adobe Coldfusion	2	2 / 3	1	1	1/ 2	0.500	0.500
VMWare VCenter	1	1 / 1	1	1	0 / 2	0	0
Red5	1	1 / 2	1	1	0/2	0	0
Hessian	5	4 / 7	1	0.800	0 / 2	0	0
XStream	14	12 / 19	1	0.857	1 / 2	0.500	0.071
Commons Collections	3	3 / 7	1	1	0 / 12	0	0
Dubbo	2	1 / 2	1	0.500	0 / 3	0	0
WebLogic	5	4 / 11	1	0.800	0 / 6	0	0
Emissary	3	2 / 4	1	0.667	0 / 3	0	0
Jenkins	2	1 / 9	1	0.500	0 / 2	0	0
Apache OFBiz	3	1 / 4	1	0.333	0 / 2	0	0
Spring	1	1 / 5	1	1	0 / 6	0	0
Total	**86**	**61 / 118**	**1**	**0.709**	**5 / 172**	**0.029**	**0.058**

* Since *GCMiner* adopted fuzzing to verify exploitable gadget chains, we used dynamically confirmed gadget chains as *Rep* to compute the precision.

Application	#KGC	*GCMiner*		*Serhybrid*	
		#Object	#Exploit	#Object	#Exploit
bsh-2.0b5	1	1	0	0	0
clojure-1.8.0	1	2	1	N/A	0
commons-beanutils-1.9.2	1	2	1	0	0
commons-collections-3.1	5	12	3	1	1
commons-collections4-4.0	2	4	2	1	1
groovy-2.3.9	1	2	0	0	0
hibernate	2	3	2	0	0
jython-standalone-2.5.2	1	1	0	N/A	0
rome-1.0	1	2	1	0	0
Total	**15**	**29**	**10**	**2**	**2**

False positives

- **(Static)** Limited support for certain dynamic features.
- **(Dynamic)** Hard constraints cannot be satisfied by our object generation.

Answer to RQ3

GCMiner significantly outperforms the state-of-the-art Java deserialization gadget chain mining tools, identifying 56 unique gadget chains that cannot be identified by baselines.

RQ4a: Impact of additional sources and sinks

- **Magic methods:** `hashCode`, `compareTo`, `toString`, `get`, `put`, `compare`, `readObject`, `readExternal`, `readResolve`, `finalize`, `equals`
- **Security-Sensitive Call Sites.**
  - *Remote Code Execution* (RCE): `getDeclaredMethod`, `getConstructor`, `exec`, `getMethod`, `loadClass`, `start`, `findClass`, `invoke`, `forName`, `newInstance`, `defineClass`, `<init>`, `exit`
  - *JDNI Injection* (JNDIi): `getConnection`, `connect`, `lookup`, `getObjectInstance`, `do_lookup`
  - *System Resource Access* (SRA): `newBufferedReader`, `newBufferedWriter`, `delete`, `newInputStream`, `newOutputStream`
  - *Server-Side Request Forgery* (SSRF): `openConnection`, `openStream`

Answer to RQ4a

Additional exploitable magic methods and security-sensitive call sites are useful to identify more potential gadget chains.

Application	#KGC	*GCMiner*		*GCMiner*$_{Var}$		*Gadget Inspector*$_{Var}$	
		#Rep	#TP	#Rep	#TP	#Rep	#TP
ysoserial	34	29	21	24	15	637	4
JBoss RESTEasy	1	3	1	2	1	14	0
Apache Camel	2	2	2	2	2	14	0
Apache Brooklyn	1	1	1	1	1	16	0
Apache XBean	1	2	0	1	0	14	0
Shiro	3	2	1	1	0	14	0
Pippo	2	5	2	3	1	14	0
Adobe Coldfusion	2	3	2	3	2	14	1
VMWare VCenter	1	1	1	1	1	12	0
Red5	1	2	1	1	1	14	0
Hessian	5	7	4	5	3	14	0
XStream	14	19	12	15	10	14	2
Commons Collections	3	7	3	7	3	69	0
Dubbo	2	2	1	2	1	16	0
WebLogic	5	11	4	8	3	21	0
Emissary	3	4	2	3	2	11	0
Jenkins	2	9	1	6	1	14	0
Apache OFBiz	3	4	1	2	1	14	0
Spring	1	5	1	4	1	46	0
Total	**86**	**118**	**61**	**91**	**49**	**982**	**7**

RQ4b: Impact of introducing method overriding

Application	#KGC	*With Overriding*		*W/O Overriding*	
		#Rep	#TP	#Rep	#TP
ysoserial	34	29	21	6	2
JBoss RESTEasy	1	3	1	0	0
Apache Camel	2	2	2	1	0
Apache Brooklyn	1	1	1	0	0
Apache XBean	1	2	0	0	0
Shiro	3	2	1	0	0
Pippo	2	5	2	1	0
Adobe Coldfusion	2	3	2	0	0
VMWare VCenter	1	1	1	0	0
Red5	1	2	1	0	0
Hessian	5	7	4	0	0
XStream	14	19	12	3	0
Commons Collections	3	7	3	2	1
Dubbo	2	2	1	0	0
WebLogic	5	11	4	1	0
Emissary	3	4	2	0	0
Jenkins	2	9	1	1	0
Apache OFBiz	3	4	1	0	0
Spring	1	5	1	0	0
Total	**86**	**118**	**61**	**9**	**3**

Answer to RQ4b

The introduction of overriding relations significantly enhances the capability in capturing potential exploitable gadgets.

RQ4c: Impact of overriding-guided object generation

Application	#KGC	*GCMiner*		*GCMiner$_{NG}$*	
		#Object	#Exploit	#Object	#Exploit
ysoserial	34	86	21	5	0
JBoss RESTEasy	1	3	1	0	0
Apache Camel	2	7	2	0	0
Apache Brooklyn	1	3	1	0	0
Apache XBean	1	2	0	0	0
Shiro	3	6	1	0	0
Pippo	2	5	2	0	0
Adobe Coldfusion	2	7	2	0	0
VMWare VCenter	1	3	1	0	0
Red5	1	2	1	0	0
Hessian	5	11	4	0	0
XStream	14	48	12	1	0
Commons Collections	3	8	3	1	0
Dubbo	2	4	1	0	0
WebLogic	5	13	4	0	0
Emissary	3	9	2	0	0
Jenkins	2	3	1	0	0
Apache OFBiz	3	5	1	0	0
Spring	1	4	1	0	0
Total	**86**	**229**	**61**	**7**	**0**

Answer to RQ4c

Overriding-guided object generation effectively guarantees the validity of injection objects.

Conclusion

Controlling Data Types => Controlling Code !

Gadget Chain:
readObject() -> eat() -> getRuntime().exec()

Research Questions

- RQ1: How are Java deserialization gadgets exploited?

TABLE I: Benchmark information.

Library	Affected Application	#Chain	Type
-	ysoserial	34	-
YAML	JBoss RESTEasy	1	RCE
	Apache Camel	2	
	Apache Brooklyn	1	
	Apache XBean	1	
JDK	Shiro	3	JNDI
	Pippo	2	RCE
BlazeDS	Adobe Coldfusion	2	RCE
	VMWare VCenter	1	
Red5	Red5	1	RCE
Hessian	Hessian	5	RCE
XStream	XStream	14	RCE SRA
Others	Commons Collections	3	RCE
	Dubbo	2	RCE
	WebLogic	5	RCE JNDI
	Emissary	3	SSRF
	Jenkins	2	RCE
	Apache OFBiz	3	RCE
	Spring	1	JNDI
	Total	86	-

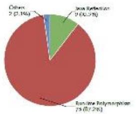
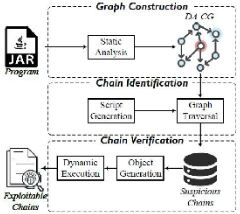


Fig. 2: Ways of exploiting available gadgets.

[Finding-1] *Java deserialization gadgets are commonly exploited by abusing advanced language features (e.g., runtime polymorphism), which enables attackers to reuse serializable overridden methods on the application's class-path.*

Workflow of GCMiner



Step 1: Graph Construction

- Constructing the *Deserialization-Aware Call Graph (DA-CG)* through static analysis to model both explicit and implicit method.

Step 2: Chain Identification

- Storing the DA-CG into the graph database and searches for suspicious gadget chains through graph traversal.

Step 3: Chain Verification

- Adopting an *overriding-guided object generation* approach to generate exploitable injection objects for fuzzing.

RQ3: Effectiveness of GCMiner

Application	#KGC	GCMiner #TP/#Rep	GCMiner P*	GCMiner R	Gadget Inspector #TP/#Rep	Gadget Inspector P	Gadget Inspector R
ysoserial	34	21 / 29	1	0.618	3 / 116	0.026	0.088
JBoss RESTEasy	1	1 / 3	1	1	0 / 2	0	0
Apache Camel	2	2 / 2	1	1	0 / 2	0	0
Apache Brooklyn	1	1 / 1	1	1	0 / 2	0	0
Apache XBean	1	0 / 2	1	0	0 / 2	0	0
Shiro	3	1 / 2	1	0.333	0 / 2	0	0
Pippo	2	2 / 5	1	1	0 / 2	0	0
Adobe Coldfusion	2	2 / 3	1	1	1 / 2	0.500	0.500
VMWare VCenter	1	1 / 1	1	1	0 / 2	0	0
Red5	1	1 / 2	1	1	0/2	0	0
Hessian	5	4 / 7	1	0.800	0 / 2	0	0
XStream	14	12 / 19	1	0.857	1 / 2	0.500	0.071
Commons Collections	3	3 / 7	1	1	0 / 13	0	0
Dubbo	2	1 / 2	1	0.500	0 / 3	0	0
WebLogic	5	4 / 11	1	0.800	0 / 6	0	0
Emissary	3	2 / 4	1	0.667	0 / 3	0	0
Jenkins	2	1 / 9	1	0.500	0 / 2	0	0
Apache OFBiz	3	1 / 4	1	0.333	0 / 2	0	0
Spring	1	1 / 5	1	1	0 / 6	0	0
Total	86	61 / 118	1	0.709	5 / 172	0.029	0.058

* Since *GCMiner* adopted fuzzing to verify exploitable gadget chains, we used dynamically confirmed gadget chains as *Rep* to compute the precision.

Application	#KGC	GCMiner #Object	GCMiner #Exploit	Serianalyzer #Object	Serianalyzer #Exploit
bsh-2.0b5	1	1	0	0	0
clojure-1.8.0	1	2	1	N/A	0
commons-beanutils-1.9.2	1	2	1	0	0
commons-collections-3.1	5	12	3	1	1
commons-collections4-4.0	2	4	2	1	1
groovy-2.3.9	1	2	0	0	0
hibernate	2	3	2	0	0
jython-standalone-2.5.2	1	1	0	N/A	0
rome-1.0	1	2	1	0	0
Total	15	29	10	2	2

False positives

- **(Static)** Limited support for certain dynamic features.
- **(Dynamic)** Hard constraints cannot be satisfied by our object generation.

Answer to RQ3

GCMiner significantly outperforms the state-of-the-art Java deserialization gadget chain mining tools, identifying 56 unique gadget chains that cannot be identified by baselines.

Thanks for listening！

DX120210088@yzu.edu.cn

https://github.com/GCMiner/GCMiner

RiSS Lab   Personal Page

揚州大學 YANGZHOU UNIVERSITY
厦門大學 XIAMEN UNIVERSITY
蚂蚁集团 ANT GROUP