



# SECURING THE JVM

Neither for fun nor for profit,  
but do you have a choice?

# ME, MYSELF AND I

- **Security-minded developer**
- **Developer advocate**





**European alternative to the “big” cloud-computing players**

- **Privacy-minded**
- **Great support**

# WHO DOESN'T KNOW THAT TRICK?

```
public class Foo {  
    private String hidden = "This should remain inaccessible!";  
}  
  
public class Oops {  
    public static void main(String[] args) throws Exception {  
        Foo foo = new Foo();  
        Class<? extends Foo> clazz = foo.getClass();  
        Field field = clazz.getDeclaredField("hidden");  
        field.setAccessible(true);  
        Object hidden = field.get(foo);  
        System.out.println(hidden);  
    }  
}
```

# REFLECTION API

- **Part of the JRE**
  - `java.lang.reflect`



# USED BY A LOT OF LANGUAGES/Frameworks

- **Dynamic languages**
  - Groovy
- **JPA frameworks**
  - Hibernate
  - Others
- **Spring**
- **Etc.**



**EXOSCALE**



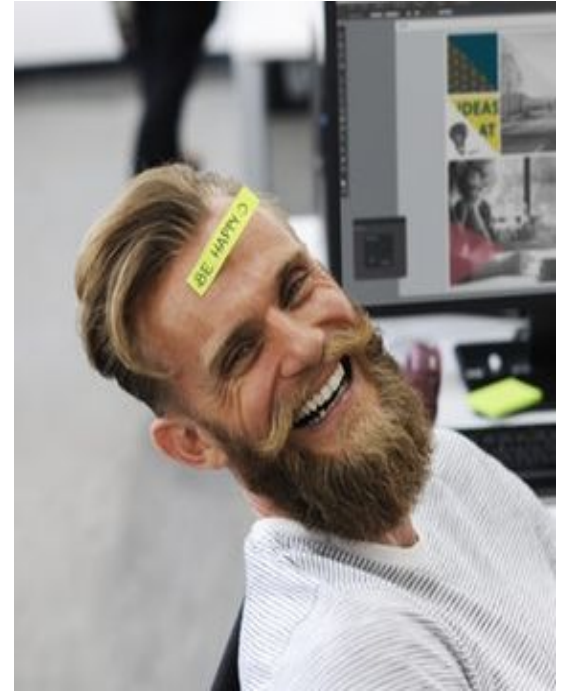
@nicolas\_frankel

# YOU CAN DO PRETTY STUPID STUFF

```
public class Bar {  
    private int hidden = 5;  
}  
  
public class OopsAgain {  
    public static void main(String[] args) throws Exception {  
        Bar bar = new Bar();  
        Class<? extends Bar> clazz = bar.getClass();  
        Field field = clazz.getDeclaredField("hidden");  
        Field type = Field.class.getDeclaredField("type");  
        AccessibleObject.setAccessible(  
            new AccessibleObject[] { field, type } , true);  
        type.set(field, String.class);  
        field.set(bar, "This should print 5!");  
        Object hidden = field.get(bar);  
        System.out.println(hidden);  
    }  
}
```

# BUT EVERYTHING IS POSSIBLE

- **Make networks calls**
- **Compile code on the fly**
- **Execute code**
- **Etc.**





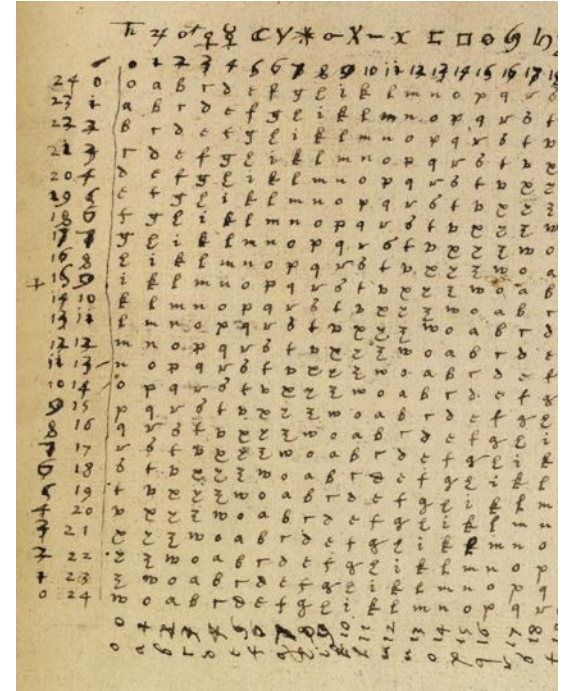
# SAFEGUARDS?

- **Static analyzers**
- **Byte-code analyzers**
- **Code reviews**
- **Security team**
- **Etc.**



# STEGANOGRAPHY

“Steganography is the practice of concealing a file, message, image, or video within another file, message, image, or video”



EXOSCALE



@nicolas\_frankel

A black laptop is shown from a front-facing perspective, slightly angled. The screen is white and displays the text "Time for DEMO" in a bold, red, sans-serif font. The text is tilted upwards from left to right. The laptop's keyboard and trackpad are visible below the screen.

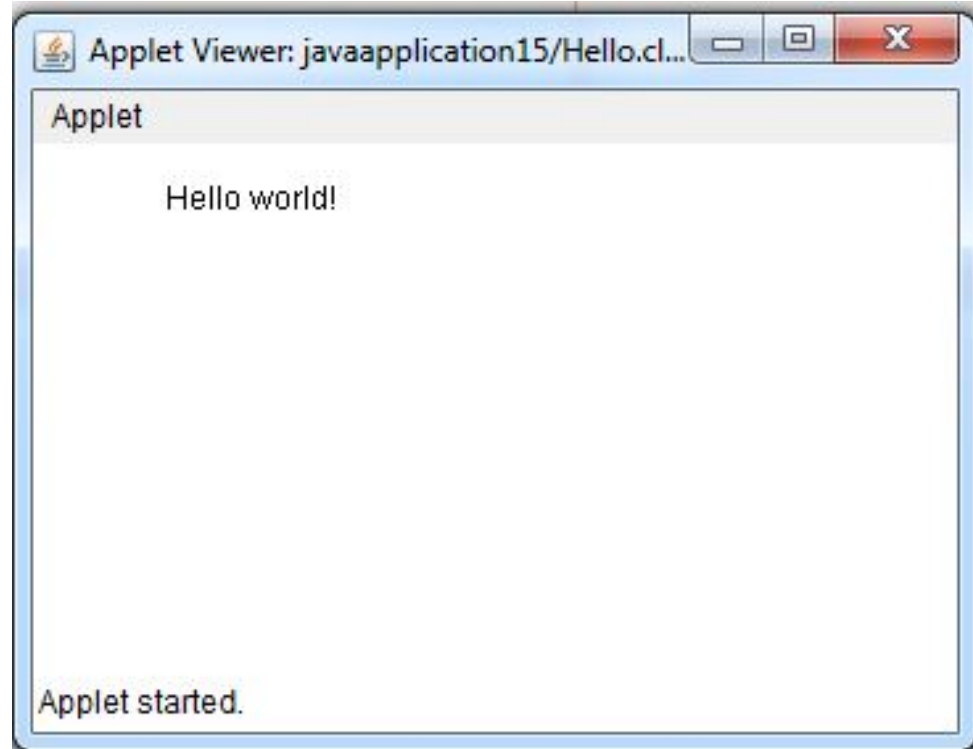
Time for **DEMO**

# A SIMPLE PROCESS

1. **Publish a new library to central**
  - With hidden steganographic/compile/run features
2. **Add the library to your POM**
3. **Add an image with embedded code**
4. **Call the library**
  - Pretend to read the image
5. **Enjoy**



# REMEMBER?



# APPLETS

- **Meant to run on the client machine**
  - Forbidden to do “dangerous” stuff
  - Run in a sandbox



# AVAILABLE MODES

- **Standard mode**
  - a.k.a. “God Mode”
- **Sandbox mode**
  - Through a SecurityManager



# ACTIVATING THE SECURITY MANAGER

- **On the command line**

- `java -Djava.security.manager ...`

- **Via the API**

- `System.setSecurityManager()`



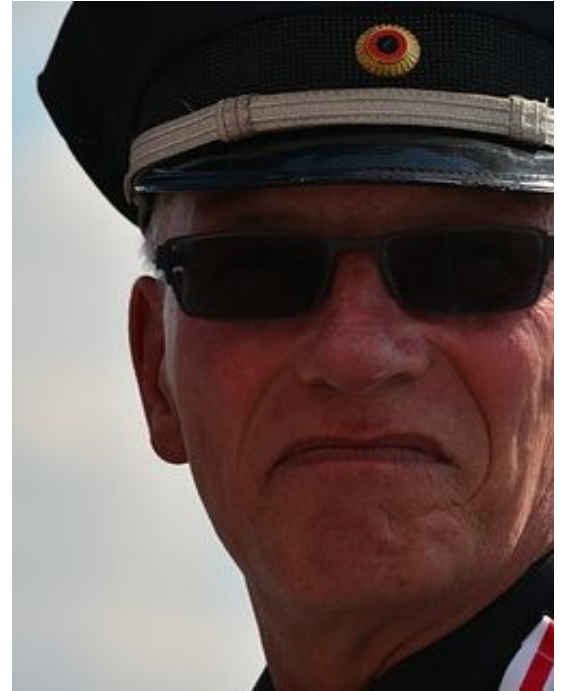
# PERMISSIONS CONFIGURATION

- **Java 9**

- `conf/security/java.security`
- → `conf/security/java.policy`

- **Java 8**

- `jre/lib/security/java.security`
- → `jre/lib/security/java.policy`

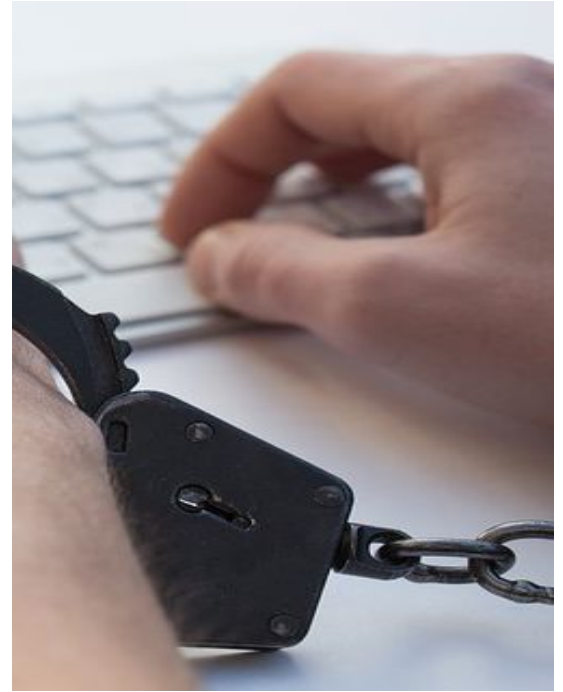


# ALTERNATIVE POLICY FILE

- **Setting the policy file**
  - `-Djava.security.policy==my.policy`
- **Adding additional permissions**
  - `-Djava.security.policy=my.policy`

# PRINCIPLE OF LEAST PRIVILEGE

**“Requires that in a particular abstraction layer of a computing environment, every module must be able to access only the information and resources that are necessary for its legitimate purpose”**



# DEFAULT PERMISSIONS

- Listen on ports
- Read a few System properties



# PERMISSIONS ARE ENFORCED IN THE JDK!

```
public class AccessibleObject
    implements AnnotatedElement {
    public void setAccessible(boolean flag)
        throws SecurityException {
        SecurityManager sm =
            System.getSecurityManager();
        if (sm != null)
            sm.checkPermission(ACCESS_PERMISSION);
        setAccessible0(this, flag);
    }
}
```

# PERMISSIONS

Permission	Details
AllPermission	☺
PropertyPermission	<ul style="list-style-type: none"><li>• Read</li><li>• Write</li></ul>
FilePermission	<ul style="list-style-type: none"><li>• Read</li><li>• Write</li><li>• Execute</li><li>• Delete</li></ul>
ReflectPermission	Suppress reflective access checks
RuntimePermission	<ul style="list-style-type: none"><li>• More granular reflective access checks</li><li>• Security-related</li><li>• Class-loading related</li><li>• Exit Virtual Machine, shutdown hooks, etc.</li></ul>

# PERMISSIONS (CONTINUED)

Permission	Details
SocketPermission	<ul style="list-style-type: none"><li>• accept</li><li>• connect</li><li>• listen</li><li>• resolve</li></ul>
SSLPermission	<ul style="list-style-type: none"><li>• Get/set SSL context</li><li>• Set hostname verifier</li></ul>
AuthPermission	
ServicePermission	Kerberos-related
AWTPermission	<ul style="list-style-type: none"><li>• Clipboard</li><li>• Tray</li><li>• Windows</li><li>• Pointer</li><li>• etc.</li></ul>



# STRUCTURE OF A POLICY FILE (SIMPLIFIED)

```
grant codebase "file:myjar" {  
    permission Foo "foo";  
    permission Bar "bar", "baz";  
};
```



A black laptop is shown from a front-facing perspective, slightly angled. The screen is white and displays the text "Time for DEMO" in a bold, red, sans-serif font. The text is tilted upwards from left to right. The laptop's keyboard and trackpad are visible below the screen.

Time for **DEMO**



# THANKS!

- <https://blog.frankel.ch/>
- @nicolas\_frankel
- <https://git.io/fx54L>
- <https://bit.ly/2DQBRrt>

