Neil Masson – IBM Java L3 Service

24th September 2013

Security in the Real World

Important Disclaimers

- THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.
- WHILST EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.
- ALL PERFORMANCE DATA INCLUDED IN THIS PRESENTATION HAVE BEEN GATHERED IN A CONTROLLED ENVIRONMENT. YOUR OWN TEST RESULTS MAY VARY BASED ON HARDWARE, SOFTWARE OR INFRASTRUCTURE DIFFERENCES.
- ALL DATA INCLUDED IN THIS PRESENTATION ARE MEANT TO BE USED ONLY AS A GUIDE.
- IN ADDITION, THE INFORMATION CONTAINED IN THIS PRESENTATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM, WITHOUT NOTICE.
- IBM AND ITS AFFILIATED COMPANIES SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.
- NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:
- \- CREATING ANY WARRANT OR REPRESENTATION FROM IBM, ITS AFFILIATED COMPANIES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS

About me

- Neil Masson
-
- Employed by IBM
- Worked on Java since the year dot (2...)
- In Core team dealing with all kinds of customer issues

What should you get from this talk?

- An understanding of the most common attack vectors used to target Java.
-
- An insight into the details of some example vulnerabilities.
-
- An understanding of the current importance of security in the context of the Java platform.
-
- Some ideas of how you can write more secure code.

The problem with keeping anything secure

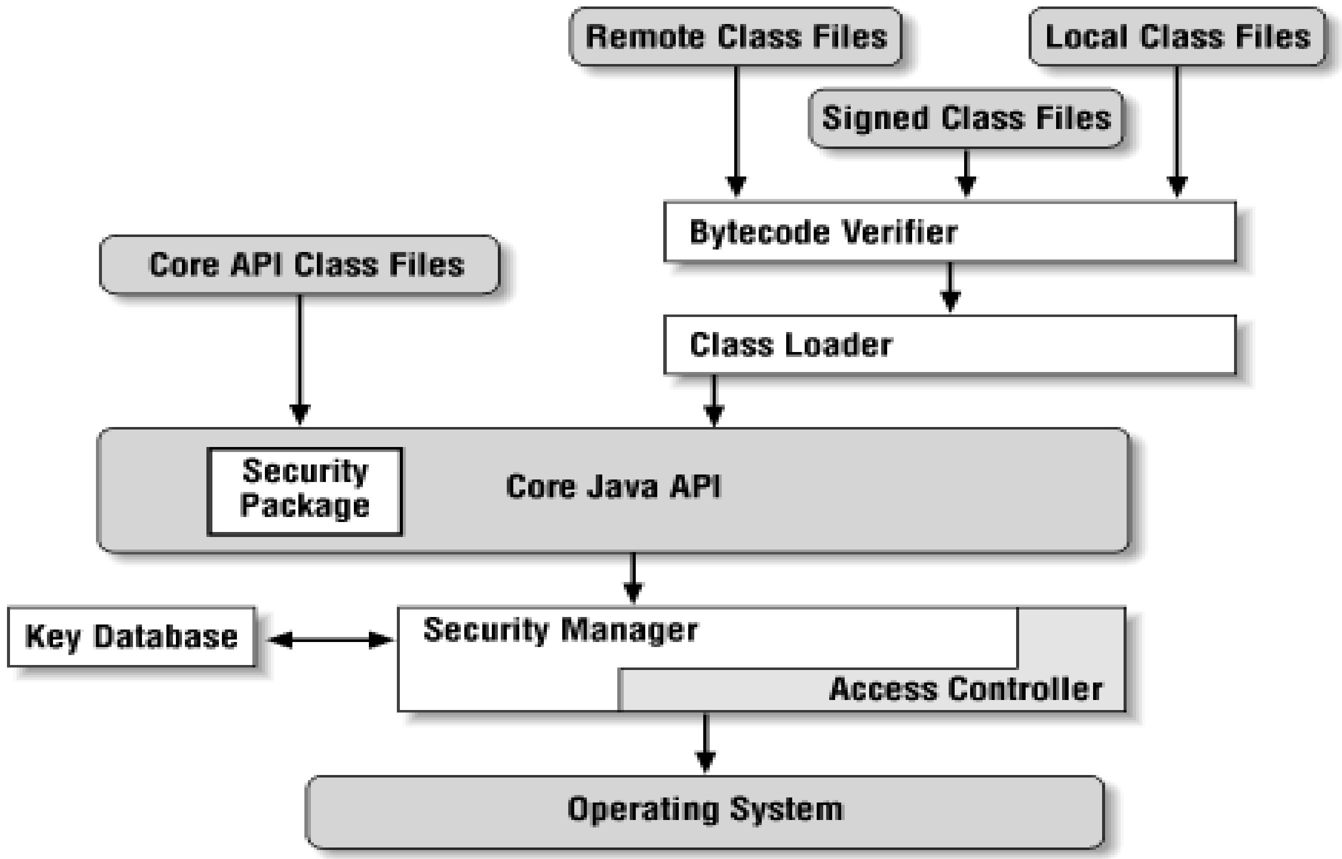
- "The only secure computer is one that's unplugged, locked in a safe, and buried 20 feet under the ground in a secret location... and I'm not even too sure about that one"
  -- (attributed) Dennis Huges, FBI.
-
- A complex system will have many attack vectors
-
- Systems need to balance stability, performance and security

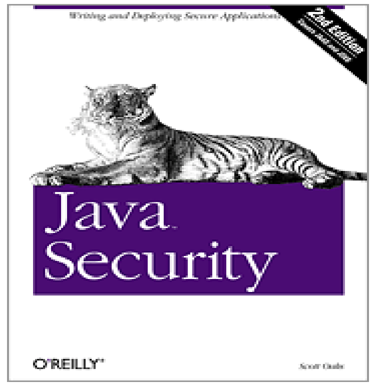
Security that doesn't interfere

- Java and the Java Virtual Machine provide defense in depth
  - Class loaders
  - Verification
  - Access Controller / Security Manager
  - Java Cryptography Extensions (JCE)
  - Java Secure Sockets Extension (JSSE)
  - Java Authentication and Authorization Service (JAAS)
- Available implicitly or explicitly during development / deployment

- Security is expected to be a trusted resource
  - It just works
  - It has been verified (thoroughly) by vendors
- Be aware of what isn't secured!
- Key: Java should negate the need to build (and verify!) your own security layers

Security Layers in Java





- Some things you get “for free”
- Others you use when you ask for them

Diagram Reference: Java Security, Scott Oaks, O’Reilly Media, May 24, 2001, Second Edition, ISBN-10: 0596001576, ISBN-13: 978-0596001575

Common attack vectors

- The most common attack vectors in the context of Java fall into four categories:
  - Untrusted Data
  - Untrusted Code
  - Applet / Browser
  - Local
-
- Through the rest of the talk we will look at each in a little more detail

Attack Vectors: Untrusted Data

- The untrusted data vector is exploitable when an application parses a specific data type from an untrusted source.
  - The vulnerability may exist in the application code or underlying JDK code.
-
- For example,
  - You are using an image parsing module that contains a vulnerability that can lead to an infinite loop when the image file is crafted in a specific manner.
  - If your server application allows users to upload images for parsing an attacker could create a Denial of Service attack by uploading maliciously formed images.
-
- This type of problem has widely varying consequences, from DOS attacks through to arbitrary code execution

Double.parseDouble Denial-of-Service Attack (**CVE-2010-4476**)

Double.parseDouble

Old but special

- The Alert was officially published in Febuary 2011
-
- Is a very worthy inclusion because:
  - It was the first alert to cause real panic
  - Easily exploitable & Pervasive

Double.parseDouble

The problem

- A bug had been present in Double.parseDouble since early versions of the JDK.
-
- Passing "2.2250738585072012e-308" to the method causes an infinite loop.
-
- The catalyst was a determination of just how easy it was to exploit
  - Populating header fields in webserver requests with the value resulted in a DOS attack.
  -
-

Double.parseDouble

The result

- A fix was made very quickly
-
- The ease of exploit and pervasiveness of the vulnerability resulted in a huge exercise to update products distributing the JDK.
-
- Fixes were provided right back to 1.3.1 and on obscure platforms like OS/2.
-

Hashing Denial-of-Service Attack (**CVE-2011-4858**)

Hashing Denial-of-Service Attack

- String hash codes and hashing structures have been around "for ever"
-
- The attack is possible through a combination of:
  - Performance short comings
  - Documented / predictable behavior
-
- Can be used to exploit vulnerabilities in existing software
-
- Algorithmic Complexity Attack

Hashing Denial-of-Service Attack

How String Hashing Works

- String hashing algorithm is well known and reversible

hashCode

```
public int hashCode()
```

Returns a hash code for this string. The hash code for a `String` object is computed as

```
s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
```

- It is easy to construct strings that have identical hash codes

```
"Aa".hashCode()
"BB".hashCode()   == 2112

"AaAa".hashCode()
"AaBB".hashCode()
"BBAa".hashCode()
"BBBB".hashCode()   == 2031744
```

Hashing Denial-of-Service Attack

How Hashing Structures Work

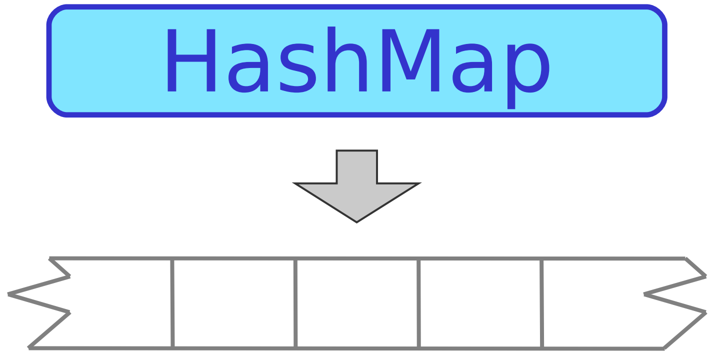
```
myHashMap.put("QuantityAa", "1234");
```

HashMap

Hashing Denial-of-Service Attack

How Hashing Structures Work

```
myHashMap.put("QuantityAa", "1234");
```



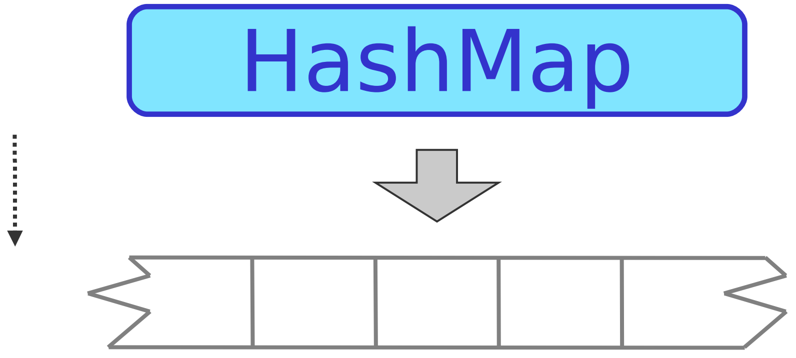
Array to hold the various <key,value> pairs

Hashing Denial-of-Service Attack

How Hashing Structures Work

```
myHashMap.put("QuantityAa", "1234");
```

Use the hash code for “QuantityAa” to find a location in the array

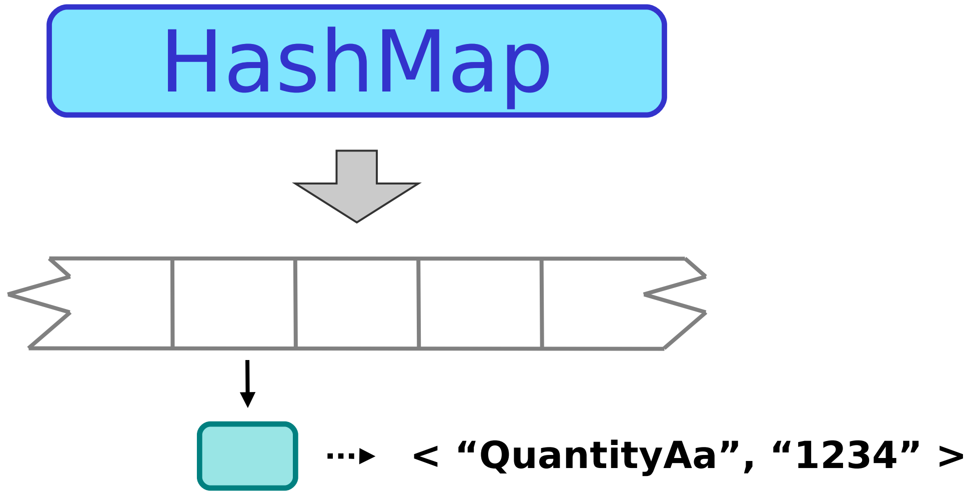


Array to hold the various <key,value> pairs

Hashing Denial-of-Service Attack

How Hashing Structures Work

```
myHashMap.put("QuantityAa", "1234");
```

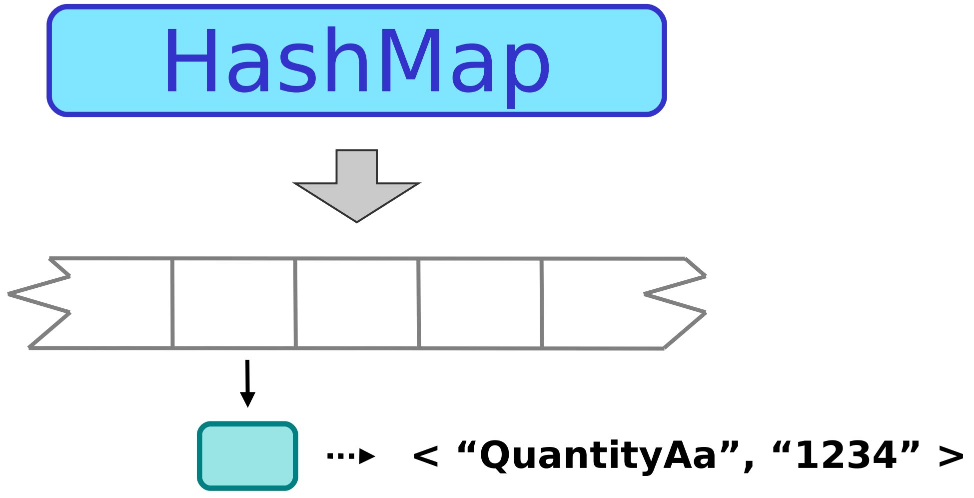


Find the appropriate “bucket” and add the entry

Hashing Denial-of-Service Attack

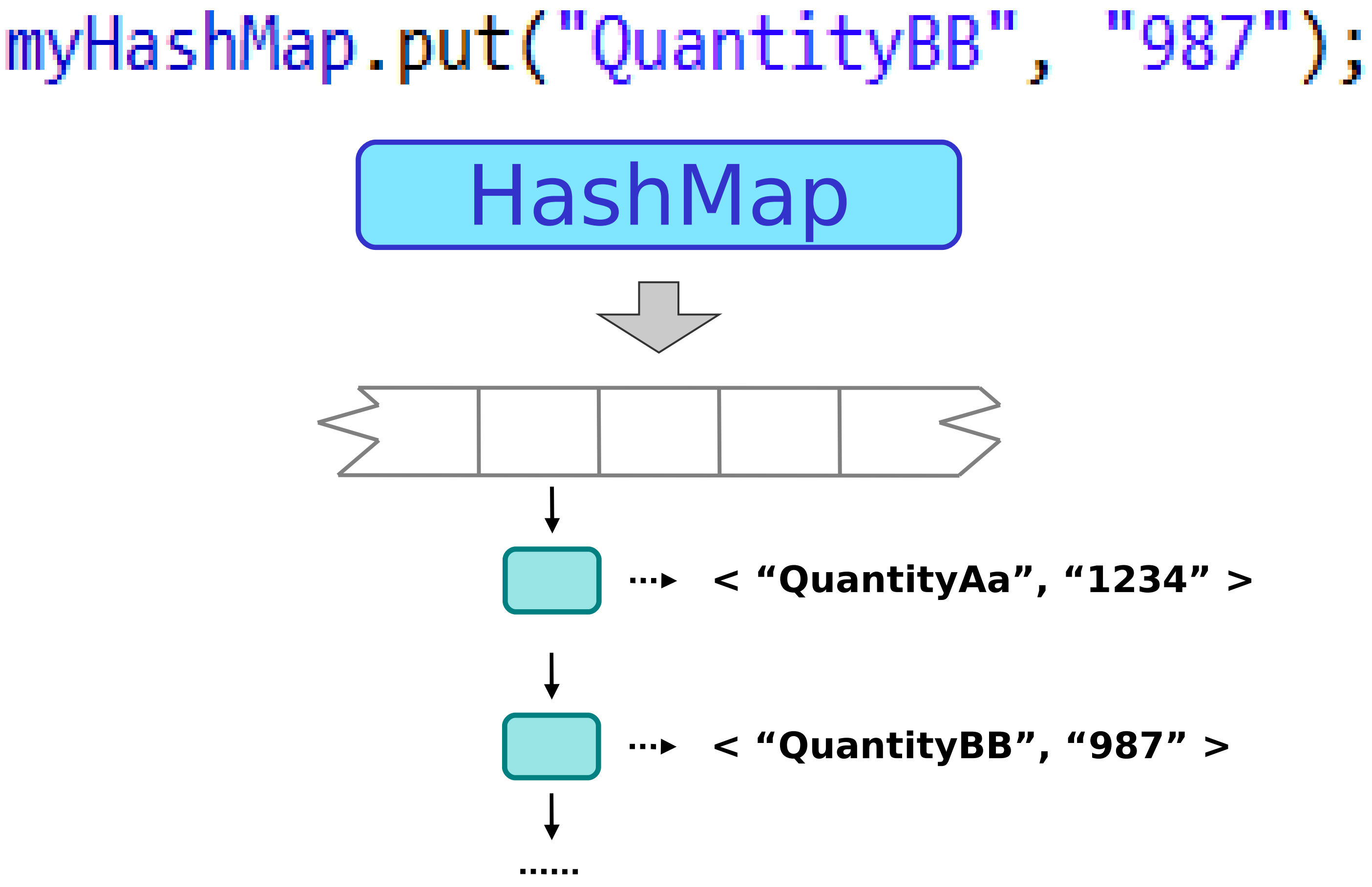
How Hashing Structures Work

```
myHashMap.put("QuantityBB", "987");
```



Hashing Denial-of-Service Attack

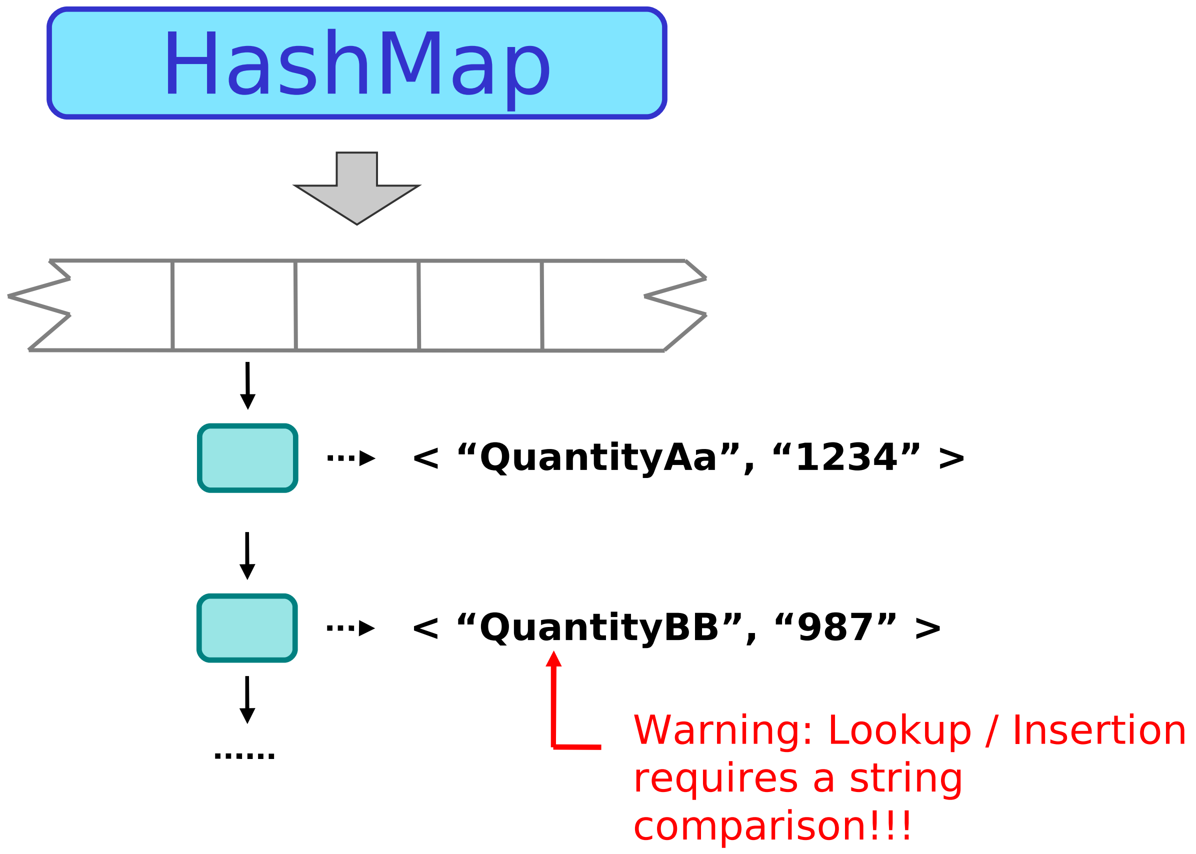
How Hashing Structures Work



Hashing Denial-of-Service Attack

How Hashing Structures Work

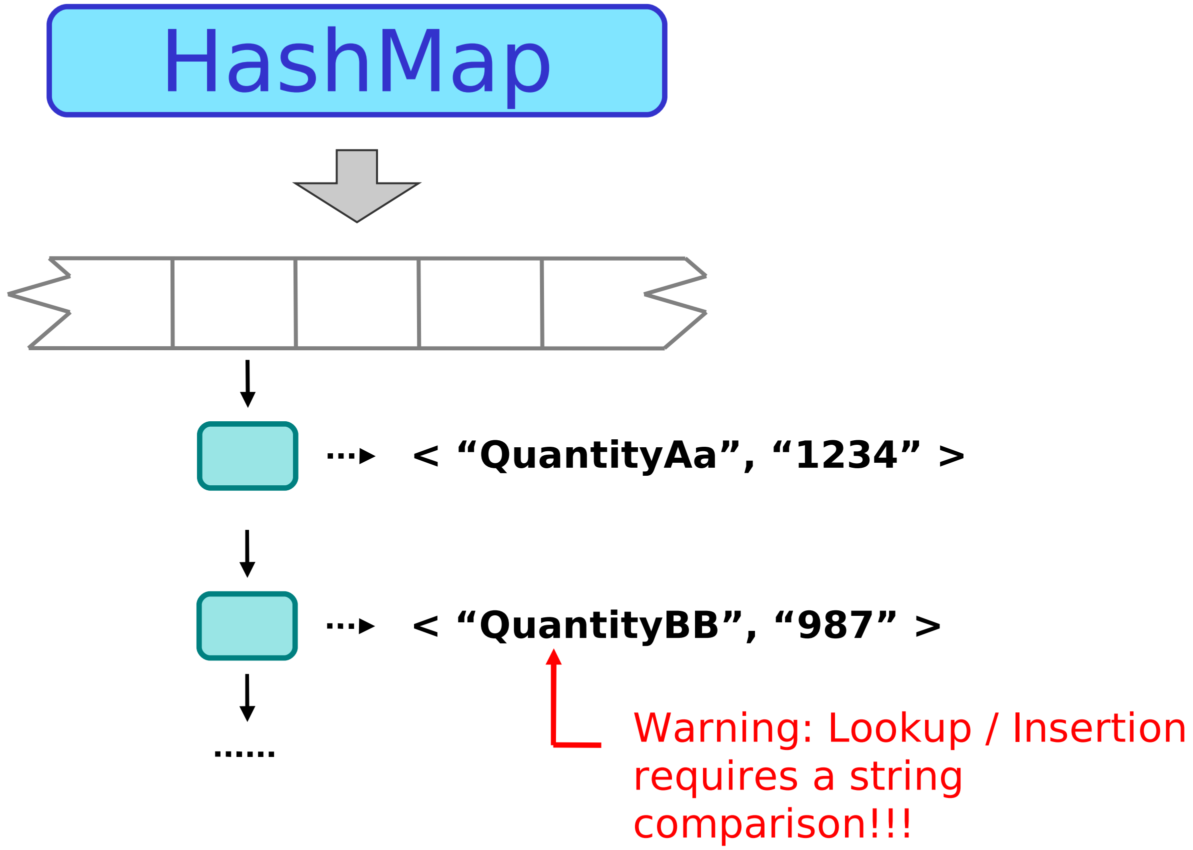
```
myHashMap.put("QuantityBB", "987");
```



Hashing Denial-of-Service Attack

How Hashing Structures Work

```
myHashMap.put("QuantityBB", "987");
```

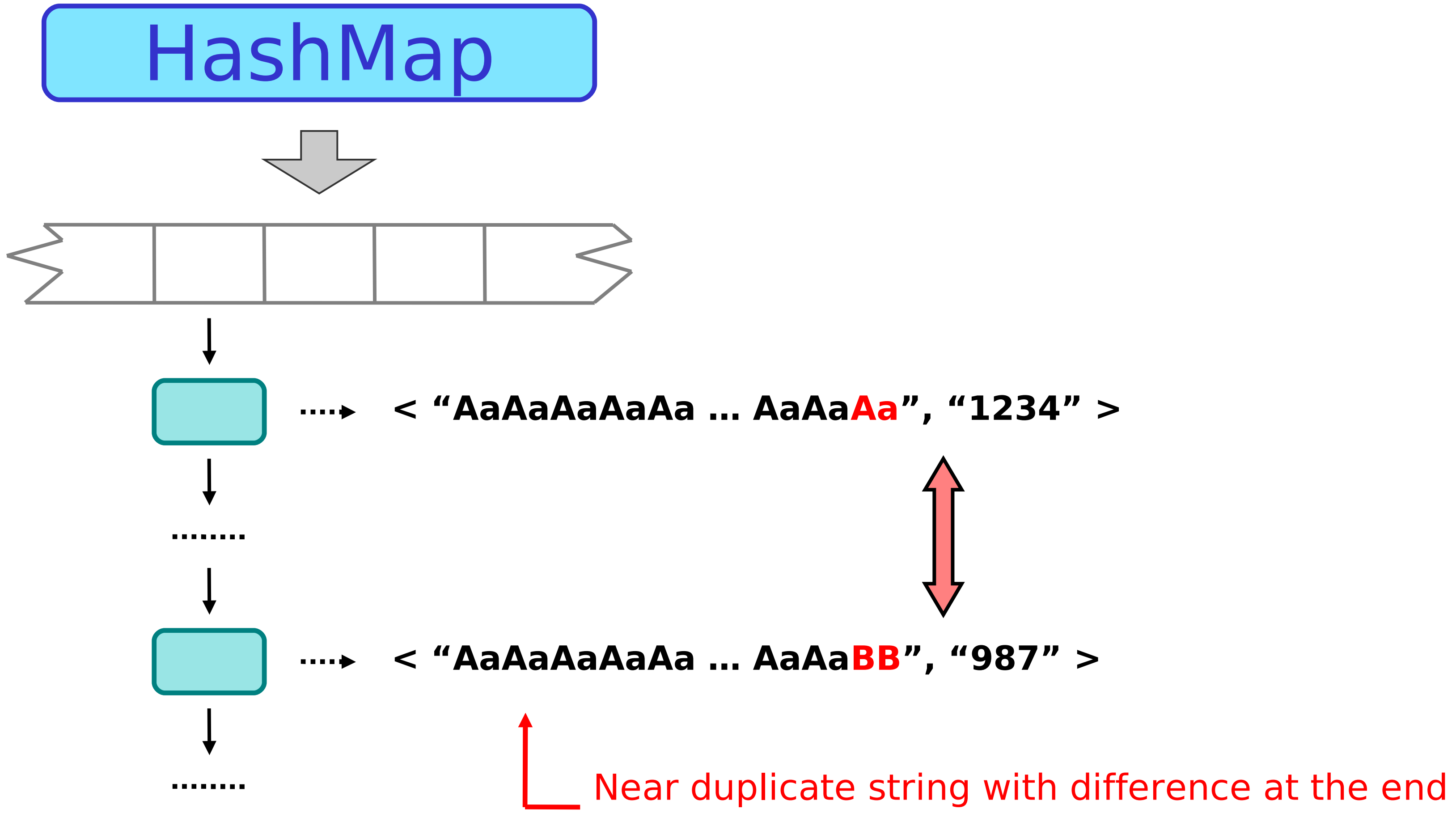


- Keys with identical hashes will always fall into the same bucket

Hashing Denial-of-Service Attack

The Danger of strings as Keys in Hashing Structures

- Deep buckets with malicious keys can cause serious performance issues



Hashing Denial-of-Service Attack

The primary exploit

- Websites make use of parameters as part of client / server communication

- The Server is responsible for managing the parameters for the servlet
- Hash structures are a typical way of managing these <key,value> pairs
- Issue: Long insert / lookup times for parameters that have high hash collision rate

> A Tomcat 6.0.32 server parses a 2 MB string of colliding keys in about 44 minutes of i7 CPU time, so an attacker with about 6 kbit/s can keep one i7 core constantly busy. If the attacker has a Gigabit connection, he can keep about 100.000 i7 cores busy.

Reference: http://www.nruns.com/_downloads/advisory28122011.pdf

- Result: Web servers could be effectively “disabled” with simple requests

Hashing Denial-of-Service Attack

Current Solution

- Hashing structures now use an alternate hash code for String
  - Use alternate only at a certain capacity
  - Algorithm where the hash code cannot be calculated externally
-
- Why not modify String.hashCode()?
  - It's spec!
  - Reliance in existing software
-
- NOTE: With alternate hash, iteration order is now changed!
  - Spec'd as "unspecified"
  - Doesn't matter – code relies on this any way
  - Solution can cause existing working software to fail!

Hashing Denial-of-Service Attack

Current Solution

- The JVM now supports a system property to enable the behavior at thresholds:
  -

-Djdk.map.althashing.threshold=<threshold>

-
- Apache Tomcat property maxParameterCount to limit number of parameters

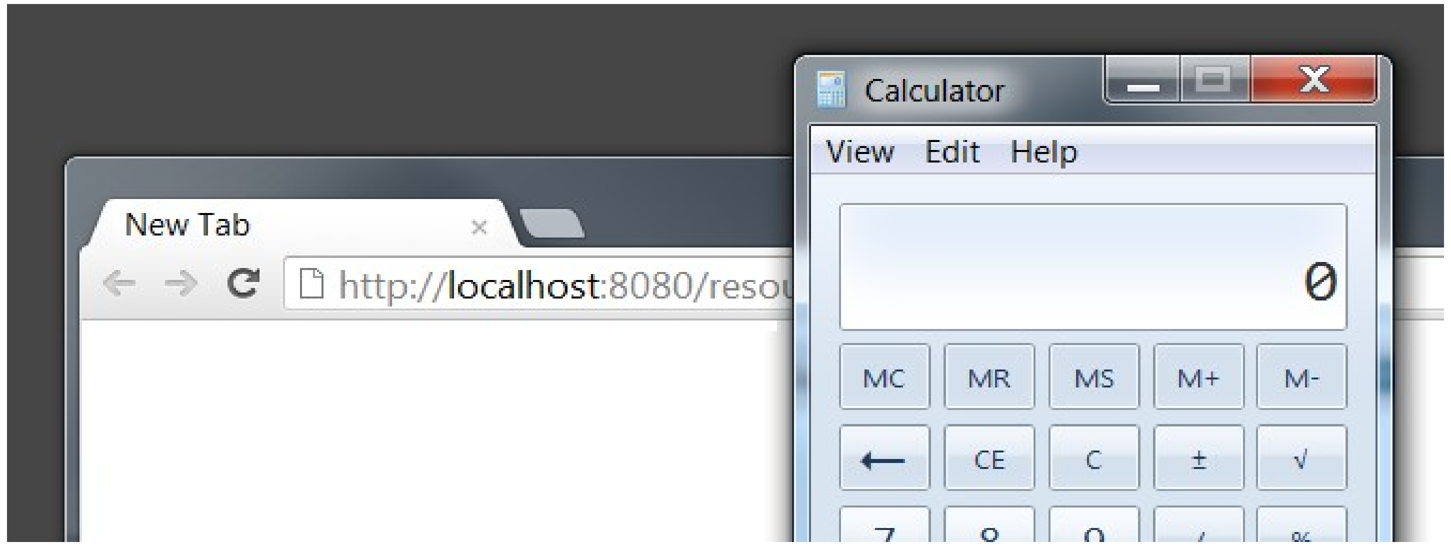
Attack Vectors: Untrusted Code

- Untrusted code originates from an unknown or untrusted source
  - It is not under the application environments control
  - It is not know to be benign.
  - It should be treated with caution
-
- Typically executed via an unsigned applet or webstart application
  - Browser based client side exploit
  - The JDK security sandbox offers protection
-
- The attack vector works due to vulnerabilities that allow the untrusted code to escape the confines of the sandbox, sometimes disabling it completely.
  - Allows the code to do whatever it likes.
-
- While most common on the client side the vulnerability applies equally to any environment where code executes under a security manager.

Gondvv Vulnerability (**CVE-2012-4681**)

Java Security Manager Bypass (Gondvv)

- Imagine visiting a website and your calculator application pops up
-
-
-
- How did that happen?
-
- Arbitrary code has been run on your machine – how compromised are you?



Java Security Manager Bypass (Gondvv)

The key change to sun.awt.SunToolkit

- A simple access modifier change (within a larger change) exposed a vulnerability

```
private static Field getField(Class klass, String fieldName)
```

Java Security Manager Bypass (Gondvv)

The key change to sun.awt.SunToolkit

- A simple access modifier change (within a larger change) exposed a vulnerability

```
private static Field getField(Class klass, String fieldName)
```

~~private~~ → public

Java Security Manager Bypass (Gondvv)

The key change to sun.awt.SunToolkit

- A simple access modifier change (within a larger change) exposed a vulnerability

```
private static Field getField(Class klass, String fieldName)

public
{
    Field myField = null;
    doPrivileged() {
        // myField = get reflect field "fieldname" on klass
        // set accessible(true) on myField
    }
    return myField;
}
```

Java Security Manager Bypass (Gondvv)

The key change to sun.awt.SunToolkit

- A simple access modifier change (within a larger change) exposed a vulnerability

~~private~~ → public

```
private static Field getField(Class klass, String fieldName)

public
{
    Field myField = null;
    doPrivileged() {
        // myField = get reflect field "fieldname" on klass
        // set accessible(true) on myField
    }
    return myField;
}
```

Set the security permissions to that of the current code (privileged) in place of the callers security permissions

Java Security Manager Bypass (Gondvv)

The key change to sun.awt.SunToolkit

- A simple access modifier change (within a larger change) exposed a vulnerability

```
private static Field getField(Class klass, String fieldName)

public
{
    Field myField = null;
    doPrivileged() {
        // myField = get reflect field "fieldname" on klass
        // set accessible(true) on myField
    }
    return myField;
}
```

Use reflection to acquire a Field object on the given class

Java Security Manager Bypass (Gondvv)

The key change to sun.awt.SunToolkit

- A simple access modifier change (within a larger change) exposed a vulnerability

~~private~~ → public

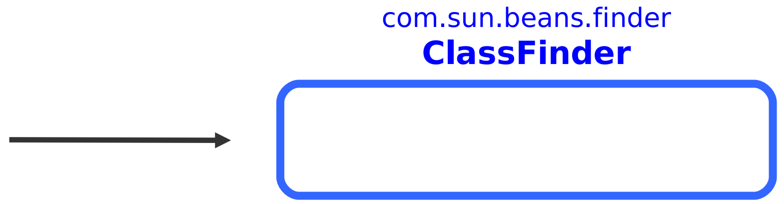
```
private static Field getField(Class klass, String fieldName)

public
{
    Field myField = null;
    doPrivileged() {
        // myField = get reflect field "fieldname" on klass
        // set accessible(true) on myField
    }
    return myField;
}
```

Set the reflect object Field usage to ignore access checks. Privileged action permitted through *doPrivileged()*

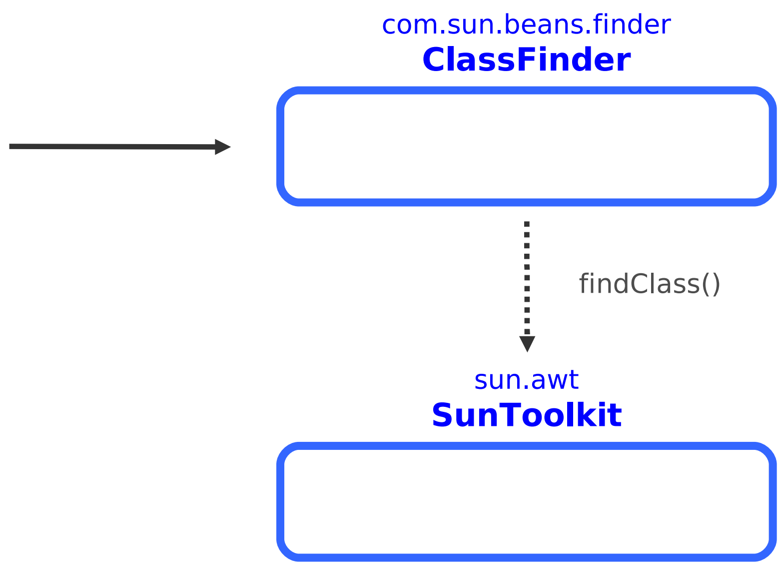
Java Security Manager Bypass (Gondvv)

How the exploit works



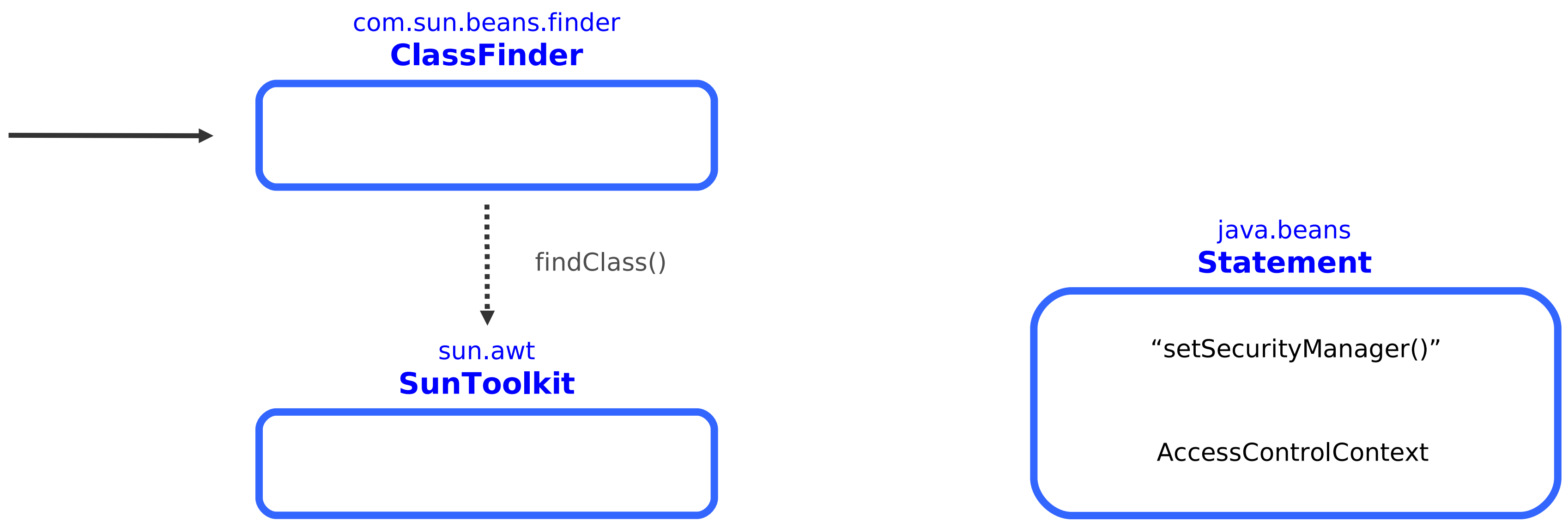
Java Security Manager Bypass (Gondvv)

How the exploit works



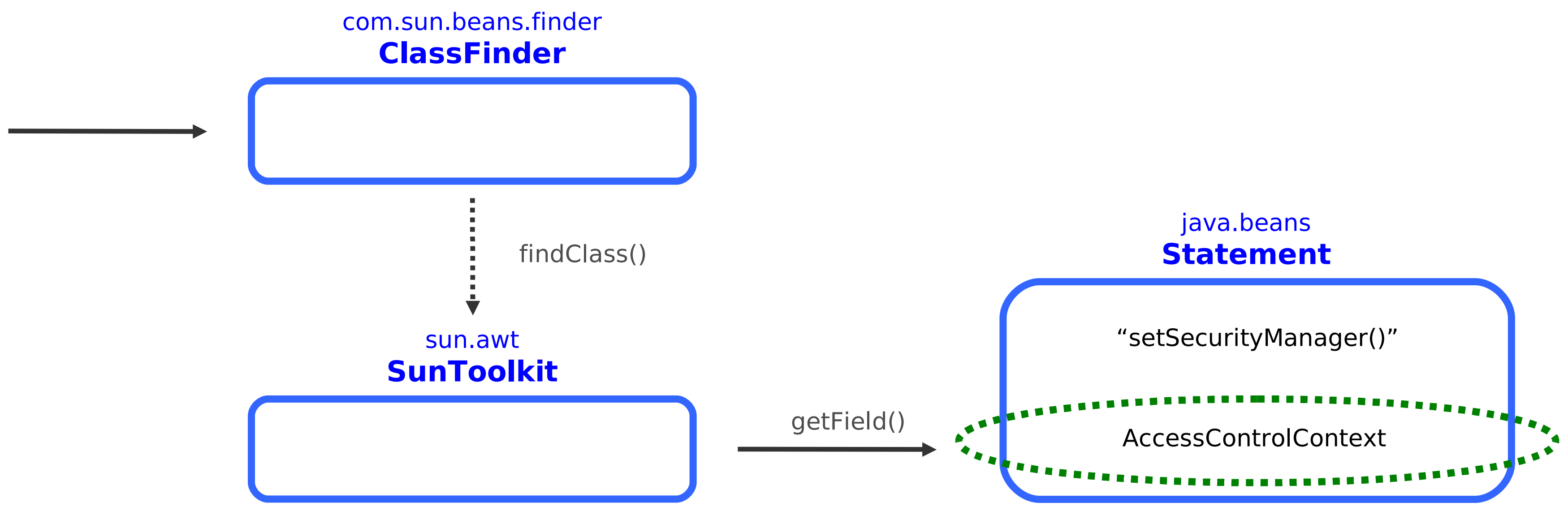
Java Security Manager Bypass (Gondvv)

How the exploit works



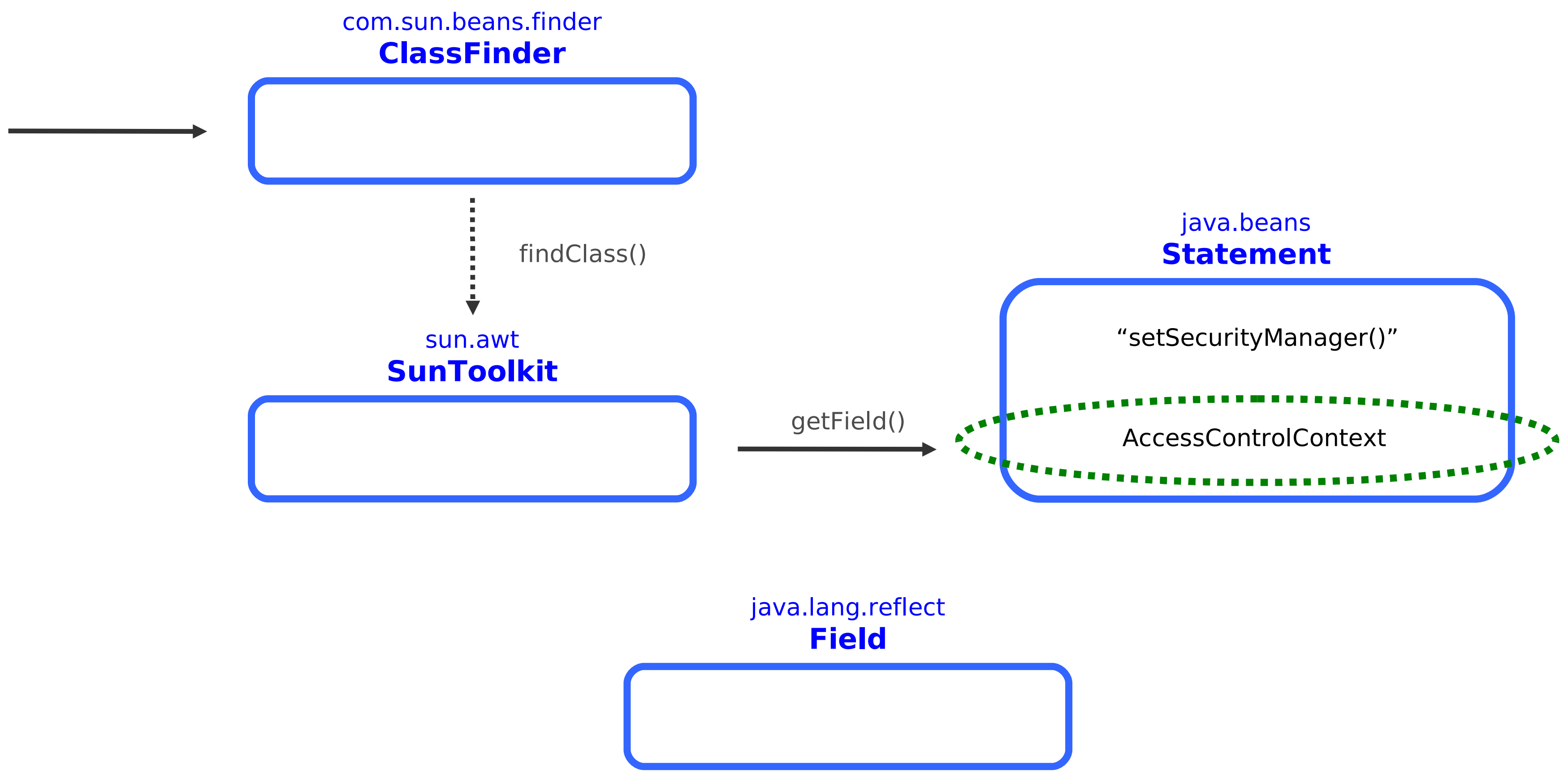
Java Security Manager Bypass (Gondvv)

How the exploit works



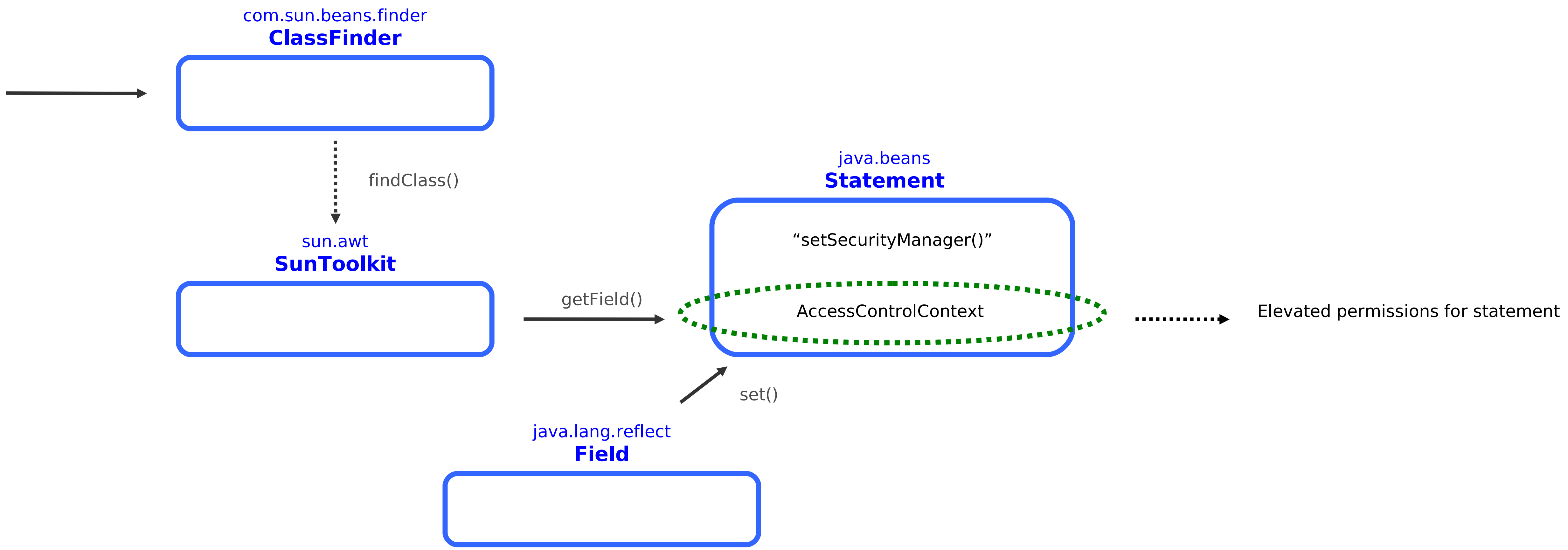
Java Security Manager Bypass (Gondvv)

How the exploit works



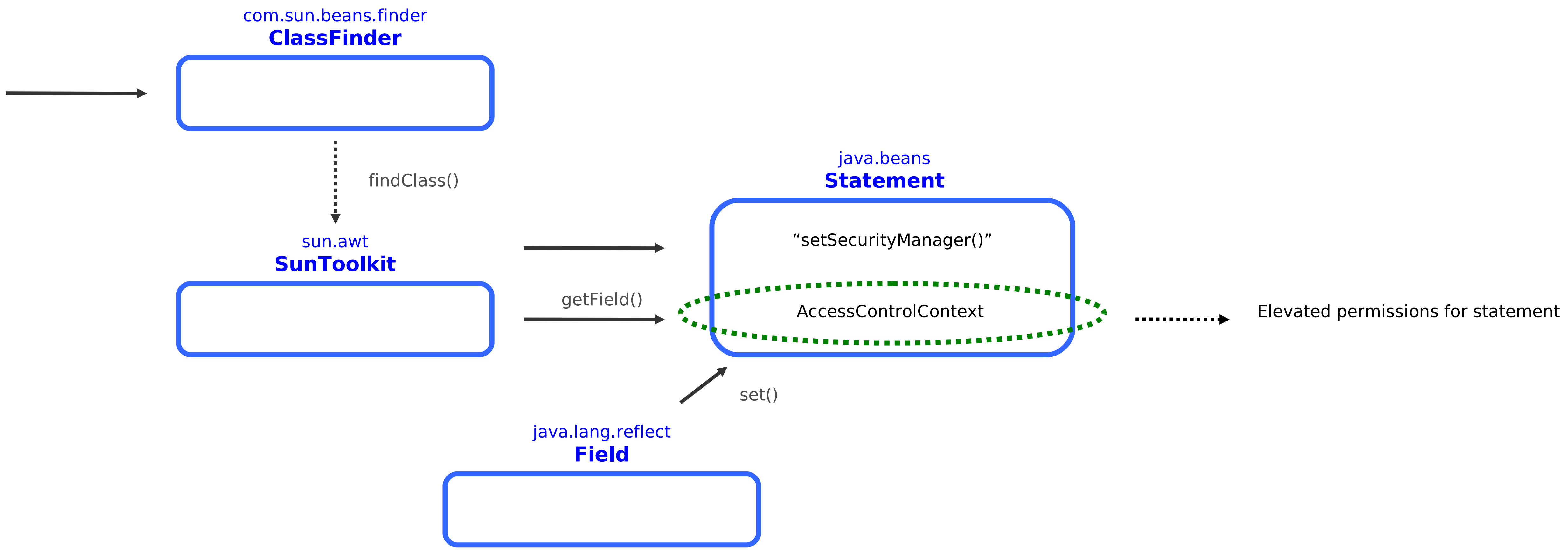
Java Security Manager Bypass (Gondvv)

How the exploit works



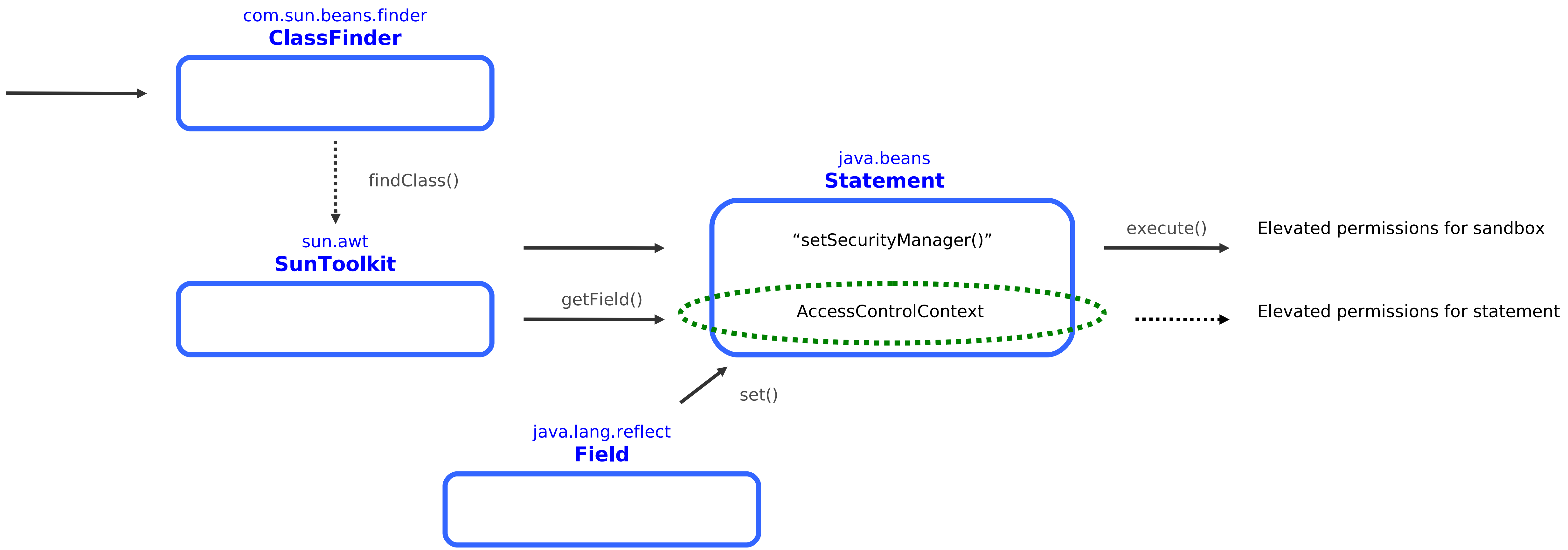
Java Security Manager Bypass (Gondvv)

How the exploit works



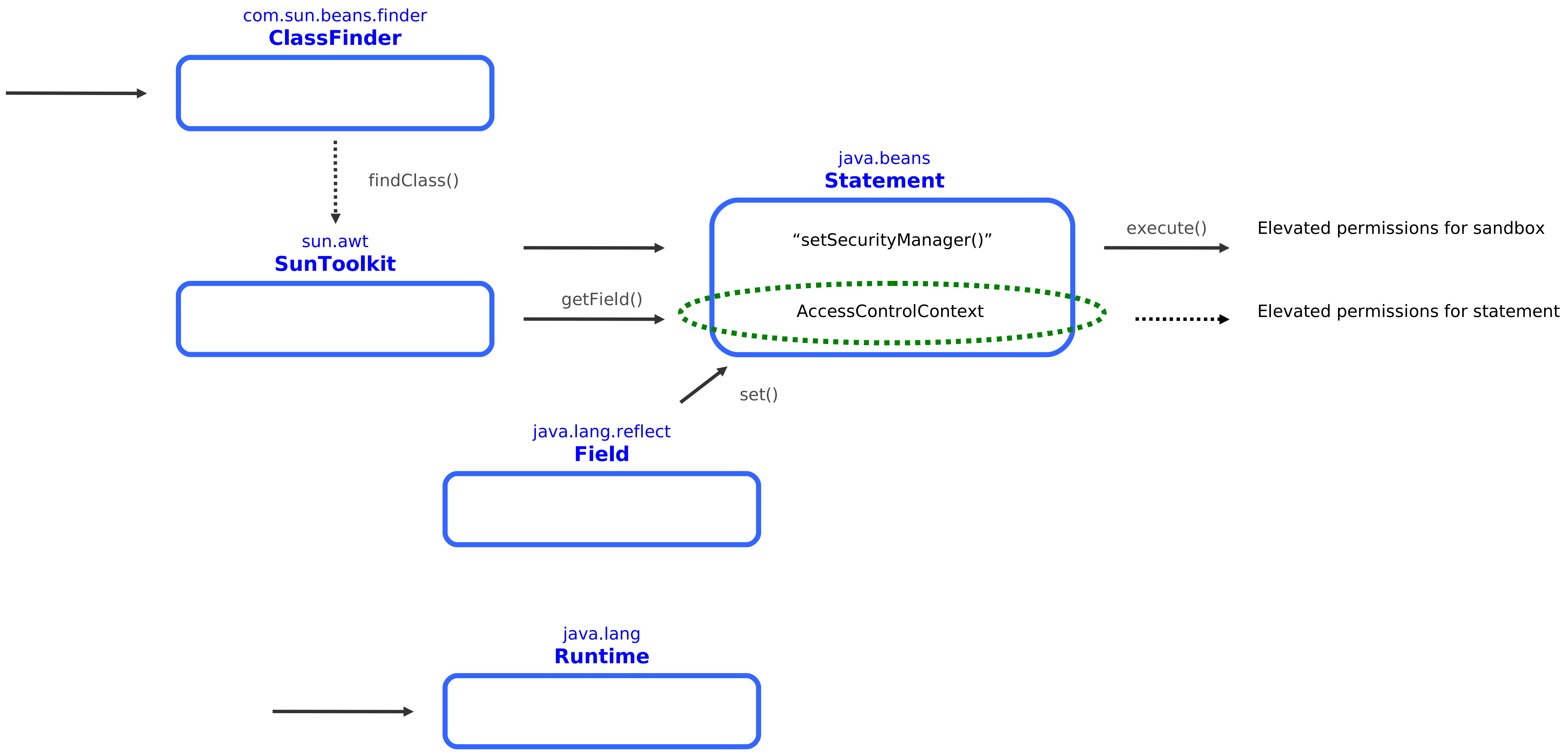
Java Security Manager Bypass (Gondvv)

How the exploit works



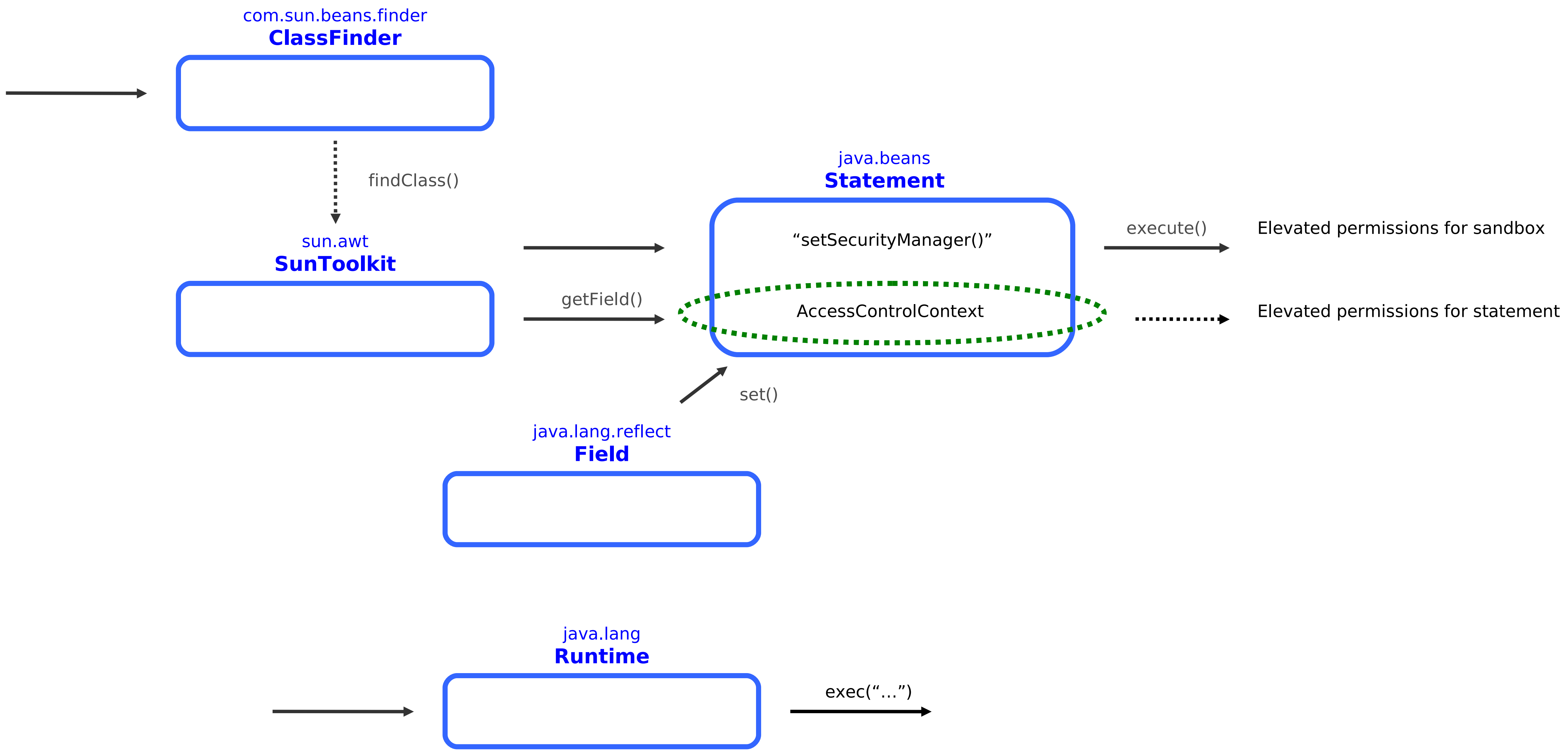
Java Security Manager Bypass (Gondvv)

How the exploit works



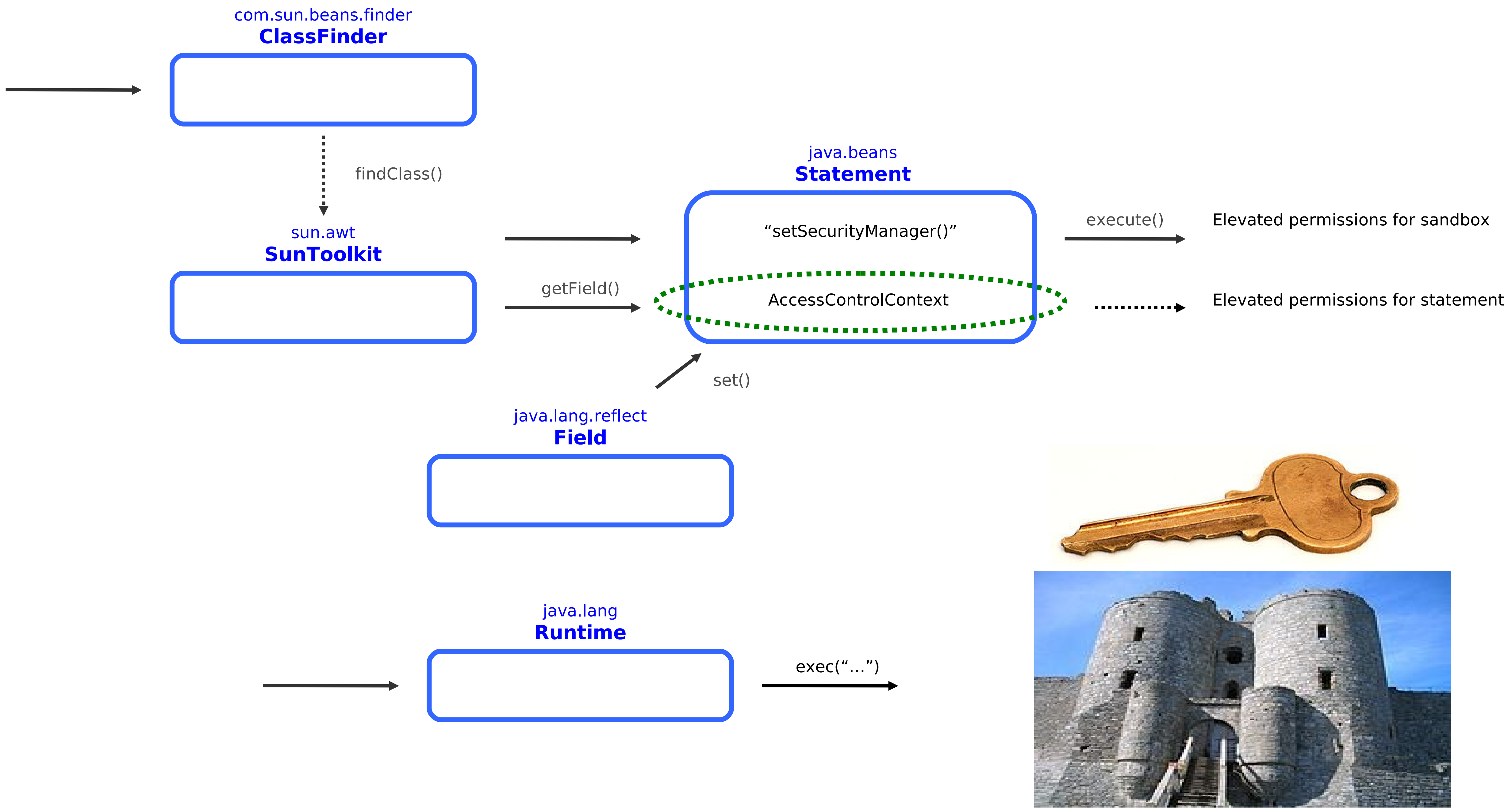
Java Security Manager Bypass (Gondvv)

How the exploit works



Java Security Manager Bypass (Gondvv)

How the exploit works



Java Security Manager Bypass (Gondvv)

Epilogue

-
- Needed to be running untrusted code
-
- Java7 VM required
  - Most users were still at 6.0
-
- A simple change to an access modifier exposed the entire system
-
- NOTE: A fix was turned around in very short order

Method Handles

Method Handles

- JSR 292: Supporting Dynamically Typed Languages on the JavaTM Platform
  - A new bytecode for custom dynamic linkage (invokedynamic)
  - MethodHandle (and support classes) as a “function pointer” interface for linkage

- Fast invocation of bound methods
  - Method handle invocation speed can be far superior to reflect methods
  -

- A MethodHandle resembles java.lang.reflect.Method
  - Access checking is performed at lookup, not at every call
  - Conversion available from reflection side to MethodHandle types

Method Handles

Access and Security Checks

	Reflection	**MethodHandles**
SecurityManager checks at lookup	Yes	Yes
Access checks at lookup	**No**	**Yes**
Access checks at invocation	**Yes**	**No**
Checks at setAccessible(true)	Yes	N/A
Anyone can invoke?	No: by default Yes: setAccessible(true)	Yes – by default

Method Handles

Security Where It Matters

```
Method m = String.class.getDeclaredMethod("length");
int i = (Integer)m.invoke("hello");
int j = (Integer)m.invoke("there");
int k = (Integer)m.invoke("everyone");
```

```
MethodHandle mh = MethodHandles.lookup().
        findVirtual(
            String.class,
            "length",
            MethodType.methodType(int.class));
int i = (int)mh.invokeExact("hello");
int j = (int)mh.invokeExact("there");
int k = (int)mh.invokeExact("everyone");
```

Method Handles

A Word of Caution

- The lookup mechanism has interesting privilege characteristics
  - Be careful about what code has access to it

lookup

```
public static MethodHandles.Lookup lookup()
```

Returns a `lookup object` on the caller, which has the capability to access any method handle that the caller has access to, including direct method handles to private fields and methods. This lookup object is a *capability* which may be delegated to trusted agents. Do not store it in place where untrusted code can access it.

"New Year Day" / "EveryDay" (**CVE-2013-0422**)

"New Year Day" / "EveryDay"

A combination of exploits

- Client side applet based attack
-
- A combination of two vulnerabilities
  - The ability to access privileged classes via JMX
  - A reflection issue in MethodHandles that prevented correct access checks
  -
- Easy to exploit
-
-

"New Year Day" / "EveryDay"

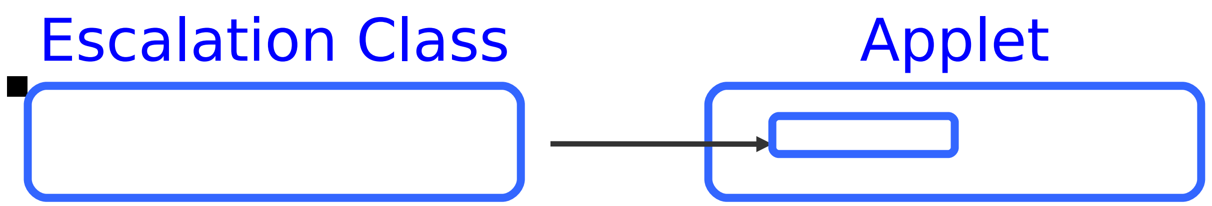
How it worked

-
- Escalation Class

- The escalation class implements privileged action
- The action sets the SecurityManager to Null

"New Year Day" / "EveryDay"

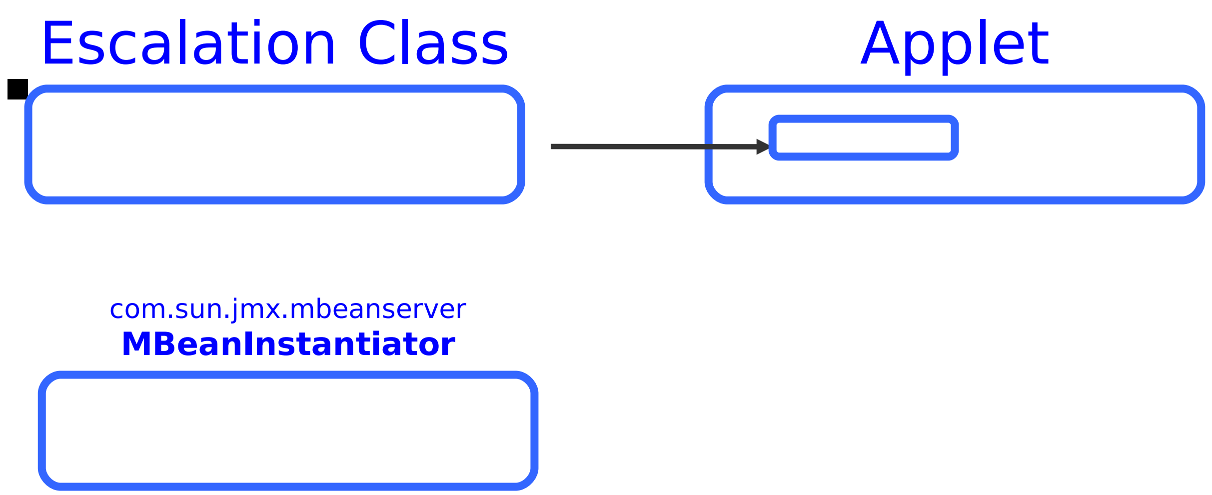
How it worked



- The bytes are stored in an array in the applet.

"New Year Day" / "EveryDay"

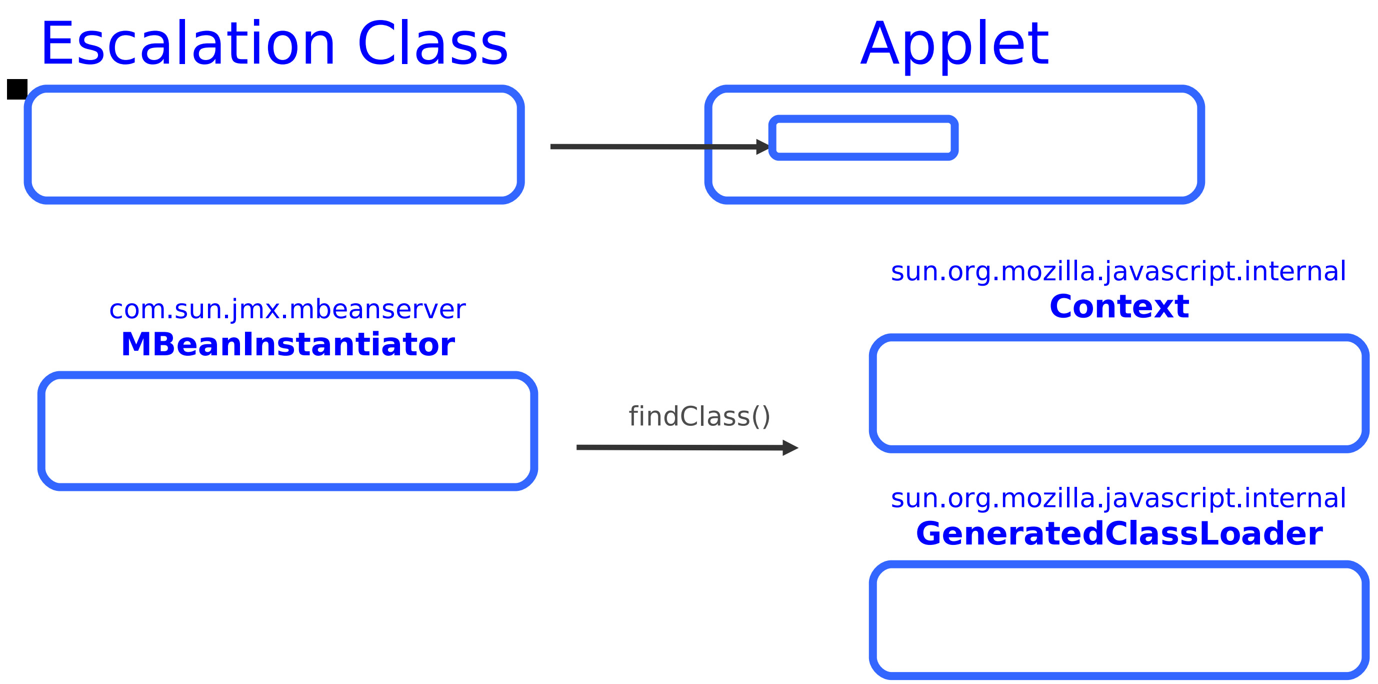
How it worked



- Next we aquire an instance of MbeanInstantiator via the public API JmxMBeanServer.getMBeanInstantiator().

"New Year Day" / "EveryDay"

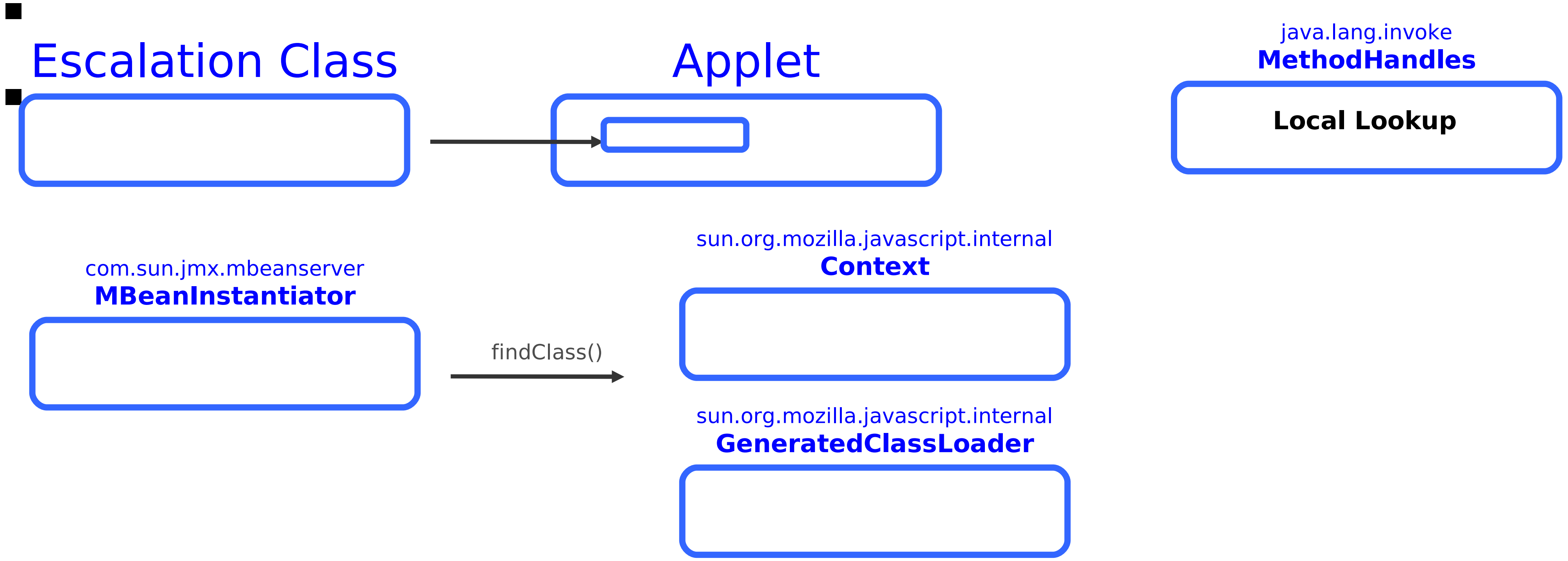
How it worked



- The findClass method is used to obtain two private classes

"New Year Day" / "EveryDay"

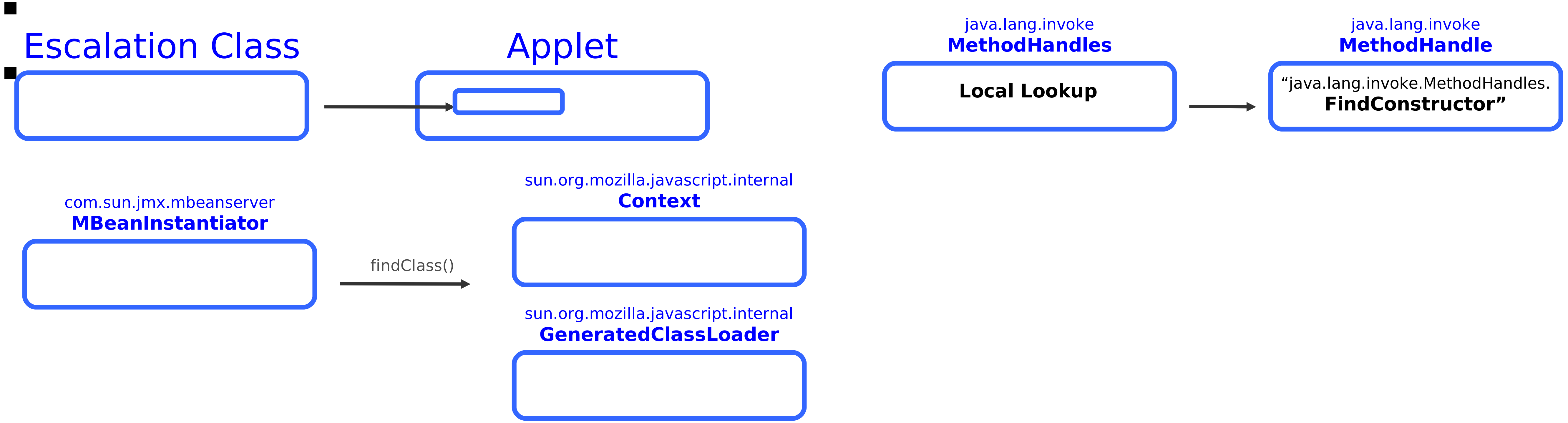
How it worked



- A local instance of MethodHandles.lookup is created

"New Year Day" / "EveryDay"

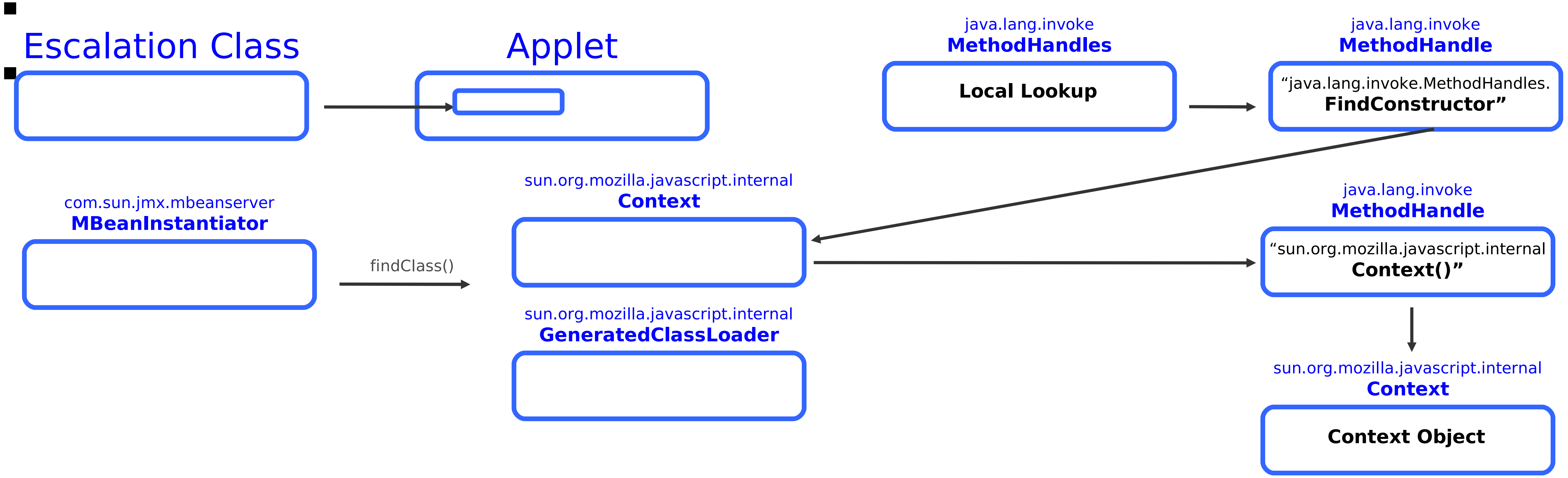
How it worked



- This is then used to create a method Handle to the findConstructor method in the MethodHandles class

"New Year Day" / "EveryDay"

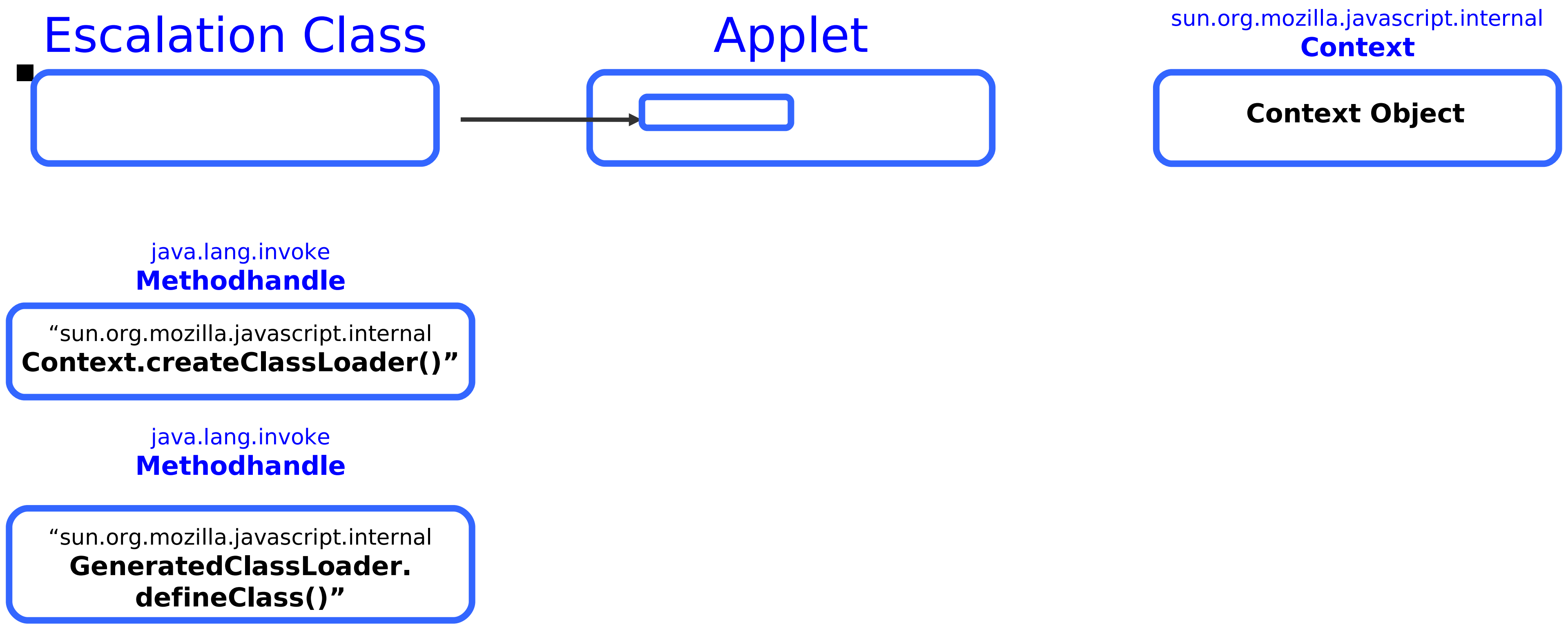
How it worked



- When invoked against our Context class we obtain a new methodHandle that allows us to create an instance of Context

"New Year Day" / "EveryDay"

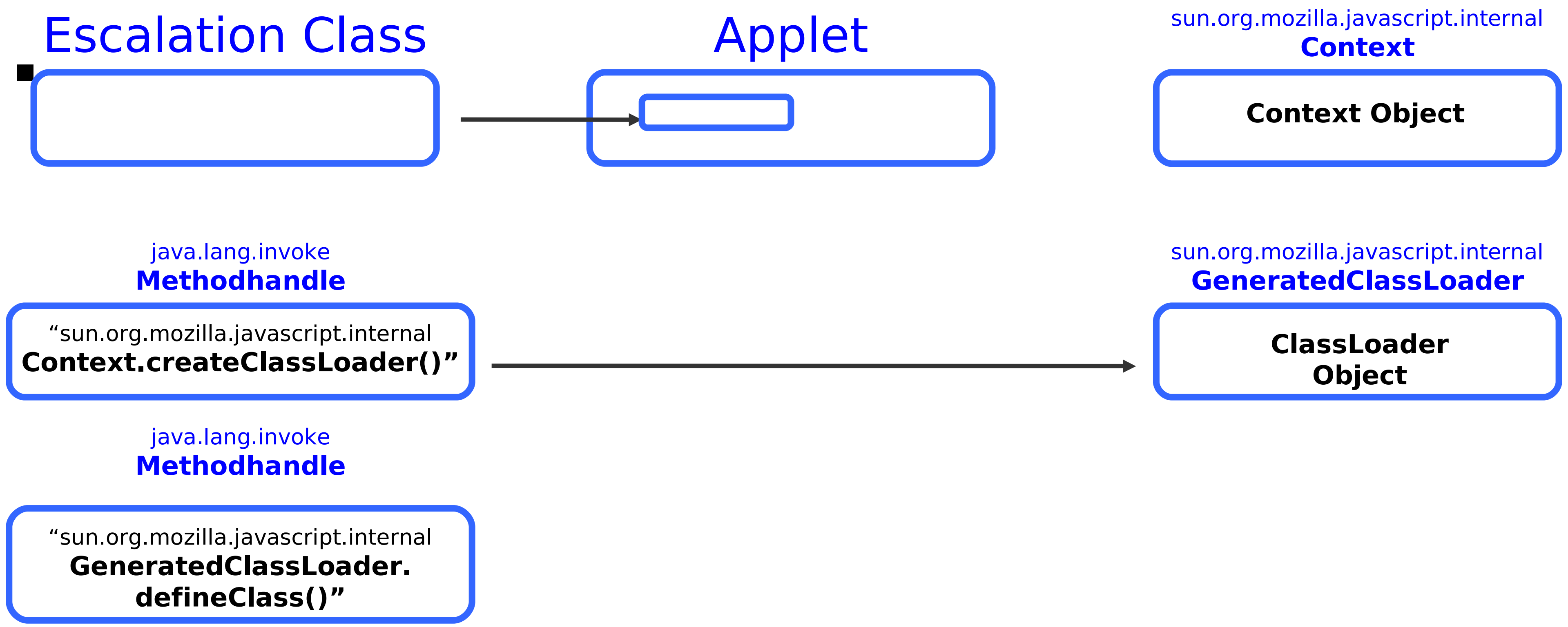
How it worked



- The same technique is used to create methodHandles for Context.createClassloader and GeneratedClassLoader.defineClass

"New Year Day" / "EveryDay"

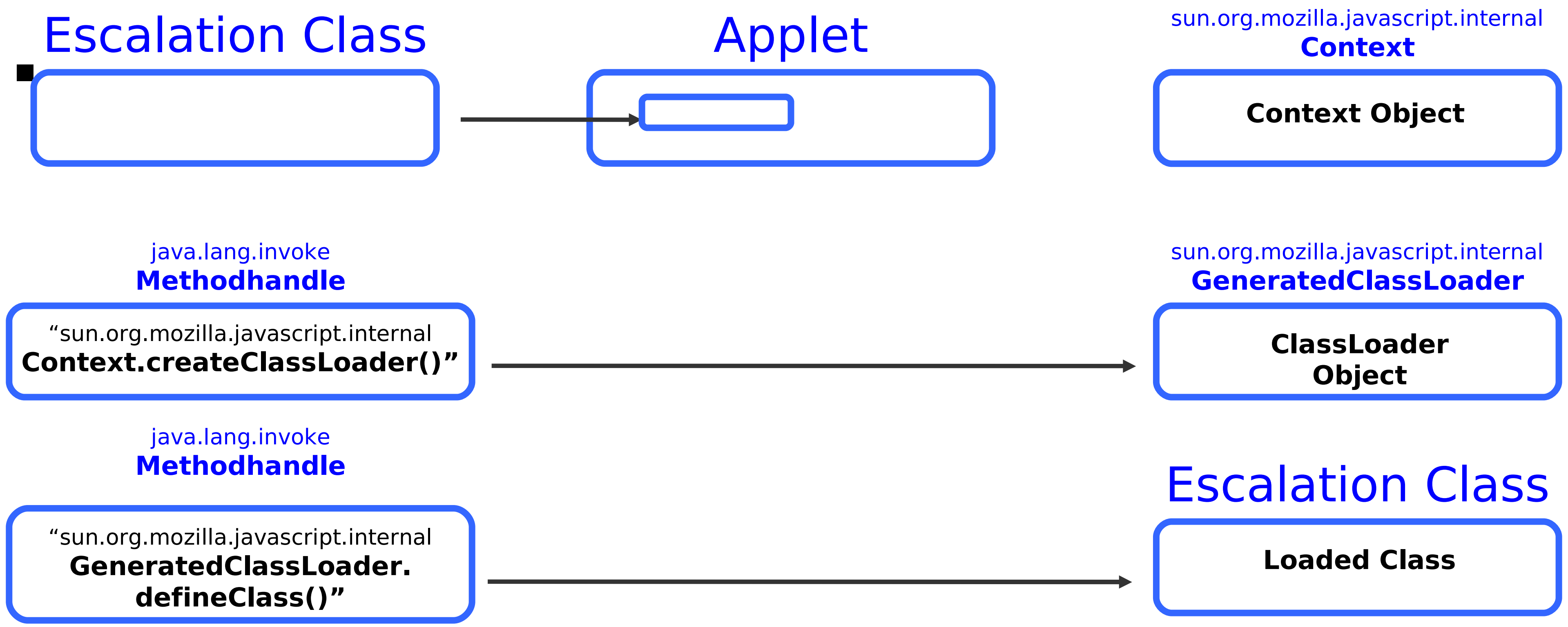
How it worked



- A classloader object is created

"New Year Day" / "EveryDay"

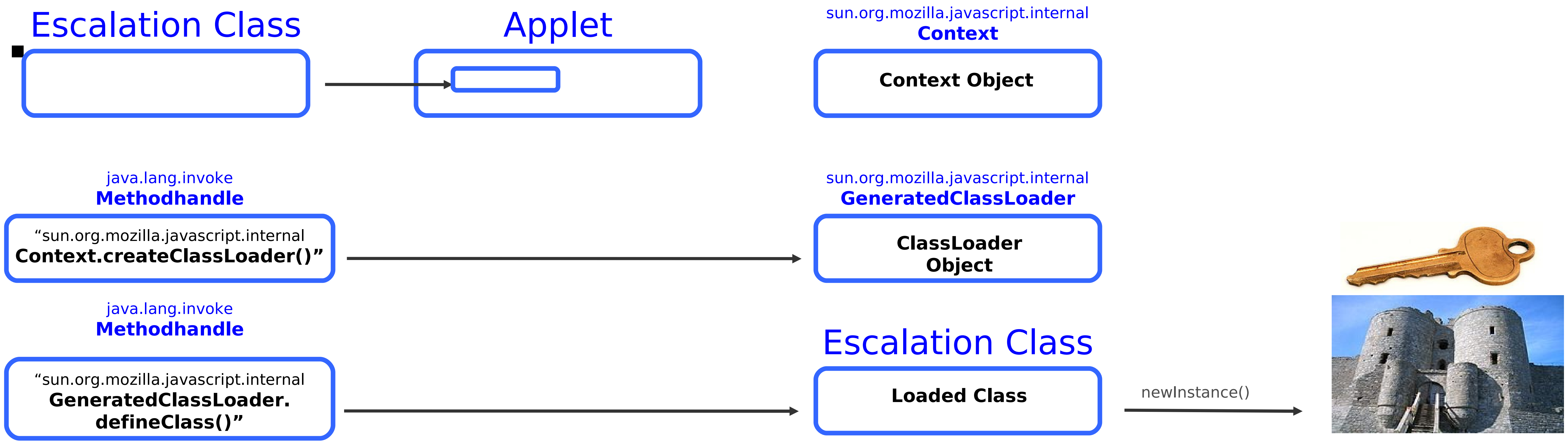
How it worked



- Enabling the defineClass method to be called, passing our escalation class bytes.

"New Year Day" / "EveryDay"

How it worked



- Create an instance of it, and the security manager is disabled.

"New Year Day" / "EveryDay"

The resolution

- JDK 7u11 included the “fix”
  - Reports suggest only the reflection exposure was closed.
  - The default security level was changed to ensure users are always prompted before running unsigned or self signed content.
  -
- This was implementation dependent, the IBM JDK was not affected.

Attack Vectors: Applet / Browser, Local

- Applet / Browser
  - These vulnerabilities are specific to applications running in or via the browser
  - The vulnerability exists either in the plugin or browser, or it is in the underlying JDK but only exposed when run in the browser environment.
  -
- Local
  - The local vector requires an attacker to have access to the system on which the JDK is running.
  - A simple example would be an application writing data to a temporary file before sending it to a printer.
    - If the files are created with inappropriate permissions any user on the system could access them.

And after all that…

Security is Important to Java

What's being done about security?

- IBM and Oracle are working to ensure Java is (and remains) secure!

  http://www.oracle.com/technetwork/topics/security/whatsnew/index.html

- Reporting Issues:

  http://www-03.ibm.com/security/secure-engineering/report.html

  http://www.oracle.com/us/support/assurance/reporting/index.html

- Writing more secure code:
  - Read and adhere to Oracles "Secure Coding Guidelines":

    http://www.oracle.com/technetwork/java/seccodeguide-139067.html

Conclusions

- Java Security is defense in depth
-
- Trust, but Verify
-
- Java and JVM designed to provide security at a low cost to developers
-
- Many moving parts in security – Things can go wrong, but quick to resolve