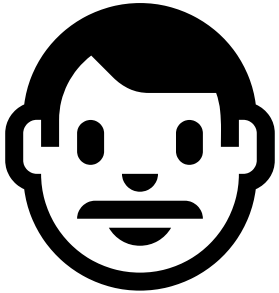
A Hacker's Guide to Securing Python Web Applications

Eyitemi Egbejule

Moscow ★ Python Conf++ 2018



EYITEMI EGBEJULE

- Hardware, Software and Bio Hacker
- (Research, RE, Web/Mobile Apps Sec, DevOps.)
- Co-Lead, OWASP Nigeria
- Individual Member, Django Software Foundation & Fellow, Python Software Foundation.
- Retired Jazz Musician
- Member of NaijaSecForce

Common Python Web App Security Flaws (OWASP TOP 10)

A1. Injection
A2. Broken Authentication
A3. Sensitive Data Exposure
A4. XML External Entities (XXE)
A5. Broken Access Control
A6. Security Misconfiguration
A7. Cross-Site Scripting (XSS)
A8. Insecure Deserialization
A9. Using Components with Known Vulnerabilities
A10. Insufficient Logging and Monitoring

SECURITY MISCONFIGURATION

Manual, ad hoc, insecure, or lack of security configurations that enable unauthorized access

SECURITY MISCONFIGURATION

- Insecure application settings
- Unused pages
- Default Credentials

RuntimeError at /taskManager/register/

generator raised StopIteration

```
8    DEBUG = True
9    TEMPLATE_DEBUG = True
0
1    ALLOWED_HOSTS = []
```

Request Method:	POST
Request URL:	http://127.0.0.1:8000/taskManager/register/
Django Version:	1.8.3
Exception Type:	RuntimeError
Exception Value:	`generator raised StopIteration`
Exception Location:	/Users/insaida/workspace/conference/vulnenv/lib/python3.7/site-packages/django/http/multipartparser.py in read, line 337
Python Executable:	/Users/insaida/workspace/conference/vulnenv/bin/python
Python Version:	3.7.0

Python Path:	`['/Users/insaida/workspace/conference/django.nV',`
	`'/Users/insaida/workspace/conference/vulnenv/lib/python37.zip',`
	`'/Users/insaida/workspace/conference/vulnenv/lib/python3.7',`
	`'/Users/insaida/workspace/conference/vulnenv/lib/python3.7/lib-dynload',`
	`'/usr/local/Cellar/python/3.7.0/Frameworks/Python.framework/Versions/3.7/lib/python3.7',`
	`'/Users/insaida/workspace/conference/vulnenv/lib/python3.7/site-packages']`

Server time:	Fri, 10 Aug 2018 07:30:20 +0000

Traceback Switch to copy-and-paste view

/Users/insaida/workspace/conference/vulnenv/lib/python3.7/site-packages/django/core/handlers/base.py in get_response

```
125.                                 response = middleware_method(request, callback, callback_args, callback_kwargs)  ...
```

▶ Local vars

/Users/insaida/workspace/conference/vulnenv/lib/python3.7/site-packages/django/middleware/csrf.py in process_view

```
174.                                 request_csrf_token = request.POST.get('csrfmiddlewaretoken', '')  ...
```

▶ Local vars

/Users/insaida/workspace/conference/vulnenv/lib/python3.7/site-packages/django/core/handlers/wsgi.py in _get_post

```
137.                             self._load_post_and_files()  ...
```

▶ Local vars

/Users/insaida/workspace/conference/vulnenv/lib/python3.7/site-packages/django/http/request.py in _load_post_and_files

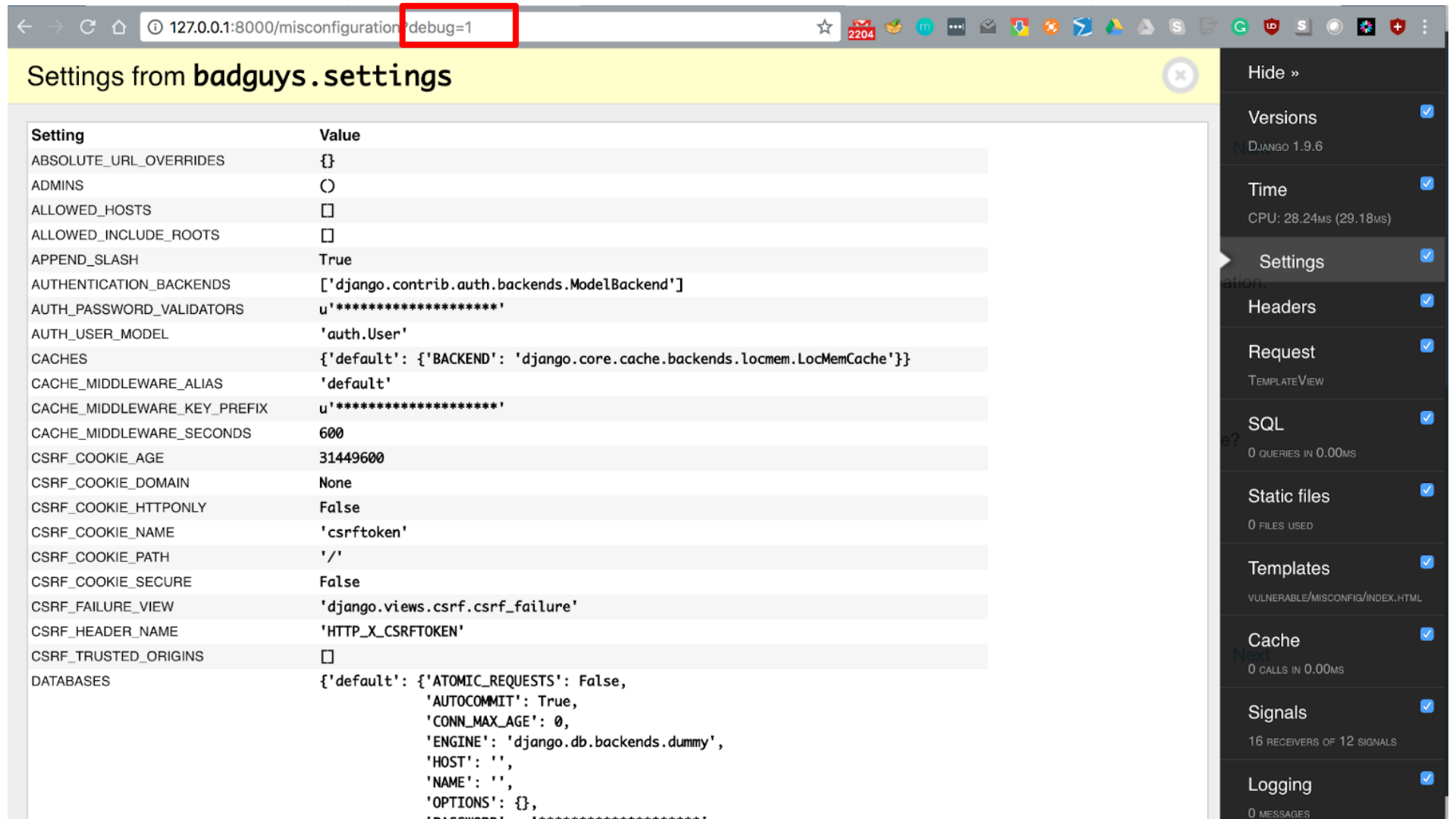
```
260.                                 self._post, self._files = self.parse_file_upload(self.META, data)  ...
```

▶ Local vars

/Users/insaida/workspace/conference/vulnenv/lib/python3.7/site-packages/django/http/request.py in parse_file_upload

```
225.                         return parser.parse()  ...
```

▶ Local vars


127.0.0.1:8000/misconfiguration?debug=1
Settings from badguys.settings
Setting
Value
ABSOLUTE_URL_OVERRIDES {}
ADMINS ()
ALLOWED_HOSTS []
ALLOWED_INCLUDE_ROOTS []
APPEND_SLASH True
AUTHENTICATION_BACKENDS ['django.contrib.auth.backends.ModelBackend']
AUTH_PASSWORD_VALIDATORS u'**********'
AUTH_USER_MODEL 'auth.User'
CACHES {'default': {'BACKEND': 'django.core.cache.backends.locmem.LocMemCache'}}
CACHE_MIDDLEWARE_ALIAS 'default'
CACHE_MIDDLEWARE_KEY_PREFIX u'**********'
CACHE_MIDDLEWARE_SECONDS 600
CSRF_COOKIE_AGE 31449600
CSRF_COOKIE_DOMAIN None
CSRF_COOKIE_HTTPONLY False
CSRF_COOKIE_NAME 'csrftoken'
CSRF_COOKIE_PATH '/'
CSRF_COOKIE_SECURE False
CSRF_FAILURE_VIEW 'django.views.csrf.csrf_failure'
CSRF_HEADER_NAME 'HTTP_X_CSRFTOKEN'
CSRF_TRUSTED_ORIGINS []
DATABASES {'default': {'ATOMIC_REQUESTS': False,
'AUTOCOMMIT': True,
'CONN_MAX_AGE': 0,
'ENGINE': 'django.db.backends.dummy',
'HOST': '',
'NAME': '',
'OPTIONS': {},
Hide »
Versions
Django 1.9.6
Time
CPU: 28.24ms (29.18ms)
Settings
Headers
Request
TemplateView
SQL
0 queries in 0.00ms
Static files
0 files used
Templates
vulnerable/misconfig/index.html
Cache
0 calls in 0.00ms
Signals
16 receivers of 12 signals
Logging
0 messages

sqlite3.OperationalError

OperationalError: no such table: urls

Traceback (most recent call last)

File "/Users/mike/.virtualenvs/envs/chiisai/lib/python2.7/site-packages/flask/app.py", line *1701*, in __call__

```
return self.wsgi_app(environ, start_response)
```

```
[console ready]
>>> import os
>>> print os.popen('ls -al').read()
total 128
drwxr-xr-x+ 18 mike  staff    612 May 10 22:12 .
drwxr-xr-x+  8 mike  staff    272 May 10 22:11 ..
-rw-r--r--+  1 mike  staff  12288 May 10 22:12 .app.py.swp
-rw-r--r--+  1 mike  staff      0 Oct 20  2012 __init__.py
-rw-r--r--+  1 mike  staff    131 May 10 22:11 __init__.pyc
-rw-r--r--+  1 mike  staff   2832 May 10 22:12 app.py
-rw-r--r--+  1 mike  staff   1289 Jun  4  2013 base.py
-rw-r--r--+  1 mike  staff   1759 May 10 22:11 base.pyc
-rw-r--r--+  1 mike  staff    269 Oct 20  2012 forms.py
-rw-r--r--+  1 mike  staff    677 May 10 22:11 forms.pyc
-rw-r--r--+  1 mike  staff    241 Oct 20  2012 schema.sql
-rw-r--r--+  1 mike  staff    132 Jun  4  2013 settings.py
-rw-r--r--+  1 mike  staff    328 May 10 22:11 settings.pyc
-rw-r--r--+  1 mike  staff   2352 Jun  4  2013 shortener.py
```

Fix for Flask

- Set app.debug = False to turn off debugging

- Read http://flask.pocoo.org/docs/ security

FIX FOR DJANGO

- Don’t run in debug mode in production
- Keep your SECRET_KEY secret!
- Keep Python code out of webserver’s root
- Don’t run admin publicly
- Don’t use the built-in admin for normal user admin tasks

"DEBUG = False"

default_settings.py file

```
1  import os
2
3  # Get environment, or set to development by default
4  app_env = os.environ.get('APPLICATION_ENVIRONMENT') or 'development'
5
6  # Settings applied to all environments
7  SECRET_KEY = 'development key'
8
9  # Settings applied to specific environments
10 if app_env == 'production':
11     DATABASE_URI = ''  # TODO: Enter your production database
12     DEBUG = False
13 elif app_env == 'development':
14     DATABASE_URI = ''  # TODO: Enter your dev database
15     DEBUG = True
16 elif app_env == 'testing':
17     DATABASE_URI = ''  # TODO: Enter your test database
```

INJECTION

Allowing untrusted data to be sent as part of a command or query.

FUN FACTS

SQL injection was used in the infamous Sony Pictures hack of 2014, when suspected North Korean operatives gained access to confidential data.

```
def upload(request, project_id):
    if request.method == 'POST':

        proj = Project.objects.get(pk = project_id)
        form = ProjectFileForm(request.POST, request.FILES)

        if form.is_valid():
            name = request.POST.get('name', False)
            upload_path = store_uploaded_file(name, request.FILES[
'file'])

            curs = connection.cursor()
            curs.execute("insert into taskManager_file ('name','pat
h','project_id') values ('%s','%s',%s)"%(name,upload_path,project_i
d))

            return redirect('/taskManager/' + project_id + '/', {'n
ew_file_added':True})
        else:
            form = ProjectFileForm()
    else:
        form = ProjectFileForm()
    return render_to_response('taskManager/upload.html', {'form': f
orm}, RequestContext(request))
```

No input validation, leading to SQLi and OS injection

FIX

- ALWAYS Validate and sanitize user input.
- For Django, use Django ORMs to create a file object.

5

BROKEN ACCESS CONTROL

Improper enforcement of what authenticated users are allowed to do

FUN FACT

A web meeting platform, Fuze, enabled meeting access via a simple URL ending with an incrementing seven-digit number. Using any number provided access to replays of the corresponding meeting. Since the URLs were unprotected, the content was then indexed by — and searchable through — popular search engines.

Exercises

- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross-Site Scripting (XSS)
- A4: Insecure Direct Object References
- A5: Security Misconfiguration
- A6: Sensitive Data Exposure
- A7: Missing Function-Level Access Control
- A8: Cross-Site Request Forgery (CSRF)
- A9: Using Known Vulnerable Components
- A10: Unvalidated Redirects and Forwards

A Happy Little Page

This is a happy little page.

It is most certainly not allowing you any 'admin' level access. What if the admin login just used an admin action to allow admins to have super-human powers?

127.0.0.1:8000/missing-access-control/happy-pag?action=admin

Exercises

- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross-Site Scripting (XSS)
- A4: Insecure Direct Object References
- A5: Security Misconfiguration
- A6: Sensitive Data Exposure
- A7: Missing Function-Level Access Control
- A8: Cross-Site Request Forgery (CSRF)
- A9: Using Known Vulnerable Components
- A10: Unvalidated Redirects and Forwards

Super-Secret Admin Interface

Welcome to the super-secret admin interface!

I sure hope no bad guys ever get in here!

It's not as common to use the querystring to grant privilege escalation, however, there are still lots of cases where cookies are used.

FIX

- Session cookies

- Implement per-user or per-session indirect object references

- This can be as much about URL design as about access control!

2

BROKEN AUTHENTICATION

Incorrectly implemented authentication and session management functions

FUN FACT

The simplest examples of this vulnerability are either storing user credentials without encryption or allowing them to be easily guessed. Other examples include using session IDs in the URL and enabling unreasonably long session timeouts.

HOW TO FIX?

- Lock down views: • Use Django’s permissions architecture
- Use Flask-Security or Flask-Login
- Customize Django queryset for looking up objects that involve user ownership

USING COMPONENTS WITH KNOWN VULNERABILITIES

Finding and exploiting already- known vulnerabilities before they are fixed

Django's Top 10 Vulnerabilities

10. Session Modification (CVE-2011-4136)

*Versions **1.2.7** and **1.3.x** before **1.3.1***

When session details are stored in the cache, root namespacing is used for both session identifiers and application-data keys. This can allow remote attackers to modify a session by triggering use of a key that is equal to that session's identifier.

9. Session Hijacking (CVE-2014-0482)

*Versions **1.4.14, 1.5.x** before **1.5.9, 1.6.x** before **1.6.6,** and **1.7** before **release candidate 3**.*

Session hijacking involves an attacker gaining unauthorized access to a system using another user's session data. In this case, when using contrib.auth.backends.RemoteUserBackend, remote authenticated users can hijack web sessions via vectors related to the REMOTE_USER header.

8. Cache Poisoning (CVE-2014-1418)

*Versions **1.4** before **1.4.13**, **1.5** before **1.5.8**, **1.6** before **1.6.5**, and **1.7** before **1.7b4***

Cache poisoning occurs when incorrect data is inserted into a DNS resolver 's cache, causing the nameserver to provide an incorrect IP address or destination. These versions of Django do not not properly include the:

1. Vary: Cookie

3.DoS: Via Unspecified Vectors (CVE-2015-5145)

Versions 1.8.x before 1.8.3

DoS is short for Denial of Service, and occurs when an attacker brings down a network/website by flooding it with data packets. The validators.URLValidator in these versions of Django allow remote attackers to cause a denial of service (CPU consumption) via unspecified vectors.

2.DoS : Via Multiple Requests With Unique Session Keys (CVE-2015-5143)

Versions before 1.4.21, 1.5.x through 1.6.x, 1.7.x before 1.7.9, and 1.8.x before 1.8.3

The session backends in Django allows remote attackers to cause a denial of service (session store consumption) via multiple requests with unique session keys.

1.Type Conversion Vulnerability (CVE-2014-0474)

Versions before 1.4.11, 1.5.x before 1.5.6, 1.6.x before 1.6.3, and 1.7.x before 1.7 beta

In these versions of Django, the following field classes do not properly perform type conversion :

- FilePathField
- GenericIPAddressField
- IPAddressField

FIX

PATCH PLEASE

CROSS-SITE SCRIPTING (XSS)

A web application includes untrusted data in a new web page without proper validation

CROSS-SITE SCRIPTING (XSS)

WHAT IS IT?

XSS allows malicious code to be added to a web page or app, say via user comments or form submissions used to define the subsequent action. Since HTML mixes control statements, formatting, and the requested content into the web page's source code, it allows an opportunity for unsanitized code to be used in the resulting page.

HOW DOES IT WORK?

When a web page or app utilizes user-entered content as part of a resulting page without checking for bad stuff, a malicious user could enter content that includes HTML entities.

WHY IS IT BAD?

Attackers can change the behavior of an app, direct data to their own systems, or corrupt or overwrite existing data.

FUN FACTS

XSS exploits have been reported for more than 20 years, and have impacted Twitter, Facebook, YouTube, and many, many others. It’s showing no signs of waining, however, as both Adobe and WordPress patched XSS vulnerabilities as recently as November 2017.

XSS MITIGATION

- By Default, Django escapes certain characters
- In Flask, Jinja2 escape untrusted input
- Make sure to always quote your attributes with either double or single quotes.

Don't do this:

```
<a href={{ url }}>Link</a>
```

Use this instead:

```
<a href="{{ url }}">Link</a>
```

For Jinja2, you can manually escape HTML by passing a value through |e.

```
{{ url|e }}
```

Another method is to escape with the escape() function.

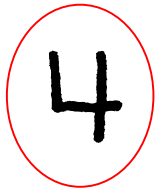
```
>>> from jinja2 import utils
>>> str(utils.escape("<h1>XSS</h1>"))
'&lt;h1&gt;XSS&lt;/h1&gt;'
```

SENSITIVE DATA EXPOSURE

Many web technologies weren't designed to handle financial or personal data transfers

SENSITIVE DATA EXPOSURE

Sensitive data, such as credit card numbers, health data, or passwords, should have extra protection given the potential of damage if it falls into the wrong hands.



XML EXTERNAL ENTITIES

XML “entities” can be used to request local data or files

XML EXTERNAL ENTITIES

WHAT IS IT?

XML is a data format used to describe different data elements. XML also uses “entities” to help define related data, but entities can access remote or local content, as harmless as pulling a current stock price from a third party website. Entities can, however, be used to request local data or files, which could then be returned — even if that data was never intended for outside access.

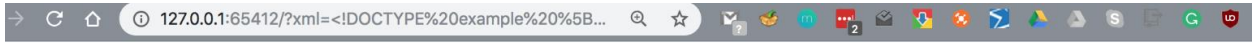
HOW DOES IT WORK?

An attacker sends malicious data lookup values asking the site, device, or app to request and display data from a local file. If a developer uses a common or default filename in a common location, an attacker’s job is easy.

WHY IS IT BAD?

Attackers can gain access to any data stored locally, or can further pivot to attack other internal systems.

http://127.0.0.1:65412/?xml=<!DOCTYPE example [<!ENTITY xxe SYSTEM "file:///etc/passwd">]><root>&xxe;</root>




```
ot>##
ser Database

ote that this file is consulted directly only when the system is running
n single-user mode.  At other times this information is provided by
pen Directory.

ee the opendirectoryd(8) man page for additional information about
pen Directory.

ody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
t:*:0:0:System Administrator:/var/root:/bin/sh
mon:*:1:1:System Services:/var/root:/usr/bin/false
cp:*:4:4:Unix to Unix Copy Protocol:/var/spool/uucp:/usr/sbin/uucico
skgated:*:13:13:Task Gate Daemon:/var/empty:/usr/bin/false
tworkd:*:24:24:Network Services:/var/networkd:/usr/bin/false
stallassistant:*:25:25:Install Assistant:/var/empty:/usr/bin/false
:*:26:26:Printing Services:/var/spool/cups:/usr/bin/false
stfix:*:27:27:Postfix Mail Server:/var/spool/postfix:/usr/bin/false
sd:*:31:31:Service Configuration Service:/var/empty:/usr/bin/false
s:*:32:32:Certificate Enrollment Service:/var/empty:/usr/bin/false
pstore:*:33:33:Mac App Store Service:/var/empty:/usr/bin/false
xalr:*:54:54:MCX AppLaunch:/var/empty:/usr/bin/false
pleevents:*:55:55:AppleEvents Daemon:/var/empty:/usr/bin/false
od:*:56:56:Geo Services Daemon:/var/db/geod:/usr/bin/false
rialnumberd:*:58:58:Serial Number Daemon:/var/empty:/usr/bin/false
vdocs:*:59:59:Developer Documentation:/var/empty:/usr/bin/false
ndbox:*:60:60:Seatbelt:/var/empty:/usr/bin/false
nsresponder:*:65:65:mDNSResponder:/var/empty:/usr/bin/false
```

XML External Entities

check if vulnerable

?xml=<root></root>

Exploit:

?xml=<!DOCTYPE example [<!ENTITY xxe SYSTEM "file:///etc/passwd">]><root>&xxe;</root>

INSECURE DESERIALIZATION

Receipt of hostile serialized objects resulting in remote code execution

INSECURE DESERIALIZATION

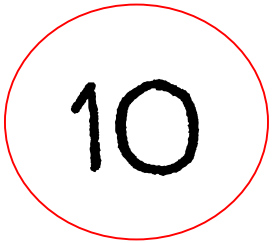
HOW DOES IT WORK?

Deserialized data can be modified to include malicious code, which is likely to cause issues if the application does not verify the data's source or contents before deserialization.

WHY IS IT BAD?

Attackers can build illegitimate objects that execute commands within an infected application.

Using Pickle for Serialization



INSUFFICIENT LOGGING & MONITORING

Insufficient monitoring allows attackers to work unnoticed

INSUFFICIENT LOGGING & MONITORING

WHY IS IT BAD?

Attackers rely on the lack of monitoring to exploit vulnerabilities before they're detected. Without monitoring and the logging to look back to see what happened, attackers can cause damage now and in the future.

FUN FACTS

Logging isn’t just important for identifying attacks in progress; it can assist with the forensic analysis after an attack has succeeded.

SECURITY SCANNERS

- Python Taint (PYT) – Static Analysis Tool
- Bandit
- Spaghetti Security Scanner
- Mona.py

THANKS!

Questions?

Twitter: @eEyitemi
insaida@protonmail.com

Moscow Python Conf++ 2018