BugEx Online - Deployment Guide

Amir Baradaran amirhossein.baradaran@gmail.com
Tim Krones tkrones@coli.uni-saarland.de
Frederik Leonhardt frederik.leonhardt@gmail.com
Christos Monogios chr_mon_@hotmail.com
Akmal Qodirov s9akqodi@stud.uni-saarland.de
Iliana Simova ili4_s@yahoo.com
Peter Stahl pstahl@coli.uni-saarland.de

July 31, 2012

Contents

1 Introduction

This tutorial covers the deployment of BugEx Online on a productive server system. The operating system used in this tutorial is Ubuntu Server 11.10 (64 bit).

The first part of the tutorial describes how to install all required dependencies, the second part explains how to configure both BugEx Online and the Apache webserver. The last part discusses some thoughts on security vulnerabilities and general Python/Django behaviour when used on an Apache with mod_wsgi.

2 Installation

2.1 Prerequisites

Following dependencies should already be installed *before* proceeding:

- Python 2.7.*
- Apache 2.2.20+
- MySQL Server 5.1.63+
- Java Version 6 Update 33+ or Java Version 7 Update 5+

You can check the versions with the following commands:

```
$ python --version
Python 2.7.2+
```

```
$ apache2 -version
Server version: Apache/2.2.20 (Ubuntu)
Server built:   Feb 14 2012 16:35:35
```

```
$ mysql -uroot -p
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 557
Server version: 5.1.63-0ubuntu0.11.10.1 (Ubuntu)
```

```
$ java -version
java version "1.6.0_33"
Java(TM) SE Runtime Environment (build 1.6.0_33-b03)
Java HotSpot(TM) 64-Bit Server VM (build 20.8-b03, mixed mode)
```

2.2 Additional prerequisites

From this base, the tutorial covers the installation of the following additional packages needed for deploying BugEx Online:

- Python-MySQL connector
- Django 1.4
- Python Imaging Library (PIL) 1.1.7
- django-simple-captcha 0.3.5
- Apache mod_wsgi

2.2.1 Python-dev libraries, mysql connector and pip

The first step is to install python-dev libraries which are needed in order to build PIL later on and the mysql connector which enables python to use MySQL databases. Some more information can be found on the official mysql-python website: http://mysql-python.sourceforge.net/MySQLdb.html.

```
$ sudo apt-get install python2.7-dev python-mysqldb
```

To install additional Python packages, it is convenient to use pip: http://pypi.python.org/pypi/pip.

```
$ sudo apt-get install python-pip
```

2.2.2 Django 1.4

At the time of writing this tutorial, Ubuntu's default repository only contains Django 1.3. This means we have to install Django 1.4 manually. Download and installation instructions can be found on the Django project website: https://www.djangoproject.com/download

```
$ wget https://www.djangoproject.com/download/1.4/tarball/
$ tar xzvf Django-1.4.tar.gz
$ cd Django-1.4
$ sudo python setup.py install
```

2.2.3 django-simple-captcha and PIL 1.1.7

Django's simple-captcha module uses the Python Image Library to synthesize the captchas. To synthesize images, PIL uses libjpeg, libpng and libfreetype. Install the following packages to satisfy those dependencies:

```
$ sudo apt-get install libjpeg8 libjpeg62-dev libpng12-0 zlib1g zlib1g-dev
                       libfreetype6 libfreetype6-dev
```

Now, to the ugly part. **You can omit this step on Ubuntu 12.04+!**
On the 64 bit version of Ubuntu 11.10, those libraries don't get installed correctly. We have to help a bit by creating appropriate symlinks:

```
$ sudo ln -s /usr/lib/x86_64-linux-gnu/libjpeg.so /usr/lib
$ sudo ln -s /usr/lib/x86_64-linux-gnu/libfreetype.so /usr/lib
$ sudo ln -s /usr/lib/x86_64-linux-gnu/libz.so /usr/lib
```

If you are interested in the details, check out this blog entry.

After this procedure we can now install PIL: http://www.pythonware.com/products/pil

```
$ sudo pip install -U PIL
```

Amongst others, you should be able to see the following output:

```
--------------------------------------------------------------------
*** TKINTER support not available
--- JPEG support available
--- ZLIB (PNG/ZIP) support available
--- FREETYPE2 support available
*** LITTLECMS support not available
--------------------------------------------------------------------
```

If it reads *not available* for either JPEG, ZLIB or FREETYPE support, uninstall PIL again (`pip uninstall PIL`), go back to the last step and properly install all dependencies.
If everything went smoothly now is the time to install django-simple-captcha:

```
$ pip install django-simple-captcha
```

2.2.4 Apache mod_wsgi

The Apache module mod_wsgi allows an Apache server to host any WSGI-compliant Python application. More details are available at the mod_wsgi website: http://code.google.com/p/modwsgi. The installation is straightforward:

```
$ sudo apt-get install libapache2-mod-wsgi
```

Now all necessary dependencies are installed and we can proceed with the setup of BugEx Online.

2.3 Setup BugEx Online

You can retrieve the newest version of BugEx Online at the Github repository: https://github.com/pemistahl/bugex-online. Alternatively, use a prepackaged zip archive. This tutorial refers to the BugEx Online root folder as the folder containing the `manage.py` file.

First create a folder for your django applications. The apache user/group must have read and write access to the folder. In the following example apache runs under the usergroup `www-data`.

```
$ sudo mkdir /var/django/
$ sudo chown root:www-data /var/django/
$ sudo chmod 775 /var/django/
```

After that, put the BugEx Online root folder in this newly created folder - named `/var/django/bugex_online/`. Now we recursively assign the rights again and give all rights to the uploads folder.

```
$ sudo chmod -R 775 /var/django
$ sudo chmod 777 /var/django/bugex_online/uploads
```

This concludes the basic setup of BugEx Online. Next step is the configuration of MySQL, BugEx Online and Apache.

2.4 Configuration

2.4.1 MySQL

Time to setup a database including an database user. You may use a graphical interface like phpMyAdmin for this task. Start the mysql shell with root rights, create a database called `bugexonline`, and a user `bugex` with the password `bugex`. Also assign rights to create a database called `test_bugexonline`, which is used by the unit tests.

```
$ mysql -uroot -p
> CREATE DATABASE bugexonline;
> GRANT ALL ON bugexonline.* to 'bugex'@'localhost' IDENTIFIED BY 'bugex';
> GRANT ALL ON test_bugexonline.* to 'bugex'@'localhost';
```

Of course, you can use arbitrary values for the database, the user and the password. The credentials BugEx online will use are configurable in the BugEx Online configuration.

2.4.2 BugEx Online

You have to tell BugEx Online where to find the BugEx executable. Navigate to `/var/django/bugex_online/bugex_webapp/core_modules` and open the `core_config.py`. You will find a configuration line called `EXECUTABLE`. Specify the absolute path of the BugEx executable here.

```
    # Absolute path of the BugEx executable.
    EXECUTABLE = '/var/django/bugex-mock-0.0.6-SNAPSHOT-jar-with-dependencies.jar'
```

Listing 1: BugEx Online configuration (Core interface to BugEx)

Additionally, in a productive system set the `CHECK_INTERVAL` to something higher, for example 30 seconds. Also you need to disable the `DEBUG` mode, which will disable the use of an artificial delay.

There is more configuration to be done in `/var/django/bugex_online/bugex_online/settings.py`. The `ROOT_PATH` variable defines the root path of BugEx Online. `getcwd()` will not work here in an Apache server setup, therefore you have to specify the absolute path on the filesystem. The `APPLICATION_BASE_URL` variable specifies the URL and path of the deployed BugEx Online application. For example, this URL will be used to build the links in the Email notifications. The `ADMINS` variable holds information about the system administrator. Uncomment the line and change the values to appropriate ones.

```
# The current working directory,
# used for creating file paths which are
# independent of a particular computer.
ROOT_PATH = '/var/django/bugex_online'

# The application base url configures the URL context of the application.
# The url will be used for example in email notifications.
# For a productive system, this could read; 'http://www.bugex.com/bugexonline'
APPLICATION_BASE_URL = 'http://130.185.104.44/bugexonline'

# A tuple that lists people who get code error notifications.
# When DEBUG=False and a view raises an exception, Django will
# email these people with the full exception information.
ADMINS = (
  # ('Your Name', 'your_email@example.com'),
)
```

Listing 2: BugEx Online configuration (Webapp)

Additionally you can find settings for the database which should be used (name, user, password) under `DATABASES` and some sections for configuring the SMTP server the system should use under `EMAIL_*`.

```
# A dictionary containing the settings for all databases
# to be used with Django. It is a nested dictionary whose contents
# maps database aliases to a dictionary containing the options
# for an individual database.
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'bugexonline',
        'USER': 'bugex',
        'PASSWORD': 'bugex',
        },
    'sqlite': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': '{0}/bugex_online_database.db'.format(ROOT_PATH),
        }
}

# The backend to use for sending emails.
EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_HOST_USER = 'bugexonline@gmail.com'
EMAIL_HOST_PASSWORD = (omitted)
```

Listing 3: BugEx Online configuration (Webapp, part 2)

Now the system is almost ready to perform a first test run. Tell django to collect all necessary static files.

```
$ cd /var/django/bugex_online
$ python manage.py collectstatic
```

And then create the database tables with the following command:

```
$ python manage.py syncdb
```

You should get feedback about database tables being created and you get the chance to create a superuser for accessing the admin interface. Now it's time for a first test run with the built in development server of Django (**DO NOT USE THIS SERVER FOR PRODUCTIVE SYSTEMS!**). The following command starts the server, listening on all available IPs on port 8000.

```
$ python manage.py runserver 0.0.0.0:8000
```

Now it can be accessed by entering the appropriate URL in a browser, e.g. `http://localhost:8000/`. If everything seems to run correctly so far, we can go on to the Apache configuration.

2.4.3 WSGI Application

A WSGI file is needed to deploy the application with Apache mod_wsgi. This file can be found in `/var/django/bugex_online/apache/bugexonline.wsgi`

```
    # add python path
    path = '/var/django/bugex_online'
```

Listing 4: BugEx Online configuration (WSGI)

In here, just the `path` variable has to be adjusted to the BugEx root folder.

2.4.4 Apache

If you are runnning Apache with a vhost configuration, it can be found for example in `/etc/apache2/sites-enabled/000-default`. Alternatively, use the global `/etc/apache2/apache2.conf`.

Let's have a closer look at the configuration elements we need.

```
        # wsgi application
        WSGIScriptAlias /bugexonline /var/django/bugex_online/apache/bugexonline.wsgi

        <Directory /var/django/bugex_online/apache/bugexonline.wsgi>
            Order allow,deny
            Allow from all
        </Directory>
```

Listing 5: Apache2 configuration: WSGI configuration

The WSGI section defines the active WSGI applications and specifies the URL path for them. Here we just need to configure the `bugexonline.wsgi` as seen above.

```
        # Possible values include: debug, info, notice, warn, error, crit,
        # alert, emerg.
        LogLevel info
```

Listing 6: Apache2 configuration: Logging configuration

You should set the logging level of the vhost to `INFO` to collect all logging output of the Python application in the Apache logging files. Even when specifying a separate logging file for your vhost, some logging output can still end up in the main apache error log, so be sure to check both on any unexpected errors. By default, all logging output of the Django application will be redirected to the Apache Error Log (even the info messages).

```
<VirtualHost *:80>
    # default configuration like DocumentRoot, CGI configuration..
    (omitted)

    # WSGI config
    # wsgi application
    WSGIScriptAlias /bugexonline /var/django/bugex_online/apache/bugexonline.wsgi

    <Directory /var/django/bugex_online/apache/bugexonline.wsgi>
        Order allow,deny
        Allow from all
    </Directory>

    # logging config
    ErrorLog \${APACHE_LOG_DIR}/error.log

    # Possible values include: debug, info, notice, warn, error, crit,
    # alert, emerg.
    LogLevel info

    CustomLog \${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Listing 7: Apache2 vhost configuration

In the end, the resulting configuration should look similar to the one in Listing 7.

3 Additional thoughts

3.1 Security

In this prototype the underlying BugEx mock-up implementation executes any JUnit tests it encounters. Since BugEx itself will not be executed in any kind of jail, **malicious code can easily harm the system**. More specifically, the Java process will be started with the same user rights of the Apache thread handling the WSGI application.

```
www-data  5349  7.6  2.4 443868 24788 ?        Sl   18:01   0:00
  java -jar /var/django/bugex-mock-0.0.6-SNAPSHOT-jar-with-dependencies.jar
    /var/django/bugex_online/uploads/user_3/1424b41f-3c04-477d-8bd6-a8213d89257e/
      failing-program-0.0.2-SNAPSHOT-jar-with-dependencies.jar
    de.mypackage.TestMyClass # testGetMin
    /var/django/bugex_online/uploads/user_3/1424b41f-3c04-477d-8bd6-a8213d89257e
    5
```

Listing 8: Java process spawned by Python with rights of Apache user

In this example, Java is executed by the `www-data` user. But even with restricted rights the possibility of elevating the java process rights to root level exists through various exploits, especially when using outdated versions of Java.

Therefore, for use in a production system BugEx should take care of running the code submitted by users in a sandbox environment.

3.2 Path handling

The mod_wsgi documentation provides an excellent explanation why the underlying Python application should never use or rely on relative paths to the current working directory - Python's `getcwd()`: See Application Issues - Application Working Directory.

3.3 Delivery of Static Files

In a production setting, the Apache webserver is responsible for running the Django project and also delivering the static media files (Javascript, images, style sheets). With the configuration above, the static media is delivered from the WSGI application path. In case you want to deliver those static files via a different directory, e.g. the normal document root to ease maintainability, you have to adapt the BugEx Online configuration.

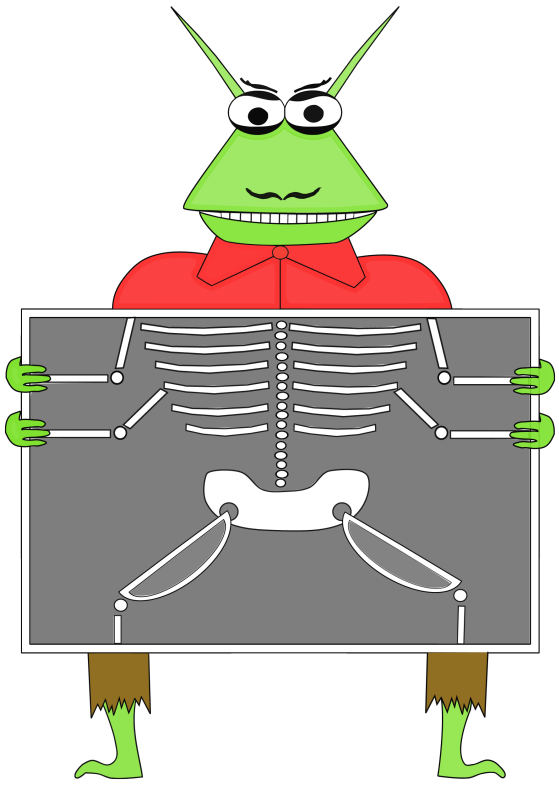
```
# Absolute path to the directory static files should be collected to.
# Don't put anything in this directory yourself; store your static files
# in apps' "static/" subdirectories and in STATICFILES_DIRS.
# Example: "/home/media/media.lawrence.com/static/"
STATIC_ROOT = '/var/www/static'

# URL prefix for static files.
# Example: "http://media.lawrence.com/static/"
STATIC_URL = '/static/'
```

Listing 9: BugEx Online configuration (Static Files)

For example, in the `settings.py` you can change the `STATIC_URL` to `/static/` and BugEx Online will look for any static media in `http://yourhost/static`. Django can automatically collect all your static files in a specified folder, so changing `STATIC_ROOT` to `/var/www/static/` will cause the `collectstatic` command to put the files in the correct directory.

3.4 Debugging

If something goes wrong, the most helpful place to look is the Apache error log. All logging output is going to end up there. Additionally, for some more output, you can enable the `DEBUG` values in the two configuration files of BugEx Online. The logging output of the BugEx program (jar) is stored in a file called `bugex.log` in the appropriate user request folder, e.g. `uploads/user_1/53ffa401-1235-4492-b55b-29cf20ad3602`.



Good luck and have fun!