Building AutoGrader as an Executable (Method 2 - Embedded Files)

This guide shows how to create a standalone executable with `config.ini` and `assignments.xlsx` **truly embedded** in the code, so they cannot be seen or modified by students.

Overview

Method 2 uses base64 encoding to embed files directly into Python code:

- Files are converted to base64 strings
- Strings are stored in `embedded_resources.py`
- Files are decoded at runtime (in memory only)
- No external config files exist
- Students cannot access or modify configuration

Prerequisites

- Python 3.8 or higher installed
- All AutoGrader files in a single directory
- Administrator/sudo access for installation

Step 1: Install Required Packages

bash

```
pip install pyinstaller pandas openpyxl matplotlib numpy reportlab
```

Step 2: Organize Your Project

Your project directory should contain:

```
AutoGrader/
├── autograder.py              # Core AutoGrader class
├── autograder_gui.py          # GUI application (updated for Method 2)
├── config.ini                 # Email configuration (will be embedded)
├── assignments.xlsx           # Assignments (will be embedded)
├── encode_resources.py        # Script to embed files (NEW)
├── autograder.spec            # PyInstaller configuration
└── icon.ico                   # Optional: Windows icon
```

Step 3: Configure Your Files

3.1 Edit `config.ini`

Create or edit `config.ini` with your actual credentials:

ini

```
[email]
smtp_server = smtp.gmail.com
smtp_port = 587
sender_email = autograder@university.edu
sender_password = your_gmail_app_password_here
instructor_email = instructor@university.edu

[settings]
# Set to false for production (hides email errors from students)
debug = false
```

Important: Use your real credentials here. This file will be embedded and hidden.

3.2 Verify `assignments.xlsx`

Make sure your `assignments.xlsx` has all the assignments you want to include.

Step 4: Embed Files into Code

This is the key step that makes files invisible to students.

Run the Encoding Script

bash

```
python encode_resources.py
```

Output:

```
Encoding resources for embedding...
============================================================
✓ Encoded config.ini (1234 characters)
✓ Encoded assignments.xlsx (56789 characters)
============================================================
✓ Generated embedded_resources.py

Embedded files:
  - config.ini
  - assignments.xlsx

These files are now embedded in Python code!
```

This creates `embedded_resources.py` which contains your files as base64-encoded strings.

Your directory now has:

```
AutoGrader/
├── autograder.py
├── autograder_gui.py
├── config.ini              # Original (can delete after building)
├── assignments.xlsx        # Original (can delete after building)
├── encode_resources.py
├── embedded_resources.py   # NEW - Contains embedded files
├── autograder.spec
└── icon.ico
```

Step 5: Build the Executable

Build Command

bash

```
pyinstaller autograder.spec
```

What Happens:

1. PyInstaller reads `autograder.spec`
2. Bundles `autograder_gui.py`, `autograder.py`, and `embedded_resources.py`
3. Includes all dependencies (pandas, matplotlib, etc.)
4. Creates executable in `dist/` folder

Build Output:

```
Building AutoGrader...
...
[Many lines of output]
...
Building EXE from EXE-00.toc completed successfully.
```

Step 6: Locate Your Executable

After building successfully:

```
AutoGrader/
├── build/                # Temporary files (can delete)
├── dist/                 # YOUR EXECUTABLE IS HERE!
│   └── AutoGrader.exe    # Windows
│   └── AutoGrader.app    # Mac
│   └── AutoGrader        # Linux
├── ... (source files)
```

The executable is in the `dist/` folder!

Step 7: Test the Executable

7.1 Test on Development Machine

1. Navigate to `dist/` folder
2. Run the executable:
    - **Windows**: Double-click `AutoGrader.exe`
    - **Mac**: Double-click `AutoGrader.app` or `open AutoGrader.app`

- **Linux**: `./AutoGrader`

3. Verify:
   - Application opens without errors
   - Assignments dropdown is populated
   - Student name can be entered
   - File browser works
   - "Check Code" works with a test submission
   - PDF export works
   - Email sending works (if configured)

7.2 Test on Clean Machine (Important!)

Test on a computer **without Python installed**:

1. Copy only the executable (`AutoGrader.exe`) to a clean machine
2. Run it
3. Verify all features work

This ensures the executable is truly standalone.

Step 8: Verify Files Are Hidden

Check That Config Is Embedded:

1. Look in `dist/` folder - You should see **only** `AutoGrader.exe`
2. No `config.ini` file visible ✓
3. No `assignments.xlsx` file visible ✓
4. Search the computer for these files after running - not found ✓

Verify With a Text Editor:

1. Try to open `AutoGrader.exe` in a text editor
2. Search for your password or email - should be obfuscated/compiled ✓

Students cannot easily access your configuration!

Step 9: Create Distribution Package

For Windows Distribution:

Create a folder with everything students need:

```
AutoGrader_v1.0_Windows/
├── AutoGrader.exe          # The executable
├── README.txt              # Instructions
└── examples/               # Example submissions (optional)
    ├── assignment1_example.py
    └── assignment2_example.py
```

README.txt:

AUTOGRADER - STUDENT GUIDE
==========

USAGE:
1. Run AutoGrader.exe
2. Enter your name
3. Select your assignment from the dropdown
4. Browse for your Python (.py) file
5. Click "Check Code"
6. Review results
7. Optional: Export to PDF for your records

SYSTEM INFO:
Your computer name and username will be automatically
included in the submission for verification purposes.

SUPPORT:
For technical issues, contact: instructor@university.edu

IMPORTANT:
- Results are automatically emailed to the instructor
- Make sure you have an internet connection
- Do not modify or rename the executable

Compress for Distribution:

Windows:

Right-click folder → Send to → Compressed (zipped) folder

Or use command line:

bash

```
# Windows
powershell Compress-Archive -Path AutoGrader_v1.0_Windows -DestinationPath AutoGrader_v1.0_Windows.zip

# Mac/Linux
zip -r AutoGrader_v1.0_Windows.zip AutoGrader_v1.0_Windows/
```

Platform-Specific Instructions

Windows (Building on Windows)

bash

```
# 1. Install PyInstaller
pip install pyinstaller

# 2. Embed files
python encode_resources.py

# 3. Build
pyinstaller autograder.spec

# 4. Test
cd dist
AutoGrader.exe
```

Troubleshooting Windows:

- **Antivirus blocks exe**: Add exception in Windows Defender
- **"Windows protected your PC"**: Click "More info" → "Run anyway"
- **Missing DLL errors**: Install Visual C++ Redistributable

Mac (Building on Mac)

```
bash
```

```
# 1. Install PyInstaller
pip3 install pyinstaller

# 2. Embed files
python3 encode_resources.py

# 3. Build
pyinstaller autograder.spec

# 4. Test
open dist/AutoGrader.app

# 5. Optional: Sign the app
codesign --force --deep --sign - dist/AutoGrader.app
```

Troubleshooting Mac:

- **"App is damaged"**: Right-click → Open (first time only)
- **Gatekeeper blocks**: System Preferences → Security → Allow
- **Code signing**: Required for distribution outside your organization

Linux (Building on Linux)

bash

```
# 1. Install PyInstaller
pip3 install pyinstaller

# 2. Embed files
python3 encode_resources.py

# 3. Build
pyinstaller autograder.spec

# 4. Make executable
chmod +x dist/AutoGrader

# 5. Test
./dist/AutoGrader
```

Troubleshooting Linux:

- **Missing tkinter**: `sudo apt-get install python3-tk`
- **Missing libraries**: `sudo apt-get install python3-dev`
- **Display issues**: Ensure X server is running

Complete Build Process Summary

Quick Reference:


bash

```
# Step 1: Install dependencies
pip install pyinstaller pandas openpyxl matplotlib numpy reportlab

# Step 2: Configure files
# Edit config.ini with real credentials
# Edit assignments.xlsx with your assignments

# Step 3: Embed files
python encode_resources.py

# Step 4: Build executable
pyinstaller autograder.spec

# Step 5: Test
cd dist
# Run AutoGrader.exe (Windows) or AutoGrader.app (Mac) or ./AutoGrader (Linux)

# Step 6: Distribute
# Copy only the executable from dist/ folder
```

Updating the Application

To Update Configuration or Assignments:

1. Edit `config.ini` or `assignments.xlsx`
2. Run `python encode_resources.py` again
3. Rebuild with `pyinstaller autograder.spec`
4. Test the new executable
5. Distribute updated version

Note: Any changes require rebuilding the executable.

Security Features of Method 2

✅ **Config file is embedded** - No external config.ini file ✅ **Excel file is embedded** - No external assignments.xlsx file ✅ **Base64 encoded** - Not plaintext in the executable ✅ **Compiled to bytecode** - Harder to reverse engineer ✅ **No temp file extraction** - Files decoded in memory ✅ **Students see only .exe** - One file, no configuration visible

Security Level: ⭐ ⭐ ⭐ ⭐ (Very Good)

While a determined attacker with reverse-engineering skills could potentially extract credentials, this method provides strong protection for typical student usage.

File Size Expectations

Typical executable sizes:

- **Windows**: 80-150 MB
- **Mac**: 90-160 MB
- **Linux**: 70-130 MB

Size includes:

- Python interpreter
- All libraries (pandas, matplotlib, numpy, reportlab)
- Your code and embedded files

Troubleshooting

"embedded_resources not found"

Problem: Forgot to run `encode_resources.py`

Solution:

bash

```
python encode_resources.py
pyinstaller autograder.spec
```

Executable crashes on startup

Problem: Missing dependencies or build error

Solution:


bash

```
# Build with console to see errors
# Edit autograder.spec: console=True
pyinstaller autograder.spec
# Run executable from terminal to see error messages
```

"Config Error" when running

Problem: Embedded resources not properly included

Solution:

1. Verify `embedded_resources.py` exists
2. Check it's in `hiddenimports` in autograder.spec
3. Rebuild: `pyinstaller autograder.spec --clean`

Email not sending

Problem: Configuration error or network issue

Solution:

1. Set `debug = true` in config.ini
2. Re-run `encode_resources.py`

3. Rebuild and test
4. Check error message displayed

PDF export not working

Problem: ReportLab not included

Solution:


bash

```
pip install reportlab
pyinstaller autograder.spec --clean
```

Assignments not loading

Problem: Excel file not properly embedded

Solution:

1. Verify `assignments.xlsx` exists and is valid
2. Re-run: `python encode_resources.py`
3. Rebuild: `pyinstaller autograder.spec`

Large file size

Problem: Including unnecessary libraries

Solution:

- Use virtual environment with only required packages
- Add more exclusions to `excludes` in autograder.spec
- Use UPX compression (already enabled in spec)

Distribution Checklist

Before distributing to students:

- [ ] Built executable successfully
- [ ] Tested on development machine
- [ ] Tested on clean machine (no Python)
- [ ] Verified config.ini is not visible
- [ ] Verified assignments.xlsx is not visible
- [ ] Set `debug = false` in config.ini before embedding
- [ ] Tested all assignment types
- [ ] Tested file browser
- [ ] Tested code checking
- [ ] Tested PDF export
- [ ] Tested email sending
- [ ] Created README for students
- [ ] Created distribution package
- [ ] Version numbered (e.g., v1.0)
- [ ] Tested on target platform (Windows/Mac/Linux)

Advanced: Building for Multiple Platforms

To create executables for all platforms, you need to build on each platform:

Option 1: Use Multiple Computers

1. **Windows PC**: Build Windows .exe
2. **Mac**: Build macOS .app
3. **Linux PC**: Build Linux binary

Option 2: Use Virtual Machines

1. Install VirtualBox or VMware
2. Create VMs for each platform
3. Build on each VM

Option 3: Use GitHub Actions (Automated)

Create `.github/workflows/build.yml`:

yaml

```yaml
name: Build AutoGrader

on:
  push:
    tags:
      - 'v*'

jobs:
  build:
    strategy:
      matrix:
        os: [windows-latest, macos-latest, ubuntu-latest]

    runs-on: ${{ matrix.os }}

    steps:
      - uses: actions/checkout@v2

      - uses: actions/setup-python@v2
        with:
          python-version: '3.10'

      - name: Install dependencies
        run: |
          pip install pyinstaller pandas openpyxl matplotlib numpy reportlab

      - name: Embed resources
        run: python encode_resources.py

      - name: Build executable
        run: pyinstaller autograder.spec
```

```yaml
      - name: Upload artifact
        uses: actions/upload-artifact@v2
        with:
          name: AutoGrader-${{ matrix.os }}
          path: dist/
```

Push a tag to automatically build for all platforms:


bash

```
git tag v1.0
git push origin v1.0
```

Comparison with Method 1

Feature	Method 1 (PyInstaller datas)	Method 2 (Embedded)
Config visible	In temp folder	Not visible
Excel visible	In temp folder	Not visible
Update without rebuild	No	No
Security	⭐⭐⭐ Good	⭐⭐⭐⭐ Very Good
Complexity	Easy	Medium
File size	Same	Same
Recommended for	Lab computers	Student devices

Method 2 is recommended when distributing to students' personal computers.

Support

Common Questions:

Q: Can students see my email password? A: Not easily. It's embedded and compiled. However, use a dedicated account with limited permissions as best practice.

Q: Can I update assignments without rebuilding? A: No. With Method 2, you must re-run `encode_resources.py` and rebuild.

Q: Does this work offline? A: Yes, except for email sending which requires internet.

Q: How do I add more assignments? A: Edit `assignments.xlsx`, run `encode_resources.py`, rebuild.

Q: Can I see what's embedded? A: Yes, check `embedded_resources.py` after running `encode_resources.py`.

Summary

Method 2 embeds configuration files directly in the executable:

1. ✅ Config and Excel files are truly embedded
2. ✅ Students cannot access or modify them
3. ✅ One executable file to distribute
4. ✅ Works on any computer (no Python needed)
5. ✅ Computer name and username automatically captured
6. ✅ Secure for student distribution

Build Process:

bash

```
python encode_resources.py  # Embed files
pyinstaller autograder.spec  # Build executable
```

Your AutoGrader is now ready for secure distribution! 🎉