Yalla-Hack Shield Enhanced - Hostinger Deployment Guide

Overview

This comprehensive guide provides step-by-step instructions for deploying the Yalla-Hack Shield Enhanced application on Hostinger's web hosting platform. The deployment process has been optimized for Hostinger's shared hosting environment while maintaining the application's full functionality and security features.

Prerequisites

Hostinger Account Requirements

- **Hosting Plan**: Premium or Business plan (required for Python support)
- **Domain**: Configured domain pointing to your Hostinger hosting account
- **SSH Access**: Enabled in your Hostinger control panel
- **Python Support**: Hostinger's Python hosting environment

Local Development Environment

- Completed Yalla-Hack Shield Enhanced application
- SSH client (Terminal on macOS/Linux, PuTTY on Windows)
- FTP/SFTP client (FileZilla recommended)
- Text editor for configuration files

Step 1: Prepare Application for Deployment

1.1 Update Configuration Files

Create a production configuration file:

```
# config.py
import os

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY') or
'YallaHack2025!ProductionKey#$%'
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or
'sqlite:///app.db'
    SQLALCHEMY_TRACK_MODIFICATIONS = False

    # Email Configuration
    MAIL_SERVER = os.environ.get('MAIL_SERVER') or 'smtp.hostinger.com'
    MAIL_PORT = int(os.environ.get('MAIL_PORT') or 587)
    MAIL_USE_TLS = os.environ.get('MAIL_USE_TLS', 'true').lower() in ['true',
'on', '1']
    MAIL_USERNAME = os.environ.get('MAIL_USERNAME')
    MAIL_PASSWORD = os.environ.get('MAIL_PASSWORD')

    # PayPal Configuration
    PAYPAL_CLIENT_ID = os.environ.get('PAYPAL_CLIENT_ID')
    PAYPAL_CLIENT_SECRET = os.environ.get('PAYPAL_CLIENT_SECRET')
    PAYPAL_MODE = os.environ.get('PAYPAL_MODE', 'sandbox')
```

1.2 Create Requirements File

Ensure your `requirements.txt` includes all necessary dependencies:

```
Flask==3.1.1
Flask-SQLAlchemy==3.1.1
Flask-CORS==6.0.0
Werkzeug==3.1.3
```

1.3 Create WSGI Entry Point

Create `wsgi.py` in your project root:

```python
#!/usr/bin/env python3
import sys
import os

# Add the project directory to Python path
sys.path.insert(0, os.path.dirname(__file__))

from src.main import app

if __name__ == "__main__":
    app.run()
```

Step 2: Hostinger Account Setup

2.1 Enable SSH Access

1. Log into your Hostinger control panel
2. Navigate to **Advanced** → **SSH Access**
3. Enable SSH access for your domain
4. Note the SSH connection details (hostname, port, username)

2.2 Configure Python Environment

1. Access **Advanced** → **Python**
2. Select Python 3.11 or the latest available version
3. Set the application startup file to `wsgi.py`
4. Configure the application root directory

2.3 Database Setup

For production deployment, consider upgrading to MySQL:

1. Navigate to **Databases** → **MySQL Databases**
2. Create a new database for your application
3. Create a database user with full privileges
4. Note the database connection details

Step 3: File Upload and Configuration

3.1 Upload Application Files

Using SFTP client (FileZilla):

1. Connect to your Hostinger account using SFTP
2. Navigate to your domain's public_html directory
3. Upload the entire application structure: `public_html/ ├── src/ | ├── main.py | ├── models/ | ├── routes/ | └── static/ ├── wsgi.py ├── requirements.txt ├── config.py └── .htaccess`

3.2 Create .htaccess File

Create `.htaccess` in your public_html directory:

```
 RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$` wsgi.py/`$1 [QSA,L]

# Security headers
Header always set X-Content-Type-Options nosniff
Header always set X-Frame-Options DENY
Header always set X-XSS-Protection "1; mode=block"
Header always set Strict-Transport-Security "max-age=31536000;
includeSubDomains"

# Cache static files
<FilesMatch "\.(css|js|png|jpg|jpeg|gif|ico|svg)$">
    ExpiresActive On
    ExpiresDefault "access plus 1 month"
</FilesMatch>
```

3.3 Set Environment Variables

Create `.env` file (ensure it's not publicly accessible):

```
SECRET_KEY=YallaHack2025!ProductionKey#$%
DATABASE_URL=mysql://username:password@localhost/database_name
MAIL_SERVER=smtp.hostinger.com
MAIL_PORT=587
MAIL_USE_TLS=true
MAIL_USERNAME=your-email@yourdomain.com
MAIL_PASSWORD=your-email-password
PAYPAL_CLIENT_ID=your-paypal-client-id
PAYPAL_CLIENT_SECRET=your-paypal-client-secret
PAYPAL_MODE=live
```

Step 4: SSH Configuration and Setup

4.1 Connect via SSH

```
ssh username@your-domain.com -p 65002
```

4.2 Install Dependencies

```
cd public_html
python3 -m pip install --user -r requirements.txt
```

4.3 Initialize Database

```
python3 -c "
import sys
sys.path.insert(0, '.')
from src.main import app, db
with app.app_context():
    db.create_all()
    print('Database initialized successfully')
"
```

4.4 Set File Permissions

```
chmod 755 wsgi.py
chmod -R 755 src/
chmod 644 .htaccess
chmod 600 .env
```

Step 5: WordPress Integration

5.1 WordPress Plugin Integration

Create a custom WordPress plugin for integration:

```php
<?php
/**
 * Plugin Name: Yalla-Hack Shield Integration
 * Description: Integrates Yalla-Hack Shield with WordPress
 * Version: 1.0
 */

// Add menu item to WordPress admin
add_action('admin_menu', 'yalla_hack_admin_menu');

function yalla_hack_admin_menu() {
    add_menu_page(
        'Yalla-Hack Shield',
        'Security Shield',
        'manage_options',
        'yalla-hack-shield',
        'yalla_hack_admin_page',
        'dashicons-shield',
        30
    );
}

function yalla_hack_admin_page() {
    $shield_url = 'https://yourdomain.com/dashboard.html';
    echo '<div class="wrap">';
    echo '<h1>Yalla-Hack Shield Dashboard</h1>';
    echo '<iframe src="' . $shield_url . '" width="100%" height="800px"
frameborder="0"></iframe>';
    echo '</div>';
}

// Add security widget to WordPress dashboard
add_action('wp_dashboard_setup', 'yalla_hack_dashboard_widget');

function yalla_hack_dashboard_widget() {
    wp_add_dashboard_widget(
        'yalla_hack_security_widget',
        'Security Status',
        'yalla_hack_security_widget_content'
    );
}

function yalla_hack_security_widget_content() {
    // Fetch security status from Yalla-Hack Shield API
    $api_url = 'https://yourdomain.com/api/devices/summary';
    $`response = wp_remote_get(`$api_url);

    if (!is_wp_error($response)) {
        $`data = json_decode(wp_remote_retrieve_body(`$response), true);
        echo '<p><strong>Total Devices:</strong> ' . $data['summary']
['total_devices'] . '</p>';
        echo '<p><strong>Online Devices:</strong> ' . $data['summary']
['online_devices'] . '</p>';
        echo '<p><strong>Compromised Devices:</strong> ' . $data['summary']
['compromised_devices'] . '</p>';
        echo '<a href="admin.php?page=yalla-hack-shield" class="button button-
primary">View Full Dashboard</a>';
    } else {
        echo '<p>Unable to fetch security status.</p>';
    }
```

```
}
?>
```

5.2 WordPress Theme Integration

Add security status to your WordPress theme:

```
// Add to functions.php
function yalla_hack_security_shortcode($atts) {
    $api_url = 'https://yourdomain.com/api/devices/summary';
    $`response = wp_remote_get(`$api_url);

    if (!is_wp_error($response)) {
        $`data = json_decode(wp_remote_retrieve_body(`$response), true);
        $output = '<div class="yalla-hack-security-status">';
        $output .= '<h3>Security Status</h3>';
        $`output .= '<p>Protected Devices: ' . `$data['summary']
['total_devices'] . '</p>';
        $`output .= '<p>Status: ' . (`$data['summary']['compromised_devices']
== 0 ? 'Secure' : 'Alert') . '</p>';
        $output .= '</div>';
        return $output;
    }

    return '<p>Security status unavailable.</p>';
}
add_shortcode('yalla_hack_status', 'yalla_hack_security_shortcode');
```

Step 6: SSL Certificate Configuration

6.1 Enable SSL in Hostinger

1. Navigate to **Security** → **SSL**
2. Enable SSL certificate for your domain
3. Force HTTPS redirection

6.2 Update Application Configuration

Modify your Flask application to handle HTTPS:

```python
# In src/main.py
from flask_talisman import Talisman

app = Flask(__name__)

# Force HTTPS in production
if not app.debug:
    Talisman(app, force_https=True)
```

Step 7: Performance Optimization

7.1 Enable Caching

Add caching headers to your `.htaccess`:

```
# Enable compression
<IfModule mod_deflate.c>
    AddOutputFilterByType DEFLATE text/plain
    AddOutputFilterByType DEFLATE text/html
    AddOutputFilterByType DEFLATE text/xml
    AddOutputFilterByType DEFLATE text/css
    AddOutputFilterByType DEFLATE application/xml
    AddOutputFilterByType DEFLATE application/xhtml+xml
    AddOutputFilterByType DEFLATE application/rss+xml
    AddOutputFilterByType DEFLATE application/javascript
    AddOutputFilterByType DEFLATE application/x-javascript
</IfModule>

# Browser caching
<IfModule mod_expires.c>
    ExpiresActive On
    ExpiresByType text/css "access plus 1 month"
    ExpiresByType application/javascript "access plus 1 month"
    ExpiresByType image/png "access plus 1 month"
    ExpiresByType image/jpg "access plus 1 month"
    ExpiresByType image/jpeg "access plus 1 month"
    ExpiresByType image/gif "access plus 1 month"
    ExpiresByType image/ico "access plus 1 month"
</IfModule>
```

7.2 Database Optimization

For MySQL databases, add these optimizations:

```sql
-- Create indexes for better performance
CREATE INDEX idx_user_email ON user(email);
CREATE INDEX idx_device_user_id ON device(user_id);
CREATE INDEX idx_security_event_device_id ON security_event(device_id);
CREATE INDEX idx_activity_log_user_id ON activity_log(user_id);
```

Step 8: Monitoring and Maintenance

8.1 Set Up Log Monitoring

Create a log monitoring script:

```
# monitor.py
import os
import smtplib
from email.mime.text import MIMEText
from datetime import datetime

def check_application_health():
    try:
        # Check if application is responding
        import requests
        response = requests.get('https://yourdomain.com/api/auth/session',
timeout=10)

        if response.status_code != 200:
            send_alert(f"Application health check failed:
{response.status_code}")

    except Exception as e:
        send_alert(f"Application health check error: {str(e)}")

def send_alert(message):
    # Send email alert
    msg = MIMEText(f"Yalla-Hack Shield Alert: {message}")
    msg['Subject'] = 'Yalla-Hack Shield Alert'
    msg['From'] = 'alerts@yourdomain.com'
    msg['To'] = 'admin@yourdomain.com'

    # Send email using Hostinger SMTP
    server = smtplib.SMTP('smtp.hostinger.com', 587)
    server.starttls()
    server.login('alerts@yourdomain.com', 'your-password')
    server.send_message(msg)
    server.quit()

if __name__ == "__main__":
    check_application_health()
```

8.2 Set Up Cron Jobs

Add to your crontab:

```
# Check application health every 15 minutes
*/15 * * * * /usr/bin/python3 /home/username/public_html/monitor.py

# Backup database daily at 2 AM
0 2 * * * mysqldump -u username -p password database_name >
/home/username/backups/backup_$(date +\%Y\%m\%d).sql
```

Step 9: Security Hardening

9.1 File Permissions

Set secure file permissions:

```
# Application files
find . -type f -name "*.py" -exec chmod 644 {} \;
find . -type d -exec chmod 755 {} \;

# Sensitive files
chmod 600 .env
chmod 600 config.py

# Executable files
chmod 755 wsgi.py
```

9.2 Security Headers

Add additional security headers to `.htaccess`:

```
# Security headers
Header always set X-Content-Type-Options nosniff
Header always set X-Frame-Options DENY
Header always set X-XSS-Protection "1; mode=block"
Header always set Referrer-Policy "strict-origin-when-cross-origin"
Header always set Content-Security-Policy "default-src 'self'; script-src
'self' 'unsafe-inline' cdnjs.cloudflare.com; style-src 'self' 'unsafe-inline'
cdnjs.cloudflare.com; img-src 'self' data:; font-src 'self'
cdnjs.cloudflare.com"

# Hide server information
ServerTokens Prod
Header unset Server
```

Step 10: Testing and Validation

10.1 Functionality Testing

Test all application features:

1. **User Registration and Login**
2. Create new user account
3. Login with credentials
4. Test password reset functionality
5. **Device Management**
6. Add new devices
7. Update device information
8. Remove devices
9. Test device scanning
10. **Subscription Management**
11. Test PayPal integration
12. Verify plan upgrades/downgrades
13. Check subscription limits
14. **Email Notifications**
15. Test security alert emails
16. Verify welcome emails
17. Check email delivery

10.2 Performance Testing

Use tools to test performance:

```
# Test response times
curl -w "@curl-format.txt" -o /dev/null -s "https://yourdomain.com"

# Load testing with Apache Bench
ab -n 100 -c 10 https://yourdomain.com/
```

10.3 Security Testing

Perform security validation:

1. **SSL Certificate Validation**
2. Use SSL Labs SSL Test
3. Verify certificate chain
4. Check for mixed content
5. **Security Headers**
6. Use Security Headers scanner
7. Verify CSP implementation
8. Check for information disclosure

Troubleshooting Common Issues

Issue 1: Application Not Loading

Symptoms: 500 Internal Server Error

Solutions: 1. Check error logs in Hostinger control panel 2. Verify Python version compatibility 3. Ensure all dependencies are installed 4. Check file permissions

Issue 2: Database Connection Errors

Symptoms: Database connection refused

Solutions: 1. Verify database credentials in `.env` 2. Check database server status 3. Ensure database user has proper privileges 4. Test connection manually

Issue 3: Email Not Sending

Symptoms: Email notifications not delivered

Solutions: 1. Verify SMTP settings in configuration 2. Check email account credentials 3. Ensure SMTP port is not blocked 4. Test email sending manually

Issue 4: PayPal Integration Issues

Symptoms: Payment processing failures

Solutions: 1. Verify PayPal API credentials 2. Check PayPal webhook configuration 3. Ensure SSL certificate is valid 4. Test in PayPal sandbox mode first

Maintenance Schedule

Daily Tasks

- Monitor application logs
- Check system resource usage
- Verify backup completion

Weekly Tasks

- Review security alerts
- Update application dependencies
- Test backup restoration
- Performance monitoring review

Monthly Tasks

- Security audit and penetration testing
- Database optimization and cleanup
- SSL certificate renewal check
- User access review

Support and Resources

Hostinger Support

- **Knowledge Base**: https://support.hostinger.com
- **Live Chat**: Available 24/7
- **Email Support**: support@hostinger.com

Application Support

- **Documentation**: Comprehensive guides and API reference
- **Email Support**: support@yalla-hack.net
- **Issue Tracker**: GitHub repository for bug reports

Additional Resources

- **Flask Documentation**: https://flask.palletsprojects.com
- **SQLAlchemy Documentation**: https://docs.sqlalchemy.org
- **PayPal Developer Documentation**: https://developer.paypal.com

---

This deployment guide provides comprehensive instructions for successfully deploying Yalla-Hack Shield Enhanced on Hostinger's hosting platform. Follow each step carefully and test thoroughly to ensure optimal performance and security.