

Forum:[Hack Forums](#)

Thread: [C++ Steganography Tool for Embedding/Extracting Data/ShellCode in Images](#)

secret_code - June 20, 2024, 8:17 am

simple and easy steganography tool i wrote during my break that allows users to embed and extract binary data or shellcode within images. Utilizing OpenCV for image processing, ensures the hidden data is seamlessly integrated, making it virtually undetectable to the naked eye. The program supports both Windows and Linux platforms, ensuring almost full compatibility.

FUNCTIONALITIES:

Embed Data: Insert binary data into an image file.

Extract Data: Retrieve embedded binary data from an image file.

HOW TO USE:

embed:

```
$ ./steganography_tool embed <input_image> <output_image> <data_file>
```

extract:

```
$ ./steganography_tool extract <input_image> <output_data>
```

example:

```
$ ./steganography_tool extract output.jpg extracted_data.bin
```

there might be some minor errors, but nothing special, i wrote this during my 30mins break so is not a big deal for me, a quick test is ok...

somehow while i was editing some "minor errors" within the post i messed the thread content, for some reason it got duplicated and i can't remove the clone anymore or i don't know how... anyway, enjoy!

```
Code #include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <bitset>
#include <opencv2/opencv.hpp>
```

```
using namespace cv;
using namespace std;
```

```
struct SteganographyData {
    Mat image;
    vector<uint8_t> data;
};
```

```
void embedData(Mat& image, const vector<uint8_t>& data) {
    string binaryData;
    for (const uint8_t byte : data) {
        binaryData += bitset<8>(byte).to_string();
    }
}
```

```

}
binaryData += "00000000"; // Null terminator

size_t dataIndex = 0;
bool done = false;

for (int y = 0; y < image.rows && !done; ++y) {
    for (int x = 0; x < image.cols && !done; ++x) {
        for (int c = 0; c < image.channels(); ++c) {
            Vec3b& pixel = image.at<Vec3b>(Point(x, y));
            if (dataIndex < binaryData.size()) {
                pixel[c] = (pixel[c] & ~1) | (binaryData[dataIndex++] - '0');
            } else {
                done = true;
                break;
            }
        }
    }
}

if (!done) {
    cerr << "Error: The data is too large to be embedded in the image." << endl;
    exit(1);
}
}

```

```

vector<uint8_t> extractData(const Mat& image) {
    string binaryData;
    vector<uint8_t> extractedData;
    uint8_t byte = 0;

    for (int y = 0; y < image.rows; ++y) {
        for (int x = 0; x < image.cols; ++x) {
            for (int c = 0; c < image.channels(); ++c) {
                const Vec3b& pixel = image.at<Vec3b>(Point(x, y));
                binaryData += to_string(pixel[c] & 1);

                if (binaryData.size() == 8) {
                    byte = bitset<8>(binaryData).to_ulong();
                    if (byte == '\0') {
                        return extractedData;
                    }
                    extractedData.push_back(byte);
                    binaryData.clear();
                }
            }
        }
    }

    return extractedData;
}

```

```

void embedOperation(const string& inputImagePath, const string& outputImagePath, const string&

```

```

dataFilePath) {
    Mat image = imread(inputImagePath, IMREAD_COLOR);

    if (!image.data) {
        cerr << "Error: Could not open or find the image." << endl;
        exit(1);
    }

    ifstream dataFile(dataFilePath, ios::binary);
    if (!dataFile.is_open()) {
        cerr << "Error: Could not open the data file." << endl;
        exit(1);
    }

    vector<uint8_t> data((istreambuf_iterator<char>(dataFile), istreambuf_iterator<char>()));
    embedData(image, data);
    imwrite(outputImagePath, image);
    cout << "Data embedded successfully into " << outputImagePath << endl;
}

void extractOperation(const string& inputImagePath, const string& outputDataPath) {
    Mat image = imread(inputImagePath, IMREAD_COLOR);

    if (!image.data) {
        cerr << "Error: Could not open or find the image." << endl;
        exit(1);
    }

    vector<uint8_t> extractedData = extractData(image);
    ofstream outFile(outputDataPath, ios::binary);
    if (outFile.is_open()) {
        outFile.write(reinterpret_cast<const char*>(extractedData.data()), extractedData.size());
        outFile.close();
        cout << "Data extracted successfully to " << outputDataPath << endl;
    } else {
        cerr << "Error: Could not open the output file." << endl;
        exit(1);
    }
}

int main(int argc, char** argv) {
    if (argc < 4) {
        cerr << "Usage: " << argv[0] << " <embed/extract> <input_image> <output_image/data> <data_file (for embed only)>" << endl;
        return 1;
    }

    const string operation = argv[1];
    const string inputImagePath = argv[2];
    const string output = argv[3];

    if (operation == "embed") {
        if (argc < 5) {

```

```

        cerr << "Usage: " << argv[0] << " embed <input_image> <output_image> <data_file>" <<
endl;
        return 1;
    }
    const string dataFilePath = argv[4];
    embedOperation(inputImagePath, output, dataFilePath);
} else if (operation == "extract") {
    extractOperation(inputImagePath, output);
} else {
    cerr << "Error: Unknown operation " << operation << endl;
    return 1;
}

return 0;
}

```

obtainhigh - July 4, 2024, 2:59 am

Hmm....

Here try this. Error free.

PHP Code

```

<?php
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <bitset>
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

void embedData(Mat& image, const vector<uint8_t>& data) {
    string binaryData;
    for (const uint8_t byte : data) {
        binaryData += bitset<8>(byte).to_string();
    }
    binaryData += "00000000"; // Null terminator

    size_t dataIndex = 0;

    for (int y = 0; y < image.rows; ++y) {
        for (int x = 0; x < image.cols; ++x) {
            for (int c = 0; c < image.channels(); ++c) {
                Vec3b& pixel = image.at<Vec3b>(Point(x, y));
                if (dataIndex < binaryData.size()) {
                    pixel[c] = (pixel[c] & ~1) | (binaryData[dataIndex++] - '0');
                } else {
                    return;
                }
            }
        }
    }
}

```

```

    }

    cerr << "Error: The data is too large to be embedded in the image." << endl;
    exit(1);
}

vector<uint8_t> extractData(const Mat& image) {
    string binaryData;
    vector<uint8_t> extractedData;

    for (int y = 0; y < image.rows; ++y) {
        for (int x = 0; x < image.cols; ++x) {
            for (int c = 0; c < image.channels(); ++c) {
                const Vec3b& pixel = image.at<Vec3b>(Point(x, y));
                binaryData += to_string(pixel[c] & 1);

                if (binaryData.size() == 8) {
                    uint8_t byte = bitset<8>(binaryData).to_ulong();
                    if (byte == '\0') {
                        return extractedData;
                    }
                    extractedData.push_back(byte);
                    binaryData.clear();
                }
            }
        }
    }

    return extractedData;
}

void embedOperation(const string& inputImagePath, const string& outputImagePath, const string&
dataFilePath) {
    Mat image = imread(inputImagePath, IMREAD_COLOR);

    if (!image.data) {
        cerr << "Error: Could not open or find the image." << endl;
        exit(1);
    }

    ifstream dataFile(dataFilePath, ios::binary);
    if (!dataFile.is_open()) {
        cerr << "Error: Could not open the data file." << endl;
        exit(1);
    }

    vector<uint8_t> data((istreambuf_iterator<char>(dataFile)), istreambuf_iterator<char>());
    embedData(image, data);
    imwrite(outputImagePath, image);
    cout << "Data embedded successfully into " << outputImagePath << endl;
}

void extractOperation(const string& inputImagePath, const string& outputDataPath) {

```

```

Mat image = imread(inputImagePath, IMREAD_COLOR);

if (!image.data) {
    cerr << "Error: Could not open or find the image." << endl;
    exit(1);
}

vector<uint8_t> extractedData = extractData(image);
ofstream outFile(outputDataPath, ios::binary);
if (outFile.is_open()) {
    outFile.write(reinterpret_cast<const char*>(extractedData.data()), extractedData.size());
    outFile.close();
    cout << "Data extracted successfully to " << outputDataPath << endl;
} else {
    cerr << "Error: Could not open the output file." << endl;
    exit(1);
}
}

int main(int argc, char** argv) {
    if (argc < 4) {
        cerr << "Usage: " << argv[0] <<
" <embed/extract> <input_image> <output_image/data> <data_file (for embed only)>" << endl;
        return 1;
    }

    string operation = argv[1];
    string inputImagePath = argv[2];
    string output = argv[3];

    if (operation == "embed") {
        if (argc < 5) {
            cerr << "Usage: " << argv[0] << " embed <input_image> <output_image> <data_file>" <<
endl;
            return 1;
        }
        string dataFilePath = argv[4];
        embedOperation(inputImagePath, output, dataFilePath);
    } else if (operation == "extract") {
        extractOperation(inputImagePath, output);
    } else {
        cerr << "Error: Unknown operation " << operation << endl;
        return 1;
    }

    return 0;
}

```

YAK_CODER - July 4, 2024, 1:15 pm

imagine i did use Steganography on yak crypter since 2020

<https://malcat.fr/blog/exploit-steganography-and-delphi-unpacking-dbatloader/>

secret_code - July 5, 2024, 1:36 am

as specified already, it wasn't something for real life usage, just pseudo code for folks who don't understand steganography in c++, a hint...

(Jul 4, 2024 02:59 AM)obtainhigh Wrote: [\[CITE\]](#) Hmm....

Here try this. Error free.

PHP Code

```
<?php
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <bitset>
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

void embedData(Mat& image, const vector<uint8_t>& data) {
    string binaryData;
    for (const uint8_t byte : data) {
        binaryData += bitset<8>(byte).to_string();
    }
    binaryData += "00000000"; // Null terminator

    size_t dataIndex = 0;

    for (int y = 0; y < image.rows; ++y) {
        for (int x = 0; x < image.cols; ++x) {
            for (int c = 0; c < image.channels(); ++c) {
                Vec3b& pixel = image.at<Vec3b>(Point(x, y));
                if (dataIndex < binaryData.size()) {
                    pixel[c] = (pixel[c] & ~1) | (binaryData[dataIndex++] - '0');
                } else {
                    return;
                }
            }
        }
    }

    cerr << "Error: The data is too large to be embedded in the image." << endl;
    exit(1);
}

vector<uint8_t> extractData(const Mat& image) {
    string binaryData;
    vector<uint8_t> extractedData;

    for (int y = 0; y < image.rows; ++y) {
        for (int x = 0; x < image.cols; ++x) {
            for (int c = 0; c < image.channels(); ++c) {
```

```

        const Vec3b& pixel = image.at<Vec3b>(Point(x, y));
        binaryData += to_string(pixel[c] & 1);

        if (binaryData.size() == 8) {
            uint8_t byte = bitset<8>(binaryData).to_ulong();
            if (byte == '\0') {
                return extractedData;
            }
            extractedData.push_back(byte);
            binaryData.clear();
        }
    }
}

return extractedData;
}

void embedOperation(const string& inputImagePath, const string& outputImagePath, const string&
dataFilePath) {
    Mat image = imread(inputImagePath, IMREAD_COLOR);

    if (!image.data) {
        cerr << "Error: Could not open or find the image." << endl;
        exit(1);
    }

    ifstream dataFile(dataFilePath, ios::binary);
    if (!dataFile.is_open()) {
        cerr << "Error: Could not open the data file." << endl;
        exit(1);
    }

    vector<uint8_t> data((istreambuf_iterator<char>(dataFile)), istreambuf_iterator<char>());
    embedData(image, data);
    imwrite(outputImagePath, image);
    cout << "Data embedded successfully into " << outputImagePath << endl;
}

void extractOperation(const string& inputImagePath, const string& outputDataPath) {
    Mat image = imread(inputImagePath, IMREAD_COLOR);

    if (!image.data) {
        cerr << "Error: Could not open or find the image." << endl;
        exit(1);
    }

    vector<uint8_t> extractedData = extractData(image);
    ofstream outFile(outputDataPath, ios::binary);
    if (outFile.is_open()) {
        outFile.write(reinterpret_cast<const char*>(extractedData.data()), extractedData.size());
        outFile.close();
        cout << "Data extracted successfully to " << outputDataPath << endl;
    }
}

```

```

    } else {
        cerr << "Error: Could not open the output file." << endl;
        exit(1);
    }
}

int main(int argc, char** argv) {
    if (argc < 4) {
        cerr << "Usage: " << argv[0] <<
" <embed/extract> <input_image> <output_image/data> <data_file (for embed only)>" << endl;
        return 1;
    }

    string operation = argv[1];
    string inputImagePath = argv[2];
    string output = argv[3];

    if (operation == "embed") {
        if (argc < 5) {
            cerr << "Usage: " << argv[0] << " embed <input_image> <output_image> <data_file>" <<
endl;
            return 1;
        }
        string dataFilePath = argv[4];
        embedOperation(inputImagePath, output, dataFilePath);
    } else if (operation == "extract") {
        extractOperation(inputImagePath, output);
    } else {
        cerr << "Error: Unknown operation " << operation << endl;
        return 1;
    }

    return 0;
}

```

hey, thanks for correcting the shit i wrong :), yeah i have made several errors due to rushing, thank you but at least use your real code and not AI generated bullshit :)