Automation of Analysis with IDAPython

Objective

Master the automation of reverse engineering using **IDAPython** – a Python scripting language integrated into IDA Pro for working with disassembled code. This practical work covers:

- Searching for specific code patterns.
- Extracting functions and their dependencies.
- Automated function call analysis.

Required Tools

- **IDA Pro** (Interactive Disassembler)
- **Python (IDAPython)** – an embedded Python interpreter in IDA Pro
- **Test binary file (test_binary.exe)** – any executable file can be used for experiments.

Theoretical Background

IDAPython provides an API for automating analysis, including:

- `idautils` – utilities for working with functions, references, and data.
- `idc` – low-level commands for controlling IDA Pro.
- `idaapi` – access to IDA Pro objects.

Tasks

Task 1: Finding Calls to a Specific Function

Objective: Identify all calls to a target function (e.g., `strcpy`) in a binary file.

Code:

```
from idautils import CodeRefsTo
from idc import LocByName, BADADDR, msg

def find_calls_to(func_name):
    """Finds all calls to a given function"""
    addr = LocByName(func_name)  # Get function address
    if addr == BADADDR:
        msg(f"[ERROR] Function {func_name} not found.\n")
        return
    msg(f"Searching for calls to {func_name}...\n")
    for ref in CodeRefsTo(addr, 0):
        msg(f" - {func_name} called at {hex(ref)}\n")

# Call the function to search for strcpy
find_calls_to("strcpy")
```

Code Explanation:

1. `LocByName(func_name)`: Retrieves the function's address in memory.
2. `CodeRefsTo(addr, 0)`: Searches for all references to this function.
3. Outputs addresses where the function is called.

Expected Result:

The script will output a list of addresses where `strcpy` is called.

Task 2: Extracting All Functions from a Binary File

Objective: Retrieve a list of all functions, their addresses, and sizes in bytes.

Code:

```
from idautils import Functions
from idc import GetFunctionName, GetFunctionAttr, FUNCATTR_START,
FUNCATTR_END, msg

def list_functions():
    """Outputs a list of all functions in the binary."""
    msg("\n[INFO] Extracting all functions:\n")
    for func in Functions():
        name = GetFunctionName(func)
        start = GetFunctionAttr(func, FUNCATTR_START)
        end = GetFunctionAttr(func, FUNCATTR_END)
        msg(f"Function: {name}, Start: {hex(start)}, End: {hex(end)}, Size:
{end-start} bytes\n")

list_functions()
```

Code Explanation:

1. `Functions()`: Iterates through all functions in the binary.
2. `GetFunctionName(func)`: Retrieves the function name.
3. `GetFunctionAttr(func, FUNCATTR_START/END)`: Retrieves the start and end addresses of the function.

Expected Result:

The script will output a list of all functions in the executable file with their addresses and sizes.

Task 3: Finding Dangerous Functions

Objective: Identify the use of potentially vulnerable functions (`strcpy`, `system`, `gets`).

Code:

```
from idautils import CodeRefsTo
from idc import LocByName, BADADDR, msg

def find_dangerous_functions():
    """Finds the use of dangerous functions"""
    dangerous_funcs = ["strcpy", "gets", "system", "memcpy"]
    for func in dangerous_funcs:
        addr = LocByName(func)
        if addr == BADADDR:
            continue
        msg(f"[INFO] Searching for {func}...\n")
        for ref in CodeRefsTo(addr, 0):
            msg(f" - {func} called at {hex(ref)}\n")

find_dangerous_functions()
```

Code Explanation:

1. Creates a list of dangerous functions (`dangerous_funcs`).
2. Searches for calls to each function in the binary (`CodeRefsTo`).

Expected Result:

A list of all detected calls to potentially dangerous functions.

Task 4: Extracting Strings from a Binary

Objective: Find all strings in a binary file that may contain useful information (passwords, paths, encryption keys).

Code:

```
from idautils import Strings
from idc import msg

def extract_strings():
    """Extracts all strings from the binary."""
    msg("\n[INFO] Extracting strings:\n")
    for s in Strings():
        msg(f"Address: {hex(s.ea)}, String: {s.string}\n")

extract_strings()
```

Code Explanation:

1. `Strings()`: Iterates through all text strings in the binary.
2. `s.ea` – Memory address of the string.
3. `s.string` – The extracted string.

Expected Result:

The script will output the strings found in the binary.

Additional Tasks

1. **Automatic Function Annotation:** Write a script that labels unknown functions based on code patterns.

2. **Analyzing Function Dependencies:** Build a call graph using NetworkX.
3. **Bypassing Anti-Debugging Techniques:** Detect and disable checks like `IsDebuggerPresent`, `NtQueryInformationProcess`.

References

1. **Chris Eagle – "The IDA Pro Book"**
2. **Official IDAPython Documentation** (https://hex-rays.com/products/ida/support/idapython_docs/)