AIM: Analyze malware in Cuckoo Sandbox, using IDA Pro or Ghidra for reverse engineering.

DESCRIPTION:

Malware analysis combines dynamic sandboxing and static reverse engineering to understand what a suspicious binary does, how it persists, and what indicators it produces. Use **Cuckoo Sandbox** for automated dynamic analysis (behavior, network IOCs, dropped files) and **IDA Pro** or **Ghidra** for deep static analysis (disassembly, decompilation, code paths). Always perform analysis in an isolated, air-gapped lab with snapshots and legal authorization.

PROCEDURE:

1. **Download IDA software from the following website -** https://hex-rays.com/ida-free
2. **Download a sample binary file, or write a sample C file and compile it.**

Code –

```c
#include <stdio.h>
#include <string.h>

int secret_math(int x) {
    int a = 1234;
    int b = 5678;
    return (x * a) ^ b;
}

int check_password(const char *pw) {
    const char *correct = "banana42";

    int fake = 0;
    for (int i = 0; pw[i]; i++) {
        fake ^= pw[i] * (i + 1);
    }

    if (fake == 0xdeadbeef) {
        return 0; // fake branch to confuse
    }

    return strcmp(pw, correct) == 0;
}

int main() {
    char buf[64];

    puts("x86_64 Reverse Engineering Sample");

    printf("Enter password: ");
    scanf("%63s", buf);

    if (check_password(buf)) {
        printf("Access granted. Value = %d\n", secret_math(5));
    } else {
        printf("Access denied.\n");
    }}
```

3. **Save the Code to `sample.c` and compile the code using the command - ` clang -arch x86_64 -O0 -g -o sample sample.c `**
4. Open IDA Pro and load the `sample` binary
5. Identify the entry point if it doesn't automatically detect it. Auto-analysis will be run.
6. Analyze different components of the reverse-engineering like the Output, Graph overview, function graph, imports, exports, functions and strings.
7. Identify techniques like obfuscation, API calls, encrypted/compressed blocks.

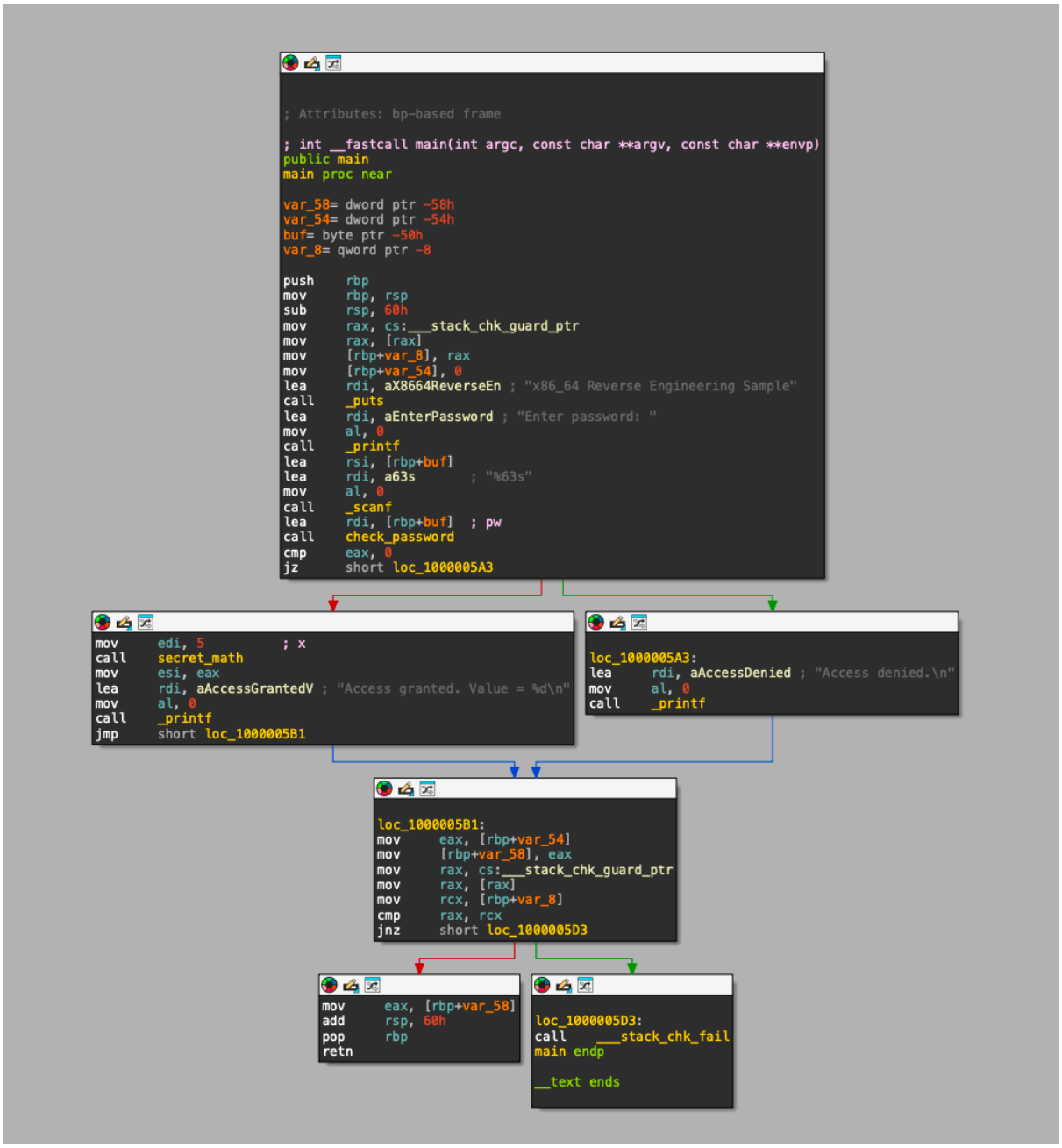
Auto-Analysis Output

Possible file format: Mach-O file (EXECUTE). X86_64 (/Applications/IDA Free 9.2.app/Contents/MacOS/loaders/macho.dylib)

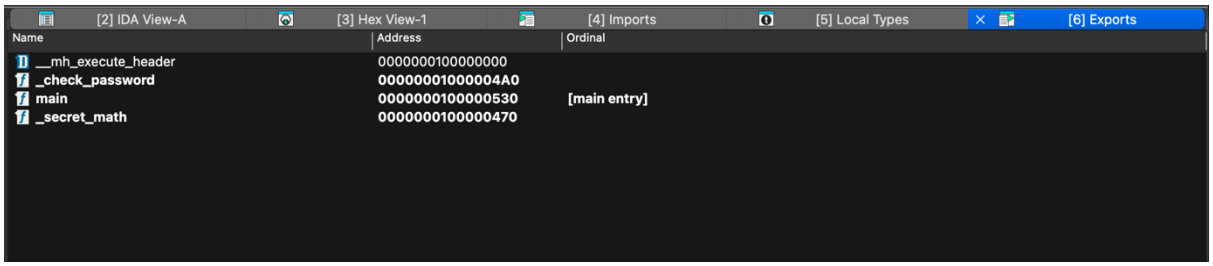
```
  bytes   pages size description
--------- ----- ---- --------------------------------------------
   524288    64 8192 allocating memory for b-tree...
    65536     8 8192 allocating memory for virtual array...
   262144    32 8192 allocating memory for name pointers...
-------------------------------------------------------------------
   851968            total memory allocated

Loading processor module /Applications/IDA Free 9.2.app/Contents/MacOS/procs/pc.dylib for metapc...OK
Autoanalysis subsystem has been initialized.
Loading file '/Users/omkarkabde/Desktop/sample_x86' into database...
Detected file format: Mach-O file (EXECUTE). X86_64
Plugin "swift" not found
Type library 'macosx64' loaded. Applying types...
Types applied to 0 names.
  0. Creating a new segment  (0000000100000470-00000001000005D8) ... ... OK
  1. Creating a new segment  (00000001000005D8-00000001000005F6) ... ... OK
  2. Creating a new segment  (00000001000005F6-0000000100000664) ... ... OK
  3. Creating a new segment  (0000000100000664-0000000100001000) ... ... OK
  4. Creating a new segment  (0000000100000000-0000000100000470) ... ... OK
  5. Creating a new segment  (0000000100001000-0000000100001030) ... ... OK
  6. Creating a new segment  (0000000100001030-0000000100001060) ... ... OK
OBJC: No objc info found
Marking typical code sequences...
Flushing buffers, please wait...ok
File '/Users/omkarkabde/Desktop/sample_x86' has been successfully loaded into the database.
DWARF: File "/Users/omkarkabde/Desktop/sample_x86.dSYM/Contents/Resources/DWARF/sample_x86" contains DWARF information.
```

DWARF: Functions: 3 symbols applied
DWARF: Globals: 0 symbols applied
Flushing buffers, please wait...ok
Hex-Rays Cloud Decompiler plugin has been loaded (v9.2.0.250908)
  The decompilation hotkey is F5.
  Please check the Edit/Plugins menu for more information.
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.

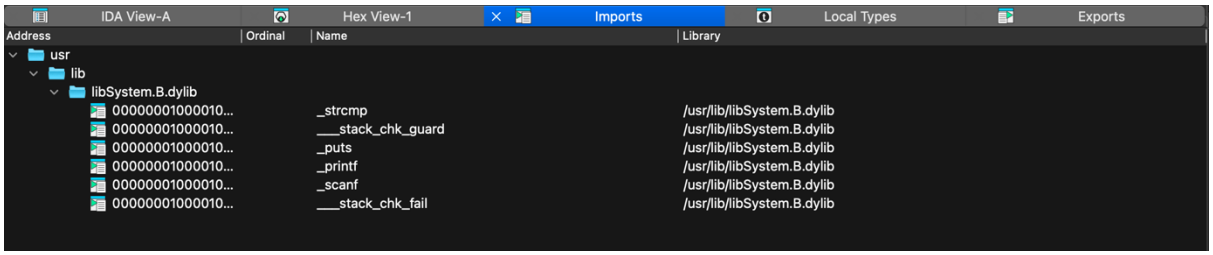
Graph Overview of the reverse-engineering



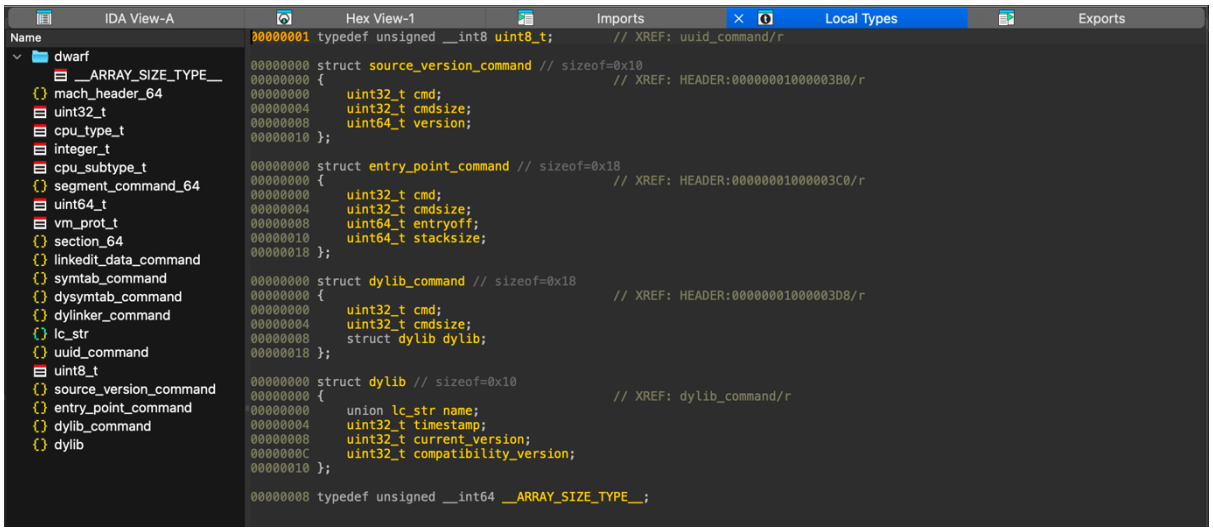
Exports View



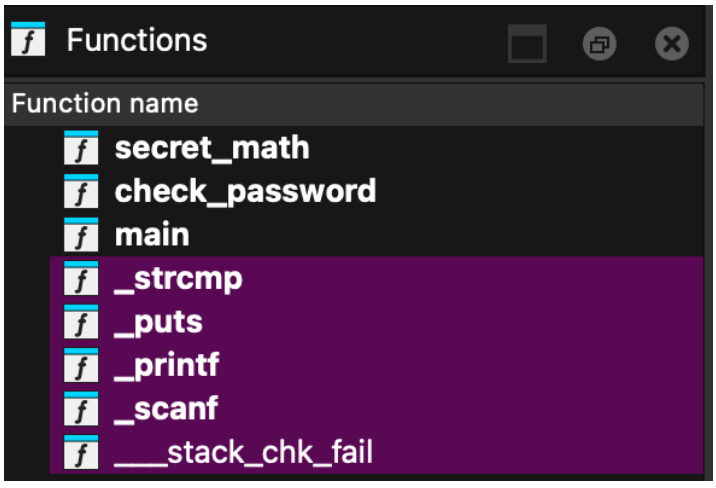
Imports View



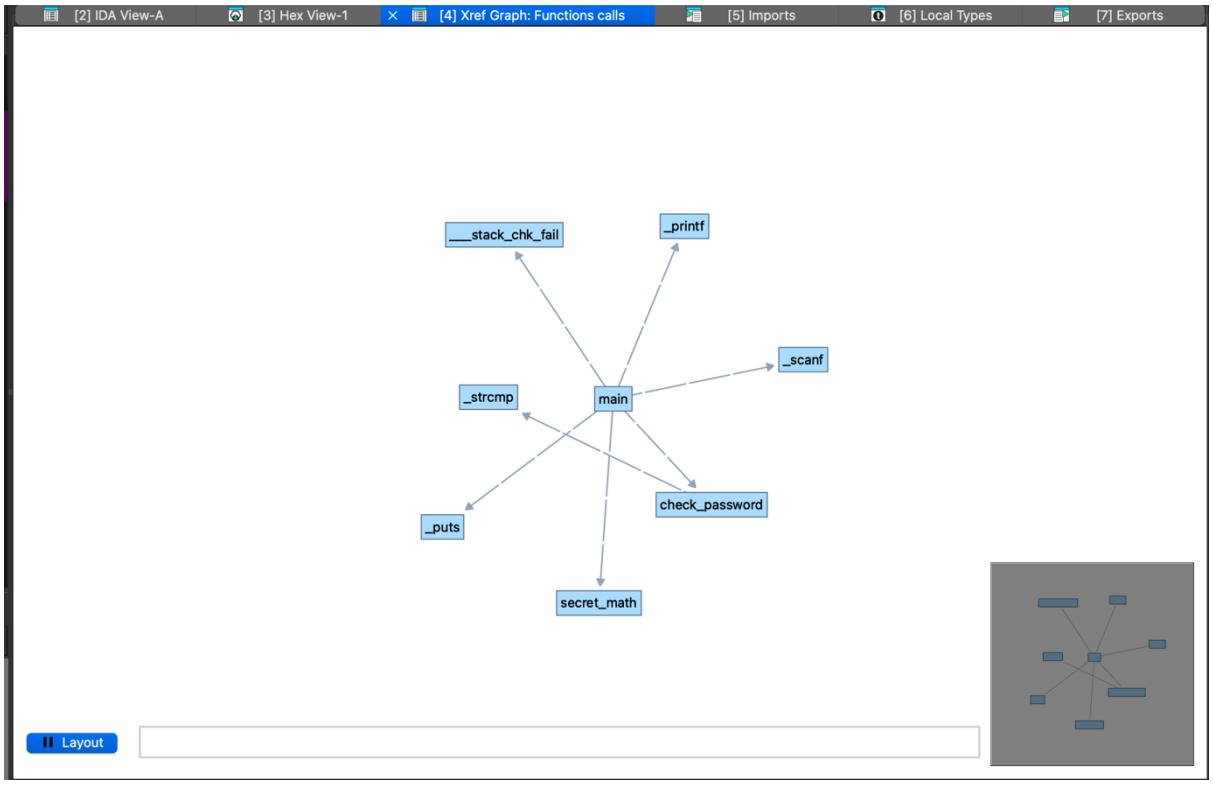
Local Types



Functions



Graph for Function Calls



CONCLUSION:

Using **IDA Pro** for static malware reverse engineering provides deep insights into a binary's capabilities, including persistence techniques, process injection, network communication, encryption, and anti-analysis behavior. By systematically analyzing imports, strings, API usage, control flow, and decompiled functions, analysts can derive comprehensive IOCs and technical understanding of malicious behavior.