Одеський національний політехнічний університет
Кафедра кібербезпеки та програмного забезпечення

Дослідження методів евристичного аналізу шкідливого програмного забезпечення

Виконав ст. Смагін А.А.
Керівник: Кушніренко Н.І.

Одеса 2020

Мета

Метою роботи є підвищення ефективності виявляння шкідливого програмного забезпечення на основі методів евристичного аналізу

Об’єкт і предмет

- Об’єктом дослідження є процес забезпечення захисту інформаційних систем від впровадження та запуску шкідливого програмного забезпечення
- Предмет дослідження є методи детектування шкідливих програм на основі евристичного аналізу

Задачі

- Проаналізувати основні типи шкідливого програмного забезпечення
- Дослідити ефективність існуючих методів и засобів захисту від шкідливих програм
- Розробити метод детектування шкідливого програмного забезпечення на основі евристичного аналізу
- Запропонувати методику виявлення шкідливих програм на основі поєднання існуючих підходів та власного метода
- Провести аналіз ефективності запропонованої методики виявлення шкідливого програмного забезпечення на основі евристичного аналізу

Наукова новизна

- Розроблено метод детектування запису зашифрованого файлу.
- Створено власну інтегральну методику ефективного виявлення шкідливих програм.
- Проведена успішна програмна реалізація запропонованої методики з переносом обчислень на відеокарту для пришвидшення процесу детектування шкідливих програм.

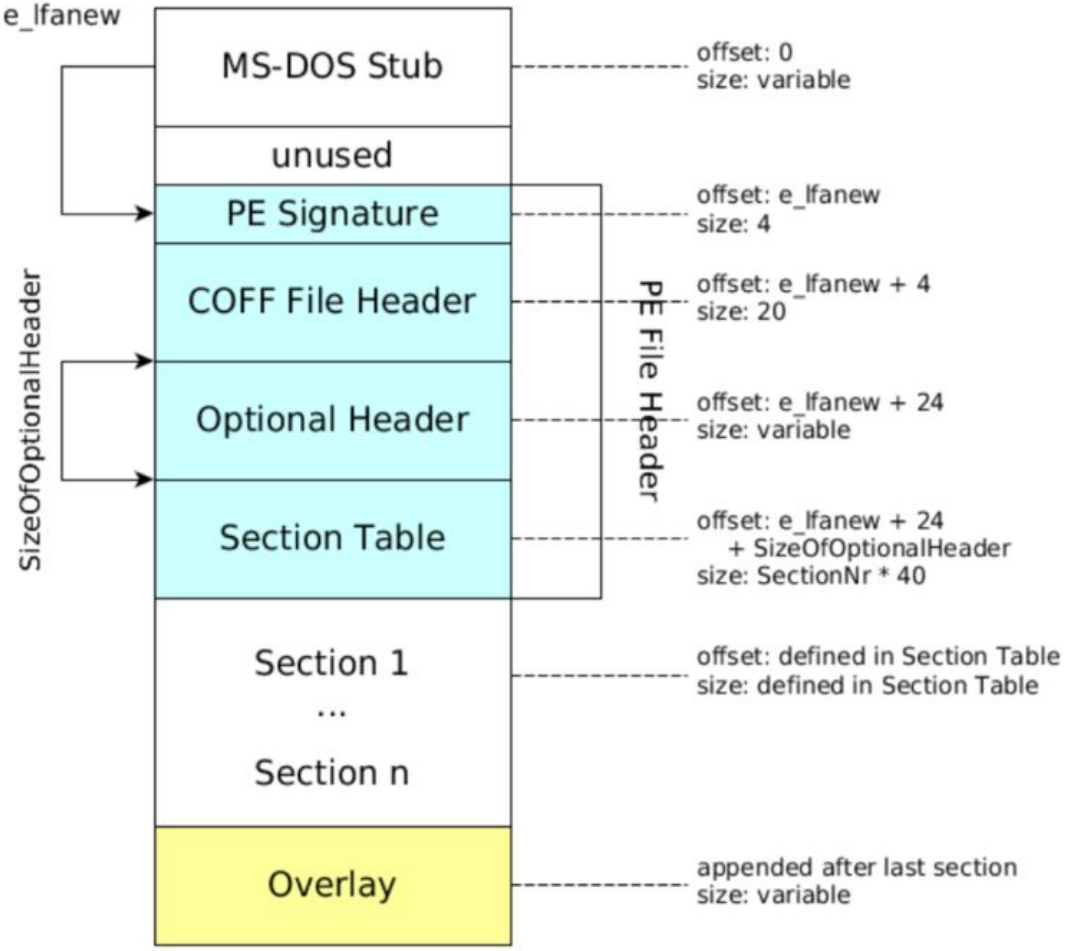
На основі розробленої методики аналізу та детектування шкідливих програм, створено програмне забезпечення, яке дозволяє на основі набору ознак класифікувати файли на зловмисні та безпечні.

Види аналізу шкідливого ПЗ

- Статичний
- Динамічний
- Евристичний

Процес аналізу файлу

Структура PE файлу

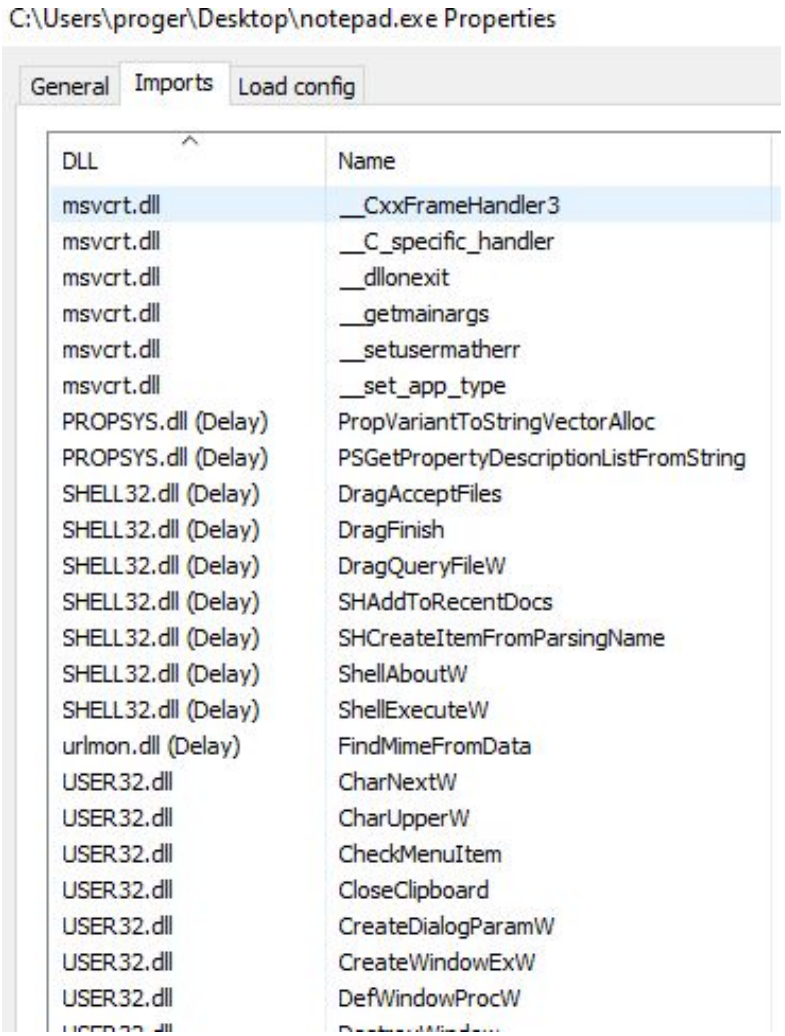


Список секцій файлу

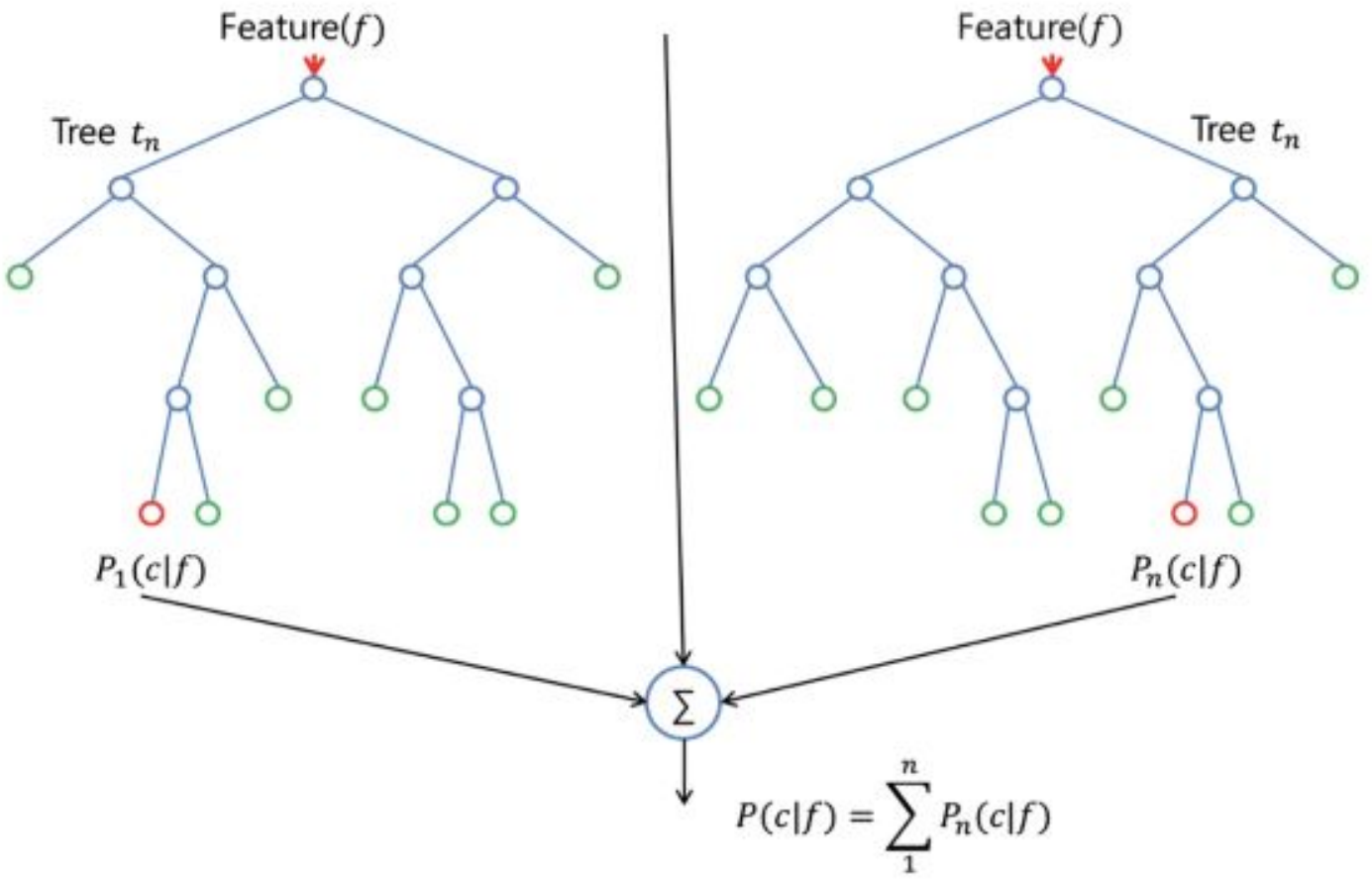
Sections:

Name	VA	Size
.text	0x1000	0x20600
.rdata	0x22000	0x8c00
.data	0x2b000	0xc00
.pdata	0x2e000	0x1000
.didat	0x2f000	0x200
.rsrc	0x30000	0xc00
.reloc	0x31000	0x400

Список таблиці імпорту



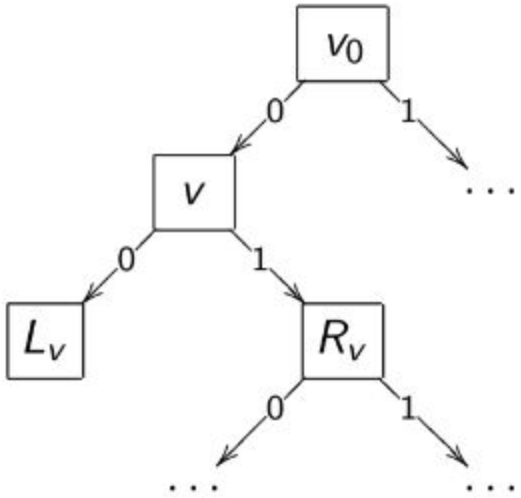
Випадковий ліс



Основні методи

- Обфускація
- Шифрування
- Перевірка цілісності
- Виявлення відлагоджувача
- Виконання коду на віртуальній машині

Дерево рішень та пошук вузлу

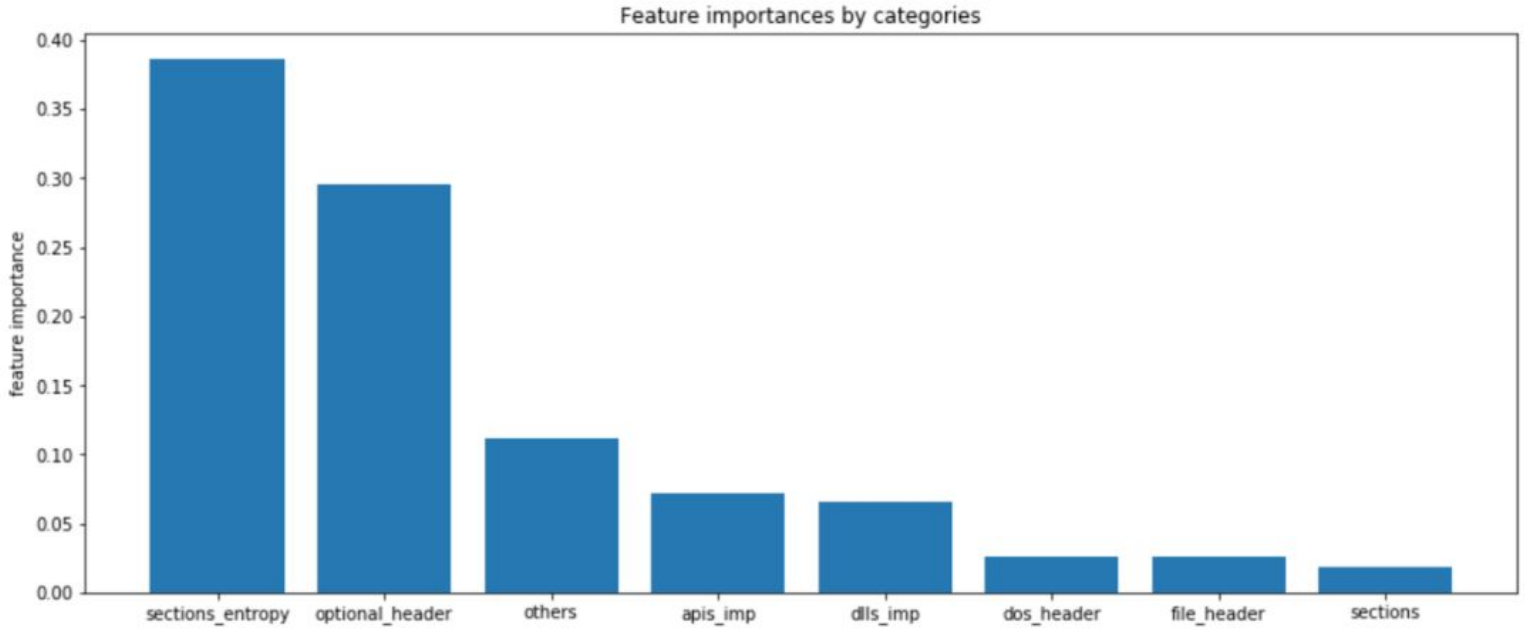


while $v \in V_{\text{внутр}}$ **do**
  if $\beta_v(x) = 1$ **then**
    go to right;
    $v := R_v$;
  else
    go to left;
    $v := L_v$;
  end
  return c_v
end

Обфускація тексту програми

```
int main(int argc, char* argv[]) {
     int c = 89;                  // змінна для відволікання
     int i[] = { 1,2,3,4 };   // ця теж
label1:;
     if (c > 100) {          // цей код ніколи не виконається
     label2:;
          foo1(c);
     }
     else if (c > 100 && c < 200) { // і цей теж
          foo2(i,c + 29);
          goto label2;
     }
     else {
          if (c > 89)
               goto end;
     label3:;
          foo2(i, 9);                  //
          goto label1;
     }
     goto label3;
end:;
     foo2(i, 4);
     printf("hello world!\n");
     return 0;
}
```

Оцінка важливості ознак



Шифрування

```
; Attributes: bp-based frame

; int __cdecl main(int argc, char **argv)
_main proc near

str= dword ptr -4
argc= dword ptr  8
argv= dword ptr  0Ch

push    ebp
mov     ebp, esp
push    ecx
mov     [ebp+str], offset aJgnnq ; "jgnnq"
mov     eax, [ebp+str]
push    eax             ; text
call    _cesar
add     esp, 4
mov     ecx, [ebp+str]
push    ecx             ; _Format
call    _printf
add     esp, 4
xor     eax, eax
mov     esp, ebp
pop     ebp
retn
_main endp
```

```
; Attributes: bp-based frame

; int __cdecl main(int argc, char **argv)
_main proc near

str= dword ptr -4
argc= dword ptr  8
argv= dword ptr  0Ch

push    ebp
mov     ebp, esp
push    ecx
mov     [ebp+str], offset aHello ; "hello"
mov     eax, [ebp+str]
push    eax             ; _Format
call    _printf
add     esp, 4
xor     eax, eax
mov     esp, ebp
pop     ebp
retn
_main endp
```

Точка зупинки

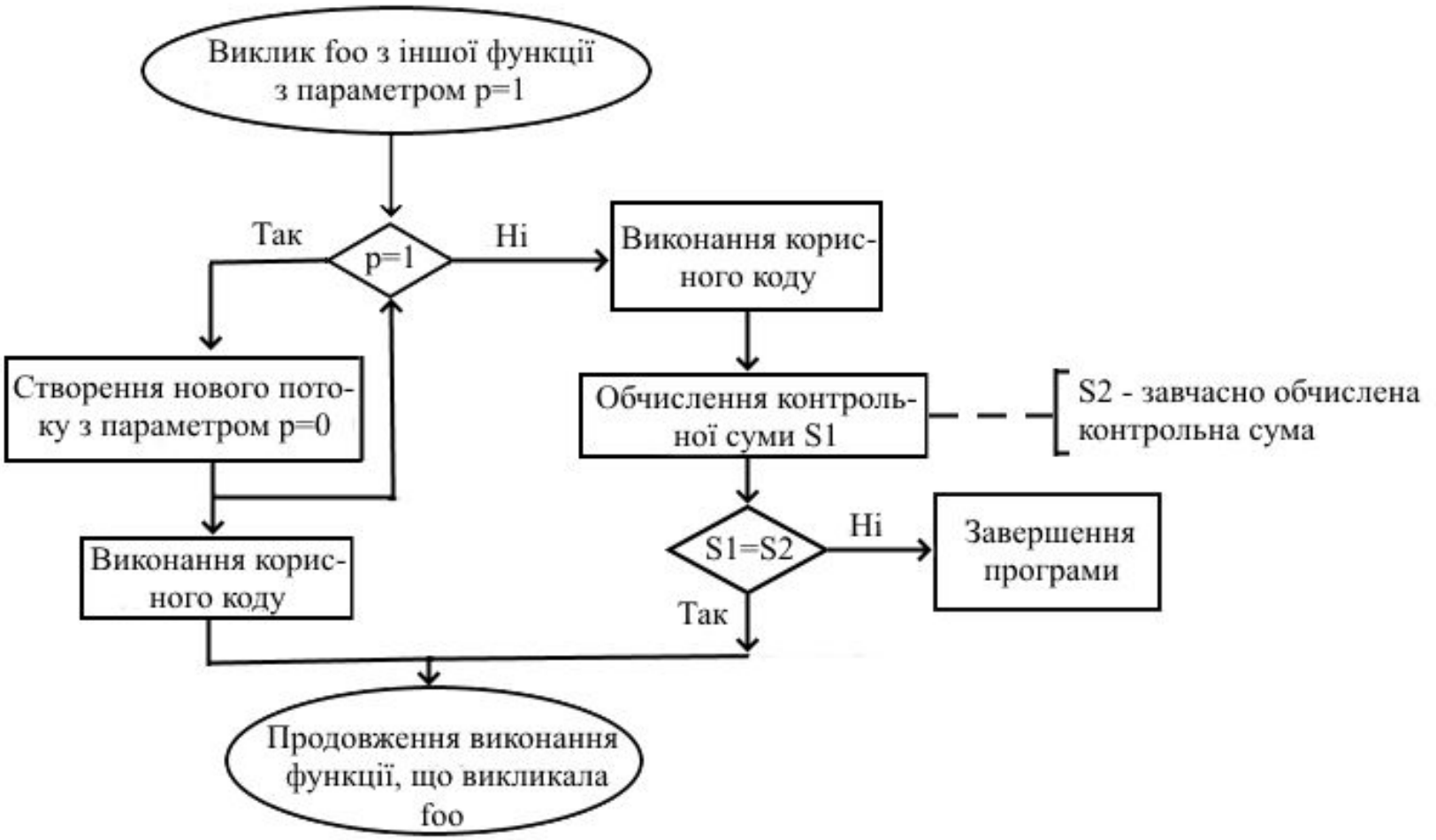
```
foo:
        push    ebp
        mov     ebp, esp
        mov     eax, [ebp + a]
        add     eax, [ebp + b]
        pop     ebp
        retn
```

(а)

```
foo:
        push    ebp
        mov     ebp, esp
        int       3h
        mov     eax, [ebp + a]
        add     eax, [ebp + b]
        pop     ebp
        retn
```

(б)

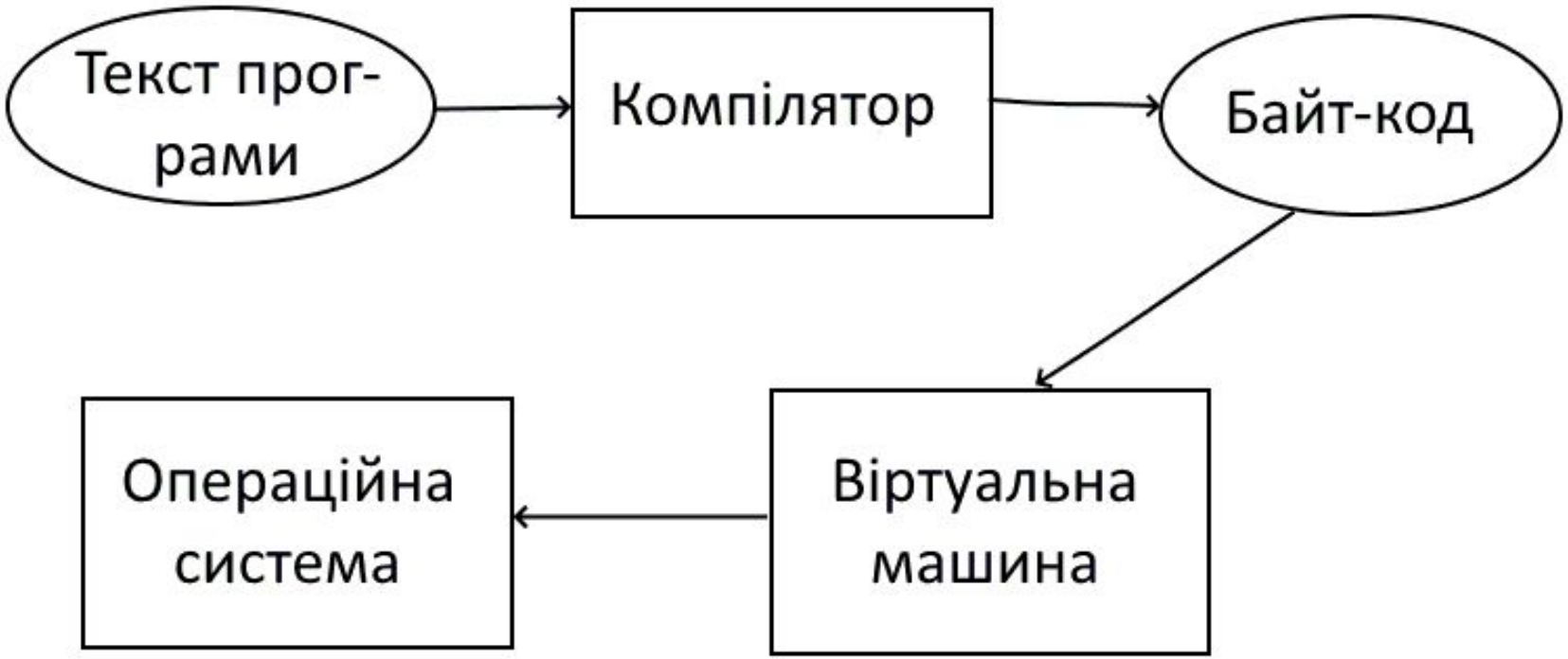
Перевірка цілісності



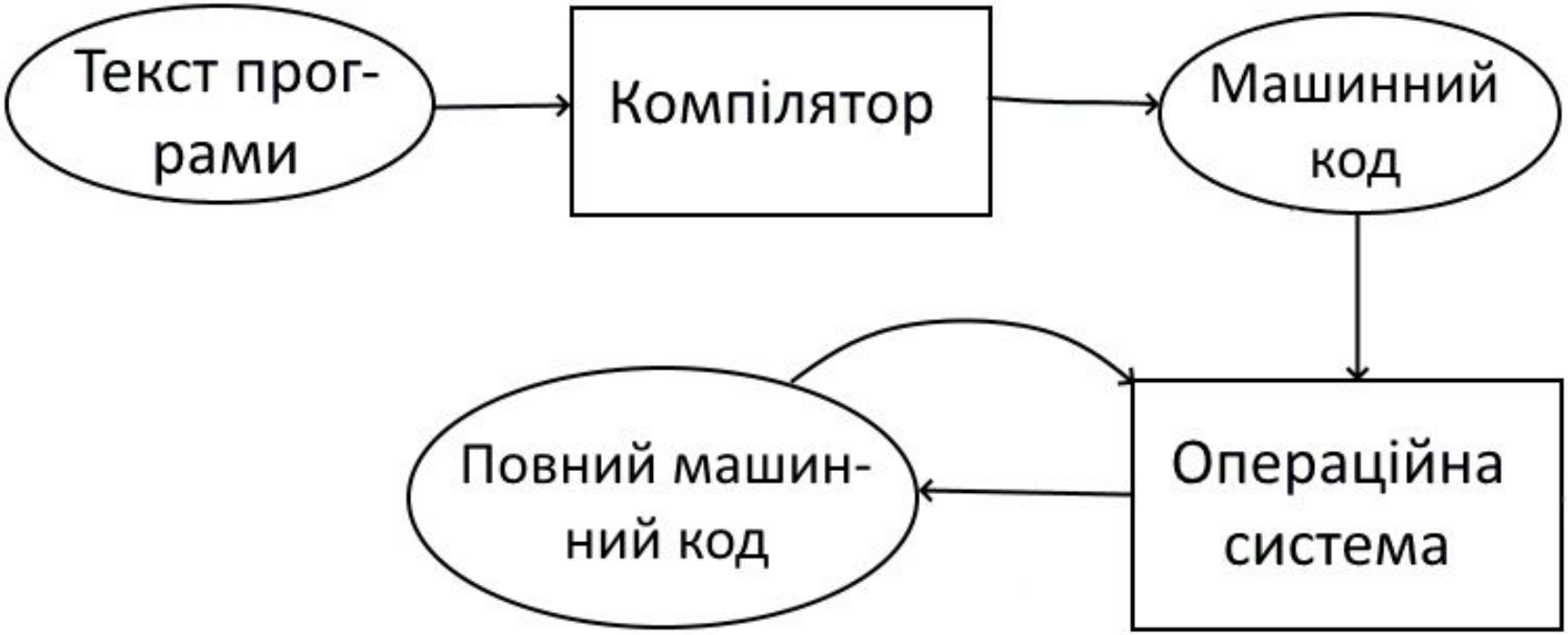
Перевірка цілісності

```
#pragma auto_inline(off)
void foo(int p,void* end_address) {
     if (p == 1) {
          thread th(foo, 0,end_address);
          printf("Виконується якась корисна робота\n");
          th.detach();
     }
     else {
          char hash[32] =              // наперед обчислений
                                       // хеш функції
          { 0xd7,0xa8,0xfb,0xb3,0x07,0xd7,0x80,0x94,0x69,
          0xca,0x9a,0xbc,0xb00,0x82,0xe4f,0x8d,0x56,0x51,
          0xe46,0xd3,0xcd,0xb7,0x62 };

          char calculated_hash[32]; // сюди запишеться хеш,
                                    // який ми обчислюємо
          // обчислення хешу
          sha256_string(foo,end_address,calculated_hash);

          for(int i = 0;i < 32;i++){
               if (hash[i] != calculated_hash[i]) {
                                    // якщо контрольні суми не
                    int i = 0;      // збігаються, то створюємо
                    i = i / i;      // аварійну ситуацію для
                                    // завершення програми
               }
          }
     }
}
void foo_end()
{
};
#pragma auto_inline(on)

int main(){
     setlocale(LC_ALL, "Russian"); //
```

Віртуальна машина



Кодогенерація на льоту (asmjit)



Виявлення налагоджувача

Багато способів:

- IsDebuggerPresent
- NtSetInformationThread (HideFromDebugger)
- NtCreateThreadEx(THREAD_CREATE_FLAGS_HIDE_FROM_DEBUGGER)
- NtQueryInformationProcess (ProcessBasicInformation)
- NtQueryInformationProcess (ProcessInformationClass)
- NtGlobalFlag

Низька ефективність та проста реалізація

Висновки

1. Проаналізовано актуальні на даний час методи та на основі таких критеріїв як швидкодія, відсутність необхідності запуску підозрілого файлу та точність
2. Розроблено новий метод виявлення зловмисних програм шифрувальнків
3. Розроблена методика кінцевої оцінки вірогідності зловмисності файлу на основі вибраних методів та алгоритму машинного навчання «випадковий ліс».
4. Проведена оцінка точності результатів та інтенсивності використання системних ресурсів.
5. Показана кількісна оцінка важливості кожного з методів, що використовується в методиці