Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Навчально-науковий фізико технічний інститут
Кафедра інформаційної безпеки

Звіт

З практичного завдання №4
із дисципліни «Методи реалізації криптографічних механізмів»

Тема: «Дослідження особливостей реалізації існуючих програмних систем, які використовують криптографічні механізми захисту інформації»

Виконав:
Студент групи ФБ-41мн
Шерстюк А. В.
Вигрaновський М.В

Київ-2025

Реалізація асиметричної криптосистеми під Linux платформу. Стандарт ECDSA

Дослідження стійкості криптопровайдера до атак, що виникають через недоліки захисту операційної системи, показує: навіть бездоганна з математичного погляду реалізація ECDSA не рятує, якщо саме оточення скомпрометоване. Надійність криптопровайдера залежить не лише від математичної моделі, але й від цілого комплексу умов, що включають безпечне зберігання ключів та налаштування доступу в ОС.

Загрози та вразливості криптопровайдера:

1. **Викрадення приватного ключа**: Найочевидніша загроза- крадіжка приватного ключа з файлової системи. Якщо файл, наприклад, ecdsa_private.pem, зберігається відкрито без належних обмежень доступу, будь-який привілейований користувач або процес може прочитати його.
2. **Компрометація генератора випадкових чисел** (ГВЧ): Стійкість ECDSA критично залежить від одноразового параметра k. Якщо зловмисник може вплинути на стан ГВЧ (наприклад, підміною джерел ентропії або використанням передбачуваних seed-значень), алгоритм може згенерувати однаковий k кілька разів. Повторення k миттєво розкриває приватний ключ d.
3. **Атаки через бічні канали**: У нативних C-реалізаціях можливі витоки інформації через час виконання, енергоспоживання чи поведінку кешу. Хоча Python (зокрема, PyCryptodome) згладжує частину цих ризиків, його скомпільовані модулі залишаються вразливими до низькорівневих операцій.
4. **Підміна бібліотек**: Атака через LD_PRELOAD дозволяє підвантажити власний шкідливий код або підмінити криптографічні бібліотеки, щоб скопіювати ключі та підписи ще до запуску справжніх криптографічних функцій.
5. **Читання пам'яті процесу**: Зловмисник може отримати тимчасові значення, такі як r, h або сам d, під час підпису, якщо система дозволяє читати пам'ять процесу (наприклад, через /proc або механізми на зразок ptrace).

Типові сценарії атак

Сценарій атаки	Механізм недосконалості ОС	Мета атаки
Атака на "холодну" пам'ять	Відсутність повного очищення оперативної пам'яті (RAM) після вимкнення живлення.	Зчитування закритого ключа (d) з дампів пам'яті.
Атака на свопінг	Запис чутливих даних процесу (включаючи закриті ключі) у	Вилучення ключа з нешифрованого файлу підкачки.

	файл підкачки (swapfile) на диску.	
Атака за допомогою ptrace	Дозволяє іншому процесу (з відповідними правами) перевіряти та змінювати пам'ять цільового процесу.	Зчитування закритого ключа з адресного простору процесу підписання.
Атака на /proc файлову систему	Можливість доступу до /proc/$PID/mem для читання пам'яті процесу.	Зчитування вмісту пам'яті процесу підписання.

Практична реалізація

Для демонстрації вразливості, пов'язаної з архітектурою ОС Linux та керуванням пам'яттю, було обрано атаку за допомогою механізму **ptrace** (з використанням GDB). Цей механізм дозволяє процесу-атакуючому приєднатися до цільового процесу (де відбувається підпис) і отримати доступ до його пам'яті.

Атака імітує ситуацію, коли зловмисник має права root або перебуває у середовищі, де політики безпеки (SELinux/AppArmor) не обмежують використання ptrace.

Код лаб3 було модифіковано таким чином, щоб після генерації ключа і перед виконанням підпису він зупинявся, утримуючи закритий ключ d у пам'яті.

```
import os
from Crypto.PublicKey import ECC
from Crypto.Signature import DSS
from Crypto.Hash import SHA256

print(f"Target PID: {os.getpid()}")
key = ECC.generate(curve='P-256')
d_value = key.d

print(f"Ключ згенеровано. {hex(d_value)} Чекаємо на команду...")
input("Натисніть Enter, щоб виконати підпис і вийти...")

message = b"Secret data to sign"
h = SHA256.new(message)
signer = DSS.new(key, 'fips-186-3')
signature = signer.sign(h)

print("Підпис виконано. Процес завершується.")
```

```
(.venv) user@ubuntu:~/kpi/mrkm25-26/lab4/Lab4_Sherstiuk_Vygranovsky$ python3 main.py
Target PID: 40613
Ключ згенеровано. 0xb0f2b4fb9288246a4e33a87ed5c0d7fe2db08bc10eee53469a8274bbf599c52f Чекаємо на команду...
Натисніть Enter, щоб виконати підпис і
```

Приватний ключ d: **0xb0f2b4fb9288246a4e33a87ed5c0d7fe2db08bc10eee53469a8274bbf599c52f**

Приєднання GDB до цільового процесу (з іншого терміналу):

$ sudo gdb -p 40613

Перевірка регістрів процесу (якщо ключ був використаний нещодавно):
(gdb) info registers

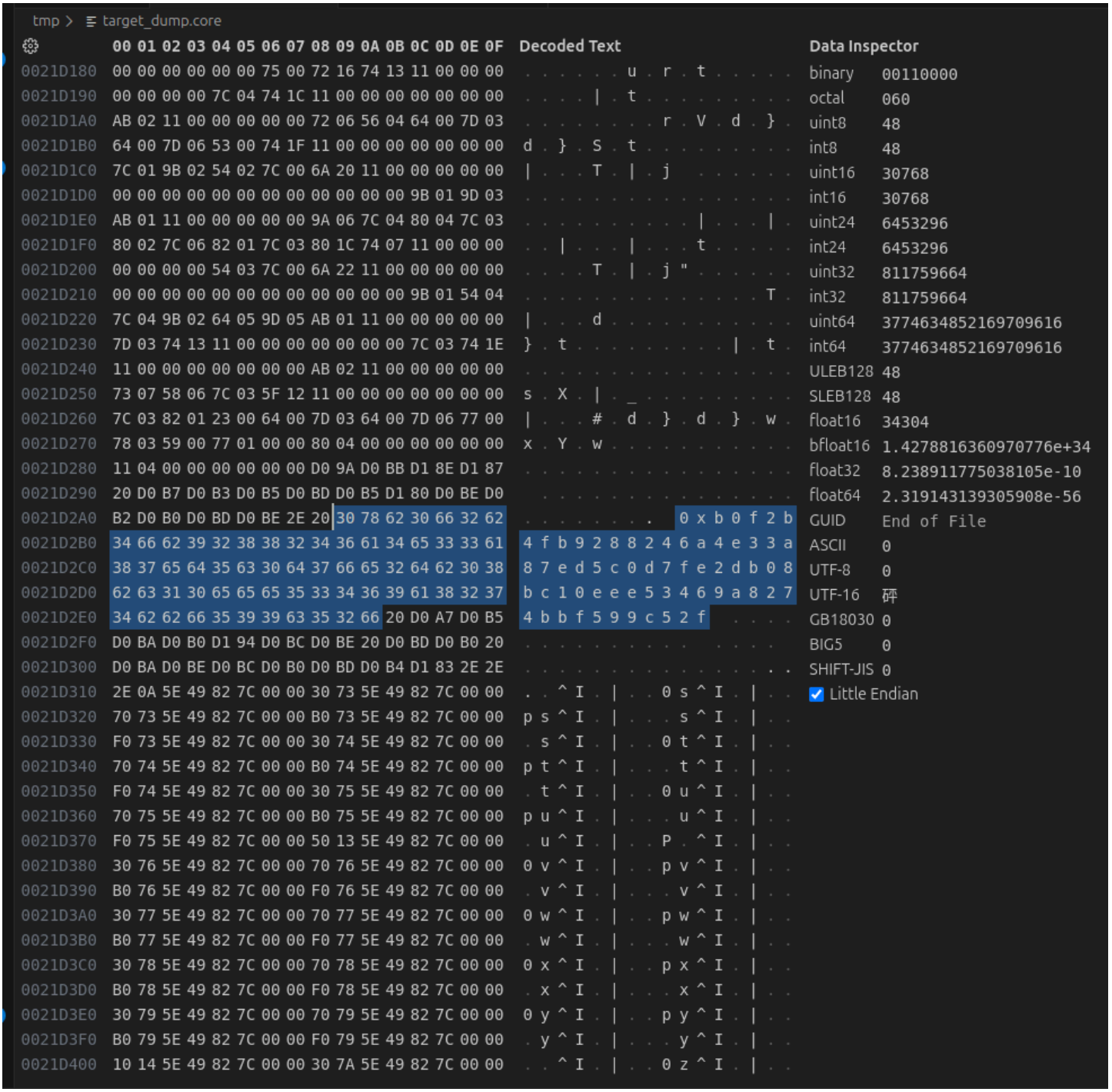
```
(gdb) info registers
rax            0xfffffffffffffe00  -512
rbx            0x7c82498038e0      136899020732640
rcx            0x7c824971ba91      136899019782801
rdx            0x400               1024
rsi            0x115f68c0          291465408
rdi            0x0                 0
rbp            0x7ffc5dae4f70      0x7ffc5dae4f70
rsp            0x7ffc5dae4f38      0x7ffc5dae4f38
r8             0x1                 1
r9             0x0                 0
r10            0x7c8249609cb8      136899018661048
r11            0x246               582
r12            0x7c8249802030      136899020726320
r13            0x7c8249801ee0      136899020725984
r14            0xba58a8            12212392
r15            0x7c82498038e0      136899020732640
rip            0x7c824971ba91      0x7c824971ba91 <__GI___libc_read+17>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
fs_base        0x7c82498b2080      136899021447296
gs_base        0x0                 0
```

Теоретично можна зробити:

- Пошук у купі: Якщо ми знаємо приблизний вміст ключа (наприклад, його перші байти), ми можемо шукати його. Однак, у PyCryptodome ключ d є об'єктом Python int. Нам потрібно знайти адресу об'єкта key.d.
- Інспекція адресного простору Python: У GDB ми можемо викликати функції Python, якщо є доступ до інтерпретатора.
  (gdb) call PyObject_Print([address_of_key_object], stdout, 0)

Зробимо дампування пам'яті для аналізу:
(gdb) generate-core-file /tmp/target_dump.core

Шукаємо 32-байтове представлення ключа d:



Аналіз дампу пам'яті підтвердив наявність значення закритого ключа d у пам'яті процесу, коли він перебував у стані очікування (після генерації, але до підпису). Цей результат підтверджує низьку стійкість криптопровайдера до атак, що використовують ptrace/дампування пам'яті, коли процес, що містить закритий ключ, знаходиться у стані очікування.

Математично стійкий алгоритм не може гарантувати безпеки, якщо ОС дозволяє стороннім процесам переглядати або модифікувати її пам’ять. Таким чином, практична безпека ECDSA визначається не лише силою еліптичної криптографії, а й якістю архітектури операційної системи та правильністю побудови середовища виконання.