



# Питання безпеки в смарт- контрактах

Лекція 7



- 1) Основні вразливості в смарт-контрактах
- 2) Вразливості в смарт-контрактах Algorand
- 3) Практичні рекомендації для забезпечення безпеки смарт-контрактів
- 4) Оптимізація роботи смарт-контрактів на Algorand



**Безпека** - найважливіший аспект розробки смарт-контрактів. Якщо буде зламано смарт-контракт, витрати можуть бути надзвичайно високими.

Тому при побудові смарт-контрактів необхідно забезпечити дотримання таких **критеріїв**:

- Мінімалізм
- Повторне використання коду
- Якість коду
- Можливість аудиту
- Тестове покриття

---

Ускладнення не сприяє безпеці. Помилки менш імовірні, якщо код **простий, короткий і виконує лише одну функцію**. Завжди намагайтеся зробити **код максимально простим**, щоб уникнути небажаних ефектів, які можуть призвести до лазівок у безпеці. **Простіше – значить безпечніше.**

+

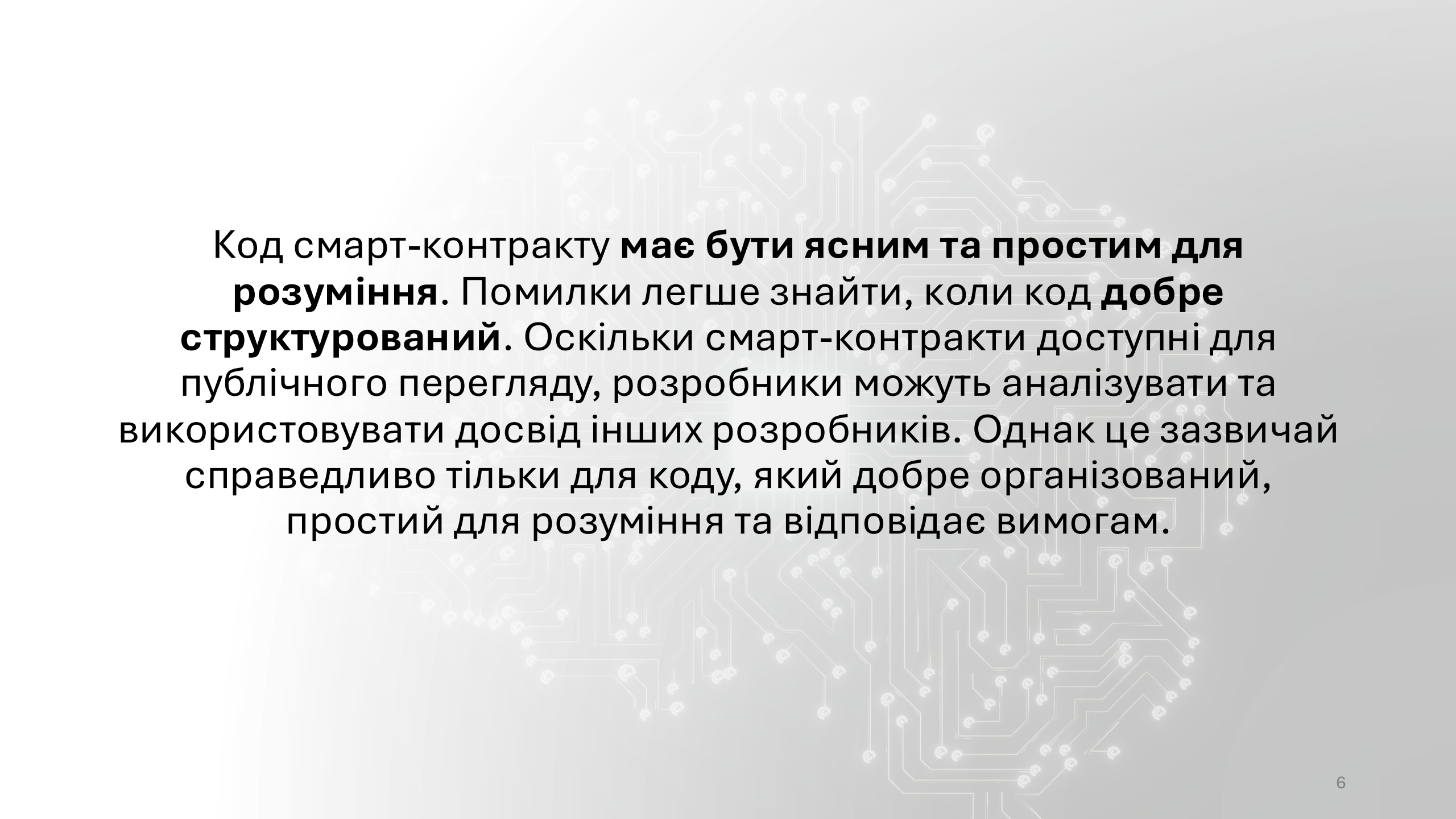
•

○

---

Розробникам рекомендується за можливості використовувати існуючі бібліотеки чи контракти. Це допомагає знизити ризик уразливості безпеки в нових контрактах. Ті смарт-контракти, які **вже** використовуються та **протестовані** іншими, набагато **безпечніші**, ніж нові.

Тому повторно використовуйте код, щоб зменшити ризик.



Код смарт-контракту **має бути ясним та простим для розуміння**. Помилки легше знайти, коли код **добре структурований**. Оскільки смарт-контракти доступні для публічного перегляду, розробники можуть аналізувати та використовувати досвід інших розробників. Однак це зазвичай справедливо тільки для коду, який добре організований, простий для розуміння та відповідає вимогам.

# Основні вразливості в смарт-контрактах

---

**Вразливості в смарт-контрактах** — це слабкі місця або помилки в коді смарт-контрактів, які можуть бути використані зловмисниками для здійснення атак або неправомірних дій. Вразливості можуть призводити до небажаних наслідків, таких як втрата коштів, викривлення бізнес-логіки, несанкціоноване виконання операцій, або навіть повне видалення контракту.





## Атаки повторного виклику (Reentrancy Attacks)

Атака повторного входу використовує вразливість у смарт-контрактах, коли функція здійснює зовнішній виклик іншого контракту перед оновленням власного стану. Це дозволяє зовнішньому контракту, можливо зловмисному, повторно увійти до початкової функції та повторити певні дії, як-от зняття коштів, використовуючи той самий стан. За допомогою таких атак зловмисник може витягти всі кошти з контракту.

**Приклад:** найвідомішим прикладом цього був злам DAO, під час якого було виведено ефір на суму 70 мільйонів доларів.

+

•

○

# Reentrancy в смарт- контрактах Algorand

В Algorand, AVM має вбудовані обмеження, які **забороняють повторний виклик (reentrancy)** функцій смарт-контракту. Це означає, що якщо код смарт-контракту намагається повторно викликати той самий смарт-контракт під час виконання іншої операції, система автоматично заблокує таку дію.

## Атаки відмови в обслуговуванні (DoS)

**DoS-атака (Denial of Service)** — це тип атаки, спрямований на те, щоб зробити певний сервіс, систему або смарт-контракт недоступним для користувачів. У контексті блокчейнів і смарт-контрактів DoS-атака зазвичай полягає в тому, щоб виснажити ресурси контракту, що призводить до неможливості виконання його функцій.

Смарт-контракти мають обмежені ресурси, такі як кількість обчислювальних операцій (газ), пам'ять або баланс, який можна використовувати для виконання транзакцій. Якщо зловмисник знайде спосіб витратити ці ресурси до їх повного вичерпання, смарт-контракт більше не зможе обробляти нові транзакції.

# DoS-атака. Особливості Algorand

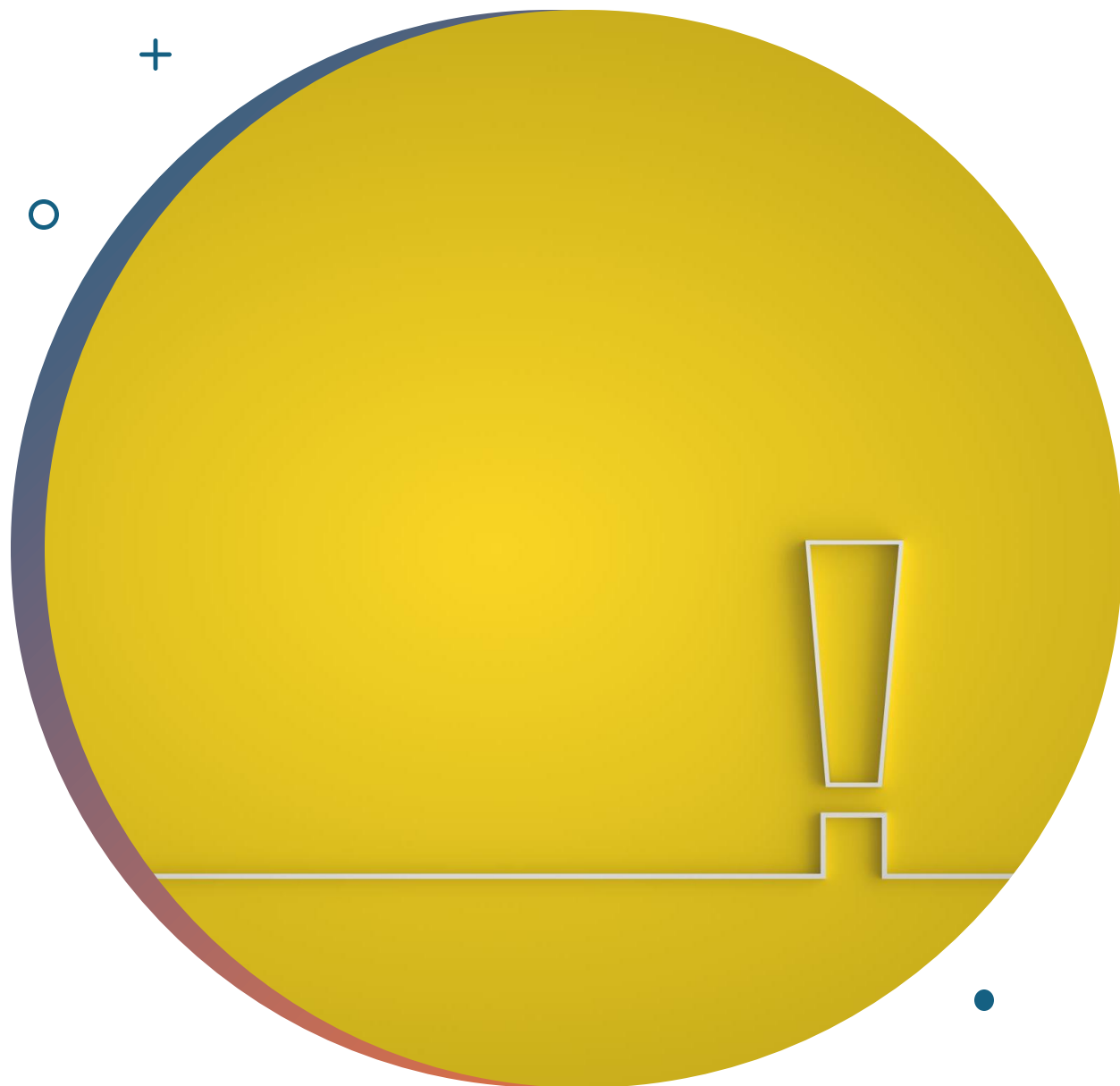
Ліміт газу: Algorand не використовує поняття "газу" як Ethereum, але має обмеження на кількість інструкцій (opcode) для кожного смарт-контракту. Це обмеження може допомогти уникнути деяких типів DoS-атак, оскільки зломисник не може виконати нескінченну кількість інструкцій.

Atomic Transaction Groups: Algorand дозволяє об'єднувати декілька транзакцій у групи, які виконуються атомарно (усі транзакції виконуються або жодна). Це може використовуватись для безпечної перевірки транзакцій, але якщо не перевіряється розмір групи або типи транзакцій, це може бути вразливим місцем для DoS-атаки.

Механізм оплати: в Algorand всі транзакції повинні бути оплачені, що зменшує ймовірність атак, оскільки зломиснику доведеться витратити реальні кошти. Проте, зломисник все ще може намагатися виснажити ресурси смарт-контракту шляхом надсилання великої кількості транзакцій, якщо в коді контракту є слабкі місця.

# Можливі сценарії DoS-атаки на Algorand

- **Атака через групові транзакції:** якщо контракт не перевіряє розмір групи транзакцій або типи транзакцій, зломисник може додати додаткові транзакції в групу, що може призвести до непередбачених наслідків, таких як витрати ресурсів або блокування доступу до контракту.
- **Виснаження балансу контракту:** якщо контракт виконує внутрішні транзакції, зломисник може спробувати виснажити баланс контракту через необмежені транзакції або через погане управління реквізитами транзакцій.



# Логічні ПОМИЛКИ

Логічні помилки, також відомі як вразливості бізнес-логіки, є прихованими недоліками в смарт-контрактах. Вони виникають, коли код контракту **не відповідає очікуваній поведінці або бізнес-логіці**, закладеній при його розробці. Ці помилки є підступними, оскільки можуть залишатися непоміченими протягом тривалого часу, поки хтось не виявить і не скористається ними.

# Логічні помилки

**Вплив:** Логічні помилки можуть призвести до несподіваної поведінки смарт-контракту або навіть зробити його повністю непридатним до використання. Такі помилки можуть спричинити втрату коштів, неправильний розподіл токенів або інші негативні наслідки, що можуть призвести до серйозних фінансових та операційних втрат для користувачів і зацікавлених сторін.

## **Виправлення:**

- Завжди перевіряйте свій код, створюючи комплексні тестові випадки, які охоплюють всі можливі варіанти бізнес-логіки.
- Проводьте ретельні перегляди коду та аудити, щоб виявити та виправити потенційні логічні помилки.
- Документуйте очікувану поведінку кожної функції та модуля, а потім порівнюйте її з фактичною реалізацією, щоб забезпечити їх відповідність.





## Відсутність або неналежна валідація вхідних даних

Якщо смарт-контракт не перевіряє належним чином дані, що **вводяться користувачами або надходять із зовнішніх джерел**, це може дозволити зловмисникам маніпулювати логікою контракту. Це може включати передання неправильних параметрів або значень, що можуть призвести до небажаної поведінки, зокрема до втрати коштів або компрометації безпеки контракту.



# Вразливості в смарт-контрактах Algorand



# Відсутня перевірка розміру групи транзакцій

Якщо смарт-контракт не перевіряє розмір групи транзакцій, зловмисники можуть додати свої транзакції до цієї групи. Це може призвести до втрати коштів, якщо ці додаткові транзакції включають переказ активів.

**Запобігання:** завжди перевіряйте розмір групи транзакцій та визначайте чіткі умови для кожної транзакції в групі.

```
def check_customer_payment():  
    return Seq(  
        Assert(Global.group_size() == Int(2)), # Check if the group size is 2  
        Assert(Gtxn[0].asset_amount() == Int(50)), # Check if the asset amount is 50  
        Assert(Gtxn[0].xfer_asset() == Int(1063265)), # Check if the asset ID is 1063265  
        Assert(Gtxn[0].asset_receiver() == Addr("2ERPXOLR7IESFS6FWX45CDA4Z3447SPNMC6YQHF2UXOXRRNLX5KZAMT4DI")),  
        Assert(Gtxn[0].rekey_to() == Global.zero_address()), # Check if RekeyTo is the zero address  
        Approve()  
    )
```



# Відсутній контроль доступу

Якщо в коді смарт-контракту відсутні перевірки для викликів додатків, таких як `UpdateApplication` та `DeleteApplication`, зломисник може змінити код додатку або повністю його видалити.

**Запобігання:** завжди додавайте перевірки на рівні смарт-контракту для викликів, які можуть змінювати або видаляти додаток. Наприклад, що видпляти або змінювати додаток може лише `creator_address`.

```
def application_deletion(self):  
    return Return(Txn.sender() == Global.creator_address())
```

## Відсутня перевірка ідентифікатора активу (Asset ID)

Якщо смарт-контракт не перевіряє ID активу, з яким взаємодіє, зломисник може маніпулювати логікою контракту, підставляючи фальшивий або інший (менш цінний) актив замість правильного.

**Запобігання:** завжди перевіряйте ідентифікатори активів, з якими працює смарт-контракт.

```
def check_customer_payment():
    return Seq(
        Assert(Global.group_size() == Int(2)), # Check if the group size is 2
        Assert(Gtxn[0].asset_amount() == Int(50)), # Check if the asset amount is 50
        Assert(Gtxn[0].xfer_asset() == Int(1063265)), # Check if the asset ID is 1063265
        Assert(Gtxn[0].asset_receiver() == Addr("2ERPXOLR7IESFS6FWX45CDA4Z3447SPNMC6YQHF2UXOXRRNLX5KZAMT4DI")),
        Assert(Gtxn[0].rekey_to() == Global.zero_address()), # Check if RekeyTo is the zero address
        Approve()
    )
```



# Відсутня перевірка комісії за внутрішню транзакцію

Якщо в смарт-контракті явно не встановлено суму комісії для внутрішньої транзакції, зловмисник може створити операції, що виконують внутрішні транзакції, та спалюють баланс додатку у вигляді комісій.

**Запобігання:** завжди перевіряйте та явно встановлюйте комісію для внутрішніх транзакцій. Найпростіше рішення для комісії за внутрішню транзакцію встановити її на рівні 0 і покластися на [об'єднання комісій](#).

# Відсутня перевірка параметра RekeyTo

Якщо в смарт-контракті немає перевірки параметра RekeyTo, зломисник може встановити цей параметр на свою адресу, що дозволить йому безпосередньо контролювати активи контракту або взяти під контроль обліковий запис підпису.

**Запобігання:** Завжди перевіряйте параметр RekeyTo у транзакціях.

```
def check_customer_payment():
    return Seq(
        Assert(Global.group_size() == Int(2)), # Check if the group size is 2
        Assert(Gtxn[0].asset_amount() == Int(50)), # Check if the asset amount is 50
        Assert(Gtxn[0].xfer_asset() == Int(1063265)), # Check if the asset ID is 1063265
        Assert(Gtxn[0].asset_receiver() == Addr("2ERPXOLR7IESFS6FWX45CDA4Z3447SPNMC6YQHF2UXOXRRNLX5KZAMT4DI")),
        Assert(Gtxn[0].rekey_to() == Global.zero_address()), # Check if RekeyTo is the zero address
        Approve()
    )
```

# Відсутня перевірка одержувача транзакції

У смарт-контрактах Algorand для підтвердження платіжних транзакцій можна використовувати атомарну групу транзакцій, яка зв'язує платіжні транзакції з викликами додатків. Якщо смарт-контракт не перевіряє одержувача платіжної транзакції або транзакції переказу активів, зломисник може вказати іншу адресу одержувача.

**Запобігання:** Завжди перевіряйте адресу одержувача у транзакціях.

```
def check_customer_payment():
    return Seq(
        Assert(Global.group_size() == Int(2)), # Check if the group size is 2
        Assert(Gtxn[0].asset_amount() == Int(50)), # Check if the asset amount is 50
        Assert(Gtxn[0].xfer_asset() == Int(1063265)), # Check if the asset ID is 1063265
        Assert(Gtxn[0].asset_receiver() == Addr("2ERPXOLR7IESFS6FWX45CDA4Z3447SPNMC6YQHF2UXOXRRNLX5KZAMT4DI")),
        Assert(Gtxn[0].rekey_to() == Global.zero_address()), # Check if RekeyTo is the zero address
        Approve()
    )
```



# Переповнення та недоповнення (Overflow/Underflow)

В Algorand Virtual Machine (AVM) за замовчуванням система зупиняє виконання контракту у випадку переповнення, недоповнення або ділення на нуль, що призводить до помилки транзакції. Для запобігання цьому можна додати обмеження на значення змінних, що беруть участь в операціях.

**Запобігання:** завжди додавайте перевірки на значення змінних, які можуть призвести до переповнення або недоповнення.



# • + Практичні рекомендації для + • забезпечення безпеки смарт- контрактів Algorand

# Правильна обробка OnComplete

Важливо не дозволяти викликати створений вами метод, якщо значення "OnComplete" не "NoOp".

**Рекомендація:** На початку кожного входу в смарт-контракт варто гілкувати код на основі значення OnComplete, і тільки якщо це **NoOp**, переходити до виконання конкретного методу.

```
program = Cond(  
    [Txn.application_id() == Int(0), handle_creation],  
    [Txn.on_completion() == OnComplete.OptIn, handle_optin],  
    [Txn.on_completion() == OnComplete.CloseOut, handle_closeout],  
    [Txn.on_completion() == OnComplete.UpdateApplication, handle_update],  
    [Txn.on_completion() == OnComplete.DeleteApplication, handle_delete],  
    [Txn.on_completion() == OnComplete.NoOp, handle_noop],  
)
```

І вже в `handle_noop` можна робити виклик конкретного методу смарт-контракту.

# Мінімізація кількості точок входу

**Точки входу (або методи) у смарт-контракті** — це місця, де контракт може бути викликаний для виконання певної дії. Чим більше таких точок входу, тим складніше тестувати та підтримувати контракт, і тим більше можливостей для помилок або зловживань.

## Рекомендація:

Зменшення кількості точок входу до найнеобхідніших спрощує тестування та підвищує безпеку контракту. Це дозволяє більш чітко визначити, які дії можливі у кожен конкретний момент часу, і легше перевірити правильність їх виконання.



# Оптимізація роботи смарт-контрактів на Algorand

# Обмеження на розмір контракту та вартість виконання

В Algorand існує бюджет на виконання операцій, що обмежує кількість операцій, які можуть бути виконані в одному контракті.

**Рекомендація:** Звертайте увагу на вартість операційних кодів (opcode) і уникайте використання дорогих операцій, таких як байтові операції, якщо це можливо.



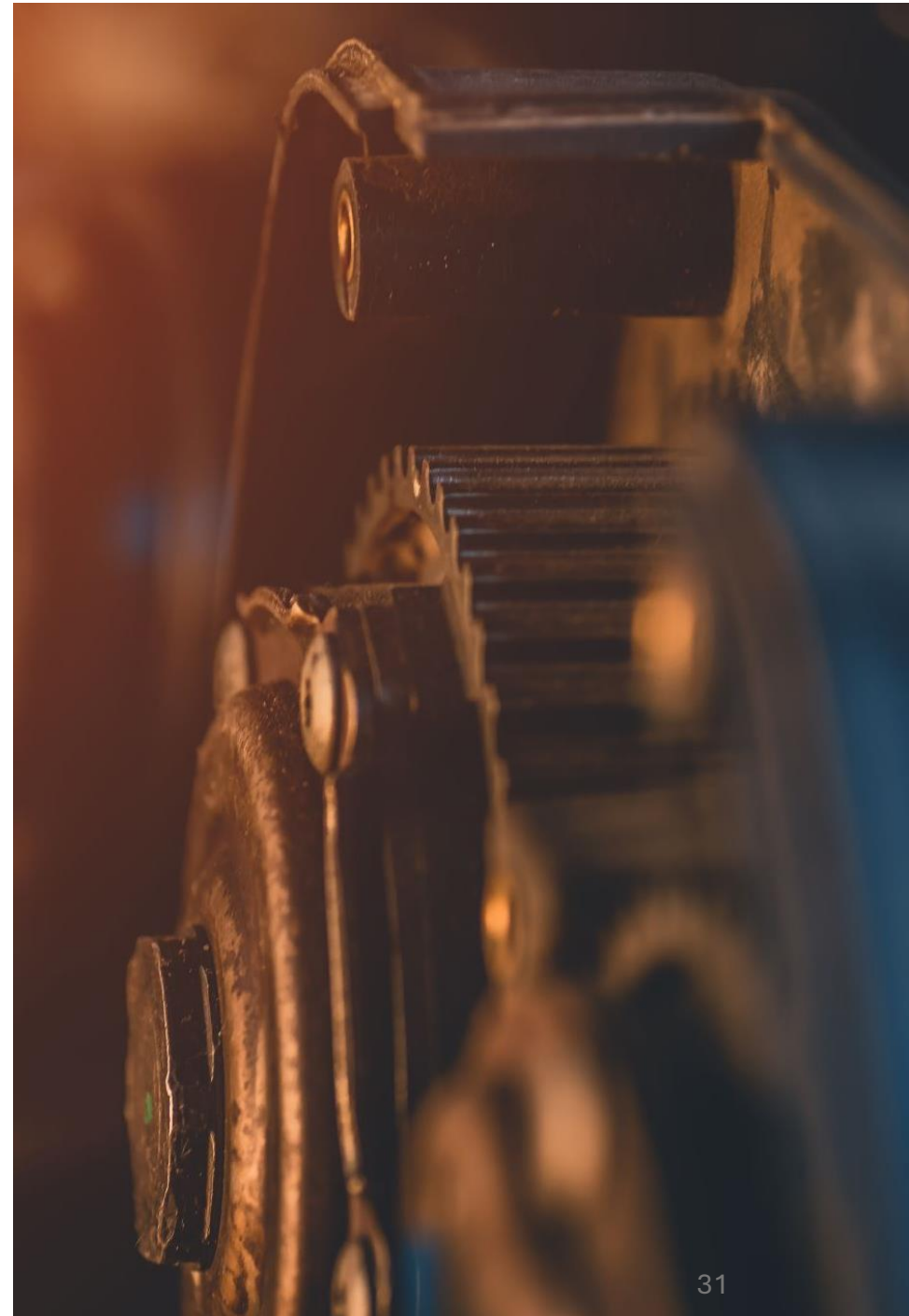
## Приклад

Проста операція, наприклад, додавання двох чисел, матиме вартість 1. Звичайно, завантаження цих чисел матиме свою власну вартість, але просте додавання коштує 1 одиницю. Коли ви складаєте два байти, вартість становить 10. Але коли ви хочете обчислити хеш чогось, вартість злітає до сотень.

# Перенесення складних обчислень off-chain

Обмеження на кількість операцій (700 операцій за одну транзакцію) в Algorand диктує необхідність виконання складних обчислень поза межами смарт-контракту (off-chain). Це дозволяє зменшити вартість транзакцій та підвищити ефективність контракту.

**Рекомендація:** Переносьте складні обчислення off-chain, якщо це можливо, для оптимізації використання ресурсів смарт-контракту.







## Приклад


Уявіть, що потрібно знайти найкоротший шлях у графі. Це завдання може бути надто складним для виконання в межах смарт-контракту через обмеження на кількість операцій. Замість цього, шлях можна знайти на бекенді, а результат передати контракту для подальшої перевірки та виконання.




# Уникайте використання байтової математики

Байтові операції в Algorand можуть бути зручними в певних випадках. Наприклад, вони дозволяють працювати з більшими числами та ширшими цілими числами, ніж стандартний `uint64`. Однак, ці операції є **набагато дорожчими** в плані вартості обчислень у смарт-контрактах, що може сильно вплинути на ефективність виконання контракту.

**Рекомендація:** варто уникати використання байтових операцій, коли задачу можна вирішити за допомогою **стандартних `uint64`**. Це дозволить знизити вартість виконання контракту та зробить його більш ефективним.



# Мінімізуйте розмір контракту



Уникайте зберігання великих обсягів даних у смарт-контракті та використовуйте максимально компактний код, щоб скоротити розмір контракту і зменшити витрати на його виконання.

# • + ◦ Стандартні правила безпеки ◦ + •

НІКОЛИ не зберігайте ключ або мнемоніку у вихідному коді. Практично неможливо безпечно видалити ключ або мнемоніку з git репозиторію.

Уникайте використання невідомих або неперевірених бібліотек, оскільки вони можуть містити вразливості.

Використовуйте тестову мережу Algorand для тестування вашого смарт-контракту перед розгортанням у виробничій мережі, щоб виявити потенційні вразливості.

Проводьте регулярні аудити.

Відстежуйте зміни в екосистемі Algorand і оновлюйте свій смарт-контракт відповідно до нових стандартів безпеки.

## Уникайте зберігання конфіденційних даних у контракті

Смарт-контракти є прозорими і доступними для перегляду всіх учасників мережі. Зберігання конфіденційних даних у контракті може призвести до витоку інформації.

**Правило:** Не зберігайте приватні ключі, паролі або інші конфіденційні дані безпосередньо у смарт-контракті. Якщо це необхідно, використовуйте шифрування.

+

•

○

# Документуйте та тестуйте Код

Без належної документації та тестування можуть виникнути помилки та вразливості, які важко виявити на ранніх стадіях.

**Правило:** Завжди документуйте функції та модулі вашого коду. Пишіть тести, що покривають усі можливі сценарії використання контракту.



## Додаткові матеріали

- [Modern guidelines for smart contracts and smart signatures on Algorand](#): офіційна документація Algorand
- [\(Not So\) Smart Contracts](#): цей репозиторій містить приклади поширених уразливостей смарт-контрактів Algorand, включно з кодом реальних смарт-контрактів.

