

The background is a dark blue space filled with numerous glowing blue cubes of varying sizes and orientations. A complex network of thin, light blue lines crisscrosses the background, resembling a web or a data network. The overall aesthetic is futuristic and technological.

Développer et déployer des contrats intelligents sur Ethereum

Optimisation du gaz
Vulnérabilités - Slither



Optimisation du gaz en Solidity

Mettre les variables de stockage en cache

- La lecture à partir d'une variable de stockage coûte au moins 100 gas
- Les écritures sont beaucoup plus cher
- Mettre en cache les variables de stockage pour effectuer une seule lecture et écriture

```
4  contract Caching {
5      uint256 public number;
6
7      function noCache(uint numberOfLoops) public view returns(uint result) {
8          for(uint i = 0; i < numberOfLoops; ++i) {
9              result += number;
10         }
11     }
12
13     function cache(uint numberOfLoops) public view returns(uint result) {
14         uint cachedVar = number;
15         for(uint i = 0; i < numberOfLoops; ++i) {
16             result += cachedVar;
17         }
18     }
19 }
```



Regrouper les structures

- Regrouper les éléments d'une structure pour réduire les coûts de gas
- Permet a minimiser les opérations coûteuses liées au stockage
- Les éléments de la premier structure sont stockés dans trois emplacements séparés
- Les éléments de la deuxième structure sont stockés dans seulement deux emplacements séparés => lecture et l'écriture moins cher

```
1  contract Packed_Struct {
2      struct unpackedStruct {
3          uint64 time;
4          uint256 money;
5          address person;
6      }
7
8      struct packedStruct {
9          uint64 time;
10         address person;
11         uint256 money;
12     }
13 }
14
```



Utiliser des variables immuables et constantes

- Les variables qui ne sont jamais mises à jour devraient être immuables ou constantes
- Les constantes et les valeurs immuables sont intégrées directement dans le bytecode du contrat et n'utilisent pas de stockage

```
4  contract ConstantAndImmutable {  
5      uint256 constant public CONSTANT_VALUE = 123;  
6      uint256 immutable public IMMUTABLE_VALUE;  
7  
8      constructor(uint256 _initialValue) {  
9          IMMUTABLE_VALUE = _initialValue;  
10     }  
11  
12     // rest of the contract code...  
13 }  
14
```



Timestamp & numéros de blocs n'ont pas besoin d'être uint256

- Un horodatage de taille **uint48** fonctionnera pour des millions d'années à venir
- Un numéro de bloc s'incrémente toutes les 12 secondes => **uint32** sera suffisant

```
4  contract TimestampAndBlockNumber {
5      uint48 public timestamp;
6      uint32 public blockNumber;
7
8      constructor() {
9          timestamp = uint48(block.timestamp);
10         blockNumber = uint32(block.number);
11     }
12
13     function updateData() public {
14         timestamp = uint48(block.timestamp);
15         blockNumber = uint32(block.number);
16     }
17 }
18
```



Calldata est moins cher que memory

- Uniquement pour les arguments de type référence des fonctions externes
- L'accès aux données depuis calldata nécessite moins d'opérations
- Utiliser la mémoire uniquement lorsque les données doivent être modifiées

```
1  contract CalldataContract {  
2      function getDataFromCalldata(bytes calldata data) public pure returns (bytes memory) {  
3          return data;  
4      }  
5  }  
6  
7  contract MemoryContract {  
8      function getDataFromMemory(bytes memory data) public pure returns (bytes memory) {  
9          return data;  
10     }  
11 }  
12
```



Utiliser ++i au lieu de i++ pour incrémenter

- **i++** renvoie sa valeur précédente (i) avant d'incrémenter => deux valeurs sont stockées dans le stack
- **++i** incrémente i puis renvoie i => une seule valeur est stockée dans le stack

```
4  contract IncrementExample {  
5      uint256 public counter;  
6  
7      function incrementWithPrefix() public {  
8          counter = 0;  
9  
10         for (uint256 i; i < 10;) {  
11             counter += 1;  
12             unchecked {  
13                 ++i;  
14             }  
15         }  
16     }  
17 }
```



Les boucles do-while sont moins cher que les boucles for

```
1  v contract LoopFor {
2  v      function loop(uint256 times) public pure {
3  v          for (uint256 i; i < times;) {
4              // execute desired code ...
5  v          unchecked {
6              ++i;
7          }
8      }
9  }
10 }
11
```

```
12 v contract LoopDoWhile {
13 v     function loop(uint256 times) public pure {
14 v         if (times == 0) {
15             return;
16         }
17
18         uint256 i;
19
20 v         do {
21             // execute desired code ...
22 v             unchecked {
23                 ++i;
24             }
25         } while (i < times);
26     }
27 }
```



N'utiliser pas public pour les variables d'état si ce n'est pas nécessaire

- Une fonction publique (getter) est créée pour les variable de stockage publique
- Augmente la taille de la "jump table"
- Augmente la taille du bytecode
- Le contrat devient plus volumineux

```
3  contract StateVariables {
4
5      uint256 private privateVar = 100;
6
7      uint256 internal internalVar = 200;
8
9      uint256 public publicVar = 300;
10
11      // rest of the contract code...
12  }
13
```



Quelques ressources supplémentaires

- The RareSkills Book of Solidity Gas Optimization :
<https://www.rareskills.io/post/gas-optimization>
- How to Optimize Smart Contracts in Solidity :
<https://medium.com/@0xkaden/how-to-write-smart-contracts-that-optimize-gas-spent-on-ethereum-30b5e9c5db85>
- Solidity Gas Optimizations Cheat Sheet :
<https://0xmacro.com/blog/solidity-gas-optimizations-cheat-sheet>





Vulnérabilités des Contrats Intelligents

Vulnérabilités des Contrats Intelligents

- Les vulnérabilités de sécurité des contrats intelligents posent des risques significatifs => pertes financières, rendre un protocole inutilisable...
- Un audit et des tests approfondis sont essentiels avant le déploiement des contrats intelligents



Contrôle d'accès manquant

- Placer des restrictions sur qui peut appeler des fonction sensible, comme le retrait d'Ether, le changement du propriétaire du contrat...
- Même si un modificateur est en place, il y a eu des cas où le modificateur n'a pas été utilisé

```
1  contract MissingAccesControl {
2
3      address public owner;
4
5      modifier onlyOwner {
6          owner == msg.sender;
7          _;
8      }
9
10     function changeOwner(address newOwner) public {
11         owner = newOwner;
12     }
13
14     function changeOwnerWithModifier(address newOwner) public onlyOwner {
15         owner = newOwner;
16     }
17 }
```



Validation d'entrée incorrecte ou manquant

```
3  contract Auction {
4      address public highestBidder;
5      uint public highestBid;
6
7      function placeBid() public payable {
8          require(msg.value < highestBid);
9
10         highestBidder = msg.sender;
11         highestBid = msg.sender;
12     }
13 }
```



Gas Griefing & déni de service (DoS)

Un contrat peut consommer malicieusement tout le gaz en entrant dans une boucle infinie.

```
1  contract DistributeETH {
2      address[] users;
3
4      function distribute(uint256 total) public {
5          for (uint i; i < users.length; ++i) {
6              users[i].call{value: total / users.length}("");
7          }
8      }
9  }
10
11 contract Attacker {
12     fallback() external payable {
13         // infinite loop uses up all the gas
14         while (true) {
15             }
16     }
17 }
18
```



Insecure Randomness

- Il n'est actuellement pas possible de générer de l'aléatoire de manière sécurisée sur la blockchain
- Les blockchains doivent être entièrement déterministes, sinon les nœuds ne pourraient pas parvenir à un consensus sur l'état
- Peu importe comment vous générez l'aléatoire, un attaquant peut le reproduire exactement

```
6 contract UnsafeDice {
7     function randomNumber() internal view returns (uint256) {
8         return uint256(keccak256(abi.encode(msg.sender, tx.origin, block.timestamp,
9             tx.gasprice, blockhash(block.number - 1))));
10    }
11
12    function rollDice() public payable {
13        require(msg.value == 1 ether);
14
15        if (((randomNumber() % 6) + 1) == 6) {
16            (bool success,) = msg.sender.call{value: 2 ether}("");
17            require(success, "Transacion failed");
18        }
19    }
20 }
21
```



Insecure Randomness

```
23 interface IUnsafeDice {
24     function rollDice() external payable;
25 }
26
27 contract ExploitDice {
28
29     IUnsafeDice unsafeDice;
30
31     constructor(address _unsafeDice) {
32         unsafeDice = IUnsafeDice(_unsafeDice);
33     }
34
35     function randomNumber() internal view returns (uint256) {
36         return uint256(keccak256(abi.encode(msg.sender, tx.origin, block.timestamp,
37         tx.gasprice, blockhash(block.number - 1))));
38     }
39
40     function attack() public payable {
41         if (((randomNumber() % 6) + 1) == 6) {
42             unsafeDice.rollDice{value: 1 ether}();
43         }
44     }
45 }
```



Variables privées

- Les variables privées sont toujours visibles sur la blockchain
- Il faut jamais stocker les informations sensibles
- Pour lire une variable, il faut connaître son emplacement de stockage
- Dans l'exemple, l'emplacement de stockage de "secretNumber" est 2

```
6  contract PrivateExample {
7      uint256 public someNumber;
8      address internal someAddress;
9      uint256 private secretNumber;
10
11     constructor(uint256 _initialValue) {
12         secretNumber = _initialValue;
13     }
14 }
15
16 //ethers.js : await provider.getStorageAt(contractAddress, slotNumber);
17
```



Erreurs d'arrondi - multiplier avant de diviser

```
6  contract RoundingErrors {
7
8      //factor = 1001
9
10     function divideFirst(uint256 factor) external pure returns (uint256) {
11         return (1000 / factor) * 100;
12     }
13
14     function test2(uint256 factor) external pure returns (uint256) {
15         return (1000 * 100) / factor;
16     }
17 }
```

Solidity n'a pas de nombres à virgule flottante, donc les erreurs d'arrondi sont inévitables. La division doit toujours être effectuée en dernier.



Quelques ressources supplémentaires

- Solidity Smart Contract Attack Vectors :
<https://github.com/Quillhash/Solidity-Attack-Vectors>
- Smart Contract Vulnerabilities :
<https://github.com/kadenzipfel/smart-contract-vulnerabilities>
- Smart Contract Security :
<https://www.rareskills.io/post/smart-contract-security>
- Solidity By Example - Hacks : <https://solidity-by-example.org/>
- Security Vulnerability Aggregator : <https://solodit.xyz/>





Slither

Analyse Statique

Slither – Trail of Bits

- Outil d'analyse statique open-source
- Spécialisé dans la sécurité des contrats intelligents Ethereum
- Recherche les vulnérabilités potentielles et les mauvaises pratiques de programmation dans le code
- Fournit des recommandations pour améliorer la sécurité et la qualité du code
- Utilisé par de nombreux développeurs et auditeurs
- Page officielle Github : <https://github.com/crytic/slither>
- **Prérequis :**
 - - Installer Python 3.8+
 - - Installer Solc-Select – requis si Hardhat, Foundry... n'est pas utilisées



Python 3.8+

- Télécharger Python : <https://www.python.org/downloads/>
- Installer Python : <https://www.datacamp.com/blog/how-to-install-python>



Solc-Select

- Outil pour passer rapidement entre les versions du compilateur Solidity
- Page officielle Github : <https://github.com/crytic/solc-select>
- Installer Solc-Select : ***pip3 install solc-select***
- Utilisation de Solc-Select :
 - Vérifier la version actuelle de solc : ***solc --version***
 - Installer une version spécifique de sol : ***solc-select install 0.8.20***
 - Utiliser une version spécifique : ***solc-select use 0.8.20***



Utilisation de Slither

- Installation de Slither : ***pip3 install slither-analyzer***
- Ajouter slither.config.json au dossier racine du projet

```
1  {  
2      "solc_remappings": "@openzeppelin=../node_modules/@openzeppelin,@chainlink=../node_modules/@chainlink",  
3      "exclude_informational": false,  
4      "exclude_low": false,  
5      "exclude_medium": false,  
6      "exclude_high": false,  
7      "disable_color": false,  
8      "filter_paths": "(node_modules/|scripts/|artifacts/)",  
9      "printers_to_run": "contract-summary,inheritance-graph,function-summary"  
10 }
```

- Exécution de Slither : ***slither .*** ou : ***slither ./contracts/myContract.sol***



Slither Filters

- Les résultats peuvent être filtrés :
 - Optimisation : `--exclude-optimization`
 - Informations : `--exclude-informational`
 - Résultats faibles : `--exclude-low`
- Utilisation du filtre : ***slither . --exclude-informational***



Slither Printers

➤ Par défaut, aucune imprimante n'est exécutée.

➤ Pour exécuter des imprimantes :

slither ./contracts/myContract.sol --print contract-summary,function-summary

➤ Autres imprimantes utiles :

- inheritance-graph
- call-graph

➤ Liste de toutes les imprimantes : <https://github.com/crytic/slither/wiki/Printer-documentation>



Slither Detectors

- Slither exécute tous ses détecteurs par défaut.
- Pour exécuter uniquement des détecteurs sélectionnés :
slither . --detect arbitrary-send,pragma
- Pour exclure des détecteurs :
slither . --exclude naming-convention,unused-state
- Liste de toutes les détecteurs : <https://github.com/crytic/slither/wiki/Detector-Documentation>

