Aula 12 – Vulnerabilidades Clássicas de Smart Contracts (Parte 2)

Bem-vindo(a) à segunda parte da nossa jornada pelas vulnerabilidades clássicas em smart contracts! No universo blockchain, onde o código é lei e as transações são imutáveis, entender as falhas de segurança não é apenas uma boa prática, é uma necessidade crítica. Imagine construir um cofre digital que guarda milhões, mas sem conhecer as técnicas mais comuns que ladrões usam para abri-lo. É exatamente isso que estamos evitando aqui.

Nesta aula, vamos aprofundar nosso conhecimento sobre como contratos inteligentes podem ser explorados, focando em cenários que, embora conhecidos, continuam a causar perdas significativas no ecossistema. Nosso objetivo é que, ao final, você seja capaz de identificar, compreender e, mais importante, mitigar riscos associados à manipulação de oráculos, ataques de negação de serviço por limites de gás e o uso indevido de tx.origin para autenticação.

Este conhecimento não só solidifica sua base em segurança de smart contracts, mas também o prepara para desenvolver soluções mais robustas e resilientes, um diferencial valioso para qualquer profissional da área. Conectaremos esses conceitos com as tendências atuais, como a Abstração de Contas e as soluções de escalabilidade, mostrando como o cenário de segurança está em constante evolução. Prepare-se para desvendar os segredos por trás de algumas das explorações mais engenhosas do mundo blockchain.

Fundamentos

Revisitando as Fundações: Uma Breve Recapitulação

Antes de mergulharmos em novas vulnerabilidades, é fundamental que tenhamos uma base sólida, relembrando os conceitos que abordamos na Parte 1. Pense nisso como um aquecimento antes de um exercício mais intenso: precisamos ter certeza de que nossos músculos estão prontos e que a memória está fresca. As vulnerabilidades que estudamos anteriormente, como reentrancy, integer overflow/underflow, dependência de timestamp e front-running, são a espinha dorsal de muitos ataques e servem como um lembrete constante da complexidade da segurança em contratos inteligentes.

Reentrancy

Ordem das operações e técnica "Checks-Effects-Interactions"

Overflow/Underflow

Gerenciamento cuidadoso dos limites numéricos

Timestamp

Perigos de dados manipuláveis por mineradores

Front-Running

Exploração da ordem das transações

Cada uma dessas falhas representa uma brecha potencial que, se não for cuidadosamente tratada, pode levar a perdas financeiras ou à interrupção de serviços. Com essa revisão em mente, estamos mais preparados para entender como novas camadas de complexidade podem introduzir outros tipos de riscos. A segurança em blockchain é um campo dinâmico, onde o aprendizado contínuo e a adaptação são essenciais. Agora, vamos explorar vulnerabilidades que se manifestam de maneiras diferentes, mas com o mesmo potencial destrutivo.

Vulnerabilidade #1

A Ponte Perigosa: Manipulação de Oráculos de Preço

> 💡 **Analogia:** Imagine que você está em um jogo de cartas online onde o valor das suas apostas é determinado pelo preço do ouro no mercado real. Para que o jogo funcione, ele precisa de uma fonte confiável que diga qual é o preço atual do ouro. Essa fonte é o que chamamos de **oráculo**.

Em blockchain, os oráculos são a ponte vital que conecta o mundo real (off-chain) com o mundo dos contratos inteligentes (on-chain), fornecendo dados externos como preços de ativos, resultados de eventos esportivos ou condições climáticas.

O Problema

O problema surge quando essa ponte não é segura. Se a fonte de dados do oráculo for única, centralizada ou facilmente manipulável, um atacante pode enganar o contrato inteligente, fazendo-o acreditar em informações falsas. Por exemplo, um contrato de empréstimo que usa o preço de um token como garantia pode ser explorado se o oráculo reportar um preço inflacionado artificialmente. O atacante poderia então pegar um empréstimo maior do que o valor real da sua garantia, drenando os fundos do protocolo.

Essa vulnerabilidade é particularmente insidiosa porque não reside no código do smart contract em si, mas na sua dependência de dados externos. É como ter um sistema de segurança de última geração em sua casa, mas a câmera de vigilância externa está quebrada ou apontando para o lugar errado. A confiança nos dados é tão crítica quanto a segurança do código que os processa.

Cenários de Ataque e Defesas Contra Oráculos Manipulados

Cenário de Ataque Comum

A manipulação de oráculos não é apenas teórica; ela já foi a causa de explorações multimilionárias. Um cenário comum envolve o uso de **flash loans** (empréstimos instantâneos sem garantia) para manipular o preço de um ativo em uma exchange descentralizada (DEX) de baixa liquidez.

01

Flash Loan

Atacante pega um grande empréstimo instantâneo

02

Manipulação

Usa fundos para inflar o preço na DEX

03

Exploração

Contrato vulnerável lê preço artificialmente alto

04

Lucro

Executa operação lucrativa (empréstimo/liquidação)

05

Devolução

Devolve o flash loan, mantendo o lucro

Soluções de Mitigação

Para mitigar esses riscos, a indústria tem desenvolvido soluções robustas:

Conceito	Âmbito/Aplicação	Base/Origem
Oráculo Centralizado	Uma única fonte de dados para o smart contract	Entidade única, ponto de falha único
Oráculo Descentralizado	Múltiplas fontes de dados, consenso on-chain	Rede de nós, agregação de dados (Chainlink, Band Protocol)
TWAP	Preço médio ponderado pelo tempo	Cálculo contínuo em um período (ex: 1 hora)

Além disso, os desenvolvedores devem ser céticos em relação a oráculos que dependem de pools de liquidez com baixo volume, pois são mais suscetíveis a manipulações. A escolha de um oráculo é uma decisão de design crítica que impacta diretamente a segurança e a resiliência de um dApp.

Vulnerabilidade #2

O Gargalo Inesperado: Ataques de Negação de Serviço (DoS) por Gas Limit

Em sistemas tradicionais, um ataque de negação de serviço (DoS) visa sobrecarregar um servidor para torná-lo inacessível. No blockchain, a natureza distribuída e o mecanismo de gás introduzem uma variação peculiar desse ataque. Imagine que você está em uma fila para pagar uma conta, e cada operação que você faz tem um custo. Há um limite de quanto "trabalho" (gás) pode ser feito em um único bloco. Se uma função de um contrato inteligente se tornar excessivamente cara para executar, ela pode se tornar inoperável, efetivamente sofrendo um DoS.

Como Funciona

Essa vulnerabilidade ocorre quando um contrato inteligente possui uma função que itera sobre uma lista ou mapa, e o tamanho dessa lista pode ser manipulado por um atacante. Se a lista crescer demais, o custo de gás para executar a função (por exemplo, distribuir recompensas para todos os participantes) pode exceder o limite de gás de um bloco.

Consequência

Quando isso acontece, a transação falha, e a função se torna inutilizável, impedindo operações legítimas. O atacante não precisa destruir o contrato, apenas torná-lo inútil para os usuários legítimos.

> ⚠️ **Analogia:** É como ter uma máquina de venda automática que funciona perfeitamente para um pequeno número de itens. Mas se alguém encher a máquina com milhares de itens, ela pode travar ou se recusar a dispensar qualquer coisa, pois o mecanismo interno não consegue lidar com a sobrecarga.

Tipos de DoS por Gas Limit e Estratégias de Mitigação

Exemplos de Ataques

Lista de Distribuição

Contrato mantém lista de usuários para distribuir fundos. Atacante adiciona grande número de endereços, tornando a função de distribuição impossível de executar.

Loops Inflacionáveis

Loops que dependem de variáveis que podem ser inflacionadas por atores maliciosos, excedendo o limite de gás do bloco.

Estratégias de Mitigação

1. **Padrão "Pull over Push"**

   Em vez de o contrato "empurrar" fundos para todos os usuários em uma única transação (que pode falhar por gás), os usuários são obrigados a "puxar" seus fundos individualmente. Isso distribui o custo de gás e evita o problema do limite de bloco.

2. **Limites de Iteração**

   Impor limites de iteração em loops e garantir que as estruturas de dados não possam ser inflacionadas por atores maliciosos.

3. **Design Cuidadoso**

   Evitar operações que crescem linearmente com o número de usuários ou dados. Considerar soluções de escalabilidade (Layer 2s) quando apropriado.

A Abstração de Contas (ERC-4337) e as soluções de escalabilidade (Layer 2s) podem, em alguns contextos, aliviar a pressão do gás, mas o problema fundamental de design de contratos que executam operações caras em um único bloco permanece e deve ser endereçado no nível do smart contract.

Vulnerabilidade #3

A Armadilha da Identidade: Uso Indevido de tx.origin para Autenticação

No mundo dos smart contracts, a autenticação é a base da segurança. Saber quem está chamando uma função e se essa entidade tem permissão para fazê-lo é fundamental. Em Solidity, temos duas variáveis globais que parecem semelhantes, mas têm diferenças cruciais: `msg.sender` e `tx.origin`. A confusão entre elas pode levar a uma vulnerabilidade grave, onde um contrato malicioso pode enganar um contrato legítimo.

msg.sender

É o endereço da entidade que chamou a função **imediatamente**. Pode ser um usuário final (uma carteira externa) ou outro smart contract.

tx.origin

É sempre o endereço da carteira externa (EOA - Externally Owned Account) que iniciou a **transação completa**.

🔍 **Exemplo de Cadeia:** Se A chama B, e B chama C, então `msg.sender` em C seria B, mas `tx.origin` em C seria A.

O Problema de Segurança

O problema surge quando um contrato usa `tx.origin` para autenticar um usuário. Um atacante pode criar um contrato malicioso que, quando chamado por um usuário legítimo (que é o `tx.origin`), chama um contrato vulnerável. O contrato vulnerável, ao verificar `tx.origin`, verá o endereço do usuário legítimo e concederá acesso, mesmo que a chamada tenha vindo do contrato malicioso. É como um impostor usando a identidade de outra pessoa para entrar em um lugar, mesmo que a pessoa real não tenha autorizado diretamente.

Detalhes do Ataque tx.origin e Boas Práticas de Segurança

Ilustração do Ataque

Para ilustrar o ataque `tx.origin`, considere um contrato de carteira simples que permite ao proprietário sacar fundos. Se este contrato usar `require(tx.origin == owner)` para verificar se o chamador é o proprietário, ele é vulnerável.

Contrato Malicioso

Atacante cria contrato com função que chama o contrato vulnerável

Phishing

Proprietário legítimo é enganado a chamar o contrato malicioso

Exploração

Contrato malicioso chama função de saque do contrato vulnerável

Bypass

tx.origin ainda é o proprietário legítimo, verificação passa

Roubo

Fundos são enviados para o contrato malicioso

Solução Definitiva

❌ NUNCA USE

```
require(tx.origin == owner)
```

Vulnerável a ataques de phishing via contratos maliciosos

✅ SEMPRE USE

```
require(msg.sender == owner)
```

Verifica o chamador imediato, seguro contra intermediários maliciosos

Conceito	Âmbito/Aplicação	Base/Origem
msg.sender	Endereço do chamador imediato da função	Usado para `require(msg.sender == owner)`
tx.origin	Endereço da carteira externa que iniciou a transação	**NÃO** deve ser usado para autenticação

A distinção entre `msg.sender` e `tx.origin` é um dos primeiros e mais importantes conceitos de segurança que todo desenvolvedor Solidity deve dominar. Ignorá-la pode levar a explorações devastadoras, pois a confiança é a base de qualquer sistema de segurança.

Tendências Atuais

A Revolução da UX: Abstração de Contas (ERC-4337) e Segurança

O cenário blockchain está em constante evolução, e com ele, surgem novas tecnologias que prometem melhorar a experiência do usuário (UX) e, consequentemente, a segurança. Uma das tendências mais significativas é a **Abstração de Contas (ERC-4337)**.

Modelo Tradicional

- **EOAs (Externally Owned Accounts):** Controladas por chaves privadas
- **Contas de Contrato:** Controladas por código
- Separação rígida entre os dois tipos

ERC-4337

- **Unificação:** Carteiras são smart contracts
- **Lógica Programável:** Recursos avançados nativos
- Flexibilidade e segurança aprimoradas

Recursos Revolucionários

Recuperação Sem Seed Phrase

Carteiras que não exigem uma seed phrase para recuperação, usando mecanismos sociais ou multi-assinatura.

Autenticação Multifator

Autenticação multifator nativa incorporada diretamente na lógica da carteira.

Pagamento de Gás Flexível

Capacidade de pagar taxas de gás em qualquer token, não apenas ETH.

Do ponto de vista da segurança, isso abre um leque de possibilidades para mitigar riscos tradicionais. No entanto, como toda nova tecnologia, a Abstração de Contas também introduz novas considerações de segurança. A complexidade aumenta, e novos pontos de falha podem surgir. O foco passa a ser a segurança do código da carteira de smart contract em si, e como ela interage com o ecossistema. É um avanço que promete mais segurança e flexibilidade, mas exige uma compreensão aprofundada de suas implicações.

Implicações de Segurança da Abstração de Contas

Benefícios de Segurança

Recuperação de Conta

Pode ser implementada através de mecanismos sociais ou de multi-assinatura, eliminando o risco de perda de seed phrase.

Autenticação Multifator

Pode ser incorporada diretamente na lógica da carteira, tornando-a muito mais resistente a ataques de phishing ou roubo de chaves.

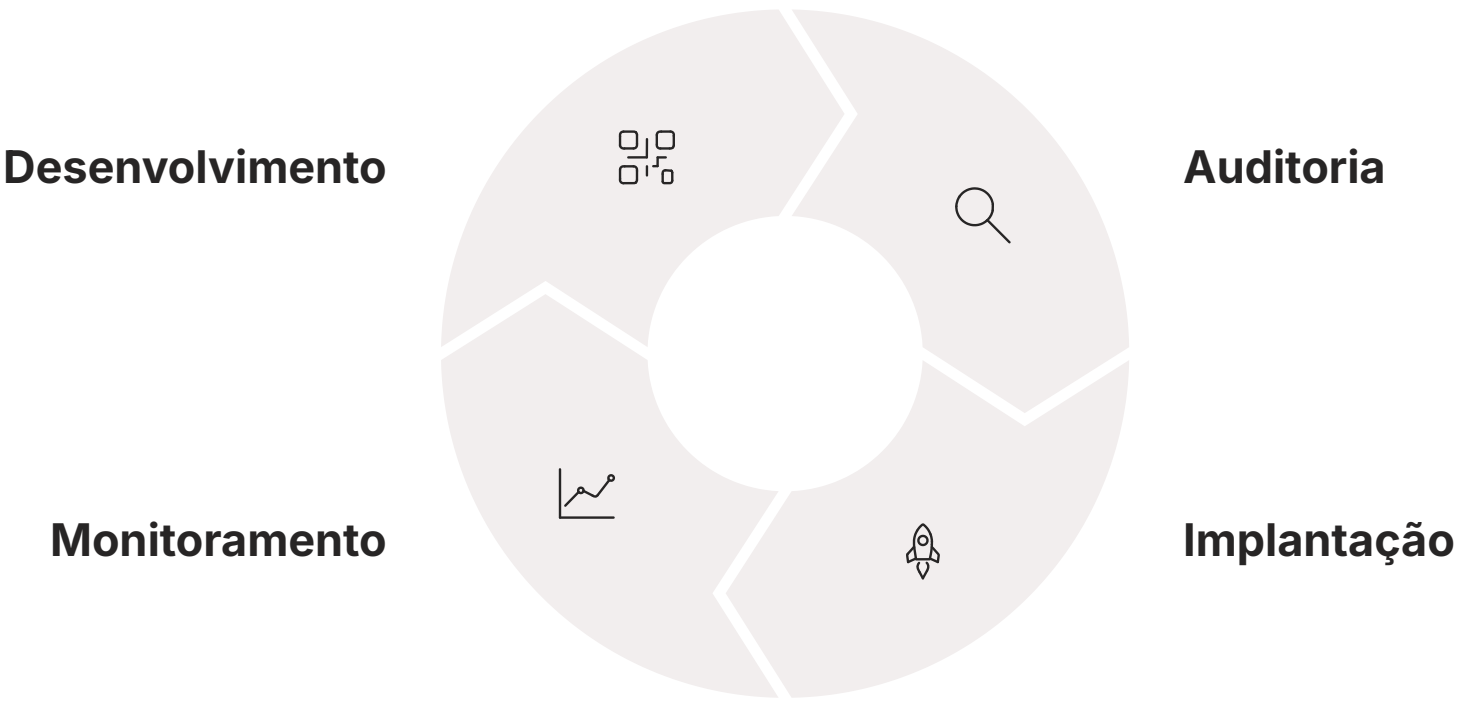
Controles Avançados

Carteiras podem impor limites de gastos, agendamento de transações e até mesmo pausar funcionalidades em caso de atividades suspeitas.

Novos Desafios

⚠️ **Atenção:** A complexidade inerente aos smart contracts significa que as carteiras baseadas em ERC-4337 são suscetíveis às mesmas vulnerabilidades de código que qualquer outro contrato. Erros de lógica, bugs de reentrancy ou falhas na implementação de padrões de segurança podem comprometer os fundos.

A segurança de um sistema de Abstração de Contas dependerá fortemente da auditoria rigorosa do código dos "entry points" e das próprias carteiras de smart contracts.



É uma troca: ganhamos flexibilidade e recursos de segurança avançados, mas introduzimos a necessidade de garantir a segurança do código que implementa esses recursos. A transição para carteiras de smart contracts é um passo crucial para a adoção em massa, mas exige uma vigilância contínua e um profundo conhecimento das melhores práticas de desenvolvimento seguro.

Escalabilidade

Segurança em Soluções de Escalabilidade (Layer 2)

A Ethereum, embora robusta, enfrenta desafios de escalabilidade. Para resolver isso, surgiram as **Soluções de Escalabilidade de Camada 2 (Layer 2s)**, como Optimistic Rollups (Arbitrum, Optimism) e ZK-Rollups (zkSync, StarkNet). Essas soluções processam transações fora da cadeia principal (off-chain) e depois as "empacotam" e as enviam de volta para a Layer 1 (Ethereum) de forma compactada. Isso aumenta drasticamente o throughput e reduz os custos de transação.

Modelos de Segurança

Optimistic Rollups

Confiam que as transações são válidas e usam um "período de desafio" (challenge period) onde qualquer pessoa pode provar que uma transação é fraudulenta. Se ninguém desafiar, a transação é considerada válida.

- Exemplos: Arbitrum, Optimism
- Finalidade atrasada
- Fraud proofs

ZK-Rollups

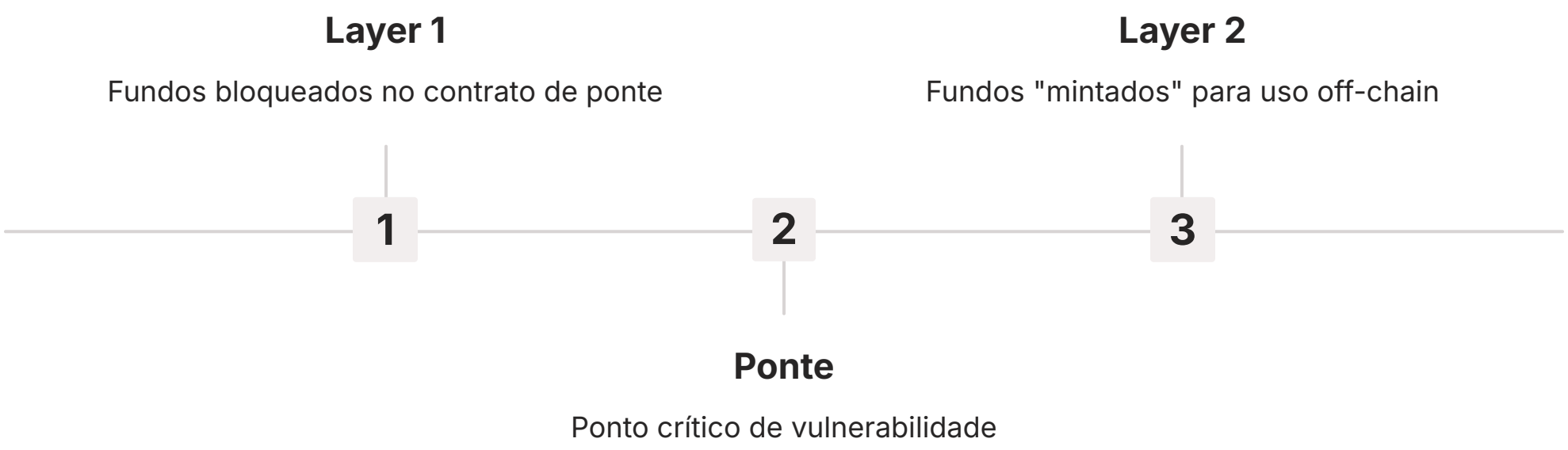
Usam provas criptográficas de validade (Zero-Knowledge Proofs) para garantir que todas as transações processadas off-chain são corretas, sem a necessidade de um período de desafio.

- Exemplos: zkSync, StarkNet
- Finalidade instantânea
- Provas de validade

Do ponto de vista da segurança, as Layer 2s são projetadas para herdar a segurança da Layer 1. No entanto, cada tipo de rollup tem seu próprio modelo de segurança e, consequentemente, seus próprios vetores de ataque potenciais. A complexidade criptográfica dos ZK-Rollups pode introduzir bugs difíceis de detectar.

Desafios de Segurança em Layer 2s

Embora as Layer 2s sejam cruciais para a escalabilidade, elas introduzem novas camadas de complexidade e, com elas, desafios de segurança. Um dos pontos mais críticos são as **pontes (bridges)** que conectam a Layer 1 com a Layer 2. Essas pontes são contratos inteligentes que bloqueiam fundos na Layer 1 e os "mintam" na Layer 2, ou vice-versa. Se uma ponte for explorada, os fundos podem ser roubados ou presos, como já vimos em vários incidentes de alto perfil.



Riscos Específicos por Tipo

Conceito	Âmbito/Aplicação	Foco de Segurança
Optimistic Rollups	Processamento off-chain, finalidade atrasada	Período de desafio, fraud proofs - Prevenção de transações inválidas por desafio
ZK-Rollups	Processamento off-chain, finalidade instantânea	Provas de validade (Zero-Knowledge Proofs) - Correção criptográfica das provas
Pontes (Bridges)	Conexão entre Layer 1 e Layer 2	Contratos inteligentes, mecanismos de bloqueio - Integridade dos fundos, prevenção de roubos/travas

Em Optimistic Rollups, o período de desafio é uma característica de segurança vital. No entanto, se os "fraud proofs" (provas de fraude) forem mal implementados ou se houver uma falha na rede que impeça os validadores de enviar desafios, um operador malicioso poderia potencialmente processar transações inválidas. Nos ZK-Rollups, a segurança depende da correção das provas de conhecimento zero e da implementação do circuito criptográfico. Qualquer bug nesses componentes pode ter consequências catastróficas.

A auditoria e o monitoramento contínuo são ainda mais críticos em ambientes Layer 2, dada a sua complexidade e a interconexão com a Layer 1. A segurança não é apenas sobre o código do smart contract, mas também sobre a infraestrutura, os operadores e os mecanismos de consenso que garantem a integridade das transações.

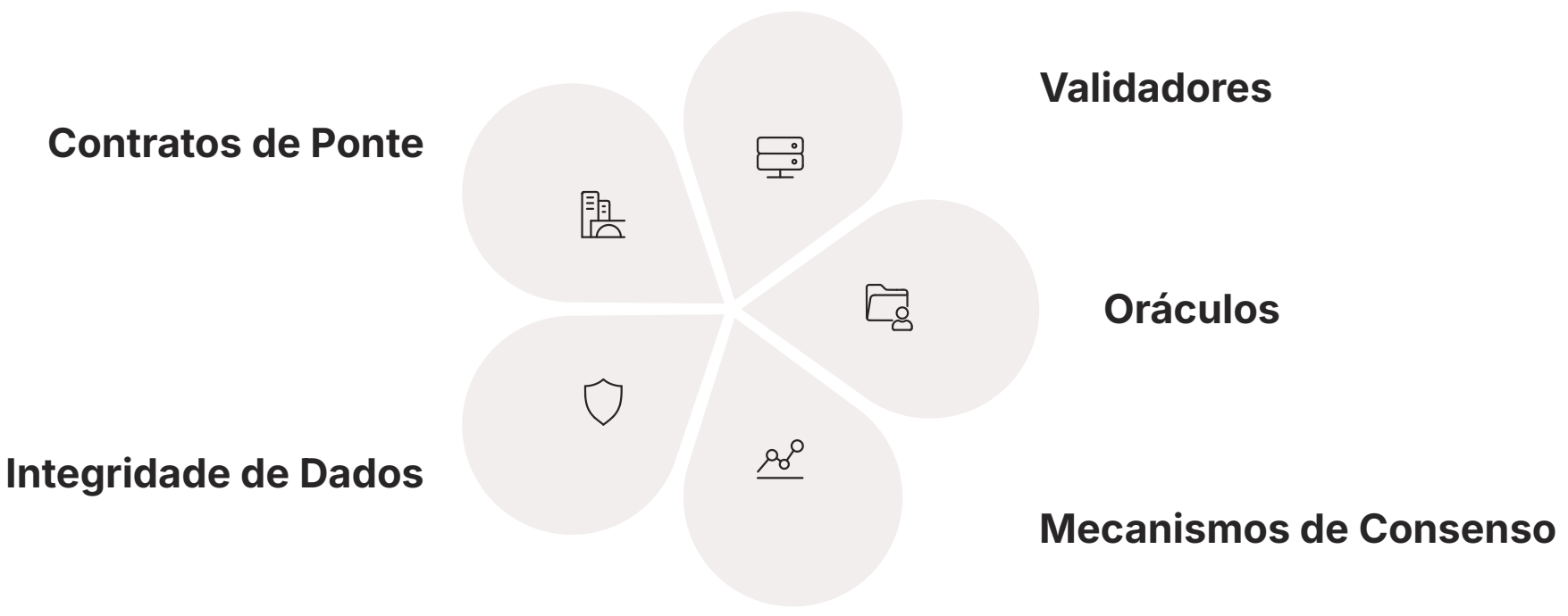
Interoperabilidade

Interoperabilidade e Cross-Chain: Novas Fronteiras de Risco

À medida que o ecossistema blockchain amadurece, a necessidade de comunicação entre diferentes blockchains (interoperabilidade) e a capacidade de mover ativos e dados entre eles (cross-chain) tornam-se cada vez mais importantes. Protocolos como **Chainlink CCIP (Cross-Chain Interoperability Protocol)** e **LayerZero** visam resolver esse desafio, permitindo que smart contracts em uma blockchain interajam com contratos ou dados em outra.

> 🌐 **Analogia:** Imagine que você está enviando uma carta importante através de vários países, cada um com suas próprias regras postais e sistemas de segurança. Se houver uma falha em qualquer ponto dessa jornada, a carta pode ser perdida ou interceptada.

Componentes de Segurança



Essa capacidade de comunicação cross-chain é poderosa, mas também introduz uma nova camada de complexidade e, consequentemente, novos vetores de ataque. Em blockchain, a segurança de uma transação cross-chain depende da segurança de todos os componentes envolvidos: os contratos de ponte em ambas as cadeias, os validadores ou oráculos que retransmitem as mensagens, e os mecanismos de consenso que garantem a integridade dos dados. Uma falha em qualquer um desses elos pode levar a perdas de fundos, duplicação de ativos ou execução de comandos não autorizados.

Mitigando Riscos em Interoperabilidade Cross-Chain

Os desafios de segurança em interoperabilidade cross-chain são multifacetados. Um dos principais é o **problema da confiança**: como um contrato em uma blockchain pode confiar na validade de uma mensagem ou transação originada em outra blockchain, especialmente se elas tiverem diferentes modelos de segurança? Isso é agravado pela possibilidade de ataques de "reprodução" (replay attacks) ou de "duplo gasto" (double-spending) se os mecanismos de validação não forem robustos.

Vetores de Ataque

1

Problema da Confiança

Como validar mensagens entre blockchains com diferentes modelos de segurança

2

Replay Attacks

Reutilização maliciosa de transações válidas em contextos diferentes

3

Double-Spending

Gasto duplo de ativos se mecanismos de validação falharem

4

Falhas de Ponte

Vulnerabilidades nos contratos que conectam as blockchains

Estratégias de Mitigação

Validação Descentralizada

Múltiplos validadores independentes devem concordar sobre a validade de uma mensagem cross-chain

Provas Criptográficas

Uso de provas criptográficas para garantir a integridade dos dados transmitidos

Auditoria Contínua

Auditoria dos contratos de ponte e protocolos de comunicação

Monitoramento em Tempo Real

Detecção de atividades suspeitas através de monitoramento constante

A construção de um ecossistema blockchain verdadeiramente interconectado exige um compromisso inabalável com a segurança em cada camada. A complexidade aumenta exponencialmente com a interoperabilidade, e a compreensão das vulnerabilidades clássicas, combinada com o conhecimento das novas fronteiras de risco, é fundamental para construir um futuro blockchain seguro e resiliente.

Conclusão

Consolidação e Próximos Passos

Chegamos ao fim da nossa exploração das vulnerabilidades clássicas de smart contracts na Parte 2. Percorremos desde a manipulação de oráculos, que nos lembra da fragilidade da confiança em dados externos, até os ataques de negação de serviço por gas limit, que expõem as limitações operacionais da blockchain. Também desvendamos a armadilha do tx.origin, um erro comum de autenticação com consequências graves. Além disso, conectamos esses conhecimentos com as tendências atuais, como a Abstração de Contas e as soluções de escalabilidade, mostrando como a segurança é um campo em constante evolução.

Principais Aprendizados

Oráculos

Sempre validar fontes de dados externas, preferindo oráculos descentralizados e TWAP

Gas Limit

Projetar contratos para evitar operações que possam exceder limites de gás

Autenticação

Nunca usar tx.origin, sempre optar por msg.sender

Tendências

Manter-se atualizado sobre ERC-4337, Layer 2s e interoperabilidade

💡 **Em prática:** Lembre-se de sempre validar suas fontes de dados externas, preferindo oráculos descentralizados e robustos. Projete seus contratos para evitar operações que possam exceder os limites de gás, especialmente em loops ou com estruturas de dados mutáveis. E, acima de tudo, **nunca use tx.origin para autenticação**, optando sempre por msg.sender. Mantenha-se atualizado sobre as implicações de segurança de novas tecnologias como ERC-4337 e Layer 2s.

Avaliação

Autoavaliação

1

Questão 1

Qual das seguintes variáveis deve ser **sempre** utilizada para autenticação em smart contracts, a fim de evitar ataques de phishing via contratos maliciosos?

- a) block.timestamp
- b) tx.origin
- c) msg.sender
- d) block.number

2

Questão 2

Um ataque de negação de serviço (DoS) por gas limit em um smart contract geralmente ocorre quando:

- a) O contrato é sobrecarregado por um grande número de transações simultâneas de usuários legítimos.
- b) Uma função do contrato se torna excessivamente cara para executar devido ao crescimento de uma estrutura de dados manipulável.
- c) O atacante envia transações com um valor de gás muito baixo, impedindo sua inclusão em blocos.
- d) O contrato tenta interagir com um oráculo de preço que está offline.

3

Questão 3

Para mitigar o risco de manipulação de oráculos de preço, qual das seguintes estratégias é mais eficaz?

- a) Utilizar um único oráculo centralizado de alta reputação.
- b) Implementar um período de desafio para todas as atualizações de preço.
- c) Empregar múltiplos oráculos descentralizados e mecanismos como TWAP.
- d) Armazenar todos os preços de ativos diretamente no smart contract.

4

Questão 4

A Abstração de Contas (ERC-4337) impacta a segurança de smart contracts principalmente ao:

- a) Eliminar completamente a necessidade de chaves privadas para todas as contas.
- b) Introduzir novas vulnerabilidades de reentrancy em carteiras de smart contracts.
- c) Permitir que carteiras sejam smart contracts com lógica programável, oferecendo recursos de segurança avançados, mas também novos pontos de falha no código.
- d) Reduzir o custo de gás para todas as transações de carteiras.

5

Questão 5 (Dissertativa)

Explique como a interoperabilidade cross-chain, embora benéfica, introduz novos desafios de segurança e quais são os principais pontos de atenção para desenvolvedores que trabalham com esses protocolos.

Respostas

Gabarito

1

Resposta: c)

msg.sender é a variável correta para autenticação segura

2

Resposta: b)

DoS ocorre quando funções se tornam caras devido a estruturas manipuláveis

3

Resposta: c)

Múltiplos oráculos descentralizados e TWAP são mais eficazes

4

Resposta: c)

ERC-4337 oferece recursos avançados mas introduz novos pontos de falha

Próximos Passos

Continue Sua Jornada

📚 **Próxima Aula:** Na Aula 13, mergulharemos no mundo das **Ferramentas de Análise Estática de Segurança**. Você aprenderá a usar softwares que inspecionam o código do seu smart contract em busca de vulnerabilidades antes mesmo de ele ser implantado, uma etapa crucial para qualquer processo de desenvolvimento seguro.

Recursos Adicionais

Documentação Oficial da Solidity

Para aprofundar nos conceitos de msg.sender e tx.origin

ConsenSys Diligence

Artigos de segurança com estudos de caso e análises detalhadas de vulnerabilidades

Documentação do Chainlink

Para entender a implementação de oráculos descentralizados e CCIP

EIP-4337

Para explorar os detalhes técnicos da Abstração de Contas

NOTA IMPORTANTE: As informações regulatórias/legais/técnicas desta aula estão atualizadas até 2025. Consulte sempre fontes oficiais para verificar alterações.