**Capítulo 19  
Segurança em Blockchain**

Além da escalabilidade e da privacidade, há outro desafio que precisa ser abordado para que as blockchains se tornem ainda mais confiáveis. A questão é a segurança. A segurança da blockchain é crucial porque ela é a espinha dorsal de muitos casos de uso, incluindo os mais sensíveis, como saúde, finanças e gerenciamento da cadeia de suprimentos. Com o advento das finanças descentralizadas (DeFi), tornou-se ainda mais importante garantir o funcionamento correto da blockchain para que não restem vulnerabilidades que possam ser exploradas, resultando em perda de fundos ou outras consequências indesejadas.

Como o ecossistema blockchain é composto por várias camadas, existem preocupações de segurança em cada uma delas, e discutiremos isso neste capítulo. Também há preocupações de segurança relacionadas à camada 2 (side chains, etc.), sobre as quais também lançaremos alguma luz neste capítulo.

Neste capítulo, cobriremos os seguintes tópicos:

* Segurança em blockchain
* Histórico e ataques anteriores
* Modelo em camadas da blockchain
* Ameaças e vulnerabilidades em cada camada da blockchain, incluindo segurança de contratos inteligentes, segurança da camada blockchain e segurança em outras camadas, como abordá-las e melhores práticas
* Preocupações de segurança na camada 2
* Ferramentas e técnicas para encontrar vulnerabilidades
* Modelos para realizar análise de ameaças

Então, vamos começar analisando primeiro o tema da segurança no contexto das blockchains.

**Segurança**

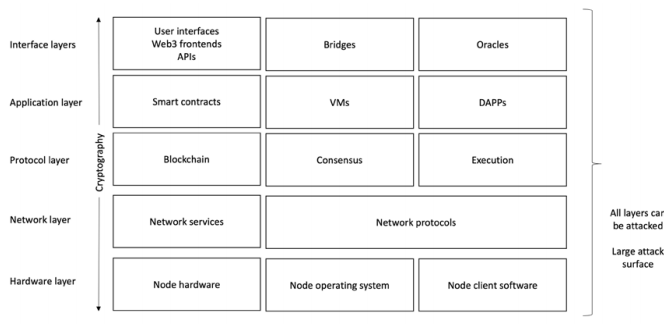
As blockchains são geralmente seguras e fazem uso de criptografia simétrica e assimétrica, conforme necessário em toda a rede blockchain, para garantir que a camada central da blockchain seja segura. No entanto, ainda existem algumas ressalvas que podem comprometer a segurança da blockchain.

Houve muitos ataques bem-sucedidos e de alto perfil ao ecossistema blockchain ao longo dos anos. Alguns foram maliciosos, outros acidentais; no entanto, todos resultaram em perdas significativas. Alguns exemplos incluem:

* O ataque à Binance Smart Chain, ocorrido em 6 de outubro de 2022. Os hackers infiltraram-se na ponte cross-chain BNB chamada Token Hub, onde conseguiram cunhar tokens BNB fraudulentos por meio de provas de saque falsas. Inicialmente, o dano estimado foi de 566 milhões de dólares.
* Em agosto de 2021, o roubo da Poly Network, que resultou no roubo de 610 milhões de dólares. Esse ataque explorou uma vulnerabilidade que ocorreu devido ao gerenciamento incorreto de direitos de acesso entre contratos inteligentes da Poly.
* Em fevereiro de 2022, o exploit da ponte Wormhole, que resultou na perda de 321 milhões de dólares. Isso ocorreu quando um hacker encontrou uma vulnerabilidade nos contratos inteligentes que permitiu cunhar 120.000 WETH na Solana sem lastro colateral, e então trocá-los por ETH.
* Em março de 2022, a ponte Ronin foi explorada em cerca de 612 milhões de dólares. A razão para esse exploit foi o acesso não autorizado a chaves privadas, o que resultou na violação de nós validadores, os quais aprovaram transações fraudulentas, drenando os fundos da ponte.
* O colapso da Luna em maio de 2022, que eliminou cerca de 60 bilhões de dólares do mundo das criptomoedas. Isso ocorreu principalmente devido a um algoritmo inadequado por trás da stablecoin Luna, onde o UST era garantido pela própria Luna, em vez de uma moeda fiduciária.

Outros hacks incluem MtGox, KuCoin, Pancake Bunny, hack da carteira Parity e muitos mais. Também houve perdas devido à má gestão e controles inadequados, como:

* Colapso da FTX: A FTX era uma das maiores exchanges de criptomoedas do mundo. Ela entrou em colapso em novembro de 2022 após um aumento nos saques de clientes que a FTX não conseguiu atender por não possuir ativos suficientes em reserva. Além disso, a situação foi agravada pelo fato de as carteiras da FTX terem sido alvo de ataques e hackeadas, o que resultou no roubo de quase 640 milhões de dólares. Acredita-se que tenha sido um ataque interno por funcionários ou o resultado de um ataque de malware. Eventualmente, a FTX entrou com pedido de falência devido à crise de liquidez. O colapso da FTX deixou mais de 1 milhão de clientes incapazes de sacar ativos no valor estimado de 8 bilhões de dólares.
* Problema com o dinheiro de clientes da Binance: Em janeiro de 2023, a Binance acidentalmente misturou os fundos dos clientes da exchange Crypto.com com colaterais de B-Token. Felizmente, esse erro foi percebido antes que qualquer dano material fosse causado, mas destaca a importância dos controles e requisitos de conformidade. Esse problema, se não fosse detectado anteriormente, poderia ter levado os proprietários a não conseguirem sacar dinheiro devido à falta de liquidez/fundos pela exchange, já que não há segregação de ativos entre o dinheiro dos clientes e qualquer colateral usado. Essa situação poderia ter levado a uma perda substancial de reputação e financeira devido à falta de liquidez, se não tivesse sido percebida.

****

**Figura 19.1: Visão em camadas da blockchain**

No diagrama anterior, observe que todas as camadas podem ser atacadas por um adversário e, como tal, a superfície de ataque é bastante grande. Em outras palavras, isso significa que a superfície de ataque se estende por várias camadas, e um invasor pode tentar atacar todas as camadas, incluindo hardware, rede, protocolo, aplicações e camadas de interface. Vamos agora ver do que é composta cada camada. Observe que um invasor pode tentar encontrar vulnerabilidades em cada camada e em todos os seus componentes. Além disso, como o ganho financeiro desses ataques pode ser significativo, hackers podem estar motivados a atacar e tentar diferentes formas de explorar qualquer vulnerabilidade possível.

**Camadas da blockchain e ataques**

Nesta seção, veremos como podemos enxergar a blockchain como uma arquitetura em camadas para estudar questões de segurança, ataques e mitigação.

* **Camada de hardware:** Esta é a camada de hardware base na qual um nó da rede blockchain é executado.
* **Camada de rede:** Esta é a camada de rede principal na qual os protocolos de blockchain são executados. Esta camada inclui protocolos como TCP/IP, P2P, UDP e outros serviços e protocolos relevantes.
* **Camada de protocolo da blockchain:** Esta é a camada da blockchain onde os processos de consenso, execução de transações, gerenciamento de incentivos e mineração são executados.
* **Camada de aplicação da blockchain:** Esta é a camada que consiste em contratos inteligentes e máquinas virtuais, por exemplo, EVMs para executar contratos inteligentes. DAPPs também existem nesta camada, entre outros.
* **Camada de interface:** Esta camada inclui pontes, oráculos, governança, interfaces de usuário para DAPPs Web3 e APIs. Embora esses componentes possam, possivelmente, existir na camada de aplicação da blockchain, acredito que diferenciá-los aqui como uma camada separada ajuda a definir e abordar desafios específicos de forma mais eficaz.
* **Camada de criptografia:** A criptografia desempenha um papel vital em cada camada. Aqui, trataremos essa camada separadamente apenas para fins de discussão; caso contrário, ela existe de alguma forma em todas as camadas do ecossistema blockchain.

**Segurança da Blockchain**

Cada camada está exposta a adversários que tentam explorar quaisquer fraquezas ou limitações. A blockchain é particularmente interessante porque, embora preocupações convencionais de segurança de software, hardware e informação em geral se apliquem — já que ela utiliza hardware e software convencionais para funcionar —, ela também possui seus próprios desafios, como a segurança de contratos inteligentes. O desafio é proteger o ecossistema blockchain em todas as camadas, especialmente nas camadas de protocolo e de aplicação da blockchain.

Em resumo, podemos dizer que há muito o que deve ser protegido em um ecossistema blockchain, incluindo a camada 1, camada 2, contratos inteligentes, máquinas virtuais, carteiras e até mesmo pessoas. Por que pessoas? Porque elas podem ser vítimas de ataques de engenharia social. Além disso, as pessoas precisam ser educadas sobre essas ameaças para que possam identificar padrões de engenharia social e se protegerem.

A criptografia desempenha um papel importante em todas as camadas; portanto, discutiremos a criptografia como uma camada/tópico separado.

Agora vamos discutir os vetores de ataque e as formas de mitigação em cada camada.

**Camada de hardware**

Como esta camada é composta por hardware que executa o sistema operacional e o software do nó, ela é impactada pelas ameaças usuais, incluindo vírus, malware e acesso não autorizado. Também existe malware específico para blockchain, que tem como alvo específico os nós da rede de blockchain de criptomoedas. Esse tipo de malware, conhecido como malware de cripto, é especialmente projetado para realizar *cryptojacking*. *Cryptojacking* pode ser definido como o uso não autorizado de recursos computacionais de outra pessoa para minerar criptomoedas.

*Cryptojacking* é um método de tomar o controle de um computador ou navegador da web para minerar criptomoedas sem a permissão do usuário. Um ataque de *cryptojacking* bem-sucedido não necessariamente resulta na instalação de software no computador da vítima, pois isso pode ocorrer diretamente dentro dos navegadores da web — a chamada "mineração no navegador". Isso acontece quando um usuário acessa um site que executa um script de mineração em seu navegador. Mesmo quando o usuário sai do site, o script de mineração pode continuar sendo executado.

O malware utilizado para *cryptojacking* é chamado de malware de cripto. Ele pode infectar sistemas operacionais como Windows, Linux, macOS, iOS e Android. Extensões de navegador às vezes também incluem JavaScript para minerar criptomoedas, por exemplo, Archive Poster e Iridium. Devido ao *cryptojacking*, o desempenho da máquina é significativamente reduzido, pois CPUs e GPUs estão executando o processo de mineração em vez de executar as operações habituais esperadas no sistema. Como sabemos, a mineração é um processo que consome muitos recursos, especialmente com Prova de Trabalho (PoW), por isso pode ser vantajoso para hackers invadir o computador de outra pessoa e executar a mineração a partir desse computador.

Também pode haver um ataque de Negação de Serviço (DoS) a um nó da blockchain, o que pode fazer com que os recursos no computador tenham dificuldades para acompanhar os pedidos recebidos. Em um ataque DoS, o invasor inunda o nó com um grande número de requisições, sobrecarregando-o e impedindo-o de processar transações legítimas.

Também podem ocorrer ataques de execução remota por meio de portas abertas, não apenas no sistema operacional, mas também no software do nó, por exemplo, a porta RPC 8332 dos nós do Bitcoin, que hackers podem tentar explorar. Uma lista de vulnerabilidades está disponível aqui para Bitcoin e Ethereum:

* Bitcoin: <https://www.cvedetails.com/vulnerability-list/vendor_id-12094/Bitcoin.html>
* Ethereum: <https://www.cvedetails.com/vulnerability-list/vendor_id-17524/Ethereum.html>

Pode haver outras vulnerabilidades que ainda não foram descobertas ou que podem ser introduzidas com atualizações futuras.

Há também a possibilidade de o sistema operacional estar desatualizado ou mesmo de o software do nó estar desatualizado, o que pode abrir portas para exploração por hackers. Além disso, sistemas operacionais ou softwares clientes de nós mal configurados podem levar a uma fraqueza que hackers podem explorar. O hardware no qual o nó está sendo executado também pode ser vulnerável devido a falhas como Spectre, Meltdown e Thunderclap. O firmware também pode ser explorado devido a fraquezas como Thunderstrike, ROCA, etc. É importante que medidas padrão de precaução e segurança, como endurecimento do sistema, atualizações do sistema operacional, atualizações do software do nó e software antivírus, sejam aplicadas ao nó de hardware que hospeda o software cliente da blockchain.

O software do nó blockchain pode ser atacado por malware tradicional e específico de blockchain, o que pode resultar no roubo de chaves privadas, criação de transações desonestas, censura de algum tráfego e exaustão de recursos, resultando em um DoS.

**Camada de rede**

Podem ser realizados vários ataques na camada de rede. Discutiremos eles a seguir:

1. **Ataque Sybil:** Primeiro, introduzimos o ataque Sybil. Um ataque Sybil é um tipo de ataque de rede em que um invasor cria e usa várias identidades falsas, ou "nós Sybil", para obter uma vantagem injusta ou realizar ações maliciosas dentro de uma rede. Isso pode ser feito criando várias identidades falsas e usando-as para obter um nível desproporcional de influência dentro da rede, como participando de um processo de consenso ou sistema de votação. O objetivo de um ataque Sybil é manipular a rede ou interromper seu funcionamento normal, frequentemente para ganho pessoal ou financeiro. Se identidades falsas suficientes forem criadas, esse ataque pode ser usado para superar em número os nós honestos da rede. Esses nós então podem assumir o controle da rede e recusar-se a receber ou transmitir um bloco, efetivamente tornando a rede inútil. Se os atacantes conseguirem controlar mais de 51% da taxa de hash, então eles podem reordenar as transações, censurar transações e efetivamente assumir o controle de toda a rede. Protocolos de consenso são projetados de forma que ataques Sybil não sejam possíveis, embora em alguns casos, devido a fraquezas e mau design, ataques Sybil ainda possam funcionar.
2. **Ataque DoS:** A rede também pode ser impactada por DoS, o que pode resultar em sobrecarga da rede com tráfego desnecessário, tornando-a inacessível aos usuários. Embora a blockchain seja uma rede descentralizada ponto a ponto (P2P) sem ponto único de falha (SPOF), essas redes são geralmente imunes a ataques de DoS de rede. No entanto, alguns pontos de centralização existem na maioria das blockchains, os quais podem ser alvos de um ataque DoS; por exemplo, nós de boot podem ser atacados, os quais são conhecidos e codificados no software cliente da blockchain. Claro, aqui podem ser utilizados mecanismos tradicionais de proteção de rede contra DoS, como firewalls, sistemas de detecção e prevenção de intrusos, balanceadores de carga, limitadores de taxa e técnicas de modelagem de tráfego.
3. **Ataque Eclipse:** Um ataque eclipse é um tipo de ataque a uma rede blockchain no qual um invasor tenta isolar um nó específico ou alguns nós do restante da rede, efetivamente “eclipsando” eles e tornando-os incapazes de se comunicar com outros nós, impedindo-os de receber novas informações.

**Segurança da Blockchain**

Nesse ataque, um nó malicioso pode controlar rotas de comunicação entre dois segmentos da rede blockchain, o que permite a esse nó controlar o fluxo de informação entre esses dois segmentos da rede. Esse nó pode parar de encaminhar transações e blocos para um nó vítima, resultando em seu “eclipse”. Um ataque eclipse pode resultar em DoS porque o nó vítima não consegue receber nem processar novas transações. Também pode resultar em gasto duplo, pois se o nó alvo não consegue se comunicar com o restante da rede, ele não tem conhecimento das transações que foram transmitidas e confirmadas por outros nós. Isso pode permitir que o atacante gaste seus ativos digitais duas vezes, pois o nó alvo não terá a informação mais atualizada sobre a blockchain.

1. **Falsificação de rede:** Outro ataque pode ser a falsificação de rede (*network spoofing*). Em redes blockchain como Ethereum e Bitcoin, nós de boot são codificados diretamente no software cliente do nó e são usados para fornecer um ponto de partida para que novos nós se conectem à rede e comecem a procurar nós vizinhos. É possível que um invasor lance um ataque de “falsificação” contra um nó de boot se passando por ele. Nós novos se conectarão normalmente a esse nó de boot falso, o que pode resultar em um ataque de negação de serviço e outras implicações de segurança.

**Camada da blockchain**

A camada da blockchain representa a camada 1 ou camada 2 da blockchain que constitui o núcleo do protocolo. Vários ataques podem ser realizados aqui.

**Ataques a transações**  
Diversos ataques podem ser realizados sobre transações. As transações são construídas pelos usuários, e elas podem ser malformadas ou inválidas. Como as transações são propagadas para todos os nós e todos os nós processam todas as transações (exceto nós leves), transações desonestas ou especialmente construídas para causar dano podem se tornar um vetor de ataque atrativo para hackers. A transação pode conter uma estrutura de dados peculiar que pode fazer com que os nós apresentem mau funcionamento. Normalmente, isso acontece devido a um bug no software do nó, mas teoricamente é possível que um vírus ou código malicioso se espalhe por meio de transações em uma rede blockchain.

Embora transações inválidas e incorretamente assinadas com assinaturas inválidas sejam rejeitadas por todos os nós, é possível que a transação esteja corretamente assinada e passe em todas as verificações antes da execução, e acabe executando algum código desonesto que resulte em um efeito indesejável. Um exemplo ocorreu em 2010, quando mais de 184 bilhões de Bitcoins foram criados do nada devido a uma vulnerabilidade de estouro de inteiro no software do nó central do Bitcoin. Alguns ataques relevantes incluem a *maleabilidade de transação*, que permite a um hacker modificar o script de desbloqueio de uma transação pendente e ainda assim passar nas verificações de validação da transação.

Esse ataque permitiria alterar o ID da transação, o que poderia levar a uma situação em que o destinatário da transação poderia alegar que nunca recebeu os fundos, enganando o remetente para enviar o pagamento novamente. O software central do Bitcoin agora é bastante estável e passou por várias correções de bugs, atualizações e testes rigorosos, como a atualização do Segregated Witness, que corrigiu o problema de maleabilidade de transação. Portanto, tais ataques são bastante difíceis de executar em uma rede Bitcoin. No entanto, outras blockchains de criptomoedas podem conter vulnerabilidades *zero-day* que podem ser exploradas por meio de transações malformadas. Mesmo o Bitcoin ainda pode ter algumas vulnerabilidades desconhecidas.

A serialização e desserialização de dados para transações e blocos é uma operação padrão que os nós blockchain executam. É possível que, se alguma estrutura de dados contiver código malicioso, quando ela for desserializada por um nó, isso possa resultar em estouros de inteiros, estouros de buffer ou até ataques de negação de serviço (DoS). Os nós poderiam acabar executando código malicioso se estiverem executando um software antigo que contenha um bug específico relacionado a essa vulnerabilidade, onde, após a desserialização, o código não verifica todas as condições.

Também é possível, devido à capacidade de codificar dados arbitrários em uma transação, que isso possa ser realizado. Por exemplo, por meio do campo Coinbase de uma transação do Bitcoin, do código OP\_RETURN, ou de alguma outra técnica relevante. Com essa capacidade de embutir dados arbitrários em uma transação, é possível que uma transação especial seja criada que, ao ser executada, possa resultar no mau funcionamento não apenas de nós, mas também de outros softwares que leem transações, como exploradores de blockchain, ferramentas de monitoramento de terceiros e alguns softwares de backend corporativo que leem transações de uma blockchain privada ou mesmo de uma blockchain pública.

Como as blockchains possuem outros softwares periféricos, incluindo exploradores de blockchain, frontends de DAPPs e outras interfaces de usuário, é possível, utilizando ataques de injeção, atacar a blockchain. Por exemplo, validação de entrada inadequada, ataques de cross-site scripting ou entradas inapropriadas, como pedir ao usuário para inserir suas chaves privadas em uma interface web, podem levar à exposição da chave privada e outras consequências indesejadas, como perda de fundos, execução indesejada de contratos inteligentes, execução de transações desonestas ou um ataque DoS. Ataques semelhantes podem ser realizados no nível dos blocos também.

**Ataques de repetição de transações**  
Um ataque de repetição é um tipo de vulnerabilidade que surge quando duas criptomoedas separadas, baseadas em um fork da mesma cadeia original, permitem que transações sejam reconhecidas como válidas em ambas as cadeias.

Em um ataque de repetição de transação, um invasor captura uma transação válida de uma rede e tenta reutilizá-la ao “repeti-la” na mesma ou em outra rede. Isso pode ser feito por um invasor que obteve acesso aos dados da transação, por exemplo, interceptando-a na rede ou acessando um dispositivo que foi usado para iniciar a transação. Se a transação for repetida em uma rede diferente, ela pode ser potencialmente usada para transferir fundos ou outros ativos para a conta do invasor.

Ataques de repetição podem ser um problema em redes que usam as mesmas chaves para assinar transações em redes diferentes, ou que não possuem um mecanismo para prevenir a reutilização de transações. Para se proteger contra ataques de repetição, é importante usar chaves únicas para assinar transações em redes diferentes e implementar um mecanismo para detectar e prevenir a reutilização de transações.

Em 2016, o Ethereum sofreu um ataque de repetição de transação quando a rede Ethereum passou por um hard fork para resolver o problema do “hack do DAO”. Um hard fork é uma mudança no protocolo de uma rede blockchain que não é retrocompatível, significando que todos os nós na rede devem atualizar para a nova versão do protocolo, ou serão incapazes de participar da rede.

Durante o hard fork, a rede Ethereum se dividiu em duas redes separadas: Ethereum (ETH) e Ethereum Classic (ETC). Isso significava que transações em uma rede não seriam reconhecidas na outra, e vice-versa. No entanto, como ambas as redes estavam usando as mesmas chaves para assinar transações, um invasor conseguiu capturar uma transação válida em uma rede e repeti-la na outra rede.

Para corrigir esse problema, o Ethereum implementou um novo algoritmo de assinatura de transação sob a EIP-155, que incluía um ID de cadeia nos dados assinados. Esse ID de cadeia é único para cada rede e permite que a rede verifique se uma transação foi destinada a ela e não a outra rede. Como resultado, ataques de repetição de transações não são mais possíveis no Ethereum.

Recentemente, ataques de repetição voltaram aos holofotes após o *merge* (a mudança do Ethereum de Prova de Trabalho para Prova de Participação) em setembro de 2022. Para evitar ataques de repetição, o ID de cadeia da antiga rede Ethereum PoW bifurcada foi alterado de 1 para 10.001. Embora essa mudança proteja contra ataques de repetição na cadeia principal, ela não protege contra algumas brechas que surgiram como resultado da não atualização de contratos inteligentes, DAPPs e outros componentes periféricos ao redor da cadeia principal. Muitos contratos continuam vulneráveis a ataques de repetição.

Um exemplo importante é um ataque aos contratos inteligentes da Omni Bridge. Esse ataque de repetição foi lançado contra a antiga cadeia Ethereum PoW e a Omni Bridge e resultou na perda de 200 WETH da cadeia Ethereum PoW. O invasor primeiro transferiu 200 WETH da nova cadeia Ethereum PoS (após o *merge*) e isso foi repetido na cadeia Ethereum PoW. A Omni Bridge não validou o ID da cadeia antes de aprovar a transação, o que resultou na perda de 200 WETH da cadeia PoW e no ganho de 200 WETH pelos atacantes. Uma coisa a se notar é que este não foi um ataque de repetição verdadeiro, mas uma repetição de *calldata*; no entanto, isso destaca a importância de garantir que, sempre que houver atualizações e outras mudanças no protocolo principal, a proteção contra ataques de repetição seja incorporada nos novos protocolos. A transação não foi um ataque de repetição na cadeia, mas sim um ataque de repetição de *calldata*, que ocorreu devido a uma falha em um contrato específico na cadeia Gnosis. Isso levanta uma preocupação geral sobre sistemas periféricos, especialmente pontes, que devem verificar o ID de cadeia correto das mensagens entre cadeias. Isso também pode ser visto como um ataque na camada de interface, que discutiremos em breve.

As pontes blockchain tornaram-se um alvo para hackers. Falaremos mais sobre isso em breve, quando discutirmos a camada de interface.

**Ataques aos protocolos de consenso**  
Vários ataques podem ser realizados contra os protocolos de consenso em blockchain, incluindo:

1. **Ataque de 51%:** Isso ocorre quando um minerador ou grupo de mineradores controla mais de 50% do poder de mineração na rede, permitindo-lhes controlar a confirmação de transações e, potencialmente, realizar gastos duplos.
2. **Mineração egoísta (*Selfish mining*):** Isso ocorre quando um minerador retém os blocos que minera da rede a fim de aumentar suas chances de encontrar o próximo bloco e receber a recompensa do bloco.
3. **Ataque eclipse:** Isso ocorre quando um invasor consegue isolar um nó do restante da rede, permitindo-lhe controlar as informações que o nó recebe e, potencialmente, manipular o processo de consenso.
4. **Ataque Sybil:** Isso ocorre quando um invasor cria múltiplas identidades ou “sybils” para controlar uma parte significativa da rede e influenciar o processo de consenso.
5. **Ataque *nothing-at-stake*:** Isso ocorre em blockchains baseadas em Prova de Participação (PoS) onde os validadores não são obrigados a oferecer colateral, permitindo-lhes votar em múltiplas cadeias e, potencialmente, minar a integridade da rede.

Há vários ataques contra o mecanismo de Prova de Participação (PoS):

1. **Problema *nothing-at-stake*:** Esse ataque ocorre quando há apenas recompensas em um mecanismo PoS, e nenhuma penalidade. Essa situação permissiva pode resultar em validadores atestando quaisquer blocos e/ou múltiplos forks/blocos de uma blockchain para aumentar suas recompensas.
2. **Ataque de moagem de participação (*Stake grinding*):** Nesse ataque, um invasor tenta manipular o algoritmo de seleção de validadores para favorecer a seleção de seus próprios validadores. Isso resultaria na seleção de seus validadores mais do que o justo, o que levaria à cunhagem de recompensas injustificadas.

**Gasto duplo**  
Em um ataque de gasto duplo, um usuário malicioso envia a mesma criptomoeda ou token para dois ou mais destinatários diferentes, efetivamente gastando os mesmos fundos duas vezes. Há diferentes tipos de ataques que podem resultar em gasto duplo. Esses ataques incluem ataque de corrida (*race attack*), ataque Finney e ataque de 51%.

Um ataque de corrida é um tipo de ataque de gasto duplo que ocorre quando uma transação rápida é transmitida antes de uma transação mais lenta que gasta as mesmas moedas.

Em um ataque Finney, um ator malicioso primeiro envia uma transação a um comerciante por uma certa quantia de criptomoeda e então rapidamente minera um novo bloco que não inclui essa transação. O atacante então envia uma segunda transação gastando as mesmas moedas para um destinatário diferente antes que a primeira transação possa ser confirmada pela rede. Como o atacante pode minerar o novo bloco e validar a segunda transação antes que a primeira transação seja confirmada, o comerciante fica com uma transação inválida e sem pagamento. Ataques Finney são de baixo risco, pois exigem que o atacante tenha poder computacional significativo para minerar um novo bloco antes que a primeira transação seja confirmada. No entanto, eles ainda são uma ameaça potencial e devem ser considerados ao projetar e implementar sistemas de criptomoeda.

**Mineração egoísta**

Um ataque de mineração egoísta pode ocorrer quando um invasor consegue resolver a prova de trabalho (PoW) e minerar um bloco, mas, em vez de anunciá-lo à rede, o retém da cadeia pública. Isso criaria um fork, e o invasor pode então continuar minerando na nova cadeia bifurcada, conhecida apenas por ele, para construir uma cadeia alternativa e superar a blockchain pública. Quando essa cadeia maliciosa estiver à frente da cadeia honesta/pública, o bloco mais recente pode ser liberado para a rede. Como a rede está programada para aceitar o bloco mais recente, o fork malicioso substituirá a cadeia honesta/original. Esse ataque pode permitir alcançar praticamente qualquer tipo de objetivo malicioso, incluindo gasto duplo, roubo de criptomoeda, censura de transações e objetivos semelhantes.

**Bifurcação e reorganização da cadeia**

A bifurcação em blockchain refere-se à criação de duas ou mais cadeias paralelas a partir de uma única blockchain. Isso pode ocorrer quando múltiplos validadores na rede geram blocos simultaneamente, levando a duas ou mais cadeias de blocos diferentes entre si. Reorganização da cadeia refere-se ao processo de alteração da cadeia mais longa atual em uma rede blockchain. Isso pode ocorrer se uma cadeia mais longa se tornar disponível após uma divisão da rede ou se um minerador malicioso criar uma cadeia alternativa mais longa.

Um invasor pode induzir esses ataques utilizando diversos métodos, como o ataque *nothing-at-stake*, o ataque de longo alcance (*long-range attack*), ou o ataque de mineração egoísta. Em tais ataques, um validador malicioso pode tentar manipular a blockchain criando múltiplas cadeias ou alterando a cadeia mais longa, com o objetivo de comprometer a integridade e a segurança da rede.

**Camada de aplicação da blockchain**

Existem diversas vulnerabilidades que podem existir nesta camada. As mais proeminentes estão relacionadas a contratos inteligentes. O que torna a segurança de contratos inteligentes particularmente interessante é que não há correções (patches) após a implantação, o que é diferente do software tradicional. Um padrão de proxy pode ajudar aqui, mas lembre-se de que é melhor evitar bugs.

**Vulnerabilidades em contratos inteligentes**

Diversos bugs de segurança em contratos inteligentes foram descobertos e analisados em campo. Eles incluem dependência da ordenação de transações, dependência de timestamp, tratamento incorreto de exceções, como exploração do limite de profundidade da pilha de chamadas, e vulnerabilidade de reentrada. Os ataques podem ser classificados como ações maliciosas, design inadequado (de protocolo ou aplicativo), bugs e engenharia social. Agora apresentamos uma lista de vulnerabilidades em contratos inteligentes:

1. **Bug de dependência de ordenação de transações** explora cenários onde o estado percebido de um contrato pode não ser o mesmo que o estado real após a execução. Essa fraqueza é um tipo de condição de corrida. Também é chamada de *front-running* e é possível devido ao fato de que a ordem das transações dentro de um bloco pode ser manipulada. Como todas as transações primeiro aparecem no *memory pool*, elas podem ser monitoradas antes de serem incluídas no bloco. Isso permite que uma transação seja enviada antes de outra, controlando assim o comportamento de um contrato inteligente.
2. **Bugs de dependência de timestamp** são possíveis em cenários onde o timestamp do bloco é usado como fonte para alguma tomada de decisão dentro do contrato, mas timestamps podem ser manipulados pelos mineradores (produtores de blocos). O limite de profundidade da pilha de chamadas é outro bug que pode ser explorado devido ao fato de que a profundidade máxima da pilha de chamadas da EVM é de 1.024 frames. Se a profundidade da pilha for atingida enquanto o contrato está sendo executado, então em certos cenários, a instrução de envio ou chamada (*send* ou *call*) pode falhar, resultando na não realização do pagamento. O bug de profundidade da pilha de chamadas foi tratado no *hard fork* da EIP-150 em:  
   <https://github.com/ethereum/EIPs/blob/master/EIPS/eip-150.md>
3. **Problema de falha ao enviar (*send fail*)**: Ao enviar fundos para outro contrato, o envio pode falhar, e mesmo se *throw* for usado como mecanismo de captura geral, ele não funcionará.
4. **Dependência de timestamp** é outra vulnerabilidade bastante comum. Normalmente, o timestamp de um bloco é acessado via now ou block.timestamp, mas esse timestamp pode ser manipulado por mineradores, influenciando assim o resultado de uma função que depende de timestamps. Frequentemente, isso é usado como fonte de aleatoriedade em jogos de loteria para selecionar o próximo vencedor. Assim, pode ser possível para um minerador modificar o timestamp de forma a aumentar suas chances de se tornar o próximo vencedor.
5. **Outros bugs usuais de software**, como estouro e subtração de inteiros (*overflow* e *underflow*), também são bastante significativos, e qualquer uso de variáveis inteiras deve ser cuidadosamente implementado em Solidity.

Por exemplo, um programa simples onde uint8 é usado para iterar por elementos de um array com mais de 255 elementos pode resultar em um loop infinito. Isso ocorre porque uint8 é limitado a 256 valores.

1. **Reentrada (*Reentrancy*)** é um bug comum em contratos inteligentes. Esse bug foi explorado no ataque ao DAO para desviar milhões de dólares para um DAO filho. O bug de reentrada essencialmente significa que uma função pode ser chamada repetidamente antes que a invocação anterior (primeira) da função tenha sido concluída. Em outras palavras, o ataque de reentrada ocorre quando um contrato controlado pelo invasor consegue chamar de volta um contrato inteligente antes da conclusão da primeira chamada da função.

Isso é particularmente perigoso em funções de saque de Ether em contratos Solidity.

1. **Selfdestruct sem proteção adequada:** Se não houver controles de acesso apropriados implementados, qualquer pessoa pode destruir o contrato chamando *selfdestruct*. Isso ocorreu de forma notória na carteira Parity, onde um usuário acidentalmente acionou a função *selfdestruct* no código do contrato inteligente da Parity, o que resultou na perda de 513.774,16 Ether. Mesmo sendo baseada em assinatura múltipla e parecendo ser um contrato seguro, a função *selfdestruct* não estava protegida e, ao ser chamada, resultou na exclusão do código.
2. **Escrita sem privilégio em armazenamento:** Esse ataque ocorre quando um contrato inteligente não valida e restringe adequadamente o acesso às suas variáveis de estado.
3. **Subtração e estouro de inteiros:** Esses erros ocorrem quando uma operação é realizada em um inteiro que resulta em um valor fora do intervalo do tipo de dado. Por exemplo, em um inteiro sem sinal de 256 bits, o valor mínimo é 0 e o valor máximo é 2^256-1. Se uma operação subtrair 1 de 0, isso resulta em *underflow*; se adicionar 1 a 2^256-1, resulta em *overflow*. Um atacante pode usar *underflow* e *overflow* para manipular o estado de um contrato ou fazer com que ele se comporte de forma inesperada. Para evitar esses problemas em Solidity, os desenvolvedores podem usar bibliotecas como SafeMath, que verificam automaticamente *underflow* e *overflow* e revertem a transação se isso ocorrer. Além disso, os desenvolvedores podem usar tipos de dados específicos, como uint256, para limitar o intervalo de valores de um inteiro e escrever testes para validar o comportamento do contrato sob diferentes entradas.

**10. Manipulação de oráculo:**  
Um ataque de manipulação de oráculo em blockchain ocorre quando um atacante modifica os dados fornecidos a um contrato inteligente por um oráculo. Isso pode ser feito comprometendo o próprio oráculo ou explorando vulnerabilidades no contrato inteligente que se integra ao oráculo. Esses ataques podem ocorrer devido a diversas vulnerabilidades, tais como:

1. **Falta de validação dos dados:** Se o contrato inteligente não valida os dados recebidos do oráculo, um invasor pode fornecer dados falsos ou maliciosos ao contrato.
2. **Acesso irrestrito:** Se o oráculo for acessível por qualquer um, um invasor pode modificar os dados sendo enviados ao contrato.
3. **Controle de acesso mal projetado:** Se o contrato inteligente não possuir controles de acesso adequados, um invasor pode modificar o oráculo para fornecer dados maliciosos ao contrato.

**Segurança da Blockchain**

Para prevenir ataques de manipulação de oráculo em blockchain, podem ser usadas validação de dados, controle de acesso apropriado e assinaturas digitais. Além disso, uma abordagem com múltiplos oráculos pode ser usada, onde múltiplos oráculos fornecem os mesmos dados ao contrato, o que pode reduzir o risco de manipulação. Se um oráculo for comprometido, o contrato ainda pode receber dados precisos das outras fontes:

1. **tx.origin:** Em Solidity, tx.origin é uma variável global que retorna o endereço da conta que enviou a transação. Usar essa variável para autorização pode tornar um contrato inteligente vulnerável se uma conta autorizada chamar um contrato malicioso. Uma chamada feita ao contrato vulnerável passaria na verificação de autorização porque tx.origin retorna o remetente original da transação, que nesse caso é a conta autorizada. Para corrigir esse problema, não use tx.origin para autorização; use msg.sender.
2. **Dependência da ordem de transações:** Uma vulnerabilidade de dependência da ordem de transações pode ocorrer quando a execução de um contrato inteligente depende da ordem na qual as transações são processadas na blockchain. Isso pode criar uma condição de corrida onde o resultado de um contrato pode ser alterado com base na ordem de execução, permitindo que invasores manipulem os resultados pretendidos. Por exemplo, um contrato inteligente pode ter uma função que recompensa a primeira pessoa que enviar uma solução para um problema. Se duas pessoas enviarem soluções, uma com taxa de gás padrão e outra (atacante) com uma taxa maior, o minerador que confirmar o bloco escolherá a transação com a maior taxa para ser incluída primeiro, premiando assim a pessoa que pagou mais, em vez da que resolveu primeiro. Essa vulnerabilidade também pode ocorrer em contratos ERC-20 onde a função approve permite que um endereço gaste tokens em nome de outro. Um atacante ciente de uma mudança na aprovação pode submeter uma solicitação de transferência com taxa de gás maior, permitindo receber mais tokens do que o pretendido.
3. **Aleatoriedade ruim:** Aleatoriedade é útil em muitas aplicações, por exemplo, aplicativos de jogos ou loterias, onde um vencedor aleatório é escolhido com base no número gerado. No entanto, gerar boa aleatoriedade não é fácil. Fontes usuais como block.timestamp são inseguras, pois o minerador pode fornecer qualquer timestamp dentro de alguns segundos e ainda ter seu bloco aceito pelos outros. blockhash e block.difficulty também são inseguros, já que os mineradores têm controle sobre eles. Para corrigir isso, recomenda-se usar uma fonte externa confiável de aleatoriedade por meio de múltiplos oráculos ou utilizar um RANDAO.
4. **Uso de compilador desatualizado:** Utilizar uma versão desatualizada do compilador Solidity pode gerar problemas, especialmente se houver bugs divulgados publicamente que afetem negativamente a versão atual. É recomendável utilizar uma versão recente do compilador Solidity para evitar tais problemas.
5. **Valor de retorno de chamada não verificado:** O valor de retorno da chamada de mensagem não é verificado. A execução continuará mesmo que o contrato inteligente chamado gere um erro. Se a chamada falhar acidentalmente ou um hacker induzir uma falha na chamada, isso pode resultar em comportamento inesperado no programa. Recomenda-se verificar o valor de retorno da chamada para lidar com falhas. Também é recomendável, às vezes, evitar chamadas externas, se possível. Pode ser possível que seu contrato seja manipulado por um contrato malicioso de terceiros/externo. Isso ocorre porque o controle da execução é transferido para um contrato externo durante uma chamada externa. Ataques como reentrância e condições de corrida podem ocorrer por causa disso.

**Ataques DeFi**

1. **Ataques de empréstimo relâmpago (*flash loan attacks*):**  
   Em um ataque de empréstimo relâmpago, um invasor pode pegar emprestado grandes quantias de fundos sem fornecer colateral. Dessa forma, o invasor consegue movimentar o mercado em seu favor; por exemplo, manipula o preço de um ativo de criptomoeda em uma exchange, obtém fundos e então imediatamente os vende em outra.
2. **Ataques sanduíche (*sandwich attacks*):**  
   Um “ataque sanduíche” é um tipo de esquema de front-running em criptomoedas que envolve o invasor escaneando a blockchain (em mempools) em busca de transações de alto valor e então posicionando uma transação antes da da vítima, com uma taxa de gás mais alta, para garantir que seja processada primeiro. Isso faz com que o preço da moeda aumente, fazendo com que a vítima pague mais. O invasor então vende suas próprias moedas, lucrando com a diferença de preço. Essa tática é chamada de “ataque sanduíche” porque intercala a transação da vítima entre duas transações submetidas pelo atacante. Entre maio de 2020 e abril de 2022, estima-se que mais de 60.000 Ether, no valor de mais de 72 milhões de dólares, foram perdidos devido a esse tipo de ataque.
3. **MEV/BEV:**  
   MEV refere-se ao valor máximo que pode ser obtido além da recompensa normal do bloco e taxas de gás por meio da manipulação de transações dentro de um bloco, como incluir, omitir ou alterar a ordem das transações.
4. **Riscos de estabilidade/segurança de stablecoins:**  
   Podem haver vários riscos para stablecoins, incluindo risco de liquidez, risco de solvência e risco regulatório. As stablecoins geralmente são lastreadas por ativos de reserva, mas se a demanda pela stablecoin aumentar, pode não haver ativos de reserva suficientes para atender a essa demanda, levando à queda de seu valor.
5. **Falsificação de identidade:**  
   No ecossistema DeFi, identidade e controle de acesso são da mais alta importância. Deve-se garantir que apenas entidades autorizadas possam operar na rede. Isso pode ser conseguido por meio de credenciais verificáveis e DIDs. Discutiremos isso com mais detalhes no Capítulo 20, Identidade Descentralizada. O problema-chave aqui é como verificar que uma identidade é verdadeira.
6. **NFTs falsificados:**  
   É quando um golpista cria arte ou mesmo gera arte online e afirma que é de um artista famoso. Além disso, em alguns casos, o golpista pode pegar arte original de um artista, copiá-la, criar um NFT com ela e vendê-la em seu próprio nome. Em ambas as situações, o artista original não recebe o pagamento justo.
7. **DoS em NFTs:**  
   Isso pode ocorrer devido ao armazenamento off-chain da arte real do NFT, onde um invasor assume o controle ou derruba o servidor onde a arte está hospedada.
8. **Geração ilimitada de tokens (sem escassez):**  
   Existe a possibilidade de que tokens sejam gerados sem qualquer escassez econômica, o que resulta na desvalorização do ativo.
9. **Caça a airdrops:**  
   Normalmente, os airdrops têm fornecimento limitado e uma quantidade fixa é destinada a um único usuário; no entanto, pode ser possível para um golpista criar múltiplas identidades (geralmente apenas múltiplos e-mails) e reivindicar mais airdrops do que o pretendido pelo provedor do airdrop.

A seguir, discutiremos a **camada de interface**.

**Camada de interface**

Diversos ataques existem na camada de interface, conforme descrito a seguir.

**Ataques de oráculo / ataques de manipulação de oráculo**

A utilização de oráculos em blockchain representa um risco potencial de segurança devido à dependência das informações externas que eles fornecem. Esses oráculos, tipicamente implementados como contratos inteligentes, têm a capacidade de fornecer dados incorretos, o que pode resultar em consequências prejudiciais para os processos ligados ao feed de dados. Tal manipulação pode ter efeitos de longo alcance, incluindo liquidações de ativos indevidas e práticas de negociação maliciosas. Nesse contexto, é crucial examinar e compreender as vulnerabilidades e falhas comuns associadas aos oráculos em blockchain. Alguns ataques de oráculo incluem:

* **Manipulação de fontes de dados:** Invasores podem manipular as fontes de dados das quais os oráculos dependem para inserir informações falsas na blockchain, potencialmente fazendo com que o contrato inteligente tome decisões incorretas.
* **Ataques Sybil:** Invasores podem criar múltiplas identidades falsas para controlar uma grande porção dos oráculos, permitindo-lhes manipular os dados que são inseridos na blockchain.
* **Suborno de oráculos:** Invasores podem subornar oráculos para inserir informações falsas na blockchain, potencialmente fazendo com que o contrato inteligente tome decisões incorretas.
* **Censura de oráculos:** Invasores podem impedir que oráculos acessem determinadas fontes de dados, levando à ausência de informações para o contrato inteligente tomar decisões.
* **DoS:** Invasores podem sobrecarregar os oráculos com muitas requisições, tornando-os indisponíveis para o contrato inteligente e, potencialmente, fazendo-o falhar.
* **Ataques de aproveitamento gratuito (*freeloading*):** Nesse tipo de ataque, um nó pode utilizar outro oráculo ou componente off-chain, como uma API, e simplesmente replicar os valores sem qualquer validação. Por exemplo, um oráculo responsável por fornecer dados climáticos pode esperar que os provedores de dados meçam a temperatura em um local designado. No entanto, os nós podem ser incentivados a usar uma API de dados climáticos publicamente acessível e simplesmente apresentar seus dados ao sistema. Isso pode levar a problemas de centralização com a fonte de dados e, se realizado em larga escala, pode afetar significativamente a precisão dos dados. Isso é particularmente perceptível quando as taxas de amostragem variam, por exemplo, quando o oráculo on-chain espera uma amostragem a cada 10 minutos, enquanto os nós que estão se aproveitando fornecem dados de uma API que é atualizada apenas uma vez por hora. A prevalência de aproveitamento gratuito em mercados de dados oraculares descentralizados pode agravar uma espiral descendente de preços, já que o aproveitamento requer apenas uma simples recuperação de dados.

**Ataques a carteiras**

Carteiras de criptomoeda podem ser alvo de diversos ataques, listados a seguir:

* **Ataques de phishing:** Onde um invasor cria um site ou aplicativo de carteira falso que se parece com um real, enganando os usuários para que revelem sua frase-semente ou chaves privadas.
* **Ataques de malware:** Onde malwares infectam o computador ou dispositivo móvel usado para acessar a carteira de criptomoeda e roubam frases-semente ou chaves privadas.
* **Ataques Man-in-the-Middle (MITM):** Onde um invasor intercepta e manipula transações se passando por uma entidade confiável durante a comunicação entre a carteira e a rede Ethereum.

Carteiras físicas (hardware wallets) podem estar sujeitas a alguns ataques, conforme descrito abaixo:

* **Ataques na cadeia de suprimentos (*Supply chain attacks*):** Onde um invasor modifica a carteira física durante a produção ou distribuição para obter acesso às chaves privadas ou frases-sementes.
* **Violação física:** Onde um invasor abre fisicamente a carteira física e acessa as chaves privadas ou frases-sementes armazenadas nela.
* **Ataques de malware:** Onde malwares infectam o computador usado para conectar à carteira física e roubam chaves privadas ou frases-sementes.
* **Ataques MITM:** Onde um invasor intercepta e manipula transações se passando por uma entidade confiável durante a comunicação entre a carteira física e o computador.
* **Ataques no nível do usuário:** Se os usuários não seguirem corretamente as instruções ou não adotarem boas práticas para proteger suas chaves, isso pode simplesmente levar à perda de fundos. Há muitos exemplos no mundo real que vão desde discos rígidos perdidos contendo carteiras de Bitcoin até esquecer senhas de carteiras e até mesmo esquecer os códigos PIN de carteiras físicas.

Ataques ao frontend podem incluir diversos tipos:

* **Scripts maliciosos:** Ataques usuais de scripts em uma página da web para roubar chaves ou detalhes da conta ou criar transações desonestas.
* **Frontend desalinhado:** Funcionalidade/UI da interface web que não corresponde ao contrato backend.
* **Invasão de conta:** Detalhes da conta da interface web sendo obtidos por phishing/perda.
* **Ataques DoS:** Ponto único de falha (SPOF) da interface web.

Na camada de interface, é possível que ataques de *cryptojacking* sejam realizados contra navegadores e o sistema operacional.

Até agora, discutimos ataques em blockchains de camada 1. No entanto, com o advento das blockchains de camada 2, novas classes de ataques surgiram, que discutiremos a seguir.

*Cryptojacking* é o uso não autorizado do computador de outra pessoa para minerar criptomoeda.  
Geralmente é feito sem o conhecimento ou consentimento do dono do computador e pode deixar a máquina mais lenta, pois utiliza seus recursos para minerar a criptomoeda. O processo de mineração envolve resolver quebra-cabeças matemáticos complexos para validar transações na rede da criptomoeda, e requer muita capacidade computacional. Invasores podem usar malwares para infectar um computador e então usar seus recursos para minerar criptomoeda, ou podem usar scripts baseados em navegador que rodam em um site e mineram criptomoeda utilizando os recursos do computador do visitante.  
Existem várias formas de remediar o *cryptojacking*. Primeiro, garantir que um antivírus atualizado esteja em execução no host. Extensões do Chrome como **No Coin** ou **MinerBlock** podem defender contra ataques de *cryptojacking*. Outra técnica é redirecionar os URIs de mineração para um endereço nulo atualizando o arquivo de hosts ou configurações de DNS. Também utilize extensões bloqueadoras de anúncios (*adblockers*), que podem impedir a execução de scripts de *cryptojacking*.

**Segurança da Blockchain**

**Ataques a blockchains de camada 2**

Alguns dos ataques são semelhantes aos ataques da camada 1; no entanto, alguns são novos e específicos da camada 2:

* **Censura de transações**: Em sistemas de camada 2, se um provedor de rollup for centralizado, isso pode resultar em censura de transações.
* **Ataques a um provedor de rollup**: Podem ocorrer vários ataques contra um provedor de rollup, incluindo DoS e quaisquer outros ataques usuais que possam ser realizados contra um nó, incluindo vírus/malware e ataques de rede.
* **Ataques de disponibilidade de dados**: Existem duas opções disponíveis para disponibilidade de dados. Uma é armazenar os dados em uma cadeia para garantir sua disponibilidade e a outra é armazená-los fora da cadeia com entidades terceiras, que os disponibilizarão quando necessário. Normalmente, incentivos econômicos são aplicados para recompensar entidades fora da cadeia por armazenar dados. No entanto, mecanismos de incentivo mal projetados ou provedores de serviço centralizados podem censurar os dados, o que pode levar a consequências indesejadas. Além disso, no caso de armazenar dados em uma cadeia da camada 1, a eficiência e os benefícios dos rollups seriam limitados.
* **Vulnerabilidades relacionadas a pontes blockchain**: Este é um problema muito comum (em meados de 2023). Vários ataques de alto perfil como o ataque à Ronin Bridge, ataque Wormhole, ataque à BNB Smart Chain e Nomad Bridge resultaram na perda de mais de um bilhão de dólares. As pontes entre cadeias são recursos muito desejáveis no ecossistema blockchain, pois permitem comunicação entre cadeias, o que possibilita interoperabilidade, resulta em melhor liquidez e fornece serviços como transferências de tokens entre cadeias. No entanto, existem várias questões que podem resultar em vulnerabilidades. Por exemplo:
  + Uma ponte mal projetada, se explorada, pode levar à perda de fundos.
  + Além disso, o design das pontes geralmente é bastante complexo, o que também pode levar a vulnerabilidades que podem ser exploradas por hackers.
  + Existem várias explorações e vulnerabilidades, como depósitos falsos, falhas no design dos validadores e tomada de controle dos validadores por agentes maliciosos. Normalmente, essas pontes funcionam com base em eventos gerados por uma cadeia e lidos por outra cadeia para gerar tokens ou iniciar uma transferência na outra cadeia. Se um ataque conseguir produzir um evento válido sem realmente depositar fundos em uma cadeia (um depósito falso), então isso pode resultar na geração de novas moedas na outra cadeia, sem realmente gastar nenhuma moeda na primeira cadeia.
  + Se os validadores se comportarem mal, coludirem ou forem tomados por uma entidade maliciosa (adversária), isso também pode levar a consequências indesejadas.

A disponibilidade de dados é necessária para tornar os dados comprometidos disponíveis a qualquer parte fora do provedor de rollup (camada 2) para garantir a vivacidade caso o provedor de rollup falhe ou desapareça por qualquer motivo (mesmo que malicioso). A disponibilidade de dados também é necessária se algum usuário quiser sacar seus fundos e precisar ignorar completamente a camada 2, seja devido à indisponibilidade da camada 2, ou a agentes maliciosos na camada 2, ou simplesmente “querer seu dinheiro de volta”.

*Tradução continuará na próxima resposta devido à extensão do conteúdo.* Deseja que eu prossiga imediatamente com a próxima parte?

Perguntar ao ChatGPT

* **Ataques relacionados a canais de estado e sidechains**: Os ataques contra canais de estado incluem obstrução de canal (channel jamming) e ataques de repetição (replay). A obstrução de canal é particularmente problemática em uma rede Lightning. Ela ocorre quando um ataque DoS impede que nós encaminhem transações enquanto sua liquidez está bloqueada, e eles são incapazes de ganhar taxas porque não podem encaminhar pagamentos. Um invasor pode rotear o pagamento por meio de outros nós sob seu controle e se recusar a finalizar o pagamento. Essa situação resultará na "obstrução do canal" até que o bloqueio de tempo do HTLC expire, e o pagamento seja reembolsado.

Os ataques a sidechains podem incluir:

* + Produtores de blocos centralizados, o que pode levar à censura de transações.
  + Falta de segurança herdada da camada 1 (segurança independente da camada 2) pode levar a situações onde uma sidechain tenta executar seu próprio conjunto de validadores e mecanismo de segurança, que pode estar em um nível muito inferior ao da camada 1. Isso pode resultar em problemas de segurança e vulnerabilidades de exploração nas sidechains.
  + Falta de atomicidade entre duas cadeias.
  + Ponte vulnerável.
* **Bugs em DSLs**: À medida que mais e mais linguagens específicas de domínio (DSLs), como ZoKrates, Circom e Cairo estão sendo escritas para facilitar a criação de provas de conhecimento zero a serem usadas em um sistema de camada 2, se elas não forem projetadas e/ou implementadas corretamente, isso pode levar a bugs no nível da linguagem, que podem tornar o sistema vulnerável a certos ataques.

**Camada de Criptografia**  
Várias vulnerabilidades e ataques podem existir na criptografia usada em cada camada:

**Ataques à criptografia de chave pública**

A criptografia de chave pública é um aspecto crucial da segurança em blockchain e, geralmente, da segurança da informação, mas não é 100% imune a ataques. Alguns ataques comuns contra a criptografia de chave pública estão listados abaixo:

* **Ataque de força bruta**: Envolve tentar todas as combinações possíveis de caracteres para encontrar a chave privada. Esse ataque pode ser evitado usando chaves mais longas, mas torna-se inviável à medida que o comprimento da chave aumenta.
* **Ataque Man-in-the-middle (MITM)**: O invasor intercepta a comunicação entre duas partes e substitui a chave pública de uma delas por sua própria. O invasor pode então descriptografar e ler todas as mensagens trocadas. Um exemplo comum é o SSL stripping, onde o invasor intercepta a comunicação entre cliente e servidor e rebaixa a conexão para uma versão insegura, como HTTP em vez de HTTPS.
* **Ataque de personificação**: Um invasor cria um certificado falso que aparenta ser de uma fonte confiável, para enganar o usuário e fazê-lo aceitar a chave pública do invasor. Ataques de phishing podem ser usados para se passar por sites confiáveis e roubar informações sensíveis.
* **Ataque de reutilização de chave**: Ocorre quando a mesma chave é usada para mais de um propósito, facilitando que um invasor obtenha a chave privada. Por exemplo, o protocolo de criptografia WEP usado em redes Wi-Fi antigas era vulnerável a esse ataque, pois a mesma chave era usada para criptografar todos os pacotes. Em redes blockchain, é aconselhável usar uma chave apenas para o propósito para o qual foi gerada.
* **Uso prolongado da mesma chave**: Como boa prática, se uma chave privada já foi utilizada, por exemplo, para assinar uma transação blockchain, é aconselhável usar uma nova chave para uma nova transação, em vez de reutilizar a mesma repetidamente.
* **Ataque de canal lateral (side-channel attack)**: Explora informações vazadas pela implementação de um algoritmo criptográfico, como consumo de energia, radiação eletromagnética ou informações de temporização, para obter a chave secreta. Por exemplo, a Análise Diferencial de Energia (DPA) é um tipo de ataque de canal lateral que analisa o consumo de energia de um dispositivo para extrair a chave secreta usada na criptografia.

Tais ataques podem ser mitigados usando algoritmos de criptografia fortes, práticas adequadas de gerenciamento de chaves e verificando a autenticidade das chaves públicas usadas na comunicação.

**Ataques a funções de hash**

Alguns ataques comuns que podem ser realizados contra funções de hash são os seguintes:

* **Ataque de colisão**: Tentativa de encontrar duas entradas que produzam a mesma saída de hash. Um ataque bem-sucedido compromete a integridade de assinaturas digitais e outras aplicações que dependem de funções de hash.
* **Ataque de pré-imagem**: Tentativa de encontrar uma entrada que gere um valor de saída específico. Um ataque bem-sucedido pode comprometer a segurança de sistemas de armazenamento de senhas e outras aplicações que usam funções de hash para armazenar segredos.
* **Ataque do aniversário (birthday attack)**: Um tipo de ataque de colisão que explora o paradoxo do aniversário para encontrar colisões de forma mais eficiente do que uma busca por força bruta. É especialmente relevante para funções de hash com saídas menores, como MD5 e SHA-1.
* **Ataque de extensão de comprimento**: Permite que um invasor adicione dados a uma mensagem depois que ela foi hasheada, sem conhecer a mensagem original ou o valor de hash. Esse tipo de ataque é possível ao usar funções de hash com certas propriedades, como a construção de Merkle-Damgård.

Em sistemas blockchain, geralmente são usadas as funções SHA-256, SHA-3 e Keccak, que são consideradas seguras contra essas ameaças. No entanto, com um computador quântico, pode ser possível encontrar algumas colisões.

**Vulnerabilidades e ataques relacionados ao gerenciamento de chaves**

Essas vulnerabilidades podem comprometer a confidencialidade, integridade e disponibilidade de dados criptografados, resultando em grandes danos financeiros e de reputação. Alguns exemplos:

* **Armazenamento inseguro de chaves**: Pode expor chaves privadas, permitindo que um invasor assine transações ilegítimas. Para mitigar isso, as chaves privadas devem ser armazenadas em locais seguros, como um Módulo de Segurança de Hardware (HSM) ou arquivos criptografados, com acesso restrito a pessoal autorizado. Também é recomendável o uso de carteiras de hardware como Trezor ou SafePal.
* **Compartilhamento não autorizado de chaves**: Pode expor chaves privadas. Para mitigar, o acesso às chaves deve ser restrito e políticas de prevenção ao compartilhamento não autorizado devem ser implementadas.
* **Perda ou roubo de chaves**: Pode levar à perda permanente do acesso aos dados criptografados. Para mitigar, as chaves devem ser armazenadas em locais múltiplos e backups devem ser mantidos.
* **Ataque de custódia de chave (key escrow)**: Ocorre quando um terceiro detém as chaves privadas dos dados criptografados. Se comprometido, o atacante pode acessar os dados. Para mitigar, as chaves devem ser mantidas em locais seguros, com acesso restrito.

**Ataques relacionados a ZKP (provas de conhecimento zero)**

Vários ataques podem ter como alvo as provas de conhecimento zero:

* **Segurança de bits inadequada**: O uso de comprimento de bits inadequado compromete a segurança. Por exemplo, usar 512 bits no lugar de 4096 bits em RSA torna-o vulnerável. No contexto da camada 2, a StarkWare afirma que o provador SHARP opera com segurança de 80 bits. No entanto, o esquema FRI opera apenas com 48 bits de segurança, sob suposições bastante fracas. A segurança formalmente comprovada chega no máximo a 22 bits. Um quebra-cabeça PoW de 32 bits é usado para mitigar essa limitação, mas ainda pode ser explorado. Isso significa que, na prática, a probabilidade de forjar uma prova pode ser maior do que o esperado.
* **Ataques à privacidade**: Incluem ataques de desanonimização, quebra de confidencialidade e divulgação de identidade.
* **Vulnerabilidades em assinaturas digitais**: A maleabilidade de assinaturas digitais refere-se à possibilidade de modificá-las sem invalidá-las, mas ainda assim alterar a mensagem subjacente. Isso pode permitir manipulações não detectadas e violar protocolos criptográficos.
* **Ameaças quânticas**: Computadores quânticos suficientemente poderosos podem quebrar esquemas como ECC e RSA. Isso comprometeria as chaves privadas e permitiria o roubo de criptomoedas. Para mitigar essas ameaças, estão sendo desenvolvidas soluções como:
  + **Criptografia pós-quântica**
  + **Esquemas de assinatura à prova de quânticos** (por exemplo, assinaturas baseadas em hash)
  + **Distribuição quântica de chaves (QKD)**
  + **Abordagens híbridas**, que combinam algoritmos clássicos com resistentes a quânticos

**Maleabilidade de provas**

Acertar a criptografia é algo bastante complicado, especialmente verdadeiro no caso dos zk-SNARKs, que ainda são uma área relativamente nova. Problemas com a matemática subjacente e sua implementação podem levar a consequências inesperadas. Uma das questões é a **maleabilidade de provas**, onde um atacante pode alterar uma prova sem invalidá-la. Em outras palavras, o atacante pode manipular a prova de maneira que ainda pareça válida e autêntica, mas com seu conteúdo alterado. Isso pode levar a vários problemas de segurança, como **ataques de repetição**, onde a mesma prova pode ser usada várias vezes para mensagens diferentes.

**Vulnerabilidades na configuração (setup)**

Se a configuração confiável (trusted setup) não for conduzida corretamente ou for comprometida e valores secretos vazarem, isso pode permitir que um provador crie declarações falsas. É importante que os valores secretos na configuração confiável — os chamados “**resíduos tóxicos**” — sejam eliminados permanentemente.

**Ferramentas e mecanismos de análise de segurança**

Existem várias técnicas para verificar a correção de programas. Primeiro, temos os **testes unitários**, que são um método comum. Em segundo lugar, temos os **testes baseados em propriedades**, isto é, *fuzzing*. No próximo nível, temos a **verificação por modelo** (*model checking*), e, finalmente, **provas formais** são a técnica mais avançada para garantir a correção de programas — neste caso, contratos inteligentes.

Os testes incluem testes unitários, testes de integração, testes completos de ponta a ponta e testes baseados em propriedades (também chamados de *fuzzers*).

A **análise estática** permite verificar o código contra um conjunto de regras para encontrar defeitos. O código não é executado; em vez disso, ele é verificado estaticamente. Por outro lado, existe a **análise dinâmica**, onde o código é executado para encontrar falhas e testado contra critérios. A análise dinâmica geralmente compreende testes unitários e não é considerada uma técnica de verificação formal. A análise estática usando técnicas formais é usada para verificar formalmente a correção do programa.

Existem três tipos de provas. Estamos familiarizados com as **provas feitas à mão**, verificadas manualmente — suscetíveis a erros. Em segundo lugar, temos os **assistentes de prova**, que auxiliam na escrita de provas. As provas ainda são escritas por humanos, mas podem ser verificadas automaticamente quanto à correção. A ferramenta também ajuda na construção da prova correta. Por fim, temos as **provas automatizadas**, mas estas sofrem com o problema da indecidibilidade e com o espaço de busca exponencial. Vale ressaltar que construir provas formais exige muito esforço, então só faz sentido utilizar essa abordagem para códigos e algoritmos complexos, onde a correção é difícil de estabelecer. Normalmente, sistemas distribuídos, sistemas concorrentes e sistemas críticos para segurança requerem verificação formal. Para códigos mais simples, não vale o esforço.

**Verificação formal de contratos inteligentes**

Ferramentas comuns incluem **VERX** e **KEVM**. A verificação formal permite detectar erros complexos que são difíceis de identificar manualmente ou com testes unitários simples. Ela prova que o contrato inteligente atende a uma especificação formal de funcionalidade. A análise manual, embora trabalhosa, também tem seu lugar para garantir a correção do código. Auditores podem revisar o código manualmente para encontrar falhas, mas esse método é demorado e menos eficaz.

Embora os testes com ferramentas automatizadas, como testes unitários, geralmente sejam aceitáveis, a **cobertura de código** pode não ser total, levando a falhas. Análises manuais e a maioria dos métodos sofrem com **falsos negativos** (não detectar vulnerabilidades que realmente existem) e **falsos positivos** (reportar vulnerabilidades inexistentes). A **execução simbólica** é uma técnica para verificar a correção de programas. Ela usa entradas simbólicas para representar estados e transições em linguagem matemática precisa.

A verificação formal ganhou destaque no espaço blockchain devido à sua capacidade de garantir correção de código, algoritmos e protocolos. Embora não seja uma técnica nova, os rigorosos requisitos de segurança dos sistemas blockchain tornam sua aplicação especialmente relevante.

**Verificação formal**

Antes de explorar diferentes técnicas de verificação formal disponíveis no universo blockchain, vamos entender o que é verificação formal, seus tipos e por que ela é desejável.

**Métodos formais** são um conjunto de técnicas usadas para modelar sistemas como objetos matemáticos. Incluem escrita de especificações em lógica formal, verificação por modelo e provas formais. Geralmente, dividem-se em duas áreas principais: **especificações formais** e **verificação formal**. A primeira lida com escrever especificações precisas e concretas; a segunda, com desenvolver provas para confirmar a correção da especificação.

Em essência, a verificação formal consiste em três etapas:

1. Criar um modelo formal do sistema a ser verificado.
2. Escrever uma especificação formal das propriedades esperadas.
3. Verificar se o modelo satisfaz a especificação.

Para a verificação, existem duas grandes categorias de técnicas: **abordagens baseadas em exploração de estados** e **abordagens baseadas em provas**. Cada uma possui vantagens e desvantagens:

* **Exploração de estados**: Onde todos os estados possíveis são verificados. É automática, mas ineficiente e difícil de escalar. Um problema comum é a **explosão de estados**, em que o número de estados cresce tanto que o modelo não cabe na memória do computador.
* **Abordagens baseadas em provas (teoremas)**: São mais precisas, mas mais difíceis de implementar, pois exigem a escrita manual de provas. Mesmo com assistência de ferramentas, elas requerem mais conhecimento e esforço.

**Verificação formal de contratos inteligentes**

Muita pesquisa tem sido conduzida sobre a verificação formal de contratos inteligentes. As técnicas propostas incluem, mas não se limitam a:

* Verificação por modelo (model checking)
* Análise estática
* Análise dinâmica
* Verificação usando assistentes de prova

A segurança de contratos inteligentes é uma área extensa e ativa de pesquisa. Embora não seja possível cobrir todos os aspectos neste capítulo, o leitor é incentivado a revisar a introdução à verificação formal e seguir com os trabalhos acadêmicos citados para mais aprofundamento.

Apesar de todas as técnicas e ferramentas para mitigar riscos em contratos inteligentes, uma pergunta persiste: **o que mais podemos fazer, considerando que sua segurança é um tema delicado que pode resultar em perdas financeiras graves?**

A resposta pode ser **tratar contratos inteligentes e blockchains como sistemas críticos de segurança** — e aplicar todos os atributos dessa categoria. Embora possa soar exagerado, imagine um contrato inteligente que acione o desligamento de um reator nuclear ao detectar superaquecimento. Todo cuidado seria justificado. Com a convergência entre blockchain e IoT, esses cenários podem se tornar realidade em breve — inclusive na saúde, aviação e defesa.

A verificação baseada em provas é realizada com assistentes como **Coq** (<https://coq.inria.fr/>) e **Isabelle** (<https://isabelle.in.tum.de/>).

**Em Solidity**

A verificação formal é realizada usando **Satisfiability Modulo Theories (SMT)** e **Horn solving**, por meio do **SMTChecker**. Este módulo analisa se o código satisfaz as condições estabelecidas pelas instruções require e assert. Se encontrar erro, fornece um contraexemplo; caso contrário, presume-se que o código é seguro.

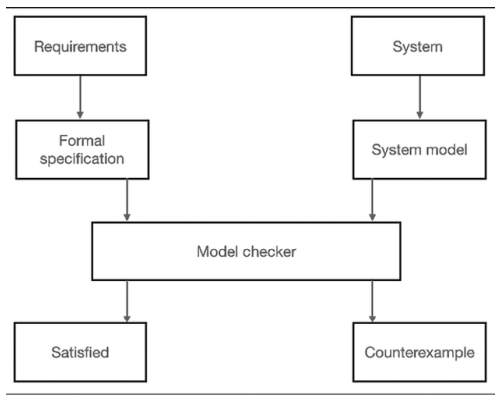
O **SMT (Satisfiability Modulo Theories)** é uma técnica bem estabelecida que avalia a satisfatibilidade de fórmulas lógicas dentro de teorias predefinidas. Muito usada em verificação formal e provas automáticas, ela permite detectar propriedades desejadas e potenciais erros antes da ocorrência.

**Horn solving** usa cláusulas lógicas chamadas cláusulas de Horn (formadas por literais com no máximo um literal positivo) para representar as propriedades desejadas de um sistema. O resolvedor de Horn verifica se há uma solução que satisfaça todas essas cláusulas. Se encontrar, é uma prova de que o programa está correto. Caso contrário, fornece um contraexemplo — uma execução que viola a propriedade.

**Model checking (Verificação por modelo)**

Model checking é uma técnica usada para verificar automaticamente sistemas de estados finitos. Executa uma **busca exaustiva** no espaço de estados para determinar se uma especificação é verdadeira ou falsa — sem necessidade de provas manuais demoradas.

O projetista escreve formalmente a especificação e suas propriedades, e o verificador de modelos explora todo o espaço de estados para validar os requisitos. Se encontrar uma execução que viole a especificação, retorna uma **trilha de erro** que pode ser analisada.



*Figure 19.2: Model checking*

A verificação por modelo usa **lógicas temporais** para descrever o comportamento dos sistemas ao longo do tempo — isto é, a ordenação de eventos.

Lógicas usadas:

* **LTL (Lógica Temporal Linear)**: descreve propriedades sobre uma única execução.
* **CTL (Lógica Computacional em Árvore)**: expressa propriedades sobre todas as execuções possíveis (ramificações).

Operadores temporais fundamentais usados em LTL:

* ⃞ P (sempre)
* ◇ P (eventualmente)
* U (até)
* ◯ (próximo estado)

Exemplos:

* ⃞((mensagem não assinada ∨ não selada) ⇒ ◯ não válida): sempre que a mensagem não for assinada ou selada, ela não será válida no próximo estado.
* ⃞(broadcast ⇒ ◇ recebida): sempre que algo for transmitido, eventualmente será recebido.
* ¬IniciarAquecimento U Porta(fechada): um micro-ondas só inicia o aquecimento quando a porta estiver fechada.

Métodos formais são amplamente utilizados em aviação, eletrônica, sistemas embarcados e agora, cada vez mais, em blockchain — especialmente após falhas notórias como o ataque à DAO.

**Verificação de mecanismos de consenso**

De outro ponto de vista, um **mecanismo de consenso** como o **PBFT** (Practical Byzantine Fault Tolerance) existe em uma blockchain para garantir que todos os nós da rede alcancem um **acordo sobre os valores propostos**, mesmo na presença de nós defeituosos. Mecanismos como **PBFT** e **Raft** foram descritos no Capítulo 5: *Algoritmos de Consenso*.

Essa é uma área de importância crucial e é considerada o mecanismo central mais vital de uma blockchain. A verificação por modelo pode desempenhar um papel essencial na **descrição formal e verificação de algoritmos de consenso**, para garantir que os protocolos atendam às propriedades de segurança exigidas.

A abordagem geral consiste em **especificar os requisitos, propriedades desejadas e a especificação em linguagem formal**.

Há várias ferramentas para modelar e verificar programas, mas algumas vêm ganhando destaque:

* **TLA+** com o verificador de modelo **TLC**
* **SPIN**, que usa **PROMELA** para escrever especificações

Não existe uma única ferramenta correta: a escolha depende da experiência do projetista, do tipo e profundidade da verificação necessária, e da usabilidade das ferramentas.

Um algoritmo de consenso distribuído é avaliado segundo duas grandes categorias de propriedades:

* **Segurança (safety)**: nada de errado acontece.
  + Acordo
  + Integridade
  + Validade
* **Vivacidade (liveness)**: algo bom eventualmente acontece.
  + Ex: término (termination)

Um programa é considerado correto se, em **todas as execuções possíveis**, ele se comportar conforme a especificação — que é formalmente definida e verificada com um verificador de modelos ou provadores de teoremas.

**Segurança de contratos inteligentes**

Nos últimos tempos, muito trabalho tem sido feito em relação à segurança de contratos inteligentes, especialmente em **verificação formal**. Isso foi impulsionado, sobretudo, pelo **infame ataque à DAO** e outros incidentes.

A verificação formal é o processo de garantir que um programa de computador satisfaça certas propriedades formais. Não é um conceito novo. Existem ferramentas como **Frama-C** (<https://frama-c.com>) para verificar programas em C.

A ideia-chave é converter o programa-fonte em **declarações matemáticas** compreensíveis por verificadores automáticos. Para isso, usa-se, por exemplo, o **Why3**, uma plataforma de verificação de programas.

A segurança de contratos inteligentes tornou-se **fundamental**, e várias iniciativas foram tomadas para **analisar programas em Solidity** e encontrar falhas. Exemplos de ferramentas:

* **Oyente**
* **Manticore**
* **Slither**

Essas ferramentas foram criadas por pesquisadores para analisar contratos inteligentes.

Além disso, o **Remix IDE** (IDE online do Solidity) passou a incluir **análise estática** do código como recurso nativo.

Inicialmente, havia um verificador Why3 experimental no Remix IDE, mas foi posteriormente removido. Atualmente, o Remix oferece opções de análise estática.  
Veja: <https://remix-ide.readthedocs.io/en/latest/static_analysis.html>

No Remix IDE, o código é analisado e as vulnerabilidades são relatadas na aba **Analysis**. A análise estática abrange diversas categorias, como:

* Segurança
* Gás e economia

**Outras ferramentas:**

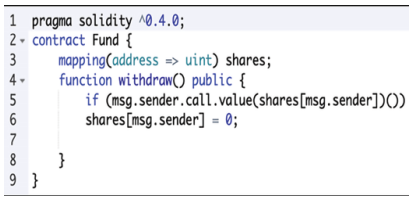
* **Slither**: estrutura de análise estática para contratos inteligentes.
* **Manticore**: estrutura de execução simbólica para binários e contratos inteligentes.  
  → Artigo: <https://arxiv.org/pdf/1907.03890>

**Execução simbólica**: técnica usada para determinar que partes do programa são executadas com base em quais entradas. Em vez de usar dados reais, usa **valores simbólicos**. Um **provador automático de teoremas** verifica se há entradas que causem falhas no programa. Usado para depuração, testes e segurança.

**Oyente**

**Oyente** está disponível como **imagem Docker**, fácil de instalar e testar: <https://github.com/melonproject/oyente>

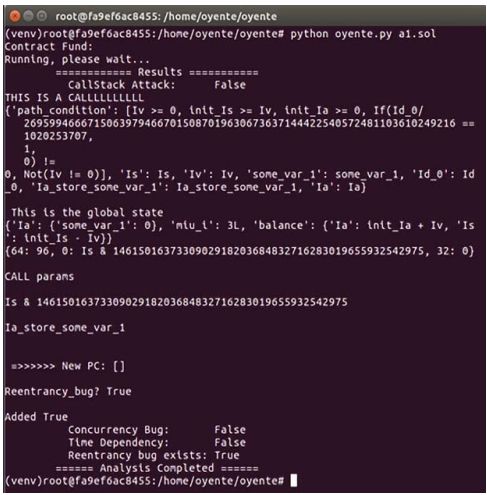
Um exemplo extraído da documentação do Solidity mostra um contrato com **bug de reentrância**.



\*\*Figura 19.3\*\*: Contrato com bug de reentrância, extraído da documentação do Solidity

Esse bug ocorre quando um contrato transfere ether ou interage com outro contrato, transferindo controle a este. O contrato chamado pode, então, **chamar novamente a função original** antes do término da execução anterior. Isso permite, por exemplo, chamadas repetidas à função withdraw() antes que a variável share seja zerada, retirando ETH várias vezes.

O Oyente consegue detectar esse bug de reentrância, conforme mostra a seguinte saída:



\*\*Figura 19.4\*\*: Ferramenta Oyente detectando bugs em Solidity

**Solgraph**

**Solgraph** é uma ferramenta que **gera gráficos de fluxo de controle** das funções de um contrato Solidity, facilitando a compreensão e análise.

→ <https://github.com/raineorshine/solgraph>

**Modelagem de ameaças**

A **modelagem de ameaças** identifica, analisa e documenta ameaças potenciais a um sistema ou aplicação. O objetivo é identificar vulnerabilidades e vetores de ataque para aplicar controles adequados de segurança.

Embora comum na segurança da informação tradicional, pode e deve ser aplicada ao blockchain.

Modelos populares de modelagem de ameaças:

* **STRIDE** (Microsoft):
  + **S** – Spoofing (falsificação)
  + **T** – Tampering (violação)
  + **R** – Repudiation (repúdio)
  + **I** – Information Disclosure (divulgação de informação)
  + **D** – Denial of Service (negação de serviço)
  + **E** – Elevation of Privilege (elevação de privilégio)
* **DREAD** (OWASP): mede risco com base na probabilidade e impacto:
  + Damage, Reproducibility, Exploitability, Affected users, Discoverability
* **CVSS**: sistema de pontuação de vulnerabilidades baseado em impacto e probabilidade.
* **Árvores de ataque** (*attack trees*): estrutura em árvore com o objetivo do ataque na raiz.
* **Matriz de ameaças**: representação visual de probabilidade e impacto para priorizar ameaças.

Aplicando **STRIDE** ao blockchain, por exemplo, na camada de consenso:

* **Spoofing**: um nó bizantino finge ser um validador legítimo.
* **Tampering**: ataques de colusão ou Sybil permitem inclusão de blocos maliciosos.
* **Repudiation**: o atacante pode negar transações anteriores ou censurá-las.
* **Information Disclosure**: dados sensíveis podem ser extraídos com influência sobre a rede.
* **DoS**: ataques direcionados ao nó líder impedem a proposição de blocos, paralisando a rede.
* **Elevation of Privilege**: um nó malicioso obtém controle elevado sobre o estado da blockchain.

Nem todas as dimensões do STRIDE são aplicáveis a todos os casos — às vezes, apenas uma ou duas se aplicam.

**Regulamentação e conformidade**

A **conformidade regulatória** é outra dimensão da segurança blockchain. Questões como:

* A plataforma está em conformidade com a **GDPR**?
* Está em conformidade com as políticas de segurança corporativa?
* Regras de **localização de dados**: alguns dados não podem sair de determinada região geográfica.

Soluções:

* **Transações privadas restritas**
* **Gerenciadores de transações off-chain localizados**
* **Protocolos criptográficos específicos**, como os exigidos pela **NIST**, por exemplo, curvas definidas na **SP 800-186**

→ <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-186.pdf>

Blockchain pode também ser usado de forma maliciosa:

* Pagamentos de **ransomware**: Petya, WannaCryptor, Locky
* **Scams de ICOs**, esquemas Ponzi
* Plataformas como Tornado Cash podem ser usadas tanto para doações quanto para **lavagem de dinheiro**

**Resumo**

Este capítulo introduziu a **segurança em blockchain**. Discutimos o modelo em camadas do ecossistema blockchain e ataques em diferentes níveis, com suas respectivas mitigações.

Foram apresentados:

* Ferramentas de análise de contratos inteligentes
* Verificação formal
* Modelagem de ameaças com **STRIDE**

A segurança em blockchain é um campo vasto. Este capítulo fornece uma base sólida para pesquisa adicional.