高级话题类 Advanced Topics

1：荷兰式拍卖和英式拍卖之间有什么区别?

What is the difference between a Dutch auction and an English auction?

答：荷兰式拍卖和英式拍卖是两种不同的拍卖方式。在荷兰式拍卖中，拍卖师会先宣布一个高价，然后逐渐降低价格，直到有人愿意出价购买物品为止。在这种拍卖中，第一个出价的人通常会赢得拍卖，因为他们愿意支付的价格最高。荷兰式拍卖通常用于出售艺术品、古董和其他昂贵的物品。英式拍卖是一种更传统的拍卖方式。在这种拍卖中，拍卖师会宣布一个起始价，然后买家可以逐渐提高价格，直到没有人再愿意出价为止。在英式拍卖中，最后一个出价的人通常会赢得拍卖，因为他们愿意支付的价格最高。英式拍卖通常用于出售房地产、汽车和其他大型物品。总的来说，荷兰式拍卖和英式拍卖都是有效的拍卖方式，但它们的实现方式略有不同。荷兰式拍卖通常更适合出售昂贵的物品，而英式拍卖则更适合出售大型物品。

Dutch auctions and English auctions are two different types of auctions. In a Dutch auction, the auctioneer announces a high price and then gradually lowers the price until someone is willing to bid for the item. In this type of auction, the first bidder usually wins the auction because they are willing to

pay the highest price. Dutch auctions are often used to sell art, antiques and other expensive items.

An English auction is a more traditional type of auction. In this type of auction, the auctioneer announces a starting price and then buyers can gradually increase the price until no one else is willing to bid. In an English auction, the last bidder usually wins the auction because they are willing to pay the highest price. English auctions are often used to sell real estate, cars, and other large items.

Overall, both Dutch and English auctions are effective ways to sell your property, but they are realized in slightly different ways. Dutch auctions are usually better suited for selling expensive items, while English auctions are better suited for selling large items.

2：为什么不应该使用 tx.origin 进行身份验证?

Why shouldn't I use tx.origin for authentication?

答：在 Solidity 中，tx.origin 是一个全局变量，它返回发送交易的账户地址。在合约代码中，最常用的是使用 msg.sender 来检查授权，但有时由于有些程序员不熟悉 tx.origin 和 msg.sender 的区别，如果使用了 tx.origin 可能导致合约的安全问题。黑客最典型的攻击场景是利用 tx.origin 的代码问题常与钓鱼攻击相结合的组合拳的方式进行攻击。因为

tx.origin 返回交易的原始发送者，因为攻击的调用链可能是原始发送者 -> 攻击合约 -> 受攻击合约。在受攻击合约中，tx.origin 是原始发送者。因此，通过调用 tx.origin 来检查授权可能会导致合约受到攻击。为了避免这种情况，建议使用 msg.sender 来检查授权。

In Solidity, tx.origin is a global variable that returns the address of the account that sent the transaction. In contract code, msg.sender is most commonly used to check authorization, but sometimes, some programmers are unfamiliar with the difference between tx.origin and msg.sender, if tx.origin is used it may lead to contract security issues. The most typical attack scenario for hackers is to utilize tx.origin's code problem often combined with phishing attacks in a combo attack. Since tx.origin returns the original sender of the transaction, the call chain for an attack may be original sender -> attacking contract -> attacked contract. In the attacked contract, tx.origin is the original sender. Therefore, checking authorization by calling tx.origin may lead to an attack on the contract. To avoid this, it is recommended to use msg.sender to check authorization.

3：什么是闪电贷?

What is a Lightning Loan?

答：闪电贷是一种无抵押借贷的 defi 产品，主要给开发者提供的，它允许在一笔交易内进行借款还款，只要支付一点闪电贷设定的手续费。很多攻击者利用闪电贷进行攻击和套利。

Lightning Loan is an unsecured lending defi product, mainly for developers, which allows borrowing repayments within a single transaction for a small fee set by Lightning Loan. Many attackers use Lightning Loans for attacks and arbitrage.

4：什么是重入？

What is Reentry?

答：重入攻击是一种安全漏洞，攻击者利用合约的 fallback 函数和多余的 gas 将本不属于自己的以太币转走的攻击手段。攻击者通过在攻击合约中调用受攻击合约的函数，然后在受攻击合约中再次调用攻击合约的函数，从而实现重复调用，造成重入攻击。重入攻击的本质是递归调用，攻击者利用这种递归调用来绕过原代码中的限制条件，从而造成攻击。为了避免重入攻击，可以使用 Solidity 内置的 transfer() 函数，或者使用检查-生效-交互模式 (checks-effects-interactions) 来确保状态变量的修改要早于转账操作。

A reentry attack is a security vulnerability in which an attacker utilizes a contract's fallback function and excess gas to transfer Ether that does not belong to him. An attacker can do this by calling a function of the attacked contract in the attacking contract, and then calling the function of the attacking contract again in the attacked contract, thereby repeating the call and causing a reentry attack. The essence of reentrant attacks is recursive calls, which are used by the attacker to bypass the constraints in the original code,

thus causing an attack. To avoid reentrant attacks, use Solidity's built-in transfer() function, or use checks-effects-interactions to ensure that changes to state variables precede the transfer operation.

5：如何防止无限循环永远运行?

What prevents infinite loops from running forever?

答：方法会受到 gas 费限制，栈深度的限制，所以循环不会一直永远运行，会报 out of gas 错误。如果题目理解成：做什么可以阻止无限循环。那就是代码实现，在循坏中使用计数器：require 判断是否达到满足中断循坏次数。

The method will be limited by the gas fee limit, the stack depth limit, so the loop will not keep running forever, it will report out of gas error.

If the question is understood as: what can be done to stop the infinite loop. That's what the code implements, using a counter in the bad loop: require to determine whether the number of interruptions to the bad loop has been reached.