撰文：宋嘉吉、任鹤义

来源：吉时通信

摘要：上一篇报告从底层语言特点，对比了 Move 和 Solidity（以太坊）的优势和特点。作为 Web3 的基础性研究，本篇从闪电贷这一最具特色的应用角度出发，分析了以太坊和 Move 分别如何实现闪电贷，Move 怎样规避了闪电贷攻击？

以太坊合约之间的交互是通过互通消息实现的状态一致，且允许重入、动态调用，这一特点为实现闪电贷提供了基础。期间，合约之间的函数可以互相来回调用——调用过程中会发生控制权转移。如果 DeFi 项目平台合约有漏洞，套利者能够利用其合约的恶意代码调用相应函数进行资产盗取——利用合约之间的状态同步信息差，在一个流程未结束时双花资产（以太坊资产只是赋值），或者重复执行函数（本该不被允许的逻辑）进行盗取。

重入攻击的前提是攻击者部署的合约存在恶意代码，但最核心的因素是：

Move 提出了闪电贷的新一种运行流程——烫手山芋模式，根本上弃用可重入。「烫手山芋（hot-potato）」模式是一种没有 key、store、copy 和 drop 能力的结构，是 Move 程序在交易执行期间仅使用一次的结构。由于没有 drop、key 或 store 能力，因此烫手山芋只能通过调用「销毁」函数来完结流程——这就如其名所示意的一样，这是一块烫手山芋，在流程中任何处置都是「烫手」的，只能交给销毁函数来完结。具体流程如下：

闪电贷完成的前提的最后正确还款、流程结束，在流程结束之前恶意合约可以在以太坊系统实现重复调用、修改相应的资产账户赋值实现盗取。Move 系统闪电贷流程结束的前提除了正确归还资金之外，还需要将烫手山芋这个资源进行一次性回收销毁处理，这确保了闪电贷的原子性。

风险提示：区块链商业模式落地不及预期；监管政策的不确定性。

闪电贷作为以太坊 DeFi 生态最具特色的应用，其基础是以太坊使用的 Solidity 语言允许动态调用和重入。虽然这种动态调用是智能合约开放性、可组合性的重要体现，但其带来的控制权转移和对合约函数的重复调用带来了不小的安全隐患，行业内利用闪电贷攻击有漏洞的 DeFi 资金池事件时有发生。由于 Move 语言不允许动态调用和重入，它使用一种「烫手山芋」模式很简单地实现了闪电贷，因此完全可以避免以太坊那样的安全问题。

两种系统在生态应用的工作模式有着明显的区别，这种差别的根本在于 Move 底层语言的特点，我们在前一篇报告《Web3 底层语言：Move 弥补了 Solidity 哪些不足？》已有详述。本篇报告从闪电贷应用角度来比较一下两者实现应用的不同玩法。

闪电贷（Flash Loan）是一种原生的 DeFi 新产物，可以理解为极速贷款。用户只需要在同一笔交易（区块）中完成借贷、套利、偿还并支付一笔手续费，由于是原子交易，那么借款人无需抵押任何资产即可实现借贷，提供了一种无本金套利方案。

我们在上一篇报告《Web3 底层语言：Move 弥补了 Solidity 哪些不足？》中提到的：

「对于模块化和合约组合性方面，Solidity（如以太坊）上面的 Contract 合约通过 library（相当于静态库）进行消息的传递，从而实现 Contract 合约之间的调用、交互。而 Move 语言使用了模块 (module) 和脚本 (script) 的设计，前者类似于 Contract 合约，Move 语言的合约组合性则是模块之间的组合，通过传递资源（即前文提到的 resources）。关于组合性方面，Solidity 和 Move 的区别非常明显。」

以太坊合约之间的交互是通过互通消息实现的状态一致，且允许重入、动态调用，就是说合约之间的函数可以互相来回调用——调用过程中会发生控制权转移。这一特点为实现闪电贷提供了基础。

具体来说，在一个以太坊交易中，可以进行转账操作以及其他一些列合约操作，通过调用智能合约中的功能函数，执行多项复杂功能——也就是说，一笔基于以太坊的交易可以融合一系列复杂交易：将借款、套利、偿还等一系列交易操作融合到一起成为可能。Flash Loan 中所有操作都在一个区块时间中完成，按照现在的以太坊的出块速度，Merge 后也就是 12 秒——核心并非是 12 秒，而是这一系列的交易要能够最终盈利并偿还，如果没有做到这一点，这笔交易就不会被打包写入区块，相当于借款人借款、套利（失败）这些操作并不是有效交易，只是临时状态——即原子交易。因此，用户必须通过编程将需要执行的所有步骤形成一项智能合约交易并完成借贷、使用和偿还的三个步骤。

上述复杂的交易操作由几个合约之间的动态调用实现，且这种调用是可重入的（也就是可以反复调用）。

目前用户也可以通过如 FURUCOMBO 这一类第三方项目，对闪电贷完成更简易的

插件性编程，无需实际编写代码完成智能合约的设计，最终实现闪电贷需要的全套操作。具体的套利流程如下图所示（利用 FURUCOMBO
平台，具体兑价均为示例），目前 Kyberswap 平台上的价格情况 1 sUSD=0.9927 DAI，而 Uniswap 上 1 DAI=1.2411 sUSD，用户发现这两个平台的 DAI-sUSD 交易对价格存在较大的套利空间，即可通过 FURUCOMBO
的界面，设计套利过程。包括：1、从 AAVE 借贷平台的闪电贷功能借出 100 DAI；2、通过 Uniswap 将 100 DAI 兑换成约 122 个 sUSD 代币；3、通过 Kyberswap 平台将 sUSD 代币兑换成约 122 DAI 代币；4、偿还从 AAVE 借出的 100 DAI 代币以及手续费 0.09
DAI；5、整个利用闪电贷的套利流程在一个以太坊交易内完成，并获利约 22 DAI。

如果在这一笔以太坊交易内借贷的资金没有得到偿还，那么整笔借贷交易不会被打包进入区块中，相当于借贷并没有实际发生，所以借贷方的资金不会受任何影响——中间的借贷和套利过程只是临时状态，并未被矿工打包确认。基于闪电贷的特性和时效要求，目前其最广泛的应用是套利交易。套利者无需自身使用资产进行套利操作，只需要通过闪电贷获得所需的资金量完成套利交易，并及时偿还借贷的资金。这极大的降低了套利者的准入门槛，因为理论上任何一个人都可以成为套利者，并且拥有没有上限的套利资金进行操作。

从上述流程可以看到，以太坊合约控制权在 FURUCOMBO 闪电合约 –
套利者账户合约 -Uniswap 合约 -Kyperswap 合约 –
闪电贷合约之间切换，且可以进行动态调用相应的函数，如果 DeFi 项目平台合约有漏洞，套利者能够利用其合约的恶意代码调用相应函数进行资产盗取——利用合约之间的状态同步信息差，在一个流程未结束时双花资产（以太坊资产只是赋值），或者重复执行函数（本该不被允许的逻辑）进行盗取。

对于 Move 生态，由于资产并非简单的赋值，且禁止动态调用和可重入，从根本上杜绝了风险，其具体实现我们在后面会阐述。

从实现方式上来看，以太坊 EVM（基于 Solidity）具有动态调度，可以通过可重入实现闪电贷。如我们在上一篇报告《Web3 底层语言：Move 弥补了 Solidity 哪些不足？》中提到的：

「以太坊（Solidity）的资产是由相应的合约控制，如果把 Token A 合约比喻为保险箱，保险箱会给所有用户分配一个数值余额，来表达用户所有拥有的 Token A 资产数量，但资产本身还是放在 Token A 合约的保险箱内。而 Move
用户账户本身就是一个单独的大保险箱，由用户自己控制，所有的 Token

资产都放在这个保险箱内。且这些 Token
并不是以数字的形式存在，而是不可复制的、权限受用户控制的资源（类型）。」

因此以太坊 EVM 实现闪电贷套利的流程是：

在上面这个过程中，相应的函数是动态调用的，闪电贷合约需要检查归还金额是否正确才会结束，因此控制权的转移过程是随时可以发生的，也就是可重入的——同时需要注意的事，以太坊账户资产是以数值余额的形式存在，重入会带来双花的可能，这也是存在漏洞的地方。调用外部合约的主要危险之一是它们可以接管控制，而这些来自外部的合约程序一旦有漏洞，攻击者可以通过反复调用实现攻击，即可重入攻击。

可重入攻击造就了以太坊历史上最严重攻击之一，直接导致了以太坊分叉，即 2016 年 6 月 17 日的 The DAO
崩溃事件。黑客部署了一个合约，作为「投资者」在 The DAO 中储存一些 ETH。然后黑客通过调用 The DAO 合约中的 withdraw 函数，使得 The DAO 合约给黑客提款，由于黑客合约（ fallback
函数）存在恶意漏洞——其没有结束的逻辑，于是 The DAO
始终未能完成提款（此时 The DAO 并不知道黑客收到了提款并将其账户余额变为 0）并拿回控制权，黑客通过不断调用 withdraw 函数发送超过其初始存款 ETH 数量。

在闪电贷攻击事件中，攻击者往往通过恶意合约实现可重入攻击。如 2022 年 3 月 16 日，黑客通过闪电贷借款，利用借贷项目 Hundred Finance
的漏洞实时重入攻击，最终获利约 2363 ETH。具体流程并不复杂，由于 Hundred Finance 是先转账后记账，黑客通过闪电贷借款，利用攻击合约存入 Hundred Finance 借款池实现抵押贷款，而攻击合约部署的 onTokenTransfer 函数在记账之前实现重复调用借款函数（重入攻击），以一笔抵押资产不断从不同借款池提取借款，由于是先转账后记账，当记账时（流程结束）黑客已经实现了攻击并获利。攻击的核心是：流程未结束之前，攻击者合约可以反复调用相关函数实现资产盗取。

就好比说，攻击者合约将资产抵押给 A 银行（项目 A 借款池）进行贷款，银行 A 在未完成记账结算之前就开始放款，而银行 A
完成记账结算流程意味着闪电贷交易要成功（原子性），这时候资产被 A
入库记账。但由于银行 A
和其他银行之间的信息不同步（合约函数之间的状态不同步），攻击者在 A 银行还

未完成资产入库记账时（即被攻击合约贷款流程未结束时，这时候闪电贷工作流程还未结束，合约函数调用还可以进行），再次以该资产在 B、C 等其他银行进行抵押贷款（重入），待流程结束，攻击者已经完成闪电贷攻击并获利。

重入攻击的前提攻击者部署的合约存在恶意代码，但最核心的因素是：

「可重入」是实现闪电贷的基础，然而一旦目标合约有漏洞，攻击者便可以实施重入攻击。这在我们前一篇报告中有详述。

Move 语言禁止动态调用和可重入，这从根本上杜绝了重入攻击。但 Move 系统的资产作为资源类型，一旦借出就相当于发生了真实转移，该如何确保闪电贷顺利还款呢？

Move 提出了闪电贷的新一种运行流程——烫手山芋模式，根本上弃用可重入。「烫手山芋（hot-potato）」模式是一种没有 key、store、copy 和 drop 能力的结构，是 Move 程序在交易执行期间仅使用一次的结构。由于没有 drop、key 或 store 能力，因此烫手山芋只能通过调用「销毁」函数来完结流程——这就如其名所示意的一样，这是一块烫手山芋，在流程中任何处置都是「烫手」的，只能交给销毁函数来完结。具体流程如下：

我们在前一篇报告中已经分析过，账户资产和收据（receipt）都是一种资源类型，「不能复制、丢弃或重用，可以被安全地存储和转移」，因此收据（receipt）必须被处理（且只能使用一次），而非像以太坊那样的对账户进行数值赋予处理就行。因此烫手山芋模式可以确保被借出的资产（资源类型，一旦借出就真实发生转移了）必须被归还。收据（receipt）作为烫手山芋，就好比是定时一个引爆器（这里的定时，是闪电贷套利作为一个原子交易的「时间」，并非具体的时长），资金和引爆器一起绑定被借出，而任何一方都无法收留引爆器，它必须被还回原处得以拆除——否则交易都无法完成，因此闪电贷资金可以确保会被归还。

闪电贷完成的前提的最后正确还款、流程结束，在流程结束之前恶意合约可以在以太坊系统实现重复调用、修改相应的资产账户赋值实现盗取。Move 系统闪电贷流程结束的前提除了正确归还资金之外，还需要将烫手山芋这个资源进行一次性回收销毁处理，这确保了闪电贷的原子性。

从应用角度看，Web3 的底层代表要在保证开放性、可重构的基础上提高代码安全性。在 Web3 中，代码不仅包含信息，还直接涉及资产调用，保证用户资产安全是重中之重，否则 Web3 将是黑暗森林。以太坊的生态让大家看到了智能合约的活力，下一个时代将在此基

础上向安全性、合规性继续演进，这也是我们当下关注 Web3 底层语言进化的核心逻辑，或者这孕育着下一轮的创新浪潮。