计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2021.20200135
58(4):834-848,2021

一种基于运行时信息的以太坊智能合约防御技术

向 杰 杨哲慜 周顺帆 杨 珉
(复旦大学软件学院 上海 200433)
(jxiang17@fudan.edu.cn)

A Runtime Information Based Defense Technique for Ethereum Smart Contract

Xiang Jie, Yang Zhemin, Zhou Shunfan, and Yang Min
(*College of Software*, *Fudan University*, *Shanghai* 200433)

Abstract As one of the most successful applications of blockchain technology, Ethereum smart contract has been widely integrated into programs and become a common implementation scheme for decentralized applications. However, smart contract suffers from security attacks since born because of its unique financial characteristics, and fresh attack forms continue to dribble out. State-of-art research works have proposed many effective mechanisms to detect vulnerabilities in smart contract, but they all have limitations in practical, such as design only for known vulnerabilities, need to modify the contract code, and the cost on-chain is too high. Because of the immutability of smart contract, these defense techniques which aim at specific vulnerabilities cannot fix the original contract, and as a result, they can hardly work on the new attack forms. To this end, we present a runtime information based upgradable defense system for Ethereum smart contract, which provides real-time data for the off-chain attack detection by collecting kinds of runtime information. At the same time, we design an access control mechanism deployed on smart contract, which restricts the access to the contract based on the dynamic detection result, so that we can secure the contract without modifying the code. Ethereum does not provide a mechanism to recognize and intercept real-time attack transactions, So we make use of race condition to enhance the defense on the real-time attack. The evaluation results show that out defense technology is extremely effective to prevent attacks, which can achieve 100% success rate for the follow-up attacks and achieve 97.5% success rate for the first attack detected by the use of race condition.

Key words blockchain; Ethereum; smart contract; attack and defense; access control

摘 要 智能合约是区块链技术最成功的应用之一,已经被广泛集成到应用程序中,成为应用去中心化的常见实现方案.然而,智能合约由于其独有的金融特性,一直以来饱受安全攻击,各种新的恶意攻击类型层出不穷.现有的研究工作提出了多种有效检测合约漏洞的方法,但在实际应用中都存在着各种局限:仅针对已知的漏洞类型,需要修改合约代码来消除漏洞,链上开销过大.由于智能合约部署到链上后的不可修改性,这些针对特定漏洞类型的检测防御手段无法对原有的合约进行修复,因此很难及时地应对新型的漏洞和攻击.为此,提出了一种基于运行时信息的智能合约可升级防御技术,通过引入运行时的各种信息,为链下对攻击和漏洞的检测提供实时的数据.同时,设计了一套部署在合约上的访问控制机制,基于动态检测的结果,对合约的访问进行限制,从而在不需要修改合约代码的情况下实现动态的

收稿日期:2020-03-09;**修回日期**:2020-08-10
通信作者:杨哲慜(yangzhemin@fudan.edu.cn)

防御.由于以太坊本身的机制无法对实时攻击进行识别和拦截，为了减小这一影响，利用竞争(race condition)的机制来增强防御的效果.实验结果分析表明：该防御技术可以有效地检测并防御攻击，对于后续的攻击交易，可以实现100%的拦截成功率，对于首次检测到的实时攻击，利用竞争可以达到97.5%的成功率.

关键词　区块链；以太坊；智能合约；攻击与防御；访问控制

中图法分类号　TP309.2

区块链技术是目前最受关注的互联网技术之一，它在去中心化和分布式技术发展中起到了巨大的作用.作为第1个应用区块链技术的去中心化加密货币，比特币被看做是区块链发展中的里程碑，受到了广泛的认可.继比特币之后，为了对区块链技术进行拓展和改进，各种新的技术层出不穷，其中以智能合约为核心的以太坊(Ethereum)被认为是比特币最好的继承者之一.智能合约为去中心化应用(decentralized application, Dapp)提供了良好的平台，其不仅是运行在去中心化节点上的程序，也是携带金钱属性的分布式账本.

作为一种新生的技术，智能合约不仅具有传统应用程序上常见的漏洞，同时由于其分布式和金融的特性还存在着各种潜在的新型漏洞，给智能合约的开发者和消费者带来了巨大的经济损失.2016年的TheDAO事件[1]首次利用了重入(reentrancy)漏洞，造成了接近6 000万美元的损失；2017年的Parity Wallet注入漏洞[2]使相关合约共计损失了近310万美元；2018年MyEtherWallet遭受边际网关协议(Border Gateway Protocol, BGP)和域名系统(domain name system, DNS)劫持攻击[3]，损失达170万美元.可以看出，随着智能合约的发展和应用，被攻击者所利用的漏洞类型也在不断的变化.这些攻击带来的巨大损失迅速引起了安全研究领域的重视，大量针对智能合约的检测和防御的方法被提出以应对日益严峻的安全形势.

目前针对智能合约的防御手段主要分为3类：1)基于修改合约代码，对合约代码的安全性进行加固，包括使用安全框架如OpenZeppelin[4]，安全的高级编程语言Vyper[5]等；2)利用静态或动态的分析工具对合约进行漏洞分析，例如Oyente[6]和Mythril[7]；3)基于合约的运行时信息，通过对比合约执行时的数据和已知攻击的特征，来检测当前的交易是否为某一类攻击行为，如果是则拦截该交易，如ContractLarva[8]和Sereum[9].其中前2类最终都要通过修改合约代码来修复漏洞.

这些现有的检测防御手段都是针对已知的漏洞，而没有考虑如何应对未来新型的漏洞和攻击，同时它们在实际应用中也存在着各自的缺陷.基于修改合约代码的防御措施对于已知的漏洞和攻击类型非常有效，然而，由于以太坊智能合约限制了其代码被部署到链上便不可修改，一旦攻击者利用新的漏洞进行攻击，这些防御措施不可能在原有的合约上对漏洞进行修复，因而也不可能使原有的合约防御新的攻击.而且，这类防御措施需要在合约上增加安全检查的代码，每次与合约的交互都会增加额外的执行开销，由于以太坊智能合约的执行需要向矿工支付金钱费用，这些额外的开销实际上都是由发起交易的消费者来承担.这意味着漏洞类型越多，需要额外增加的防御代码就越复杂，消费者额外承担的经济开销也就越多，这实际上形成一种恶性循环，为了解决这个问题，要么采用新的防御思路，要么想办法将部分执行开销转移到服务提供商上去.

相比于基于修改合约代码的防御方式，基于运行时信息的实时检测防御技术更加适合智能合约的特点和需求，但是现有的研究工作所提出的方法存在着极大的缺陷，难以被实际应用.ContractLarva[8]需要对合约进行插桩以获取控制流信息，这会大量增加合约代码，在合约部署时造成1 100%的额外开销，与合约交互时在最差情况下会造成20%的额外开销[8]，这在实际应用中是难以接受的.Sereum[9]基于修改后的以太坊虚拟机(Ethereum virtual machine, EVM)进行污点分析来检测重入攻击，为了实现拦截攻击交易的目的，一种方案是矿工接受1个中心节点的检测结果；另一种方案是所有矿工都在本地修改后的EVM上各自进行检测，前者违背了去中心化的理念，后者开销巨大且难以部署.

鉴于现有防御手段的缺陷和智能合约紧迫的安全形势，我们迫切需要一种新的实际可行的防御机制，能够实时地检测攻击，一方面对已经发出的交易进行拦截，另一方面限制对漏洞函数接口的访问，以

防止进一步的攻击.同时要方便部署和实际应用，不需要对合约进行频繁的修改，也不需要修改以太坊底层技术和共识协议，能够在面对新的攻击类型时灵活地更新并及时地进行防御.

针对以上的防御需求，我们提出基于以太坊交易运行时信息的防御框架，其中主要包含基于访问控制的防御机制和基于竞争的攻击阻断方法.下面我们用一个现实的案例来说明我们的整体设计思路.图1是一段防御整数溢出攻击的合约代码，行⑧～⑨，if语句用于检测溢出，当溢出被检测到后调用合约代码指令revert阻止交易，这里实际上分为攻击检测和防御响应2个步骤.我们将这2个步骤完全分离开，在链下进行攻击的检测，在合约上通过访问控制机制来进行防御响应.

```
① library SafeMath{
②   function add(uint256 a, uint256 b)
③     internal pure
④     returns (uint256)
⑤   {
⑥     uint256 c=a+b;
⑦     if (c≥a)
⑧       revert;
⑨
⑩     return c;
⑪   }
⑫   …
⑬ }
```

Fig. 1 Source code screenshot of a defense example

图1 防御示例源代码截图

对于链下的检测，我们通过本地同步的节点实时地获取多种运行时的信息，而不需要修改EVM或者对合约进行插桩，由于链下检测不依赖于合约代码，不再受到合约部署后不可修改的限制，因而可以灵活地进行更新.部署在合约上的防御响应机制不再包含攻击检测的部分，只需要根据链下检测的结果调整函数接口的访问控制策略即可，因此不需要随着新型攻击的出现而对合约进行修改，同时也避免了在合约上增加大量针对不同漏洞的防御代码，减少了合约执行的开销.由于以太坊本身的机制无法对攻击交易进行识别和处理，因而对攻击交易的实时阻止是很困难的，为此我们提出了基于竞争(race condition)的方案，利用矿工打包交易的机制，对实时攻击进行拦截.

本文的主要贡献有4个方面：

1）设计了一套基于以太坊交易运行时信息的链下可升级异常行为检测框架.

2）实现了一种基于访问控制的通用防御机制.

3）提出了一种基于竞争的主动攻击阻断方法，实现对于攻击交易的实时防御.

4）提出了一套完整的链下检测链上防御的系统模型，在降低合约执行开销的同时提供了可升级的特性.

我们进行了实验对本文提出的防御技术进行评估，实验结果表明，我们的技术可以有效地检测并防御攻击，对于后续的攻击交易，可以实现100%的拦截成功率，对于首次检测到的实时攻击，利用竞争可以达到97.5%的成功率，对于历史上的著名攻击事件，如果部署我们的防御系统，即使在不拦截首次攻击的情况下也能减少99%以上的损失.

1 背景知识

1.1 智能合约

以太坊首次提出了智能合约，被认为是继比特币之后的“第2代的区块链平台”[10].智能合约看作是可以运行在区块链上的程序和账本，其拥有唯一的账户地址和Ether(以太坊的虚拟货币)余额，也拥有可以执行的代码，并在被创建时一同发布到区块链上.当合约被部署到链上后，其代码便不可修改，直到合约被销毁掉，代码也一同被清除.普通账户可以发起交易(transaction)与合约进行交互，合约之间也可以相互调用并转移Ether，交互过程受交易的字段控制和影响，其中有6个重要的字段：

1）Transaction Hash.每个交易都对应唯一一个散列值(Hash).

2）Block Hash和Transaction Index.打包该交易的块散列值和该交易在块中被打包的序号，序号越小越先被打包，同时也被认为执行在前.

3）From Address和To Address.交易的发起方和接收方的地址.

4）Value.转移的Ether的数量.

5）Gas Limit和Gas Price. Gas是交易执行开销的最小单位，Gas Limit限制了在最多交易中使用的Gas数量，Gas Price指定了该交易提供的每个Gas的价格，最小单位是Wei，相当于10^{-18}个Ether.

6）Data.交易发送的数据，头部的4 B可以用来指定交互的函数接口，这4 B实际上是函数签名的散列值，当没有找到对应的函数时会调用默认的回调函数[11].

当发起交易调用执行智能合约的代码，合约

可以通过 Call 等 API 指令进一步调用其他合约或账户，实际上 1 笔交易可能由多个合约间的调用组成，这些合约间调用的参数和数据以 Trace 的形式进行表示.

1.2　以太坊虚拟机

为了运行智能合约，以太坊实现了 EVM[12] 作为计算和执行的环境.EVM 基于栈(stack)来处理和解释执行智能合约的字节码(bytecode)，并读取和修改合约的状态.每个以太坊节点都可以看作一个本地的 EVM，矿工节点在打包交易时需要在 EVM 上执行合约并更新状态，为了激励矿工消耗计算资源，执行智能合约的每条操作指令都需要消耗一定数量的 Gas，最终消耗的 Gas 总量和 Gas Price 计算出打包费用支付给矿工.对于矿工而言，挖矿的收益来自打包的奖励(mining reward)和交易的开销(gas cost)[13]，这意味着 Gas Price 越高收益就可能越大，因此矿工默认都会优先选择 Gas Price 较高的交易进行打包.

1.3　开发语言

开发者可以使用多种高级语言开发智能合约，然后编译成 EVM 字节码创建到链上.目前最为常用的开发语言是 Solidity[14]，它是一种图灵完备的高级编程语言，拥有和 JavaScript 相似的语法，其编译器 solc 负责将其编译成字节码.本文的工作也以 Solidity 作为合约的编写语言来实现.

2　系统设计

本文提出了一种基于运行时信息的可升级智能合约检测防御技术，本节首先对该防御技术的系统模型进行介绍；然后基于本文的防御技术，提出了一种新的智能合约安全生态系统；最后阐述了在这种安全生态系统下，本文的防御技术如何更加有效地防御新型的攻击和漏洞.

2.1　系统模型

整个系统模型分为 3 个部分，如图 2 所示：

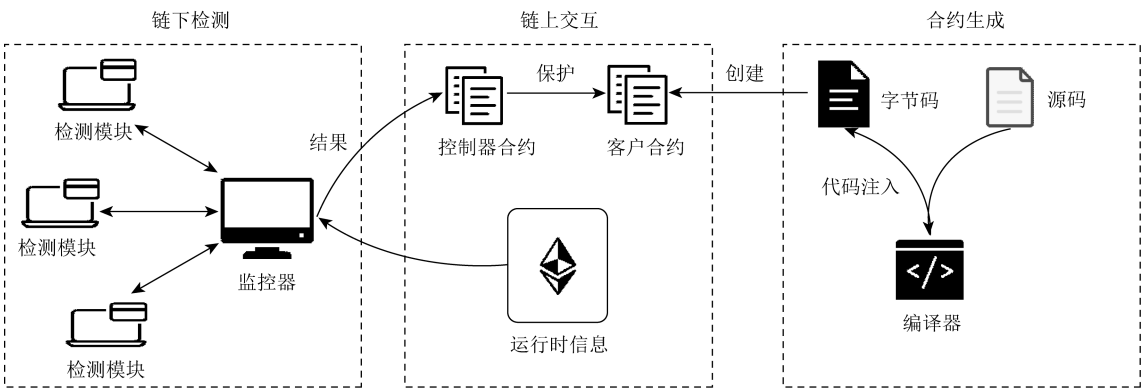

Fig. 2　Smart contract detection and defense system model

图 2　智能合约检测防御系统模型

1) 合约生成(contract generation).我们通过 1 组控制参数来对合约的访问进行限制，通过修改编译器，将实现访问控制的相应代码自动化地插入到合约中并编译为字节码，这一过程对开发者透明，开发者无需对合约进行任何修改和干预.随后发起的交易(transaction)将客户合约(client contract)创建到区块链上，并自动将新创建的合约地址注册到控制器(controller)合约上.

部署在合约端的访问控制机制是实现防御效果的关键之一，根据攻击检测的结果，至少需要 3 种粒度的访问限制：

① 基于访问者身份和访问的函数接口进行限制.这是最细粒度的访问限制，只在特定的访问者对特定的函数接口进行调用时才限制其访问，最大程度地减少对于正常交易的影响.

② 基于访问者身份进行限制.一旦交易被检测确认为攻击交易，那么发起交易的相关账户会被认为是受恶意攻击者所控制的，因此会拦截这些攻击者账户发起的所有合约交互.

③ 基于函数接口进行限制.这是最粗粒度的访问限制，当某个函数接口频繁受到攻击，需要对访问该接口的交易频率进行一定程度的限制.

之所以至少需要设置 3 种粒度的访问限制，是为了平衡防御的效果与访问限制对正常交易造成的影响，由于对攻击的检测很难做到 100%的准确率，可能存在部分误报，有些攻击造成的影响可能并不

严重,如果采取过度的反应可能对正常交易的影响要超过攻击带来的损失.

2) 异常行为检测.链下的异常行为检测框架实例(monitor)会对以太坊交易进行监控.通过本地同步的以太坊节点提供的接口,监控器能够实时获取最新交易的运行时数据(runtime data),并经过整理和中间处理后交给各个不同的异常行为检测模块(checker)进行分析,最终分析结果会被整理汇总,并以发起交易的形式递交给链上的控制器合约以对遭受攻击的客户合约实施防御.检测模块只需根据交易的多种运行时信息进行分析检测,而无需访问控制器和客户合约链上的数据.

3) 防御响应.作为防御系统在链上最为重要的一环,控制器负责根据检测结果和存储在控制器合约上的状态信息来决定防御的措施,通过更新客户合约的控制参数来实现防御.

在我们的系统模型中,各个部分分别对各自的数据和状态进行存储和处理,而无需进行数据的访问和同步,避免了数据访问带来的开销和不一致的问题.同时,整个检测防御系统处于一种松耦合的状态,在确保接口一致的情况下,各个部分和模块都可以自由替换和更新.

2.2 智能合约安全生态系统

在本文提出的系统模型中,检测防御的过程完全可以和合约代码开发过程进行分离,这使得一个防御系统实例可以为多个合约提供服务.

现有的智能合约生态中,已有安全厂商为开发者提供代码审计和漏洞检测等服务,但不能做到实时的防御.基于本文提出的基于运行时信息的智能合约检测防御技术,我们可以搭建一套新的安全生态系统,为智能合约提供更强更及时的安全保障,如图3所示.消费者(customer)使用基于智能合约实现的去中心化应用(Dapp)所提供的服务和功能,Dapp

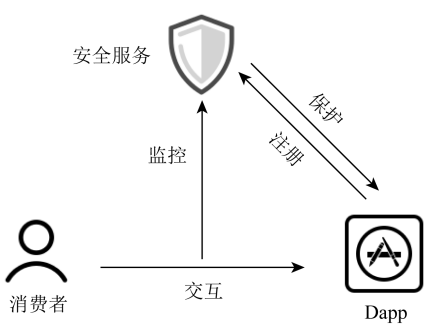

Fig. 3 Smart contract ecosystem

图3 智能合约生态系统

向第三方的安全服务商(security service provider)付出一定费用以获取安全服务从而专注于合约的功能实现,安全服务商根据运行时的信息对Dapp进行实时的防御.在这样一种生态系统中,安全服务商专注于对漏洞和攻击的检测防御,在新型的漏洞刚被发现时就及时对防御进行升级,从而有效降低Dapp被攻击的风险.

2.3 新型攻击和漏洞的防御

基于本文提出的防御技术和智能合约安全生态系统,整个防御的流程如图4所示:

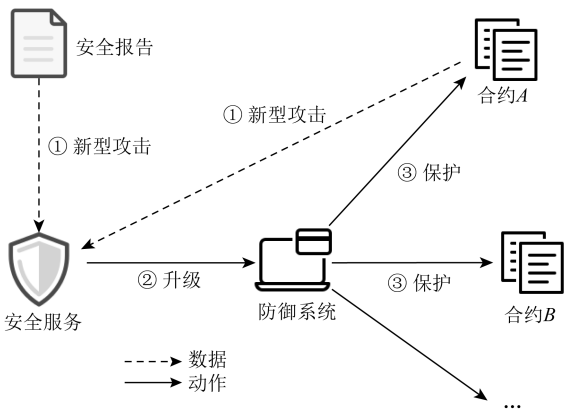

Fig. 4 Defense on new attack and vulnerability

图4 新型攻击和漏洞的防御

1) 安全服务提供商实时地对合约进行监控并关注最新的智能合约安全报告,一旦发现新的攻击和漏洞,可以采取应急措施关闭被攻击合约相应的接口,并对攻击的特征进行分析.

2) 获取新型攻击的行为特征后对防御系统的检测模块进行更新升级.

3) 对于已经被攻击的合约A,我们可以在无需修改替换合约A的情况下防御后续的攻击.对于暂未受到攻击合约B,可以在无需人工确认是否存在相同漏洞的情况下实现对新型攻击的防御.

相比现有的防御方式,本文提出防御技术能够更快地实现对现有合约的防御,减少新型攻击造成的损失,并且无需对现有的合约进行漏洞检测和修复.

3 系统实现

本节详细阐释了系统的实现,按照系统设计模型分为3个部分:访问控制机制、异常行为检测框架、防御响应机制.访问控制机制实现在合约端,基于访问者身份和访问的接口进行限制来防御攻击,

异常行为检测框架实时地监控以太坊网络上的交易并检测异常的行为，防御响应机制根据检测的结果调整合约的访问控制策略来对攻击进行防御.

3.1　访问控制机制

为了使智能合约在无需修改的情况下应对新的攻击类型，我们需要一种通用的防御手段，通过响应控制中心的指令来更新合约的访问控制策略，从而达到拦截攻击或限制可疑交易的目的.

我们设计了一种运行在智能合约上的简单高效的访问控制机制，通过多组动态调整的控制参数来实现对合约访问的限制：

1）允许通行名单(transit allowed list).合约项目方的某些账户可能会发起大额转账、权限修改等敏感操作，这类行为可能会被检测模块认定为异常行为，并被防御系统拦截，为此需要设置允许通行名单对这些账户发起的交易直接放行.

2）禁止通行名单(transit banned list).当交易被防御系统检测为攻击行为时，比如重入或溢出攻击等，其交易发起的相关地址被当做攻击者加入禁止通行名单，所有这些地址直接或间接与客户合约进行的交易都会被拦截.账户地址一旦加入该名单只能被防御系统管理员或者合约拥有者人工解除.

3）限制通行名单(transit restricted list).某些异常的交易不能被明确认定为攻击或者危害较小，其相关发起账户会被放入限制通行名单，对其访问频率进行限制，在不完全阻断可疑账户交易的同时尽可能减少损失.

4）访客通行时间限制(visitor transit time restriction).账户第1次加入限制通行名单时会初始化限制时间和下一次允许访问时间，通过与当前交易的块时间戳(block timestamp)比较来决定是否放行.当该账户多次被检测到异常行为时，限制时间和下一次允许访问时间间隔会被增加，直到被移出限制通行名单.

5）接口通行时间限制(interface transit time restriction).在以太坊中攻击者可以很容易地创建新的账户来更换身份，因此上述基于身份的访问限制手段可能会失效.当检测到针对某一函数接口的异常行为频繁发生，需要对该接口受访问的时间频率加以限制，直到异常行为不再发生或减少.

6）防御等级(security level).对可疑交易的访问进行限制可能会误伤正常合法的交易，对合约的正常使用造成影响，因此需要设置不同的防御等级来调整对可疑交易的防御策略.防御等级越高，限制就越严格，这一过程主要通过动态调整限制通行时间来实现.

在我们的访问控制机制中存在4种角色：客户合约、合约拥有者、控制器和访问者.合约拥有者会被初始化为客户合约的创建者，并且只能被当前拥有者所修改，其拥有更新控制参数和更换控制器合约的权限.控制器是防御系统部署在链上的开源合约，通过修改客户合约的控制参数来实现访问控制从而达到防御的目的.访问者是对合约直接或间接进行访问的账户或者合约，既包括交易的发起账户也包括到客户合约中间路径上的其他合约，是潜在的攻击者.

整个访问控制流程如图5所示.当交易访问到客户合约，合约端的防御机制会基于访问控制的参数对交易进行校验，分为5个步骤：

1）获取交易的访问者信息，由于以太坊的智能合约只能获取到交易的发起者账户tx.origin和当前调用的直接发起者msg.sender，我们只对交易的这2个身份信息进行检查，对于绝大多数攻击案例，这2个身份信息已经足够鉴别攻击者的身份.

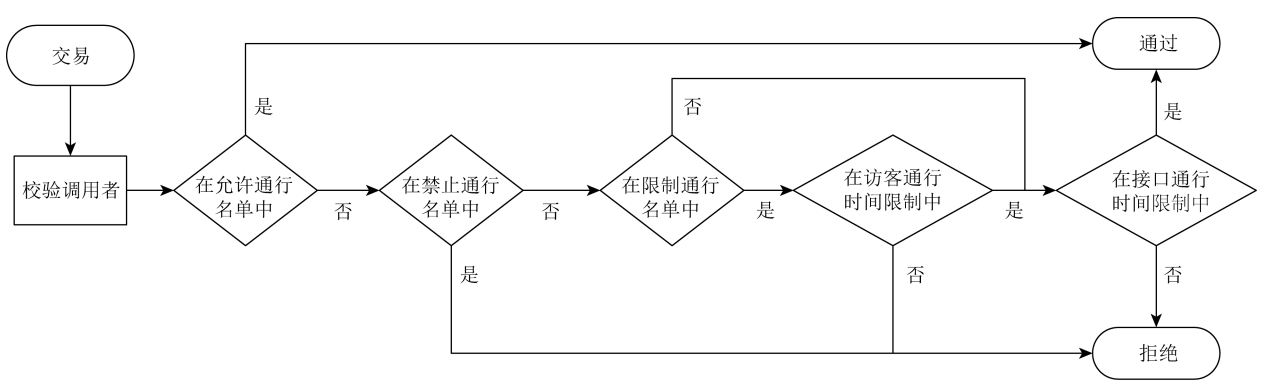

Fig. 5　Smart contract access control processes

图5　智能合约访问控制流程

2）检查访问者的地址是否在允许通行名单中，如果是则直接放行.

3）检查访问者的地址是否在禁止通行名单中，如果是则说明该交易是由已知的攻击者发起的，且可能会造成严重的危害，因此直接拒绝访问，否则进行下一步.

4）检测访问者的地址是否在在限制通行名单中，如果是且还在限制时间内，则根据该交易的块时间戳来判断是否达到下一次允许访问时间，是则允许通行，否则拒绝访问.如果访问者的地址不在限制通行名单中，或者已经超过了限制时间则直接进行下一步.

5）检测当前交易访问的函数接口是否被限制访问，如果是则根据当前交易的时间戳判断是否达到该接口的下一次允许访问时间，是则放行，否则拦截.

我们基于 Solidity 编译器实现了以上的访问控制机制.编译器会在合约编译过程中自动插入访问控制和初始化的代码，对开发者透明，使得本文提出的防御系统更加具有可行性.

3.2 异常行为检测框架

作为整个防御系统中最为重要的部分之一，异常行为检测框架对所有实时的交易进行检测并反馈结果，为了方便升级或增加新的检测模块以应对新型的攻击，我们将整个检测过程尽可能地解耦，如图 6 所示，分为 3 个部分：

1）数据收集器(collector).通过本地同步的以太坊节点提供的接口获取实时的运行时数据，包括 2 类，最新被打包进块中的(latest)和等待被矿工打包的(pending).

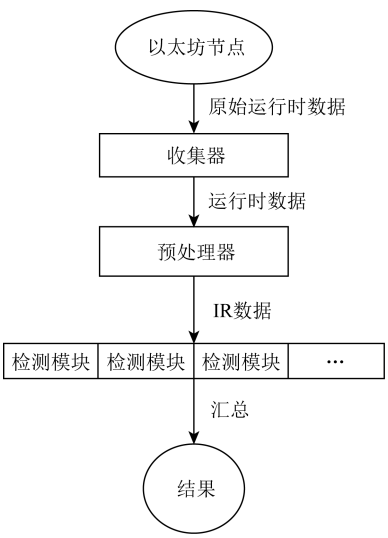

Fig. 6 Abnormal-behavior detection framework

图 6 异常行为检测框架

2）预处理器(pre-processor).对数据收集器获取到的实时数据进行中间处理，以便后续的检测分析.用于减少分析模块的重复工作，提取通用的中间表示数据(IR data).

3）分析模块(checker).基于提取后的中间表示数据与多种类型的攻击行为特征进行匹配，并将分析的结果进行整理汇总.

我们利用本地同步的节点(Ethereum node)，为检测框架提供实时的区块链数据：

1）Block.最新被打包的块本身的信息，其中块交易数量、Gas 消耗等数据可以用来分析区块链网络的拥堵情况.

2）Transaction.以太坊账户发起的交易，包含发起方和接收方的地址、Gas 价格以及总的 Gas 消耗等数据.

3）Trace.交易中账户(包括合约)之间详细的调用信息，包括调用方和被调用方的账户地址、调用的接口和参数等输入，以及返回值和错误信息等数据.

4）vmTrace.与 Trace 不同，vmTrace 反映的是交易在 EVM 中指令级的执行路径，可以作为很多现有的合约分析工具的数据来源.

5）Event.可以看作是合约执行的日志，一些合约接口标准会要求规范一致的 Event，比如 ERC20[15] 确立的代币(token)接口标准中要求代币的转移需要发出 Event 来标明转移的发送地址、接收地址和代币的金额，通过 Event 我们可以方便地提取出这些信息.

数据收集器(collector)通过本地同步的以太坊节点提供的接口获取以上 5 种运行时数据，包括 2 类：Latest 和 Pending.

1）Latest 是最新打包的块，数据收集器会不断尝试获取最新的块数据，确保交易一旦被打包到链上便立即被我们获取分析.

2）当交易刚被发起还没有被矿工打包时便处于 Pending 状态[16]，在矿池中等待被打包，矿工为了收益的最大化并不会按照交易的发起顺序进行打包，更多时候会优先选择 Gas 价格更高的.

已经被打包的交易数据全部可以被直接获取，处于 Pending 状态的交易的一部分数据可以被直接获取，但 Trace 和 vmTrace 的数据需要通过在本地 EVM 上执行才能获取.我们每隔很短时间就会尝试获取 Pending 中的交易，从而尽量使得交易刚被发起还未被打包就能被分析检测，这对防御效果能起到很大的提升.

经数据收集器收集整理后，运行时数据会被传递给预处理器（pre-processor）进行中间处理，提取通用的中间表示数据（IR data）：

1）交易在合约间的调用关系树.利用 Trace 中的字段可以重建出完整的调用链并以树的形式来表示.

2）被调用函数.合约调用 Data 字段的头部 4 B 可能对应着多个不同的函数签名，因此仅通过 Data 字段来确定被调用函数和参数是很困难的，我们只能对常见的函数进行解析.

3）转移金额.包括 Ether 和代币.Ether 转移的数量很容易从 Trace 的字段中获取；代币需要解析 Event 才能获取到代币合约的地址、转出转入地址以及数量，如果是 Pending 交易，链上并没有对应的 Event 数据，我们只能从 Data 字段中解析函数和参数来获取.

这些中间表示数据会和原始数据一同传递给各个异常行为分析模块（checker）进行分析检测.新的分析模块只需要保持一致的数据输入和结果输出接口便可以方便地注册到检测框架中.

我们基于 4 种攻击的特征和运行时信息实现了相应的检测模块以进行后续的实验，并以图例的形式来表示攻击的特征，一种是行为上的特征（action clause），另一种是收益上的特征（result clause），如图 7～10 所示：

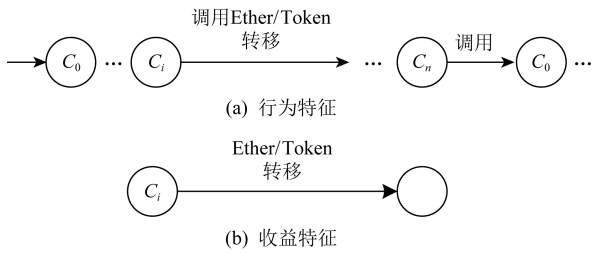

Fig. 7　Reentrancy attack feature

图 7　重入攻击特征

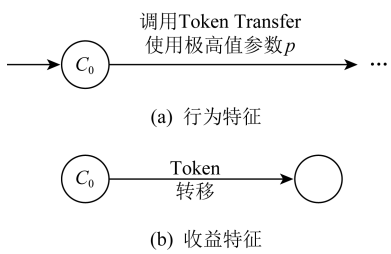

Fig. 8　Integer overflow attack feature

图 8　整数溢出攻击特征

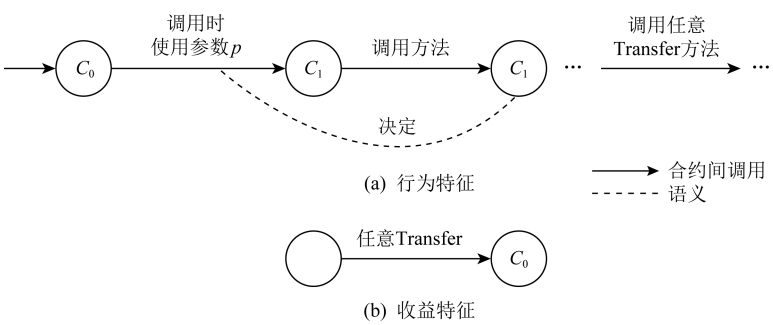

Fig. 9　Call injection attack feature

图 9　代码注入攻击特征

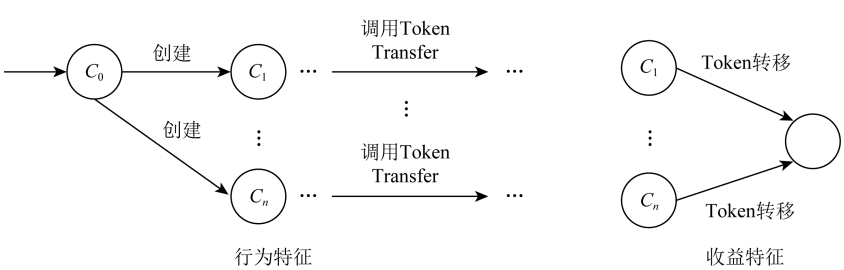

Fig. 10　Airdrop hunting attack feature

图 10　代币空投攻击特征

1）重入攻击（reentrancy attack）[17].攻击者利用智能合约函数调用的缺陷，在被攻击合约状态未更新时通过回调函数执行攻击合约的代码，重复进入被攻击合约进行提现等操作.C_0 是攻击者部署的

中间合约,经过对1个或者多个合约的调用后,被攻击者合约 C_n 再次调用 C_0,重复以上的过程,如果将合约地址作为图上的节点、合约间的调用作为连线,那么重入攻击的调用过程在图中会形成多次重复的环.其在收益上的特征表现为,每重复一次上述的过程,都会向同一个账户转移一定的以太币或者代币.

对于重入攻击在行为上的特征,我们可以利用交易运行时的 Trace 数据,构造图并根据环的特征来进行匹配,同时分析每一次循环中金钱的转移情况,从而检测重入攻击.

2) 整数溢出攻击(integer overflow attack).智能合约的整数溢出攻击一般都是针对代币合约,如果代币发行和转移的函数中没有检查整数算术溢出或者实现的方式不正确,很可能允许攻击者盗取大量的代币.它在行为上的特征是对代币转移的标准函数接口进行调用,在收益上的特征是大量的代币转移到攻击者账户.

对于整数溢出攻击行为上的特征,我们可以对交易 Data 字段进行解析,根据建立的函数签名对照表,检查是否调用了代币转移的标准函数接口,同时分析该交易的 Event,检查代币的实际转移数量是否高于函数调用参数中的值且远远超过正常值.

3) 代码注入攻击(call injection attack).在以太坊智能合约中,大部分合约通过 msg.sender 来检测调用者是否是合约的拥有者,msg.sender 是合约间的直接调用者.合约可以对其他合约发起调用,调用的接口和参数可以被上级调用的 Data 字段所指定,假如攻击者控制 Data 字段使被攻击合约对自身进行调用,就可能绕过权限的检测.

其在行为上的特征一般是发生自环调用,且调用的函数由前一个调用 Data 字段所指定,收益上的特征是发生了虚拟货币的转移或者是权限的转移.我们可以通过构造图来检查是否发生了自环调用和收益来检测注入攻击.

4) 代币空投攻击(airdrop hunting attack).代币发行方为了吸引人气可能会以免费发放代币的方式吸引新的用户,由于代币只通过调用者的地址来区分是否为新用户,攻击者为了大量重复地获取免费的空投,可以通过攻击合约自动创建大量不同地址的新合约,来无限地领取空投.

这种攻击在调用树上具有明显的行为特征,一个中间合约大量自动创建合约,并调用转移代币的合约接口将获取到的空投转移到同一个账户中去.对于每一个新创建的合约,它的子调用路径上都会产生代币的收益.我们可以构造调用关系树并分析其中的代币收益情况来检测这类针对代币空投机制的攻击.

最终检测后的结果会进行整理汇总,包含5个必要信息:

1) 受害者合约(victim contract).当检测到异常行为时,需要检查受害者合约是否注册到控制器合约上,只有已注册的客户合约才能对其进行防御.

2) 攻击者账户地址(attacker account address).用来调整客户合约中的控制参数以防御后续的攻击.

3) 被攻击合约接口(attacked interface).用于更新客户合约的接口访问时间限制.

4) 风险等级(risk level).用于调整防御强度.

5) 交易打包状态(transaction mining status).包含已打包(mined)或者待打包(pending)两种情况.

3.3 防御响应与控制

异常行为检测框架完成对实时交易的检测分析后,如果存在异常会以发起交易的方式将检测结果递交给链上的控制器实施防御.如果异常交易处于 Pending 状态,意味着我们有机会使得防御措施在异常交易被打包之前生效,从而拦截实时的攻击交易.为了实现这一点,我们需要利用竞争[18]的机制,在以太坊中,矿工会基于 Gas Price 来选择交易打包,这为我们提供了契机.当发现异常交易处于 Pending 状态时,检测框架会根据该交易的 Gas Price 计算出一个远高于它的值,作为向控制器合约发起的防御交易的 Gas Price.如果检测流程足够快,异常交易和防御交易发起的时间差会非常小,由于以太坊出块时间相差数秒,在理想情况下,防御交易会被打包在异常交易之前,从而实现对实时攻击的拦截.

如图11所示,当防御交易被发起后,控制器会根据检测的结果和链上的信息来决定防御的措施,其链上保存的信息包括:

1) 已注册客户合约名单(registered client contract).只有注册到控制器并实现了相应访问控制机制的合约才能进行防御.

2) 限制通行名单续期时间(renewal time of transit restricted list).访问者多次被检测到异常行为时增加的限制通行时间.

3) 访客通行时间间隔(visitor transit time interval).访问者下一次访问客户合约所需的时间间隔,为了限制可疑账户的访问频率.

4) 接口通行时间间隔(interface transit time interval).客户合约接口下一次被允许外部访问的时间间隔.

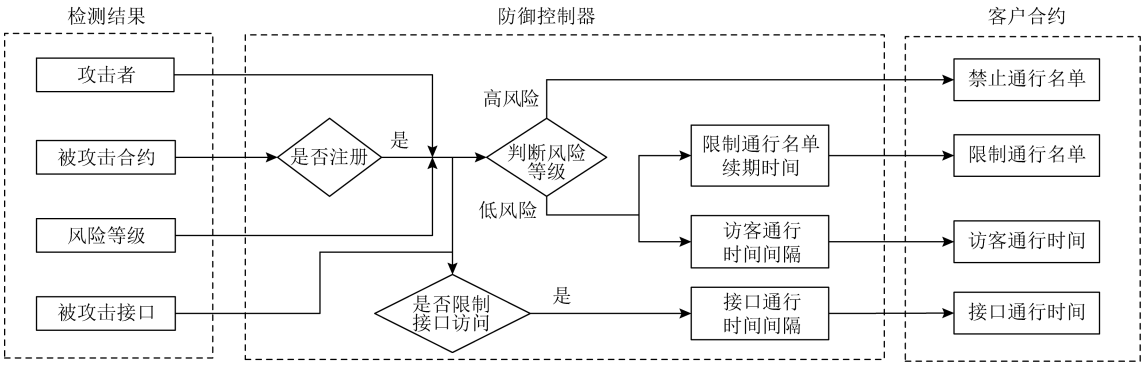

Fig. 11　Defense controller workflow

图 11　防御控制器工作流程

当控制中心收到检测的结果后，按照 2 个流程对客户合约的访问控制参数作出调整：

1）检查受害者合约是否是已注册的客户合约.如果没有注册说明该合约上没有部署相应访问控制的代码，无法对防御控制中心的防御指令进行响应，因此直接拒绝操作；如果已注册，则根据后续的流程来决定防御措施，并调整相应的状态和访问控制的参数.

2）检查异常行为的风险等级.对于高风险的攻击行为，比如重入攻击、溢出攻击等，我们认为其发起者是危险的攻击者，该交易所有相关的账户和中间合约的地址会被加入到客户合约的禁止通行名单中，对它们相关的所有后续交易直接进行拦截.对于低风险的异常行为，由于其危害较低，并且可能是某些账户偶尔的误操作，不应对其完全封锁，应进行适当的限制和观察.我们会将这些账户加入限制通行名单中，在一段限制时间内对其访问合约该函数接口的频率进行一定程度的限制.为了实现这种防御措施，我们需要对控制器的 2 种防御限制进行更新：

① 限制通行名单续期时间(renewal time of transit restricted list).当已被加入限制通行名单的账户继续发起异常行为时，其增加限制时间，同时该时间也会按照一定规则进行增加.

② 访客通行时间间隔(visitor transit time interval).用来限制基于身份的访问频率.如果某个账户被加入到限制通行名单中，那么它在这段限制时间内只能按照访客通行时间间隔进行访问，未达到时间间隔要求的访问会被直接拒绝.

3）检查是否需要对被攻击的函数接口进行访问限制.我们基于 2)中①②的防御措施对于基于身份的防御十分有效，但是一旦攻击者不再使用固定的账户或者中间合约，基于 2)中①②的防御措施就一定程度失效.在以太坊中创建普通账户的代价为零[19]，用户只需要在本地节点或通过线上的区块链钱包生成账户秘钥即可创建一个新的账户，创建合约的开销也极低，只需要为创建合约的交易支付一定数量 Gas[20].因此，攻击者可以通过链下的脚本，自动化地创建新的账户和中间合约来进行攻击，从而绕过基于身份的防御措施.在这种极端的情况下，我们需要对被攻击接口的访问作出无差别的限制(允许通行名单中账户发起的访问除外).

为了应对绕过基于身份防御措施的攻击，对于被攻击客户合约的函数接口，控制中心会根据 3 个参数作出反应：

1）接口访问时间间隔(interface transit time interval).决定了该函数接口被允许访问的频率，会根据客户合约的防御等级设置一个初始值.

2）风险分值(risk score).风险分值用来反映该函数接口目前被攻击的风险程度，每一次对该函数接口的异常访问行为都会根据其风险等级一定程度地增加风险分值.

3）风险阈值(risk threshold).根据客户合约的防御等级设置初始值，只能被手动地更改，这是考虑到防御强度和防御副作用之间的平衡需要开发者根据需求去人工调整.当该客户合约的函数接口累积的风险分值超过了其对应的风险阈值，就会触发对函数接口的访问限制.

如果当前异常交易成功触发了对相应函数接口无差别的访问限制，那么该接口需要每隔一段时间间隔才能够被访问.假如在这段时间间隔内又检测到对该接口的异常交易，那么时间间隔会被按照一定规则增加，直到整个时间间隔都没有异常交易发生，限制才会解除.通过这种机制，我们可以在一定程度上应对高频率的匿名攻击，将其损失降低到最少，

同时也会误伤到正常的交易,但相比于攻击带来的风险,我们认为这种防御的副作用是可接受的.

4 实验结果与分析

我们设计了实验来检验本文提出的智能合约检测防御技术的有效性,主要分为4个方面:

1) 当检测到攻击后,能否有效地进行防御;

2) 竞争的机制能否有效地拦截实时的攻击;

3) 能否有效地降低攻击造成的损失;

4) 与其他相关工作的比较.

4.1 实验准备

由于实验设备和资金的限制,我们选择在以太坊测试网络 Ropsten[21] 上进行测试,本地系统为 macOS 10.15.3,以太坊客户端为 Parity[22] v2.7.1,同步模式为 trace+fast,最多可以获取到最近10个块的完整数据,基于 Solidity v0.5.13 实现了定制的编译器.同时,我们从 Google BigQuery[23] 获取了以太坊创世块到2019年3月之间所有的交易数据,以对历史攻击交易进行重放检测.

我们从公开报道的攻击事件中选取合约进行实验,由于需要将旧版本的合约移植到较新的 Solidity 版本上去,我们对重入漏洞、溢出漏洞、注入漏洞和空投漏洞每种漏洞分别选取3种开源合约,设定攻击交易的 Gas Price 为16 Gwei,对于以上每一种漏洞的每个合约,分别进行10轮攻击测试,每轮测试包含连续3次攻击,完成后重置合约状态.其中每轮测试中第1次攻击主要是为了测试竞争机制的防御效果,后续2次攻击是为了验证防御响应是否生效.

为了测试竞争机制对于第1次攻击的防御效果,我们设置了较高的 Gas Price 发起攻击交易,以尽可能模拟真实的攻击场景,提高防御的难度.当检测到攻击交易处于 Pending 状态时,我们会为防御响应的交易设置2倍于攻击交易的 Gas Price 以进行竞争.

同时,我们选取了以太坊智能合约上著名的攻击事件,TheDAO[1], Parity Wallet Attack[2], SpankChain[24],对它们从合约创建到2019年之间的历史交易数据进行重放,以评估本文的检测防御技术在后续攻击成功拦截的情况下减少损失的程度.为了与其他工作进行比较,我们重放检测了2019年3月前的所有以太坊交易.

4.2 结果与分析

在我们的实验结果中,每种漏洞每一轮的第2,3次攻击都被100%拦截,表明攻击被检测到后防御机制都成功生效,后续的攻击无法对漏洞合约造成进一步的损害.对于每轮的第1次攻击,当被我们的防御系统检测到时,攻击交易已经在打包池中等待被矿工打包,由于以太坊本身并没有拦截异常交易的机制,我们只能尝试通过竞争使得防御响应的交易在攻击交易之前生效,从而实现对首次攻击的拦截.竞争能否成功取决于3个因素:

1) 攻击交易的 Gas Price. Gas Price 越高,被矿工打包的速度就越快.

2) 区块链网络状况.在网络通畅或者出块速度较快时,发起的交易可能很快就被打包.

3) 攻击交易与防御交易发起的时间间隔.间隔越长,防御交易就越可能后被打包.

表1是竞争机制对首次攻击交易的防御结果.从实验的结果来看,即使攻击交易设置较高的 Gas Price,97.5%的首次攻击都被成功拦截,其中3次拦截失败的原因可能是攻击交易发起后正好被矿工打包,因此后发起的防御交易未能打包在攻击交易之前.表2是首次攻击交易发起后的平均时间间隔,可以看出,从首次攻击交易发起到防御交易发起,其间的平均时间间隔只需要0.533 s左右,而以太坊 Ropsten 测试链的出块时间间隔为3 s以上[25],即使是主链也远高于0.5 s.只有在极少数情况下,攻击交易在这0.5 s左右的间隙间正巧被矿工打包,导致对

Table 1 Defense Result of the 1st Attack

表1 首次攻击防御结果

攻击	实验次数	防御成功次数	成功率/%
重入	30	29	96.67
溢出	30	30	100
注入	30	28	93.33
空投	30	30	100
总计	120	117	97.5

Table 2 Average Time Interval After the 1st Attack Launched

表2 首次攻击发起后的平均时间间隔 s

攻击	检测攻击时间	发起防御时间
重入	0.410	0.453
溢出	0.566	0.610
注入	0.509	0.556
空投	0.469	0.511
平均值	0.489	0.533

首次攻击的防御失败.本次实验对首次攻击的成功防御中,98%的攻击交易和防御交易被打包到同一个块中,但防御交易的 Transaction Index 更小,因为矿工在打包时优先选择了 Gas Price 更高的防御交易.

即使在不考虑竞争的情况下,仅对后续攻击进行拦截也能够极大地减少被攻击合约的损失,我们对 3 个著名事件的攻击交易进行了重放,以测试本文防御技术的拦截效果.如表 3 所示,从第 2～7 列开始分别表示攻击合约数量、攻击交易数量、直接损失的 Ether 数量、拦截的攻击交易数量、减少损失的 Ether 数量、减少损失的比率.对于每个攻击事件,除了每个攻击账户的第 1 次攻击交易,其他所有后续的攻击都能被成功拦截.

Table 3　Loss Saved of Well-Known Incidents with the Defense Technique in This Paper

表 3　著名攻击事件使用本文防御技术能够减少的损失

事件	攻击合约数量	攻击交易数量	直接损失的 Ether 数量	拦截的攻击交易数量	减少损失的 Ether 数量	减少损失的比率/%
TheDAO	23	1 848	11 829 473	1 825	11 814 833	99.88
Parity Wallet Hack	54	2 710	204 851	2 710	204 851	100
SpankChain	2	8	165	6	164.567	99.74

TheDAO 遭受了 1 848 次重入攻击但实际上只有 23 个不同的攻击者账户,这其中不仅包括恶意攻击者,还包括白帽子(WhiteHat)发起的善意的攻击,这些攻击总共造成了 1 000 多万个 Ether 的流失,其中大约 360 万个为被恶意攻击者窃取的.尽管对 DAO 的攻击很快就被发现,但是开发者并没有办法对攻击进行阻止,而在应用本文防御技术的情况下,我们可以对攻击交易进行拦截,当检测到这些攻击者的第 1 次攻击只对后续攻击交易进行防御,也能减少 99.88%的损失.SpankChain 只有 2 个攻击账户,而且首次攻击的损失极小,可能是作为正式攻击前的测试,大部分损失产生于后续的攻击交易,因此对后续的攻击拦截能够减少 99.74%的损失.Parity Wallet Hack 的攻击实际上分成 2 个步骤:1)利用注入攻击获取权限;2)获利.这 2 步由 2 笔交易完成,当第 1 步的攻击交易被成功检测到后攻击者的账户会被加入禁止通行名单,攻击者的第 2 步进行获利的交易实际上会被拦截,因此我们可以避免全部的损失.

4.3　与相关工作的比较

现有的提出了智能合约防御技术的研究工作按照防御的实现方式分为 2 类:1)基于修改客户端的 EVM 实现对攻击交易的拦截,目前有 Sereum[9] 和 ÆGIS[26];2)在合约源码中增加访问控制的代码,目前有 ContractLarva[8] 和 ContractGuard[27],以及本文的工作.这 2 类防御技术的核心都是对包括攻击在内的异常行为进行检测和匹配,再根据结果拦截交易.

Sereum 和 ÆGIS 的工作极为相似,都是通过修改 EVM 获取交易的控制流和数据流的信息与已知的攻击特征进行匹配,并在矿工层面拦截交易.Sereum 仅对重入这一类的特定攻击进行检测,只检测出了 2 个真正被重入攻击的合约,ÆGIS 基于 Sereum 拓展了对多种攻击类型的检测,并明确了通过矿工客户端拦截交易的防御方法,其检测出了 12 个被重入攻击的合约,包含 Sereum 的 2 个合约.本文对历史交易进行重放,检测出并经确认的被重入攻击的合约共 26 个,ÆGIS 公开地址的 7 个合约中 6 个同样被我们检测出受到了重入攻击,另外 1 个合约地址为 0x304a554a310c7e546dfe434669c62820b7d83490,该地址在我们检测结果的相关合约地址中,ÆGIS 认为这是跨合约重入攻击的受害者合约,但根据我们的重入攻击特征,该地址在重入攻击中是获利的合约,因此我们认为这是攻击者部署的中间合约而不是受害者合约,由于合约的源码并不开源,没有办法对此进行确认.

ContractLarva 和 ContractGuard 的方法也比较相似,都是通过对合约代码插桩以获取控制流的信息,与制定的异常行为或安全行为的规则进行匹配,在合约代码层面阻止交易.由于需要通过插桩的方式获取交易的运行时信息,会大量增加合约的代码,在合约创建和交互时带来极高的 Gas 开销.本文的防御技术同样需要将访问控制的代码插入合约中,但是由于不需要获取合约和交易的信息,仅根据相关的参数和访问者身份进行控制,因此插入的代码量和额外的开销是远远低于两者的.

5 相关工作

在智能合约安全学术研究领域,已经有大量的工具被提出以增强合约的安全性.目前研究最广泛的是分析合约代码以检测漏洞和风险,例如Vandal[28],Slither[29],SmartCheck[30]利用静态分析对合约的代码进行检查.Oyente[6],teEther[31],Manticore[32]通过符号执行的方法探索合约执行的路径以尝试触发漏洞.ZEUS[33]利用抽象解释来检测违反静态符号模型的异常行为,ETHBMC[34]基于符号执行构造了Model Checker以自动地检测漏洞,ContractFuzzer[35]采用模糊测试的方法.HoneyBadger[36]通过符号执行对蜜罐攻击进行了检测和分析,Simplicity[37]在中间表示层(inter-mediate representation)对合约进行形式化分析以找出安全问题.Vyper[5]和Flint[38]设计了更加安全的智能合约编程语言,移除了部分存在风险的功能,对溢出等漏洞进行了防范.

由于以上基于合约代码分析的工作在合约部署后难以发挥作用,另外一些工具尝试基于交易的运行时信息来进行检测和防御.ContractLarva[18]和ContractGuard[35]对合约代码插桩生成新的合约,并将获取的控制流信息与合约的行为规则(specification)或安全执行路径(safe path)进行对比来检查异常行为.Sereum[19]修改EVM利用污点分析(taint analysis)来监测合约执行过程中的数据流以检测重入(reentrancy)这一类攻击行为.ÆGIS[33]基于Sereum,提出一种描述攻击特征的语言,来对攻击交易进行实时的检测和拦截.

这些工作都没有很好地解决智能合约的防御需求,基于合约分析的方法不能对原有的合约进行修复,难以在合约部署后发挥作用.对合约插桩的防御方法带来的额外开销过大,并且会由消费者来承担,而修改EVM的防御方法违背了去中心化的原则且要求矿工承担额外的开销,不具备实际应用的可能性.

6 讨 论

本文提出的防御技术存在的问题和不足有2个方面:

1) 如果发起的攻击交易过多,控制器合约发起的防御交易的Gas Price是否会造成大量损失?

发起的攻击交易分为2种情况:1)由同一个或者少数几个攻击账户发起的大量攻击交易.在这种情况下每个账户只有第1次攻击会被检测出并发起防御拦截的交易,后续的攻击交易访问被攻击合约时实际上会被拒绝,其在运行时信息上的行为已经不具有攻击的特征,因此不会被检测为攻击也不会发起防御的交易,这种情况下防御交易实际发起的数量和带来的开销并不高.2)攻击者构造大量不同的账户发起的攻击交易.这种情况仅发生在攻击目标合约高价值的情况下,相比于少数攻击账户发起大量重复的攻击交易,对大量攻击账户和中间攻击者合约的创建及初始化需要耗费攻击者大量的开销,因此在这种情况下,总的防御开销会比较高,但是不会高于攻击者的开销,这部分额外的防御开销是可以接受的.

2) 接口通行时间限制功能当检测到某一接口异常行为频繁发生时会对该接口的访问进行限制,在实际使用中,是否会对正常交易造成影响?

当针对某一接口的异常行为频繁发生时,说明基于攻击者身份的防御措施已经失效,或者攻击者成功绕过了检测模块,这种情况下,在人工介入前对接口的访问进行限制是有必要的,这会在一定程度上影响正常的交易,但也会有效地避免合约的财产损失.

7 总 结

鉴于传统的以太坊智能合约防御手段存在着难以更新、执行开销大的缺点,本文提出了一种基于交易运行时信息的智能合约防御技术,通过链下可升级的攻击检测框架、链上通用的访问控制机制,实现了对合约实时攻击的有效防御,能够极大地降低损失.本文为智能合约的防御提供了一种全新的思路,相比于传统的方法,更加适应智能合约的安全形势.

参 考 文 献

[1] Vessenes P. Deconstructing TheDAO attack: A brief code tour[OL].[2020-02-10]. https://vessenes.com/deconstructing-thedao-attack-a-brief-code-tour/

[2] Palladino S. The parity wallet hack explained[OL].[2020-02-10]. https://blog.openzeppelin.com/on-the-parity-wallet-multisig-hack-405a8c12e8f7/

[3] Poinsignon L. BGP leaks and cryptocurrencies[OL].[2020-02-10]. https://blog.cloudflare.com/bgp-leaks-and-crypto-currencies/

[4] OpenZeppelin. A library for secure smart contract development [OL]. [2020-02-10]. https://github.com/OpenZeppelin/openzeppelin-contracts

[5] Eisenbach B. Pythonic smart contract language for the EVM [OL]. [2020-02-10]. https://github.com/vyperlang/vyper

[6] Luu L, Chu D H, Olickel H, et al. Making smart contracts smarter [C] //Proc of the 2016 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2016: 254-269

[7] ConsenSys. Mythril [OL]. [2020-02-11]. https://github.com/ConsenSys/mythril

[8] Ellul J, Pace G J. Runtime verification of Ethereum smart contracts [C] //Proc of the 14th European Dependable Computing Conf (EDCC). Piscataway, NJ: IEEE, 2018: 158-163

[9] Rodler M, Li W, Karame G O, et al. Sereum: Protecting existing smart contracts against re-entrancy attacks [C/OL] //Proc of the 26th Annual Network & Distributed System Security Symp (NDSS). Piscataway, NJ: IEEE, 2019 [2020-06-09]. https://www.ndss-symposium.org/wp-content/uploads/ndss2019_09-3_Rodler_slides.pdf

[10] Buterin V. Ethereum: A next-generation cryptocurrency and decent-ralized application platform [OL]. [2020-02-12]. https://bitcoinmagazine.com/articles/ethereum-next-generation-cryptocurrency-decentralized-application-platform-1390528211

[11] Solidity. Contract ABI specification [OL]. [2020-02-11]. https://solidity.readthedocs.io/en/v0.5.3/abi-spec.html

[12] Ethereum. Ethereum virtual machine (EVM) awe some list [OL]. [2020-02-11]. https://github.com/ethereum/wiki/wiki/Ethereum-Virtual-Machine-(EVM)-Awesome-List

[13] Ethereum. Ethereum mining [OL]. [2020-02-12]. https://github.com/ethereum/go-ethereum/wiki/mining

[14] Ethereum. Solidity, the contract-oriented programming language [OL]. [2020-02-12]. https://github.com/ethereum/solidity

[15] Vogelsteller F, Buterin V. EIP 20: ERC-20 token standard [OL]. [2020-02-12]. https://eips.ethereum.org/EIPS/eip-20

[16] dfuse. The 8 things that can happen to your Ethereum transaction and how to navigate them in your Dapp [OL]. [2020-02-12]. https://www.dfuse.io/en/blog/the-8-things-that-can-happen-to-your-ethereum-transaction-and-how-to-navigate-them-in-your-dapp

[17] ConsenSys. Smart contract security best practices [OL]. [2020-02-13]. https://consensys.github.io/smart-contract-best-practices/known_attacks/

[18] Wikipedia. Race condition [OL]. [2020-02-12]. https://en.wikipedia.org/wiki/Race_condition

[19] Ethereum. Managing your accounts [OL]. [2020-02-12]. https://github.com/ethereum/go-ethereum/wiki/Managing-your-accounts

[20] Ethereum. Contract tutorial [OL]. [2020-02-13]. https://github.com/ethereum/go-ethereum/wiki/contract-tutorial

[21] Ethereum. Ropsten testnet PoW chain [OL]. [2020-02-13]. https://github.com/ethereum/ropsten

[22] Parity. The fastest and most advanced Ethereum client [OL]. [2020-02-13]. https://wiki.parity.io/Parity-Ethereum

[23] Google. Google cloud platform [OL]. [2020-05-18]. https://console.cloud.google.com/

[24] SpankChain. We got spanked: What we know so far [OL]. [2020-02-13]. https://medium.com/spankchain/we-got-spanked-what-we-know-so-far-d5ed3a0f38fe

[25] Etherscan. Ropsten blocks [OL]. [2020-02-13]. https://ropsten.etherscan.io/blocks

[26] Ferreira T C, Baden M, Norvill R, et al. ÆGIS: Smart shielding of smart contracts [C] //Proc of the 2019 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2019: 2589-2591

[27] Wang Xinming, He Jiahao, Xie Zhijian, et al. ContractGuard: Defend Ethereum smart contracts with embedded intrusion detection [J]. IEEE Transactions on Services Computing, 2019, 13(2): 314-328

[28] Brent L, Jurisevic A, Kong M, et al. Vandal: A scalable security analysis framework for smart contracts [J]. arXiv preprint, arXiv:1809.03981, 2018

[29] Feist J, Grieco G, Groce A. Slither: A static analysis framework for smart contracts [C] //Proc of the 2nd Int Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). Piscataway, NJ: IEEE, 2019: 8-15

[30] Tikhomirov S, Voskresenskaya E, Ivanitskiy I, et al. SmartCheck: Static analysis of ethereum smart contracts [C] //Proc of the 1st Int Workshop on Emerging Trends in Software Engineering for Blockchain. Piscataway, NJ: IEEE 2018: 9-16

[31] Krupp J, Rossow C. teEther: Gnawing at Ethereum to automatically exploit smart contracts [C] //Proc of the 27th USENIX Security Symp (USENIX Security'18). Berkeley, CA: USENIX Association, 2018: 1317-1333

[32] Mossberg M, Manzano F, Hennenfent E, et al. Manticore: A user-friendly symbolic execution framework for binaries and smart contracts [C] //Proc of the 34th IEEE/ACM Int Conf on Automated Software Engineering (ASE). Piscataway, NJ: IEEE, 2019: 1186-1189

[33] Kalra S, Goel S, Dhawan M, et al. ZEUS: Analyzing safety of smart contracts [C] //Proc of the NDSS. Piscataway, NJ: IEEE, 2018: 1-12

[34] Frank J, Aschermann C, Holz T. ETHBMC: A bounded model checker for smart contracts [C/OL] //Proc of the 29th USENIX Security Symp (USENIX Security 20). Berkeley, CA: USENIX Association, 2020 [2020-06-09]. https://www.usenix.org/conference/usenixsecurity20/presentation/frank

[35] Jiang Bo, Liu Ye, Chan W K. ContractFuzzer: Fuzzing smart contracts for vulnerability detection [C] //Proc of the 33rd ACM/IEEE Int Conf on Automated Software Engineering. New York: ACM, 2018: 259-269

[36] Torres C F, Steichen M. The art of the scam: Demystifying honeypots in Ethereum smart contracts [C] //Proc of the 28th USENIX Security Symp (USENIX Security'19). Berkeley, CA: USENIX Association, 2019: 1591-1607

[37] O'Connor R. Simplicity: A new language for blockchains [C] //Proc of the 2017 Workshop on Programming Languages and Analysis for Security. New York: ACM, 2017: 107-120

[38] Schrans F，Eisenbach S，Drossopoulou S. Writing safe smart contracts in Flint [C] //Proc of the 2nd Int Conf on Art，Science，and Engineering of Programming. New York：ACM，2018：218-219

Xiang Jie，born in 1994. Master. His main research interests include Android security and blockchain security.

向　杰，1994 年生.硕士.主要研究方向为安卓安全和区块链安全.

Yang Zhemin，born in 1984. PhD. Associate professor，master supervisor. His main research interests include operation system，system security，program analysis and runtime environment technology.

杨哲慜，1984 年生.博士，副教授，硕士生导师.主要研究方向为操作系统、系统安全、程序分析和运行环境技术.

Zhou Shunfan，born in 1992. PhD candidate. His main research interests include system software，binary security，blockchain security. (sfzhou17@fudan.edu.cn)

周顺帆，1992 年生.博士研究生.主要研究方向为系统软件、二进制安全、区块链安全.

Yang Min，born in 1979. PhD，professor，PhD supervisor. Senior member of CCF. His main research interests include network security，vulnerability exploiting and analysis，AI security，blockchain security and system security mechanism. (m_yang@fudan.edu.cn)

杨　珉，1979 年生.博士，教授，博士生导师，CCF 高级会员.主要研究方向为网络安全、漏洞挖掘和分析、人工智能安全、区块链安全和系统安全机制.