L-015

Solidity コンパイラ更新による脆弱性への影響調査

加道ちひろ† 矢内 直人† 山下 恭佑† JasonPaul Cruz† 岡村 真吾††

† 大阪大学 〒565–0871 大阪府吹田市山田丘 1–1

†† 奈良工業高等専門学校 〒639–1080 奈良県大和郡山市矢田町 22

1. はじめに

近年，分散アプリケーションのプラットフォームであるスマートコントラクトが注目されている．ここでいうスマートコントラクトとは，暗号通貨の取引に加え，実行コードをブロックチェーン上へ保存することで自動的に取引を行うプログラムのプラットフォームである，その最も一般的なものが，Ethereum [1]．(これ以降，Ethereum スマートコントラクトをスマートコントラクト，スマートコントラクトにおけるプログラムをコントラクトと呼称する.)

スマートコントラクトは誰でもコントラクトの閲覧が可能であり，攻撃者による解析も容易である．さらに，一度デプロイされたコントラクトはその後変更することができない．このような点から，スマートコントラクトでは様々なセキュリティに関する事件が発生している．これらの被害を減らすために，多くの脆弱性解析ツールが開発されており [2], [3]，Solidity のコンパイラでは脆弱性を除外するような機能の追加などが更新によって行われている [4].

Solidity は 2015 年に誕生した若い言語で，深刻な脆弱性に晒されている可能性があるため，コンパイラ更新による脆弱性対策の有効性を明らかにすることは重要である．これを考慮し，本稿では **Solidity のコンパイラが各脆弱性にどのように影響しているか**という問いを明らかにする．そのために，脆弱なコントラクトの数が，実際にデプロイされたコントラクトにおいてどのように変化しているのかについて調査を行う．

本稿では，2022/8/31 までにデプロイされた 503,572 件のコントラクトを収集し，重要度の高い脆弱性，*Locked Money*，*Using tx.origin*，*Unchecked Call* について解析を行った．それにより，**Solidity コンパイラのメジャー更新が脆弱性の減少に効果がある**ことを示した上で，次の 3 つの知見を得た．1) *Locked Money* は v0.6 以降で大幅に減少している．2) *Using tx.origin* は初めから数が少なく，コンパイラ更新による影響はごく僅かである．3) *Unchecked Call* は v0.8 で減少しているが，偽陽性や脆弱なコードのクローンによって，最新のコンパイラにおいても多く残っている．

本稿のフルバージョンは arXiv に掲載済みである [5].

2. 背景

本章では，Ethereum スマートコントラクトの技術的背景と，本研究におけるその関連研究について述べる．

2.1 Ethereum スマートコントラクト

Ethereum では，Solidity [6] などの高級言語によってスマートコントラクトのソースコードが実装される．ブロックチェーン上にデプロイされたコントラクトは，ネットワークのピアと呼ばれるノードによって管理される．スマートコントラクトには，識別子であるコントラクトアドレスが割り振られ，それを用いて Ethereum の通貨である Ether の受け取りや関数の実行が行われる．コントラクトはブロックチェーンに記録されるトランザクションの宛先として，そのコントラクトアドレスが指定された場合に呼び出され，トランザクションに含まれる関数や引数などの情報を元にピアが実行環境 Ethereum Virtual Machine 上でコントラクトを実行する．また，ピアによる実行に対する動機付けとして，Ethereum には Ether によって支払われる**ガス**と呼ばれる実行手数料が導入されている．

スマートコントラクトの脆弱性は，The DAO のように甚大な金銭的被害を発生せる場合がある．そのため，様々な脆弱性への対策が研究されており，特に脆弱なコードのコンパイルを防ぐようコンパイラの更新が行われている．

2.2 関連研究

実存するコントラクトのうち 98.14% が，既知の脆弱性へのパッチが当てられていないことを示したもの [4] や，脆弱なコントラクトによる金銭的被害の調査 [7] が行われている．本稿はこれらの研究と組み合わせ，脆弱性による被害額を各コンパイラごとに推測することが可能である．また，脆弱性解析ツールの開発状況に関する研究 [8], [9] も行われている．スマートコントラクト開発者の使用するコンパイラについて調査を行った研究 [10] では，半分の開発者が最新バージョンを使用していることが示されている．本稿とこれらの研究を照らし合わせることで，今後対策すべき脆弱性を明らかにすることが可能であると考える．

脆弱性解析ツールは様々開発されているが，これに関する実態調査 [8], [9] から SmartCheck [11] や Mythril [3] が高い精度を出すことが示されている．また，SmartCheck は解析に必要な時間が短いため，本稿では本ツールを用いて調査を行う．

3. 調査手法

調査では，Solidity で記述されたソースコードを対象とする静的解析ツール，SmartCheck [1] [11] を用いて脆弱性解析を行う．本稿における脆弱性の有無は，このツールに従う．これを用いて，重要度が高い *Locked Money*，*Using tx.origin*，*Unchecked Call* の 3 つの脆弱性を解析する．

またコントラクトの収集には，Ethereum ブロックチェーンに記録された情報を提供する Etherscan [2] を使用し，2022/8/31 までにデプロイされた全てのコントラクトを収集する．ただし，v0.1 から v0.3 のコンパイラはほとんど使用されていないため，v0.4 以降のコンパイラを用いたコントラクトに関して，Solidity で記述されたソースコードが公開されている 503,572 件のコントラクトを解析している．

最後に，結果に関して脆弱性発生率という指標を用いる．脆弱性発生率は，作成されたコントラクトの数に対する，脆弱なコントラクトの数の割合と定義する．作成されたコ

1 https://github.com/smartdec/smartcheck
2 https://etherscan.io/

表 1　各コンパイラバージョンごとの脆弱性発生率

Version	*Locked Money*	*Using tx.origin*	*Unchecked Call*
v0.8	6.86% (9120)	1.87% (2491)	22.04% (29297)
v0.6	6.66% (3624)	2.97% (1614)	49.10% (21271)
v0.5	33.60% (28017)	1.84% (1538)	32.90% (27434)

ントラクトは，Solidity コードが公開されているコントラクトであり，脆弱なコントラクトは SmartCheck によって脆弱性が検知されたものとする．これらの手法による脆弱性発生率の減少が，実際に発生している脆弱性が減少していることを表す．

4. 調査結果と考察

コンパイラ更新による脆弱性への影響についての調査結果を示し，議論を行う．ただし，更新による機能変更や更新期間などを踏まえて，挙動が不安定だと考えられる v0.4 と v0.7 を除いた結果を示す．

表 1 は，各コンパイラのバージョンごとに各脆弱性の脆弱性を示している．結果としては，Solidity コンパイラの更新に伴い脆弱性が減少傾向にあるといえる．具体的に，各脆弱性の変化を確認する．まず，*Unchecked Call* の v0.6 への更新に伴う増加は，例外処理を行うが直接的にはこの脆弱性への対策として機能しない `try-catch` 構文についての開発者の勘違いが要因の 1 つであると考えている．さらに，*Locked Money* は v0.6 への更新時に，*Unchecked Call* は v0.8 への更新時に脆弱性発生率が減少している．これについてより詳細に確認するため，*Locked Money* では v0.6 リリース以降，*Unchecked Call* では v0.8 リリース以降における脆弱性発生率を確認する．その結果，同様の減少傾向が見られることから，これらの減少はコンパイラの影響であると考えられる．しかし，その詳細な要因については今後調査が必要である．*Using tx.origin* については，全てのバージョンで脆弱性発生率が 5% 以下であることから，コンパイラによる影響を確認することは難しい．

また，マイナーアップデートについても調査を行ったものの，次のバージョン登場までの期間や用いられた数が大きく異なり，その影響を確認することはできていない．

またここから，最新の v0.8 において，*Unchecked Call* が他二つの脆弱性と比較して多く残っている点について考える．偽陽性とコードクローンがこの要因として考えられることを発見した．まず偽陽性については，複数のコントラクトで使用されているライブラリに対して，*Unchecked Call* が検出されていた．このライブラリは，安全なコードを提供しているとして知られる OpenZeppelin [3]のライブラリであり，そのライブラリについて詳細に確認したところ，その検知が SmartCheck による偽陽性であることを確認した．

さらにコードクローンによる脆弱性への影響について考える．コードクローンが要因の可能性の一つであることを発見したため，最後の半年間におけるクローンを確認した．ここでいうクローンは，全く同一のコードか，あるいは行の追加が行われているものを指す．最後の半年間において，クローンを除外し，オリジナルのコントラクトのみを対象として脆弱性発生率を確認した．その結果，v0.8 において，*Unchecked Call* と *Locked Money* の 2 つで，クローンの除外による脆弱性発生率の減少が確認できた．従って，*Unchecked Call* のみで脆弱性発生率が高い理由とはならないものの，間違いなくコードクローンによって脆弱性発生率が高くなっていると言える．また，古いバージョンの方がクローンが多く存在した．このことから，開発者が古いバージョンのコンパイラを使用する理由として，クローンを作成している可能性が挙げられる．

5. 結　　論

本稿では，2022/8/31 までに v0.4 以降のコンパイラを用いて作成された 503,572 件のコントラクトを収集し，重要度の高い脆弱性 [11] を SmartCheck により解析した．その結果，Solidity コンパイラの更新に伴いこれらの脆弱性が減少していること，また，*Locked Money* は v0.6 で急減，*Using tx.origin* は初めから少数，*Unchekced Call* は最新 v0.8 でも多く残っていることを確認した．ただし，偽陽性やコードクローンがその一因である可能性を示した．

また今後の課題として，より長い期間でのコードクローンの確認，偽陽性/偽陰性，開発者のスキルなどを考慮した調査が必要であると考えている．

謝辞 本研究の成果の一部は JST CREST（課題番号：JPMJCR21M5）の支援を受けたものである．

文　　献

[1] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger byzantium version,” `https://ethereum.github.io/yellowpaper/paper.pdf`, 2022.

[2] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making smart contracts smarter,” Proc. of CCS 2016, pp.254–269, ACM, 2016.

[3] B. Mueller, “Smashing ethereum smart contracts for fun and real profit,” 9th HITB Security Conference, pp.1–54, 2018.

[4] S. Hwang and S. Ryu, “Gap between theory and practice: An empirical study of security patches in solidity,” Proc. of ICSE 2020, pp.542–553, ACM, 2020.

[5] C. Kado, N. Yanai, J.P. Cruz, K. Yamashita, and S. Okamura, “An empirical study of impact of solidity compiler updates on vulnerabilities in ethereum smart contracts,” CoRR, vol.abs/2306.04250, , 2023. https://doi.org/10.48550/arXiv.2306.04250

[6] “Solidity v0.8.0 breaking changes,” `https://docs.soliditylang.org/en/latest/080-breaking-changes.html`.

[7] D. Perez and B. Livshits, “Smart contract vulnerabilities: Vulnerable does not imply exploited,” Proc. of USENIX Security 2021, pp.1325–1341, USENIX Association, 2021.

[8] S.S. Kushwaha, S. Joshi, D. Singh, M. Kaur, and H.-N. Lee, “Systematic review of security vulnerabilities in ethereum blockchain smart contract,” IEEE Access, vol.10, pp.6605–6621, 2022.

[9] T. Durieux, J.F. Ferreira, R. Abreu, and P. Cruz, “Empirical review of automated analysis tools on 47,587 ethereum smart contracts,” Proc. of ICSE 2020, pp.530–541, 2020.

[10] Z. Wan, X. Xia, D. Lo, J. Chen, X. Luo, and X. Yang, “Smart contract security: A practitioners’ perspective,” Proc. of ICSE 2021, pp.1410–1422, IEEE/ACM, 2021.

[11] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, “Smartcheck: Static analysis of ethereum smart contracts,” Proc. of WETSEB 2018, pp.9–16, ACM, 2018.

3 `https://github.com/OpenZeppelin/`