URL https://journal.ntt.co.jp/article/39532
DOI https://doi.org/10.60249/26061301

NTTソフトウェアイノベーションセンタ
特別研究員
須田瑛大 Akihiro Suda

OSSサプライチェーンの信頼性回復に向けたセキュリティ対策

2024年に発覚した，OSS「liblzma」（ファイル圧縮ライブラリ）のバックドア事件は，OSの基幹ライブラリすら標的となることを示し，OSSサプライチェーンの信頼性を根本から揺るがしました．幸いにしてこの事件は，世界中にある無数のシステムに導入される前に発見・対処されましたが，このような深刻な事態を受け，サイバーセキュリティのさらなる強化が急務となっています．今回は「OSSサプライチェーンのセキュリティ強化」のトップランナー，須田瑛大特別研究員に話を伺いました．

◆PROFILE：2012年京都大学工学部情報学科卒業．2014年京都大学大学院情報学研究科通信情報システム専攻修士課程修了．同年，日本電信電話株式会社入社．基盤分野でのオープンソースソフトウェア（OSS）の研究開発・普及に従事．Docker（Moby）やCloud Native Computing Foundation（CNCF）のcontainerd，Limaなどのメンテナ（開発委員）を務める．主な受賞歴として，CNCF Top Committer Award（2023），Google Open Source Peer Bonus Award（2023）など．

安全性向上のためのOSSセキュリティ強化の必要性

■具体的にどのような研究に取り組まれているのですか．

ソフトウェアの多くは，不特定の開発者によって開発・公開されているOSS（オープンソースソフトウェア）に多くを依存しています（**図1**）．開発会社がプログラムを作成するにあたって，独自の機能に関しては当然自社で開発をしますが，それ以外の汎用の機能についてはさまざまなOSSライブラリを組み合わせているのが実情です．このOSSの価値は8.8兆ドル[(1)]に相当するといわれており，OSSを一切使用せずに汎用機能のプログラムもわざわざ自社で実装することは非効率です．そもそも，自分自身ではOSSを採用しているつもりがなくても，WindowsやmacOSなど非オープンソースのOS（オペレーティングシステム）やア

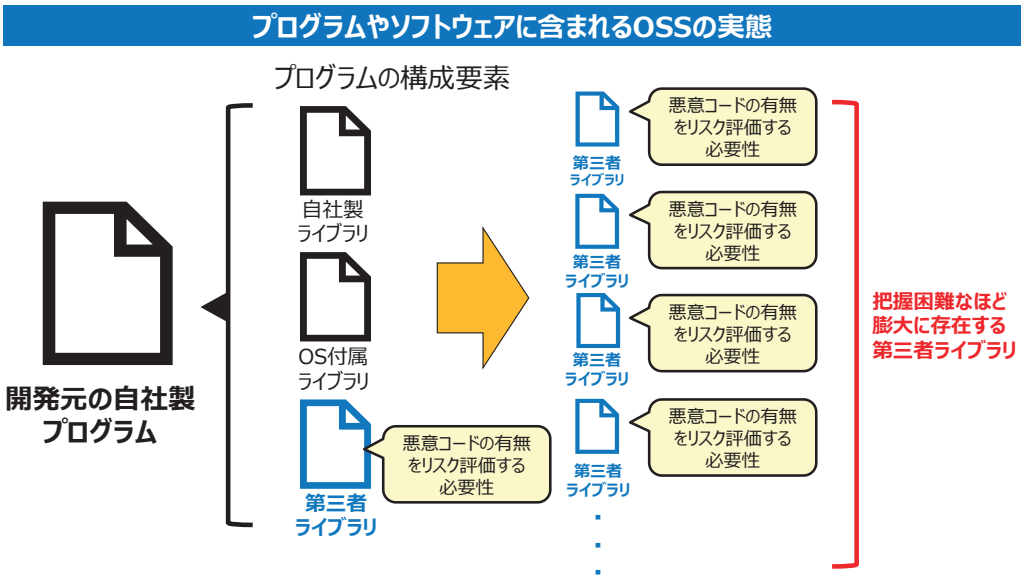

図1　プログラムを構成する多数のライブラリとOSS

プリケーションにも多数のOSSライブラリが含まれており，OSSからは逃れようがありません．

ここで問題となるのが，このOSSには悪意のあるソースコードが仕込まれていることがあり得るという点です．実際に2024年2月には，Linux系OSなどに含まれる「liblzma」というOSSライブラリに，開発者（途中から参加した開発者であり原作者ではない）自身の悪意によるバックドアが仕込まれていたことが発覚しました．こうしたOSSライブラリについてはすべてを緻密にリスク評価することが本来は望ましいのですが，1つのプログラムが数百ものライブラリに依存することも多く，現実的にその1つひとつに対してリスク評価を行うのは難しいといわざるを得ません．1つのOSSに含まれるソースコードは，それだけでかなりの分量におよびます．ソースコードとはプログラムの「設計図」ともいわれるテキストファイルのことです．ソースコードがどのような動作を記述したものか，ましてや悪意のあるコードなのか否かの判別は容易ではありません．特に近年は生成AI（人工知能）の普及によって，依存ライブラリの選定やコード生成の自動化に伴い，ソースコードの量が爆発的に増加しているため，習熟した開発者にさえ判別が難しくなっています．OSSサプライチェーンの安全性を確保することは，従来にも増して重要になっているといえます．

では，この悪意あるコードからどのようにコンピュータを守るのか，その具体的な方法論，セキュリティ対策が私の研究の1つです．

■サンドボックス方式の3層構造について教えてください．

図1で示したとおり，1つのプログラムには多数のOSSが使用されており，その1つひとつのソースコードをリスク評価することは理屈のうえでは不可能ではないものの現実的ではありません．そこで私はOSSの技術的・社会的特性を分析してリスク評価を可能にし，攻撃を未然に防ぐ手法や，プログラムをあらかじめサンドボックス*1に閉じ込めて動作させることで，攻撃を防げなかった場合の被害を軽減する手法を考案しています．これは，次のようなライブラリサンドボックス，コンテナ，VM（仮想マシン）の3層での方法となります（**図2**）．

(1)　ライブラリサンドボックス

ライブラリとはプログラムの機能部品のことで，活用するとプログラムの機能を再実装しなくて済み，開発コストを削減できるメリットがあります．しかし，ライブラリにはバックドアが含まれているおそれがあります．本研究ではライブラリをサンドボックスに閉じ込めてシステムコールの呼び出しを制限し，意図しないファイルアクセスやネットワークアクセス，コマンド実行が試みられた場合，そのシステムコールの実行を禁止することで，バックドアが仕込まれていた場合の被害を軽減する手法の確立とその普及展開をめざしています．後述するコンテナとも類似していますが，コンテナよりも細かい粒度でのサンドボックス化により，従来は対処できなかった攻撃にも対応できるようになっています．また，ライブラリのシステムコール呼び出し動作を解析するだけではなく，誰によって開発されたりレビューされたりしているかといった社会的特性も解析することで，ライブラリの採用を決定する前にリスクを評価することも可能にしています．ライブラリのエコシステムはプログラミング言語によって大きく異なり，そのすべてに対応することは困難ですが，クラウドネイティヴ分野ではGo言語*2が広く使われていることから，まずはGoのライブラリを対象に，「gomodjail」というサンドボックス機構を提唱しています（**図3**）．将来的にはほかの言語のライブラリにも

＊1　サンドボックス：プログラムを保護された領域で動作させることによって，システムが不正に操作されるのを防ぐセキュリティ機構のこと．
＊2　Go言語：Googleが2009年に発表したプログラミング言語．Webサービスの開発や，クラウドネイティヴ領域などで広く利用されています．DockerやKubernetesで採用されていることで知られています．

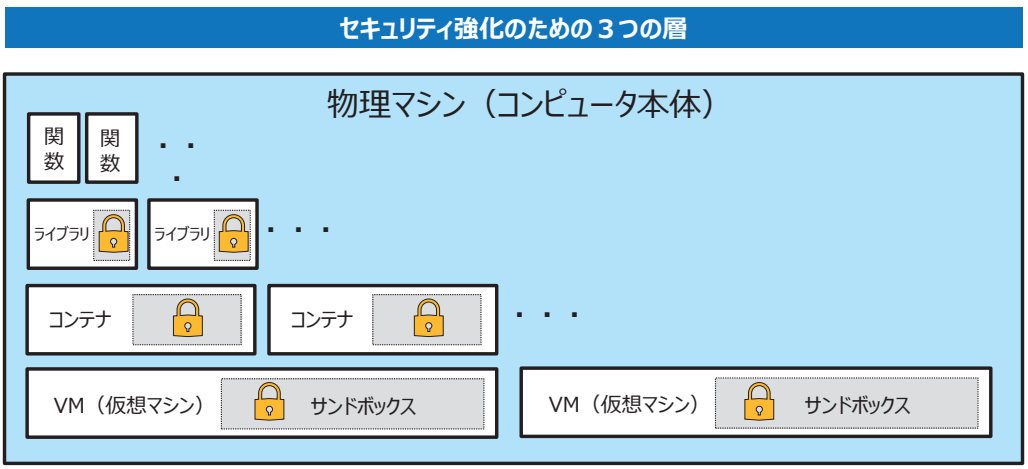

図2　3層構造とサンドボックスを活用したセキュリティ

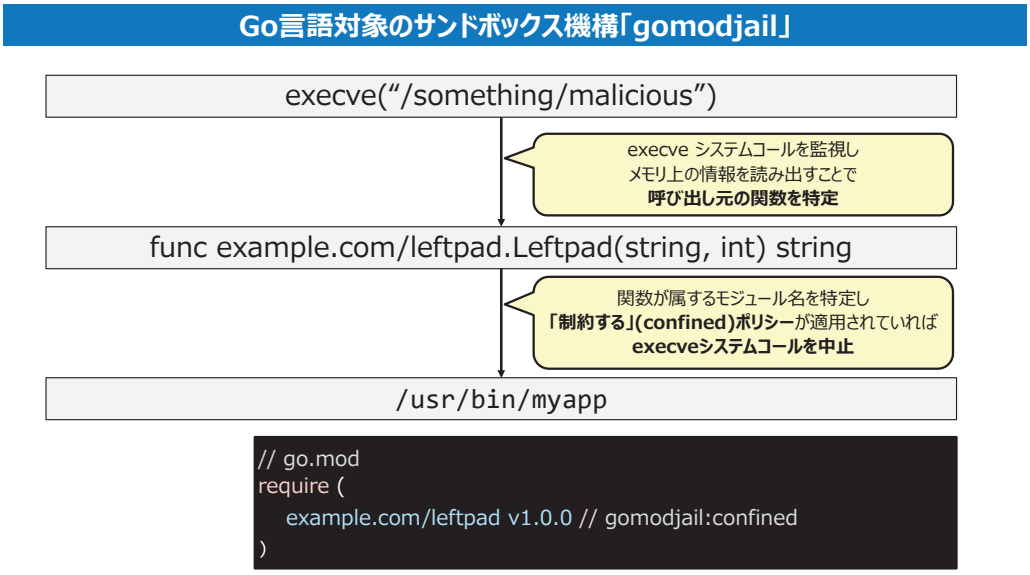

図 3　サンドボックス機構「gomodjail」の仕組み

対応することをめざしています．

⑵　コンテナ

コンテナとは，ファイルシステムやプロセスツリーなどのリソースの一部を切り出して，OS本体とは別の実行環境をつくり出す技術のことです．プログラムをコンテナに閉じ込めることで，プログラム間での不正な干渉を防ぐことができるなどの利点があります．後述するVMに似ていますが，VMよりも処理速度では優れ，セキュリティでは劣るという特徴があります．

⑶　VM

VM とは，1台のコンピュータを仮想的に複数台に見せかける技術のことです．コンテナに処理速度では劣るものの，セキュリティでは優ります．VMとコンテナは必ずしも競合するものではなく，よく組み合わせて使われます．VMを使う場合は異なる種類のOSを同時に実行できるなどの利点もあります．

このように，ライブラリ，コンテナ，VMと3層に渡ってサンドボックス方式によるセキュリティ対策を行うことで，より安全性を高めることが可能になります．コンテナやVMについてはすでに普及していますが，特にライブラリにもサンドボックス方式でのセキュリティ対策を適用しようというアイデアは，NTTが先行している試みであり，まだこれからという段階です．

■今までの研究の成果について教えてください．

私はもっとも有名なコンテナ実装である「Docker」（別名Moby）の開発に早期から参加しており，2016年よりそのメンテナ（開発委員）を務めています．Dockerに関連するプロジェクトである「BuildKit（ビルドキット）」「containerd（コンテナディー）」「runc（ランシー」「Open Container Initiative（OCI）Runtime Spec」などのメンテナも務めています．特に大きな貢献としては，root権限（管理者権限）なしでコンテナを安全に実行できる「Rootlessモード」を，これらのプロジェクトに実装したことです．「Rootlessモード」を用いると，仮にコンテナランタイムに脆弱性があって悪意あるプロセスがコンテナから脱出することがあっても，その被害を軽減することができます（**図4**）．

また，containerdを活用したDocker互換プロジェクトとしてnerdctl（contaiNERD ConTroL：ナードシーティーエル）を立ち上げました．OSSとしてのDockerが一時停滞しており，containerd側で進んでいたセキュリティや性能面での改善を取り込みにくい状態が続いていたことが新しい互換プロジェクトを立ち上げた理由です．OSSとしてのDockerは現在では活気を取り戻しています．

そのほかにも，VMを便利に使うためのツールである Lima（Linux Machine）というプロジェクトも立ち上げています（**図5**）．Limaは元々，containerdおよびnerdctlを含むLinux環境をMac上で簡単にデモする目的で作成したツールでしたが，人気を博してユーザコミュニティが拡大したことから，今では幅広いユースケースとOSに対応しています．特に最近はAIコーディングエージェント[*3]を安全に実行するためのツールとしても注目されつつあります．AIがファイルを変更しようとしてもユーザが確認・承認するまでは実際のファイルに反映されないので，仮にAIが暴走したとしても安心です．また，確認・承認作業はファイル

＊3 AIコーディングエージェント：自然言語の指示に基づいてコードの生成，実行，テスト，デバッグまでを自律的に行うAIツール．従来のAIコーディングアシスタントとは異なり，複雑なタスクを分解し，ファイルシステムの操作やコマンド実行なども行うことができます．

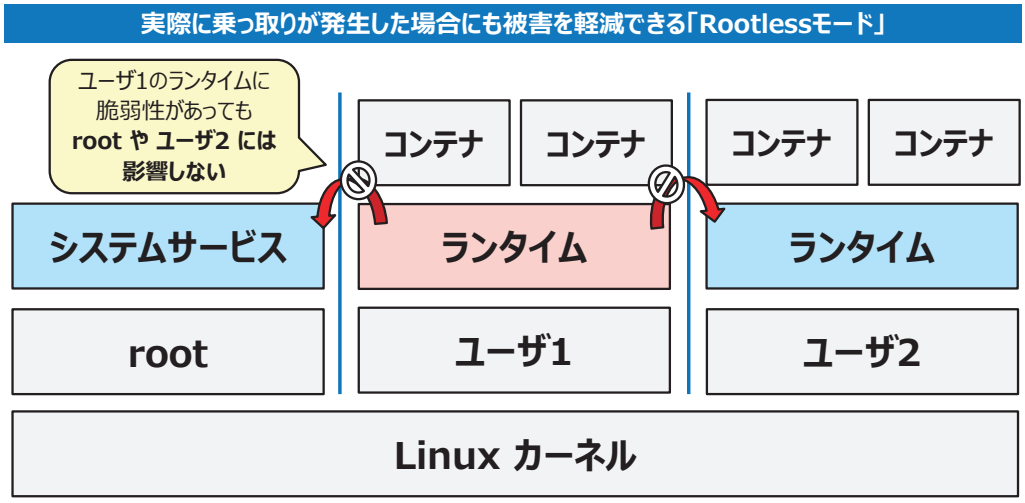

図 4　「Rootless モード」の概念図

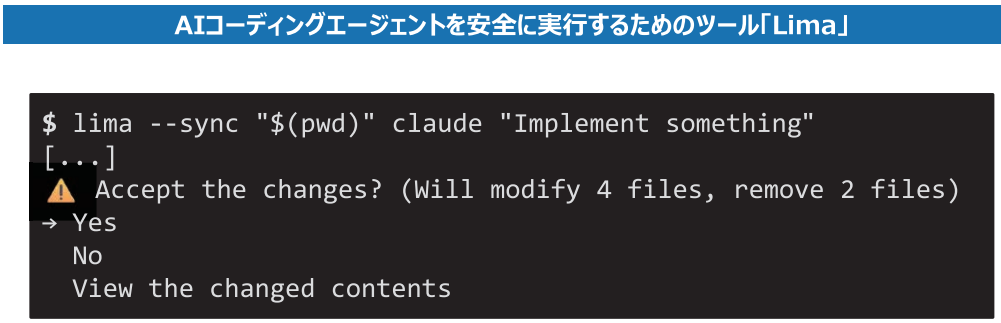

図 5　Lima の動作例

変更のたびに必要なのではなく，AI終了時にまとめて 1 回だけ実施すればよいので煩わしさもありません．

このLima プロジェクトではGitHub[*4]のstar（いわゆる「いいね」）数が約 2 万件に，開発者数が約180名に達しました．nerdctl プロジェクトはGitHub star数が約 1 万件，開発者数が約200名です．Lima，nerdctlのいずれも，NTT発のOSSとしては前例のない規模となっています．NTTグループ内で利用されるにとどまらず，AWS（Finch）やSUSE（Rancher Desktop）など他社の製品にも組み入れられており，世の中で広く使われています．

■研究で苦労された点や今後の課題について教えてください．

OSS活動全般に共通する苦労として，提案活動の難しさがあります．提案に対して反対されるのは苦ではありませんが，何の反応もないときはもどかしく感じます．わずか数行のソースコードを変更するだけの提案でも，それが承認され，リリースされるまでには数カ月や数年かかることも珍しくありません．

また，メンテナとしての私は提案をする側であると同時に，大量の提案を受ける側でもあり，ほかの開発者の提案に迅速に反応できていないときは申し訳なく感じています．正常に機能しているプログラムの改善提案には誰しも慎重にならざるを得ないのは理解できます．OSSの質を担保しつつメンテナの負担を軽減するためには，今後のコミュニティの構造的な変革が必要だと感じています．

OSSのセキュリティのさらなる強化で，高次元の安全性を確保する

■この研究によって実現できることや将来の応用先，今後の取り組みも教えてください．

リスクが高くても有用なOSSを，そのリスクを低減させたうえで安心して活用できるようになります．応用先としてはAIコーディングエージェントを想定しています．今後，数年ないし数カ月の

＊4 GitHub：開発者がソースコードを保存，管理，共有できるようにするプラットフォーム．

うちには，AIが生成したコードを人手でレビューせずに実行することが当たり前になると思われます．AIがハルシネーションを起こしたり不正なWeb検索結果に騙されたりして有害なコードを生成したとしても，サンドボックス化により被害を抑えることができます．

今後の具体的な取り組みとしては，2026年はライブラリサンドボックスの設計を大幅に見直し，プログラムのコンパイル時に静的解析やソース変換を行うようにすることを計画しています．プログラム実行時のオーバヘッドを除去し実用性を高めることで，広範なOSSへの普及を図ります．また，2027年以降は，Go言語に限らず多様なプログラミング言語についてライブラリサンドボックスを実現することをめざします．

コンテナやVMについては，AIコーディングエージェントとの連携を強化したり，脆弱性の影響を軽減する技術を創出したりすることをめざしています．ほかにもOSSのセキュリティリスクの評価や軽減に資する技術をさらに模索していきます．

また，NTTのIOWN（Innovative Optical and Wireless Network）構想への応用もめざします．IOWN構想は170以上のIOWN Global Forum加盟企業・団体を含む多様なベンダのハードウェアやソフトウェアに依存しており，その多くは直接的・間接的にOSSに依存しています．特にDCI（Data-Centric Infrastructure）では複数ベンダの製品を自在に組み合わせるため，依存するOSSの数は膨大となります．OSSサプライチェーン全体のセキュリティを底上げすることで，IOWNをより安全に普及展開することが可能になると期待しています．

それぞれの提案技術によるセキュリティの向上にあたっては，性能や使い勝手が犠牲にならないように気を付けています．例えば，前述のRootlessコンテナでは，当初はネットワーク性能（TCPスループット）が87％も低下する問題があったためAPN（All-Photonics Network）環境には到底適用できないものでしたが，のちに通常のコンテナと同等の性能を出せるようになりました．

■研究を進めるうえで大切にしていることがありましたら教えてください．

“Unusable security is not security” です．2015年の「DockerCon」（Docker社が主催する開発者向けイベント）会議で使われた言葉で，直訳すれば「使用できないセキュリティはセキュリティではない」という意味です．独りよがりにセキュリティ技術を創出しても，それが実際にユーザに使われなければ意味がありません．無闇にセキュリティ強化をめざすのではなく，ユーザの使い勝手への影響を考慮したうえで，無理のない範囲にとどめておくほうがかえって強いセキュリティを実現できるという意味だと，私はとらえています．ユーザが意識しなくても，自ずとセキュリティが向上していくような技術の創出を今後もめざしていきます．

■所属しているNTTソフトウェアイノベーションセンタについて教えてください．

私の所属するNTTソフトウェアイノベーションセンタ（SIC）は，IOWNやtsuzumi（大規模言語モデル）などの新サービスを支えるコンピューティング基盤について，基礎的な研究から実用化開発まで通貫して担っている組織です．SICは単に技術だけを研究しているわけではありません．OSSコミュニティや外部ベンダと連携することで，社会と会社の双方に共通の価値を創造するプロセスを培う役割も果たしています．SICで培われている共通価値創造のノウハウは，ほかの組織にもきっと役立つことでしょう．

■最後に研究者，学生へメッセージをお願いします．

OSSの研究活動においては，技術力ももちろん重要ですが，交渉力がもっとも重要というのが実情です．動作するコードが手元にあっても，それを採用するように開発者を説得できなければ，世の中で使われるものとはなりません．特にここ1～2年のAIコーディングエージェントの発達によって，コントリビュータの技術的な熟練度を評価することが著しく困難になりつつありますから，今後のOSSコミュニティでは技術以外の面でもコントリビュータを評価する傾向が強まると思われます．OSSに興味がある方は，こうしたコミュニケーション能力や営業力といった部分も，気にかけておくほうが有利になると思います．私たちNTTも大きく変わりつつあるOSSコミュニティの中で，どう立ち振る舞っていけば影響力を高められるかといった検討を進めていきますので，お互いに精進していきましょう．

■参考文献

(1) M. Hoffmann, F. Nagle, and Y. Zhou : “The Value of Open Source Software,” Harvard Business School Working Paper, 2024.

（今回はリモートにてインタビューを実施しました）