特集論文

組込みシステム向けセキュリティソフトウェア

牧村悠司*
Yuji Makimura

Security Software for Embedded Systems

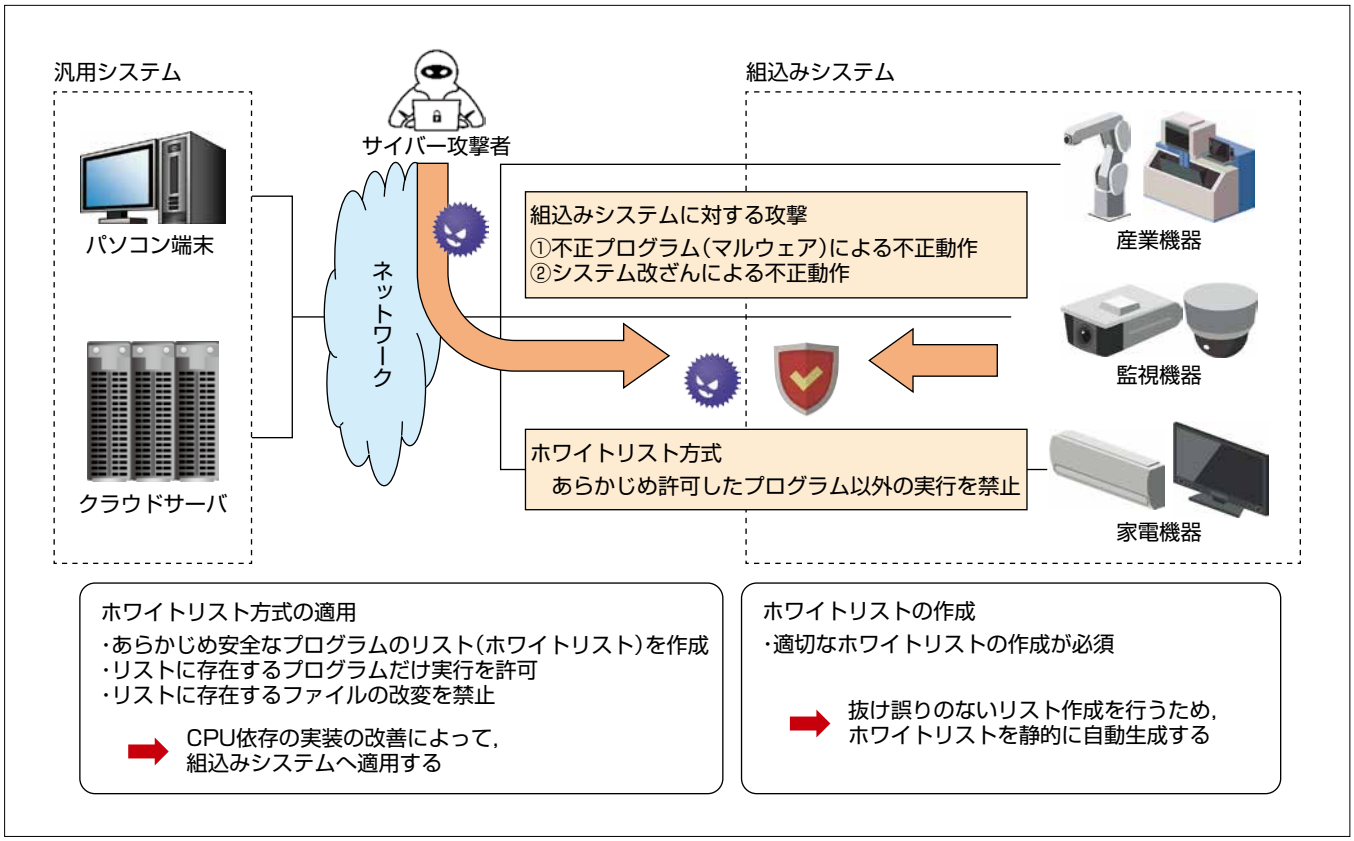
要　旨

近年，IoT(Internet of Things)の普及に伴い，ネットワークには汎用システム(パソコン端末等)に加えて，多数の組込みシステム(IoT機器等)が接続されるようになった。今後5Gの導入によって組込みシステムの接続数は更に増加することが予想される。一方，組込みシステムを対象にした不正プログラム(マルウェア)攻撃の事例も増加しており，セキュリティ対策の必要性が高まっている。

不正プログラムに対するセキュリティ対策としては，セキュリティソフトウェアによる不正プログラムの実行制限が一般的である。実行制限手法には，不正プログラムの実行を禁止するブラックリスト方式と，あらかじめ許可したプログラム以外の実行を禁止するホワイトリスト方式がある。組込みシステムでは実行するプログラムをあらかじめ規定可能であり，ホワイトリスト方式が適している。ホワイトリスト方式のセキュリティソフトウェアには市販の製品やOSS(Open Source Software)等が存在しているが，多様な組込み向けCPU環境で実行できないという課題や，抜け誤りのないホワイトリストの作成が困難といった課題がある。

これらの課題の解決に向けて，三菱電機ではCPU環境に依存せず実行可能であり，抜け誤りのないホワイトリストを静的に自動生成する組込みシステム向けセキュリティソフトウェアを開発した。

当社で開発する組込みシステムにこのセキュリティソフトウェアを搭載することで，IoTシステムがもたらす恩恵を安心・安全に享受できる社会の実現を目指す。



組込みシステム向けセキュリティ対策

組込みシステムに対する攻撃には，不正プログラムによって不正な動作を実行させるものと，システムを改ざんして不正な動作を実行させるものがある。これらへの対策として組込みシステムにホワイトリスト方式セキュリティソフトウェアを導入することが有効である。ホワイトリスト方式のセキュリティソフトウェアでは，抜け誤りのない適切なホワイトリストを作成する必要があるが，人手作成では抜け誤りのリスクがある。ホワイトリストを自動生成することで，ホワイトリストでの抜け誤りを防ぐ必要がある。

*コミュニケーション・ネットワーク製作所

1. まえがき

近年，IoTの普及に伴い，汎用システム(パソコン端末等)だけでなく，組込みシステム(IoT機器等)がネットワークに接続されるようになり，その数は2020年には300億個を超えることが予想されている。今後5Gの導入によって，通信速度，同時接続数，遅延時間が大幅に向上し，ネットワークに接続される組込みシステムの数は爆発的に増加することが予想される[1]。

ネットワークに接続される組込みシステムが増加するにつれて，組込みシステムを標的にしたサイバー攻撃事例も増加している。2016年には14万５千台以上の組込みシステムが不正プログラムMiraiに感染し，大規模な分散型サービス妨害(Distributed Denial of Service：DDoS)が発生している[2]。このようなサイバー攻撃事例から組込みシステムでも，不正プログラムに対するセキュリティ対策の必要性が高まっている。

当社では，CPU環境に依存せず実行可能であり，抜け誤りのないホワイトリストを静的に自動生成する組込みシステム向けセキュリティソフトウェアを開発した。

本稿では，**２章**でセキュリティソフトウェアについて，**３章**で組込みシステム向けセキュリティソフトウェアの開発について述べ，**４章**でホワイトリスト自動生成手法について述べる。

2. セキュリティソフトウェア

Miraiのような不正プログラムによる攻撃を防ぐ対策として，①不正アクセスを防ぐこと，②不正なプログラムを実行させないことの２点が挙げられる。①の対策として，パスワード等を用いたアクセス認証が一般的に利用されるが，組込みシステムは機器数が多いことから，パスワード等のアクセス権管理が十分でない場合が多い。また，管理者が定期的にアクセスすることなく長期間運用されることから，不正アクセスの検出も困難である。そのため，組込みシステムでは①の対策に加えて，②の対策を行うことが重要である。②の対策としてはセキュリティソフトウェアを用いた不正プログラムの実行禁止手法が一般的に用いられている。

2.1 不正プログラムの実行禁止手法

不正プログラムの実行禁止手法には，ブラックリスト方式とホワイトリスト方式がある。各方式の特徴を**表１**に示す。ブラックリスト方式は，不正プログラムのパターンをリスト化し，プログラム実行時にリストとのパターン照合を行うことで，パターンにマッチした不正プログラムの実行を禁止する方式である。この方式では，過去に蓄積されてきた膨大な不正プログラムパターンとの照合が必要になるため，不正プログラム判定の処理負荷が大きい。また，リストにないプログラムは全て実行可能になるという特徴がある。そのため，この方式は汎用システムのようにリソースに余裕があり，動作させるプログラムをあらかじめ規定しない場合に適している。一方，ホワイトリスト方式は，実行を許可するプログラムのリストを作成し，リストにあるプログラムだけ実行を許可する方式である。この方式では，実行するプログラムがリストに存在するか否かを判定するだけでよいため，処理負荷が小さい。また，不正プログラムかどうかに関わらず，リストにないプログラムは実行できないという特徴がある。そのため，ホワイトリスト方式を適用する場合，使用するプログラムを網羅した抜け誤りのないホワイトリストを適切に作成する必要がある。組込みシステムでは，実行するプログラムをあらかじめ規定することが可能であるため，ホワイトリスト方式の不正プログラム実行禁止手法が適していると考えられる。

2.2 既存ソフトウェアの組込みシステムへの適用可能性

組込みシステムへ適用可能なホワイトリスト方式のセキュリティソフトウェアとして，当社試作ソフトウェア，市販ソフトウェア，OSSの三つを検討した。各ソフトウェアの特性を**表２**に示す。

(1) 当社試作ソフトウェア

当社では，製品に適用するセキュリティ機能に対して保守性を確保するため，Linux(注1)サーバ向けにホワイトリスト方式のセキュリティソフトウェア[3]を試作している。このソフトウェアの実現方式はサーバ向けの汎用的なCPUアーキテクチャに依存しているので，組込みシステムのCPUアーキテクチャへそのまま適用することができない(移植性の課題)。また，このソフトウェアではホワイトリスト作成を容易にするため，実行したプログラムを学習して動的にホワイトリストへ登録する機能を持つが，プ

表１．不正プログラム実行禁止手法の特徴比較

特徴	ブラックリスト方式	ホワイトリスト方式
処理負荷	大	小
リスト定期更新	必要	不要
新種の脅威への耐性	なし	あり
未登録プログラムの実行	可能	不可

表２．セキュリティソフトウェアの特性比較

	保守性	移植性	ホワイトリスト完全性
当社試作ソフトウェア	○	×	△
市販ソフトウェア	×	×	○
OSS	×	○	×

ログラムの要否を人手で判定する必要があり，リストに抜け誤りが生じ得る(ホワイトリスト完全性の課題)。

⑵ 市販ソフトウェア

特定のOS及びCPU環境で動作し，組込みシステム向けのホワイトリスト方式セキュリティソフトウェアとして実績がある。しかし，特定のOS及びCPU環境でだけ動作を保証しており，対応していない組込みCPUへ移植することが困難である(移植性の課題)。また，製品がバイナリー形式で提供されるためソースコードの入手ができず，製品適用時の保守性の担保が困難である(保守性の課題)。

⑶ OSS

Linux向けホワイトリスト方式のセキュリティOSSとして，SELinux(Security-Enhanced Linux)[4]があり，汎用システム向け，組込みシステム向け，ともに実績がある。OSSであるためソースコードを容易に入手できる一方で，不具合発生時に開発元から十分なサポートを受けることが難しく，製品適用時に保守性の担保が困難である(保守性の課題)。また，SELinuxはリッチなアクセス制御が可能であるが，ホワイトリスト生成時にアクセス権を詳細に設定する必要があり，抜け誤りのないホワイトリストの生成が非常に困難である(ホワイトリスト完全性の課題)。

組込みシステムは長期間運用されるため，保守性に優れる当社試作ソフトウェアを採用し，当社試作ソフトウェアが持つ移植性とホワイトリスト完全性の課題を解決する方法を検討した。移植性の課題に対して，サーバ向けの当社試作ソフトウェアを組込みシステムへ適用する手法を検討し，多様な組込みシステムに対して汎用的に適用できる見込みを得た。また，ホワイトリスト完全性の課題に対して，組込みシステムのモジュール生成時にホワイトリストを静的に自動生成する手法を実現した。

(注1) Linuxは，Linus Torvalds氏の登録商標である。

3. 組込みシステム向けセキュリティソフトウェアの開発

3.1 当社試作ソフトウェアのセキュリティ機能の実現手法

当社試作ソフトウェアでは，Linux OSのカーネル内システムコールフローに介入し，プログラムの実行要求を行う処理とプログラムの展開を行う処理の間に，ホワイトリスト照合による実行判定処理を挿入する。この実行判定処理では，プログラムが実行される場合に対象のプログラムがホワイトリストに存在するかを判定し，存在すれば元のシステムコールフローへ戻すことでプログラムを実行する。もし，存在しなければ認証エラーとしてプログラムの実行を禁止する。また，ファイルが書き込みモードでオープンされた場合，対象のファイルがホワイトリストに存在するかを判定し，存在すれば認証エラーとして処理を終了する。もし，存在しなければ元の処理フローへ戻す。これによって，ホワイトリストに登録されているプログラムだけ実行を許可し，ホワイトリストに登録されているファイルの改変を禁止する。当社試作ソフトウェアのセキュリティ機能の実行処理フローを**図1**に示す。

ホワイトリスト照合による実行判定処理をOSのシステムコールフローに挿入するためには，システムコールフローの処理で，次処理を呼び出している場所を特定し，次処理の呼出先をホワイトリスト照合による実行判定処理に置き換える必要がある。システムコールフローの各処理はメモリ上にCPU命令(アセンブラ命令)の形式で格納されており，CALL命令等の次処理へ移動する命令を解析することで，次処理の呼出先メモリアドレスを特定できる。当社試作ソフトウェアでは，Intelアーキテクチャの CPU命令セット(アセンブラ命令)を使用したプログラム実行要求処理で，次処理を呼び出している場所を特定し，呼出先のメモリアドレスを実行判定処理のアドレスに書き換えることによって，実行判定処理の挿入を実現している。挿入手法を**図2**に示す。

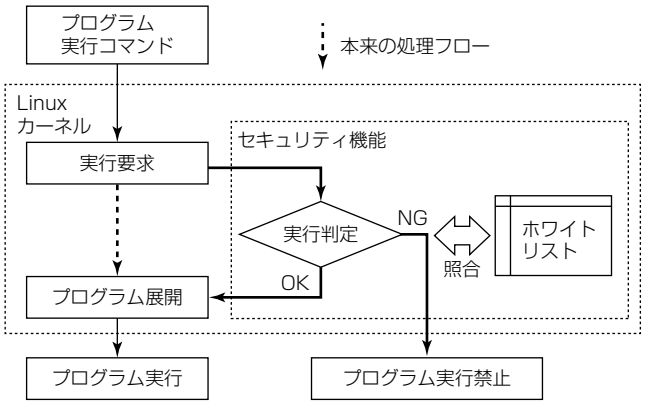


図1. 当社試作ソフトウェアのセキュリティ機能の実行処理フロー

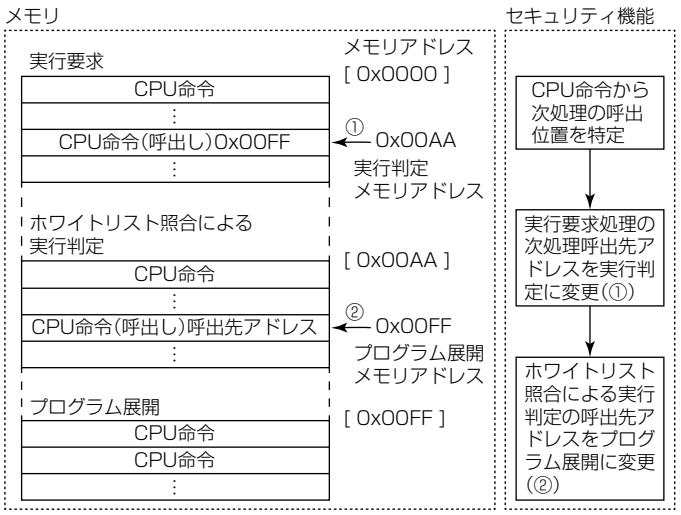


図2. 当社試作ソフトウェアでの実行判定処理の挿入手法

3.2 組込みシステムへの適用課題

CPUアーキテクチャによって使用できるCPU命令セットは決まっているため，当社試作ソフトウェアを異なるCPUアーキテクチャの環境に移植する場合は，移植先のCPU命令セットを解析し，実行判定処理の挿入が実現可能か個別に確認する必要がある。組込みシステム開発では，多様な組込みCPUからシステム構築に適したものを選定するため，開発機種ごとに使用するCPUが異なる。そのため，多様なCPUが存在する組込みシステムへ当社試作ソフトウェアを汎用的に適用することは困難である。

3.3 解 決 手 法

当社試作ソフトウェアを，多様な組込みシステムへ汎用的に適用するためには，CPU命令セットを使用してメモリ上で実現していた実行判定処理の挿入をCPU命令に依存しない別の手法で代替する必要がある。

当社試作ソフトウェアはLinuxサーバに適用することを前提にした実現方式にしており，組込みシステムへの適用は想定していなかった。Linuxサーバ開発の場合，開発元からバイナリー形式で提供されるOSを汎用的なCPUを搭載したサーバマシンに適用することで実行環境を構築し，実行環境と同一のシステム上でソフトウェア開発を行う。Linux OSがバイナリー形式で提供されること，サーバシステムが汎用的なCPUを使用することから，当社試作ソフトウェアではCPU命令を使用してメモリ上でOSのシステムコールフローに介入し，Linux OSに対して後天的に実行判定処理を挿入する手法を採用していた。一方，組込みシステム開発では，構築するシステムに応じて多様なCPUを使い分ける。また，リソースの制約から実行環境とは異なるシステム上でソフトウェア開発を行い，Linux OSを含めた全てのソースコードを実行環境のシステム向けにクロスコンパイルしている。Linux OSのソースコードをコンパイル可能であることから，組込みシステム開発では，Linux OSのソースコード上でシステムコールフローに介入し，Linux OSに対して先天的に実行判定処理を挿入することが可能である。

そこで，CPU命令に依存しない実行判定処理の挿入方法として，Linux OSソースコード上の実行判定処理を挿入すべき場所にあらかじめ高水準言語で定義したフック関数を挿入しておき，挿入したフック関数から当社試作ソフトウェアの実行判定処理を呼び出す方法を検討した。この解決手法によるセキュリティ機能の実行処理フローを**図３**に示す。これによって，CPU命令セットで実現していた実行判定処理の挿入を代替可能であり，当社試作ソフトウェアを多様な組込みシステムへ汎用的に適用することが可能になる。

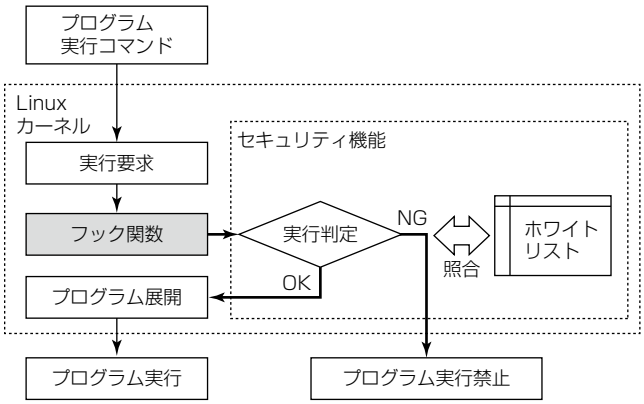


図３．解決手法によるセキュリティ機能の実行処理フロー

3.4 性 能 評 価

性能評価検証のため，当社試作ソフトウェアに実行判定処理の挿入代替手法を使用して開発した組込みシステム向けセキュリティソフトウェアを，組込みシステムで使用されることが多いARM系CPUを使用したシステムに適用し，動作確認と性能評価を実施した。動作環境の詳細を**表３**に示す。

結果として，ホワイトリストに登録されているプログラムだけ実行を許可し，ホワイトリストに登録されているファイルの改変を禁止する処理ができていることを確認した。さらに，リソースに制約がある組込みシステム環境での処理負荷を計測するため，組込みシステム向けセキュリティソフトウェアを適用した際の処理オーバーヘッドを計測した。この計測ではファイルのopen及びclose処理を1,000万回繰り返し実行し，１回当たりの処理時間平均を算出した。結果を**表４**に示す。この結果から，検証環境での処理オーバーヘッドはシステムコール全体の約３％(260ns)程度になっていることが分かる。システムの動作周期を１msと仮定した場合でも，動作周期の１％以下に

表３．セキュリティ機能移植環境比較

環境	(a)移植元	(b)移植先
CPUアーキテクチャ	Intel Xeon(注2) E5-2630(2.3GHz)	ARM v8(注3) Cortex(注3)-A53(１GHz)
メモリ	32GB	１GB
Linuxカーネル	3.10	4.4.120
ディストリビューション	RHEL(注4)，CentOS(注4)	Buildroot[5]

RHEL：Red Hat Enterprise Linux(注5)
（注２） Xeonは，Intel Corp.の登録商標である。
（注３） ARM，Cortexは，ARM Ltd.の登録商標である。
（注４） RHEL，CentOSは，Red Hat, Inc.の登録商標である。

表４．処理オーバーヘッド計測結果

ホワイトリスト判定処理の有無	処理時間(ns)
判定処理あり	9,146
判定処理なし	8,886

収まっているため，このセキュリティソフトウェアの処理負荷が極めて小さいことを確認した。

4. ホワイトリスト自動生成手法

4.1 当社試作ソフトウェアのホワイトリスト生成手法

当社試作ソフトウェアは，ホワイトリストの作成を容易にするために，ホワイトリストの作成補助機能を持つ。この機能は，学習モードでプログラムを実行させることで，プログラムから呼び出された動的ライブラリを捕捉し，ホワイトリストに登録するものであり。この機能の実行処理フローを**図４**に示す。この機能を使用することで，必要なプログラムとそのプログラムが参照するライブラリだけを登録した，必要最低限のホワイトリストを作成することが可能になる。一方で，抜け誤りのないホワイトリストを作成するためには，システム運用時に必要なプログラムを網羅的に実行し，ホワイトリストに登録する必要がある。システム運用時に必要となるプログラムは，機械的に判別することが難しく，人手による判断が必要である。このため，ホワイトリストに抜け誤りが生じる可能性があった。

4.2 組込みシステム向けセキュリティソフトウェアのホワイトリスト自動生成手法

組込みシステム向けセキュリティソフトウェアでは，ホワイトリストに抜け誤りを生じさせない手法として，ホワイトリストを静的に自動生成する方法を検討した。まず，組込みシステムのように非アドオンな設計となる開発では，システム更新以外でプログラムの追加や改変が発生しない。このため，実行環境でホワイトリストの作成や更新を行う必要はなく，開発環境でホワイトリストを事前に作成可能である。また，組込みシステムの開発では，開発環境やソースコードを全て自社内で管理しているため，開発環境で生成されるリリースモジュールは全て信頼できる。こ

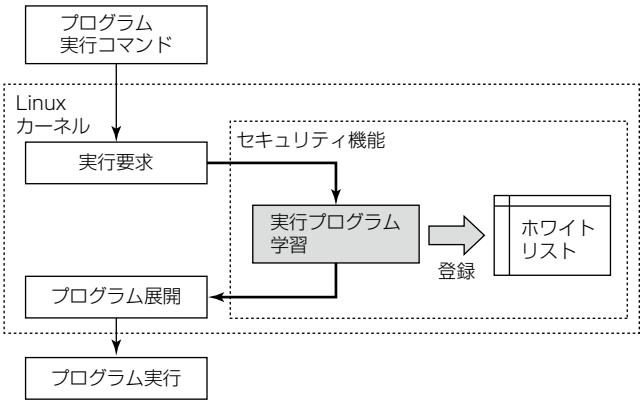


図４．当社試作ソフトウェアのホワイトリスト生成処理フロー

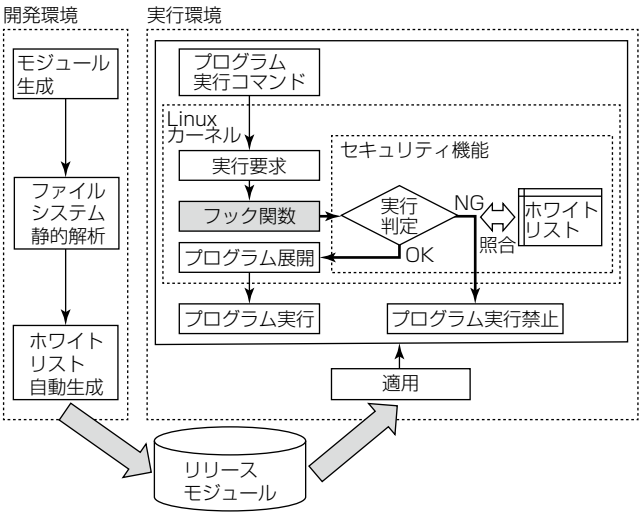


図５．システム静的解析によるホワイトリスト生成フロー

れらのことから，ホワイトリストを静的に自動生成する方法として，組込みシステムのリリースモジュール生成時に，ファイルシステムを静的解析し，リリースモジュールに含まれる全てのプログラムとライブラリを抜け誤りなくホワイトリストに登録する手法を開発した。処理フローを**図５**に示す。この手法を適用することで，リリース後に追加される不正プログラムの実行を禁止することが可能である。

5. む　す　び

当社がLinuxサーバ向けに試作したセキュリティソフトウェアを組込みシステム向けに移植開発したセキュリティソフトウェアと，組込み向けARM系CPUの環境での性能評価結果について述べた。これによって多様な組込みCPUに対して汎用的に当社セキュリティソフトウェアを適用できる見込みを得た。また，従来は人手で作成が必要であったホワイトリストをモジュール作成時に静的に自動生成する手法を開発し，抜け誤りのないホワイトリストを容易に生成することを可能にした。

今後，開発した組込みシステム向けセキュリティソフトウェアを当社の組込みシステムへ適用することで，IoTシステムがもたらす恩恵を安心・安全に享受できる社会の実現を目指す。

参 考 文 献

(1) 総務省：令和元年版情報通信白書（2019）
https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r01/pdf/index.html
(2) OVH：The DDoS that didn't break the camel's VAC（2016）
https://www.ovh.com/world/news/articles/a2367.the-ddos-that-didnt-break-the-camels-vac
(3) 伊藤孝之，ほか：Linuxにおけるプログラムホワイトリスト化試作，情報処理学会第77回全国大会，431～432（2015）
(4) SELinux
https://github.com/SELinuxProject
(5) Buildroot
https://buildroot.org/