4E-02

Linuxにおけるプログラムホワイトリスト化試作

伊藤 孝之† 菅井 尚人† 藤田 淳文† 鶴 薫†

三菱電機株式会社†

1. はじめに

外部のネットワークと接続していない制御システムなどで，システムに持ち込んだPCや記憶媒体を介して，悪意のある不正プログラム(マルウェア)の感染が懸念されている[1]．マルウェア対策には，不正プログラムを実行させないブラックリスト方式と，正当なプログラムのみ実行を許可するホワイトリスト方式[2]があるが，ブラックリスト方式は外部のネットワークからマルウェア情報をダウンロードして，常に最新の状態にしておく必要がある．このため，外部のネットワークと接続していないシステムにはホワイトリスト方式が適していると言われており，セキュリティ対策ソフトウェア会社からホワイトリスト方式の製品が販売されている[3][4]．

今回，システム開発者からの仕様要求にきめ細かく対応するために，ホワイトリスト方式で自製化が可能かを確認することを目的として，制御システムがOSとして採用しているLinux上で試作を行った．

2. 実現機能

ホワイトリストの作成の煩わしさを無くしたいとのシステム開発者の要望をもとに，試作では，ホワイトリストによるプログラムの起動可否判定機能に加えて，ホワイトリストの自動生成機能を実現した．ホワイトリスト生成環境(セキュアな環境)でシステムを実行してホワイトリストを自動生成し，それを運用環境に適用してプログラムの起動可否判定に利用する(図 1)．ホワイトリストの自動生成環境，および運用環境で実現する機能は次のとおりである．

ホワイトリスト生成環境での実現機能

- 起動したプログラム，およびプログラムからロードした動的ライブラリを捕捉し，ホワイトリストとして登録する．

運用環境での実現機能

- マルウェアの実行を防ぐため，ホワイトリストに登録したプログラムのみ起動可能とする．
- ホワイトリストに登録されたファイルの不正な改変を防ぐため，ファイルに対する書き込み，削除，名前変更を捕捉し，ホワイトリストに登録されたファイルに対するものであればエラーとする．

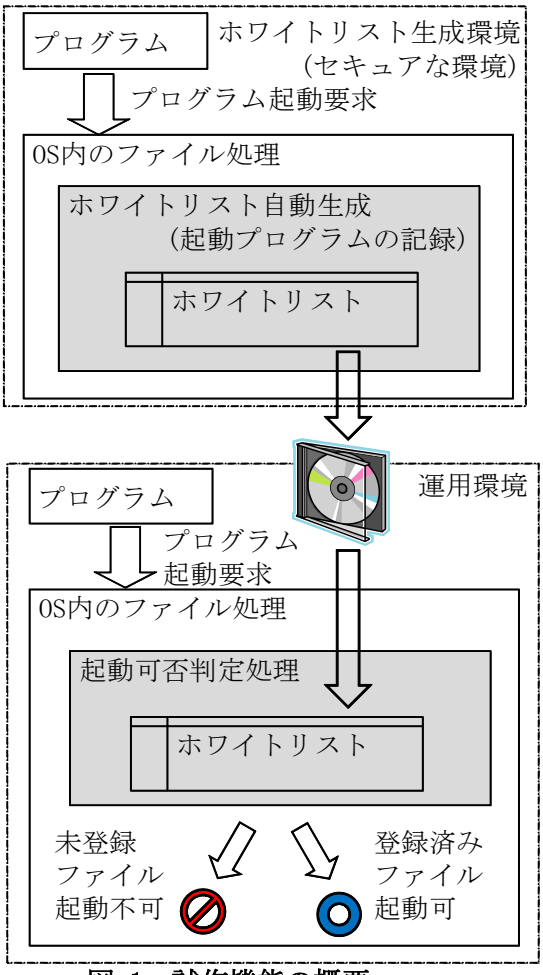

図 1 試作機能の概要

3. Linux上でのホワイトリスト処理の実装

ホワイトリストの管理，プログラムの起動可否判断，ファイルの不正改変防止，ホワイトリストの自動生成のLinux Kernelへの実装方法を以下に記す．

3.1 ホワイトリストの管理

Linux Kernel内部では，プログラムの起動やファイルのopen処理に与えたファイルパス(例えば/bin/ls)を，ファイルシステム内で固有の番号(i ノード番号)と，そのファイルシステムが存在するデバイス固有の番号(デバイス番号)に変換して管理する．そのため，ホワイトリストでも，このiノード番号とデバイス番号で管理する．Linux Kernelでは，平衡二分木の処理関数が用意されており(プロセスのアドレス空間の領域の管理に利用)，ホワイトリストの管理では高速に検索を行うためにこれを利用した．

i ノード番号は，異なる計算機上では同じファイル

Implementation for Program whitelists on Linux
Takayuki ITO†, Naoto SUGAI†, Atsufumi FUJITA†, Kaoru TSURU†
†Mitsubishi Electric Corporation.

パスであっても異なる値になるので，ホワイトリストを生成環境から運用環境へ移す場合は，i ノード番号とデバイス番号をファイルパスに変換する．そして，運用環境の計算機を起動時に，このファイルパスを読み込み，i ノード番号とデバイス番号に変換して，Linux Kernel 内に格納する．

3.2 プログラムの起動可否判断

プログラムの起動処理では，kernel は渡されたプログラムファイルのパスを i ノード番号とデバイス番号に変換し，ファイルへのアクセス権を確認した後，アドレス空間にプログラムをロードする．ファイルへのアクセス権を確認する際に，プログラムファイルの i ノード番号とデバイス番号についてホワイトリストへの登録有無を確認し，登録されていなければアクセス権エラーとする処理を追加した．

3.3 不正改変の防止

プログラムや動的ライブラリのファイルの不正改変防止は，ホワイトリスト登録済みファイルを open した時に書き込みモードであった場合エラーとして返す．ファイルの open 処理では，kernel は渡されたファイルパスを i ノード番号とデバイス番号に変換し，ファイルのアクセス権の確認を行っている．アクセス権確認の際に，ホワイトリストへの登録有無と書き込みモードを調べ，登録済みファイルを書き込みモードで open する場合はアクセス権エラーとする処理を追加した．

ファイル削除，およびファイル名変更処理においても同様に，kernel が削除対象や名前変更前，変更後のファイルを i ノード番号とデバイス番号に変更した後で，ホワイトリスト登録有無を確認し，改変に該当するケースはエラーとする．

3.4 ホワイトリストの自動生成

ホワイトリストの自動生成では，プログラムについては，kernel 内のプログラム起動処理において，プログラムファイルのファイルパスから i ノード番号とデバイス番号を求めた後，ホワイトリストに未登録であれば登録する処理を追加した．

また，プログラムから呼び出される動的ライブラリについては，ライブラリファイルの命令コードが格納されている領域を仮想アドレス空間にマップ(mmap)する処理において，実行権付きのものであれば動的ライブラリとみなし，ホワイトリストに登録する．

4. オーバーヘッドの測定

表 1は，ホワイトリストの処理に要するオーバーヘッドを知るために，open と close の処理時間をホワイトリスト処理の有る場合と無い場合で比較したものである．ホワイトリストに登録されていないファイルを書き込みモードで open すると，ホワイトリスト処理は不正な改変でないことを確認するために，ホワイトリストの平衡二分木を根から葉まで探索して，一致しないことを確認する．プログラムの起動処理でオーバーヘッドを測定した場合は，平衡二分木の葉に至る前のノードで一致するケースがあるので， open 処理での場合より少ないオーバーヘッドとなるため，より多くのオーバーヘッドとなる open 処理で測定した．ホワイトリストには 2047 個のファイルを登録しておき，すべての葉まで同じ深さとしている．この open と close の処理を 1 億回繰り返し，1 回あたりの処理時間を求めた．この時の測定環境を表 2に示す．

表 1　ファイルの open+close の処理時間

ホワイトリスト処理有無	処理時間[ns]
ホワイトリスト処理有り(*)	1422
ホワイトリスト処理無し	1407

*) 2047 ファイル登録のホワイトリスト

表 2　測定環境

項目	諸元
CPU	Xeon E5-2630(6core, 15MB L3 cache, 6×256KB L2 cache, 2.3GHz)
メモリ	32GB (DDR3 1333MHz)
OS	CentOS5.11 32bit (kernel2.6.18)

5. おわりに

本稿では，Linux でのホワイトリスト化で実現した機能，実装方法，処理のオーバーヘッドについて述べた．今後は，実際のシステムに適用し，問題点の洗い出し，実際に使ったうえでの要望を反映していく予定である．

参考文献

[1] 小熊 信孝, "Stuxnet - 制御システムを狙った初のマルウェア -", JPCERT/CC, https://www.jpcert.or.jp/ics/2011/20110210-oguma.pdf", 参照2014/11/26

[2] Janet Lo, "Whitelisting for Cyber Security: What It Means for Consumers", The Public Interest Advocacy Centre, http://www.piac.ca/files/whitelisting_final_nov2010.pdf, 参照2014/11/26

[3] マカフィ, "McAfee Application Control", http://www.mcafee.com/jp/products/application-control.aspx, 参照2014/11/26

[4] トレンドマイクロ, "不正プログラムの侵入をロックダウンで防止 - Trend Micro Safe Lock", http://www.trendmicro.co.jp/jp/business/products/tmsl/index.html, 参照2014/11/26