Vol.2018-OS-144 No.7
2018/7/31

ホワイトリスト型実行制御機構 WhiteEgret™ における Intel SGX による保護方式の検討

内匠真也[†1]　藤松由里恵[†1]　小池正修[†2]　金井遵[†1]　花谷嘉一[†1]

概要：WhiteEgret™はホワイトリストにないプログラムの実行を防止する実行制御技術である．WhiteEgret™は Linux®のカーネルをベースに構築しているが，柔軟な実行制御を実現するために多くの処理をユーザ空間上で実装できる特徴を持つ．このユーザ空間上の処理は差異化技術となりうるため，第三者による解析から保護できることが望ましい．そこで，今回，Intel®社製 CPU 搭載のセキュリティ機能 Intel ®SGX を活用することで WhiteEgret™の処理の機密性を保護する方法を検討した．WhiteEgret™の各処理について機密性の必要性を明確化し，Intel® SGX を利用して Trusted 領域と Untrusted 領域に処理を分割した．さらにプログラム難読化技術を併用することで，処理の機密性を保護できることを確認した．

キーワード：ホワイトリスト，実行制御，Intel SGX，IoT

1. はじめに

センサーなどの多様な機器をネットワークに接続することで，様々なサービスを実現する IoT (Internet of Things)の普及が進んでいる．工場や発電所などの重要インフラシステムにおいてもネットワーク化や標準技術の導入が進められているが，その副作用として重要インフラシステムを対象としたサイバー攻撃事例も増加している．重要インフラシステムの安全を守るためには，それを構成する IoT 機器へのセキュリティ対策が必要となる．

このような IoT 機器に対するセキュリティ対策の 1 つとして，ホワイトリスト型の実行制御技術が知られている．これは，機器上で実行を許可するプログラムを予めホワイトリストに記録しておき，ホワイトリストに存在しない未知のプログラムの実行を防止する技術である．この技術により，攻撃者が機器に送り込むマルウェアの実行を防止することで，機器を保護できる．ホワイトリスト型の実行制御技術として WhiteEgret™[1]がある．ホワイトリスト型の実行制御技術は，プログラムの追加・更新が生じない限りホワイトリストを更新する必要がないため，頻繁な機能変更が生じない制御機器・システムと相性が良いとされる．

WhiteEgret™ は，プログラムの実行可否を判定する処理をユーザ空間のアプリケーションとして実装できる特徴を持つ．そのため，カーネル側で実行可否を判定する処理を実装する場合に比べ，システム固有の要求を反映した複雑な判定処理を実装することが容易である．例えば，システムの状態によってホワイトリストを切り替えることや，ネットワーク経由で実行可否を判断することなどもできる．これらはセキュリティ強度やホワイトリスト運用の柔軟性・利便性を向上するためビジネス上の差異化技術になりうる．また，攻撃者が，判定基準や判定に用いるデータからシステムの構成を推測する恐れもある．このため，処理や処理に用いるデータを第三者による解析から保護できることが望ましい．

しかし，カーネルの脆弱性等によって管理者権限が奪取された場合には，プログラムのバイナリやデータをストレージから抜き出して静的に解析することや，デバッガ等を悪用し，動作中のプログラムを動的に解析することが可能となる．この課題を解決するために，本稿では Intel®[a] SGX（Intel Software Guard Extensions）を用いた WhiteEgret™ のプログラム保護方式を検討する．

2 章では，Intel SGX の概要を説明し，3 章で WhiteEgret™ の概要を説明する．4 章では Intel SGX を適用した WhiteEgret™ の設計についてまとめる．5 章ではセキュリティ評価を行い，6 章で課題をまとめ，7 章で関連研究について述べ，8 章で本稿を結ぶ．

2. Intel SGX

2.1 Intel SGX の概要

Intel SGX は Intel 社製 CPU の第 6 世代以降に標準搭載された CPU セキュリティ機能で，メモリ空間隔離によりプログラムのコードやデータの保護を実現する[2, 3]．Intel SGX を利用するプログラムは Untrusted 領域で動作するプログラムと Enclave と呼ばれる Trusted 領域で動作するライブラリに分割される．Intel SGX により，メモリ上での Enclave の完全性と機密性を実現できる．

完全性は Enclave に付与された Enclave 証明書に Enclave のデジタル署名情報が含まれており，Enclave の起動時にこの情報を検証することにより保証される．メインメモリ上

†1 (株)東芝　研究開発センター
Toshiba Corporation Corporate Research & Development Center
†2 (株)東芝デジタルソリューションズ
Toshiba Digital Solutions Corporation

a) Intel は，アメリカ合衆国およびその他の国における Intel Corporation またはその子会社の商標または登録商標です．

b) Linux は，Linus Torvalds 氏の米国，日本およびその他の国における登録商標または商標です．

その他本稿に掲載の商品，機能等の名称は，それぞれ各社が商標として使用している場合があります．

で Enclave のコードと扱うデータは認証付き暗号の AES-GCM-128 により暗号化されるため，メモリからの情報漏洩の防止や改ざんも検出できる．また機密性は暗号化だけでなく，メモリ空間の隔離によっても保証される．

Enclave へのアクセスは Call Gate により制限されており，OS や VMM などの権限の大きいソフトウェアからもアクセスできない．そのため，従来のセキュリティ技術では防止困難な OS 権限を不正に利用してアプリケーションが持つ情報を詐取する攻撃も Intel SGX は防止できる．

2.2 Intel SGX の機能

WhiteEgret™ に Intel SGX を適用するにあたり Intel SGX および，Intel SGX の SDK(Software Development Kit)である Intel SGX SDK が持つ機能をまとめる．

- OCALL/ECALL 機能
  Intel SGX では Untrusted 領域と Enclave 間の遷移は EENTER/ERESUME/EEXIT と呼ばれる命令を用いて行われる．Intel SGX SDK には，Untrusted 領域から Enclave を呼び出すための ECall(Enclave Call)，Enclave 領域から Untrusted 領域を呼び出すための OCall(Outside Call)と呼ばれる機能が用意されている．ECall/OCall では領域をまたいで引数や戻り値の授受が可能である．
- sealing 鍵生成機能
  Intel SGX の各 Enclave は EGETKEY と呼ばれる命令を用いて，Enclave 固有の sealing 鍵と Enclave 証明書由来の sealing 鍵の二つを常時取得できる． Enclave 固有の sealing 鍵は Enclave のハッシュ値と CPU 固有の HW 鍵により計算されるため，その CPU で動作する Enclave 固有の sealing 鍵となる．よって，Enclave 以外にその sealing 鍵を知ることはできない．Enclave 証明書由来の sealing 鍵は Enclave 証明書に含まれている Enclave のデジタル署名の検証に用いた公開鍵のハッシュ値と CPU 固有の HW 鍵より計算される．つまり，この sealing 鍵は同じ Enclave 署名用の鍵ペアを持つ Enclave 作成者（ISV）が作成した Enclave 間で共有利用できる鍵となる．この sealing 鍵を利用することにより，容易に同じ Enclave 作成者の Enclave 間でデータを安全に共有できる．
- Intel® Protected File System Library[4, 5]機能
  Intel SGX SDK は Intel Protected File System Library と呼ばれるライブラリを持つ．Intel Protected File System Library は Enclave 内のデータを暗号化してファイルとしてストレージに保存して扱うための API である．この API では AES-GCM-128 による暗号化を行う．また，ストレージからファイルを読み出す API では AES-GCM-128 の復号時に改ざんを検知し，完全性を検証できる．この API のデフォルト設定では Enclave 固有の鍵が利用される．一方で，任意の鍵を利用可能であり，Enclave 証明書由来の sealing 鍵も利用できる．
- Anti-debug 機能
  Intel SGX は Enclave に対するシングルステップ実行やブレイクポイント等のデバッグ機能を無効化する機能を持つ．この機能とメモリの暗号化機能により，Enclave は実際に動作させて情報を抜き出そうとする動的解析に対して耐性を持つ．

一方で，Intel SGX には以下のような制約が存在する．

- 保護対象はユーザ空間で動作する処理のみ
  Enclave として実装できるのはユーザ空間（Ring 3）で動作する処理のみであり，カーネル空間で動作する処理は Enclave 化できない．つまり，カーネル機能自体の保護は Intel SGX では不可能である．
- Enclave からは直接カーネル機能を呼出し不可
  Enclave 内ではシステムコールの呼出しが許可されておらず，カーネルを呼び出すためには OCall により Untrusted 領域に遷移する必要がある．この際に機密性を担保すべきデータを Untrusted 領域側に出力しないように留意する必要がある．
- Enclave ファイルの暗号化機能を持たない
  Intel SGX はメモリ上の Enclave を暗号化する機能を持つが，ストレージ上の Enclave ファイルを暗号化する機能は持っていない．そのため，Enclave ファイルにアクセスする権限を持つユーザはバイナリ解析ツールを用いて Enclave ファイルのコードやデータを静的解析できる．

以上のような制約を考慮して Intel SGX を利用する必要がある．特にカーネル機能を利用するアプリケーションについては，アプリケーションを Enclave 内で動作する処理と，Untrusted 領域で動作するカーネルを呼び出す処理に分割して実装する必要がある．

3. WhiteEgret™ 概要

3.1 WhiteEgret™ 概要

WhiteEgret™ は Linux®[b]機器上で実行を許可するプログラムを予めホワイトリストとして定義しておき，ホワイトリストにないプログラムの実行を拒否することで，機器・システムを保護する技術である．

WhiteEgret™ は，Linux のカーネル部の処理とユーザ空間上のデーモンプログラムで構成される．WhiteEgret™ のカーネル部の処理は，プログラムの実行要求を保留してユーザ空間上のデーモンプログラムに実行可否を問い合わせ，

デーモンプログラムがホワイトリストに基づいて実行を許可した場合に限り，保留した実行要求を再開する．

WhiteEgret™ のカーネル部分は，Linux の標準機能である LSM(Linux Security Modules)を利用して，アプリケーション実行の検知と保留/再開/中止を実現しており，Linux の改変が少なく保守性が高い．また，実行可否を判定するデーモンプログラムはユーザ空間上に実装されるため，システム環境に応じて，柔軟に，ホワイトリストの管理法や実行制御の判定処理を調整することも可能である．

ユーザ空間上に実装する WhiteEgret™ の判定プログラムは用途に応じて任意に実装することが可能だが，サンプルとして，ホワイトリスト生成・編集処理を行うプログラム weusr とカーネルからの問い合わせに対してホワイトリストとのマッチングを行うプログラム weusrd が用意されている．weusr や weusrd には高速化を施したハッシュ計算処理等の保護すべき差異化技術を含みうるため，機密性を担保することが望ましい．本稿では weusr と weusrd に対する保護方式を検討する．

3.2 ホワイトリスト生成・編集処理プログラム

本節では図 1 に示すホワイトリスト生成・編集処理を行う weusr の構成について述べる．weusr は実行ファイルのパス名とハッシュ値を記録するホワイトリストに対して，実行ファイルの追加・削除処理を行う．weusr は与えられたディレクトリのパスを走査し，発見した全ての実行ファイルをホワイトリストに登録する機能を持つ．また，weusr はコマンドライン引数として与えられたファイルのパス名に基づき，任意の実行ファイルをホワイトリストに追加，またはホワイトリストから削除することもできる．

ホワイトリストの追加・削除が終了した場合，その結果をログファイルに書き込む．ログとして出力する情報は，ログレベルにより設定することも可能である．

以下に weusr が持つ各処理の概要をまとめる．

- ホワイトリスト追加対象取得処理
  パス名に基づいて実行ファイルをホワイトリストに追加する．ディレクトリのパス名が与えられた場合はそのディレクトリ以下に存在する実行ファイルをホワイトリスト追加対象にする．ファイルのパス名が与えられた場合はそのパスをホワイトリスト追加対象にする．その後，後述のホワイトリスト条件付与処理に追加対象のファイルについてホワイトリストの追加を依頼する．
- ホワイトリスト条件付与処理
  ホワイトリスト追加対象のファイルについて，ハッシュ値など，パス名以外のホワイトリスト適用の条件を取得する．また，パス名とハッシュ値等の情報をホワイトリストアクセス処理によりホワイトリストに追加する．
- ハッシュ値計算処理
  与えられたパス名のファイルのハッシュ値を生成する．SHA-256 をはじめとしたハッシュアルゴリズムを利用可能である．
- ホワイトリストアクセス処理
  要求に基づいてストレージ上に存在するホワイトリストの読出し処理やホワイトリストの書込み処理を行う．
- ホワイトリスト削除処理
  ホワイトリストアクセス処理に基づいて読みだしたホワイトリストに対し，与えられたパス名のファイルをホワイトリストから削除する．変更したホワイトリストはホワイトリストアクセス処理により書き戻す．
- ログ生成処理
  ホワイトリストの変更結果を weusr ログファイルに書き込む．

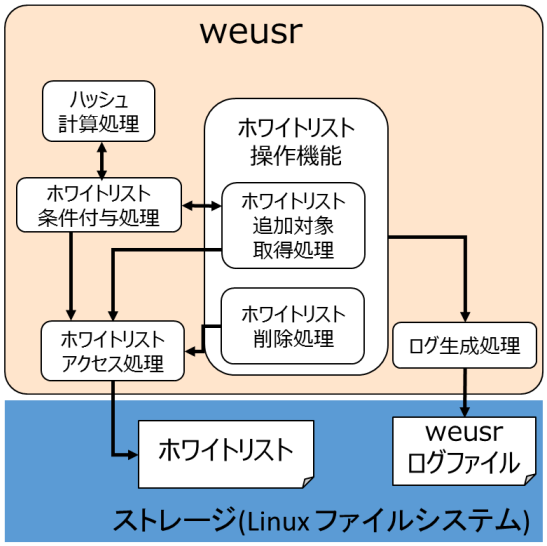

図 1　weusr の構成

3.3 ホワイトリストマッチングプログラム

ホワイトリストとのマッチングを行う weusrd の構成を図 2 に示す．weusrd では，LSM のメモリ空間にユーザ空間からアクセスする機能を有する securityfs を利用して，カーネル空間とユーザ空間の間の通信を実現している．securityfs は RAM ベースのファイルシステムであり，通常 /sys/kernel/security にマウントされる．WhiteEgret™ではカーネル側のモジュールが通信用の仮想ファイルを作成する．

weusrd は起動時にホワイトリストを読み込み，カーネル側のモジュールから判定処理要求があるまで待機する．判定要求があった場合はホワイトリスト判定処理を行い，仮想ファイルを通じて結果を返す．また，判定処理の結果についてはログファイルに記録する．ログの情報量についてはログレベルにより設定することも可能である．

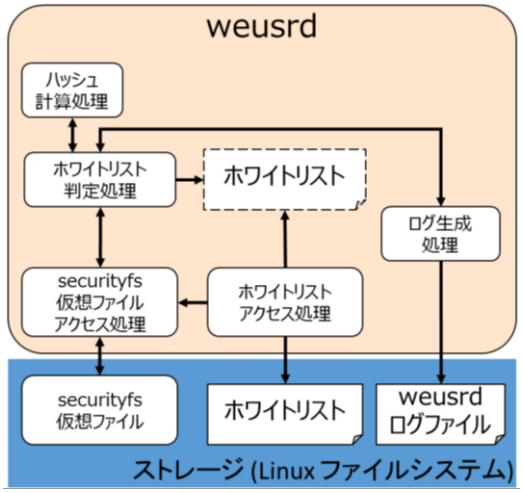

図 2　weusrd の構成

以下に weusrd の各処理の概要をまとめる．図 1 の weusr と共通のホワイトリストアクセス処理，ハッシュ値計算処理については 3.2 節を参照すること．

- Securityfs 仮想ファイルアクセス処理
  仮想ファイルにアクセスし，カーネル空間との通信を行う．仮想ファイル読込み時には，実行ファイルのパス名を含むファイル情報と判定要求プロセスの pid をカーネルから受け取る．仮想ファイル書込み時には，ホワイトリスト探索処理の実行制御判定結果をカーネルに送信する．
- ホワイトリスト判定処理
  与えられたパス名とハッシュ値などの条件に基づいてホワイトリストとのマッチングを行い，実行制御判定結果を求める．パス名と条件がホワイトリストと一致する場合は実行許可と判定し，一致しない場合は実行不可と判定する．
- ログ生成処理
  判定結果等を weusrd ログファイルに書き込む．

3.4 WhiteEgret™ の保護方針

WhiteEgret™ のホワイトリスト生成や実行可否判定に用いる weusr と weusrd では実行ファイルのハッシュ計算処理のノウハウ保護が必要である．また，今回の weusr は単純な実行可否判定処理のみを持つが，実行可否判定処理が複雑化すると判定処理自体やホワイトリストの内容についてもノウハウ保護の観点で保護することが望ましい．また，これらの処理やデータは攻撃者に対して攻撃に対するヒントを与えないという観点でも保護することが望ましい．これらの保護資産に対しては，特に機密性を保証することが重要となる．weusr の保護資産を表 1 に示し，weusrd の保護資産を表 2 に示す．

表 1　weusr の保護資産

保護資産	種別	機密性要否	理由
ホワイトリスト	データ	要	差異化となりうるホワイトリスト判定・管理技術を導入する可能性があるが，ホワイトリスト自体からその処理内容が推測可能であるため，保護する必要がある．
ホワイトリストアクセス処理	コード	不要	単純なファイルアクセス処理のコードであるため，機密性は必要ない．
ホワイトリスト追加・削除対象ファイルのパス	データ	不要	WhiteEgret™のインストール時のみに入力する情報であり，入力された情報はホワイトリストに保存される．ホワイトリストの機密性が担保されれば，本データ自体の機密性を保護する必要はない．
ホワイトリスト追加対象取得処理	コード	不要	ホワイトリストからデータを追加するのみで単純な処理である．またホワイトリスト追加対象のファイルパスに機密性が必要無いため，このコードの機密性を保護する必要はない．
ホワイトリスト削除処理	コード	不要	ホワイトリストからデータを削除するのみの単純な処理であり，ノウハウが含まれないため，機密性を保護する必要がない．
ハッシュ値計算処理	コード	要	ハッシュ値計算処理は高速化のために独自技術を導入する可能性がある．よって，対象のコードの機密性を保護する必要がある．
ホワイトリスト条件付与処理	コード	要	独自のホワイトリスト管理技術となる可能性があるため，機密性の保護が必要となる．
計算したハッシュ値	データ	不要	十分な暗号強度を持ったハッシュ関数であれば，ハッシュ値が衝突する入力データを生成ことは困難である．そのため，機密性の保護は不要である．
ログ生成処理	コード	要	コードに含まれる開発者用のデバッグ用のログは攻撃者に対するヒントとなる可能性があるため，機密性・完全性を保護する必要がある．
書込みログデータ	データ	不要	適切に出力ログを設定することで，攻撃のヒントになりうる情報の出力を制御できる．ログ生成処理の分析，ログレベルの改ざんが不可能であればログ自体には機密性は不要である．
weusr ログファイルの内容	データ	不要	適切に出力ログを設定することで，攻撃のヒントになるうる情報の出力を制御できるため，機密性は不要とした．

表 2　weusrd の保護資産

保護資産	種別	機密性要否	理由
ホワイトリスト	データ	要	表 1 と同じ.
ホワイトリストアクセス処理	コード	不要	表 1 と同じ.
ハッシュ値計算処理のコード	コード	要	表 1 と同じ.
ホワイトリスト判定処理	コード	要	本判定処理は差異化技術となりうる処理が含まれる可能性があり，機密性を保証することが望ましい.
実行制御判定対象の実行ファイルのファイル情報	データ	不要	Linux 側で実行されるファイルの情報であり，機密性は必要ない.
実行制御可否判定結果	データ	不要	実行可否情報は Linux に伝えるべき情報であり，それ自体に対する機密性は必要ない.
securityfs の仮想ファイルアクセス処理	コード	不要	単純なファイルアクセス処理のコードであり，機密性は必要ない.
ログ生成処理	コード	要	表 1 と同じ.
weusrd ログファイルの内容	データ	不要	適切に出力ログを設定することで，攻撃のヒントになりうる情報の出力を制御できるため，機密性は不要とした.
書込みログデータ	データ	不要	表 1 と同じ.
WhiteEgret™ のカーネル部	コード	不要	カーネルのコードはオープンソースであり，秘匿すべき内容が含まれない.

今回，表 1，表 2 の保護すべきコードおよび，データに対して Intel SGX による保護を適用することで WhiteEgret™ のセキュリティ課題である各保護資産の機密性保証の問題を解決する.

4. Intel SGX を適用した WhiteEgret™ の設計

WhiteEgret™に Intel SGX を適用するにあたり，表 1，表 2 で示した保護資産の機密性を保護できるようにコード・データを Enclave（Trusted 領域），Untrusted 領域に配置する．Intel SGX を適用した WhiteEgret の設計を図 3 に示す.

設計の詳細を以下に示す.

(1)　Trusted 領域・Untrusted 領域へのコードの配置

Enclave には機密性を担保すべきコードを配置する. コード自体にはノウハウが含まれないものであっても，そのコードが機密性を担保すべきデータを扱う必要があれば，Enclave に配置する必要がある.

まず，weusr のハッシュ計算処理・ホワイトリスト条件付与処理・ログ生成処理についてはコードに機密性が必要なため，Enclave に配置する．weusr のホワイトリストアクセス処理には機密性は必要ないが，ホワイトリストの機密性を実現するためにホワイトリストを操作するホワイトリストアクセス処理も Enclave に配置した．また，同様の理由

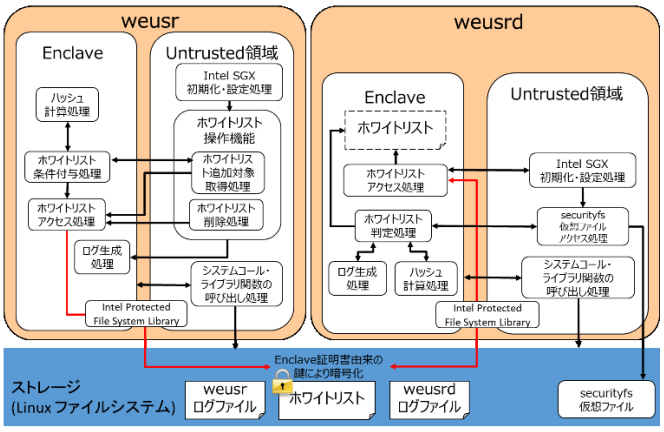

図 3 Intel SGX を適用した WhiteEgret の設計

で，weusrd についても，機密性が必要なホワイトリスト判定処理・ログ生成処理・ハッシュ計算処理と，ホワイトリストアクセス処理も Enclave に配置する.

(2)　Enclave ファイルの保護

Intel SGX はストレージ上に存在する Enclave ファイルを暗号化する機能を持たない. そのため，攻撃者が Enclave ファイルのアクセス権限を取得した場合，静的に Enclave の動作を解析することが可能である．Intel SGX では Enclave の機密性を保護できないが，ファイルのコード・データの機密性を保護する手法として難読化と呼ばれる手法がある [6]．そこで，今回の設計では難読化の手法を Enclave ファイルに適用し，静的な解析から保護することとした.

(3)　ホワイトリストファイルの保護

ホワイトリストアクセス処理部は Intel Protected File System Library 機能を用いる．これにより，ストレージに保存するホワイトリストファイルの暗号化を容易に実現できる．weusr と weusrd はどちらもホワイトリストファイルにアクセスするため，Intel Protected File System Library 機能で使用する鍵を weusr と weusrd 間で共有する必要がある．そこで本設計では Enclave 証明書由来の sealing 鍵を利用することとした．weusr と weusrd は同じ鍵ペアにより署名されているため，両者は同じ Enclave 証明書由来の sealing 鍵を生成し，利用できる．本機能により，Enclave 間のみで安全にホワイトリストを共有することができる.

(4)　ライブラリ・カーネル呼び出し機能

本システムでは Enclave に配置する多くの処理がライブラリ関数を呼び出す．ライブラリ関数を Enclave から利用するためにはライブラリ関数のコピーを Enclave に置く方法と，Untrusted 領域に置いたライブラリを Enclave から呼び出す方法がある.

前者のライブラリ関数のコピーを Enclave に配置する方法では，コード量が増大し，メモリの消費量が増加する.

また，ライブラリ関数はその処理内容・コードが公開されているため，機密性は必要ない．そこで今回は後者の Enclave が Untrusted 領域にあるライブラリ関数を呼び出せるように Ocall にライブラリを呼び出す機能を登録する方法をとることとした．また，Enclave はシステムコールを直接呼び出せないため，システムコールを呼び出す処理についても Untrusted 領域に機能を用意し，Ocall に登録する．

ただしこの方法では，機密性を担保すべきデータをシステムコールやライブラリに渡してはならない．表 1, 表 2 の保護資産のデータの内，ホワイトリストだけが Enclave で扱う機密情報となる．すなわち復号済みのホワイトリストのデータを Untrusted 領域にあるライブラリ関数に渡すと情報漏洩となる危険性がある．そのため，復号済みのホワイトリストを扱うライブラリ関数は Enclave にコピーを置き，平文のホワイトリストのデータが Enclave 外部に出ないようにした．

5. セキュリティ評価

本章では評価として Intel SGX による保護すべき情報の機密性担保の実現について考察する．3.4 節で機密性が必要と判断された情報は Enclave 内に配置した各種のコードと，ホワイトリストである．これらのコードやデータはメモリ上およびストレージ上に存在する．以下，それぞれに対する保護について考察する．

(1) メモリ上のコード・データ保護

Intel SGX は CPU の HW により生成された Enclave 固有の鍵によるメモリ暗号化機能を持つため，Untrusted 領域や他の Enclave は Enclave から情報を抜き出すことはできない．よって，攻撃者がシステムへの侵入を成功させたとしても，メモリ上に存在する Enclave から情報を抜き出すのは困難である．このため表 1, 2 の機密性を担保すべきコードやデータをメモリ上から抜き出すことは不可能である．また，Intel SGX は Anti-debug 機能により，メモリ上に存在する実行中の Enclave に対する動的解析の耐性を提供する．

(2) Enclave に含まれるコードおよびデータのストレージ上での保護

攻撃者は暗号化されていないストレージ上の Enclave ファイルから情報を抜き出す可能性がある．この攻撃についてはプログラム難読化により保護する．攻撃者が難読化を解読して，ファイルから情報を抜き出す攻撃も考えられるが，一般的に動的解析を行わずに難読化されたプログラムの内容を解読することは困難とされている[7]．そのため，静的解析については難読化で，動的解析に対しては(1)で述べたように Intel SGX の anti-debug 機能により，保護情報の機密性が保証される．

(3) ホワイトリストファイルに対する保護

ホワイトリストは Enclave 証明書由来の sealing 鍵で暗号化されている．ストレージ上に存在するホワイトリストファイルは上記暗号化により保護されるため，機密性を保護できる．また，認証付き暗号の AES-GCM-128 アルゴリズムによりホワイトリストファイルを暗号化するため，ホワイトリストファイルの改ざん検出も可能である．

また，メモリ上に読込まれたホワイトリストは Enclave 内でのみ扱われており，メモリ暗号化により保護されるほか, 復号したホワイトリストは Untrusted 領域で動作するカーネルやライブラリに渡されない．このため，Untrusted 領域のメモリ書換えやライブラリ，カーネル呼び出しを監視したとしてもホワイトリストの情報を得ることはできない．

以上の評価より，Intel SGX によって本研究の目的である WhiteEgret の機密性保護が達成できることが確認できた．

6. 課題

Intel SGX により機密性に関して保証できることが確認できたが， それ以外にも WhiteEgret™ にはセキュリティ課題が存在する．本章では今後の課題としてセキュリティ課題を列挙する．

6.1 完全性

本稿で述べた設計では Enclave から OS 渡すデータやファイルに記録するデータの完全性を向上させることができない．Intel SGX は Enclave 内からシステムコールを発行できず，Enclave と OS 間で通信するために，Untrusted 領域を通る必要がある．つまり，Untrusted 領域で Enclave と OS 間の通信を書き換えられる恐れがある．例えば，攻撃者が動作する weusrd と同レベルの権限を奪取した場合，ptrace 等のシステムコールを用いて Untrusted 領域において Enclave と OS 間の通信が書き換えられる危険性がある．

攻撃者により weusrd の OS 間の通信を改ざんされてしまうと，マルウェア等の意図しないアプリケーションの実行を許してしまう恐れがある．そのため，カーネルと weusrd 間の通信の完全性を担保することが望ましい．

6.2 可用性

Intel SGX の利用には Untrusted 領域の処理が不可欠であり，Intel SGX を利用するプロセスの管理も OS に委ねられている．このため，Intel SGX によりシステムの可用性を保証することが困難である．

Enclave を起動する Intel SGX の設定は Untrusted 領域で行われる．そのため，攻撃者は Intel SGX の設定処理を書き換えることで，Enclave の起動を妨げることができる．また，動作中もシステムコールの呼出し等のために Untrusted

領域に遷移するため, 攻撃者はその Untrusted 領域の処理を書き換えることで Enclave を利用するプロセスの動作を妨害できる．さらに，攻撃者はプロセスを終了させる kill コマンドを OS に送信することで，Enclave を利用するプロセスの動作を終了させることができる．これらの問題に対して可用性を向上させることが望ましい．

6.3 機密性

本稿の設計では Enclave に難読化を利用した．難読化にはセキュリティ強度と実行速度の間にトレードオフの関係がある．一般的にはセキュリティ強度を上げようとすると，コード実行速度の低下を引き起こすという課題がある．別の機密性確保の方式として，機密性が必要となるコード・データを暗号化した Enclave とその Enclave を復号して展開するローダ機能を持つ Enclave（以下，ローダ Enclave）を用意する方法も考えられる．この方法を用いるアプリケーションはまず平文のローダ Enclave を起動し，暗号化 Enclave をローダ Enclave により復号してもらい, Trusted 領域に配置して起動する．暗号化 Enclave は暗号化により機密性が保護される．また，暗号化鍵の提供方法にも依存するがローダ Enclave は機密性のある処理を含まない仕組みにすることも可能であり，平文のままで問題ない．暗号化による方式では性能低下を抑えることができる．

一方でこの手法は暗号化 Enclave を復号するための鍵を管理するコストが発生する．鍵を安全な管理・運用を考慮して設計する必要がある．

7. 関連研究

文献[8]では Intel SGX を拡張し, Enclave 間のアクセス制御を行う方法が述べられている．CPU の Ring 3 以外の空間で動作する OS やドライバにも Intel SGX が適用可能で，Trusted 領域からのシステムコール発行が可能になれば，よりセキュアなシステムを構築できる．一方で，独自に HW を拡張する必要があるため, 実現のためのコストが大きい．

文献[9]では Intel SGX と仮想化技術を併用し，Enclave が I/O を安全に行うことができるパスを構築する方法が述べられている．この手法では Enclave が利用する I/O のデバイスドライバのコードを仮想化で分離する．さらに，Enclave とデバイスドライバの間で認証を行い，Enclave が安全に I/O を発行できるパスを構築する．この方法を利用すれば, Enclave が利用するファイルやネットワーク処理の安全性を高めることができる．デバイスに対するアクセス制御等を実現する際には，本技術と併用することでセキュアな実行制御が可能となる．

文献[10]の SCONE では Docker 等のコンテナ保護に Intel SGX を利用する．SCONE ではコンテナ全体を Enclave 化する．SCONE ではシステムコールを発行するために Enclave から Untrusted 領域に遷移するときには Untrusted 領域に渡すデータを暗号化し，OS 側でそれを復号する．この暗復号処理が性能劣化を引き起こす危険性があるため，SCONE では非同期処理によるシステムコール実行の並列化を実現している．SCONE を利用することにより信用できないコンテナやコンテナのホスト OS から Enclave 化したコンテナの機密情報を保護することが可能となる．しかしながら保護対象のコンテナ内に脆弱性が存在した場合，情報漏洩や不正な制御が可能となってしまう欠点が存在する．

8. おわりに

本稿では，ホワイトリスト型実行制御技術 WhiteEgret™ の保護資産に対する Intel SGX を用いた機密性の保護方法について検討した． Intel SGX により機密性を保証できることを確認した．WhiteEgret™ の保護資産は完全性等が重要となるケースも存在する．今後は，WhiteEgret™ の保護資産について，完全性等を保護する方法について検討を行ってゆく．

参考文献

[1] 小池，小椋，内匠，花谷，春木，”Linux 上でのホワイトリスト型実行制御機能 WhiteEgretTM の開発”，CSS2017.
[2] Intel SGX：Protect Application Code & Secrets from Attack, https://software.intel.com/sites/default/files/managed/50/8c/Intel-SGX-Product-Brief.pdf, (2018).
[3] V.Costan, L.Lebedev, S.Devadas, Intel SGX Explained, eprint, https://eprint.iacr.org/2016/086.pdf, (2018).
[4] “Intel Protected File System Library”, https://software.intel.com/en-us/node/738203, (2018).
[5] “Overview of Intel Protected File System Library Using Software Guard Extensions”, https://software.intel.com/en-us/articles/overview-of-intel-protected-file-system-library-using-software-guard-extensions, (2018).
[6] A. Balakrishnan, et. al., “Code Obfuscation: Literature Survey”, Technical report, Computer Science Department, University of Wisconsin, Madison, USA, 2005.
[7] A Moser, et. al., “Limits of static analysis for malware detection”, In Proceedings of the 23rd Annual Computer Security Application Conference (ACSAC), pages 421–430, 2007.
[8] 濱本他，“Trusted Execution Environment 搭載デバイスのためのセキュア空間へのアクセス権限設定法の基礎検討”, CSS 2017, 3D1-4.
[9] S. Weiser, M. Werner, “SGXIO: Generic Trusted I/O Path for Intel SGX”, CODASPY 2017.
[10] S. Arnautov, et. al., “SCONE: Secure Linux Containers with Intel SGX”, 12th USENIX Symposium on OSDI, 2016.