データ指向攻撃の防御に関する一考察

福山 遼[1]　矢内 直人[1]　岡村 真吾[2]　藤原 融[1]

概要：データ指向攻撃はメモリ領域におけるデータ値を別の値に書き換えてプログラムのデータフローを操作し，秘密鍵などの機密情報の流出や権限昇格を可能にする．この攻撃に対して現状では確立された防御手段が著者の知る限りではないことから，本稿ではデータ指向攻撃の防御方法を議論する．本稿ではデータ指向攻撃の基本的な性質の考察を通じ，攻撃の早期検知による防御が有効であると考えている．さらに本稿ではデータ指向攻撃の検知ロジックも考察する．データ指向攻撃ではデータフローが強制的に変更されるため，データフローを動的に解析することで攻撃検知が可能となる．このような知見のもと，データフローが本来意図した正常なものかの判定問題に帰着する”データ指向攻撃検知システム”を形式的に定義する．

キーワード：データフロー，データ指向攻撃，非制御データ攻撃，攻撃検知，データ指向攻撃検知

A Study on Protection against Data-Oriented Attacks

Ryo Fukuyama[1]　Naoto Yanai[1]　Shingo Okamura[2]　Toru Fujiwara[1]

Abstract: A data-oriented attack allows an adversary to rewrite data values stored in memories by manipulating data flow of a program, and brings serious damage such as leakage of secret keys or escalation of privileges. To the best of our knowledge, there is no practical method for resisting the data-oriented attack, and thus we discuss a protection method against the attack. Based on consideration of fundamental features of the data-oriented attack, we especially focus on detection of the attack. More precisely, the data-oriented attack can be detected by dynamically analyzing data flow since the flow is manipulated via the attack. Under this observation, we formally define data-oriented attack detection system. That is, the detection of the attack is reduced to some decision problem whereby automata accept the data flow or not. We also discuss the feasibility of the proposed system.

Keywords: data flow, data-oriented attack, non-control-data attack, attack detection, data-oriented attack detection

1. 序論

1.1 背景

“プログラムの脆弱性”は情報技術において常に付きまとう課題である．脆弱性には明白なバグもあれば一見してわからないような潜在的な問題も存在し，脆弱性を孕んだプログラムが世に出回ることは多い．悪意を持った者にそこを突いた攻撃を受けて深刻な被害が出る恐れもある，そのため，様々な脆弱性をついた攻撃とその対抗手段の研究は重要である．

脆弱性をついた攻撃にはメモリの上書きを引き起こしコントロールフロー (プログラムの処理の流れ) を誤作動させる攻撃があり，その代表的なものの一つとしてあげられるのがバッファオーバーフロー攻撃である．これに対する有効な手段として，実行アドレスをランダム化するASLR(Address Space Layout Randomization [1], [2], [3]) やコントロールフローの整合性を確認する CFI(Control-Flow Integrity [4]) などといった技術が挙げられる．こういった対抗技術が実用的になり，コントロールフローを

[1] 大阪大学
Osaka University
[2] 奈良工業高等専門学校
Nara National College of Technology

狙った攻撃は困難になってきた．一方で，近年の研究ではプログラムにおけるデータの流れ (変数間の値の受け渡しなど) であるデータフローへの攻撃が盛んである [5]．これらの攻撃は “ データ指向攻撃 [5]”，“ 非制御データ攻撃 (non-control-data attack)[6]” と呼ばれているが，本稿ではデータ指向攻撃と呼称する．データ指向攻撃は，例えばメモリを上書きする脆弱性を利用することで，秘密鍵の搾取など影響が大きい被害を与える．文献 [5] によると，データ指向攻撃への有効な対抗手段がメモリの完全な保護以外に利用できるものがなく，また，著者らの知る限り，“ 完全な保護” を現実的に達成する方法もない．

このような背景から，本稿ではデータ指向攻撃に着目し，対抗手段を議論する．

1.2 貢献

本稿では，データ指向攻撃に対する基本的な防御の考察とそれを踏まえたデータ指向攻撃の検知システムのフォーマルな定義を行う．

まず基本的な防御の考察として，データ指向攻撃に対して一般的な防御手段から検討する．攻撃の一般的な対抗手段は，攻撃を未然に防ぐことと，攻撃の早期検知である．データ指向攻撃において前者の場合，メモリ情報を書き換える脆弱性の対策技術やメモリアクセスパターンの秘匿がある．メモリ情報を書き換える脆弱性の対策は命令制御を操作しないという攻撃の特性から有効性が期待できない．また，メモリアクセスパターンの秘匿はメモリへのアクセスパターンをランダム化することで外部からその挙動を隠蔽するものだが，データへのアクセスを制限するものではない．このため，データフローに影響せず，有効性が期待できない．一方で，後者の攻撃の検知は不正な値の出力を認識することによる実現が期待できる．

このような知見のもと，本稿でデータ指向攻撃の検知ロジックを考察する．データ指向攻撃では異常なデータフローを発生することから，データフローの情報を動的に解析することが攻撃検知につながる．これにあたり，プログラム実行処理におけるアドレスとデータ情報を表す”データフロートレース”を定義し，それを入力とした”データ指向攻撃検知システム”を示す．データ指向攻撃検知システムでは，データフローが本来意図した正常なものかの判定問題に帰着することで攻撃を検知する．また，データ指向攻撃システムの実現性についても，データフローの追跡・解析を行う既存技術の調査をもとに、議論する．

1.3 本稿の構成

以降，2 節では関連研究について触れ，3 節でデータ指向攻撃の詳細について述べる．4 節ではデータ指向攻撃に対する防御の根本的な議論を行う．そして 5 節ではそれを元にデータ指向攻撃検知システムを定義し，その実現性について議論する．最後に，実装に向けた考察を 6 で行い，7 節で本稿のまとめを述べる．

2. 関連研究

2.1 データ指向攻撃

データ指向攻撃は近年活発化しているデータフローを操作する攻撃であり，主にバッファオーバーフローや整数オーバーフローといったメモリを破壊する脆弱性を突いて行われる．そしてその脅威が現実的なものであるということが文献 [6] で示されている．スティッチングという手法が主にとられており，これはメモリエラーを利用してシステムコールのパラメータのデータフローの値を直接書き換えたり標準出力のバッファのポインタの中身を機密情報を保持した変数のアドレス値に書き換えたりするものである．このような攻撃は従来のコントロールフローを操作する攻撃とは異なる対策が必要となる．

また，文献 [5] によると，データ指向攻撃のエクスプロイトを自動生成することも可能になった．さらに文献 [7] では，データ指向攻撃によるアドレスの漏洩なしに ASLR を回避する手法，標的サーバにおけるボットなどプログラムのメモリ領域で任意の処理を可能にする手法，CFI を回避して read-only であるページの権限を改変する手法がエクスプロイトの生成を通して示されている．しかしながら攻撃の重要性が述べられていた一方で，実用的な対策については示されていない．

2.2 プログラム異常検知

プログラム異常検知システムはプログラムの流れを表すトレースを言語として処理し，異常な実行状況を検知する．文献 [8] によると言語はレベル 1〜4 まで分類されており，レベル 1 は文脈依存言語，レベル 2 は文脈自由言語，レベル 3 は正規言語，レベル 4 は制約のある正規言語となっている．

言語の制約はレベル 1 から 4 に行くにつれて強くなるが，それは同時に言語の表現力が低下することを意味する．すなわち，レベル 1 に属する言語が最も表現力が高く，検知システムで扱われる言語がレベル 1 に近づけば近づくほど，その検知能力もより高くなる．

3. データ指向攻撃のモデル

データ指向攻撃とは，バッファオーバーフローなどの脆弱性を突いたメモリの上書きを利用し，コントロールフローに干渉することなくデータフローを任意のものに変更する攻撃である [5]．この攻撃はプログラムのデータの流れを変えられるため，パスワードや秘密鍵などの機密情報の流出や管理者権限を取得するなどの権限昇格といった被害を起こすことができる．なお，以下に示す攻撃は OS やプログラムレベルなど特定の攻撃対象・環境に依存しない

汎用的な攻撃である．これは文献 [5] で示されたモデルに従っている．なお，文献 [7] ではよりプログラムレベルでのデータ指向攻撃のエクスプロイトの生成が示されているが，本稿では汎用的な防御策の提案を目指すため文献を採用する．

3.1 脅威モデル

3.1.1 実行環境

コントロールフローに干渉する攻撃に対する既存の防御ツールが使用できる．そこで，プログラム実行のコントロールフローを動的に解析して整合性を検証する CFI(Control-Flow Integrity)[4] や，メモリ上のデータ領域での命令実行を禁止するデータ実行防止，メモリのアドレス空間をランダム化する ASLR といったコントロールフローを操作する攻撃に対する防御ツールが有効な環境でデータ指向攻撃を仕掛ける．

3.1.2 仮定

CFI の ID(分岐命令や呼び出し命令の飛び先と戻りアドレスを検証するための値) を生成するような非決定性システムは秘匿されており，攻撃者はその詳細を知りえない．また，コントロールフローを操作する攻撃に対する防御ツールが有効であるため，攻撃者はコントロールフローへの干渉ができない．

3.1.3 攻撃者の目的

攻撃者の目的は情報の流出または権限昇格である．前者で標的になるのは秘密鍵やパスワードなどの機密情報，またはメモリのスタックが上書きされていないかを検証するスタックカナリアやランダム化されたアドレスなどプログラムが実行時間に生成するランダム値である．ランダム値の流出取得はそれを用いた防御を回避することにつながる．一方で権限昇格において標的になるのはシステムコールのパラメータやコンフィグデータである．権限を指定するシステムコールのパラメータやサーバにおけるパーミッションやルートディレクトリを指すコンフィグデータを改変することで，権限昇格を実現する．

3.2 動作例

攻撃者はメモリの値の書き換えを起こさない無害な入力によって得られる正常な実行トレース (良性トレース) を解析してプログラムのデータフローを把握する．データフローには機密情報や標準出力データを扱うものがあるが，例えばメモリエラーを利用して一方に関連するポインタの値をもう一方に関連するメモリ領域のアドレス値に書き換えることで，両者を繋げることで情報を流出させる．以降，攻撃者が欲しい機密情報の流れるデータフローを “ ソースフロー”，標準出力など攻撃者がソースを得るために利用するデータフローを “ ターゲットフロー”とする．

4. データ指向攻撃の防御に向けて

データ指向攻撃の対策として，攻撃を未然に防ぐ手法と攻撃に即時対応する手法の二つを検討する．前者の未然に防ぐ手法としてはメモリの上書きを許す脆弱性対策技術やメモリアクセスパターンの秘匿が，後者の即時対応する手法としては攻撃の検知が考えられる．

メモリの上書きへの対策：データ指向攻撃への対策としては，メモリの上書きを許す脆弱性の対策技術を強化することが標準的といえる．しかし，既存の対策ツールは主にコントロールフローに焦点を置いており，命令制御を操作しないという特性を持つデータ指向攻撃には有効性が期待できない．脆弱性が存在しそこを突かれることを想定した防御は必要であり，データ指向攻撃に対してはメモリの上書きが発生した場合に任意のデータフローの操作を阻止する技術を確立しなければならない．

メモリアクセスパターンの秘匿：メモリアクセスパターンの秘匿とは，メモリへのアクセスパターンをランダム化することでその挙動を外部から隠蔽する技術である．これを実現化しオープンソース化したものにパス ORAM(Path Oblivious Random Access Memory[9], [10]) がある．パス ORAM はメモリ領域を分割して木構造で処理し，アクセスごとにその配置をランダムに変更するツールである．
しかし，このような技術を用いてもデータへのアクセス自体を制限するものではなく，データフローへの影響はない．そのため，データ指向攻撃に対する有効性は期待できない．

攻撃の早期検知：攻撃は検知できれば即座にプログラムの実行を停止させることにより，その深刻な被害を防ぐことができる．また不正な値の出力を認識することが期待できるので，実現が見込める技術といえる．そこで本稿では，データ指向攻撃への対抗策として，データ指向攻撃の検知について議論する．さらに，近年の研究でプログラム異常検知の定式化が知られており [8]，それを応用して考察を行う．

5. データ指向攻撃検知システム

データ指向攻撃の防御手段としてその検知方法を議論する．前節で述べた通り，攻撃を検知できれば即座にプログラムを停止することで機密情報流出などの被害を防ぐことができる．本章では文献 [8] におけるプログラムトレースの定式化を紹介した後，応用概念としてデータフロートレースおよびデータ指向攻撃検知システムを定義する．

5.1 トレースの定義

プログラム異常検知はその名の通り，プログラムトレースからプログラムの異常を検知するシステムである．異常検知はプログラムが正常に動作しているかの判定問題で実

現しており，正常な動作であればトレースを受理し，不正な動作であれば受理しないようなオートマトンなどの受理マシンを用いている．

5.1.1 プログラムトレースの定義

本節では，プログラム異常検知について文献 [8] で記されている定義について述べる．

定義 1 (プリサイスプログラムトレース [8])．プリサイスプログラムトレース $\mathbf{T}$ はプログラムの自律的な実行部分において実行される全命令のシーケンスである．

プリサイスプログラムトレース $\mathbf{T}$ を処理するにはかなりのオーバーヘッドを伴うため，現実的ではない．そこで現実には以下のプラクティカルプログラムトレースを用いる．

定義 2 (プラクティカルプログラムトレース [8])．プラクティカルプログラムトレース $\ddot{\mathbf{T}}$ はプリサイスプログラムトレース $\mathbf{T}$ の部分シーケンスである．部分シーケンスはアルファベット Σ，システムコールなど全命令から選択した部分集合から成り立つ．

5.1.2 データフロートレースの定義

データ指向攻撃を検知するには，プログラム実行時における秘密鍵などのデータから変数のアドレスに至るまでのあらゆる値のメモリ領域での受け渡しを示すデータフローの情報が必要となる．以下では 5.1.1 節で述べたプログラムトレースの定義から，新たにデータフローを扱うトレースを定義する．

定義 3 (プリサイスデータフロートレース)．プリサイスデータフロートレース $\mathbf{T}_d$ は，プログラムの自律的な実行部分において実行される，データを含む全命令のシーケンスである．

定義 4 (プラクティカルデータフロートレース)．プラクティカルデータフロートレース $\ddot{\mathbf{T}}_d$ はプリサイスデータフロートレース $\mathbf{T}_d$ の部分シーケンスにあたるプラクティカルプログラムトレースである．部分シーケンスはアルファベット Σ やシステムコールなど，データを含む全命令から選択した部分集合から成り立つ．

以降，プログラムを静的解析することによって得られる既知のプログラムトレースと一致するトレースを “ 正常” であるとする．

5.2 データ指向攻撃検知システムの定義

前節における定義を元に，データ指向攻撃に対する検知システムを定義する．検知システムの定義に向けて，まず以下を仮定する．

仮定 1. 正当なデータフローを示すプリサイスプログラムトレース, プラクティカルプログラムトレースが取得可能.

仮定 2. 仮定 *1* のトレースから生成される言語を受理する機械 *(*オートマトン*)* が生成可能.

これらの仮定から，検知システムを定義する．

定義 5 (データ指向攻撃検知システム)．データ指向攻撃検知システム Λ_d は正常なプラクティカルデータフロートレースの決定性集合である言語 L_{Λ_d} を定義し，プラクティカルデータフロートレース $\ddot{\mathbf{T}}_d$ が L_{Λ_d} に属するかを検証する証明手続き G_{Λ_d} を生成する．プログラムの実行時間において入力として与えられるプラクティカルデータフロートレースは $\ddot{\mathbf{T}}_{d_{in}}$ とする．

データ指向攻撃検知 Λ_d は主に以下の二つのフェーズに分けられる．

学習 既知の正常なトレース $\{\ \ddot{\mathbf{T}}_d \mid \ddot{\mathbf{T}}_d$ が正常 $\}$ から $L_{\Lambda_d}, G_{\Lambda_d}$ を生成．なお，仮定 2 から証明手続き G_{Λ_d} はオートマトンを前提とし，決定性で言語を生成する．すなわち，$L_{\Lambda_d}, G_{\Lambda_d}$ は以下のように生成する．

$L_{\Lambda_d} = \{\ \ddot{\mathbf{T}}_d \mid \ddot{\mathbf{T}}_d$ が正常 $\}$ (決定性)

G_{Λ_d} : L_{Λ_d} を受理するオートマトン

検知 入力されるプラクティカルデータフロートレース $\ddot{\mathbf{T}}_{d_{in}}$ が L_{Λ_d} に属するかどうかを，G_{Λ_d} を用いて実行時間中に検証する．トレース $\ddot{\mathbf{T}}_{d_{in}}$ が G_{Λ_d} によって受理されない，すなわち $\ddot{\mathbf{T}}_{d_{in}} \notin L_{\Lambda_d}$ であれば，攻撃検知としてプログラムの実行を停止させる．

データ指向攻撃検知システム Λ_d は直観的には図 1 で表現される形で使用される．図 1 では，まず対象となるプログラムのソースコードをコンパイラを用いて静的に解析することによってデータフロートレース $\ddot{\mathbf{T}}_d$ を取得する．Λ_d は学習フェーズで L_{Λ_d} と G_{Λ_d} を取得した $\ddot{\mathbf{T}}_d$ から生成し，プログラムの実行に伴い検知フェーズに移行する．検知フェーズでは，L_{Λ_d} はプログラム実行から動的にデータフロートレース $\ddot{\mathbf{T}}_{d_{in}}$ を取得していき，$\ddot{\mathbf{T}}_{d_{in}}$ が G_{Λ_d} に受理されなければ，Λ_d はプログラムの実行を停止させる．

5.3 データ指向攻撃検知システムの能力について

検知システムは受理するトレースを共有している正常なトレースが多いほど，その精度は高い．データ指向攻撃検知システム Λ_d の精度については，次のように定義する．

定義 6 (データ指向攻撃検知システム Λ_d の精度)．Λ_d の精度は，Λ_d が受理する全ての $\ddot{\mathbf{T}}_d$ それぞれに対して同じトレースを部分シーケンスに持つ $\mathbf{T}_d$ の数の平均である．

また，Λ_d の検知能力について以下が導ける．

定理 1 (データ指向攻撃検知システム Λ_d の検知能力)．データ指向攻撃検知システム Λ_d の検知能力は，Λ_d に対応する言語 L_{Λ_d} の表現力によって決まる．

証明 ある二つの検知システム Λ_{d1} と Λ_{d2} を考える．$\Lambda_{d1}, \Lambda_{d2}$ はそれぞれ言語 $L_{\Lambda_{d1}}, L_{\Lambda_{d2}}$ を生成し，検知の精度は Λ_{d1} の方が Λ_{d2} より上とする．ここで二つのプリサイスデータフロートレース $\mathbf{T}_{d1}, \mathbf{T}_{d2}$ に対して，Λ_{d1} がそれぞれ異なるプラクティカルデータフロートレース $\ddot{\mathbf{T}}_{1\Lambda_{d1}}, \ddot{\mathbf{T}}_{2\Lambda_{d1}}$ で表現して区別できる一方で，$L_{\Lambda_{d2}}$ がどちらも同じプラクティカルデータフロートレース $\ddot{\mathbf{T}}_{\Lambda_{d2}}$ でのみ表現できるとする．これは $L_{\Lambda_{d1}}$ の方が $L_{\Lambda_{d2}}$ より表現力が高いことを示

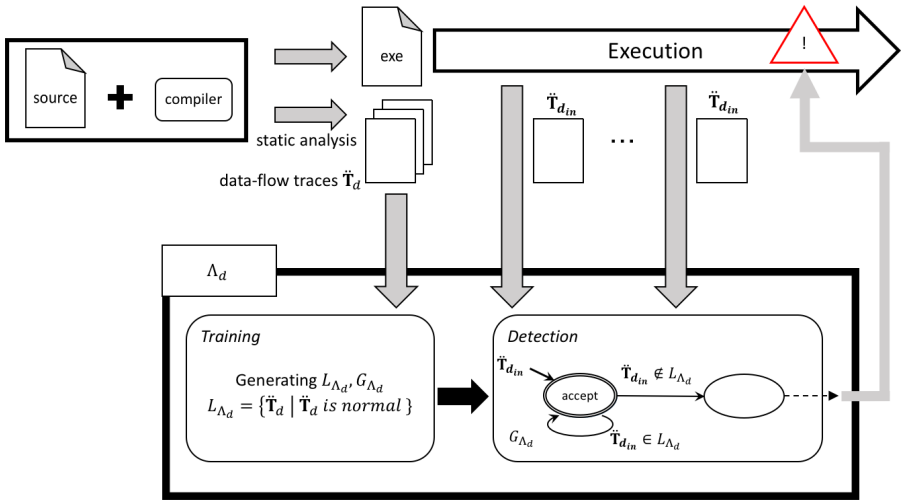

図 1　データ指向攻撃検知システム Λ_d の適用例

す．このとき Λ_{d1} は $\mathbf{T}_{d1}, \mathbf{T}_{d2}$ がそれぞれ正常か異常かを $\ddot{\mathbf{T}}_{1\Lambda_{d1}}, \ddot{\mathbf{T}}_{2\Lambda_{d1}}$ により独立して判定できるが，Λ_{d2} はそれができない．すなわち，Λ_{d1} の方が Λ_{d2} より検知能力が優れているといえる．□

定理 1 により，異なる検知システムの能力をそれらが扱う言語の表現力の比較を通してそれぞれ評価することができる．これら精度の定義 6 と定理 1 に関しては，コントロールフローに対するプログラム異常検知システムにおいて同様のものが文献 [8] で示されており，それをデータ指向攻撃検知システムのものに拡張したものである．

正常値のスコープ：データ指向攻撃検知システム Λ_d では，言語 L_{Λ_d} を生成するためにデータフロートレースの正常値，すなわち“データフロートレースが正常”とはどういうことであるかを明確にしなければならない．それを決めるものとしてプログラム異常検知では”正常値のスコープ”が定義されているが，これをデータ指向攻撃検知システムに向けて拡張する．

定義 7 (正常値のスコープ)．データ指向攻撃検知システム Λ_d の正常値のスコープ S_{Λ_d} は，プログラムにおいてトレースが正常であるかを判断する．すなわち Λ_d における S_{Λ_d} は正当なデータフローであり，L_{Λ_d} は正常値のスコープ S_{Λ_d} によって正常であるとみなされたプラクティカルデータフロートレースの集合である．

正常値のスコープ S_{Λ_d} は L_{Λ_d} の生成範囲を強調したものであり，L_{Λ_d} の表現力に言及したものではない．このため，Λ_d 検知能力には影響しない．正常値のスコープは決定性と確率性に分けられ，言語は以下のように定められる．

決定性　$L_{\Lambda_d} = \{\ \ddot{\mathbf{T}}_d \mid$ が正常 $\}$

確率性　$L_{\Lambda_d} = \{\ \ddot{\mathbf{T}}_d \mid P(\ddot{\mathbf{T}}_d) > \eta\ \}$

閾値 η は S_{Λ_d} や L_{Λ_d} による．

文献 [8] によると，一般的に検知システムは決定性では有限状態オートマトン (FSA) やプッシュダウンオートマトン (PDA) などが，確率性では隠れマルコフモデル (HMM) や 1 クラス SupportVectorMachine(SVM) などがベースになる．

5.4　データ指向検知システムの実現性

データ指向攻撃検知システム Λ_d の実現性を以下に示す．データフロートレースの処理が可能であれば，Λ_d はデータフローの判定問題として成立する．したがって問題となるのはデータフロートレースの処理可能性であり，それはすなわち 5.2 節の仮定 1,2 を議論することにつながる．よって以下より仮定の検証を行う．

仮定 1 では“正当なデータフローを示すトレースが取得可能”としている．これに対して攻撃者の立場ではあるが，文献 [5] によると，標的となるプログラムの実行トレースを静的に解析することによりそのデータフローの情報を取得している．このことから，トレースからデータフローを示すことができることがわかる．良性トレースも生成していることから正常なデータフローを示すトレースは取得可能といえる．ただし，その良性トレースからデータフローの内容を保持した実用的な部分シーケンスを取得できるかどうかを検証しなければならず，今後の課題となっている．ソースフローとターゲットフローを追いかけることで被害を防げるはずなので，実行の仕方によっては実行時間を抑えることが期待できる．

仮定 2 では“正当なデータフローを示すトレースで生成される言語を受理する機械が生成可能”としたが，データフローの読み取り処理が加わることでより多くの時間を費やすことが予想される．そこで検知システムではセンシティブな情報や権限を与えないことを防御のゴールとする．これによって扱うデータフローをクリティカルなものに限定できるため処理の負荷を軽減できる．

6. データ指向攻撃検知システムの具体的構成の候補

本章では，検知システムの実装に向けた議論を関連研究含め行う．

DFI(Data-Flow Integrity) とは，プログラムを静的に解析してデータフローのグラフを生成し，実行時間におけるデータフローがそれに則していることを保証する技術である [11], [12]．事前のプログラムの静的解析と実行時のデータフローの動的解析が必要なことから，プログラムが複雑になればなるほどデータフローグラフも複雑になり，オーバーヘッドが大きくなる．現在もそれを削減する研究がなされており [11], [12]，まだ実用化には至っていない．DFI と似た手法として特定のデータを追跡する動的テイント解析 [13] もあるが，データのタグを追跡しながら演算を行う必要があり，より負荷がかかる [14]．一方で, DF-Salvia[15] や KENALI[14]，PoL-DFA[16] といった提案システムの実現に利用が期待できるツールもあり，以降ではそれらについて検討していく．

DF-Salvia：DF-Salvia はプログラムに対し，データへのアクセスを制限するシステムである．DF-Salvia はデータの機密度をデータの保護方針として定義し，それをファイル単位に設定する．そしてデータが外部に出力される際に，出力データの根源であるファイルに設定された保護方針に従ってアクセスコントロールを行う．このように DF-Salvia は OS のアクセスコントロールに焦点を当てたものであり，OS 段階でデータ指向攻撃検知システムを導入する際に利用が期待できる．DF-Salvia を用いる場合，プログラムのソースコードをコンパイルした際にデータフローの情報を取得することができる．その情報をもとにデータフローごとに ID を割り当て，さらに ID テーブルを用いて実行時にデータフローを監視してアクセス制限を設ける．DF-Salvia は一演算あたりマイクロ秒単位でプログラムを処理することが可能となっており，データフローの動的な処理の観点からもデータ指向攻撃検知システムに適用できると考えられる．

KENALI：KENALI はカーネルアクセスコントロールシステムの不変関係 (平常時におけるプログラムの命令やデータの関係性) を利用したものであり，対象となるデータを重要なものに絞ってカーネルレベルで変化を検査する．カーネルを活用した観測システムは大きな負荷が予想されたが，オーバーヘッドを押さえた上での権限昇格を引き起こす攻撃を緩和することが確認できている．提案システムがトレースを用いてデータフローの異常を検知するのに対し KENALI はカーネルレベルで不変関係と実行状況の分析を行っているが，それを応用して異常なデータフローを検出することで提案システムと同様の効果を期待できる．

Pol-DFA：PoL-DFA は実行トレースを用いてサーバプログラムのデータフローを解析する技術である．実行トレースの全命令を順次解析し，データのソースのオペランドと出力など目的のオペランド間のデータフローが伝播する処理に読み替える．オペランドはプログラムオブジェクトまたはその一部であるため，データフローの伝播をプログラムオブジェクトレベルで解析することが可能になる．PoL-DFA は提案システム同様実行トレースをもとにしており，最も近い技術といえる．データ指向攻撃に対して PoL-DFA がどれほど有効であるかを検証した上で提案システムへ応用することが重要であると考えられる．

今後の課題はこれらのツールにおけるデータ指向攻撃耐性の評価，およびこれらを通じたデータ指向攻撃検知システムの実装である．

7. 結論

データ指向攻撃は脆弱性のあるプログラムにおいてデータフローに干渉してコントロールフローを操作せずに機密情報の流出や権限昇格を実現させる．本稿ではこの攻撃に対し，不当なデータフローを発生させるという性質からプログラムのデータフローを動的に解析して異常を検出する検知手法が防御として有効であるという見解を示した．そこでデータ指向攻撃検知システム Λ_d を定義した．このシステムは正当なデータフローを示すトレースからなる言語とそれを受理するオートマトンを生成する．そしてプログラムの実行時間においてその実行におけるデータフロートレースを動的に取得し，オートマトンに処理させることで正当なものかどうかを判定する．不当なものと判断した場合はプログラムの実行を直ちに停止させる．なお，正当なデータフローはプログラムを静的に解析することにより入手する．

プログラムのデータフローの処理は一般的にオーバーヘッドが大きく実用化するにはそれを削減する必要がある．そのため本システムではセンシティブな情報や権限を与えないことを防御の手段とし，扱うデータフローを極めて重要な情報が流れるものに限定して処理の負担を軽減する．また，実装には前節 (6) で述べた DF-Salvia[15] や KENALI[14]，PoL-DFA[16] といった技術の検知システムへの応用が期待でき，これらを使った実装が今後の課題である．さらには，脅威モデルで攻撃者に課した “ 攻撃成功のためにコントロールフローにへの干渉ができない”，といった制限をなくした上で検知システムが防御ツールとしての効力を発揮するかを検証していく予定である．

謝辞　本研究の一部は，JSPS 科研費 16K16065 の助成を受けている．

参考文献

[1] Shacham, H., Page, M., Pfaff, B., Goh, E.-J., Modadugu, N. and Boneh, D.: On the Effectiveness of Address-Space Randomization, *Computer and Communications Security*, ACM conference, pp. 298–307 (2004).

[2] Team, P.: PaX address space layout randomization (ASLR) (2003).

[3] Durden, T.: Bypassing PaX ASLR protection, *Phrack magazine, 11(59)*, (online), available from ⟨http://phrack.org/issues/59/9.html⟩ (2002).

[4] Abadi, M., Budiu, M., Erlingsson, U. and Ligatti, J.: Control-Flow Integrity, *Computer and Communications Security*, ACM conference, pp. 340–353 (2005).

[5] Hu, H., Chua, Z. L., Adrian, S., Saxena, P. and Liang, Z.: Automatic Generation of Data-Oriented Exploits, *USENIX Security*, USENIX, pp. 177–192 (2015).

[6] Chen, S., Xu, J., Sezer, E. C., Gauriar, P. and Iyer, R. K.: Non-Control-Data Attacks Are Realistic Threats, *USENIX Security*, USENIX, pp. 1–15 (2005).

[7] Hu, H., Shinde, S., Adrian, S., Chua, Z. L., Saxena, P. and Liang, Z.: Data-Oriented Programming: On the Expressiveness of Non-Control Data Attacks, *IEEE Security and Privacy*, IEEE, pp. 969–987 (2016).

[8] Shu, X., Yao, D. D. and Ryder, B. G.: A Formal Framework for Program Anomaly Detection, *Research in Attacks, Intrusions and Defenses*, Lecture Notes in Computer Science, Vol. 9404, Springer, pp. 270–292 (2015).

[9] Goldreich, O.: Towards a Theory of Software Protection and Simulation by Oblivious RAMs, *Symposium on Theory of Computing*, ACM, pp. 182–194 (1987).

[10] Ren, L., Yu, X., Fletcher, C. W., van Dijk, M. and Devadas, S.: Design Space Exploration and Optimization of Path Oblivious RAM in Secure Processors, *International Symposium on Computer Architecture*, ACM conference, pp. 571–582 (2013).

[11] Castro, M., Costa, M. and Harris, T.: Securing Software by Enforcing Data-flow Intefrity, *USENIX Conference on Operating Systems Design and Implementation*, USENIX, pp. 147–160 (2006).

[12] Akritidis, P., Cadar, C., Raiciu, C., Costa, M. and Castro, M.: Preventing Memory Error Exploits wth WIT, *IEEE Symposium on Security and Privacy*, IEEE, pp. 263–277 (2008).

[13] Enck, W., Gilbert, P., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P. and Sheth, A. N.: TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones, *USENIX Conference on Operating Systems Design and Implementation*, USENIX, pp. 1–15 (2010).

[14] Song, C., Lee, B., Lu, K., Harris, W., Kim, T. and Lee, W.: Enforcing Kernel Security Invariants with Data Flow Integrity, *Network and Distributed System Security Symposium*, Internet Society, pp. 1–15 (2016).

[15] Ida, S., Kashiyama, T., Takimoto, E., Saito, S., Cooper, E. W. and Mouri, K.: Design and Implementation of DF-Salvia which Provides Mandatory Access Control based on Data Flow, *International MultiConference of Engineers and Computer Scientists*, Newswood Limited, pp. 182–189 (2012).

[16] Xiao, G., Wang, J., Liu, P., Ming, J. and Wu, D.: Program-object Level Data Flow Analysis with Applications to Data Leakage and Contamination Forensics, *ACM Conference on Data and Applications Security and Privacy*, ACM, pp. 277–284 (2016).