Vol.2012-DPS-150 No.6
Vol.2012-CSEC-56 No.6
2012/2/29

マルウェア通信における暗号化鍵特定手法の提案

河内清人[†]　　桜井鐘治[†]

近年、防衛や重要インフラの機密情報を狙ったと思われる新種の標的型攻撃が海外，国内を問わず確認されており，重大な脅威として注目されている。このような標的型攻撃は，標的組織内の端末にマルウェアを送り込み，インターネットから制御することで行われる．攻撃は標的組織のセキュリティ対策を回避するよう巧妙に行われるため，検知・防止を行うことが難しく，攻撃発覚時には，既に情報が漏えいしてしまっている可能性が高い．そのため，攻撃を受けた事が明らかになった場合，外部へ漏れた情報を特定する被害特定作業が不可欠となる．そのためにはマルウェアが外部へ送信したメッセージに施した暗号化を解く必要がある．本稿では，漏えいした情報を特定するため，通信記録からマルウェアが使用した暗号化鍵を特定する手法を提案する．本方式では，マルウェアに復号処理を行わせ，その処理中にアクセスされたマルウェア内部のメモリ間の依存関係を解析することで鍵の候補の抽出を行う．本研究は現在方式検討段階であり，今後本方式に対する試作，評価を実施し，有効性について確認していく予定である．

Detecting An Encryption Key Used in Malware Communication

Kiyoto Kawauchi[†]　and Shoji Sakurai[†]

In recent years, new type of targeted attack has become a significant threat. The attack is done by malwares which infect into a target network and are controlled by an attacker in the Internet. This type of attack is so sophisticated that can avoid being detected and prevented. So, if such an attack is found in a network, it is highly probable that some secret information has been leaked. This is the reason why identifying the damage of information properties is needed when it becomes clear that an attack has hit the network. To do so, the incident response team of the network must find an encryption key used to encrypt messages sent by the malware, and decrypt all messages by the key. We propose a method for detecting an encryption key used in malware communication. In our method, the malware found in the network is executed on an emulator and its message decryption process is recorded to an execution log. The recorded execution log is analyzed to mark all memories accessed by the malware. And finally, candidates of the encryption key are identified by tracking dependencies among them. This work is currently concept level, and prototyping and evaluation are for future work.

1. はじめに

近年、防衛や重要インフラの機密情報を狙ったと思われる新種の標的型攻撃が海外，国内を問わず確認されており，官公庁や企業を中心に、重大な脅威として注目されている。これらの攻撃は，攻撃者が侵入した企業内で様々な攻撃手法を組み合わせ，長期間にわたって執拗に攻撃を行うことから，Advanced Persistent Threat（APT）とも呼ばれている．情報処理推進機構（IPA）ではこの攻撃を「新しいタイプの攻撃」と命名し，次のように定義している[1].

ソフトウェアの脆弱性を悪用し，複数の既存攻撃を組み合わせ，ソーシャル・エンジニアリングにより特定企業や個人を狙った攻撃の総称

攻撃者は，標的型メール等を通じて標的組織内のクライアント端末に RAT（Remote Administration Tool）と呼ばれるマルウェアをインストールし，それをインターネット上に構築した制御サーバを使ってリモートからコントロールすることで攻撃を行う．攻撃によって組織内部で取得した情報も，RAT を通じて制御サーバへ転送され攻撃者のもとに渡る．

このような標的型攻撃は，アンチウイルスソフト等，標的組織のセキュリティ対策を回避するよう巧妙に行われるため，検知・防止を行うことが難しく，攻撃が発覚したときには，実際に情報漏えい被害を受けてしまっている可能性が高い．

そのため，万一自組織が攻撃を受けてしまった場合，端末を隔離する等緊急的な対策を講じるのと並行して，実際にどのような情報が漏れてしまったのかを特定する被害特定作業も不可欠となる．

情報は RAT からインターネットへ送信されるため，日頃から組織内端末とインターネットとの通信を記録しておくことは，攻撃発生時に，RAT によって漏えいされた情報を把握する上で有効な手段と考えられるが，RAT は内部にハードコードされた鍵を使い，AES や Camellia といった暗号を用いて情報を暗号化した上で外部へ送信することも多く[2][3]，このような暗号化された情報を復元するためには，RAT を解析して暗号化鍵を取り出す必要がある．

そこで本稿では，標的型攻撃による情報漏えいが発生した際に，通信記録から漏えいした情報を特定するため，マルウェアが使用した暗号化鍵を特定する手法を提案する．鍵の特定は，マルウェアに復号処理を行わせ，その処理中にアクセスされたマルウェア内部のメモリの中から鍵と思われるデータを抽出することで行われる．マルウェアがアクセスする様々なメモリの中から鍵となる候補を絞り込むため，メモリ間の

[†]三菱電機株式会社 情報技術総合研究所
Mitsubishi Electric Corporation, Information Technology R&D Center

Vol.2012-DPS-150 No.6
Vol.2012-CSEC-56 No.6
2012/2/29

依存関係の解析を行う．

なお，本方式で特定される鍵は，実際には制御サーバから送られるメッセージに対する復号鍵である．そのため本方式は，暗号化鍵と復号鍵は同じものが利用されるとの仮定に基づくものである．

2. 関連研究

2.1 暗号アルゴリズムの特定

プログラム中で行っている暗号に対するアルゴリズムを特定する手法として，解析対象とするプログラムの実行ファイル内を検索し，アルゴリズム固有の定数データ（S-Box 等）を見つける手法が知られている．このような手法は既に PEiD[4]の KANAL を始め，様々なツールとして実装され，公開されている．しかし，これらの方式では，特徴となる定数データが存在しないアルゴリズム，例えば RC4[5]は検出できない．

これに対し，F.Gröbert らは[6]において，各アルゴリズムに対する実装コードをデータベース化し，一致する命令列を検出する手法，アルゴリズムで特徴的な命令をオペランドに含んだ命令（例えば cmp 18h 等）を検出する方法を提案している．

2.2 静的な鍵特定

プログラム中から鍵を探索する手法としては，Shamir らの提案した手法が知られている[7]．本手法は，静的にメモリ中のバイト列を検索し，高エントロピーなデータが連続して格納されている箇所を鍵データの候補とすることを提案している．しかし，この方式では圧縮されたデータのように鍵以外にも高いエントロピーを示すデータを含んだプログラムを解析した場合，誤検知が発生してしまう．

Halderman らは[8]において，鍵ではなく鍵スケジュールによってメモリ上に展開されたサブキーの配列を探す手法を提案している．各サブキーは，互いに関連しあっており，自由な値を取ることができないため，互いに一種のチェックサムとして機能する．この性質を利用し，AES[9]や DES[10]におけるサブキーの制約を満たしたメモリ配列を探すことで，（スケジュール後の）鍵を見つけるというものである．

2.3 動的な鍵特定

本稿で提案する方式は実際にプログラムを実行し，その結果から鍵を特定する手法である．本提案手法に最も近い手法は，先にあげた[6]において，暗号アルゴリズムの特定手法に加えて提案された鍵特定手法である．

[6]では，解析対象としたプログラムに対する実行トレースを記録し，実行時に参照されたメモリ中のデータから鍵を特定しようとする．しかし，[6]ではアクセスされたメモリ中のデータが，鍵，平文，および暗号文のいずれであるかが不明であるため，鍵，平文，および暗号文の候補をメモリ中のデータから選択し，暗号アルゴリズムに対応したリファレンス実装を使って正しい組み合わせを探索する必要がある．

一方，本稿で提案する手法は，データ依存関係を，実行トレース情報を使って追跡することにより，アクセスされたメモリの中から鍵の候補を絞り込むことができるという点が既存手法に無い特徴となっている．

3. 暗号化鍵特定システム

本稿による暗号化鍵の特定方式について，以下に説明する．

3.1 システム概要

図 1，図 2 に示すように，本システムは，業務ネットワーク上に設置された通信記録装置と，通信記録装置で記録された通信記録を解析するマルウェア解析環境で構成されている．

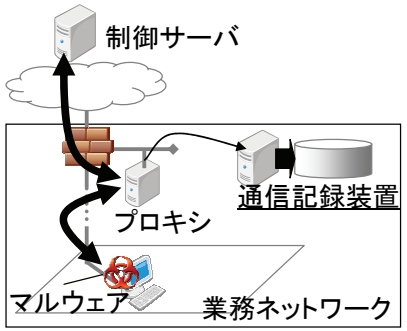

図 1 提案方式システム構成（業務ネットワーク）

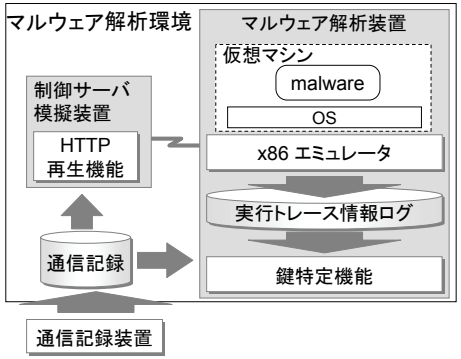

図 2 提案方式 システム構成（マルウェア解析環境）

通信記録装置は，平常時から外部と業務ネットワーク内の端末との HTTP 通信をへ

Vol.2012-DPS-150 No.6
Vol.2012-CSEC-56 No.6
2012/2/29

ッダ，ボディ部も含め全て記録する．トラフィックの異常やアンチウイルスソフトからの警告等により，ある端末上でのマルウェア感染が確認できた場合，端末からマルウェア本体を抽出するとともに，通信記録装置からマルウェアと制御サーバとの通信記録を抽出し，両者をマルウェア解析環境で解析を行う．

マルウェア解析環境には，通信記録を再生する制御サーバ模擬装置とマルウェアを動作させて挙動を記録するマルウェア解析装置が設置されている．解析担当者は，マルウェア解析装置上でマルウェアを実行する．マルウェアは，x86 エミュレータ上で実行され，その動作が x86 のインストラクションレベルで記録される．本稿では，インストラクション毎の動作記録を実行トレース情報と呼び，実行トレース情報の記録されたログファイルのことを，実行トレース情報ログと呼ぶことにする．

マルウェア解析装置上で実行されたマルウェアが制御サーバにリクエストを送信すると，そのリクエストは制御サーバ模擬装置へ送られる．制御サーバ模擬装置は，業務ネットワーク上で記録された通信記録のなかから，制御サーバから送信されたレスポンスメッセージを一つ選び，マルウェアへと送信する．レスポンスを受け取ったマルウェアは，レスポンス内に暗号化して格納されたコマンドを解釈するために内部で復号処理を行い，結果としてその時の実行トレース情報が実行トレース情報ログに逐次記録されていく．

マルウェアが復号処理を行った時点で，マルウェアの実行を止め，次に鍵特定機能上で実行トレースを解析することで鍵の特定が行われる．鍵特定機能では，復号処理中にアクセスされたメモリの中から鍵候補と思われるデータを抽出し，利用者に通知する． 以下の節では，マルウェア解析装置について詳しく見ていくこととする．

3.2 実行トレース情報ログ

3.1 で述べた通り，実行トレース情報ログは，x86 エミュレータ上でマルウェアを動作させることで記録する．x86 エミュレータは，QEMU[11]や，BOCHS[12]といったオープンソースツールに，実行トレース情報ログを出力するための機能を追加して実現する．

本提案方式で使用する実行トレース情報ログには，マルウェアが 1 インストラクションを実行するごとに以下の情報が追記されていく．

- 実行アドレス
- 実行命令
- 読み取りメモリアドレス，及び読み取られた値
- 書き込みメモリアドレス，及び書き込まれた値
- 実行後のレジスタ値

例えば，表 1 に示した実行トレース情報ログの例では，マルウェアが addr1 にある“mov eax,[ecx]”を実行したことを表し，その結果 mem1 に格納されていた val1 が読み込まれ，結果，eax レジスタの値が val1 に変化したことを表している．

同様に，addr4 において，mem3 へ，eax の内容である val3 が書き込まれたことを表している．

表 1　実行トレース情報ログ

実行アドレス	命令	Read		Write		レジスタ
		アドレス	値	アドレス	値	
addr1	mov eax,[ecx]	mem1	val1			eax=val1, ebx=…
addr2	sar eax					eax=.,,..
addr3	xor eax,[edx]	mem2	val2			eax=val3 …
addr4	mov [ebx+4*ecx+aabbccdd],eax			mem3	val3	eax=val3 …
...	...	...	...	...	...	...

3.3 鍵特定方式

3.3.1 処理概要

実行トレース情報ログを記録後，鍵の特定が行われる．本節では鍵の特定手法について説明を行う．本特定処理は，次の 3 ステップで行われる．

(1) 復号関数の特定

実行トレース情報ログを解析し，復号を行う関数のアドレスと，その暗号アルゴリズムを特定する．本稿では詳しい説明は省略するが，このような技術としては，関連研究で述べたような[6]で提案された方式を利用できると考えられる．

(2) メモリ間のデータ依存関係の追跡

（1）で復号関数を特定後，マルウェアがメッセージを受信してから復号を完了するまでの間にアクセスしたメモリ間の依存関係を，実行トレース情報ログに記録された読み書き記録をもとに抽出する．ここで述べる依存関係とは，あるメモリの内容がメモリやレジスタを通じて他のメモリの内容に反映されていることを指す．途中，算術演算等により，読み取られた内容とは異なる値が他のメモリに書き込まれたとしても，メモリ間には依存関係があるとみなす．

(3) 鍵候補の特定

抽出されたメモリ間の依存関係を使い，マルウェアが復号時にアクセスしたメモリ群の依存関係を追跡し，鍵の候補となるデータを抽出する．

3.3.2 メモリ間のデータ依存関係の追跡

データ依存関係の抽出は，実行トレース情報ログのなかで，InternetReadFile 等のネットワーク受信関数を呼び出した場所から解析を開始する．受信関数に引数として渡される受信バッファのアドレスとバッファ長を記録しておく．

その後，先の処理で特定した復号関数の処理が終了するまで，実行トレース情報を追跡し，メモリ間のデータ依存関係を表 2 で示すようなデータ構造を持ったデータ依存関係履歴情報に記録していく．「(line)」等，各見出しにある表記は，後のアルゴリズム説明で使用するためにつけた略称である．

表 2 データ依存関係履歴情報

実行トレース行番号（line）	実行アドレス（exec）	依存関係情報		
		データ格納場所（store）	依存データ情報	
			アドレス（address）	有効実行トレース行番号（d_line）
x	0x90000000	eax	0x11223344	z
		0xaabbccdd	0x55667788	m
			0x11223344	n
		...	...	...
...	...	...	...	...

表にある通り，データ依存関係履歴情報は，以下の要素で構成された構造体のリストである．

- 実行トレース行番号
- 実行アドレス
- 依存関係情報

依存関係情報は，その実行トレース情報に対応した処理をマルウェアが実行した時点で，データが格納された場所（メモリあるいはレジスタ）と，同格納場所に格納されたデータが依存しているデータの情報（依存データ情報）を表す情報であり，(データ格納場所，依存データ情報の集合）のリストの集合として表現される．依存データ情報は，当該データのアドレス，および同データが生成された実行トレース行番号（有効実行トレース行番号）で構成される．一つのデータ格納場所に格納されたデータが複数のデータに依存している場合，依存データ情報は一つのデータ格納場所に対し複数記録される．

例えば，表 2 で示されたデータ依存関係履歴情報は，実行トレース行番号 x では，実行アドレス 0x90000000 にある命令が実行されたこと，実行後，レジスタ eax の値は，実行トレース行番号 z でアドレス 0x11223344 に書き込まれたデータに依存した状態にあることを示している．同様に，0xaabbccdd に格納されたデータは，実行トレース行番号 m で 0x5566788 に書き込まれたデータと，n で 0x11223344 に書き込まれたデータに依存していることも示している．

各実行トレースに対し，データ依存関係履歴情報は， 図 3 に擬似コードとして示したアルゴリズムを適用することで更新される．なお，本擬似コードにおいて，データ依存関係履歴情報，依存関係情報，および依存データ情報は，表 3 のようなリストおよびリストの集合として表現されている．

表 3 擬似アルゴリズム中でのデータ依存関係履歴情報等の表現

データ依存関係履歴情報	リスト(実行トレース行番号，実行アドレス，依存関係情報)の集合
依存関係情報	リスト(データ格納場所，依存データ情報の集合)の集合．
依存データ情報	リスト(アドレス，有効実行トレース行番号)

このアルゴリズムでは実行トレース情報（行番号 tr）に対する依存関係情報を，直前の実行トレース情報で算出した依存関係情報と，当該実行トレース情報で記録された命令の内容を元に，次の式に従って新たな依存関係情報を計算し，データ依存関係履歴情報へ登録する．

$$F_{tr} = F_{tr-1} - \mathrm{kill}(tr) \cup \mathrm{gen}(tr)$$

ここで，F_{tr} は，実行トレース情報の行番号 tr に対する依存関係情報である．kill(tr) 及び gen(tr)は，行番号 tr の実行トレース情報に記録された命令を実行した結果，失われる依存関係情報および新たに発生する依存関係情報の集合を現す関数である．

kill は tr に記録された命令に従って値が変化する．命令が“mov”命令であった場合，そのディスティネーションオペランドで示される格納先のレジスタ/メモリに対して直前まで関連付けられていた全ての依存データ情報が返される．mov 以外の場合，例えばレジスタに対する xor 演算などでは，新たに格納されるデータは，過去に記録されていたデータの影響も受ける可能性がある．そのため，このような命令を実行するトレース情報に対して，kill は ϕ を返す．これにより，それまでに記録されていた依存データ情報は全て継承される．

gen は，命令のソースオペランドによって参照されたレジスタ/メモリとディスティネーションオペランドによって更新されたレジスタ/メモリとの間に新たな依存関係を表す集合を返す．もし，ソースオペランドが参照している先が，既に何らかの依存データ情報と関連付けられていた場合，それらも新たにディスティネーションオペラ

Vol.2012-DPS-150 No.6
Vol.2012-CSEC-56 No.6
2012/2/29

ンドに関連付けられる．

x86 系 CPU では，メモリ参照を行う際，ベース、インデックス、スケール、ディスプレースメントの 4 パラメータを組み合わせてメモリアドレスが決定される[13]。そのため，スケールを除くその他のパラメータを構成するアドレス，レジスタに関連付けられていた依存関係も，ディスティネーションオペランドの指す格納場所と関連付けが行われる．

もし，ソースオペランドが即値であった場合，新たな依存関係は発生しないため，gen は φ を返す．

UpdateDependency(**FS,**trace)
入力:
FS :データ依存関係履歴情報
trace:実行トレース情報
戻り値:
FS' :更新されたデータ依存関係履歴情報
1: **DepS** ← φ // **S** はソースオペランドの依存アドレスが格納される集合
2: OpSrc ← trace に記録されたマシン語命令のソースオペランド
3: OpDst ← 〃 ディスティネーションオペランド
4: F_{last} ← **FS**[line = trace の行番号-1]
5: **Src** ← OpSrc 内で参照しているレジスタ，メモリアドレス
6: **DepS** ← { (addr, tr) | s ∈ **Src**, (addr,tr)∈F_{last}[store = s] }
7: **if** OpSrc はメモリ参照 **then**
8: **DepS** ← **DepS** ∪ { (OpSrc が指すメモリアドレス, trace の行番号) }
9: **end if**
10: F_{new}←F_{last}
11: dest ← OpDst が指すメモリアドレス又はレジスタ
12: **if** trace に記録されたマシン語命令は mov **then**
13: F_{new}[store = dest] ← **DepS**
14: **else**
15: F_{new}[store = dest] ← F_{new}[store = dest] ∪ **DepS**
16: **end if**
17: **FS'** ← **FS** ∪ { (trace の行番号, trace の実行アドレス, F_{new}) }
18: **return FS'**
【表記】
var ← value 変数 var へ value を代入
FS[line = n] データ依存関係履歴情報 **FS** の実行トレース行番号=n に格納された依存関係情報
F[store=s] 依存関係情報 F の中で，データ格納場所=s に合致するレコードに含まれた依存データ情報の集合

図 3 依存関係の追跡処理

復号パラメータ候補情報の記録

メモリ間のデータ依存関係の追跡処理では，データ依存関係履歴情報を更新後，トレース実行情報に記録された命令がビット演算，あるいは算術演算だった場合，その演算結果の格納先が，ネットワークから受信したデータのバッファとの間に依存関係がないか確認する．もし依存関係があった場合，このトレース実行情報に対応する命令は，受信データもしくは受信データに由来する何らかのデータとのビット演算，算術演算を行ったと推測される．このように受信データとビット演算/算術演算を行うようなデータは，鍵、あるいは拡大鍵等、鍵から派生したデータである可能性があるため，同データに関連付けられた依存データ情報を，別途記録していく．この情報のことを以降，復号パラメータ候補情報と呼ぶ．

3.3.3 鍵候補の特定

メモリ間のデータ依存関係の追跡を終了すると，以下の情報が抽出される．

- 各実行トレース情報におけるメモリ間の依存関係情報を記録したデータ依存関係履歴情報
- 鍵から派生したデータをさす可能性のあるデータに対する依存データ情報の集合である復号パラメータ候補情報

次に、これらの情報を用いて，鍵候補の特定を行う．特定方式の概略を図 4 に示した．本方式のアイデアは，復号パラメータ候補情報の各依存データ情報に記載されているメモリアドレスに対し，さらに依存関係をさかのぼっていき，ハードコードされた，あるいは少なくともメッセージ受信後には変更を受けていない連続したメモリ領域に到達したときに，それを鍵候補とみなすというものである．

しかし，図 5 のように，復号関数内でたまたま他のビット/算術演算が含まれており，さらにその演算の依存先が，本来の鍵情報に隣接する形で配置されていた場合，どこまでが鍵情報で，どこからがその他の情報なのか，識別できなくなるという問題がある．

そこで，本提案方式では，復号を行うために実装されたビット/算術演算コードは，別の目的で実装されたビット/算術演算コードと比較し，アドレス上より近くに配置されているという仮定に基づき，復号関数内の同演算コードを，アドレス上の「近さ」で分類し，それぞれについて，個別に鍵候補の特定を行うこととした．

分類はビット/算術演算コードに対し，アドレスが低位のものから順に行っていく．直前に分類したコードからアドレスが閾値 T 以内であったならば同グループとみなし，そうでない場合には別グループとみなす．

Vol.2012-DPS-150 No.6
Vol.2012-CSEC-56 No.6
2012/2/29

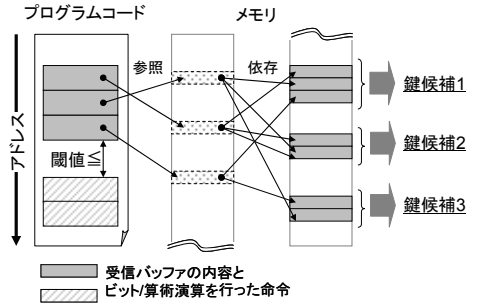

図 4 鍵候補特定方式概略

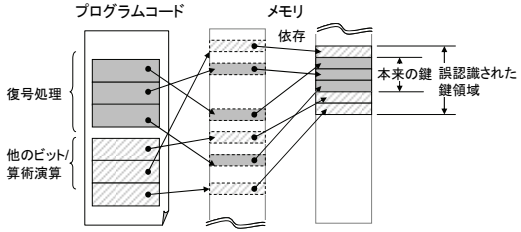

図 5 他のビット/算術演算による鍵候補推定誤りの発生

最後に，以上の処理を，擬似コードとしてまとめたものを図 6 に示す．図 6 中で参照している GetDependingAddress()は，依存関係のチェーンをさかのぼっていき，ルートとなるメモリ，つまり他のどのメモリとも依存関係が無いメモリのアドレスを抽出する処理である．図 7 に同処理の擬似コードを示した．

4. 考察

4.1 本方式の制限

本方式では，復号に使用する鍵=暗号化に使用する鍵という前提をおいている．暗号化に使用する鍵が復号に使用する鍵と異なっていた場合，直接本方式を適用して暗号化鍵を特定することはできない．

RAT は一般に，定期的に制御サーバに対してビーコンと呼ばれるメッセージを定期的に送信するため，ビーコン送信時に実行されるメッセージ暗号化処理を解析する方法が考えられる．具体的には，実行トレース情報を記録しながらマルウェアをしばらく実行させ，ビーコンとして送信されたデータが依存するメモリ群を本提案の依存関係追跡によって特定し，その中から暗号化鍵を含んだメモリ領域を特定する．

```
GetKeyCandidates(P, FS,TR)
入力:
   P  : 復号パラメータ候補情報
   FS : データ依存関係履歴情報
   TR : 実行トレース情報ログ
戻り値:
   K  :鍵候補データ集合

1:  K ← φ
2:  Q ← φ
3:  for each  (address, d_line) in P do
4:        exec ← FS[ line = d_line ].exec
5:        Q ← (address, d_line, exec)
6:  end for
7:  Q の各要素を，exec の値の近さで分割 (閾値 T) → G_i|_{i=1..N}
8:  for i = 1 to N do
9:        for each (address, d_line, exec) in G_i do
10:             A ← GetDependingAddresses( address, d_line, FS )
11:             A の各要素を，アドレスが連続しているかどうかで分割→M_m|_{m=1..L}
12:             for j = 1 to L do
13:                   K_j←null
14:                   M_j の各要素(addr, tr)を addr 順にソートし，TR[tr]から，addr を
                      参照した時に取得された値を K_j に連結
15:                   K ← K ∪ { K_j }
16:             end for
17:       end for
18: end for
19: return K
```

図 6 鍵候補特定アルゴリズム

ただし，このような方式で特定したメモリ領域には，暗号化鍵の他に，暗号化前のビーコンデータに由来するデータも含まれるため，鍵の特定にはより多くの試行が必要になると考えられる．

より困難な課題として，マルウェアが制御サーバから実行時に鍵を受け取って利用する場合が考えられる．そのような場合に対応するためには，通信記録装置に記録された通信ログを順にマルウェアに入力していき，初期段階にマルウェアが受信したメッセージから取得されたデータのうち，ビーコンの暗号化に使用したデータと関連のあるデータを求めていく必要がある．

4.2 データ依存関係情報のサイズ

本提案方式では，依存関係の追跡のために，各実行トレース情報に対し，その時点での依存関係情報を保持している．そのため，実行トレース情報ログの，後半になる

```
GetDependingAddresses(addr, tr, FS)
入力:
   addr    :メモリアドレス
   tr      :実行トレース行番号
   FS      :データ依存関係履歴情報
戻り値:
   A       : 依存アドレス集合 ⊆ アドレス × トレース行番号

1:   A ← φ
2:   F ← FS[ line = tr ] // FS の中で，line=tr である要素を返す
3:   D ← { ( m, i ) | (s, m, i) ∈ F 内の依存関係情報 ∧ s = addr }
4:   if D = φ then
5:       A ← (addr, tr)
6:   else
7:       for each (m,i) in D do
8:           A ← A ∪ GetDependingAddresses( m, i, FS )
9:       end for
10:  end if
11:  return A
```

図 7 依存関係チェーンのルートにあるメモリアドレスの抽出

ほど大量の依存関係情報を保持しなければならない．これを改善するためには，各実行トレース情報に対しては，そのトレースに対応するマシン語命令によって更新された依存関係情報だけを保持する方法が考えられる．

しかし，このような記録方式を適用した場合，あるメモリに対して依存関係のあるメモリを特定する際に，データ依存関係情報全体を調べる必要が出てくるため，計算量が増大する可能性がある．

4.3 抽出された鍵候補の確認

本提案方式では，複数の鍵候補が抽出される可能性があり，そのうちのどれが本当の鍵なのかは手動で確認を行うことを想定している．この作業を自動化するには，復号に成功したかどうかを自動で判定する必要がある．

一つのアプローチとしては，復号を試行した結果得られたデータのランダム性を測定する方式が考えられる．これは，復号に成功して平文が得られた場合には，データ中にテキスト情報や何らかのデータ構造が現われるため，復号に失敗した場合に比べ，ランダム性が下がると予想されるためである．

ランダム性を測定する簡便な方法としては，既存の圧縮アルゴリズムを適用し，圧縮率を測定する方法が考えられる．元のデータ，および復号後のデータを各々共通の圧縮アルゴリズムを用いて圧縮し，復号後のデータを圧縮した場合の圧縮後データサイズが，復号前のデータを圧縮した場合の圧縮後データサイズと比較して顕著に（例えば 10%）以上小さくなった場合に復号に成功したとみなす，といった方法が有効と思われる．

5. まとめ

近年、防衛や重要インフラの機密情報を狙ったと思われる新種の標的型攻撃が海外，国内を問わず確認されており，官公庁や企業を中心に、重大な脅威として注目されている。このような標的型攻撃は，アンチウイルスソフト等，標的組織のセキュリティ対策を回避するよう巧妙に行われるため，検知・防止を行うことが難しく，攻撃が発覚したときには，実際に情報漏えい被害を受けてしまっている可能性が高い．

そのため，万一自組織が攻撃を受けてしまった場合，端末を隔離する等緊急的な対策を講じるのと並行して，実際にどのような情報が漏れてしまったのかを特定する被害特定作業も不可欠となるが，そのためにはマルウェアによってメッセージに施された暗号化を解き，平文データを復元する必要がある．

本稿では，標的型攻撃による情報漏えいが発生した際に，通信記録から漏えいした情報を特定するため，マルウェアが使用した暗号化鍵を特定する手法を提案した．本方式では，マルウェアに復号処理を行わせ，その処理中にアクセスされたマルウェア内部のメモリ間の依存関係を解析することで鍵の候補となるデータの抽出を行う．

本方式では，マルウェア内で使用される暗号化鍵と復号鍵が同一であり，かつハードコードされている場合にのみ適用可能である．それ以外の場合に対する対処法について考察を行った．また，データ依存関係情報のサイズ削減方式，抽出された鍵候補の確認方式についても検討した．

今後，本方式に対する試作，評価を実施し，本方式の有効性について確認していく予定である．

参考文献

1) 独立行政法人 情報処理推進機構. 「新しいタイプの攻撃」の対策に向けた設計・運用ガイド 改訂第 2 版, 2011
2) Symantec. The Nitro Attacks Stealing Secrets from the Chemical Industry, 2011
3) Symantec. W32.Duqu The precursor to the next Stuxnet, 2011
4) PEiD, http://www.peid.info/
5) K. Kaukonen and R. Thayer. A Stream Cipher Encryption Algorithm "Arcfour".The Internet Society, 1999.
6) F.Gröbert,C.Willems, and T.Holz. Automated Identification of Cryptographic Primitives in Binary Programs. 14th International Symposium on Recent Advances in Intrusion Detection (RAID),2011
7) A.Shamir, and N.van Someren. Playing “hide and seek”with stored keys. Lecture Notes in Computer Science 1648 (1999),118–124.

Vol.2012-DPS-150 No.6
Vol.2012-CSEC-56 No.6
2012/2/29

8) J. Halderman, et.al. Lest We Remember: Cold Boot Attacks on Encryption Keys. In USENIX Security Symposium, pages 45–60, 2008.
9) Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197(FIPS PUB 197),2001
10) Data Encryption Standard (DES), Federal Information Processing Standards Publication 197(FIPS PUB 46-3),1999
11) QEMU, http://wiki.qemu.org/Main_Page
12) BOCHS, http://bochs.sourceforge.net/
13) IA-32 インテル® アーキテクチャー・ソフトウェア・デベロッパーズ・マニュアル http://www.intel.com/jp/download/index.htm