動的 API 検査方式によるキーロガー検知方式

松本隆明[†] 高見知寛[††] 鈴木功一[††]
馬場達也[†] 前田秀介[†]
水野忠則[†††] 西垣正勝[†††]

本論文では，キーボード入力を取得するというキーロガーの挙動を定式化し，キーボード入力に用いられる API の使用を検出することでキーロガーの検知を行う方式を提案する．API の使用を検出する機能を付加した検査用 DLL をプログラムにロードさせたうえで実行させることが本方式の特徴である．本論文では本方式の有効性を示すための実験を行い，本方式を Windows OS 上に実装したときの検知率と誤検知率，オーバヘッドについて評価する．

A Keylogger Detection Using Dynamic API Inspection

Takaaki Matsumoto,[†] Tomohiro Takami,[††] Koichi Suzuki,[††]
Tatsuya Baba,[†] Shusuke Maeda,[†] Tadanori Mizuno[†††]
and Masakatsu Nishigaki[†††]

This paper proposes a keylogger detection scheme by monitoring APIs employed by keylogger to capture user's keyboard input. API inspection is one of efficient ways for keylogger detection, since the use of keyboard-input-related APIs is a typical behavior found in keyloggers. To achieve this, we create a DLL which can detect the use of these APIs. By executing a program with the DLL, we can check whether the program includes any of these APIs or not. This paper carries out basic experiments to evaluate its detection rate, false detection rate and overhead.

1. はじめに

近年，スパイウェアと呼ばれる悪質なプログラムがインターネット上を横行し，その被害が増加している[2]．本論文では，スパイウェアの中でも近年オンラインバンクの不正送金事件[1] など深刻な被害を引き起こしている「キーボード入力を取得するキーロガー」に焦点を当てる．

キーロガーの被害が急速に拡大している原因の 1 つとして，スパイウェアに対するユーザの認識の低さがあげられる．キーロガーがインストールされている可能性のある環境（インターネットカフェなど）でのオンラインバンクやクレジットカードによるオンラインショッピングの利用により，キーロガーの被害に遭うケースが少なくない．さらにスパイウェアは，ウイルスのように感染活動を行わず，ユーザに気付かれないように行動するという特徴を持つ．そのためユーザは自分の PC がキーロガーに感染していることにすら気付かず，それがキーロガーによる被害が拡大する要因となっている．

このような現状に対して，各アンチウイルスベンダはキーロガー（をはじめとしたスパイウェア）検知機能を搭載したウイルス対策ソフトウェアを開発している．その多くはウイルス検知の場合と同様にパターンマッチング法を用いたものが主流となっている[2] が，最近になって，キーロガーの挙動をとらえることにより，キーロガーらしい振舞いをするプログラムを検出する方法も採用され始めている[3)~5]．この方法であれば，パターンマッチング法では検知することができない未知キーロガーに対しても対処が可能である．

しかし著者らの調べた限り，現在までにキーボード入力を取得するキーロガーに対して，「キーロガーらしさ」の定義を明確に示した文献は見当らない．そこで本論文では，API ベースのキーロガーを対象に，キーロガーがキーボード入力を取得するにあたっての API

† 株式会社 NTT データ技術開発本部
R&D Headquarters, NTT Data Corporation
†† 静岡大学大学院情報学研究科
Graduate School of Informatics, Shizuoka University
††† 静岡大学創造科学技術大学院
Graduate School of Science and Technology, Shizuoka University

の用例を定式化し，これを「キーロガーの挙動」として規定する．すべてのプログラムに対し，本論文で規定された挙動で API が使用されているか否かを検査することにより，キーロガーらしい振舞いを行うプログラムをリアルタイムで検知することが可能となる．具体的には，API の使用を検出する機能を付加した検査用 DLL をプログラムにロードさせたうえで実行させ，プログラム中での当該 API の使用を検査することによって，キーロガーを検知する．本論文では本方式の有効性を示すために，本方式を Windows OS 上に実装して実験を行い，検知率，誤検知率，およびオーバヘッドについて評価する．

2. キーロガーの特徴と既存の検知方式

2.1 キーロガーの特徴

本論文では，クライアント PC の OS として広く普及しており，キーロガーの被害が拡大している Windows に焦点を当て，Windows 環境下で動作するキーロガーを対象とする．

キーロガーとはユーザのキーボード入力を取得するスパイウェアであり，API ベースのキーロガーとカーネルベース（または Rootkit 型）のキーロガーに大別される[6]．前者は Windows のユーザモードで動作するタイプのキーロガー[6] であり，Windows に用意されているキーボード入力取得用 API を利用してユーザのキーボード入力を取得する．後者は Windows のカーネルモードで動作するタイプのキーロガー[7] であり，デバイスドライバやフィルタドライバとして動くことにより，ユーザのキーボード入力を取得する．

本論文では API ベースのキーロガーが検知対象である．

2.2 パターンマッチング法の限界

現在のキーロガー（をはじめとしたスパイウェア全般）の検知方式の主流はパターンマッチング法である．パターンマッチング法は，キーロガーの特徴的なプログラムコードをパターンとしてデータベースにあらかじめ登録しておき，検査対象となるプログラムと比較することでキーロガーの検知を行う方式である[8]．この方式は，データベースにパターンが登録されている既知のキーロガーに対しては確実かつ容易に検知することができるが，未知のキーロガーを検知することができないという問題が残る．

またウイルスの場合であれば，広範囲に感染活動を行うため比較的検体が入手しやすく，アンチウイルスベンダもウイルスのパターンを生成することができる．そのため，ウイルス対策とパターンマッチング法は比較的相性が良いともいえる．事実，IPA の調査では，ウイルス届出件数はここ数年増加傾向にあるものの，実害のあったケースが毎年減少してきている理由を，ウイルス対策ソフトウェアの導入の増加による効果であると分析している[9]．しかしスパイウェアはウイルスと異なり，感染活動を行わないため，パターンを生成するための検体を入手することが困難である[10]．特にキーロガーはその目的から特定のユーザやグループを狙う場合も多く，その場合検体の事前入手は実質不可能である．

2.3 キーロガーの挙動に基づく検知方式の現状

最近になって，キーロガーの挙動をとらえることにより，キーロガーらしい振舞いをするプログラムを検出する方法も採用され始めている[3)~5)]．この方法であれば，パターンマッチング法では検知することができない未知キーロガーに対しても対処が可能である．

しかし，著者らの調べた限り，現在までに「キーロガーらしさ」の定義を明確に示した文献は見当らない．事実，未知キーロガー検知機能を有するとされる製品である Keylogger Hunter[3]，Keylogger Stopper[4]，Anti-Keylogger[5] を用いて各種キーロガーの検知を実施したところ，**表 1** の結果が得られた．ここで，○ は「検知」，× は「非検知」，△ は「誤検知（Casper のプロセスではなく，explorer.exe のプロセスがキーロガーであると検知された）」を示す．このように，上記の製品群においても「キーロガーらしさ」の挙動を完全に定式化できておらず，検知漏れが発生している．

表 1 既存製品のキーロガー検知実験
Table 1 Experimental results for keylogger detection with the existing products.

検査対象	Spybot	Keylogger Hunter 2.1	Keylogger Stopper 2.0	Anti-Keylogger Elite 3.3.3
SpyAnywhere [12]	○	○	○	○
Perfect Keylogger Lite [13]	○	○	○	○
Activity Logger [14]	○	○	○	○
XPCSpy [15]	×	○	○	○
きいろがぁ [16]	×	×	×	×
キーロガー [17]	×	×	×	×
Parasite [18]	×	○	○	○
WingKEY [19]	×	○	○	○
キーのログをとる者 [20]	×	×	×	×
Casper [21]	×	×	×	△

表 2 既存製品の誤検知実験
Table 2 Experimental results for false detection with the existing products.

検査対象	種類	Spybot	Keylogger Hunter 2.1	Keylogger Stopper 2.0	Anti-Keylogger Elite 3.3.3
Mozilla Firefox 2.0	WEB ブラウザ	×	×	×	×
Microsoft Word 2000	ワードプロセッサ	×	×	×	×
Microsoft Excel 2000	表計算ソフト	×	×	×	×
Microsoft PowerPoint 2000	プレゼンテーションツール	×	×	×	×
Internet Explorer 6	WEB ブラウザ	×	×	×	×
Outlook Express 6	メーラ	×	×	×	×
Mozilla Thunderbird 1.5.0.7	メーラ	×	×	×	×
Orchis [22]	ランチャツール	×	○	○	×
xkeymacs [23]	キーバインドツール	×	○	○	○
AltIME [24]	キーバインドツール	×	○	○	○

参考までにこれらの製品群の誤検知実験の結果を**表 2** に記す．ここで，○ は「誤検知（キーロガーとして検知）」，× は「非検知」を示す．また，表 1 および表 2 においては，パターンマッチング法の代表として Spybot—Search & Destroy 1.4[11]（2007 年 3 月 14 日現在の定義ファイル）による実験結果も併記した．

3. API ベースのキーロガーの挙動の定式化

3.1 キーボード入力を取得する挙動の規定

Windows 環境下において，API を利用してキーボード入力を取得する方法は，著者らが確認した限りでは 2 つ存在する．キー判定 API を用いる方法と，フック API を用いる方法である．以下，これらの仕組みとその検出方法について説明する．

3.1.1 キー判定 API を用いる方法

キー判定 API である GetAsyncKeyState は，呼び出し時に指定したキーが押されているか，また前回の呼び出し時以降に指定したキーが押されていたかどうかを判定する API である．GetAsyncKeyState は，単に指定したキーが押されているか否かを判定する API であるため，任意のプロセスへのキーボード入力を取得することが可能である．

キーロガーがキー判定 API を用いてキーボード入力を取得する場合，ID やパスワードなどを盗み出すために，少なくとも A〜Z と 0〜9 のキーボード入力を取得しなければならない．またキーロガーは，キーロガー自身のプロセスへのキーボード入力だけでなく，他プロセスへのキーボード入力も取得する．そのため，他プロセスがキーボードフォーカスを所持している場合でも，キーロガーは GetAsyncKeyState を用いてキーボード入力を取得できなければならない．

そこで本方式では，キー判定 API を用いたキーロガーの挙動を以下のように規定する．

【挙動 1：キー判定 API に関する挙動】

他プロセスがキーボードフォーカスを所持しているときにも，GetAsyncKeyState によって A〜Z，0〜9 のすべてのキーボード入力を取得する．

3.1.2 グローバルフック API を用いる方法

フックとは，Windows OS が各プロセスへ送信するメッセージを SetWindowsHookEx という API を用いて取得する方法であり，自身のプロセスへのメッセージのみを取得するローカルフックと，全プロセスへのメッセージを取得することが可能なグローバルフックが存在する[25]．

たとえば，ユーザが閲覧している Web サイトにログインするにあたって入力する ID やパスワードをキーロガーがフックを用いて盗むためには，Web ブラウザへのキーボード入力メッセージを取得する必要がある．キーロガーから見て Web ブラウザは他プロセスであるため，キーロガーは必然的にグローバルフックを用いることになる．なお，フックには様々なタイプが存在し，その中にはキーボード入力メッセージをフックできないタイプも存在する．キーロガーは，他プロセスへのキーボード入力メッセージを取得することが目的であるため，キーボード入力メッセージをフックできるフックタイプを指定する必要がある．

そこで本方式では，フックを用いるキーロガーの挙動を以下のように規定する．

【挙動 2：グローバルフック API に関する挙動】

キーボード入力メッセージをフックできるフック

タイプのいずれかを指定し，SetWindowsHookExによるグローバルフックを用いてキーボード入力を取得する．

3.2 他プロセスに寄生してキーボード入力を取得する挙動の規定

3.1 節では，キーロガー自らが他プロセスへのキーボード入力を取得する方法を説明した．しかしキーロガーは，自身の存在をユーザに知られないように，他プロセスに寄生してキーロギングを行う場合もある．

3.2.1 他プロセスに寄生する方法

Windows 環境下において，他プロセスに寄生する方法は，著者らが確認した限りでは 2 つ存在する．CreateRemoteThread を用いる方法とSetWindowsHookEx を用いる方法である[26]．

CreateRemoteThread は，他プロセスのメモリ空間にスレッドを生成する API であり，この API を用いれば，たとえばキーボード入力を取得するスレッドを他プロセスに生成することも可能である．

SetWindowsHookEx は 3.1.2 項で示したとおり，フックを行うための API であるが，グローバルフックを行う場合には，フック先のプロセスすべてにフックプロシージャ（フックしたメッセージを処理するルーチン）が含まれている DLL をアタッチする．よって，キーボード入力を取得するためのルーチンを，フックプロシージャを含む DLL に仕込むことにより，任意のプロセスにキーロガー機能を有する DLL をアタッチさせることが可能となる．

そこで本方式では，キーロガーの寄生の挙動を以下のように規定する．

【挙動 3：寄生 API に関する挙動】

CreateRemoteThread または SetWindowsHookEx のグローバルフックを発行する．

3.2.2 寄生先プロセスにキーロガー行為を行わせる方法

いったん他プロセスに寄生すれば，キーロガーはそのプロセスの一部として動作することが可能となるため，寄生したプロセスへのキーボード入力を取得することが OS により許可される．また，寄生したプロセス以外のプロセスへのキーボード入力についても取得したい場合は，寄生したプロセスに 3.1 節に示した 2 つの方法のいずれかを行わせることで，他のプロセスへのユーザのキーボード入力を取得することが可能となる．

(a) 寄生先プロセスへのキーボード入力を取得する方法

寄生したプロセスへのキーボード入力を取得する方法としては，GetKeyState を用いる方法，GetKeyboardState を用いる方法，そして SetWindowsHookEx によるローカルフックを用いる方法があげられる．GetKeyState は自プロセスに対して，あるキーが押されているか否かを判定する API である．GetKeyboardState は自プロセスに対して，すべてのキーの状態（押されているか否か）を取得できる API である．SetWindowsHookEx によるローカルフックは，3.1.2 項に示したとおり，自プロセスへのメッセージのみを取得する方法である．

そこで本方式では，寄生先プロセスのキーボード入力を取得するキーロガーの挙動を以下のように規定する．

【挙動 4：寄生先プロセスへのキーボード入力を取得する API に関する挙動】

寄生先プロセスが自プロセスへのキーボード入力を取得する．

(b) 寄生先プロセスで他プロセスへのキーボード入力を取得する方法

寄生したプロセス以外のプロセスへのキーボード入力を取得するには，寄生先プロセスにおいて 3.1.1 項もしくは 3.1.2 項で示した方法を実行すればよい．

そこで本方式では，全プロセスへのキーボード入力の取得を寄生先プロセスに代行させるというキーロガーの挙動を以下のように規定する．

【挙動 5：寄生先プロセスで他プロセスへのキーボード入力を取得する API に関する挙動】

寄生先プロセスが挙動 1 もしくは挙動 2 の行動を示す．

3.2.3 寄生を繰り返す方法

他プロセスへの寄生は多重に行うことも可能である．巧妙な攻撃者は，「寄生先プロセスにさらに別のプロセスへの寄生行為を行わせ，そのプロセスにキーロガー行為を行わせる」という方法を必要な回数だけ繰り返すことも考えられる．このような挙動を検出するためには，検査対象プログラムにおける寄生行為を再帰的に検査していく必要がある．

そこで本方式では，寄生行為を繰り返すというキーロガーの挙動を以下のように規定する．

【挙動 6：多重寄生に関する挙動】

寄生先プロセスが挙動 3 の行動を示す．

3.3 キーロガーの挙動の定式化

挙動 5 は，キーロガーが寄生活動（挙動 3）を行った際における挙動 1 および挙動 2 の再帰検査である．また，挙動 6 が見受けられた際には，寄生先プロセスを基点としてさらに挙動 3〜挙動 6 の検査を繰り返す必要がある．これらに注意すると，3.1 節，3.2 節に示

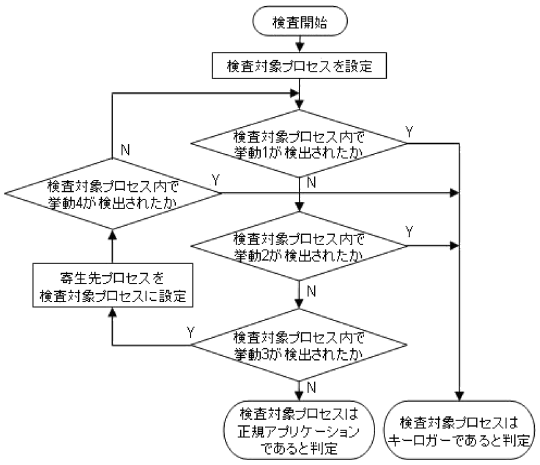


図 1 キーロガー検知フロー
Fig. 1 Flow of keylogger detection.

した挙動 1〜挙動 6 に基づくキーロガー検査は**図 1** のフローチャートのように定式化される．

4. キーロガー検知システムの実装

4.1 実 装 要 件

本方式を実装するためには，以下の要件を満たす必要がある．

(1) 図 1 のフローチャートに従ってキーロガー検知を行う．
(2) 任意の API の呼び出しを監視可能．
(3) 検査対象プログラムの改変は不要．
(4) リアルタイムにキーロガーの検知が可能．
(5) 全プログラムに対して起動直後からのキーロガーの検知が可能．

(2) は，キーボード入力を取得する API の挙動ならびに寄生を行う API の挙動を監視する本方式において必要となる要件である．(3)〜(5) は，キーロガー検知方式として基本的に備えなければならない要件である．(1) に関しては，本論文ではコンセプトの提案に重きをおいているため，実装の手間に鑑み，挙動 4 と挙動 6 の検出処理に関しては実装を割愛することとした．すなわち本実装におけるキーロガー検知フローは，以下の挙動 A〜C の検出となる．

【挙動 A】 他プロセスがキーボードフォーカスを所持しているときにも，GetAsyncKeyState によって A〜Z，0〜9 のすべてのキーボード入力を取得する（挙動 1）．

【挙動 B】 キーボード入力メッセージをフックできるフックタイプのいずれかを指定し，SetWindowsHookEx によるグローバルフックを用いてキーボード入力を取得する（挙動 2）．

【挙動 C】 CreateRemoteThread, SetWindowsHookEx を用いて他プロセスに寄生した（挙動 3）後，寄生先プロセスが挙動 1 もしくは挙動 2 を行う（挙動 5）．

4.2 検査用 DLL の作成

本実装では，4.1 節に示した (2)〜(4) の要件を満たす方式として，Microsoft Research の提供するライブラリである Detours[27] を採用する．

Detours は，x86 マシン上で任意の Win32 関数にインターセプト用のコードを挿入するための API フックライブラリ関数群である[28]．API α をインターセプトするにあたり，Detours はメモリ上の API α の先頭アドレスに，インターセプト用コードの先頭アドレスへの無条件ジャンプ命令を挿入する．プログラムの中で API α が呼び出されると，無条件ジャンプ命令によって制御がインターセプト用コードに移行するため，API α をインターセプトすることが可能となる．インターセプト用コードの実行が終了すると，API α に制御が渡され，API の本来の動作が実行される．

本論文では, Detours を用いて挙動 A〜C に関連する 3 つの API（GetAsyncKeyState, SetWindowsHookEx, CreateRemoteThread）をインターセプトし，それぞれの API においてキーロガーらしい挙動（挙動 A〜C）が検出されるか否かを監視することでキーロガーを検知するための検査用 DLL を実装する．なお，検査用 DLL の開発環境には Microsoft Visual C++ 6.0 を用いた．

図 2 が，Detours ライブラリを用いることにより作成したキーロガー検査用 DLL である．ここで，DetourAttach(A, B) は Detours のライブラリ関数であり，第 1 引数 A に指定された API をインターセプトし，第 2 引数 B に指定されたインターセプト用のコードを挿入する．検査用 DLL は，自身がプロセスにロードされた際に，関数 DetourAttach(A, B) を実行することにより GetAsyncKeyState, SetWindowsHookEx, CreateRemoteThread の 3 つの API をインターセプトする．また検査用 DLL は，自身がプロセスからアンロードされた場合に，インターセプトを自動的に解除するように実装されている．インターセプトの解除には，Detours のライブラリ関数である DetourDetach(A, B) を用いた．DetourDetach(A, B) は，第 1 引数 A に指定された API から，第 2 引数 B に指定されたインターセプト用のコードを取り除く関数である．

そして**図 3** が各 API のインターセプト用コードである．各 API のインターセプト用コードは，挙動 A〜C

```
検査用 DLL　{
    if （DLL_load）{　　//　DLL ロード時
        DetourAttach ( Real_GetAsyncKeyState,
                Mine_GetAsyncKeyState ) ;
        DetourAttach ( Real_SetWindowsHookEx,
                Mine_SetWindowsHookEx ) ;
        DetourAttach ( Real_CreateRemoteThread,
                Mine_CreateRemoteThread ) ;
        DetourAttach ( Real_CreateProcess,
                Mine_CreateProcess );
    }　else if （DLL_unload ) {　//　DLL アンロード時
        DetourDetach ( Real_GetAsyncKeyState,
                Mine_GetAsyncKeyState ) ;
        DetourDetach ( Real_SetWindowsHookEx,
                Mine_SetWindowsHookEx ) ;
        DetourDetach ( Real_CreateRemoteThread,
                Mine_CreateRemoteThread ) ;
        DetourDetach ( Real_CreateProcess,
                Mine_CreateProcess ) ;
    }
}
```

図 2　検査用 DLL
Fig. 2　DLL for inspection.

のいずれかを満たす場合にアラートが発生するように実装されている．また，処理を正常に継続させるために，各インターセプト用コードの最後で本物の当該 API を呼び出している．

上記の検査用 DLL を検査対象プロセスにロードさせれば，これら 3 つの API の挙動が監視されるため，対象プロセスがキーロガーか否かを検査することが可能となる．なお，SetWindowsHookEx と CreateRemoteThread の発生は他プロセスへの寄生（の可能性があること）を意味するため，検査対象プロセスでこれらの API が呼び出された場合には，寄生先プロセスに検査用 DLL をさらにアタッチさせる必要がある．この機能を有する関数として AttachParasiteProcess 関数を自作した．

また，4.1 節に示した (5) の要件に関しては，今回は，Windows の起動後にユーザのクリックにより実行されるすべてのアプリケーションに対して，アプリケーションの起動直後からのキーロガーの検知を行うという形態の実装とした．

ユーザのクリックにより実行されるアプリケーションは，Windows のファイルマネージャである explorer.exe が子プロセス生成用 API である

```
Mine_GetAsyncKeyState ( 引数 ) {
    if ( 他プロセスがキーボードフォーカスを所持 ) {
        if （ 引数 ==A〜Z，0〜9 )
            alert ;
    }
    return Real_GetAsyncKeyState ( 引数 ) ;
}

Mine_SetWindowsHookEx ( 引数 ) {
    if （ 引数 == グローバルフック ) {
        if ( キーボード入力メッセージ取得用
            フックタイプ )
            alert ;
        else
            // 寄生先プロセスに DLL をアタッチ
            AttachParasiteProcess( 検査用 DLL );
    }
    return Real_SetWindowsHookEx ( 引数 ) ;
}
Mine_CreateRemoteThread ( 引数 ) {
    // 寄生先プロセスに DLL をアタッチ
    AttachParasiteProcess( 検査用 DLL );
    return Real_CreateRemoteThread ( 引数 ) ;
}
```

図 3　各 API のインターセプト用コード
Fig. 3　Interception code for each API.

```
Mine_CreateProcess ( 新規プロセス ) {
    return DetourCreateProcessWithDll(新規プロセス, 検査用 DLL);
}
```

図 4　CreateProcess のインターセプト用コード
Fig. 4　Interception code for CreateProcess.

CreateProcess を呼び出すことにより実行される．たとえばユーザがデスクトップ上の Internet Explorer のアイコンをダブルクリックした場合には，explorer.exe は内部で CreateProcess を呼び出し，Internet Explorer を子プロセスとして生成することにより，Internet Explorer が実行されるという仕組みになっている．

そこで，CreateProcess に対しても Detours によるインターセプトを行い，explorer.exe が生成する子プロセスに対して，子プロセスが生成される瞬間に検査用 DLL をロードさせるようにする．CreateProcess のインターセプト用コードを**図 4** に示す．

ここで，DetourCreateProcessWithDll(A，B) は Detours のライブラリ関数であり，第 2 引数 B に指

定された任意の DLL をロードさせたうえで，第 1 引数 A に指定された新規プロセスを生成する．

4.3 検査用 DLL を用いてのキーロガー検知手順

本節では，検査用 DLL を用いてのキーロガー検知の手順を示す．

(1) Windows 起動時に，ユーザのクリックにより実行されるプログラムに検査用 DLL をロードさせるための前処理を実施：

Windows 起動時に，4.2 節の図 2〜図 4 に示したキーロガー検査用 DLL を explorer.exe にロードさせておけば，図 4 で説明した CreateProcess に関するインターセプトの機構によって，ユーザのクリックにより実行されるすべてのプログラムに検査用 DLL を自動的にロードさせることが可能である．

(2) ダブルクリックによりアプリケーションプログラムを実行：

explorer.exe にキーロガー検査用 DLL がロードされているため，ユーザのクリックにより起動されるすべてのアプリケーションプログラムには，その起動時に検査用 DLL が自動的にロードされる．検査用 DLL はアプリケーションプログラムを常時監視し，挙動 A〜C が発生した場合には，その時点でアラートをあげる．

ここで，手順 (1) においてキーロガー検査用 DLL を explorer.exe にロードさせるには，図 4 のインターセプトコードを利用して「Mine_CreateProcess (explorer.exe) の API コールを発行するプログラム」を作成しておき，(i) いったん explorer.exe を終了したうえで，(ii) Mine_CreateProcess(explorer.exe) を発行して，explorer.exe をキーロガー検査用 DLL がロードされた形で再起動させればよい．この作業は，一般ユーザ権限でも実行可能である．また，ユーザが各自のスタートアップメニューにこの作業を登録することで，本検知システムを常駐させることも可能である．さらに，管理者権限を有するユーザであれば，この作業を Windows のレジストリの Run エントリに登録することもできる．たとえばインターネットカフェの管理者が，カフェ内のパソコンに本検知システムを導入（検査用 DLL を explorer.exe にロードさせる作業をレジストリに登録）しておけば，カフェ利用者をキーロガーの脅威から保護することが可能となる．

5. 実　　験

本章では，キーロガー検知実験を行い，4 章で実装した検知システムの可用性を示す．また，規定した挙動によって正規のプログラムがキーロガーとして検知されてしまう可能性もあるため，誤検知実験を行い，その結果についても考察する．さらに，実装した検知システムのオーバヘッドについても調査する．

表 3 キーロガー検知実験
Table 3 Experimental results for keylogger detection.

検査対象	挙動 A	挙動 B	挙動 C
SpyAnywhere	×	○	×
Perfect Keylogger Lite	×	○	×
Activity Logger	×	○	×
XPCSpy	×	○	×
きいろがぁ	○	×	×
キーロガー	○	×	×
Parasite	×	○	×
WingKEY	×	○	×
キーのログをとる者	○	×	×
Casper	×	×	○

5.1 キーロガー検知実験

4 章で実装した検知システムを用い，実際にキーロガーの検知実験を行った．ただしキーロガーは検体の入手が非常に困難であるため，実験には商用のキーロガーやインターネット上で公開されているキーロガーを用いた．なお本実験は，物理的に隔離されたネットワーク内の Windows 2000 Professional SP4 がインストール済みの PC 上で行った．本実験に使用したキーロガーとその実験結果を**表 3** に示す．ここで，○ は「検知」，× は「非検知」を示す．

表 3 から，今回の実験で用いたすべてのキーロガーに関して，本論文で規定したキーロガーらしい挙動（挙動 A，挙動 B，挙動 C のいずれか）が確認された．このことから，本方式がキーロガー検知に有効であることが確認された．

【考察】

本実装では，挙動 4 と挙動 6 の実装を割愛したが，本実験に用いたキーロガーにおいてはすべての検知が可能であった．しかし，将来的には挙動 4 または挙動 6 を利用したキーロガーが出現する可能性がある．よって，今後は図 1 のフローチャートに従い，3.1 節，3.2 節に示した挙動 1〜挙動 6 のすべてを監視する形態のキーロガー検知システムを実装していく必要がある．

5.2 誤検知評価実験

正規のアプリケーションを用いて，本システムの誤検知実験を行った．本実験に用いるアプリケーションには，Microsoft Office 製品や Web ブラウザ，メーラなどのアプリケーションに加え，誤検知の可能性の

表 4 誤検知評価実験
Table 4 Experimental results for false detection.

検査対象	種類	挙動 A	挙動 B	挙動 C
Mozilla Firefox 2.0	WEB ブラウザ	×	×	×
Microsoft Word 2000	ワード プロセッサ	×	×	×
Microsoft Excel 2000	表計算ソフト	×	×	×
Microsoft PowerPoint 2000	プレゼンテーション ツール	×	×	×
Internet Explorer 6	WEB ブラウザ	×	×	×
Outlook Express 6	メーラ	×	×	×
Mozilla Thunderbird 1.5.0.7	メーラ	×	×	×
Orchis	ランチャツール	×	○	×
xkeymacs	キーバインド ツール	×	○	×
AltIME	キーバインド ツール	×	○	×

あるアプリケーションを重点的に選択した．しかしこれらのアプリケーションは多機能であり，すべての機能を網羅して実験を行うことは非常に困難である．そこで本実験では，基礎実験として各アプリケーションの基本機能（WEB ブラウザであればブラウジング，メーラであればメール送受信など）を中心に実行した場合に対してのみの挙動監視となっている．なお本実験は，物理的に隔離したネットワーク内の Windows 2000 Professional SP4 がインストール済みの PC 上で行った．本実験に使用したアプリケーションとその結果を**表 4** に示す．ここで，○ は「誤検知（キーロガーとして検知した）」，× は「非検知」を示す．

表 4 から，基本的な機能を使用する限りの検査においては，一部のアプリケーションを除いて正規のアプリケーションを本方式によって誤検知することはないことが確認された．一部のアプリケーションに挙動 B が確認されたため，誤検知が発生した．誤検知が発生したアプリケーションについては，以下でその理由を示す．

xkeymacs と AltIME は，グローバルフックを用いてキーボード入力を取得しキーバインドを置換するアプリケーションである．また Orchis はランチャ（プログラムのショートカットの管理ツール）であり，特定のキーを押下した場合にランチャがアクティブ化する機能が実装されている．これらの機能を実現するにあたってグローバルフックを用いてキーボード入力を取得していたため，本方式ではこの 3 つのアプリケーションをキーロガーとして誤検知した．

表 5 キーロガーらしい挙動の検出に要するオーバヘッド
Table 5 Overhead of checking keylogger behavior.

	オリジナル [μs]	検査用 DLL 付き [μs]	オーバーヘッド [μs]
GetAsyncKeyState の呼び出し	2.9	15.8	12.9
SetWindowsHookEx の呼び出し	2.8	3489.4	3486.6
CreateRemoteThread の呼び出し	178.1	191.2	13.1

【考察】

本実験において誤検知となった xkeymacs，AltIME，Orchis においては，実際にキーロガーとしての機能を果たしうる可能性を有しているわけであるので，ユーザにその旨を通知することは有用であるという考え方もできる．

また，このような正規アプリケーションとキーロガーを切り分けるために，さらに詳細な「キーロガーらしい挙動」を規定することも必要であろう．たとえば xkeymacs や AltIME などは，SetWindowsHookEx によりユーザのキーボード入力を取得しているが，その情報を外部へ送信するといった挙動は行わない．このため，取得したキーボード入力を「外部に送信する」という動作をキーロガーらしい挙動として規定すれば，誤検知を低減させることができると考えられる．

5.3 オーバヘッド

本検知システムのオーバヘッドを測定した．オーバヘッドの測定に用いた実験環境は，Windows をインストールした直後の PC ではなく，ある程度の期間，ユーザが実際に使用していた Windows PC を用いた．これは，測定環境をより実環境に近づけるためである．測定に用いた Windows PC のスペックは，OS：Windows 2000 Professional SP4，CPU：Athlon 900 MHz，メモリ：128 MB である．

ここでは特に，監視対象プロセスにおいて挙動 A〜C を検査するために要するオーバヘッドを測定する．オーバヘッドは，GetAsyncKeyState，SetWindowsHookEx，CreateRemoteThread の各 API に対して，オリジナルの API を呼び出した際の処理時間と検査用 DLL 付き API を呼び出した際の処理時間の差として求めた．各 API に対して，それぞれを単独で 10,000 回呼び出したときの処理時間の平均とそのオーバヘッドを**表 5** に示す．

表 5 から，最大でも約 3.5 ミリ秒のオーバヘッドしか生じていないことが確認された．

【考察】

CreateRemoteThread は，ひとたびスレッドを生成

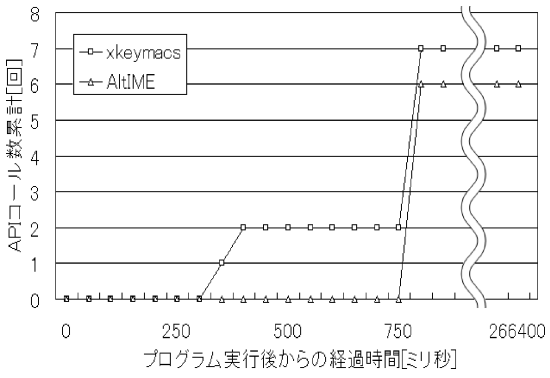


図 5 SetWindowsHookEx の API コールの発行数
Fig. 5 The number of API calls to SetWindowsHookEx.

すれば，そのスレッドが終了するまでスレッドは実行され続けるため，CreateRemoteThread を何度も発行し続ける必要はない．よって，CreateRemoteThread の API コールが呼ばれる頻度は比較的少ないことが一般的である．したがって，表 5 に示したオーバヘッドであれば，実害は小さいと考えられる．

GetAsyncKeyState は，API コールを発行し続けなければキーボード入力を取得し続けることができない．そのため GetAsyncKeyState の API コールが呼ばれる頻度は多くなるが，表 5 に示した程度のオーバヘッドであれば，実害は小さいと考えられる．

SetWindowsHookEx は，ひとたびフックを開始すれば，そのフックを解除するまでフック処理が継続するため，CreateRemoteThread と同様に，呼ばれる頻度が比較的少ない API コールであるといえる．ただし，表 5 から，SetWindowsHookEx の API コールにおけるオーバヘッドは，他の API コールと比べ桁違いに大きいことが分かる．このため，一般的なアプリケーションにおいて SetWindowsHookEx の API コールがどれくらいの頻度で発行されているのか調査を行った．調査に用いたアプリケーションは，5.2 節の実験で誤検知した xkeymacs と AltIME である．調査結果を**図 5** に示す．

図 5 から，SetWindowsHookEx の API コールはプログラム起動時に集約されており，またその発行総数も少ないため，表 5 に示したオーバヘッドであれば，実害は小さいことが示された．

6. ま と め

本論文では，キーボード入力を取得する API を悪用してキーロギング行為を行うキーロガーに対して「キーロガーらしい挙動」を定式化し，これを利用したキーロガー検知システムを実装した．検知実験，誤検知実験およびオーバヘッド測定実験を通じ，本システムがキーロガー検知に有効であることを示した．

本方式はパターンマッチングによらないキーロガー検知方式であるため，たとえ特定の相手を狙うようにカスタマイズされたキーロガーであっても，これを検知することが可能であると期待できる．

今後は，本論文では実装を割愛した挙動を加え，本方式を完全な形で実装したうえで，より広範囲にわたっての実用実験を繰り返す予定である．また，xkeymacs や AltIME などの誤検知を引き起こすことのない「キーロガーらしい挙動」の規定や，本論文では検知対象から外していた他種のキーロガー，および，キーロガー以外のスパイウェアに対する「マルウェアらしい挙動」の規定に関しても検討を行っていきたい．

謝辞 NTT データ東川淳紀様，情報通信研究機構安藤類央様には方式の実装に関する情報を提供していただいた．ここに深く謝意を表する．また，本研究は一部，(財) セコム科学技術振興財団の研究助成を受けた．ここに謝意を表する．

参 考 文 献

1) ITmedia: スパイウェアによる不正送金被害が拡大，みずほ銀行やジャパンネット銀行でも (2005/07/06). http://www.itmedia.co.jp/enterprise/articles/0507/06/news024.html
2) 与那原亨，大谷尚通，馬場達也，稲田 勉：トラフィック解析によるスパイウェア検知の一考察，情報処理学会研究報告，Vol.2005, No.70, 2005-CSEC-30, pp.23–29 (2005).
3) Keylogger Hunter. http://www.styopkin.com/keylogger_hunter.html
4) Keylogger Stopper. http://www.chithai.com/keystop.htm
5) Anti-Keylogger Elite. http://www.remove-keyloggers.com/index.php
6) 愛甲健二：ハッカー，プログラミング大全—攻撃編，DATA HOUSE (2006/04).
7) Symantec.com, Spyware.InvisibleKey.B. http://www.symantec.com/smb/security_response/writeup.jsp?docid=2004-110609-2102-99
8) トレンドマイクロ：ウイルス検出技術. http://www.trendmicro.com/jp/security/general/tech/overview.htm
9) 情報処理推進機構：2005 年のコンピュータウイルス届出状況 (2006/01/10). http://www.ipa.go.jp/security/txt/2006/documents/2005all-vir.pdf
10) アットマークアイティ：急速に広がるスパイウェアの脅威 (2005/11/30). http://www.atmarkit.co.jp/fsecurity/special/

86antispy/antispy01.html
11) Spybot—Search & Destroy 1.4. http://www.spybot.info/en/
12) Symantec Security Response, Remacc. SpyAnywhere. http://www.symantec.com/region/jp/avcenter/venc/data/jp-remacc.spyanywhere.html
13) Symantec Security Response, Spyware.Perfect. http://www.symantec.com/region/jp/avcenter/venc/data/jp-spyware.perfect.html
14) Symantec Security Response, Spyware.ActivityLog. http://www.symantec.com/region/jp/avcenter/venc/data/spyware.activitylog.html
15) Symantec Security Response, Spyware.XpcSpy. http://www.symantec.com/region/jp/avcenter/venc/data/jp-spyware.xpcspy.html
16) Vector, きいろがぁ. http://rd.vector.co.jp/soft/win95/util/se322072.html
17) Vector, キーロガー. http://www.vector.co.jp/soft/win95/util/se334334.html
18) Vector, Parasite. http://www.vector.co.jp/soft/winnt/util/se327656.html
19) Vector, WingKEY. http://www.vector.co.jp/soft/winnt/util/se263226.html
20) Vector, キーのログをとる者. http://www.vector.co.jp/soft/win95/util/se369025.html
21) S-CENTER, Casper. http://www.s-center.net/index.php
22) Vector, Orchis. http://www.vector.co.jp/soft/win95/util/se127007.html
23) xkeymacs. http://www.cam.hi-ho.ne.jp/oishi/
24) AltIME. http://www.chombo.com/
25) Microsoft: SetWindowsHookEx. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/hooks/hookreference/hookfunctions/setwindowshookex.asp
26) Richter, J.（著），長尾高弘，株式会社ロングテール（訳）：Advanced Windows 改訂第 4 版，アスキー出版局 (2001/05).
27) Microsoft Research: Detours. http://research.microsoft.com/sn/detours/
28) Hunt, G.C. and Brubacker, D.: Detours: Binary Interception of Win32 Functions, *Proc. 3rd USENIX Windows NT Symposium*, pp.135–143 (July 1999).

(平成 18 年 11 月 27 日受付)
(平成 19 年 6 月 5 日採録)



松本　隆明（正会員）

1978 年東京工業大学大学院電子物理工学専攻修士課程修了．同年日本電信電話公社（現 NTT）入社．2003 年株式会社 NTT データ技術開発本部長．オペレーティングシステム，コンピュータアーキテクテャ，情報セキュリティに関する研究に従事．現在，NTT データ先端技術株式会社勤務．静岡大学客員教授．



高見　知寛

2006 年度静岡大学情報学部情報科学科卒業．現在，同大学大学院修士課程．情報セキュリティに関する研究に従事．



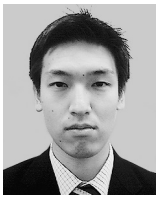
鈴木　功一

2005 年静岡大学情報学部情報科学科卒業．2007 年同大学大学院修士課程修了．同年東芝ソリューション株式会社入社．在学中，情報セキュリティに関する研究に従事．



馬場　達也（正会員）

1995 年慶應義塾大学理工学部電気工学科卒業．同年 NTT データ通信株式会社（現，株式会社 NTT データ）入社．同社技術開発本部にてネットワークセキュリティに関する研究に従事．現在，同社ビジネスソリューション事業本部勤務．著書に『マスタリング IPsec』（オライリー・ジャパン）がある．IEEE 会員．



前田　秀介（正会員）

2004 年電気通信大学大学院電気通信学研究科情報工学専攻博士前期課程修了．同年株式会社 NTT データ入社．現在，同社技術開発本部にてネットワークセキュリティに関する研究に従事．



水野 忠則（フェロー）

1945 年生．1969 年名古屋工業大学経営工学科卒業．同年三菱電機（株）入社．1993 年静岡大学工学部情報知識工学科教授．1996 年情報学部情報科学科教授．2006 年より創造科学技術大学院長．工学博士．情報ネットワーク，モバイルコンピューティング，ユビキタスコンピューティングに関する研究に従事．著訳書としては『コンピュータネットワーク』（日経 BP），『モダンオペレーティングシステム』（ピアソン・エデュケーション）等がある．電子情報通信学会，IEEE，ACM 各会員．情報処理学会フェロー．



西垣 正勝（正会員）

1990 年静岡大学工学部光電機械工学科卒業．1992 年同大学大学院修士課程修了．1995 年同博士課程修了．日本学術振興会特別研究員（PD）を経て，1996 年静岡大学情報学部助手．1999 年同講師，2001 年同助教授．2006 年より同大創造科学技術大学院助教授．2007 年より准教授．博士（工学）．情報セキュリティ，ニューラルネットワーク，回路シミュレーション等に関する研究に従事．