情報処理学会論文誌 Vol.60 No.2 696–705 (Feb. 2019)

推薦論文

可用性を考慮したプロセスの複製によるライブフォレンジック手法

山内 利宏[1,a)] 時松 勇介[1] 谷口 秀夫[1]

受付日 2018年7月11日, 採録日 2018年11月7日

概要：従来のハードディスクを調査対象とするデジタルフォレンジック手法は，ファイルシステムに痕跡を残さない攻撃に対処できない．また，ハードディスク上の証拠が改変されるのを防ぐためにシステムの電源断や処理の停止を必要とし，システムの可用性を低下させる．本論文では，可用性を考慮したプロセスの複製によるライブフォレンジック手法を提案する．提案手法は，プロセスを調査対象とし，対象プロセスの仮想記憶空間を複製して，低オーバヘッドでスナップショットを作成し，複製先のプロセスのメモリ上の証拠を収集する．このようにプロセスの複製処理において，プロセスのテキスト部などのメモリ間コピーを抑制することで，システムの可用性への影響を抑制しつつ，ファイルシステムに痕跡を残さない攻撃に対処できる．また，周期的に処理を実行するプロセスに提案手法を適用した場合の遅延時間を評価した結果，および提案手法の有効性について述べる．

キーワード：ライブフォレンジック，インメモリマルウェア，高可用性システム

Live Forensic Method Using Process Duplication to Maintain High System Availability

Toshihiro Yamauchi[1,a)] Yusuke Tokimatsu[1] Hideo Taniguchi[1]

Received: July 11, 2018, Accepted: November 7, 2018

Abstract: Most conventional digital forensic methods are designed to target hard disk drives, making them ineffective at detecting in-memory malware. In addition, in order to prevent a target system from changing the evidence on hard disk drives, it is necessary to shut down the system or stop its processing, reducing system availability. In this paper, we propose a live forensic method using process duplication to maintain high system availability. The proposed method duplicates the virtual address space of a target process for investigation, and obtains the relevant evidence from the duplicate. By reducing the occurrence of memory copy in the duplication process, it is possible to detect in-memory malware while retaining system availability. We describe the effectiveness of the proposed method, and furthermore, evaluate and report on the delay time when this method is applied to a periodically executing process.

Keywords: live forensics, in-memory malware, high availability system

1. はじめに

標的型攻撃をはじめとするサイバー攻撃に用いられる技術や手法は，高度化および複雑化の一途をたどっている [1], [2]．これにともない，サイバー攻撃によるインシデントの発生を未然に防ぐことは難しくなりつつあり，インシデントの発生を前提としたセキュリティ対策の重要性が高まっている．サイバー攻撃によるインシデント対応のセキュリティ技術の 1 つとして，デジタルフォレンジックがある．デジタルフォレンジックとは，システム上でインシ

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University, Okayama 700–8530, Japan
[a)] yamauchi@cs.okayama-u.ac.jp

本論文の内容は 2017 年 10 月のコンピュータセキュリティシンポジウム 2017（CSS2017）にて報告され，同プログラム委員長により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

デントが発生した際，不正者や犯罪者の特定，事故原因の究明，および責任の究明のため，システム内に残る電磁的証拠を調査する手段や技術を指す [3]．

しかし，ファイルシステムに攻撃の痕跡を残さない攻撃手法が増加している [4]．ハードディスクを調査対象とするデジタルフォレンジック手法は，この攻撃手法に対処できない．また，ハードディスクを調査対象とするデジタルフォレンジック手法は，証拠収集時にシステムの電源断を必要とし，システムの可用性を低下させる．

これに対処するには，システムを稼働させた状態でメモリ上の証拠をはじめとする揮発性の証拠を調査するデジタルフォレンジック手法（以降，ライブフォレンジック）が有用である．しかし，ライブフォレンジック手法においても，証拠収集時には証拠の改変を防ぐために処理の停止が必要となるため，証拠収集に要する時間によってはシステムが本来行うべき処理に影響を及ぼす恐れがある．たとえば，実メモリをすべてダンプして解析する手法があるものの，ダンプするサイズが大きくなるため，可用性への影響が大きい．このことから，可用性を重視するシステムでは，従来のデジタルフォレンジック手法をそのまま適用できない可能性がある．

可用性を重視するシステムにおけるデジタルフォレンジック手法の研究としては，制御システムに対する従来手法の有効性評価 [5], [6] や制御システムへの適用を想定した手法の提案 [7], [8] がある．ただし，これらはハードディスクを調査対象とする手法に焦点を当てており，可用性を重視するシステムにおけるファイルシステムに痕跡を残さない攻撃の対処は検討されていない．

そこで，上記の問題の対処として，可用性を考慮したプロセスの複製によるライブフォレンジック手法を提案する．提案手法は，プログラムの実行状態であるプロセスを調査対象とし，対象のプロセスの仮想記憶空間を複製して，低オーバヘッドでスナップショットを作成し，複製先のプロセスのメモリ上の証拠を収集する．提案手法は，可用性への影響を抑えるため，プロセスの複製処理においてプロセスのテキスト部などのメモリ間コピーを抑制し，複製にかかる処理時間を短縮する．これにより，対象プロセスの停止時間を抑制し，システムの可用性を維持することを目指す．また，提案手法は，複製要求時で一貫性のとれたプロセスの仮想記憶空間の複製を複数保持できる．さらに，対象プロセスを停止することなく，証拠を収集でき，可用性への影響を抑制できる．

本論文では，提案方式の設計と実現方式について述べ，周期的に処理を実行するプロセスを用いて提案手法を評価した結果を報告する．

2. 可用性を考慮したプロセスの複製によるライブフォレンジック手法

2.1 ライブフォレンジックの必要性

従来のサイバー攻撃では，有用な証拠の多くがファイルシステムに残っていたため，従来のデジタルフォレンジック手法の多くはハードディスクを調査対象としている．しかし，ファイルシステムに痕跡を残さない攻撃手法が増加していることから，ハードディスクの調査のみでは証拠が不十分な可能性がある．また，ディスクの大容量化，セキュリティチップを用いたディスクの暗号化，およびモバイル端末の普及といった要因もあり，従来のデジタルフォレンジック手法による証拠収集は難しくなってきている [9]．以上の要因から，ライブフォレンジック手法の適用により，システム上に存在する揮発性の証拠を調査する必要性が高まっている．

2.2 要件

これまでに述べた研究背景をふまえ，提案手法が満たすべき要件を以下に示す．

（**要件 1**） ファイルシステムに痕跡を残さない攻撃手法への対処

従来のデジタルフォレンジック手法の多くは，ハードディスク上の証拠を調査するため，ファイルシステムに痕跡を残さない攻撃手法による被害を受けた場合に有用な証拠を入手することができない．この攻撃手法に対処するためには，稼働中のシステムにライブフォレンジック手法を適用し，メモリ上の揮発性の証拠を調査する手法が有効である．

（**要件 2**） 対象となるシステムの可用性維持

従来のデジタルフォレンジック手法は，ハードディスク上の証拠の改変を防止するため，システムの可用性を低下させる．このため，可用性を重視するシステムには適用できない可能性がある．可用性を重視するシステムに適用するには，システムに与える影響をできる限り抑える必要がある．

また，前提として，提案手法の適用対象は，物理メモリのすべてをダンプする処理がシステムの可用性などに影響を与えるため適用できないシステムとする．

2.3 設計方針

本研究は，システムの可用性への影響を抑えつつ，メモリ上の痕跡を証拠保全する手法の確立を目的とする．また，可用性に影響を与えずに証拠保全と解析を実現するために，まずは，特定のプロセスの仮想記憶空間上の揮発性情報を保全する手法の実現を目指す．これは，物理メモリ全体を保全する手法は，主記憶全体を保全するため，その保存する記憶容量が大きく，システムを止めずに，また可用性への影響を抑えて証拠保全を行うことが難しいためである．

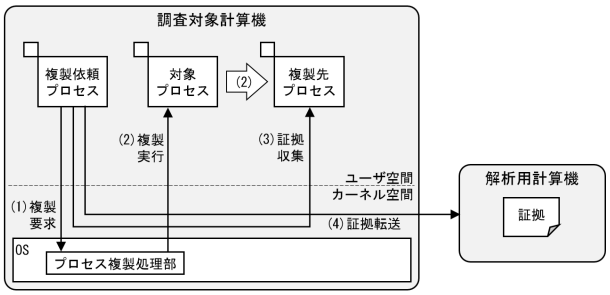

図 1 提案手法の全体像
Fig. 1 Overview of the proposed method.

そこで，本研究では，指定したプロセスのメモリ上の情報を取得し，保全するライブフォレンジック手法を提案する．また，証拠保全の観点から，取得するプロセスの状態は，ある時点における一貫性のとれたメモリ内容を保全する．つまり，メモリ上のデータの保全作業中に，未保全のデータが更新されない手法を提案する．これを実現するために，低オーバヘッドで仮想記憶空間上のメモリデータのスナップショットを作成する手法を検討する．これにより，保全処理開始時点における対象プロセスの状態を保全しておき，可用性への影響を抑制しながら複製先プロセスから証拠を収集することを可能にする．

さらに，提案手法では，収集した証拠を対象のシステムから切り離された環境で扱うことを想定している．デジタルフォレンジックにおける証拠解析や報告書作成の工程を対象のシステムとは別の環境で実施することで，システム上で実施する工程を証拠の収集と保存に限定する．これにより，提案手法の適用による対象のシステムの可用性への影響を抑制する．

2.4 基本方式

提案手法の全体像を**図 1** に示す．提案手法では，(1) 複製依頼プロセスが複製を要求し，(2) プロセス複製処理部が対象プロセスを複製する．複製先プロセスを動作させないことで，対象プロセスのプロセス複製処理開始時と同一のメモリ内容を，複製先プロセスの仮想記憶空間に保持できる．次に，(3) 複製依頼プロセスは複製先プロセスから証拠を収集し，証拠収集後には必要に応じて複製先プロセスを削除する．最後に，(4) 複製依頼プロセスは収集した証拠を保存する．

プロセスの複製は，プロセス複製処理部を介して実行される．提案手法では，複製を要求するプロセスと対象となるプロセスは異なることから，プロセス複製処理の方式としては，以下の 2 つが考えられる．

(方式 1) 複製依頼プロセスのコンテキストで複製を実行する方式

(方式 2) 対象プロセスのコンテキストで複製を実行する方式

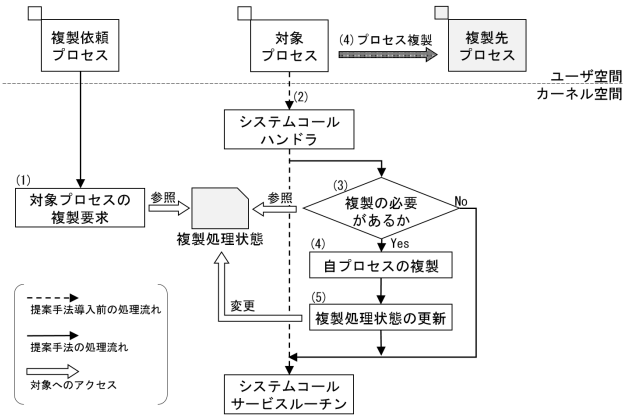

図 2 指定したプロセスの複製
Fig. 2 Process duplication of a target process.

このうち，(方式 1) は，マルチコア環境で動作させる場合には複製途中であっても対象プロセスが動作しうるため，一貫性のあるプロセスの状態を複製できないことがある．このため，提案手法では，(方式 2) によるプロセスの複製処理を実現する．

また，プロセス複製処理部では，複製要求の有無や複製状況（以降，複製処理状態と呼ぶ）を保持する領域を新たに定義し，この領域を介して，プロセス間で複製要求や複製処理状態の確認を行う．各複製処理状態を以下に示す．

初期状態：複製要求がない状態を指す．新たにプロセスを生成する場合，複製処理状態はこの状態に設定される．

複製要求状態：複製要求があり，まだ複製が開始されていない状態を指す．対象プロセスによってシステムコールが呼び出された際に，対象プロセスがこの状態であれば，プロセスの複製が必要だと判断し複製を開始する．

複製途中状態：プロセスの複製要求があり，複製を開始しているが，まだ複製が完了していない状態を指す．

複製完了状態：プロセスの複製要求があり，複製が完了している状態を指す．複製依頼プロセスは，対象プロセス複製が完了しているか判断するために利用する．

複製を要求する処理と複製処理状態を確認する処理は，複製依頼プロセスのコンテキストで実行する．また，複製対象プロセスのコンテキストでは，システムコールを発行した時に複製要求を確認する処理，および複製要求があった場合にプロセスの複製処理を実行する．複製要求の確認処理は，プロセスごとに用意した変数をチェックするのみであるため，オーバヘッドは小さい．また，プロセスの複製処理は，後述するように仮想記憶空間を複製し，コピーオンライト機構により物理メモリを共有し，プロセスのメモリイメージのメモリ間コピーを行わないことで処理オーバヘッドを削減する．これらの設計により，複製対象プロセスのコンテキストでの可用性への影響を抑制する．

プロセスの複製処理の処理流れについて，**図 2** を用いて以下で説明する．

(1) 複製依頼プロセスは，対象プロセスの複製を要求
　複製依頼プロセスは，複製処理状態を複製要求状態に変更することで，対象プロセスの複製を要求する．
(2) 対象プロセスは，何らかのシステムコールを発行
　この時，システムコール本来の処理の実行直前で処理をフックし，提案手法の処理に移行する．
(3) 対象プロセスの複製の要否を確認
　プロセス複製処理部は，複製処理状態を確認する．複製要求状態の場合，(4) に移行する．そうでない場合，システムコール本来の処理に復帰する．
(4) プロセス複製処理部が対象プロセスを複製
　プロセス複製処理部は複製途中状態に変更し，対象プロセスを複製する．
(5) 対象プロセスの複製処理状態を更新
　複製処理完了後，複製完了状態に変更する．その後，システムコール本来の処理に復帰する．なお，複製先プロセスの仮想記憶空間を保全するため，提案手法で複製したプロセスは ready 状態になることはない．

3. 実現方式

3.1 課題

提案手法を実現するうえでの課題を以下に示す．

（課題 1） 複製処理状態を保持する領域
　複製処理状態を保持する領域を，プロセスごとにどのように確保するか検討する必要がある．

（課題 2） プロセスからの複製処理状態の参照と変更処理の実現
　プロセス間で複製要求や複製処理状態の確認を行うために，指定プロセスの複製処理状態の変更および参照する操作を実現する必要がある．

（課題 3） 可用性への影響を抑制したプロセスの複製の実現
　処理負荷や処理時間を抑えたプロセスの複製処理を実現する必要がある．

（課題 4） プロセスからの証拠の収集
　提案手法は，プロセスに対してライブフォレンジック手法を適用する．ファイルシステムに痕跡を残さない攻撃手法に対処するためには，プロセスが保持する揮発性の証拠を収集する必要がある．

上記の課題のうち，（課題 1）から（課題 4）に対処し，Linux 3.13.0（64 bit）に提案手法を実現した．また，対処では，task_struct への変数の追加などカーネルの修正が必要であるため，Linux カーネルの再コンパイルが必要となる．なお，収集した証拠は，USB 経由で外付けハードディスクに保存する．

3.2 複製処理状態を保持する領域

提案手法では，異なるプロセス間で複製要求や複製処理状態を確認できるように，複製処理状態を保持する領域を新たに確保する．このために，プロセス情報を管理する task_struct 構造体に，2 つの変数（proc_copy_state 変数と proc_copy_pid 変数）を新たに追加することで対処した．**表 1** に task_struct 構造体に追加した変数を示す．また，**表 2** に複製処理状態と追加変数の関係を示す．proc_copy_state 変数には，各 task_struct に対応するプロセスの複製処理状態を格納する．また，proc_copy_pid 変数には，proc_copy_state 変数の値に応じて，複製要求元または複製先プロセスの PID を格納する．これらの変数の変更および参照により，プロセス間の複製要求の有無や複製処理状態を判断できる．

表 1 提案手法における task_struct 構造体の追加変数
Table 1 Additional valuables of task_struct structure in the proposed method.

形式	`struct task_struct {`
	`  volatile long state;`
	`  void *stack;`
	`  atomic_t usage;`
	`  unsigned int flags;`
	`  ⋮`
	`  int proc_copy_state; /* 追加 */`
	`  pid_t proc_copy_pid; /* 追加 */`
	`}`

追加変数	proc_copy_state：自プロセスの複製処理状態を保持
	proc_copy_pid：proc_copy_state 変数の値に応じ，複製要求元または複製先の PID を保持

表 2 複製処理状態と追加変数の関係
Table 2 Relationship between duplication processing state and additional valuables.

	初期状態	複製要求状態	複製途中状態	複製完了状態
proc_copy_state	0	1	2	3
proc_copy_pid	0	複製要求元の PID	複製要求元の PID	複製先の PID

3.3 複製処理状態の操作

複製処理状態の操作は，task_struct 構造体に追加された proc_copy_state 変数と proc_copy_pid 変数の値を参照および変更することで実現する．

まず，**表 3** に示す require_proc_copy() システムコールにより，指定したプロセスの複製を要求する．このシステムコールは，指定したプロセスの proc_copy_state 変数の値を初期状態に設定する．

また，**表 4** に示す get_proc_copy_state() システムコールにより，プロセスの複製処理状態を確認する．この

表 3 指定したプロセスの複製を要求するシステムコール
Table 3 System call requesting a duplication of a target process.

形式	`int require_proc_copy(pid_t pid)`
引数	`pid_t pid`：対象プロセスの PID
戻り値	成功：1 失敗：0
機能	プロセス `pid` の複製を要求する．具体的には，プロセス `pid` の `proc_copy_state` 変数の値を 1 に設定することで，当該プロセスを複製要求状態にする．

表 4 指定したプロセスの複製処理状態を確認するシステムコール
Table 4 System call checking the duplication processing state of a target process.

形式	`int get_proc_copy_state(pid_t pid, pid_t *c_pid)`
引数	`pid_t pid`：対象プロセスの PID `pid_t *c_pid`：複製先プロセスの PID
戻り値	成功：複製処理状態に対応する値（0–3） 失敗：−1
機能	プロセス `pid` の複製処理状態を確認する．具体的には，プロセス `pid` の `proc_copy_state` 変数の値を返す．また，プロセス `pid` が複製完了状態である場合には，`proc_copy_pid` 変数の値を `c_pid` 変数の指す領域に格納し，`proc_copy_state` 変数の値を 0 に設定することで，当該プロセスを初期状態に戻す．

システムコールは，指定したプロセスの `proc_copy_state` 変数の値を返す．また，第 1 引数で指定したプロセスの `proc_copy_state` 変数の値が複製完了状態である場合には，第 2 引数が指す領域に `proc_copy_pid` 変数の値を格納する．このとき，`proc_copy_state` 変数の値を初期状態に設定する．

3.4 プロセスの複製

提案手法では，システムコール呼び出しを契機にプロセスを複製する．このため，システムコールハンドラの処理におけるシステムコール本来の処理の直前に提案手法の処理を実行する．また，処理負荷や処理時間を抑えたプロセスの複製を実現するため，`fork()` システムコールの実装を参考にし，対象プロセスの仮想記憶空間を複製する．`fork()` システムコールはコピーオンライト機構を利用しており，プロセス間でメモリ上のデータのコピーを行わずに，ページテーブルのコピーのみでプロセスを複製できる．この処理以降，複製先プロセスが実行可能（ready）状態になることを不可にすることで，複製先プロセスの仮想記憶空間を保全する．

3.5 プロセスからの証拠の収集

提案手法では，ファイルシステムに攻撃の痕跡を残さない攻撃手法に対処するため，プロセスの仮想記憶空間から揮発性の証拠を収集する．提案手法では，`ptrace()` システムコールと `read()` システムコールを用いることにより，プロセスのメモリイメージのスナップショットを証拠として収集する．収集手順を以下に示す．

(1) `/proc/[pid]/maps` ファイルにアクセスし，プロセスが現在マッピングされている仮想メモリ領域のアドレスを確認
(2) `ptrace()` システムコールによるプロセスのトレースを開始
(3) `/proc/[pid]/mem` ファイルにアクセスし，`read()` システムコールを用いることでプロセスの仮想メモリ領域の内容を読み込み
(4) `ptrace()` システムコールによるプロセスのトレースを終了

複製依頼プロセスの仮想記憶空間に収集した証拠は，ファイルとして保存する．例として，`top` コマンドを実行しているプロセスに対して提案手法を適用した場合に得られるインデックスファイルとメモリイメージファイルを**図 3** と**図 4** に示す．

- 仮想メモリアドレスのインデックスファイル（図 3）
  `/proc/[pid]/maps` ファイルから得られる情報に対応しており，プロセスがマッピングされている仮想メモリ領域のアドレス，アクセスパーミッション，およびパス名を含むテキスト形式のファイルである．対象プロセスごとに，1 つのインデックスファイルを作成する．

```
tokimatsu@tokimatsu-exp: ~/tools/approach
# ProcMemInfo(generated by Approach)
# PID 2614 (/usr/bin/top)
# Date: (null)

[000] Address: 0x00400000 - 0x00417000 (94208 bytes)
     <src> /usr/bin/top
     <dst> mem-000.bin
     00400000-00417000 r-xp 00000000 08:07 9570226

[001] Address: 0x00616000 - 0x00617000 (4096 bytes)
     <src> /usr/bin/top
     <dst> mem-001.bin
     00616000-00617000 r--p 00016000 08:07 9570226

[002] Address: 0x00617000 - 0x00619000 (8192 bytes)
     <src> /usr/bin/top
     <dst> mem-002.bin
     00617000-00619000 rw-p 00017000 08:07 9570226

:
```

図 3 仮想メモリアドレスのインデックスファイル例（一部）
Fig. 3 Example of virtual memory address index file.

```
tokimatsu@tokimatsu-exp: ~/tools/approach
0000000 042577 043114 000402 000001 000000 000000 000000 000000
0000020 000002 000076 000001 000000 034244 000100 000000 000000
0000040 000100 000000 000000 000000 101250 000001 000000 000000
0000060 000000 000000 000100 000070 000011 000100 000034 000033
0000100 000006 000000 000005 000000 000100 000000 000000 000000
0000120 000100 000100 000000 000000 000100 000100 000000 000000
0000140 000770 000000 000000 000000 000770 000000 000000 000000
0000160 000010 000000 000000 000000 000003 000000 000004 000000
0000200 001070 000000 000000 000000 001070 000100 000000 000000
0000220 001070 000100 000000 000000 000034 000000 000000 000000
0000240 000034 000000 000000 000000 000001 000000 000000 000000
0000260 000001 000000 000005 000000 000000 000000 000000 000000
0000300 000000 000100 000000 000000 000000 000100 000000 000000
0000320 066054 000001 000000 000000 066054 000001 000000 000000
0000340 000000 000040 000000 000000 000001 000000 000006 000000
0000360 066740 000001 000000 000000 066740 000141 000000 000000
0000400 066740 000141 000000 000000 011620 000000 000000 000000
0000420 105550 000002 000000 000000 000000 000040 000000 000000
0000440 000002 000000 000006 000000 066770 000001 000000 000000
:
```

図 4 メモリイメージファイル例（一部）
Fig. 4 Example of memory image file.

- メモリイメージファイル（図 4）
  `/proc/[pid]/mem` ファイルから得られる情報に対応しており，プロセスがマッピングされている仮想メモリ領域の内容を含むバイナリ形式のファイルである．1 つのメモリイメージファイルは，対象プロセスの 1 つのセグメントに対応しており，仮想メモリアドレスのインデックスファイルに記載されているセグメントの数だけ，メモリイメージファイルを作成する．

利用例として，インデックスファイルから，疑わしい仮想メモリ領域を探索することや，その領域のメモリイメージファイルを参照することで，内容の調査や逆アセンブルなどでのコードを確認することがある．

3.6 他 OS への適用可能性

上記で述べた Linux 以外の OS への適用可能性について述べる．提案手法を適用するには，カーネル内に状態を管理する変数領域をプロセスごとに追加できること，カーネルに処理を依頼したとき（システムコール発行直後）に複製要否を判断する処理を追加できること，および多重仮想記憶とコピーオンライト機構をサポートしており，仮想記憶空間の複製により，プロセスのメモリスナップショットを取得できることが必要である．また，複製したプロセスのメモリ情報を取得するために，`ptrace()` システムコールのように，他プロセスの仮想記憶空間のデータを取得できる機能が必要である．これらの機能のサポートと OS へ機能の追加のための修正が可能であれば，提案手法を適用できる可能性がある．

4. 評価

4.1 評価項目と評価環境

提案手法で取得した内容の検証，および可用性への影響を評価するために，以下に示す評価を行った．また，評価環境を**表 5** に示す．

（評価 1） プロセスのメモリイメージのスナップショットの同一性検証

対象プロセスから得られるメモリイメージのスナップショットと複製先プロセスから得られるメモリイメージのスナップショットが同一であることを検証する．

（評価 2） プロセスの複製と削除の処理時間

プロセスの複製と削除に要する処理時間を評価する．

（評価 3） システムコールのオーバヘッド

提案手法は，システムコール呼び出し時の処理に提案手法の処理を挿入することで実現する．提案手法の導入によって発生するシステムコールのオーバヘッドを評価する．

（評価 4） 対象プロセスに発生する遅延時間

提案手法は，可用性を重視するシステム上で動作するプロセスへの適用を想定している．提案手法を適用した場合において，周期的に処理を実行するプロセスの遅延時間を評価する．

（評価 5） Memfetch との比較評価

類似ツールであるプロセスメモリダンプツール Memfetch [10] と提案手法を比較し，提案手法の有効性を考察する．

表 5 評価環境
Table 5 Evaluation environment.

CPU	Intel(R) Core(TM) i5-4460，3.20 GHz
メモリ	8 GB
OS	Ubuntu 14.04.3 LTS
カーネル	Linux 3.13.0-67，64 bit

4.2 プロセスのメモリイメージのスナップショットの同一性検証

提案手法は，複製先プロセスが複製時点における対象プロセスの仮想記憶空間の状態を保持している．そこで，対象プロセスと複製先プロセスからそれぞれメモリイメージのスナップショットを収集し，`diff` コマンドを用いてそれらを比較することで，同一のメモリイメージのスナップショットを収集できるか否かを検証した．

`/usr/bin/top` を実行し，実行中のプロセスを対象として評価した．評価により，対象プロセスと複製先プロセスから得られるメモリイメージのスナップショットに差分はなく，同一のメモリイメージのスナップショットを収集できることを確認した．

4.3 プロセスの複製と削除にかかる処理時間の評価

提案手法におけるプロセスの複製と削除にかかる処理時間を測定した．プロセスの複製の測定では，複製の要否を判定する処理から複製後に複製処理状態を更新する処理（図 2 の (3)–(5)）を測定区間とし，削除の測定では，`kill()` システムコールの呼び出し前後を測定区間とした．

本評価では，`getpid()` システムコールを繰り返し実行するプログラムを作成し，対象プロセスとして使用した．また，対象プロセスの仮想メモリ領域のサイズは，4,188 KB だった．

評価結果を**表 6** に示す．プロセスの複製の処理時間は 17.84 μs と小さい．しかし，プロセスの複製処理は，対象プロセスのシステムコールハンドラの実行前に行われるため，複写処理が行われた場合は，対象プロセスのシステムコール処理の終了が，この分だけ延びると推察できる．ま

表 6 プロセスの複製と削除にかかる処理時間（単位：μs）
Table 6 Processing time of process duplication and deletion (unit: μs).

	複製	削除
処理時間	17.84	49.47

表 7 getpid() システムコールのオーバヘッド（単位：μs）
Table 7 Overhead of getpid() system call (unit: μs).

	導入前	導入後	オーバヘッド
処理時間	0.0868	0.0955	0.0087

た，プロセスの削除の処理時間は 49.47 μs と小さいため，複製対象プロセスへの影響は小さいと推察できる．

4.4 システムコールのオーバヘッド

プロセスの複製要求がない場合における getpid() システムコールのオーバヘッドを測定した．本評価では，提案手法の導入の前後で getpid() システムコールを 100 回実行するのにかかる処理時間を測定し，getpid() システムコール 1 回あたりの処理時間とオーバヘッドを算出した．

評価結果を**表 7** に示す．getpid() システムコールのオーバヘッドは 0.0087 μs であり，小さい．これは，プロセスの複製要求がない場合，システムコールに追加される処理は，システムコールのフックとプロセスの複製要求の有無の判定のみであるためである．

4.5 対象プロセスに発生する遅延時間

提案手法は，システムを簡単に止められない可用性を重視するシステム上で動作するプロセスへの適用も想定している．そこで，複製対象プロセスに発生する遅延を測定することで，提案手法の適用による可用性への影響を評価した．

重要インフラ制御システム上で動作するタスクを想定した場合，対象プロセスは一定の時間間隔で周期的に割り込み処理を実行することが考えられる．そこで，本評価では，周期的にリアルタイム処理を行うタスクを想定し，対象プロセスとしてオープンソースのベンチマークツールの cyclictest を使用した．cyclictest は，指定した時間で周期的な割り込み要求を実行し，その割り込み要求によって実行されるスリーププログラムの遅延時間を測定できる．

また，本評価では，提案手法との比較として，Memfetch の適用による遅延時間もあわせて測定した．Memfetch は，オープンソースのプロセスメモリダンプツールであり，提案手法と同様，プロセスが現在マッピングされている仮想メモリ領域のアドレスとそのアドレスに対応するメモリ領域の内容を収集できる．

本評価では，以下の場合における遅延時間をそれぞれ測定した．

（場合 1） cyclictest に何も適用しない場合の遅延時間

（場合 2） cyclictest に Memfetch を繰り返し適用する場合の遅延時間

cyclictest を対象プロセスとして Memfetch による処理を繰り返し実行する．Memfetch による処理は，対象プロセスからの証拠収集（外付けディスクへの書き出しを含む）である．

（場合 3） cyclictest に提案手法を繰り返し適用する場合の遅延時間

cyclictest を対象プロセスとして提案手法による処理を繰り返し実行する．提案手法による処理は，対象プロセスの複製，複製先プロセスからの証拠収集（外付けディスクへの書き出しを含む），および複製先プロセスの削除である．

表 8 対象プロセスに発生する遅延時間（SCHED_FIFO の場合）
Table 8 Delay time occurring in a target process (SCHED_FIFO).

	遅延時間（単位：μs）		
	最小	平均	最大
非適用	1.047	1.557	51.984
Memfetch 適用	0.992	1.724	253.212
（増加時間）	(−0.055)	(+0.167)	(+201.228)
提案手法適用	0.997	1.685	189.015
（増加時間）	(−0.050)	(+0.128)	(+137.031)

表 9 対象プロセスに発生する遅延時間（SCHED_OTHER の場合）
Table 9 Delay time occurring in a target process (SCHED_OTHER).

	遅延時間（単位：μs）		
	最小	平均	最大
非適用	3.012	51.714	3636.908
Memfetch 適用	2.491	55.251	5365.859
（増加時間）	(−0.521)	(+3.537)	(+1728.951)
提案手法適用	3.308	54.510	3840.909
（増加時間）	(+0.296)	(+2.796)	(+204.001)

cyclictest は，周期的な割り込み要求の時間間隔を 1 ms，スケジューリングポリシを SCHED_FIFO（優先度：99）あるいは SCHED_OTHER（優先度：0）に指定し，10 分間連続して動作させる．また，本評価では，cyclictest を一部改変し，bss 部のサイズを 100 MB 増加させている．なお，改変後の cyclictest の仮想メモリ領域の合計サイズは，約 117 MB だった．

スケジューリングポリシを SCHED_FIFO と SCHED_OTHER に設定した場合の各評価結果を，**表 8** と**表 9** に示す．文献 [11] によると，制御システムの通信における問合せから応答までの時間は，250 μs から 10 ms 程度であることから，この範囲内の遅延は許容できると推察できる．

まず，表 8 の SCHED_FIFO 場合，提案手法を適用した場合の平均遅延時間は，1.685 μs であり，提案手法を適用したことによる増加時間は 0.128 μs と小さい．また，提案手法を適用した場合の遅延時間は，最小値，平均値，および最大値のいずれにおいても 250 μs 以内であることから，許容範囲内であると推察できる．

次に，表 9 の SCHED_OTHER の場合，提案手法を適用した場合の平均遅延時間は，54.510 μs であり，提案手法の適用による増加時間は 2.796 μs と小さい．このことから，

SCHED_FIFO の場合の評価結果と同様に，影響は小さいと推察できる．また，提案手法を適用した場合の遅延時間は，最小値と平均値は 250 μs 以下に収まっていることから，許容範囲内である．最大遅延時間は，非適用時に 250 μs を超えているものの，適用時の処理時間の増加はできるだけ小さいことが望ましい．提案手法の適用時の最大遅延時間は約 3.8 ms であり，250 μs を大きく上回っているものの，非適用時からの増加は約 200 μs 程度（6%未満）である．ただし，非リアルタイムスケジューリングポリシで運用されるような制御システムの場合では，制御システムの通信における問合せから応答までの時間のデッドラインを極端に短い時間に設定する必要性は低いと考えられるため，数 ms のオーバヘッドは許容できると推察できる．

4.6 Memfetch との比較評価

Memfetch と提案手法を比較すると，どちらの場合も表 9 の最大遅延時間が 250 μs を大きく上回っているものの，提案手法の方が約 1.5 ms 小さいことが分かる．この要因として，対象プロセスの停止時間の差が考えられる．Memfetch は，対象プロセスから直接証拠を収集するため，証拠収集時には対象プロセスを停止させておく必要がある．これに対し，提案手法は，対象プロセスを複製して複製先プロセスから証拠を収集するため，証拠収集時に対象プロセスを停止させる必要がない．提案手法では，対象プロセス cyclictest の複製による遅延が発生するものの，複製にかかる処理時間は約 31 μs と短い時間であったことから，発生する遅延は小さいと推察できる．この要因により，Memfetch を適用した場合の遅延時間が増加し，提案手法を適用した場合の最大遅延時間が抑制できていると推察している．

また，Memfetch は，対象プロセスから直接証拠を収集することから，証拠収集時に対象プロセスの停止をともなう．これは，3.1 節で述べた（要件 2）を満たせなくなる可能性がある．これに対して，提案手法は，（要件 2）を満たすため，対象プロセスを複製して複製先プロセスから証拠を収集するため，証拠収集時には対象プロセスを停止させる必要がない．

さらに，提案手法は，複製対象プロセスのメモリイメージをスナップショットとして，複数保持できる利点がある．すでにスナップショットとして仮想記憶空間を複製している場合，複製先プロセスから情報を一度に取得せずに，間隔を空けながら部分的に取得し保存することを繰り返すことで，対象プロセスの可用性への影響をさらに抑制できる．

一方で，提案手法は `task_struct` への変数追加などのカーネルの修正が必要なため，Memfetch と異なり，導入のためにカーネルの再コンパイルが必要となる．

5. 関連研究

可用性を重視するシステムの 1 つである制御システムでは，オープン化によるセキュリティ脅威の表面化にともない，セキュリティ対策の重要性が高まっている．制御システムにおけるデジタルフォレンジック手法の研究として，文献 [5], [6], [7], [8] がある．文献 [5] では，`dd` コマンドによるディスクイメージ収集手法を実行した場合におけるシステムへの負荷を測定し，制御システムでの有効性を評価している．また，文献 [6] では，文献 [5] の評価をふまえ，`dd` コマンドによるディスクイメージ収集手法を実行した場合における制御システムで動作するタスクに対する遅延時間を測定し，制御システムへの適用可能性を評価している．

文献 [7] では，インシデントの検知漏れやシステムの性能低下の対処としてホットデジタルフォレンジックという手法を提案している．この手法では，ホットバックアップとイメージバックアップを組み合わせたバックアップにより，証拠の収集時間を短縮する．また，インシデントの検知や証拠の解析は，対象のシステムから切り離された環境で実施することで，システムへの影響を抑える．文献 [8] では，状況に応じてシステムをモニタモードとフォレンジックモードに切り替える方式を提案している．平常時にはモニターモードを動作させ，制御システムの状態を監視し，異常を検出した場合にはフォレンジックモードに切り替えてインシデント対応を行う．また，低負荷な証拠の収集の実現を課題としており，この対処としてプロセスやメモリから得られる証拠の収集をあげている．

クラウド環境を対象としたライブフォレンジックを実現するハイパーバイザとして，ForenVisor が提案されている [12]．ForenVisor には，インシデント発生後に動的にロードできる特徴がある．また，文献 [13] では．メモリフォレンジックにおいて，メモリ内データ構造を自動的に解釈して，レンダリングするシステムである DSCRETE を提案している．Volatility [14] などのメモリフォレンジックツールで解析するメモリダンプには，カーネルメモリのみや物理メモリ全体をダンプしたものなどがある．一方，提案手法は，プロセスの仮想記憶空間を対象とし，負荷が小さく，プロセスのメモリのスナップショットを複数保持できる特徴がある．物理メモリベースのダンプ手法では，特定のプロセスのメモリだけをダンプしようとすると，ページテーブルを解析して該当する個々の物理ページをダンプしないといけないため，物理メモリベースで個々のプロセスをダンプするのは処理負荷などの観点から現実的ではない．

ディスクの大容量化や対象となるシステムの多様化にともない，デジタルフォレンジックにおける証拠収集にかかる時間が長大化している．これに対し，迅速な証拠収集を目的としたデジタルフォレンジック手法の研究として，文献 [15], [16], [17] がある．文献 [15] では，広範囲の Linux カーネルに対応するため，システム上にすでに存在するカーネルモジュールにメモリ収集処理を組み込む手法を提

案している．これにより，Linux カーネルごとにメモリ収集用のカーネルモジュールを用意する必要がなくなり，迅速なデジタルフォレンジック手法の適用を可能にする．文献 [16] では，仮想化環境におけるサイバー攻撃に対するデジタルフォレンジック手法として，VMMF という手法を提案している．また，従来のデジタルフォレンジック手法の課題として，収集する証拠が膨大となり，解析が困難になる点をあげている．この対処として，VMMF では，システム上で動作する不審なプロセスから証拠を収集することで，収集する証拠を削減している．文献 [17] では，モバイル端末向けのデジタルフォレンジック手法として，デバイスストレージの自動差分フォレンジック収集手法を提案している．差分解析を用いて差分のある情報に絞って収集することで，証拠の収集にかかる時間の短縮を実現する．

Linux において，プロセス単位でメモリダンプするツールとして，ProcDump-for-Linux [18] がある．このツールは，CPU 使用率が指定した条件を満たしたときなどに，プロセスのダンプを出力できる特長がある．一方で，提案手法と異なり，ダンプは保存時にファイルを作成して行われる．

可用性を重視するシステムにおけるデジタルフォレンジック手法の研究は，主にハードディスクを調査対象とする手法に焦点をあてており，ファイルシステムに攻撃の痕跡を残さない攻撃手法への対処の検討は不十分である．提案手法は，プロセスを調査対象とし，プロセスを複製して複製先プロセスから揮発性の証拠を収集することで，システムの可用性の低下を抑えつつファイルシステムに攻撃の痕跡を残さない攻撃手法に対処できる．

6. おわりに

システムの可用性の低下を抑えつつ，ファイルシステムに痕跡を残さない攻撃手法に対処可能なライブフォレンジック手法として，可用性を考慮したプロセスの複製によるライブフォレンジック手法を提案した．提案手法は，プロセスを調査対象とし，複製した仮想記憶空間から揮発性の証拠を収集することでファイルシステムに痕跡を残さない攻撃に対処する．また，複製先プロセスから証拠を収集することで，対象プロセスを停止することなく証拠を収集でき，システムの可用性への影響を抑制できる．さらに，複製対象プロセスのメモリスナップショットを複数保持できる利点がある．

また，評価として，提案手法の動作確認と基本性能を測定した．さらに，周期的に処理を実行するプロセスを使用し，提案手法の適用による遅延時間を測定することで，可用性を重視するシステムに対する提案手法の有効性を示した．

謝辞 本研究の一部は，科学研究費補助金基盤研究（B）（課題番号：16H02829）による．

参考文献

[1] デジタル・フォレンジック研究会：証拠保全ガイドライン第 6 版，入手先 ⟨https://digitalforensic.jp/wp-content/uploads/2017/05/idf-guideline-6-20170509.pdf⟩（参照 2017-11-06）．
[2] 武部達明：制御システムセキュリティにおける業界標準・国際標準の動向（制御システムのセキュリティ特集），横河技報，Vol.57, No.2, pp.31–34 (2014).
[3] 上原哲太郎：ディジタルフォレンジック—電磁的証拠の収集と分析の技術，情報処理，Vol.48, No.8, pp.889–898 (2007).
[4] McAfee Labs：McAfee 脅威レポート：2015 年第 3 四半期，入手先 ⟨http://www.mcafee.com/jp/resources/reports/rp-quarterly-threats-nov-2015.pdf⟩（参照 2017-11-06）．
[5] Ahmed, I., Obermeier, S., Naedele, M. and Richard III, G.G.: SCADA Systems: Challenges for Forensic Investigators, *Computer*, Vol.45, No.12, pp.44–51 (2012).
[6] 田村研輔，松浦幹太：制御システムにおけるライブフォレンジックの適用可能性に関する実験的評価，2015 年暗号と情報セキュリティシンポジウム（SCIS2015）予稿集，電子媒体 (2015).
[7] 越智貴夫，小島孝夫，外川政夫，板倉征男：ホットデジタルフォレンジックによるインシデント検知方法の提案，情報処理学会研究報告，Vol.2008-CSEC-040, No.21, pp.267–272 (2008).
[8] Taveras, P.: SCADA live forensics: Real time data acquisition process to detect, prevent or evaluate critical situations, *European Scientific Journal*, Vol.9, No.21 (2013).
[9] 今野直樹，田中英彦：ライブフォレンジックにおける有効性の検討及び具体的実施手法の提案，情報科学技術フォーラム講演論文集，Vol.14, No.4, pp.205–212 (2015).
[10] [lcamtuf.coredump.cx]:Memfetch, available from ⟨http://lcamtuf.coredump.cx/soft/memfetch.tgz⟩ (accessed 2017-12-05).
[11] Galloway, B. and Hancke, G.P.: Introduction to Industrial Control Networks, *IEEE Communications Surveys & Tutorials*, Vol.15, No.2, pp.860–880 (2013).
[12] Qi, Z., Xiang, C., Ma, R., et al.: ForenVisor: A Tool for Acquiring and Preserving Reliable Data in Cloud Live Forensics, *IEEE Trans. Cloud Computing*, Vol.5, No.3, pp.443–456 (2017).
[13] Saltaformaggio, B., Gu, Z., Zhang, X. and Xu, D.: DSCRETE: Automatic rendering of forensic information from memory images via application logic reuse, *23rd USENIX Security Symposium* (*USENIX Security '14*), pp.255–269 (2014).
[14] The Volatility Foundation - Open Source Memory Forensics, available from ⟨https://www.volatilityfoundation.org/⟩ (accessed 2018-06-25).
[15] Stüttgen, J. and Cohen, M.: Robust Linux memory acquisition with minimal target impact, *Digital Investigation*, Vol.11, pp.S112–S119 (2014).
[16] Li, Y.G., Cui C.Y., Wu, Y. and Sun B.Y.: VMMF: Virtual Machine Memory Forensics Based on Event Trigger Mechanism, *DEStech Trans. Computer Science and Engineering ICEITI* (2016).
[17] Guido, M., Buttner, J. and Grover, J.: Rapid differential forensic imaging of mobile devices, *Digital Investigation*, Vol.18, pp.S46–S54 (2016).
[18] ProcDump-for-Linux: available from ⟨https://github.com/Microsoft/ProcDump-for-Linux⟩ (accessed 2018-10-01).

推薦文

本論文はファイルレスマルウェアの流行などを背景に，ディスクではなくメモリデータを対象としたライブフォレンジック手法を提案している．メモリ上のデータの保全作業中に未保全のデータが更新されないことを考慮して設計するなど実用的な提案であり大きく評価できる．よって推薦論文として推薦する．

（コンピュータセキュリティシンポジウム 2017（CSS2017）
プログラム委員長　須賀祐治）

山内 利宏　（正会員）

1998 年九州大学工学部情報工学科卒業．2000 年同大学大学院システム情報科学研究科修士課程修了．2002 年同大学院システム情報科学府博士後期課程修了．2001 年日本学術振興会特別研究員（DC2）．2002 年九州大学大学院システム情報科学研究院助手．2005 年岡山大学大学院自然科学研究科助教授．現在，同准教授．博士（工学）．オペレーティングシステム，コンピュータセキュリティに興味を持つ．2010 年度 JIP Outstanding Paper Award，2012 年度情報処理学会論文賞等受賞．電子情報通信学会，ACM，USENIX，IEEE 各会員．本会シニア会員．

時松 勇介

2016 年岡山大学工学部情報系学科卒業．2018 年同大学大学院自然科学研究科博士前期課程修了．コンピュータセキュリティに興味を持つ．

谷口 秀夫　（正会員）

1978 年九州大学工学部電子工学科卒業．1980 年同大学大学院修士課程修了．同年日本電信電話公社電気通信研究所入所．1987 年同所主任研究員．1988 年 NTT データ通信株式会社開発本部移籍．1992 年同本部主幹技師．1993 年九州大学工学部助教授．2003 年岡山大学工学部教授．2010 年岡山大学工学部長．2014 年岡山大学理事・副学長．博士（工学）．オペレーティングシステム，実時間処理，分散処理に興味を持つ．著書『並列分散処理』（コロナ社）等．電子情報通信学会，ACM 各会員．本会フェロー．