仮想マシンモニタによるパスワードクラッキング検出

三浦俊朗[†] 本田惇[†]
高橋一志[†] 大山恵弘[†]

クラウドコンピューティング基盤は今や人々の生活の土台となっている．しかし残念ながら，それは悪い目的にも利用されうる．例えば，Amazon EC2 がパスワードクラッキングにとって魅力的なプラットフォームであることが複数の技術者によって指摘されている．大半のクラウドコンピューティング基盤においては，仮想マシンモニタが提供する仮想マシン上で OS やアプリケーションが動作している．仮想マシンモニタは仮想マシン内で動くプログラムの実行を監視することが可能である．本研究では，仮想マシンモニタの層からパスワードクラッキングの挙動を検出するための方法を提案する．その方法はゲスト OS 内で動いているプロセスのメモリデータを定期的に取得し，パスワード候補として用いられている可能性がある文字列を収集する．その方法は，収集した文字列集合を元に，パスワードクラッキングを行っているプロセスを発見する．

Detection of Password Cracking by Virtual Machine Monitors

Toshiro Miura ,[†] Jun Honda ,[†] Kazushi Takahashi [†]
and Yoshihiro Oyama[†]

Cloud computing infrastructure is now a basis of our lives. Unfortunately, it can also be used for malicious purposes. For example, several engineers pointed out that Amazon EC2 was an attractive platform for password cracking. In most cloud computing infrastructure, operating systems and applications run on virtual machines provided by virtual machine monitors, which can monitor the execution of programs running in virtual machines. In this work, we propose a method for detecting the behavior of password cracking from the layer of a virtual machine monitor. The method periodically obtains memory data of processes running in a guest operating system and collects character strings that may be used as password candidates. Based on a set of collected strings, it finds a process that is performing password cracking.

1. はじめに

クラウドコンピューティング基盤は今や人々の生活の土台となっているが，残念ながら，悪用される可能性もある．例えば，著名なクラウドコンピューティング基盤である Amazon EC2 をパスワードの総当たり攻撃に有効に利用できたことが報告されている[3),4),11)]．また，2011 年 4 月に起きたソニーの個人情報流出事件では，攻撃側が Amazon EC2 のサービスを利用したという報道もあった[1)]．

大半のクラウドコンピューティング基盤においては，仮想マシンモニタ（VMM）が提供する仮想マシン上で OS やアプリケーションが動作している．VMM により，1 台の物理マシンを複数台の仮想マシンとして提供することができる．特に IaaS (Infrastructure as a Service) と呼ばれる形態のサービスでは，仮想化された CPU やストレージなどのハードウェアがユーザに提供され，その上でユーザが自分の好みの OS を実行できる．例えば著名な IaaS の Amazon EC2 では，Xen VMM によって提供される仮想マシン上で，ユーザは自分でダウンロードまたは作成した Linux や Windows などの OS イメージを実行することができる．仮想マシン上で実行される OS はゲスト OS と呼ばれる．

IaaS においては，クラウド事業者側による悪用への対処が単純ではない．IaaS ではユーザは OS を含む実行環境全体を自分で構築できるため，ゲスト OS 内でセキュリティソフトウェアを動かすことを事業者側で技術的に強制することが難しい．例えば，パスワードクラッカーを検出するプログラムを，ユーザのゲスト OS 内で動かすには，そのユーザの協力，同意，手間が必要になる．ただし，VMM は事業者が管理しているため，事業者はセキュリティ機構を VMM 層に入れることはできる．そして，VMM は仮想マシン内で動くプログラムの実行を監視することが可能である．

† 電気通信大学
The University of Electro-Communications

本研究では，ゲスト OS 内でパスワードクラッキングを行っているプロセスを仮想マシンモニタ層（VMM 層）から検出する方法を提案する．その方法では，VMM がゲスト OS のレジスタやメモリの内容を定期的に取得し，文字列であると推測されるデータを収集する．そして，収集したデータの集合を元に，パスワードクラッキングを行っている可能性があるプロセスを検出する．我々は，提案方法に基づくシステム Pacrader (PAssword CRAcking DEtectoR) を現在実装中である．Pacrader は KVM を改造して実装されるが，本方法は KVM に特化した機構を用いてはいないため，様々な VMM に適用できると考えている．

本研究とは異なる方法として，悪意によるツールのファイルのシグネチャやメモリデータのシグネチャを準備し，それらと照合することにより，悪意によるツールをゲスト OS から検出することも考えられる[5),10),13)]．しかし，今回はその方法は採用しない．それは，特定のシグネチャだけを検出するようにすると，特定のツールだけしか検出できなくなり，ツールの亜種，未知のツール，パッカーでパックされたツールなどが見逃されるからである．本研究の方法は，動的な挙動の情報を用いるので，それらすべてのツールによるパスワードクラッキングを検出できる可能性がある．

2. パスワードクラッキング

パスワードクラッキングはローカルクラッキングとリモートクラッキングの 2 種類に分類される．ローカルクラッキングは，攻撃者がパスワードファイルを入手し，攻撃者のコンピュータ上や，攻撃者が利用できるサーバなどの上でクラッキングを行う方法である．パスワードファイルには通常，パスワードのハッシュ値が保存されており，そのハッシュ値を与える文字列をみつけることがクラッキングの目的となる．一方，リモートクラッキングは，外部のサーバやサービスに対して，ネットワーク経由で ID やパスワードを繰り返し送信して認証に成功するかを試す方法である．本研究ではローカルクラッキングを扱う．

パスワードクラッキングの方法としては，総当たり攻撃と辞書攻撃の 2 つが良く用いられる．総当たり攻撃は，候補となる全ての文字列をパスワード候補と考え，片端からパスワードを試す手法である．例えば，aaa, aab, aac, ..., aaz, aba, ... というように 1 文字ずつ順番に変化させ，かつ，パスワード候補の文字列を 1 文字ずつ順番に長くしていく．この方法には，いつかは必ずパスワードをみつけることができるという利点がある．その反面，この方法には，長いパスワードに対しては非常に長い時間がかかるという欠点や，英単語をそのまま利用したような脆弱なパスワードに対しても長い時間がかかるという欠点がある．

辞書攻撃は，パスワードに利用されやすい文字列の載っている辞書ファイルを予め用意し，その中の単語をパスワードとして試す手法である．辞書ファイルとしては，著名人の名前や都市の名前が載っているものなどの様々な種類のものがインターネット上に存在する．著名なパスワードクラッキングソフトウェアである John the Ripper[6)] のバージョン 1.7.9 には，3546 文字列からなる辞書ファイルが含まれている．John the Ripper を配布している Openwall 社は，4000 万以上の文字列を含む辞書ファイルを販売している．

3. 提案方法

提案方法は，総当たり攻撃のための文字列や辞書攻撃のための文字列が自身のアドレス空間内に頻繁に検出されるプロセスを，パスワードクラッキングを行っている可能性があるプロセスとして検出する．提案方法を実現する VMM である Pacrader は，ゲスト OS にタイマ割り込みを一定回数注入するごとに，ゲスト OS 内にある以下の 2 つのデータを取得する．

1 つ目のデータは，現在スケジュールされているプロセスのプロセス ID である．この取得処理を可能にするために，ゲスト OS のタスク構造体のレイアウト情報をあらかじめ VMM に与えておく．ゲスト OS が Linux の場合には，まずスタックポインタレジスタの値を得る．その値の下位ビットをクリアすることにより，現在のスレッドの情報を保持する `thread_info` 構造体のアドレスを得る．`thread_info` 構造体は，現在のプロセスの情報を保持する `task_struct` 構造体のアドレスをメンバとして持つ．`task_struct` 構造体はプロセス ID をメンバとして持つ．VMM はこの流れでポインタを辿ることによりプロセス ID を取得する．

2 つ目のデータは，ユーザレベルスタックに積まれた値である．VMM は，現在スケジュールされているプロセスのユーザレベルスタックに積まれた値を，スタックトップから順に，予め決められたワード数（デフォルトでは 20 ワード）だけ取得する．それらの値のうち，文字列を指すポインタである可能性があるものを認識し，かつ，文字列と推測されるデータを取得する．そして，そのデータが，そのプロセス ID のプロセスから検出されたことを VMM 内部のデータ構造に記録しておく．記録する際には，過去に同じ文字列がそのプロセスから検出されたかどうかを確認し，検出されていたら記録しないようにする．

検出された文字列の数が一定の閾値を越えたら，そのプロセスがクラッキングを行っていると判定する．総当たり攻撃では，パスワード候補の文字列の接頭語に共通部分が発生しやすい．そこで，共通部分を持つ文字列が一定数以上検出される事象が一定回数以上発生したら，クラッキングを行っていると判断する．デフォルトでは，先頭 4 文字が共通である文字列が 10 個以上検出される事象が 5 回発生したらクラッキング

とみなす．辞書攻撃については，予め VMM に辞書ファイルを読ませておく．ゲスト OS から検出された文字列と辞書ファイルの文字列を照合して，一致する文字列の数が閾値（デフォルトでは 1000）を越えたらクラッキングとみなす．

ゲスト OS からレジスタの値やメモリデータを取得する処理は，改造前の VMM が元々提供している関数を呼び出すことによって実装する．例えば KVM では，`kvm_vcpu_ioctl()` 関数を呼び出すことにより，レジスタの値が入っている構造体のデータを取得できる．また，`kvm_read_guest()` 関数では，引数にメモリアドレスと読み込むバイト数を指定することで，ゲスト OS のメモリの内容を取得できる．

上述したユーザレベルスタックに積まれた値および文字列の取得について，以下で詳しく述べる．VMM はまず，ゲスト OS のページテーブルを参照して，現在スケジュールされているプロセスが使用している仮想メモリのアドレス範囲（有効な仮想アドレス範囲）を調べる．ページテーブルの先頭アドレスは CR3 レジスタから取得する．次に，スタックトップから順に 1 ワードずつ値を取得し，その値をポインタとして解釈したときに，それが有効な仮想アドレス範囲内にあるかどうかを調べる．範囲内にある場合には，それをポインタとみなし，指す先のメモリの内容を取得する．そして，取得した内容の先頭バイトから順に，各バイトに対して以下の処理を行う．

(1) パスワードに用いられうる文字であった場合，引き続き，次のバイトを調べる．
(2) NUL 文字であった場合，それまでに読んだバイト群は文字列であると判定し，スタック内の 1 つ下のワードの検査に移る．
(3) それ以外のバイトデータ（制御文字など）であった場合，それまでに読んだバイト群は文字列ではないと判定し，スタック内の 1 つ下のワードの検査に移る．

取得する文字列の長さには上限を設ける，32 バイト目まで調べてもまだ文字列が続く場合，それはパスワード候補の文字列ではないと判定し，捨てられる．また逆に，取得された文字列の長さが閾値（デフォルトでは 4 文字）以下であったら，それもパスワード候補の文字列ではないと判定され，捨てられる．

この方法では，スタックに積まれた値が文字列を指すポインタでなかった場合でも，その値が偶然有効な仮想アドレス範囲内に入っていて，それが指すメモリ領域に文字列と判断できるデータが発見されれば，文字列として扱われることになる．それによって検出精度は下がる可能性はあるが，VMM がクラッシュするなどの障害は発生しない．

4. 議　　論

現在はタイマ割り込みが一定回数注入されたタイミングでゲスト OS のメモリを検査することにしている．しかし，どのタイミングで検査すべきかは，単純な問題ではない．ただ，パスワードクラッキングを行うプロセスは，通常，CPU を常に消費しようとするため，タイマ割り込み時に CPU がそのプロセスに割り当てられている可能性は高いと推測され，その意味では悪くはないタイミングだと考えている．

提案方法において，プロセスのスタック内のみから文字列を探している理由は，実装を単純にし，オーバヘッドを小さくするためである．もし，大域変数やヒープの領域の場所を効率的に知ることができるならば，それらの領域からも文字列を探すことは妥当な拡張となる．一般には，多くの領域を検査すれば，より多くのパスワードクラッキングの挙動を検出しやすくなるが，より多くの時間やメモリを消費する．よって，トレードオフを考える必要がある．

5. パスワードクラッキングソフトウェアの調査

パスワードクラッキングソフトウェアである John the Ripper 1.7.9 を入手して内部構造を理解し，Pacrader がパスワードクラッキングの挙動を検出できるかどうかを検討した．

John the Ripper の総当たり攻撃におけるパスワード候補の生成処理は，`inc.c` ファイルの `inc_key_loop()` 関数内で行われる．対応するソースコードの抜粋を**図 1** に示す．この部分では，変数 `key` が指すメモリ領域にパスワード候補の文字列を書き込む．そして，ハッシュ値の計算などを行う `crk_process_key()` 関数に，変数 `key` の値を渡している．提案方法は，変数 `key` が指すメモリ領域の内容を，パスワード候補の文字列として取得する可能性が高い．

John the Ripper の辞書攻撃におけるパスワード候補の生成処理は，`wordlist.c` ファイルの `do_wordlist_crack()` 関数内で行われる．対応するソースコードの抜粋を**図 2** に示す．辞書ファイルを `fgetl()` 関数で 1 行ずつ読み取り，読んだ文字列を変数 `word` に指させている．そして，総当たり攻撃と同様に，`crk_process_key()` 関数に変数 `word` の値を渡している．提案方法は，変数 `word` が指すメモリ領域の内容を，パスワード候補の文字列として取得する可能性が高い．

6. 予 備 実 験

John the Ripper 1.7.9 からどのような文字列が取得できるかを知るための予備実験を行った．環境は

```
static int inc_key_loop(int length,
  int fixed, int count, char *char1,
  char2_table char2, chars_table *chars)
{
    char key_i[PLAINTEXT_BUFFER_SIZE];
    char key_e[PLAINTEXT_BUFFER_SIZE];
    char *key;
    ... /* 鍵の生成 */
    key = key_i;
    if (!f_filter
        || ext_filter_body(key_i,
                           key = key_e))
        if (crk_process_key(key))
            return 1;
    ...
}
```

図 1 John the Ripper の総当たり攻撃のコード

```
void do_wordlist_crack(struct db_main *db,
                    char *name, int rules)
{
  ...
  while (fgetl(line, LINE_BUFFER_SIZE,
               word_file)) {
    line_number++;
      if (line[0] != '#') {
not_comment:
        if ((word = apply(line, rule, -1,
                          last))) {
          last = word;
          if (ext_filter(word))
            if (crk_process_key(word)) {
              rules = 0;
              break;
            }
        }
        continue;
      }
      if (strncmp(line, "#!comment", 9))
          goto not_comment;
  }
  ...
}
```

図 2 John the Ripper の辞書攻撃のコード

Intel Core 2 Duo 上で動作する Linux 2.6.35-20 である．John the Ripper のビルドに用いた gcc のバージョンは 4.4.5 であり，32 bit で SSE2 ありというオプションを make に与えた．

この予備実験では，VMM は用いず，アプリケーションプログラムを ptrace システムコールで追跡しながら実行するプログラム（以下，監視プログラム）を用いた．監視プログラムは，アプリケーションプログラムの全 call 命令にブレークポイントを入れ，その場

```
（総当たり攻撃を実行させた場合）
...,
"- Switching to length %d",
 "%u:%02u:%02u:%02u ", "%u:%02u:%02u:%02u ",
"maris", "maris", "marie", "marie", "marty",
"marty", "marth", "marth", "macky", "macky",
"macks", "macks", "macha", "macha", "mache",
"mache", "misha", "misha", "mish1", "mish1",
"missy", "missy", "misse", "misse", "miker",
"miker", "mikey", "mikey", "mikki", "mikki",
"mikko", "mikko", "mikko",
"%u:%02u:%02u:%02u ", "%u:%02u:%02u:%02u ",
"sandy", "sandy", "sanda", "sanda", "sanny",
"sanny", "sanna", "sanna", "sanna",
...

（辞書攻撃を実行させた場合）
...
"library", "lincoln", ..., "lincoln",
"lionking", "lincoln", "lionking", "lincoln",
"lionking", ..., "lionking", "london",
"lionking", "london", "lionking", "london",
...
```

図 3 John the Ripper を追跡する監視プログラムによって取得された文字列

所でアプリケーションを停止させる．停止したら，スタックの上部から 6 ワードを読み込み，3 節で述べた方法で，それらをポインタとみなして文字列を取得する．これはすなわち，関数呼び出しの第 1 引数から第 6 引数までに文字列へのポインタが与えられた場合に，その文字列の取得を試みていることになる．よって，Pacrader とはメモリ検査のタイミングや検査部分が少し異なる．なお，パスワードに用いられうる文字かどうかは，C 言語の `isprint` 関数の返り値が真かどうかで決定している．

この予備実験では，パスワード候補の文字列を取得するたびに端末にそれを表示させた．John the Ripper を追跡する監視プログラムが端末に表示した文字列を，表示順に**図 3** に示す．John the Ripper に総当たり攻撃を実行させた際には，確かに総当たり攻撃に見える文字列が，辞書攻撃を実行させた際には，確かに辞書攻撃に見える文字列が表示されている．John the Ripper の総当たり攻撃では，文字の組の出現頻度に応じた賢い順番でパスワード候補が作成されるため，取得された文字列も単純なアルファベット順にはなっていない．総当たり攻撃では，明らかにパスワード候補ではない文字列も多く表示されており，これらをどう除去するかは今後の課題である．

7. 関 連 研 究

プログラムの動的挙動に関する情報に基づいてマル

ウェアを検出する方法が複数提案されている．文献 12) は，VMM を用いてキーロガーを検出する方法を提案している．文献 8) は，ブラウザ拡張として実現されたスパイウェアを，プログラムの動的解析と静的解析を組み合わせて検出する方法を提案している．これらの研究とは異なり，本研究では，パスワードクラッキングソフトウェアを対象にして検出方法を構築している．

TaintBochs の研究[2] では，CPU シミュレータ上で OS を動かし，パスワードなどの機密情報がどのくらいの期間，アプリケーションのプロセス内に存在するかを調査している．彼らの研究は，仮想マシンを提供するソフトウェアからゲスト OS 内の文字列情報を取得し，アプリケーションの動作を解析する点が本研究と同じであるが，文字列情報の利用法や目的が異なる．

8. 現状と今後の課題

現在，Ubuntu Linux 上で動作する KVM を改造して Pacrader を実装する作業を進めている．検出対象のツールとしては，まずは John the Ripper を使用する予定である．

今後の課題を以下で述べる．第一に，検出処理の効率化である．現状の方式では，小さくないオーバヘッドがゲスト OS の実行に加わるため，改善が必要である．第二に，ゲスト OS への依存を小さくすることである．現状では，ゲスト OS のタスク構造体のレイアウト情報を VMM に与える必要がある．これを排除すれば，ゲスト OS に依存しない形でパスワードクラッキングが検出できる．レイアウト情報が必要な理由はプロセス ID を得るためであるが，Antfarm[7] と同様に，CR3 レジスタの値を仮想的なプロセス ID として用いれば，レイアウト情報は与えなくてもよくなる．ただし，VMM がプロセスを認識する精度は下がるので，トレードオフを考慮する必要がある．第三に，誤検知を少なくすることである．プロセスから検出できる文字列の中には，ファイルの中身やファイルパスもある．特にディレクトリを接頭語に含むファイルパスには注意が必要である．例えば，Firefox のプロセスからは，`/usr/lib/firefox-x.x.xx/` のような接頭語を含む文字列が多数検出される．提案方法は，このような実行を総当たり攻撃であると誤検知する可能性がある．また，辞書を用いる正規のプログラム，例えばスペルチェッカなどの実行を辞書攻撃であると誤検知する可能性がある．第四に，John the Ripper 以外のパスワードクラッキングソフトウェアを用いて実験を行うことである．Linux 上で動作する他のオープンソースソフトウェアとしては，ophcrack[9] などがある．それらについても動作の理解および実験をしていくことが必要である．

謝辞 本研究は JSPS 科研費 23700032 の助成を受けたものです．

参 考 文 献

1) Bloomberg: Amazon.com Server Said to Have Been Used in Sony Attack, `http://www.bloomberg.com/news/2011-05-13/sony-network-said-to-have-been-invaded-by-hackers-using-amazon-com-server.html` (2011).
2) Chow, J., Pfaff, B., Garfinkel, T., Christopher, K. and Rosenblum, M.: Understanding Data Lifetime via Whole System Simulation, *Proceedings of the 13th USENIX Security Symposium*, pp. 321–336 (2004).
3) Cracking Passwords With Amazon EC2 GPU Instances, `http://it.slashdot.org/story/10/11/16/1549245/cracking-passwords-with-amazon-ec2-gpu-instances` (2010).
4) Electric Alchemy: Cracking Passwords in the Cloud: Insights on Password Policies, `http://news.electricalchemy.net/2009/10/password-cracking-in-cloud-part-5.html` (2009).
5) Jiang, X., Wang, X. and Xu, D.: Stealthy Malware Detection and Monitoring through VMM-Based "Out-of-the-Box" Semantic View Reconstruction, *ACM Transactions on Information and System Security*, Vol. 13, No. 2 (2010).
6) John the Ripper, `http://www.openwall.com/john/`.
7) Jones, S. T., Arpaci-Dusseau, A. C. and Arpaci-Dusseau, R. H.: Antfarm: Tracking Processes in a Virtual Machine Environment, *Proceedings of the 2006 USENIX Annual Technical Conference*, pp. 1–14 (2006).
8) Kirda, E., Kruegel, C., Banks, G., Vigna, G. and Kemmerer, R. A.: Behavior-based Spyware Detection, *Proceedings of the 15th USENIX Security Symposium*, pp. 273–288 (2006).
9) ophcrack, `http://ophcrack.sourceforge.net/`.
10) Oyama, Y., Giang, T. T. D., Chubachi, Y., Shinagawa, T. and Kato, K.: Detecting Malware Signatures in a Thin Hypervisor, *Proceedings of the 27th ACM Symposium on Applied Computing*, pp. 1807–1814 (2012).
11) Roth, T.: Breaking encryption in the cloud: GPU accelerated supercomputing for everyone, Black Hat DC 2011 (2011).
12) 小林卓嗣, 山田浩史, 河野健二: 仮想機械技術を利用したキーロガー検知システム, 情報処理学会研究報告, Vol. 2008-OS-107, pp. 1–8 (2008).
13) 河崎雄大, 大山恵弘: VMM を用いたマルウェア検出システムのためのシグネチャデータ更新機能とメモリデータ検査機能, 情報処理学会研究報告, Vol. 2012-OS-122 (2012).