マルチメディア通信と分散処理　106－45
コンピュータセキュリティ　16－45
（２００２．２．１５）

サーバ・クライアント間の同期型情報管理による ソフトウェア保護

中村 直己[†], 西垣 正勝[††], 曽我 正和[†††], 田窪 昭夫[††††]

[†] 静岡大学大学院情報学研究科 〒432-8011 浜松市城北 3-5-1

[††] 静岡大学情報学部情報科学科 〒432-8011 浜松市城北 3-5-1

[†††] 岩手県立大学ソフトウェア情報学部 〒020-0173 岩手県滝沢村滝沢字巣子 152-52

[††††] 三菱電機情報システム製作所 〒247-8520 鎌倉市上町屋 325

E-mail: nisigaki@cs.inf.shizuoka.ac.jp

あらまし：
一般的なソフトウェアの不正使用を防止するために導入されている本人認証のフェーズは、ソフトウェアの主目的処理とは独立していることがほとんどであり、プログラムの改造によって容易にバイパスされてしまう。また、正規ユーザであることを示すパスワード等は単なるマジックワードであることが多く、正規ユーザからパスワードが漏洩するという本質的な問題があった。そこで本稿では、サーバ・クライアント型ソフトウェアをベースに、サーバ・クライアント間で共有する情報を動的に変動させる方式を提案する。サーバ・クライアント間の情報を同期させることにより、認証フェーズとソフトウェアの主目的である処理を密接に結合し、かつ、正規ユーザがパスワードを漏らした場合に本人にデメリットが生じるようにすることができる。

キーワード：ソフトウェア保護, 不正使用防止, プログラム, 同期型情報管理, 動的情報管理

A software protection by synchsronized information management between server and client

Naoki NAKAMURA[†], Masakatsu NISHIGAKI[††], Masakazu SOGA[†††] and Akio TAKUBO[††††]

[†] Graduate school of Information, Shizuoka University,

[††] Department of Computer Science, Faculty of Information, Shizuoka University,
3-5-1 Johoku, Hamamatsu, 432-8011, Japan

[†††] Faculty of Software and Information, Iwate Prefectural University,
Sugo 152-52, Takizawa, Iwate, 020-0173, Japan

[††††] Mitsubishi Electric Corp., 325 Kamimachiya, Kamakura, 247-8520, Japan
E-mail: nisigaki@cs.inf.shizuoka.ac.jp

Abstract:
Application softwares have a user authentication process to prevent the illegal use. However, it is not difficult for a cracker to disable the protection, since the user authentication is generally a extra process for the main routine of the software. Moreover, in this kind of user authentication, the passwords are often leaked out from legal users, since a legal user who leaks out his/her password will not incur any demerit. This paper proposes a software protection using synchronized information management between server and client. By synchronizing server's and client's information, a) the user authentication process can be an indispensable part of the main routine, and b) it is possible to punish the legal user who leaks out his/her password.

Keywords : software protection, protection against illegal use, program, synchronized information management, dynamic information management

1 はじめに

コンピュータとネットワークの爆発的普及により、様々なディジタルデータが物理的な制限なくやりとりされるようになった。ネットワークを通じて不特定多数に瞬時にデータを配信することはもちろん, 最近では P2P 技術により不特定多数間でファイル共有 [1] さえ可能になっている。しかしその技術は正規購入者以外の手にソフトウェアが渡ることも助長している。

これに対し、ソフトウェアメーカも自社の収益・著作権を守るために様々な不正コピー・不正使用防止手段, すなわちプロテクションをソフトウェアに付加するようになった。しかしプロテクションをかけるたびに誰かがそれをクラックし (その情報はネットワークを通して瞬く間に世界中に流される)、更にメーカーが新たなプロテクションを施すというイタチゴッコを繰り広げているのが現状である。

ここで既存のプロテクション技術を分析すると、その多くにおいて使用者の確認部分 (以下、「チェッカ」と呼ぶ) がソフトウェアの主目的である処理と切り離されている事に気付く。これではチェッカ部分のバイパスにより当該ソフトウェアを不正使用することができてしまう。

一方、最近では一般ユーザを対象としたネットワークの常時接続が普及し始めており、販売元のサーバ上で実行するタイプのソフトウェアも現れた。今後のブロードバンド化、マイクロソフトの.NET 構想 [3] やユビキタスコンピューティング [4] 等を考えると、このような新たな形態でのサーバ・クライアント型ソフトウェアは今後ますます発展が見込まれると思われる。サーバ・クライアント型ソフトウェアはソフトウェアの本質的部分がサーバ側に存在するため、プログラムを直接クラックすることは困難であると思われる。しかしユーザ認証を成りすまし等によりすり抜けることができれば、当該ソフトウェアを不正使用することが可能となる。

ただし、成りすましが無意味であるようなサーバ・クライアント型ソフトウェアも存在する。例えば RPG 型のオンラインゲームの場合、ユーザごとにキャラクタや場面が異なっており、ユーザ自身が積極的に認証のためのパスワード等を流すことはない。このようにチェッカ部や認証部がソフトウェアの主目的である処理と密接に関係していれば、チェッカ部のバイパスや認証の成りすましに対抗することができる。

そこで本稿では、サーバ・クライアント型をベースとして、サーバ・クライアント間で共有する情報を動的に変動させることにより、チェッカ部とソフトウェアの主目的である処理を密接に結合させる方式を提案する。以下、2 章でソフトウェアのプロテクション技術をまとめ、問題点を指摘する。3 章で本方式の説明をし、本方式によるソフトウェアの不正使用防止の具体的なシステム例を紹介する。4 章で本稿をまとめる。

2 従来のプロテクト技術

分析をより明確にするため、ソフトウェアを 2 つに分類する。

一般パッケージ型 ：ソフトウェアの本質的部分 (通常はソフトウェアの全て) がユーザの計算機に存在する。現在の大部分のソフトウェアがこのタイプである。

サーバ・クライアント型 ：ソフトウェアの本質的部分または本質的なデータがサーバに存在する。オンラインゲーム等がその代表的な例である。

2.1 一般パッケージ型ソフトウェア

店頭販売や公開サーバからのダウンロードによって手に入るソフトウェアのほとんどが「一般パッケージ型」である。ユーザの手元にソフトウェア全てが存在するため「不正コピーや解析・改竄に弱い」という性質を持つ。

ソフトウェアはディジタルデータであるため、ユーザの手元に渡されてしまっている一般パッケージ型ソフトウェアに対しては基本的にそれをコピーすることができてしまう。よって、「不正コピー」を防止するのではなく、ソフトウェアに何らかの負荷情報を加えることによりその「不正使用」を防止する方法を採らざるを得ない。不正使用防止のために加えられる不可情報としては、シリアルナンバー、特殊ハードの接続、暗号化などが実用化されている。負荷情報の正当性を検査する処理がプロテクションにおけるチェッカ部である。

既存のソフトウェアのチェッカ部はほぼ全て「シリアルナンバー、特殊ハードに記録されているデー

タや復号鍵などの正当性を検査するための値 (以下、「チェックコード」と呼ぶ) を読み取り、その値に応じて処理の条件分岐を行う」ことにより実現されている。チェッカは大きく 4 つのステップに分けることができる。

1) チェックコードを読み込む
2) チェックコードを変数に格納する
3) 変数の値によってフラグを立てる
4) フラグによって条件分岐する

すなわちチェッカ部は使用者認証の作業に特化した完全に独立したサブルーチンであり、ソフトウェアの主目的である処理とは切り離されている。これではチェッカ部分のバイパスにより当該ソフトウェアを不正使用することができてしまう。具体的には 1)〜4) の各ステップに対して例えば以下のようなクラック手法があり、クラッカーがこれらのクラックに 1 つでも成功すれば、無断でインストールしたソフトウェアを不正に使用することができてしまう。

A： 1) →読み込んだチェックコードを遮断し、正しい値を不正に返す
B： 2) →格納された値を正しい値で上書きする
C： 3) →必ずフラグを真にする
D： 4) →条件分岐命令をバイパスする

一般パッケージ型ソフトウェアはプログラムの全てがユーザの手元に存在するので、高度なクラッカーがこれをリバースエンジニアリングして上記 A〜D の改造を施すことが可能である。すなわち一般パッケージ型ソフトウェアにおいてはチェッカ部のクラックを回避することは理論上不可能である。改造されたソフトウェアは (通常は改造個所のバイナリコードを示すという形で) インターネット等により不特定多数に公表されることになる [*1]。

また、例えば起動時にパスワードの入力を求められるようなソフトウェアの場合は、ひとまずそのソフトウェアを正規に購入すればパスワードが与えられるので、リバースエンジニアリングをするまでもなくチェックコードを取得することが可能となる。このようなソフトウェアでは、一般にパスワード等のチェックコードにはチェッカ部を通過するためのマジックワードとしての意味しかなく、正規ユーザはチェックコードを他に漏らしても何らデメリットがない。ソフトウェアのプロテクションを考えるにあたり、正規ユーザ自身からの情報流出は様々なチェッカを揺るがす脅威である。

[*1] ソフトウェア自身を改造するのではなく、フラグが真になる条件を解析して有効なチェックコードをジェネレートするプログラムを作成するようなクラック方法も存在する。

2.2 サーバ・クライアント型ソフトウェア

「サーバ・クライアント型」ソフトウェアはソフトウェアの本質的部分がサーバ側に存在するため、プログラムを直接クラックすることは困難であると思われる。また、ソフトウェアのバージョンアップや一括管理の簡易化などの点で、本来大きなメリットを持っている。PC の能力が向上した今日ではサーバ・クライアント型環境よりも分散コンピューティング環境に注目が集まっているが、一般ユーザのネットワーク常時接続、ブロードバンド化、マイクロソフトの.NET 構想 [3] やユビキタスコンピューティングの普及 [4] 等を考えると、今後販売元のサーバ上で実行されるサーバ・クライアント型ソフトウェアに脚光が集まる可能性も十分見込まれる。

サーバ・クライアント型ソフトウェアの場合、サーバ側で行われる認証により正規使用者であるか否かが認証される。すなわち一般パッケージ型ソフトウェアのチェッカ部に相当するものがサーバでの認証となる。既存のサーバにおける認証のフェーズは本人確認の作業に特化した完全に独立した処理であり、認証後に使用できるソフトウェアの主目的の処理とは無関係である場合が一般的である。よって、クラッカーは成りすましによって当該ソフトウェアの不正使用が可能である。

サーバ・クライアント型ソフトウェアの場合は (サーバのセキュリティ管理が十分になされていれば) 認証フェーズを不正にバイパスされることはないが、認証用パスワードの盗聴・漏洩といった危険性が考えられる。正規ユーザに悪意がない (正規ユーザがパスワードを自ら漏らすことがない) 場合にはワンタイムパスワード等を導入することにより認証を強化することができる。しかし、一般にパスワードは認証を通過するためのマジックワードとしての意味しかなく、正規ユーザはパスワードを他人に漏らしても何らデメリットがない。正規ユーザによりパスワードが不正配布される場合 (ワンタイムパスワード発生ルーチンをリバースエンジニアリングして不正

に解析したり、ワンタイムパスワード発生モジュールを不正に貸し出したりすることを含む) には、成りすましを完全に防ぐ手段はない。

ただし、正規ユーザからの認証用パスワードの漏洩がまずありえないサーバ・クライアント型ソフトウェアも存在する。例えば RPG 型のオンラインゲームではユーザごとにキャラクターや場面が異なっているため、またインスタントメッセンジャー [5] では呼び出す友達リストやメッセージを送る際の ID などが固有なため、正規ユーザは自らのアカウントを他人に使わせようとは思わない。このように認証フェーズがソフトウェアの主目的である処理と密接に関係していれば、正規ユーザからのパスワード漏洩を防ぐことができる。

更に、サーバ・クライアント型ソフトウェアの場合は、サーバ側でユーザのソフトウェア使用状況がリアルタイムで把握できるため、従量課金制での運用が容易である。従量課金制ならば正規ユーザが自らのパスワードを他人に渡すと自らに課金されることになるため、正規ユーザからの漏洩は考慮しなくてよい[*2]。

3 サーバ・クライアント間の同期型情報管理

3.1 同期型情報管理

前章で議論したように、ソフトウェアの不正使用を防止するためには、チェッカ部または認証フェーズをソフトウェアの主目的の処理と密接に関連付けることにより、

- チェッカ部または認証フェーズを単純にバイパスするだけでは不正使用を不可能となるようにする
- 正規ユーザがチェックコードやパスワードを漏らした場合には本人にデメリットが生じるようにする

ことが肝要だという事がわかる。そしてこれを実現するにはサーバ・クライアント型のソフトウェアの形態を採ると都合がよい。そこで本稿では、サーバ・クライアント型ソフトウェアをベースとして、サーバ・クライアント間で共有する各ユーザごとの情報を動的に変動させることにより、認証フェーズとソフトウェアの主目的である処理を密接に結合させる方式を提案する。

ただし本方式では、プライバシ保護やサーバ負荷を考慮して、本質的な部分の内の最小限のプログラムをサーバ側に置く方式を採る。すなわち、プログラムの大部分はクライアントであるユーザの PC 内に存在しており、本方式は分散コンピューティング環境にも適した「一般パッケージ型とサーバ・クライアント型のハイブリッド型のソフトウェア実装方式」であるといえる。

以下、本方式のシステムの実装例として、機能順序変動型ソフトウェアを説明する。

3.2 機能順序変動型ソフトウェア

3.2.1 システム説明

ソフトウェアは様々な機能を実装しているが、一時に使う機能は数個である。そこで本方式ではソフトウェアを機能ごとに分割し、ユーザは実行にあたって適宜必要な機能をサーバに問い合わせる形態を採る。すなわちインストール直後にはユーザの手元にはソフトウェアの起動とサーバへの問い合わせためのプログラムしかない。起動後に、何かの機能を使おうとするたびにそれをサーバから受け取るのである。

ユーザの PC に一旦送られたプログラムモジュール (以下、「機能モジュール」と呼ぶ) は基本的に PC 内にそのまま蓄えられる。ただし、機能を使う度にその格納場所が動的に変動する。格納場所の変動方法については前もってサーバ・クライアント間で共有しておく。そして現時点で各機能モジュールがユーザの PC のどこに格納されているかという情報 (以下、「機能テーブル」と呼ぶ) がサーバ側で管理される。クライアント側には機能テーブルを管理する機能はないことを留意して頂きたい。

本方式の基本的な流れは以下の通りである。

1) クライアントは機能使用の都度、機能テーブルの参照要求をサーバに送る
2) サーバはその参照要求に対してユーザ認証を行う
   2-A) 認証を通れば 3) へ
   2-B) 認証を通らなければ機能を使用できない
3) サーバは参照要求に対応したモジュールを機能テーブルから探す

[*2] ただし、ユーザにとっては完全な従量課金は好まれず、一定金額により使い放題という形態が採られることが多い。

3-A) その機能モジュールがすでに当該ユーザの PC に存在するならばその格納場所をクライアントに返す

3-B) その機能モジュールが PC 内になければ当該モジュールをクライアントに返す

4) クライアントはその機能を使用する

5) クライアントは各機能モジュールの格納場所を変更する

6) サーバは機能テーブルを変更する

なお、3) においてすでに当該機能モジュールが PC 内に存在したとしても、それが旧バージョンであった場合には最新の機能モジュールが返される。また、クライアントにある数以上の機能モジュールがたまった場合や使用してからある程度の時間が経過した機能モジュールに対しては、これらを PC から消去するような方法を採ってもよい。

テキストエディタを例にとり、ユーザ A が初回の起動後にファイルをオープンし (File Open)、テキストに対してコピー (Copy)、ペースト (Paste)、カット (Cut) の順で編集を行った時点のサーバおよびクライアントの状態を概説した図が図 1 である。この後、ユーザ A が更にコピー (Copy) 機能を使うと、サーバおよびクライアントの状態は図 2 のように変更される。なお、この例では機能モジュールの収納場所は LRU(Least Recently Used) 方式で変更させているが、実際にどのような方式を用いるかについては特に制限はない。

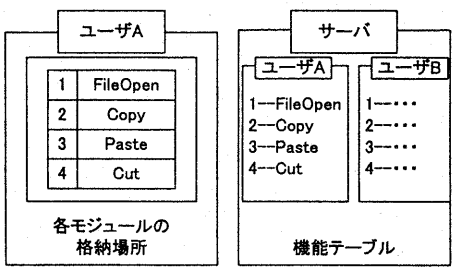

図 1: サーバ・クライアントの状態

3.2.2 考察

不正コピー対策：

ユーザの PC に存在するプログラムには機能テーブルが存在しないため、コピーされても問題ない。したがって、プログラムを不正コピーしただけではソフトウェアを不正使用することはできない。例えばクラッカーが図 1、図 2 で説明したソフトウェアを不正にインストールして、ユーザ A としてサーバにアクセスすることに成功したとしても、その時点でのサーバおよびクライアントの状態は図 3 のようになっており、サーバ側の機能テーブルとクラッカーの PC 内に格納されている機能モジュールの同期が取れておらず、ソフトウェアは正しく動作しない。

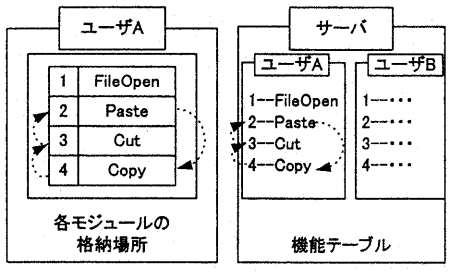

図 2: 図 1 の後に Copy を使った状態

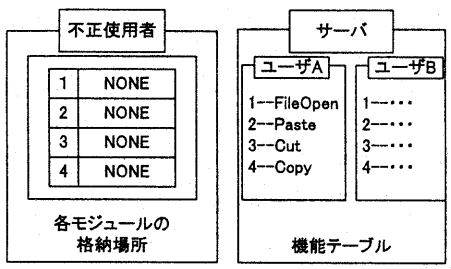

図 3: 同一アカウントを不正使用した場合

リバースエンジニアリング対策：

認証フェーズをバイパスしたソフトウェアがスタンドアロンで動作するようにプログラムを改造したとしても、サーバから機能テーブルの情報を返してもらわなければソフトウェアを使用することができない。更に各機能モジュールの格納場所は動的に変動するので、仮にクラッカーがある時刻の機能テーブルの情報を入手することができたとしても、その後にそれを利用することはできない。全ての機能モジュールが PC に蓄えられた時点で、それ以上は格納場所が変更されないようにソフトウェアを改造し、かつ機能テーブルの情報を手にいれることができた場合には、クラッカーがソフトウェアを不正使用することが可能となる。だが基本的には、サーバのセ

キュリティ管理が十分になされていれば機能テーブルの情報が漏洩する心配はない。

なりすまし対策：

サーバ側の機能テーブルの情報とクライアントのPC内に実際に格納されている機能モジュールの場所が等しくなければソフトウェアは正しく動作しない。すなわち本方式の認証は前節に示した手順2ではなく手順3がその本質と言える。よって手順2の認証においては単純なパスワード方式を用いたとしても、本方式における攻撃耐性は理論的には劣化しない。クラッカーがある時点で正規ユーザAの機能モジュールとパスワードを盗むことができれば、ユーザAに成りすますことが可能である。しかしクラッカーによる不正使用によってサーバ側の機能テーブルとユーザA側の機能モジュールの場所の同期が崩れるため(図4)、ユーザAはそれ以降ソフトウェアを使用することが不可能となり、不正アクセスがあったことに気付くことができる。このように、他人が自らのソフトウェアを利用するとそれ以降に自分がソフトウェアを使用することができなくなってしまうので、正規ユーザが自らのプログラムやパスワードを進んでクラッカーに与えるようなことも阻止できる。なお、各機能を使用する度に認証フェーズを通るため、同一ユーザが同時に複数のマシンからアクセスをしている場合には、いずれかが不正アクセスとして検知できる。すなわちソフトウェアの同時使用は不可能である。

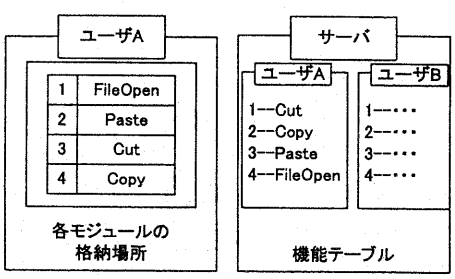

図4: 不正ユーザが使用した直後の正規ユーザAのPCとサーバの状態

クラッカーが正規ユーザと結託すると、以下のような不正が可能である。

1) 正規ユーザのある時点のプログラム全てとパスワードをクラッカーのPCに移す
2) クラッカーがソフトウェアを不正使用する
3) 不正使用後のクラッカーのPC内の全てを正規ユーザのPCに移す
4) 正規ユーザがソフトウェアを使用する

しかしこの場合は、プログラムの全てを毎回コピーしあうのは十分に手間がかかる上、不正に使用できるクラッカーは各時刻で1名のみであるため、それほど大きな問題にはならないと思われる。

4 まとめ

本稿では、ソフトウェアの不正使用防止を目的としてサーバ・クライアント間の同期型情報管理方式を提案した。本方式ではサーバ・クライアント間の情報を同期させることにより、認証フェーズとソフトウェアの主目的である処理を密接に結合し、かつ正規ユーザがパスワードを漏らした場合に本人にデメリットが生じるようにすることができる。具体的なシステム構成も示し、その耐性について考察した。本システムは必要最小限の部分のみがサーバに存在し、分散コンピューティング環境にも適した形態となっている。

今後は、シミュレーション等により本方式のサーバ負荷やネットワークトラフィックを評価するとともに、実際にシステムのプロトタイプを実装する予定である。

参考文献

[1] Jnutella, http://www.jnutella.org/
[2] 稲村他, "ウェアーズの世界", フォレスト出版, 1999
[3] MicroSoft.NET, http://www.microsoft.com/net/
[4] 原田他,"特集：その技術、ユビキタス時代の非常識", 日経エレクトロニクス, 7/30,pp97-141, 2001
[5] MSN Messenger, http://messenger.msn.co.jp/
[6] 中村他, "プログラムの冗長化に関する検討", 情報処理学会研究報告, CSEC2000-10-7, pp41-48, July 2000.