ミニマム・オンライン・アプリケーション方式による プログラムの不正使用防止

西垣正勝[†1] 中村直己[†2] 曽我正和[†3]
田窪昭夫[†4] 中村逸一[†5]

一般的なアプリケーションプログラム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，プログラム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など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．そこで本論文では,「プログラムの本質であり，かつ，各ユーザの使用履歴に依存する必要最小限の情報または機能」をサーバに寄託するミニマム・オンライン・アプリケーション方式を提案する．本方式のプログラムはその本質部分がサーバ上にあるため，ユーザ認証（サーバへのアクセス）をバイパスするとプログラムが使用不能となる．また，ユーザがプログラムを使用するたびにサーバ側の当該ユーザ情報が変化するため，他人が自分の代わりにプログラムを使用するとその後自分がプログラムを使うことができなくなる．本方式は，a) プログラムの本質部分がサーバ上にあるため，プログラムのリバースエンジニアリングは不可能である，b) 正規ユーザがパスワードを漏らすと本人にデメリットが生じるため，正規ユーザからの安易なパスワード漏洩は起こらない，c) 万一プログラムの不正使用が行われた場合に，正規ユーザがその事実に気付く，などの利点を有する．

Minimum Online Applications for Copy Protection of Binary Programs

Masakatsu Nishigaki,[†1] Naoki Nakamura,[†2] Masakazu Soga,[†3]
Akio Takubo[†4] and Itsukazu Nakamura[†5]

Application programs have a user authentication process to prevent the illegal use. However, it is not difficult for a cracker to skip the authentication process. Moreover, most of user authentications have a big security concern; the passwords are often leaked out from “legal users”, since a user who leaks out his/her password will not incur any demerit. This paper proposes to deposit a server with the minimum essential information/functions of a program. By so doing, any cracker who skips user authentication can never use the program since the cracker does not get data on the server. Here, the information/ functions on the server have to vary after each use of the program so that the program’s behavior may differ at each use. Thus, even a legal user is unable to use the program, once his/her password was cracked and the program was used by some body. We call this kind of programs as “minimum online applications”. Minimum online application has the following advantages; a) no cracker can use the program illegally as long as the server keeps the information/functions secret, b) it is possible to punish a user who leaks out his/her password, and c) even if a program is used by cracker, a legal user can suspect that.

†1 静岡大学情報学部情報科学科
Faculty of Information, Shizuoka University
†2 株式会社ネットマークスネットワークセキュリティ事業部
Network Security Division, NETMARKS INC.
†3 岩手県立大学ソフトウェア情報学部
Faculty of Software and Information Science, Iwate Prefectural University
†4 東京電機大学情報環境学部情報環境工学科
School of Information Environment, Tokyo Denki University
†5 株式会社 NTT データセキュリティ事業部
Security Business Division, NTT Data Corp.

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 技術によりエンドユーザ間でのファイル共有[1)]さえ可能となっている．しかし，これらの技術は正規購入者以外の手に不正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），さらにメーカが新たなプロテクションを施すというイタチゴッコが繰り広げられているのが現状である．

本論文ではアプリケーションプログラムの不正使用防止に焦点を当てる．すなわち，文書・画像エディタやゲームソフトなどの各種プログラムがクラッカにより不正使用されることを防ぐことを目的とする（画像データや音楽データなどのコンテンツそのものの不正コピーの防止は対象外である）．

プログラムには基本的に冗長成分がないため，電子透かしを適用してその抑止効果により不正を防止することは難しい．そのため，正規ユーザを確認するための特別な処理ルーチン（以下,「チェッカ」と呼ぶ）をプログラムに付加することにより，不正使用を直接的に阻止するというタイプのプロテクションが行われることが一般的である．ここで，既存のプログラムのプロテクション方式を分析すると，その多くにおいてチェッカがプログラムの主目的である処理と切り離されていることに気付く．これではチェッカ部のバイパスにより当該プログラム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，オンラインゲームのようにメーカのサーバ上で実行されるタイプのアプリケーションプログラムも現れた．今後のブロードバンド化，マイクロソフトの.NET 構想[2]やユビキタスコンピューティング[3]などを考えると，このような形態でのオンライン型アプリケーションは今後ますますの発展が見込まれると思われる．オンライン型アプリケーションはプログラム自身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などによりすり抜けることができれば，当該プログラム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オンライン型アプリケーションも存在する．たとえばロールプレイング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いる．ユーザは自分の代わりに他人にゲームを進めてもらおうとは思わず，クラッカも他人のゲームの続きをプレイすることは楽しくない．よって，正規ユーザが自らの認証パスワードを進んで他人に流すことはなく，クラッカも（嫌がらせが目的ではない限り）他人のパスワードを知ろうとは思わない．

以上のように，チェッカや認証をプログラムの主目的である処理と密接に関連させ，かつ，ユーザ個々人を識別して各ユーザの使用履歴に応じたサービスを提供する形態のプログラムを実装することができれば，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のではないかと考える．そこで本論文は，「プログラムの本質であり，かつ，各ユーザの使用履歴に依存する必要最小限の情報または機能」をサーバに寄託する「ミニマム・オンライン・アプリケーション方式」によるアプリケーションプログラムの実装を提案する．本方式のプログラムはその本質部分がサーバ上にあるため，ユーザ認証（サーバへのアクセス）をバイパスするとプログラムが使用不能となる．また，ユーザがプログラムを使用するたびにサーバ側の当該ユーザ情報が変化するため，他人が自分の代わりにプログラムを使用するとその後自分がプログラムを使うことができなくなる．

以下，2 章でアプリケーションプログラムのプロテクション技術をまとめ，従来のプロテクションの問題点を指摘する．3 章でミニマム・オンライン・アプリケーション方式の説明をし，4 章で本方式による具体的なアプリケーションプログラム不正使用防止システムを 2 例紹介する．5 章で本論文をまとめる．

2. 従来のプロテクション技術

分析をより明確にするため，アプリケーションプログラムを 2 つに分類する．

- スタンドアロン型アプリケーション
  プログラムの本質的部分（通常はプログラムすべて）がユーザの PC に存在する．現在の大部分のアプリケーションプログラムがこのタイプである．
- オンライン型アプリケーション
  プログラム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．いわゆるサーバ・クライアント型のアプリケーションプログラムである．telnet（による外部マシンのリモート操作），オンラインゲーム，WEB の CGI スクリプトなどがその代表的な例である．

2.1 スタンドアロン型アプリケーション

販売されているパッケージソフトのインストールや公開サーバからのダウンロードによって手に入るアプリケーションプログラムのほとんどが「スタンドアロン型」である．PC の性能が飛躍的に向上したため，

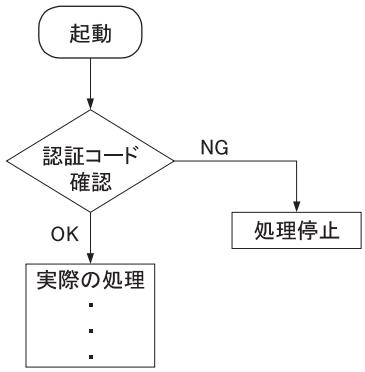

図 1 スタンドアロン型アプリケーションのチェッカ
Fig. 1 Checker in stand-alone applications.

非常に高機能なスタンドアロン型アプリケーションが市場に供されている．プログラムのアクティベーションのためにサーバなどにいったん接続しなければならないアプリケーションも現れている[4]が，ネットワークが普及している今日においてもスタンドアロン型アプリケーションはその主流となっている．

プログラムはディジタルデータであるため，ユーザの手元にプログラムのすべてが存在するスタンドアロン型アプリケーションにおいては，基本的にそのコピーを防止することは不可能である☆．よって,「不正コピー」を防止するのではなく，プログラムに何らかの付加情報を加えることにより「不正使用」を防止する方法をとらざるをえない．不正使用防止のために加えられる付加情報としては，パスワード，プロダクトナンバ，特殊ハードやメディアなどの接続，暗号化などが実用化されている．付加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である．

既存のアプリケーションプログラムのチェッカは，ほぼすべて「パスワード，プロダクトナンバ，特殊ハードやメディアなどに記録されているデータ，復号化鍵などの正当性を検査するための値（以下，これらを総じて「認証コード」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（**図 1**）．

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. 認証コードを読み込む．
2. 認証コードを変数に格納する．
3. 変数の値に応じてフラグを立てる．
4. フラグによって条件分岐する（OK であれば処理を継続し，NG であれば停止する）．

すなわちチェッカは正規ユーザ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，アプリケーションプログラム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のバイパスにより当該プログラム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，たとえば以下のようなクラック手法があり，クラッカ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) →読み込んだ認証コードを遮断し，正しい値を不正に返す．
B. 2) →変数に格納された認証コード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) →条件分岐命令をスキップする（条件分岐命令を NOP 命令に改竄する）．

スタンドアロン型アプリケーションはプログラムのすべてがユーザの手元に存在するので，高度なクラッカがプログラム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．すなわちスタンドアロン型アプリケーションにおいてはチェッカ部のバイパスを阻止することは根本的に不可能である．改造されたプログラムは（通常は改造個所のバイナリコードを示すという形で）インターネットのアンダーグラウンドサイトなどにより不特定多数に公表されることになる．

また，たとえば起動時にパスワードの入力を求められるようなアプリケーションプログラムの場合，ひとまずそのプログラムを正規に購入すればパスワードなどが与えられるので，リバースエンジニアリングをするまでもなく認証コードを取得することが可能である．このようなアプリケーションプログラムにおいては，一般にパスワードなどの認証コードにはチェッカ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，正規ユーザは認証コードを他人に漏らしても何らデメリットがない．アプリケーションプログラム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認証コードの流出は様々なチェッカを揺るがす脅威である．

☆ CD や FD などのメディアに細工をしてメディアの複製を防止する技術が存在するが，高度なクラッカが十分な設備と時間をかければメディア内のすべての情報を取り出すことは理論的には可能である．また，そのメディアがなくてもプログラムの実行が可能となるようにプログラムが改造されてしまった場合には，メディアの複製を阻止しても，プログラムの不正使用防止は達成できない．そこで，本論文では，不正使用防止の本質はメディアの複製防止ではないという観点に立ち，メディアの複製を防止する技術については基本的には本論文の対象外としている．

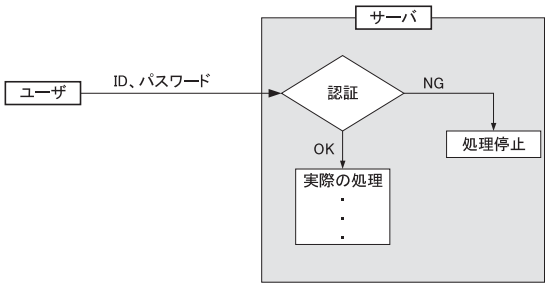

図 2 オンライン型アプリケーションの認証
Fig. 2 Authentication in online applications.

2.2 オンライン型アプリケーション

リモートアクセスによりサーバ上のプログラムを使用する形態が「オンライン型」アプリケーションである．プログラムの本質的部分がサーバ側に存在するため，プログラムを不正コピーしたり，直接クラックしたりすることは困難であると思われる．また，プログラムが 1 カ所で管理されるので，バージョンアップやメンテナンス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[2]やユビキタスコンピューティングの普及[3]などを考えると，今後メーカのサーバ上で稼動・実行されるオンライン型アプリケーション（サーバ・クライアント型のアプリケーション）に再び脚光が集まる可能性も十分見込まれる．

オンライン型アプリケーションの場合，サーバ側で行われる認証により正規ユーザであるか否かが検査される（**図 2**）．すなわちスタンドアロン型アプリケーションのチェッカに相当するものがサーバでの認証となる．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，クラッカは成りすましによって当該プログラムの不正使用が可能である．正規ユーザに悪意がない（正規ユーザが自分のパスワードを他人に漏らすことがない）場合には，ワンタイムパスワードなどを導入して盗聴を防止すれば，認証用パスワードの漏洩を予防することができる．しかし，一般にパスワードに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，正規ユーザは自分のパスワードを他人に漏らしても何らデメリットがない．正規ユーザからクラッカにパスワードが不正配布される場合☆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漏洩の心配がほとんどないオンライン型アプリケーションも存在する．たとえばロールプレイング型のオンラインゲームではユーザごとにキャラクタや場面が異なっている．また，インスタントメッセンジャ[5)~7)]では呼び出す友達リストやメッセージを送る際の ID などがユーザごとに固有となっている．よって，正規ユーザは自分の代わりに他人にゲームを進めてもらおうとは思わず，また，他人に自分のアカウントを使わせようとは思わない．そして，クラッカも（嫌がらせが目的なのではない限り）他人のゲームの続きをプレイしたり，他人に成りすまして他人の友人と情報共有をしたりすることは楽しいとは思わない．このため，正規ユーザが自分の認証用パスワードを進んで他人に流すことはなく，クラッカも他人のパスワードを知ろうとは思わない．

以上より，ユーザ個々人を識別して各ユーザの使用履歴に応じたサービスを提供する形態のプログラムを実装することができれば，認証の成りすましに対抗することができるのではないかと考える．**図 3** に通常の認証（タイプ 1）とユーザごとに個別化されたプログラムの認証（タイプ 2）の違いを概念的に示した．

さらに，ユーザ個々人を識別して各ユーザに応じたサービスを提供する場合は，サーバ側で各ユーザのプログラム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り，超流通[8)]とも親和性が高い．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使用させると自らに課金されることになるため，正規ユーザからのパスワードの漏洩はありえない．

しかし，サーバ側がプログラムの使用状況を把握できるということは，その一方で，誰がいつどのような処理を行ったかという情報やその処理において使用されたデータがサーバ側に筒抜けになるということを意味する．これはプライバシや著作権の保護という観点から好ましくない．これらについては，サーバ側にすべてのプログラムを置くのではなく，処理の一部のみをサーバで実行する「プログラムの虫食い実装方式[9)]」などにより，その問題を軽減することが可能である．また，虫食い実装方式はサーバの負荷やネットワークのトラフィックを軽減させる意味でも有効である．ただし，文献 9) の虫食い実装方式は「ユーザ個々人を識

☆ 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．

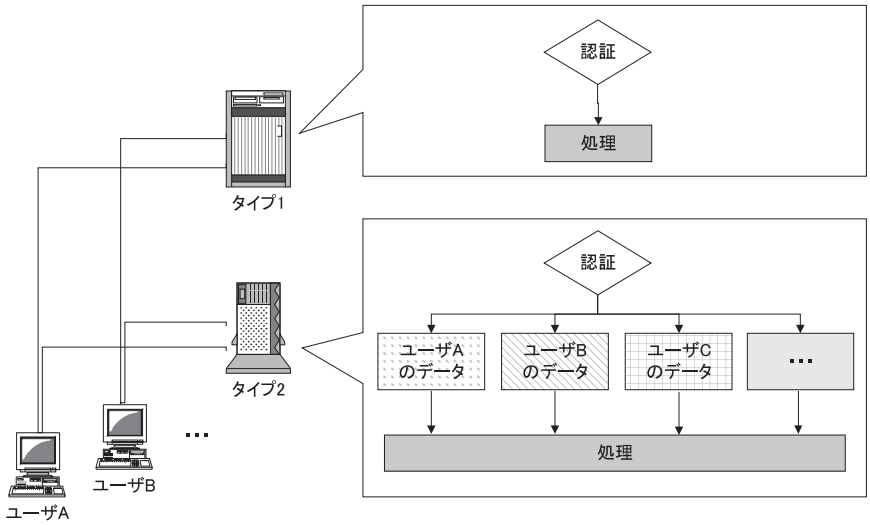

図 3 認証の方式の違い
Fig. 3 Authentications in the conventional application and online game.

別して各ユーザの履歴に応じたサービスを提供する」というタイプの方式ではなく，正規ユーザからのパスワード漏洩の阻止を狙ったものではない．

3. ミニマム・オンライン・アプリケーション方式

3.1 ユーザ依存部の寄託

前章で議論したように，プログラムの不正使用を防止するためには，

- チェッカ（認証）とプログラムの主目的である処理を密接に関連させることにより，チェッカ部（認証フェーズ）を単純にバイパスするだけではプログラムを不正使用することは不可能となるようにする，
- ユーザ個々人を識別して各ユーザの使用履歴に応じたサービスを提供する形態のプログラムとすることにより，正規ユーザがパスワードや認証コ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ことが分かる．そこで本論文は，「プログラムの本質であり，かつ，各ユーザの使用履歴に依存する必要最小限の情報または機能」のみをサーバに寄託する「ミニマム・オンライン・アプリケーション方式」によるアプリケーションプログラムの実装を提案する．本方式のプログラムはその本質部分がサーバ上にあるため，ユーザ認証（サーバへのアクセス）をバイパスするとプログラムが使用不能となる．また，ユーザがプログラムを使用するたびにサーバ側の当該ユーザ情報が変化するため，他人が自分の代わりにプログラムを使用するとその後自分がプログラムを使うことができなくなる．

具体的には，ミニマム・オンライン・アプリケーション型のプログラムの実装方法および使用方法は以下のようになる．

1. スタンドアロン型アプリケーションの中で，各ユーザごとに動作が異なる部分（以下,「ユーザ依存部」と呼ぶ）を切り出す．ここでユーザ依存部とは，ユーザ ID を入力とし，各ユーザごとに固有であるなんらかのデータ（以下,「ユーザ依存データ」と呼ぶ）を出力するサブルーチンである．ユーザ依存データは，ユーザの当該プログラムの使用履歴に応じてそのつど変化する．
2. スタンドアロン型アプリケーションのチェッカ部とユーザ依存部を取り除き，サーバ側にそれらに相当する機能を実装する．サーバは全ユーザの当該プログラムの使用履歴を管理しており，ユーザからの要求に応じ，ユーザ認証とユーザ依存データの生成を行う．
3. ユーザの PC には，チェッカ部とユーザ依存部が取り除かれたアプリケーションプログラムがインストールされる．PC 側のプログラムの動作はサーバから送られてくるユーザ依存データにより異なるため，正当な使用履歴から作られたユーザ依存データをサーバから返信してもらわないと PC のアプリケーションプログラムは正しく動かない．
4. PC にインストールされたアプリケーションプログラムを使用する場合には，ユーザは必ずサーバに認証を依頼してチェッカ部とユーザ依存部の処

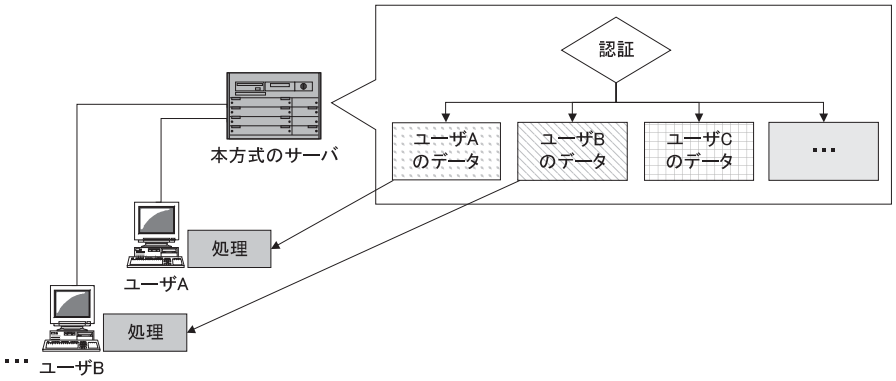

図 4 ミニマム・オンライン・アプリケーション方式の認証
Fig. 4 Authentication in minimum online applications.

理を代行してもらう．正規ユーザであれば，ユーザ依存部の処理結果としてサーバから正しいユーザ依存データが PC に返される．

5. 正規ユーザは，サーバから返された自分用のユーザ依存データを用いることにより，自分の PC 内のアプリケーションプログラムを正しく使用することができる．

ここで重要な前提は,「ユーザ依存データはプログラムの実行に対して必要不可欠なものであり，かつ，各ユーザの当該プログラムの使用履歴に応じて変化するものである」ということである．すなわち，クラッカが正規ユーザ A の PC 内のプログラムを不正コピーして自分の PC にインストールし，ユーザ A に成りすましてプログラムを使用した場合，サーバ側のユーザ A の使用履歴とユーザ A の PC 側の使用履歴に齟齬が生じることになり，その後は正規ユーザ A さえも当該プログラムを使用することができなくなる．以上の結果，

a) プログラムの本質部分がサーバ上にあるため，プログラムのリバースエンジニアリングは不可能である，

b) 正規ユーザがパスワードを漏らすと本人にデメリットが生じる（その後，本人もプログラムを使えなくなる）ため，正規ユーザからの安易なパスワード漏洩は起こらない，

c) 万一プログラムの不正使用が行われた場合に，正規ユーザがその事実に気付く，

などの性質を兼ね備えるアプリケーションプログラムが実現する．

従来のスタンドアロン型アプリケーションを本方式の形態に移行させることにより，アプリケーションプログラムの不正コピー防止効果を格段に高めることができると期待される．図 3 に対比した形式で，ミニマム・オンライン・アプリケーション方式の認証の概念を**図 4** に示す．

本方式は，プライバシ保護やサーバ負荷を考慮して，アプリケーションプログラムの中の本質的な最小限の部分のみがサーバ側に置かれるようになっている．すなわち，プログラムの大部分はユーザの 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「スタンドアロン型アプリケーションとオンライン型アプリケーションのハイブリッド型実装方式」であるともいえるだろう．

3.2 クラッキングに対する耐性

ミニマム・オンライン・アプリケーション方式のプログラムは，

1. サーバ側で認証を行い，ユーザを特定し，当該ユーザのプログラム使用履歴に応じたユーザ依存データをユーザの PC に送信する，

2. サーバから送られてきたユーザ依存データを用いて，ユーザの PC でプログラムが動作する，

という仕組みで動作する．よって，クラッカが本方式のプログラムを不正使用する場合には，

i. サーバからの情報がなくてもアプリケーションプログラムが動作するように PC 側のプログラムを改造する，

ii. サーバ側の処理をエミュレートするプログラムを作成して，不正にユーザ依存データを取得できるようにする，

のいずれかのクラッキングを試みるものと考えられる．

i に対する耐性を高めるためには，アプリケーションプログラムの本質的な部分を見極め，これをサーバ側に寄託する必要がある．ii に対する耐性を高めるためには，サーバ側で行われる処理内容を十分複雑にし，サーバ側の動作が容易に類推されないようにしなければならない．これらの条件が満たされるようにユーザ

依存部を適切に設定することができたならば，サーバのセキュリティ管理が十分になされている限り，ミニマム・オンライン・アプリケーション方式により実装されたプログラムは 3.1 節に示された a)〜c) の性質を満足することが保証される．本論文では，4 章でミニマム・オンライン・アプリケーション方式の具体的な実装方式を 2 例示し，本方式が現実にどの程度のセキュリティ強度およびパフォーマンスを有するか詳細な検討を行う．

なお，高度なクラッカといえども，機械語プログラムの全文をリバースエンジニアリングすることはやはり容易なことではないと思われる．たとえば 2.1 節の D のタイプのクラッキングは，

1. デバイスや外部コンピュータにアクセスするためのシステム関数コールやセキュリティ違反に関する警告メッセージの文字列を手がかりにして，チェッカ部のルーチンをある程度の粒度で推測する，
2. クラッカはチェッカ部のルーチン近辺の機械語のみを解読する，
3. チェッカの条件分岐命令を見つけて，これを NOP 命令に置き換える，

という手順で完了するので，クラックは比較的容易である．一方，

- チェッカを無効化するためには，プログラム中のいろいろなルーチンの中の複数の命令を変更しなければならない，
- チェッカを無効化するためにある命令を変更すると，プログラム全体にその影響が現れる（その影響が及ぶすべての命令を見つけて，修正しなければならない），

ようになっている場合には，クラックは急激に難しくなると考えられる．そこで，4 章では「プログラムの改造に手間がかかるか否か」という観点からクラッキングに対する耐性を測ることにする．なお，ソフトウェアメーカが用意しているサーバはしかるべき管理者により十分な管理がなされると考えてよいと思われるので，サーバへの不正アクセスはないという仮定をおくことにする．

4. ミニマム・オンライン・アプリケーション方式の実装例

4.1 機能順序変動型アプリケーション方式

4.1.1 システム説明

アプリケーションプログラム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．そこで本方式ではプログラム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とる．すなわちインストール直後にはユーザの手元にはアプリケーションの起動とサーバと通信する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当該機能のプログラムモジュール（以下,「機能モジュール」と呼ぶ）をサーバから受け取るのである．

ユーザの PC にいったん送られた機能モジュールは基本的に PC 内にそのまま蓄えられる．ただし，機能を使うたび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・PC 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）がサーバに寄託され，サーバ側で管理される．各ユーザの PC のプログラムには機能テーブルを管理する機能はないことに留意されたい．当然，機能テーブルの情報は各ユーザごとに異なっており，サーバは全ユーザの機能テーブルを管理している．

ユーザは自分の PC にインストールされているプログラムを実行する際には，各機能の使用ごとにサーバにアクセスし，当該機能の格納場所を教えてもらう必要がある．サーバから返される各機能の格納場所の情報が，3.1 節の説明における「ユーザ依存データ」に相当する．

本方式の基本的な流れは以下のとおりである．

1) PC は任意の機能を使用しようとするたびに，機能使用の要求をサーバに送る．
2) サーバはその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は使用できない．
3) サーバは要求に対応した機能モジュールを当該ユーザの機能テーブルから探す．
  3-A) その機能モジュールがすでにユーザの PC に存在するならばその格納場所を PC に返す．
  3-B) その機能モジュールが PC 内になければ当該機能モジュールを PC に送信する．
4) ユーザは PC でその機能を使用する．
5) PC 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，PC 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

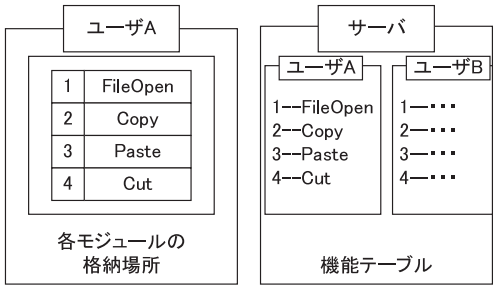

図 5 サーバと PC の状態
Fig. 5 Conditions of server and PC.

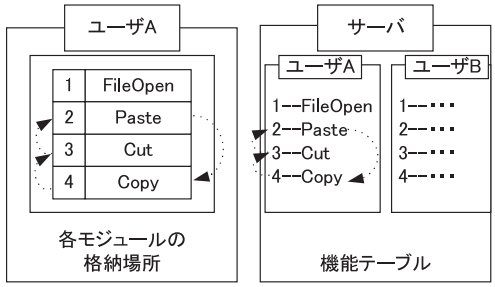

図 6 図 5 の後に Copy を使った状態
Fig. 6 Condition change caused by Copy command.

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 PC の状態を概説した図が**図 5** である．この後，ユーザ A がさらにコピー（Copy）機能を使うと，サーバおよび PC の状態は**図 6** のように変更される．なお，この例では機能モジュールの格納場所は LRU（Least Recently Used）方式で変更させているが，実際にどのような変更方式を用いるかについては特に制限はない．

4.1.2 評 価

(1) 不正コピーに対する耐性

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，PC 内のプログラムが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プログラムを不正使用することはできない．たとえば図 6 の状態の際に，クラッカが新たに本アプリケーションプログラム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したとしても，その時点でのサーバおよびクラッカの PC の状態は**図 7** のようになっており（クラッカのプログラムはインストールした直後の状態であるので，機能モジュールがまだ 1 つも存在しない），サーバ側の機能テーブルとクラッカの PC 内に格納されている機能モジュールの同期がとれておらず，プログラムは正しく動作しない．

(2) プログラムの改造に対する耐性

機能テーブルはプログラムの本質的部分である．認証フェーズをバイパスしてアプリケーションプログラム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プログラムを使用することができない．よって，3.2 節の i に対する耐性を有する．

3.2 節の ii に関しては，クラッカに対して機能テーブルの情報および機能モジュールのオブジェクトコードを隠蔽できなければならない．サーバからそれらが直接漏洩することはないという仮定をおいたとしても，クラッカが

a) 機能モジュールの使用要求に対するサーバからの返信を収集・解析することにより，機能テーブルの情報や機能モジュールのオブジェクトコードを少しずつ不正に獲得する

ことは可能性である．そして，すべての機能モジュールが PC に蓄えられた時点で，クラッカが，

b) 機能モジュールの格納場所の変更方式を解析して，サーバ機能をエミュレートするプログラムを作成する

ことに成功するか，または，

c) それ以上，機能モジュールの格納場所が変更されないように PC 内のプログラムを改造する

ことができれば，その後，クラッカがアプリケーションプログラムを不正使用することが可能となる．

a) に関しては，サーバとの通信を SSL などにより暗号化すれば，通信文を単純に盗聴するだけではサーバから送られる機能テーブルや機能モジュールのデータを収集することはできなくなる．b)，c) に関しては，クラッカは機能モジュールの格納場所を変更するルーチンを完全に解析し，エミュレータを作成するか，または，当該ルーチンを無効化しなくてはならない．よって，従来のスタンドアロン型アプリケーションと比べ，本方式のプログラムのクラッキングにかかる手間は決して小さくはないと思われる．ただし，これらのクラッキングは十分な技術と時間を有するクラッカにとっては不可能なことではない．

(3) 成りすましの防止

サーバ側の機能テーブルの情報と 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プログラムは正しく動作しない．すなわち本方式の認証は 4.1.1 項に示した手順 2) ではなく手順 3) がその

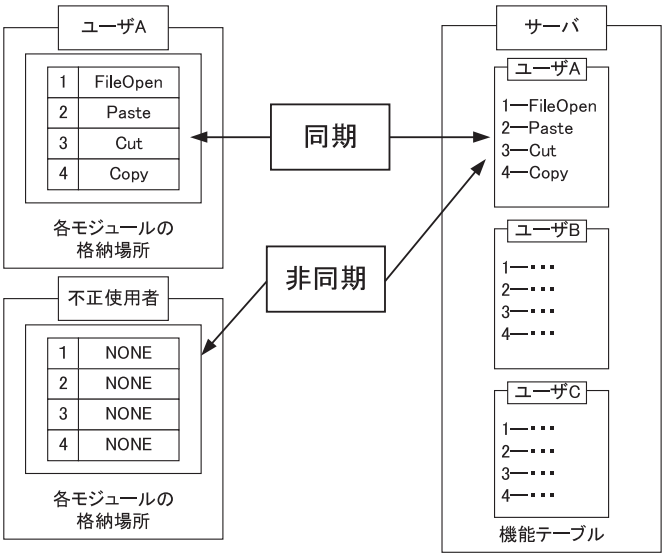

図 7 図 6 の際にクラッカがユーザ A のアカウントを不正使用した状態
Fig. 7 Conditions of server and cracker's PC.

本質といえる．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低下しない．

クラッカ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み，ユーザ A の PC の状態を完全にコピーすることができれば，ユーザ A に成りすますことが可能である．ただし，ユーザ A の機能モジュールとパスワードが盗まれたとしても，クラッカよりも先にユーザ A がプログラムを使用すれば，機能テーブルが更新され，盗まれた情報は使いものにならなくなる．

たとえ，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取得したクラッカがユーザ A に成りすましてプログラムを使用したとしても，クラッカの不正使用によってユーザ A とサーバの間の機能テーブルの同期が崩れることになる（**図 8**）．すなわち，ユーザ A はそれ以降プログラム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．また，他人が自らのアプリケーションプログラムを利用するとそれ以降，自分がアプリケーション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に教えるようなことも起こらないと思われる．

各機能を使用するたび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．すなわち複数人によるプログラムの同時使用は不可能である．

クラッカ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すべてとパスワードをクラッカの PC に移す．
2) クラッカがプログラムを不正使用する．
3) 不正使用後のクラッカの PC 内のすべてを正規ユーザの PC に移す．
4) 正規ユーザがプログラムを使用する．

特に，正規ユーザがクラッカの便宜を図ってアプリケーションプログラムを共有ディレクトリに置くことにより，この不正は容易となる．しかしこのような不正を行ったとしても，プログラムを使用できるクラッカは各時刻で 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考える．

(4) プライバシの保護

本方式においてサーバが把握できるのは，「ユーザがいつ，どの機能を使用したか」という情報のみである．たとえば上記のテキストエディタの例では，ユーザが編集している文章データそのものはいっさいサーバ側に漏れることはなく，プライバシも保護される．

(5) パフォーマンス

機能モジュールがユーザの PC にいったん格納され

☆ アプリケーションプログラムを共有ディレクトリに置いたとしても，ユーザがプログラムを使用する時点においてはプログラム（および機能テーブル）はユーザの PC のメモリにローディングされて実行されているので，ユーザとクラッカが同時にプログラムを使用すると機能テーブルの一貫性が崩れ，それ以降プログラムを使うことができなくなる．

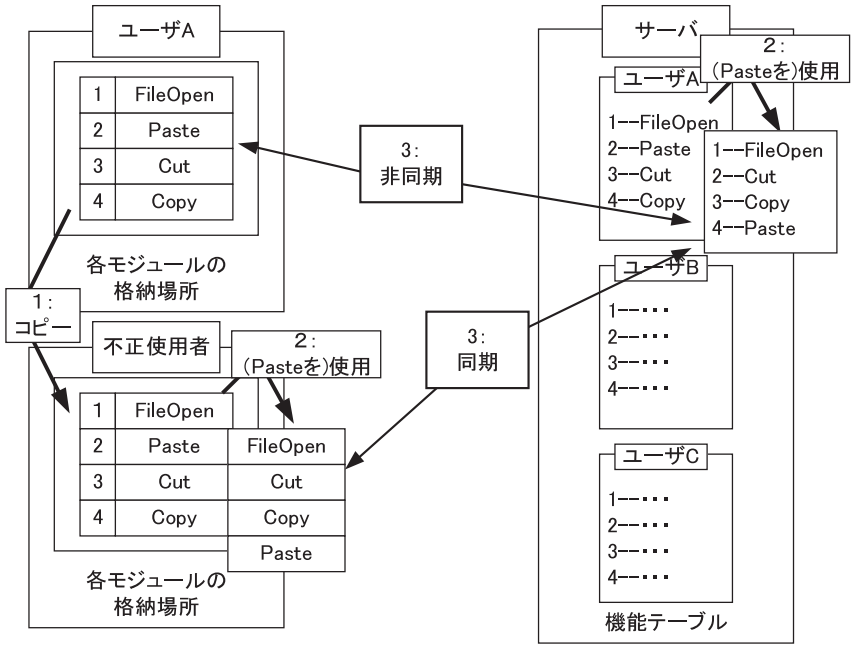

図 8 不正ユーザが使用した直後の正規ユーザ A とサーバの状態
Fig. 8 Condition change caused by illegal use.

てしまえば，プログラムのほとんどがユーザの PC 内で処理されるので，本方式は PC の性能を十分に活用できる．すなわち本方式は，オンライン型アプリケーションの利点を持ちつつ，かつ，分散コンピューティング環境に適した実装形態となっている．

各機能モジュールを使用するたびにサーバに要求を送り，機能モジュールの格納場所を受けとる（さらに，初めて使う機能の場合はオブジェクトコードも返される）ため，ユーザ数，サーバの処理能力，ネットワーク・トラフィックなどによってはプログラムのパフォーマンスが低下する可能性がある．ただし，オンライン・ストレージ・サービスなどが現実のものとなっている今日のネットワーク環境[10]においては致命的な機能低下が頻繁に発生することはないと思われる．

また，本方式においては，今から使用する機能モジュールがすでに PC 内に存在していたとしても，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サーバから送られてくるようになっている．すなわち，ソフトウェアメーカはサーバ上の機能モジュールのみをアップデートしておけば，すべてのユーザの PC に最新の機能モジュールが順次取り込まれていくことなる．よって，本方式はプログラムのアフターケア（バージョンアップサービス）に関するコストパフォーマンスも高いといえる．

(6) セキュリティとユーザビリティのトレード オフ

一般にセキュリティとユーザビリティ（利便性）はトレード オフの関係にある．本方式においては，正規ユーザの機能モジュールを自分の PC に不正コピーしたクラッカがプログラムを使用すると，正規ユーザとサーバの間の機能テーブルの同期が崩れるため，

- 正規ユーザがクラッキングに気付くことができる，
- 正規ユーザが容易にパスワードなどを他人に教えない

という効果が期待できる反面，

- 正規ユーザであっても 2 台目以降の PC で当該プログラムを使用できない，
- 一度クラッキングされると，被害者である正規ユーザもそれ以降，プログラムを使用できなくなる（これは，クラッカによるサービス妨害攻撃が容易であることにも通ずる）

という不便さが発生することになる．

一方，現在は共有ディレクトリやオンライン・ストレージ環境も普及しているため，アプリケーションプログラムを共有ディレクトリやオンライン・ストレージに保存することも考えられる．この場合，

- （4.1.2 項 (3) で述べたように）クラッカが正規ユーザと結託した場合，クラッカがプログラムを不正使用することができる，
- クラッキングにあっても気付かない

というセキュリティホールが発生してしまうが，

- 正規ユーザが任意の PC で当該プログラムを使用可能，
- クラッカによる不正使用の後に正規ユーザがプログラム使用不可になることがない

いう意味では利便性が確保される．

セキュリティとユーザビリティのどちらを重視するかは，ユーザやアプリケーションの性質などにより左右されると思われるが，本論文では

a) 現在の個人用途のソフトウェアの販売においては，PC1 台ごとのライセンス形態となっていることが多いため，2 台目以降の PC で当該プログラムを使用できないことは一概に利便性の欠如とはいえない，

b) 基本的に正規ユーザは不正アクセスの攻撃を受けた際にはその事実を知りたいと思う場合が多く，かつ，不正アクセスに気付いた後に正規ユーザの環境を復旧することは運用により対処が可能である

という観点から，本方式のアプリケーションプログラムを共有ディレクトリやオンライン・ストレージ環境におく実装形態を採るべきではないと考える．

（4.1.2 項 (3) で述べたように）プログラムを共有ディレクトリやオンライン・ストレージにおいて共用したとしても，当該プログラムを使用できるのは各時刻で 1 人のみなので，他人が使い終わるのを待たなくてはならない．よって，特に誰もが頻繁に使用するアプリケーションプログラムに対しては，実際に本方式のプログラムが不正に共用されることは少ないのではないかと期待される．しかし本来ならば，プログラムを共有ディレクトリやオンライン・ストレージにおくことを根本的に阻止する技術を採用すべきであろう．たとえば，以下のような方法が考えられるが，詳細については今後の課題としたい．

- PC 内に格納される各機能モジュールの一部を携帯デバイスに格納する．携帯デバイスを PC に挿入しないとプログラムを使用することができない．ある時点で携帯デバイスを複製したとしても，携帯デバイス内の各機能モジュールの位置がプログラム使用のたびに動的に変更されるので，その後に使い続けることができるのは本物または複製のいずれか 1 つの携帯デバイスのみである．
- 機能テーブルに加え，ユーザが他人に見られたくない/使用されたくない/変更されたくない何らかの情報（の一部）をもサーバ側に寄託する．これにより，そもそもプログラムを他人と共用するということが抑止される．

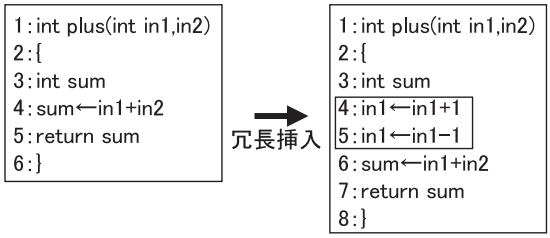

図 9 冗長命令挿入例
Fig. 9 An example of redundant programs.

4.2 冗長命令変動型アプリケーション方式

4.2.1 システム説明

文献 9) の虫食いプログラム方式は，プログラムの一部をサーバに置いてスタンドアロン型アプリケーションをオンライン化するという観点でミニマム・オンライン・アプリケーション方式と同様のコンセプトを有しているが，「ユーザ個々人を識別して各ユーザの履歴に応じたサービスを提供する」というタイプの方式ではないため，正規ユーザからのパスワード漏洩に対する解決策とはなっていなかった．虫食いプログラム方式に対して本節に示すような拡張を施すことにより，虫食いプログラム方式をミニマム・オンライン・アプリケーション方式へと改良することができる．

本方式では，虫食い部のプログラムに動的に冗長命令を付加し，この情報をサーバ・PC 間で分散管理する．ここで，冗長[11]とはプログラムの処理結果に影響を与えない命令群のことであり，たとえば**図 9** の例では右のプログラムの 4 行目：“in1 ← in1+1” と 5 行目：“in1 ← in1-1” がこれにあたる．なお，虫食い部はプログラムの実行にあたり必要度の高い部分が選ばれる．

虫食い部のプログラム（サーバに格納されている）に冗長命令が挿入されている．冗長命令はプログラムを使用するたびに動的に変更される．サーバ側で冗長命令が変更されるたびに冗長命令が何行目に挿入されているかの情報（以下，「冗長箇所情報」と呼ぶ）が PC に通知される．なお，虫食い部は全ユーザで共通としてもよいが，冗長命令の挿入箇所はユーザごとに異なる．サーバは全ユーザの冗長命令付き虫食い部を管理している．

ユーザは自分の PC にインストールされているプログラムを使用する際には，虫食い部の処理を行うごとにサーバにアクセスし，引数と冗長箇所情報を送り，処理結果を返してもらう必要がある．サーバに寄託される虫食い部のプログラムが 3.1 節の説明における

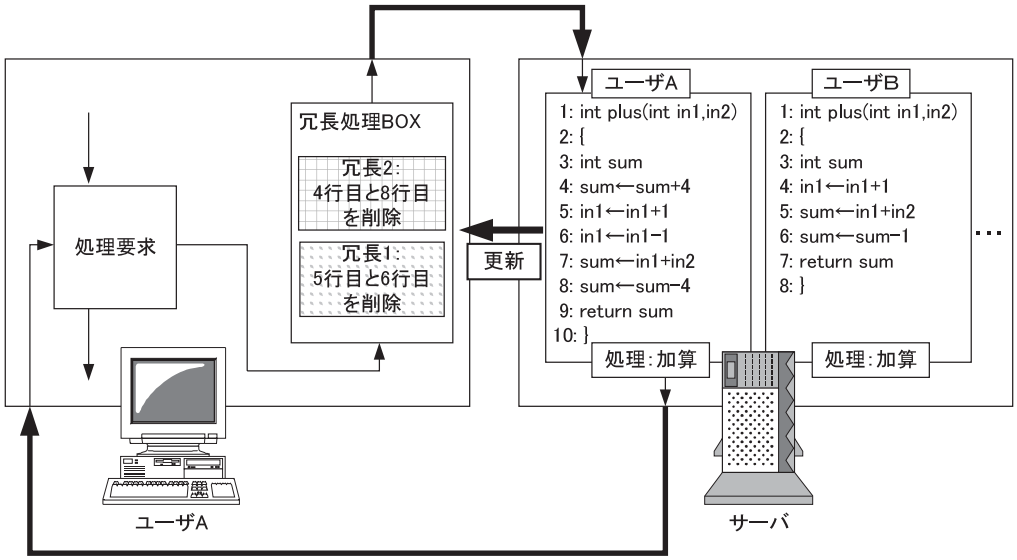

図 10 冗長命令変動型アプリケーション方式
Fig. 10 Minimum online application with redundant program.

「サーバ依存部」に相当し，サーバから返される虫食い部の処理結果が「ユーザ依存データ」に相当する．

本方式の基本的な流れは以下のとおりである（**図 10**）．

1) ユーザは虫食い部の処理を実行しようとするたびに，処理依頼とともに冗長箇所情報（冗長命令が挿入されている行番号）をサーバに送る．

2) サーバはユーザが指定した行を虫食い部から削除したうえで，虫食い部の処理を実行する．

3) サーバは虫食い部のプログラムに新たな冗長命令を付加する．

4) サーバは処理結果と共に新たな冗長箇所情報（新たに挿入された冗長命令セットの行番号）をユーザに返す．

冗長命令のセットは削除されてもプログラムの実行に影響はない．上記の手順 1) で冗長命令の行番号を正しく指定できた者だけが本プログラムを正常に使用することができる．

4.2.2 評　　価

(1) 不正コピーに対する耐性

ユーザの PC に存在するプログラムには虫食い部が存在しないため，PC 内のプログラムがコピーされても問題ない．すなわち，プログラムを不正コピーしただけではプログラムを不正に使用することはできない．

(2) プログラムの改造に対する耐性

認証フェーズを単純にバイパスしてアプリケーションプログラム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，サーバから虫食い部の処理結果を返してもらわなければプログラムを使用することができない．すなわち，当該プログラムを不正使用するためには，クラッカは虫食い部の処理を解析し，PC 内のプログラムを改造して虫食い部を補うか，または，サーバ機能を代行するエミュレータを外部に作成する必要がある．しかし，「プログラムの本質的部分であり，かつ，処理が十分複雑である部分[9]」がプログラムの虫食い部として適切に選ばれているならば，クラッカはサーバに渡される引数とサーバから返される処理結果から虫食い部の処理を推測することはできないと考えられる．サーバ側の処理が類推できない以上，クラッカが盲目的に PC 内のプログラムをいくら改造したとしてもアプリケーションプログラムを不正に使用することは不可能である．よって，サーバのアクセス管理が十分で虫食い部の処理内容が秘匿されている限り，本方式は 3.2 節の i)，ii) に対する耐性を有する．

なお，冗長命令の挿入箇所が漏洩しても，虫食い部の処理内容が守られている限り，プログラムを改造してアプリケーションプログラムを不正に使用することはできないことに注意されたい．(3) に示すように，冗長命令の挿入箇所の漏洩はプログラムの改造に対してではなく，成りすましに対する脅威となる．

4.1 節の機能順序変動型アプリケーション方式においては，いかなるプログラムであってもその機能テーブル部分を切り出してサーバに寄託しさえすればミニマム・オンライン・アプリケーションとして実装されうるが，高度な技術を持つクラッカが十分な時間をか

けるとプログラムを改造してこれを不正使用することが可能であるという問題があった．一方，冗長命令変動型アプリケーション方式では，クラッカがサーバ側の処理を類推できないため PC 側のプログラムを改造してもアプリケーションプログラムを不正使用することはできなくなっているが，全体のプログラムの中から虫食い部として適切な部分を選ぶ方法がまだ確立されておらず，実装が難しい☆．

(3) 成りすましの防止

クラッカがある時点で正規ユーザ A の冗長箇所情報を知ることができれば，ユーザ A に成りすますことが可能である．ただし，ユーザ A の冗長箇所情報が盗まれたとしても，クラッカよりも先にユーザ A がプログラムを使用すれば，冗長箇所情報が更新され，盗まれた情報は使いものにならなくなる．

たとえ，ある時点で正規ユーザ A の冗長箇所情報を取得したクラッカがユーザ A に成りすましてプログラムを使用したとしても，クラッカの不正使用によってユーザ A とサーバの間の冗長箇所情報の同期が崩れることになる．すなわち，ユーザ A はそれ以降プログラム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．また，他人が自らのアプリケーションプログラムを利用するとそれ以降，自分がプログラム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冗長箇所情報を進んでクラッカに教えるようなことも起こらないと思われる．

虫食い部の処理をしようするたびに認証を行うようにすれば，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．すなわち複数人によるプログラムの同時使用は不可能である．

クラッカ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すべてをクラッカの PC に移す．
2) クラッカがプログラムを不正使用する．
3) 不正使用後のクラッカの PC 内のすべてを正規ユーザの PC に移す．
4) 正規ユーザがプログラムを使用する．

特に，正規ユーザがクラッカの便宜を図ってアプリケーションプログラムを共有ディレクトリに置くことにより，この不正は容易となる．しかしこのような不正を行ったとしても，プログラムを使用できるクラッカは各時刻で 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考える．

(4) プライバシの保護

本方式においては，虫食い部のルーチンに渡される引数に関してはサーバがこれを知ることができる．虫食い部を適切に選出することにより，引数から PC 上のデータや処理が類推されないようにする工夫が必要となる．

4.1 節の機能順序変動型アプリケーション方式は機能テーブルがサーバで管理されているにすぎないため，ユーザ側のデータがサーバに漏洩することはないが，高度な技術を持つクラッカが十分な時間をかけるとプログラムを改造してこれを不正使用することが可能であるという問題があった．一方，冗長命令変動型アプリケーション方式では，クラッカがサーバ側の処理を類推できないため PC 側の虫食いプログラムを改造してもアプリケーションを不正使用することはできない反面，プライバシがある程度侵害される．

(5) パフォーマンス

プログラムのほとんどがユーザの PC 内で処理されるので，本方式は PC の性能を十分に活用できる．すなわち本方式は，オンライン型アプリケーションの利点を持ちつつ，かつ，分散コンピューティング環境に適した実装形態となっている．

虫食い部の処理がサーバに任されるため，ユーザ数，サーバの処理能力，ネットワーク・トラフィックなどによってはアプリケーションのパフォーマンスが低下する可能性がある．虫食い部として大規模または複雑な処理がサーバに寄託されているほど，この問題は大きくなると思われる．

(6) セキュリティとユーザビリティのトレード オフ

一般にセキュリティとユーザビリティ（利便性）はトレード オフの関係にある．本方式においては，正規ユーザの冗長箇所情報を自分の PC に不正コピーしたクラッカがプログラムを使用すると，正規ユーザとサーバの間の冗長箇所情報の同期が崩れるため，

- 正規ユーザがクラッキングに気付くことができる，
- 正規ユーザが容易にパスワードなどを他人に教えない

☆ 文献 9) に，虫食い部を選定する際の条件となる「虫食い部が備えるべき性質」について示されている．

☆☆ アプリケーションプログラムを共有ディレクトリに置いたとしても，ユーザがプログラムを使用する時点においてはプログラム（および冗長箇所情報）はユーザの PC のメモリにローディングされて実行されているので，ユーザとクラッカが同時にプログラムを使用すると冗長箇所情報の一貫性が崩れ，それ以降プログラムを使うことができなくなる．

表 1 (a) ミニマム・オンライン・アプリケーション方式と他の方式の比較（実装形態）
Table 1 (a) Comparison of implementation/configuration.

	スタンドアロン型アプリケーション	オンライン型アプリケーション	ミニマム・オンライン・アプリケーション		プログラムの虫食い実装方式[9]
			機能順序変動型アプリケーション方式	冗長命令変動型アプリケーション方式	
実装形態					
ユーザのPCに置かれるもの	• プログラムの全て	• サーバとの通信機能	• サーバとの通信機能 • 全ての機能モジュールのプログラム	• サーバとの通信機能 • 虫食い部以外のプログラム • 冗長箇所情報	• サーバとの通信機能 • 虫食い部以外のプログラム
サーバに置かれるもの	なし	• プログラムの全て • ユーザ認証機能	• ユーザ認証機能 • 各ユーザの機能テーブル	• ユーザ認証機能 • 各ユーザの虫食い部のプログラム(冗長命令が付加されている)	• ユーザ認証機能 • 虫食い部のプログラム
サーバへのアクセス	不要	要	要	要	要
実装容易性(スタンドアロン型，オンライン型の実装との比較)	—	—	**容易** スタンドアロン型のプログラムから機能テーブルのみをサーバに寄託すればよい	**やや困難** プログラムの本質であり，かつ，類推が不可能な処理をユーザ依存部として切り出す必要あり	**やや困難** プログラムの本質であり，かつ，類推が不可能な処理を虫食い部として切り出す必要あり

という効果が期待できる反面，

- 正規ユーザであっても 2 台目以降の PC で当該プログラムを使用できない，
- 一度クラッキングされると，被害者である正規ユーザもそれ以降，プログラムを使用できなくなる（これは，クラッカによるサービス妨害攻撃が容易であることにも通ずる）

という不便さが発生することになる．

一方，本方式のプログラムを共有ディレクトリやオンライン・ストレージに保存することにより，セキュリティのレベルを抑え，利便性の向上を図ることも可能であるが，4.2.2 項 (6) で述べたように，本論文ではプログラムを共用するような実装形態を採るべきではないと考える．特に誰もが頻繁に使用するアプリケーションプログラムに対しては，実際に本方式のプログラムが不正に共有されることは少ないのではないかと期待されるが，本来ならば，プログラムを共有ディレクトリやオンライン・ストレージにおくことを根本的に阻止する技術を採用すべきであろう．たとえば，以下のような方法が考えられるが，詳細については今後の課題としたい．

- PC 内に格納される冗長箇所情報（図 10 における冗長処理 BOX）を携帯デバイスに格納する．携帯デバイスを PC に挿入しないとプログラムを使用することができない．ある時点で携帯デバイスを複製したとしても，携帯デバイス内の冗長箇所情報がプログラム使用のたびに動的に変更されるので，その後に使い続けることができるのは本物または複製のいずれか 1 つの携帯デバイスのみである．
- アプリケーションプログラムの中でユーザが他人に参照されたくない情報を用いての処理を行う部分を虫食い部としてサーバに寄託する．これにより，そもそもプログラムを他人と共用するということが抑止される．

4.3 総　　括

ミニマム・オンライン・アプリケーション方式の具体的な実装方式として，4.1 節で機能順序変動型アプリケーション方式を，4.2 節で冗長命令変動型アプリケーション方式を説明した．この 2 つの具体的実装方式の性能を，従来のスタンドアロン型アプリケーションおよびオンライン型アプリケーションと比較することにより，本ミニマム・オンライン・アプリケーション方式によるアプリケーションプログラムの実装が現実にどの程度の有効性を有するのか検討してみる．

比較内容は，実装形態，セキュリティ強度，ユーザビリティおよびパフォーマンスに関する計 15 項目を選んだ．なお，冗長命令変動型アプリケーション方式の性能を見極めるため，特にプログラムの虫食い実装方式[9]に関しては陽に比較対象としてあげることにした．

実装形態に関する比較結果，セキュリティ強度に関する比較結果，ユーザビリティに関する比較結果，および，パフォーマンスに関する比較結果を，それぞれ**表 1** (a)〜1 (d) にまとめた．すべての比較結果から，機能順序変動型アプリケーション方式も冗長命令変動型アプリケーション方式も，基本的にはスタンドアロン型アプリケーションとオンライン型アプリケーションの両者の長所を最大限に，かつ，両者の短所を最小限に上手に集積した実装方式となっていることが分かる．こうして，本ミニマム・オンライン・アプリケー

表 1 (b) ミニマム・オンライン・アプリケーション方式と他の方式の比較（セキュリティ強度）
Table 1 (b) Comparison of security level.

	スタンドアロン型アプリケーション	オンライン型アプリケーション	ミニマム・オンライン・アプリケーション		プログラムの虫食い実装方式[9]
			機能順序変動型アプリケーション方式	冗長命令変動型アプリケーション方式	
セキュリティ強度					
プログラムの改造	**可能** チェッカ部のバイパスにより不正使用が可能	**不可能** サーバのセキュリティ管理が十分である限り，プログラムの改造による不正使用は不可能	**困難** 高度なクラッカーが十分な時間をかけると，プログラムを改造して不正使用することは不可能ではない	**不可能** サーバのセキュリティ管理は十分であり，かつ，虫食い部の処理を推定することができない限り，プログラムの改造による不正使用は不可能	**不可能** サーバのセキュリティ管理は十分であり，かつ，虫食い部の処理を推定することができない限り，プログラムの改造による不正使用は不可能
成りすまし攻撃	**可能** 正規ユーザが認証コードを他人に貸与しても実害はない	**可能** 正規ユーザが認証コードを他人に貸与しても実害はない	**不可能** 自分がプログラムを使えなくなるので，正規ユーザが認証コードを他人に貸与することはない(プログラムを共有ディレクトリなどに格納すれば，プログラムを共用することは可能であるが，プログラムを同時に使用できるのは一人)	**不可能** 自分がプログラムを使えなくなるので，正規ユーザが認証コードを他人に貸与することはない(プログラムを共有ディレクトリなどに格納すれば，プログラムを共用することは可能であるが，プログラムを同時に使用できるのは一人)	**可能** 正規ユーザが認証コードを他人に貸与しても実害はない
不正使用に気付く	**気付かない**	**気付かない**	**気付く** 他人に不正使用されると，自分がプログラムを使えなくなる(プログラムを共有ディレクトリなどに格納した場合には，他人に不正使用された後も正規ユーザがプログラムを使用することが可能)	**気付く** 他人に不正使用されると，自分がプログラムを使えなくなる(プログラムを共有ディレクトリなどに格納した場合には，他人に不正使用された後も正規ユーザがプログラムを使用することが可能)	**気付かない**

表 1 (c) ミニマム・オンライン・アプリケーション方式と他の方式の比較（ユーザビリティ）
Table 1 (c) Comparison of usability.

	スタンドアロン型アプリケーション	オンライン型アプリケーション	ミニマム・オンライン・アプリケーション		プログラムの虫食い実装方式[9]
			機能順序変動型アプリケーション方式	冗長命令変動型アプリケーション方式	
ユーザビリティ					
アプリケーション使用の際に要求される事項	**認証コードの提示** パスワードの押下や認証デバイスの挿入などにより，チェッカをパスしなければならない	**ネットワーク接続と認証** ネットワークに接続し，ユーザ認証をパスしなければならない	**ネットワーク接続と認証** ネットワークに接続し，ユーザ認証をパスしなければならない(各機能モジュールの一部を携帯デバイスに格納した場合は携帯デバイスの挿入が必要)	**ネットワーク接続と認証** ネットワークに接続し，ユーザ認証をパスしなければならない(冗長処理 BOX を携帯デバイスに格納した場合は携帯デバイスの挿入が必要)	**ネットワーク接続と認証** ネットワークに接続し，ユーザ認証をパスしなければならない
使用 PC の透過性	× 前もって別の PC にアプリケーションをインストールしておく必要がある☆ (PC のハードウェア情報等が認証コードになっている場合は，別の PC での使用は不可能) ☆スタンドアロン環境での使用を前提としているので，共有ディレクトリなどの使用は対象外とする	○ 基本的に PC は端末であるので，どの PC からもサーバにアクセスしてアプリケーションを使用することが可能	△ アプリケーションを共有ディレクトリなどに用意しておけば，どの PC からもアプリケーションを使用可能となる(セキュリティ強度とトレードオフの関係となる)	△ アプリケーションを共有ディレクトリなどに用意しておけば，どの PC からもアプリケーションを使用可能となる(セキュリティ強度とトレードオフの関係となる)	△ アプリケーションを共有ディレクトリに用意しておけば，どの PC からもアプリケーションを使用可能となる

表 1 (d) ミニマム・オンライン・アプリケーション方式と他の方式の比較（パフォーマンス）
Table 1 (d) Comparison of performance.

	スタンドアロン型アプリケーション	オンライン型アプリケーション	ミニマム・オンライン・アプリケーション		プログラムの虫食い実装方式[9]
			機能順序変動型アプリケーション方式	冗長命令変動型アプリケーション方式	
パフォーマンス					
実行速度	**ユーザの PC のパワーに依存** サーバへのアクセスはない	**ユーザ数に依存** 同時にプログラムを使用するユーザ数が増加するとサーバの負荷が高くなる	**スタンドアロン型とほぼ同値** サーバへアクセスする分だけ遅い	**スタンドアロン型とほぼ同値** 虫食い部をサーバで処理する分だけ遅い	**スタンドアロン型とほぼ同値** 虫食い部をサーバで処理する分だけ遅い
分散コンピューティングとの親和性	◎ ユーザの PC のパワーのみを使用	× サーバ・クライアント型	○ サーバ側の処理は少ない	○ サーバ側の処理は少ない(サーバ側にどれだけの処理を寄託したかに依存)	○ サーバ側の処理は少ない(サーバ側にどれだけの処理を寄託したかに依存)
サーバの負荷	−	× 全ての処理をサーバが行う	○ 機能テーブルの情報を返信するのみ	△ 虫食い部の処理を行う	△ 虫食い部の処理を行う
ネットワークトラフィック	−	△ サーバに依頼する全ての処理に対して，処理依頼の送信と処理結果の返信のトラフィックが発生	× 機能モジュールの使用ごとに，使用要求の送信と機能テーブルの情報の返信のトラフィックが発生(必要に応じて機能モジュールのコードも返信される)	○ 虫食い部の処理を行うごとに，虫食い部に渡す引数の送信と処理結果の返信のトラフィックが発生	○ 虫食い部の処理を行うごとに，虫食い部に渡す引数の送信と処理結果の返信のトラフィックが発生
プライバシ保護	◎ 完全にローカルでの処理	× データがサーバに送られて処理される	○ 使用した機能のみサーバ側で把握可能	△ ユーザ依存部に送られる引数のみサーバ側で把握可能	△ ユーザ依存部に送られる引数のみサーバ側で把握可能
メンテナンス(バージョンアップ等の管理)	× ユーザが自分でバージョンアップを行う	◎ サーバ上のプログラムを最新バージョンに保つだけでよい	○ サーバ上の機能モジュールを最新バージョンに保てば，順次，ユーザの PC に送られる	× ユーザが自分でバージョンアップを行う(虫食い部のメンテナンスのみは容易)	× ユーザが自分でバージョンアップを行う(虫食い部のメンテナンスのみは容易)

ション方式の有効性が実証された．

5. ま と め

本論文では，アプリケーションプログラムの不正使用防止を目的としてミニマム・オンライン・アプリケーション方式を提案した．本方式では,「プログラムの本質であり，かつ，各ユーザの使用履歴に依存する必要最小限の情報または機能」のみをサーバに寄託する．本方式のプログラムはその本質部分がサーバ上にあるため，ユーザ認証（サーバへのアクセス）をバイパスするとプログラムが使用不能となる．また，ユーザがプログラムを使用するたびにサーバ側の当該ユーザ情報が変化するため，他人が自分の代わりにプログラムを使用するとその後自分がプログラムを使うことができなくなる．この結果，

- プログラムの本質部分がサーバ上にあるため，プログラムのリバースエンジニアリングは不可能である，
- 正規ユーザがパスワードを漏らすと本人にデメリットが生じるため，正規ユーザからの安易なパスワード漏洩は起こらない，
- 万一プログラムの不正使用が行われた場合に，正規ユーザがその事実に気付く

などの特長を有する理想的なアプリケーションプログラムの実装が可能となる．

2 種類の具体的なシステム構成も示し，その攻撃耐性や性能について評価した．これらのシステムは必要最小限の部分のみがサーバに存在しており，オンライン型アプリケーションの性質を備えつつ，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，シミュレーションなどにより本方式のスケーラビリティ（ユーザ数に対するアプリケーションの動作速度やネットワークトラフィック）など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) MicroSoft.NET 構想. http://www.microsoft.com/net/
3) 原田ほか：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, pp.97–141 (2001).
4) MicroSoft Office アクティベーション. http://www.microsoft.com/japan/piracy/mpa/

5) MicroSoft, MSN Messenger. http://messenger.msn.co.jp
6) Mirabilis Inc, ICQ. http://www.mirabilis.com/
7) AOL/Netscape, AIM. http://www.netscape.com/aim/
8) 森 亮一：ソフトウェアサービスシステムについて，電子通信学会誌，Vol.67, No.4 (1984).
9) 門田ほか：コピー防止を目的とするプログラムの虫食い実装方式，情報処理学会 夏のプログラミングシンポジウム報告集，pp.47–54 (1997).
10) STORAGE NETWORKING WORLD/TOKYO 2001 Fall. http://www.idg.co.jp/expo/snw/
11) 中村ほか：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, pp.41–48 (2000).

(平成 14 年 6 月 6 日受付)
(平成 15 年 1 月 7 日採録)

西垣 正勝（正会員）

平成 2 年静岡大学工学部光電機械工学科卒業．平成 4 年同大学大学院修士課程修了．平成 7 年同大学院博士課程修了．日本学術振興会特別研究員（PD）を経て，平成 8 年静岡大学情報学部助手，平成 11 年同講師，平成 13 年同助教授，現在に至る．博士（工学）．回路シミュレーション，ニューラルネットワーク，情報セキュリティ等に関する研究に従事．

中村 直己

平成 12 年静岡大学情報学部情報科学科卒業．平成 14 年同大学大学院情報学研究科博士前期課程修了．現在，株式会社ネットマークス ネットワークセキュリティ事業部に勤務．在学中，情報セキュリティに関する研究に従事．

曽我 正和（正会員）

昭和 33 年京都大学工学部電子工学科卒業．昭和 35 年京都大学大学院修士課程電子工学専攻修了．昭和 35 年～平成 8 年三菱電機計算機製作所，情報電子研究所，本社開発本部．平成 8 年静岡大学情報学部教授，平成 11 年岩手県立大学ソフトウェア情報学部教授，現在に至る．博士（工学）．汎用計算機，制御用計算機，制御用システムの開発に従事．フォールトトレラントシステム，セキュリティシステムに関する研究に従事．

田窪 昭夫（正会員）

昭和 17 年生．昭和 41 年早稲田大学理工学部電気工学科卒業．昭和 43 年同大学大学院理工学研究科修士課程修了．同年三菱電機株式会社入社，平成 10 年静岡大学大学院博士後期課程修了．平成 14 年東京電機大学情報環境学部教授．博士（工学）．モバイルコンピューティング，ネットワーク，セキュリティ等に興味を持つ．電気学会，IEEE，ACM 各会員．

中村 逸一（正会員）

昭和 60 年茨城大学工学部卒業．昭和 62 年同大学大学院修了．同年日本電信電話株式会社入社．LAN システムの研究に従事．平成 8 年より（株）NTT データにてセキュリティ技術の研究・開発に従事．現在，同社セキュリティ事業部部長．