第 13 部

利用者認証

第 1 章

はじめに

現在コンピュータは社会の中で大きな位置を占めている。企業や教育機関はコンピュータを積極的に導入する傾向にある。それに加え、近年のコンピュータの低価格化とローカルエリアネットワークの普及が、その傾向をますます強いものにしている。その結果、LAN を構築し、さらにそれを WIDE Internet に代表されるインターネットに接続するサイトも年々急激に増加している。このように多くの人がさまざまな計算機と OS を用いて、インターネットとの接続を確立し、インターネットで提供されているさまざまなネットワークサービスを利用するようになってきている。

現在のインターネット環境では何十万という計算機が接続されており、利用者の数はこれをはるか超える。インターネット環境に接続されている計算機全てが厳重に管理されているわけではなく、システムのアカウントさえ取得できれば、比較的誰でもインターネットにアクセスし、インターネットのサービスを受けることができるようなオープンな環境である場合が多い。これは、非常に多くの利用者が比較的簡単にインターネットを利用することを意味する。このため、特定の人々だけにアクセスを許しているシステムやサーバに対して不正アクセスを試みる利用者が後を絶たないのも事実であり、インターネットにおける一つの問題となっている。

また、インターネット環境で提供される代表的なサービスの一つであるリモートログインでは、利用者がログインを行う際のパスワードは、この文字コードがそのままの形でネットワークに伝送される。Ethernet に代表されるバス型のネットワークでは盗聴が容易に行うことができる。また、インターネットを介してリモートログインを利用した場合には、基本的に通信は第三者のネットワークを経由して送られることになる。このような場合に、通信路中の中継ホストにおいて盗聴が行なわれる危険性が、常に発生している。このような盗聴の問題は、現在のインターネット環境で提供されるサービスでは対処してあることが少なく、特にパスワードの盗聴は深刻な問題を招く。このため、現在のインターネットサービスに対して認証機構を強化したシステムや、通信データの暗号化機能を提供するシステムが研究、開発されてきている。

WIDE Project Security WG では、これまで大阪大学のグループを中心としてネットワークセキュリティ、特に認証 (authentication) に関して研究が行われ、SPLICE/AS と呼ばれる認証システムが開発されてきた [134]。SPLICE/AS は RSA 暗号を用い、Needham 等によって提案された認証プロトコル [135] に基づいたプロトコルを用い、利用者の認証を行うシステムであったが、暗号処理性能の悪さ、管理構造の問題等いくつかの問題が

報告されていた [136]。1992 年度の Security WG の活動では、この SPLICE/AS を基にコミュニティ(community) の概念を導入し、それらの問題を解決する新たな認証システムである SPLICE/AS-II の設計・開発を行った。SPLICE/AS での問題は、RSA 暗号処理の高速化、二重化された認証サーバ、プロトコルの書換えによって解決し、その応用として telnet の SPLICE/AS-II 対応化を行った。

本論文の第 2 章では、現在のネットワーク環境におけるセキュリティ問題、つまり他の計算機になりすますことやネットワークを流れるデータの盗聴・改ざん等の問題について述べる。第 3 章では利用者を認証するためにはどのような方法があるか、さらに認証に用いることが可能な暗号技術について述べる。第 4 章では SPLICE/AS-II が導入したコミュニティに基づいた認証モデル、SPLILCE/AS-II のプロトコル、開発したシステムが利用する設定ファイルについて述べる。第 5 章では SPLICE/AS-II の応用として telnet のソースコードに対して SPLICE/AS-II の機能を適用する (以後 SPLICE 化という) ために行った変更点について述べる。6 章ではシステムの評価、7 章が本論文のまとめとなっている。

第 2 章

ネットワーク環境におけるセキュリティ問題

本章では、インターネット環境におけるネットワークセキュリティを考えた場合に、脅威 (threat) となる行為について述べる。

1. **情報の盗聴**
   インターネット環境で行なわれる通信では、第三者の組織内のネットワーク機器を複数経由して行なわれることが多い。このため、盗聴を完全に防ぐことは容易ではない。また、Ethernet のようなバス型の LAN では、パーソナルコンピュータレベルのネットワークモニタ機器を Ethernet ケーブルに直接接続し、通信内容を盗聴することが比較的容易である。さらに、リモートログインサービスを提供する telnet コマンド、rlogin コマンドはパスワード入力時の文字を暗号化せずに送信しており、パスワードの盗聴は深刻な問題を含んでいる。

2. **他ホストへのなりすまし**
   通信されるデータに含まれる発信元情報を偽ることにより、他ホストになりすまし、情報を不正に得ることが可能である。TCP/IP では IP データグラムヘッダに含まれる発信元アドレスフィールドの IP アドレスをもとに合いてホストを識別している。このため、簡単なプログラムを用いてこのフィールドを適当に変更すれば、その IP アドレスを持つホストになりすますことが可能である。

   さらに計算機の低価格化も他のホストへのなりすましを容易にしている。現在、Ethernet インタフェースを具備した計算機は非常に安価になっている。そのような計算機が自由に扱えるのならば、その計算機のアドレスを適当な値に設定し、ネットワークに接続することによって他の計算機になりすますことが可能である。ただしこの時、なりすました相手の計算機が同じ Ethernet 上で稼働中であればアドレスが重複することになるが相手の計算機がネットワークに接続されていない、もしくは稼働を停止していることがわかっているならば、なりすましは簡単にできてしまう。

3. **ケーブルを流れるデータの改ざん**
   ネットワークを流れるデータを盗聴するだけでなく、そのデータの一部分を改ざんして戻すことも可能である。こうすることにより、不正なアクセス権を得られる可能性がある。

4. **他ホスト、ネットワークへの不要メッセージ発信による運用妨害**
   これは一般ユーザでも簡単なプログラムを書くことによって可能となる。例えば、TCP/IP におけるネットワークサービスの一つである echo サービスを要求するパケットを、特定のホストに対して無限に送信するプログラムを書いたとする。このプログラムを実行すると、そのターゲットとなるホストは echo サービスの処理に追われ、コンソールからの入力さえ正常にできなくなってしまう。この種の攻撃は発見することはできても防ぐことは難しい。もし、異なるネットワークからの攻撃ならばゲートウェイでパケットフィルタリング (packet filtering) を行なえば、多くの場合回避することができる。しかし、攻撃者がパケットのソースアドレスを書き換えて、同じネットワーク内から発信されたように見せかけた攻撃を行った場合はパケットフィルタリングが有効にはならない。

5. **誤ったネットワーク制御情報の発信**
   現在使われているルーティング情報を交換するプロトコルとして RIP (Routing Information Protocol) が挙げられる。、RIP では受け取ったメッセージが通信中に改ざんされていないか、あるいは不正なホストから送り込まれたメッセージではないかといった点についてチェックを行わない。このことにより、偽のルーティング情報が流されると簡単にルーティングテーブルがおかしくなってしまう。

6. **トラヒック解析**
   これは学術研究ネットワークにおいてはさしあたっての脅威とはならないが、トラヒックを解析することにより攻撃者にとって有効な情報が得られる場合がある。パケットの発信元と送信先の統計を取ることにより、各種ネットワークサービスのサーバを推測することができる。

いくつかの問題について述べてきたが、(2)、(3)、(5)、(6) の問題は通信路での暗号化機能を提供することにより一部阻止できる。

(4) については攻撃の発見はできるが完全に防ぐことは不可能である。

(1)、(5) の問題については利用者認証機構を提供することにより解決できるようになる。この問題に対しバークレー r コマンド (rsh, rlogin 等) は、trusted host という信頼できるホストをファイルに記述しておき、そのホストからのサービス要求ならばそれに答えるという認証方法を用いている。しかし、そのホストはいつも信頼できるとは限らない。また、そのホストのユーザ全てが信頼できるとは限らないという理由でこの方法は十分とは言いがたい。

第 3 章

利用者認証技術と暗号

本章では、計算機環境での利用者認証技術と、それに用いることが可能な暗号技術について述べる。

3.1 利用者認証技術

計算機環境での利用者認証技術としては、これまでさまざまな方法が考案されてきており、代表的には以下の方法がある。

1. 鍵やカードなど、本人だけが持っている物理的なもので認証する方法。
2. パスワード、暗証番号など、本人だけが知っている情報で認証する方法。
3. 指紋、声紋、サイン、掌紋など、本人の身体的特徴を何らかの形で入力させ、それを元に認証する方法。

これらの方法のうち、(1) については様々な形で計算機環境に採り入れられてきている。例えば最近の PC では、電源の投入を通常のスイッチとロータリーキーによってコントロールするものが多くなってきている。また、UNIX ワークステーションに対しても、カードによる認証を行なうハードウェアが利用可能になっている。また (3) についてはあまり一般的ではないが、掌紋や眼底血管パターンなどの情報を利用したシステムが開発されており、一部では利用されている。

しかしながら、(1)、(3) を一般的な計算機利用環境に導入することを考えると、これらには特別なハードウェアが必要である。このため、全てのインターネット環境で用いることは現実的に不可能である。また、ネットワークサービスについてこれらの技術を利用することは、現実的ではない。したがって、インターネットサービスを考えた場合には、このようなハードウェアの使用を仮定するのは無理があり、さらに拡張性が悪くなるので採用しにくい。

一方、(2) については、現在多くの計算機システムにおいて採用されている。(2) の方法はシステムに登録した情報が他人に漏れてしまったときに、確かに本人かどうかという認証がうまく行なえくなってしまう。しかしながら、登録された情報の保護を暗号技術を使って行なうことによりシステム側から情報が漏れることをある程度であるが防ぐ

ことができる。ただし、このときの暗号は一対一の暗号でなければならず、また、容易に逆変換できてはならない。さらに、ネットワーク環境での認証についても、Needham らが提案した方法 [135] を適用することにより、原理的にはネットワーク環境においても (2) の方法を実装することが可能である。

3.2 暗号

ネットワークを流れるデータを盗聴、改ざんから守るためにはデータの暗号化が必要不可欠である。ここでは代表的な暗号方式として DES 暗号、RSA 暗号、ブートストラップ暗号について述べる。

3.2.1 DES 暗号

米国 NBS(National Bureau of Standards) が連邦政府関係の一般データに適用されるものとして 1997 年に発表したものが DES(Data Encryption Standard) であり [137]、そのアルゴリズムは LUCIFER を発展させたものである。

DES は 56bit の鍵を用いて 64bit の平文を置換と排他的論理和 (XOR) よって暗号文にする暗号化方式である。また、ソフトウェアによってもハードウェアによっても実現されており、たいへん高速に暗号化、復号化が可能で、des コマンドとして UNIX リリーステープに含まれている (ただし米国のみ) ほど容易に使える暗号である。

米国において標準となった DES だが、Diffe、Hellman その他専門家により脆弱性が懸念されている [138][139]。その理由は、56bit では鍵の長さが十分ではないかも知れない。そして、S ボックスには何らかの落し穴 (pitfall) がありうると言われている [140]。

64bit 以上の長さを持つメッセージの暗号化には DES を繰り返し適用する必要がある。どのように DES を適用するかにより ECB(Electoronic Code Book) モード、CBC(Cipher Block Chaining) モード、CFB(Cipher FeedBack) モード、OFB(Output FeedBack) モードと呼ばれる適用方法がある。この中で CBC モードと CFB モードはあらゆる不法侵入 (暗号文の探索、再生、挿入、削除等) に対して強いと言われている。

図 3.1に DES の CBC モード、図 3.2に DES の CFB モードの動作を示す。

3.2.2 RSA 暗号

RSA(Rivest–Shamir–Adleman) 暗号 [141] は平文 M を暗号化鍵 (e, n) を用いて暗号文 Cをつくり、復号化鍵 (d, n) を用いて元の平文を得るアルゴリズムである。

$$
\begin{aligned}
C &= M^e \bmod n \\
M &= C^d \bmod n
\end{aligned}
$$

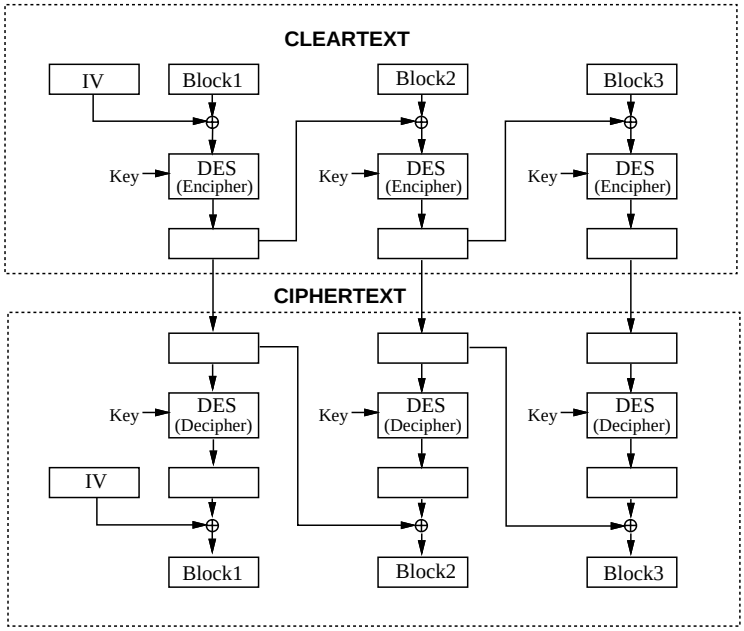

図 3.1: CBC モードの DES の適用方法

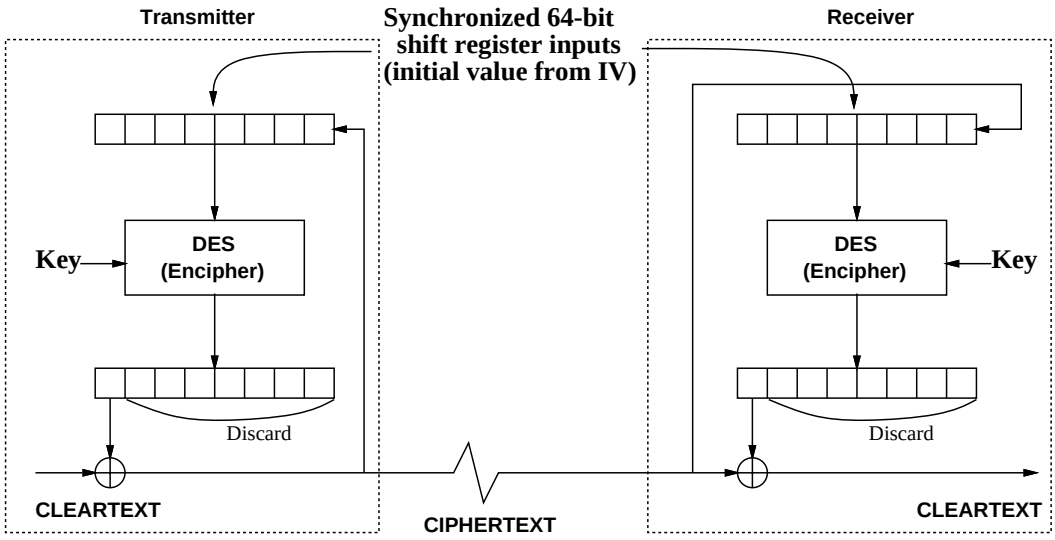

図 3.2: CFB モードの DES の適用方法

RSA 暗号では $0 \leq M \leq n-1$ を満たす M が与えられたとき、

$$\begin{aligned}(M^e \bmod n)^d mod n &= M \\ (M^d \bmod n)^e mod n &= M\end{aligned}$$

を満たす。この対称性により RSA 暗号を公開鍵システムとして秘密を守ること (confidentiality)、真正性 (authentication) を保障するために適用することができる。

ここで n は十分大きな素数 p と q の積

$$n = pq$$

であり、RSA 暗号の安全性は n を p と q に因数分解する計算量によっ ている。現在知られている最も高速の因数分解アルゴリズムを使った場合の計算所要ステップ数は

$$T = \exp(\sqrt{\ln(n) ln(ln(n))})$$

で与えられる (図 3.3)。ここで仮に p と q に各々100 桁の素数を用いたとすると n は 200 桁となり 1 ステップに 1μ秒かかったとすれば n の因数分解に数十億年かかることになる。

また、p と q の選び方にも RSA 暗号の安全性は左右され、もし p と q が非常に近い数であれば

$$p \simeq \sqrt{n}, q \simeq \sqrt{n}$$

により p と q のおおよその検討をつけることができる。

因数分解に対する防御手段として p と q の選定について以下の条件を満たせばよいと言われている [142]。

1. p と q の桁数はほんの数桁だけ異なるようにする
2. $p-1$、$q-1$ も大きな素数を因数に持つようにする
3. $\gcd(p-1, q-1)$ は小さくする

暗号化で注意しなければならないのが $M < n$ が成り立っていなければならないことで、もし、平文が十分長ければ $M > n$ となることがある。この時は平文を $M < n$ の条件が成り立つようなところで区切って ($M = M_1 M_2 \cdots M_n$) 各々を暗号化すればよい。

3.2.3 ブートストラップ暗号

RSA 暗号方式の問題は暗号化、復号化の処理が DES 暗号に比べ非常に遅いことである。長いメッセージを暗号化するには RSA 暗号を複数回適用する必要があるが、実際の通信ではそのオーバーヘッドが問題となってしまう。この問題を解決するために Zimmermann によってブートストラップ暗号が提案されている [143]。

ブートストラップ暗号では単一鍵暗号の鍵ブロック ($Key_1, Key_2, Key_3, \cdots, Key_n$)を用いてメッセージを暗号化し、次にその鍵のブロック (ブートストラップブロック) を RSA で

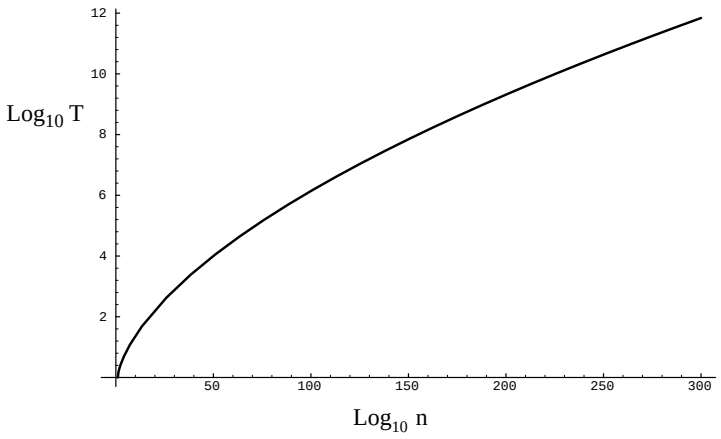

図 3.3: n 桁の数字の因数分解の計算量

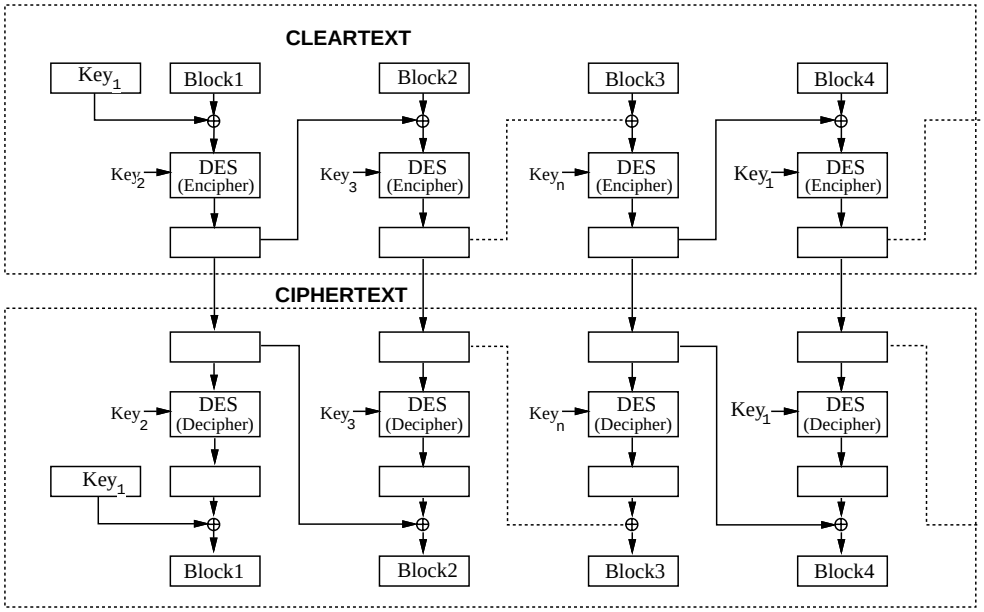

図 3.4: ブートストラップ暗号のアルゴリズム

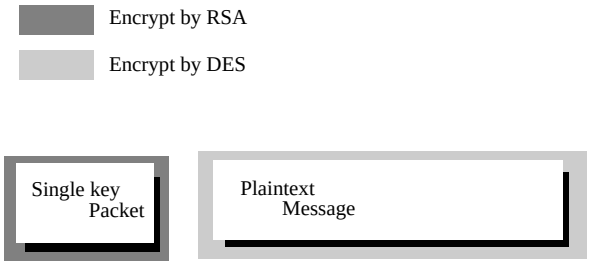

図 3.5: ブートストラップ暗号されたメッセージ

暗号化する。復号化では、まず RSA でブートストラップブロックを復号化し、そこで得られた鍵を用いてそれに続く暗号文を復号化する。これにより、RSA 暗号の多用を減らし、かつ RSA 暗号の特性を利用しながら高速に暗号化、復号化が可能となる。Zimmermann は単一鍵暗号の種類は限定しなかったが、SPLICE/AS-II では DES の CBC モードの変形を使っている。

第 4 章

認証システム

2章で述べた問題の解決策としていくつかの認証システムが開発、利用されている。ここでは、それらのうちから Kerberos、Sun SecureRPC、SPLICE/AS について述べる。

4.1 Kerberos

Kerberos[144] は M.I.T. で Athena プロジェクトの一環として開発された認証システムである。Kerberos では、パスワードのような他人に知られては困るようなデータをネットワークに流す場合、すべてを DES で暗号化する。

Kerberos は既存のシステムに利用者が付加するシステムであり、既存のネットワークプロトコルの必要最低限の変更で使用できる。現在 NFS、リモートログイン、電子メール等が Kerberos 対応化 (Kerberized) されている。

Kerberos システムはセキュリティ問題に対する解決策の 1 つであるが以下のような問題がある [145]。

- Kerberos を使用するためにはすべてのネットワークサービスを利用するプログラムを作り直さなければならない。このためにはたいへんな労力が必要となる。
- タイムシェアリングシステムではうまく動かない。Kerberos では全ての計算機利用者が読み出す可能であるディレクトリに機密情報を保存する。もし他人がこの情報を複製し使用すれば不正なアクセス権を得ることができる。
- システムソフトウェアの改ざんには無防備である。プログラムにトロイの木馬等が仕込まれることには対応していない。
- 認証サーバに侵入されたときシステムの安全性が全くなくなってしまう。

4.2 Sun SecureRPC

Sun SecureRPC(Remote Procedure Call) は以前の SunRPC に認証機能を付加した RPC であり、SunOS4.0 以降からシステムに付随している。DES を用いる点で Kerberos と同じであるが、SecureRPC には以下の特徴がある。

- 利用者の秘密鍵は公開鍵方式により暗号化され NIS で管理されている。これにより安全な認証サーバを仮定する必要がない。
- 付加システムではないので、新たにインストールする必要がない。

現在、SecureRPC は NIS、NFS で利用することができる。そのための設定は非常に簡単であり、便利なシステムであるが、鍵の管理が十分に安全でないという問題がある [145]。

4.3 SPLICE/AS

SPLICE/AS は WIDE Project Security WG の大阪大学のグループで開発を行なってきた認証システムである。 SPLICE/AS は Needham 等によって示された公開鍵暗号を用いた利用者認証プロトコル [135] を基本にシステムが構成されている。さらに管理構造として Internet における DNS(Domain Name System)[146] での名前管理構造に対応した階層的なドメイン構造を導入し、ドメイン間の鍵の配送を階層的に行なうことで広域ネットワークでの分散管理・運用を実現したシステムであるが、以下のような問題がある [136]。

- 暗号処理性能が悪い。暗号方式に RSA を用いそれをソフトウェアで実現しているので認証サーバの処理が遅い。
- 管理構造の問題。あるドメインに属する計算機や利用者の数が膨大になる場合があり、DNS に対応したドメイン構成では認証サーバに対してトラヒックが集中してしまう場合がある。
- 認証サーバが多重化されていない。
- 認証サーバによって提供される情報が十分ではない。

第 5 章

SPLICE/AS-II の設計

前章で述べたように SPLICE/AS には実際にインターネット環境に適用する場合に障害となる、幾つかの問題が解決されていなかった。そこで、本研究ではこれらの問題を解決するために、コミュニティという新しい概念を導入し、新たな認証システムである SPLICE/AS-II の設計と開発を行った。コミュニティを考えることにより、他の認証システムにはない柔軟なアクセス制御を実現することができる。この章ではシステムのモデル、プロトコル等について述べる。

5.1 SPLICE/AS-II でのモデル

SPLICE/AS-II ではコミュニティ(community) という概念を導入した。通常、人間社会では人はグループを作り、それらのグループに属している。そして、ある特定のグループに属しているというその人の「立場」によりそのグループ内でのサービスが受けられるようになっている。SPLICE/AS-II のコミュニティとは、そのグループに対応したものであり、利用者とその利用者にサービスを提供するサーバから構成される。利用者は複数のグループに属することができるのと同様に、複数のコミュニティに属すことができる。また、利用者が複数のコミュニティに属することができるのと同様に、サーバも複数のコミュニティに属することができ、このサーバは複数のコミュニティに対してサービスを提供する。

現在のアクセス制御は、利用者のログイン名、グループ名、どのホストからのアクセスかといった情報で行われるが、このホスト情報つまり、trusted host の危険性は 2 章で述べた。このため SPLICE/AS-II ではコミュニティという概念を導入し、その新たな情報に基づくアクセス制御を行う。この新しいアクセス制御は、利用者のログイン名、利用者の属するコミュニティ名、コミュニティの認証サーバが利用者を認証したことを表す証明となるもの (認証子 authenticator) を用いて認証・アクセス制御を行い、利用者が「コミュニティに属するその利用者であること」であることでアクセスが許可されたり、拒否されたりする。

コミュニティは 1 つまたは複数のリージョン (region) に分かれる。コミュニティをリージョンに分ける理由は以下の通りである。

- 認証サーバの管理の分散

- 認証サービスの分散
- データベースに侵入を受けたときの被害の軽減
- 認証サーバ、利用者間のネットワーク切断による被害の軽減

利用者、サーバはコミュニティ内では 1 つのリージョンにのみ属する。また、リージョンは DNS ドメインとは全く関係なく分けることができる。

各リージョンには 1 つの認証サーバがあり、認証サービスを提供する。1 つのコミュニティ内に複数の認証サーバが存在するとき、それらはそれぞれ自分のリージョンに属する利用者を認証し、他の認証サーバはその結果を信用する。このような信頼関係にある認証サーバをリンクが張られているという。

利用者がサーバにサービスを要求するときに、サーバと通信を行なうプログラムをクライアントと以後呼ぶ。クライアントは、それを起動した利用者が属するコミュニティに属すると考える。

コミュニティ内の利用者を識別するために、各利用者になんらかの情報を持たせる必要がある。SPLICE/AS-II では各利用者に 32 文字の名前を与え、これをログイン名とする。この名前を構成する要素となる文字はヌル文字以外の全ての文字の使用を許している。また、この名前は一つのデータベース内で一意、つまりリージョン内で一意であるとする。

5.2 サーバの二重化

クライアントとサーバから見た場合、SPLICE/AS-II 認証サーバは「第三者」に見えるということから、このような認証システムのことを Trusted Third-party Authentication System と呼ぶ。このようなシステムでは認証サーバの安全性がシステムの安全性に直接つながる。そのために認証サーバが稼働する計算機を物理的に安全なところに設置するとともに、不必要なサービスをすべて停止しなければならない。これ以外に、認証サーバのサービスの信頼性を高めるためには、計算機がダウンしたときに備えてサーバの多重化が必要である。

SPLICE/AS-II では認証サーバの二重化を行ない、通常認証を行なう認証サーバをマスタサーバ、マスタサーバがダウンしたときに認証サービスを行なう認証サーバをスレーブサーバという。マスタサーバとスレーブサーバが管理するデータベースは同一性が保証される。そのためマスタサーバが持つデータベースに変更があると、マスタサーバはスレーブサーバにそれを知らせる。しかも、管理用ツールを用いてデータベースを変更できるのはマスタサーバに対してのみとし、スレーブサーバはクライアント、サーバに対してデータを提供するだけで、スレーブサーバのデータベースを変更できるのはマスタサーバからの要求のみとする。クライアント、サーバプロセスは、まずマスタサーバにコネクションを張ろうとし、それに失敗するとスレーブサーバにコネクションを張る。それにも失敗すると処理を断念する。

5.3　SPLICE/AS-II の構成

SPLICE/AS-II は以下の要素から構成される。

- 認証サーバ
  認証を必要とするプロセスに対して認証情報を提供する。

- データベース
  先に述べたとおり、データベースはネットワーク的にも物理的にも安全な環境で管理されなければならない。SPLICE/AS-II が用いるデータベースでは以下の情報を管理する。

  – リージョン内の利用者の秘密鍵、公開鍵、パスワード
  – リージョン内のサーバの秘密鍵、公開鍵、パスワード
  – 信頼できる他の認証サーバの管理者のパスワード

- ライブラリ群
  認証サービスを利用するプログラムのためのライブラリ

- 管理用ツール
  利用者登録、パスワード変更のためのプログラム

5.4　プロトコル

ここでは SPLICE/AS-II が用いるプロトコルについて述べる。プロトコルの表記については表 5.1を用いる。

5.4.1　前提

SPLICE/AS-II では以下の点をプロトコルの設計の前提としている。

1. クライアントは自分のコミュニティの認証サーバのあるネットワーク内でのロケーションを知っている

2. サーバは自分のコミュニティの認証サーバのあるネットワーク内でのロケーションを知っている

3. 認証サーバが用いるデータベースは安全に管理されている

記号	意味
AS	認証サーバ
S	サーバ
C	クライアント
L	クライアントのログイン名
Com	コミュニティ名
PW	AS に登録されている暗号化されたパスワード
PW'	ユーザが入力する暗号化されたパスワード
U	ユーザ
timestamp	認証子を発行したタイムスタンプ
life	認証子の有効期間
SK_X	X の秘密鍵
PK_X	X の公開鍵
SSK	セッション鍵。十分大きな整数 (乱数)
msg	セッション鍵よりも大きな整数 (乱数)
$\{M\}^K$	K を鍵としてメッセージ M の RSA 暗号化
$[M]^K$	K を鍵としてメッセージ M の DES 暗号化

表 5.1: プロトコルの表記に用いる略称

5.4.2　利用者の認証

利用者 (ユーザ) は SPLICE/AS-II のセキュリティ機能を利用するにあたり利用者自身の公開鍵と秘密鍵を認証サーバから手に入れる必要がある。しかし、認証サーバそのものが偽物であるかもしれず、偽物の鍵を入手すると以後の通信を盗聴・改ざんされる可能性がある。よって認証サーバが利用者を認証するとともに、利用者も認証サーバを認証しなければならない。これには以下のプロトコルを用いる。

1. 利用者はコミュニティ名、ログイン名、パスワードを入力する。
2. ログインプログラムはコミュニティ名から AS を知る。
3. U→AS:
$$U, AS, L \tag{1-1}$$
このメッセージには利用者のログイン名しか含んでいないので暗号化する必要はない。
4. 認証サーバは受け取ったメッセージのログイン名を用いて利用者のパスワードをデータベースから検索し、そのパスワードを鍵として認証サーバの公開鍵を DES 暗号化する。利用者の名前がデータベースになければこのメッセージは偽物であるとする。
5. AS→U:
$$AS, U, L, PK_{AS_1}, \{AS, PK_{AS_2}, [PK_{AS_3}]^{PW}\}^{SK_{AS}} \tag{1-2}$$
6. ログインプログラムは PK_{AS_1}で $\{\cdots\}$ を復号し、PK_{AS_1}と PK_{AS_2}を比較する。つぎにユーザが入力したパスワード PW'で $[PK_{AS_3}]^{PW}$を復号する。ここで、PWは本物の認証サーバだけが知っているので、$[PK_{AS_3}]^{PW}$を作れるのは本物の認証サーバだけである。また、これを復号できるのは本物のユーザだけである。この復号で得られた PK_{AS_3}と前の 2 つの PKを比較する。これでユーザは認証サーバを認証することができ、正しい公開鍵を得ることができた。
7. ログインプログラムはセッション鍵 SSKと msgを生成する。
8. U→AS:
$$U, AS, L, \{U, L, Com, SSK, msg, [msg]^{PW'}\}^{PK_{AS}} \tag{1-3}$$
9. 認証サーバはメッセージを SK_{AS}で復号しコミュニティ名のチェックを行い、msgを得る。次にユーザのパスワードと msgから $[msg]^{PW}$を生成し、$[msg]^{PW'}$と比較する。これらが同一になるためにはユーザが正しいパスワードを入力する必要があるので認証サーバはユーザを認証することができる。
10. 利用者の認証ができれば認証サーバは以下の認証子 (authenticator) を作成する。
$$Authenticator = \{L, Com, PK_U, timestamp, life\}^{SK_{AS}} \tag{1-4}$$

11. AS→U:

$$AS, U, L, [AS, PK_U, SK_U, Authenticator]^{SSK}$$

12. このメッセージを復号することにより利用者は自分の本物の公開鍵と秘密鍵を得ることができる。ここでユーザのログインセッションを終了する。利用者はこの $Authenticator$をサーバにサービスを要求するときまで保持しておく。

ここで用いている SSKは乱数であり SSKを新たに必要とする度に生成し、replay attack を発見するために用いている。メッセージ (1–1) と (1–2) と (1–3) は replay attack であったとしても実害はない。攻撃者が (1–3) を盗聴したとしても復号するためには認証サーバの秘密鍵が必要となり、メッセージ内の SSKを知ることはできない。(1–4) が replay attack によるメッセージであるとするとクライアントがこのメッセージを正しく復号するためには SSKで暗号する必要がある。SSKは毎回新しく生成するので (1–4) のメッセージは正しく復号できない。このようにして replay attack を発見する。以後同様の目的でセッション鍵を用いる。

これで利用者は自分の鍵を得たわけであるが、このプロトコルで重要な点は利用者のパスワードが一度もネットワークに流れていないことである。SPLICE/AS-II では利用者のパスワードが他人に知られてしまえばその人になりすますことができる。このプロトコルを用いれば盗聴によってパスワードが他人に知られることはあり得ない。データベースが他人に覗かれることがないと前提しているので、利用者が自分のパスワードを他人の目に触れるところに書いておかない限り他人が自分になりすますことはできない。また、パスワードは容易に推測できるものであってはならない。

5.4.3 サーバの起動

サーバを起動できるのは、通常そのサーバの管理者だけである。サーバの起動では、そのサーバの管理者のみが知っているパスワードを入力し、正しく入力したときのみ起動するようにする。

サーバも利用者と同じく自分自身の公開鍵と秘密鍵、認証サーバの公開鍵を入手する必要がある。そのためのプロトコルは、基本的には利用者の認証のものと同じだが、サーバには認証子が不必要な点が異なる。また、サーバのログイン名、サーバが属するコミュニティ名はサーバプログラムの中に書かれているので入力する必要はない。

1. サーバの管理者はパスワードを入力する。
2. コミュニティ名から AS を知る。
3. S→AS:

$$S, AS \qquad (2-1)$$

4. AS→S:

$$AS, S, PK_{AS_1}, \{AS, PK_{AS_2}, [PK_{AS_3}]^{PW}\}^{SK_{AS}} \qquad (2-2)$$

5. S→AS:

$$S, AS, \{S, Com, SSK, msg, [msg]^{PW'}\}^{PK_{AS}} \quad (2-3)$$

6. AS→S:

$$AS, S, [AS, PK_S, SK_S]^{SSK} \quad (2-4)$$

7. このメッセージを復号することによりサーバは自分の本物の公開鍵と秘密鍵を得ることができる。

5.4.4　サーバのリンク

1 つのコミュニティには複数の認証サーバを置くことが可能であるが、他の認証サーバとリンクを張るときにその認証サーバがお互いに相手の公開鍵を得る必要がある。ここでは認証サーバ AS_1の管理者が AS_2にリンクを張るとする。このとき認証サーバどうしがお互いを認証しなければならないが、そのプロトコルは利用者の認証のものとほぼ同じである。

1. AS_1の管理者は AS_2のホスト名、パスワードを入力する。

2. $\mathrm{AS}_1 \rightarrow \mathrm{AS}_2$:

$$AS_1, AS_2, L \quad (3-1)$$

ここで L は AS_1の管理者であることを識別する名前である。

3. $\mathrm{AS}_2 \rightarrow \mathrm{AS}_1$:

$$AS_2, AS_1, PK_{AS_{2_1}}, \{AS_2, PK_{AS_{2_2}}, [PK_{AS_{2_3}}]^{PW}\}^{SK_{AS_2}} \quad (3-2)$$

4. $\mathrm{AS}_1 \rightarrow \mathrm{AS}_2$:

$$AS_1, AS_2, L, \{AS_1, L, Com, SSK, PK_{AS_1}, msg, [msg]^{PW'}\}^{PK_{AS_2}} \quad (3-3)$$

5. $\mathrm{AS}_2 \rightarrow \mathrm{AS}_1$:

$$AS_2, AS_1, [AS_2]^{SSK} \quad (3-4)$$

6. このメッセージを復号することにより認証サーバはお互いの本物の公開鍵を得ることができる。

5.4.5 クライアント・サーバ間の認証

クライアントがサーバに対して何らかのサービスを要求するときはクライアントがサーバを、サーバがクライアントを認証する必要がある。認証をより確かなものにするために、サーバはクライアントが認証サーバによって認証されたことの証明である認証子を要求する。

ここではクライアントはサーバの公開鍵を、サーバはクライアントの公開鍵を、また利用者の認証を行った認証サーバの公開鍵を知っているものとする。もし、サーバとクライアントが同じリージョンに属しているとすると、クライアントは認証サーバにサーバの鍵を要求し、認証サーバはその鍵を自分のデータベース内に管理しているのでそれを取り出し返送すればよい。しかし、クライアントとサーバが異なるリージョンに属している時は、認証サーバはサーバの鍵を取り出すことができない。このため認証サーバはそのサーバの鍵を管理している認証サーバに転送を要求するようにする。この時認証サーバが、どのようにして相手の認証サーバを決定するかという問題がある。この問題に答えるプロトコルは 5.4.6章、5.4.7章で述べる。

1. クライアントはセッション鍵 SSK_Cを生成する。

2. C→S:

$$C, S, \{\{C, AS, SSK_C, Com, Authenticator\}^{SK_C}\}^{PK_S} \qquad (4-1)$$

   ここで ASは利用者を認証した認証サーバ、つまり $Authenticator$ を発行した認証サーバのことを表わす。

3. サーバはこのメッセージを復号し、コミュニティ名、認証子の有効期間のチェックを行なう。このメッセージはクライアントの秘密鍵で暗号化されているので確かにクライアントが作成したメッセージであるといえる。

4. S→C:

$$S, C, \{\{S, SSK_C, SSK_S\}^{SK_S}\}^{PK_C} \qquad (4-2)$$

   SSK_Sはサーバが生成したセッション鍵である。

5. クライアントは SSK_Cの整合性を確かめてサーバがクライアントを認証したと知る。

6. C→S:

$$C, S, \{\{C, SSK_S\}^{SK_C}\}^{PK_S} \qquad (4-3)$$

7. サーバは送られてきた SSK_Sの整合性を確かめて replay attack を発見する。

ここでクライアントとサーバがお互いを認証できた。以後のクライアント、サーバ間の通信は SSK_C、SSK_Sを鍵として暗号化する (図 5.1)。

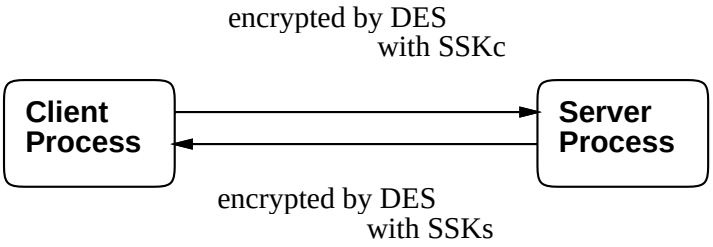

図 5.1: 認証後のクライアント・サーバ間の通信

5.4.6　サーバの公開鍵の入手

クライアントがサーバの公開鍵を入手するプロトコルを述べる。このときに問題となるのが「認証サーバは自分のリージョン外のサーバがどのリージョンに属しているか分からない」ことである。このため、認証サーバは「そのサーバを知っていそうな認証サーバ」を知っている必要がある。実際のプログラムではコンフィグレーションファイルとして与えるがその詳細は 5.5章で述べる。

ここではクライアントが属するリージョンの認証サーバを AS_1、サーバを知っていそうな認証サーバを AS_2とする。

1. C→ AS_1:
$$C, AS_1, \{\{C, S, SSK_C\}^{SK_C}\}^{PK_{AS_1}} \qquad (5-1)$$

2. ここで Sがクライアントと同じリージョンに属しているならば認証サーバ AS_1はサーバの公開鍵を知っているのでプロトコル (5–4) を送る。一方 Sが異なるリージョンに属しているならば AS_1は Sを知らない。AS_1は Sを知っていそうな認証サーバ AS_2に次のメッセージを送る。

3. $\mathrm{AS}_1 \rightarrow \mathrm{AS}_2$:
$$AS_1, AS_2, \{\{AS_1, S, SSK_{AS_1}\}^{SK_{AS_1}}\}^{PK_{AS_2}} \qquad (5-2)$$

4. もし AS_2が Sの公開鍵を知っているならば、それを AS_1に送る。

5. $\mathrm{AS}_2 \rightarrow \mathrm{AS}_1$:
$$AS_2, AS_1, \{\{AS_2, S, SSK_{AS_1}, PK_S\}^{SK_{AS_2}}\}^{PK_{AS_1}} \qquad (5-3)$$

6. もし AS_2が Sを知らなければ、同様に Sを知っていそうな認証サーバ AS_3に (5–2) のメッセージを送り (3–3) のメッセージを受け取る。PK_S を知った AS_1はクライアントにそれを送る。

7. $\mathrm{AS}_1 \rightarrow$ C:
$$AS_1, C, \{\{AS_1, S, SSK_C, PK_S\}^{SK_{AS_1}}\}^{PK_C} \qquad (5-4)$$

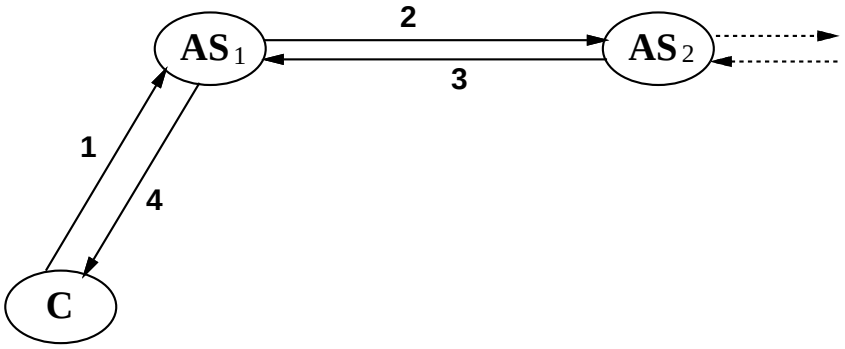

図 5.2: サーバの公開鍵を入手する手順

サーバを知っている認証サーバを求めて (5–2) のメッセージがループするという可能性があるので、実装するときはホップカウントを付け加える必要がある。

5.4.7 クライアントの公開鍵の入手

このプロトコルは 5.4.6章のものと同様である。サーバはクライアントが送ってきた認証子の有効期限を調べる。このとき、この認証子を発行した認証サーバの計算機とサーバの計算機が同じ時刻を刻んでいればよいのだが、内部クロックは事故または故意に変更されることがあり、サーバのワークステーションの内部クロックを信用するのは安全とは言えない。そこで、ネットワークで接続されている計算機の内部クロックは同期がとれていないという前提のもとでプロトコルの設計を行う。認証子の有効期限を調べるためには認証子を発行した認証サーバの内部クロック時刻を知る必要があるのでクライアントの公開鍵とともにその認証サーバの現在時刻をサーバに送るようにする。

1. S→ AS_1:

$$S, AS_1, \{\{S, C, SSK_S\}^{SK_S}\}^{PK_{AS_1}} \qquad (6-1)$$

2. $\mathrm{AS}_1 \rightarrow \mathrm{AS}_2$:

$$AS_1, AS_2, \{\{AS_1, C, SSK_{AS_1}\}^{SK_{AS_1}}\}^{PK_{AS_2}} \qquad (6-2)$$

3. $\mathrm{AS}_2 \rightarrow \mathrm{AS}_1$:

$$AS_2, AS_1, \{\{AS_2, C, SSK_{AS_1}, PK_C, timestamp\}^{SK_{AS_2}}\}^{PK_{AS_1}} \qquad (6-3)$$

4. $\mathrm{AS}_1 \rightarrow$ S:

$$AS_1, S, \{\{AS_1, C, SSK_S, PK_C, timestamp\}^{SK_{AS_1}}\}^{PK_S} \qquad (6-4)$$

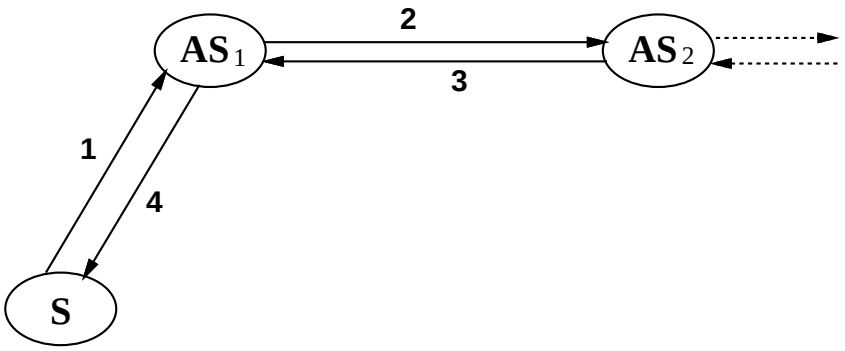

図 5.3: クライアントの公開鍵を入手する手順

5.5　リージョンファイル

自分のリージョン内にないサーバがどのリージョンに属するのかを知るためにどうするかという問題があった。他のリージョンに属するサーバの名前とその公開鍵を知っている認証サーバが稼働している計算機をすべて並べたファイルを書けば良いが、サーバの数が何百にもなる場合があるので、この方法は現実的とは言えない。

リージョンは DNS ドメインとは関係ないのだが、実際に SPLICE/AS-II を利用するときは DNS ドメイン毎にリージョンを分けることが多いと予想される。このため、サーバ名、そのサーバが稼働している計算機の DNS ドメイン名でエントリを検索する方法が有効と思われる。ただし、DNS ドメインに従わないサーバのエントリも許さなければならない。SPLICE/AS-II では以下のような設定ファイルを考えた。

$$
\begin{array}{rcl}
< Regionfile > & ::= & \{< Regionentry >\} \\
< entry > & ::= & < \text{ログイン名} > \quad < \text{サーバアドレス} > \quad < \text{認証サーバアドレス} > \\
< \text{ログイン名} > & ::= & \text{サーバ名} \quad | \quad \texttt{ALL} \\
< \text{サーバアドレス} > & ::= & \text{ホスト名} \ . \ DNS\text{ドメイン名} \\
 & | & DNS\text{ドメイン名} \\
 & | & \texttt{ALL} \\
< \text{認証サーバアドレス} > & ::= & \text{ホスト名} \ . \ DNS\text{ドメイン名}
\end{array}
$$

サーバ名とその計算機のフルネームが与えられたとき、このファイルを 1 行目から順に検索していき、一番最初に見つかったエントリを利用する。"ALL" はすべてにマッチする予約語であり、これをデフォルトルールと呼ぶ。以下に例を示す。

```
ALL         rd.ecip.osaka-u.ac.jp        asama.rd.ecip.osaka-u.ac.jp
ALL         ecip.osaka-u.ac.jp           raicho.rd.ecip.osaka-u.ac.jp
stelnet     arpeggio.ics.osaka-u.ac.jp   raicho.rd.ecip.osaka-u.ac.jp
```

```
ALL          ics.osaka-u.ac.jp          helios.ics.osaka-u.ac.jp
ALL          ALL                        a.b.c.d
```

このとき、*.ecip.osaka-u.ac.jp のサーバはすべて raicho のリージョンに属しているが、そのサブドメイン*.rd.ecip.osaka-u.ac.jp のサーバは $asama$のリージョンに属している。一方、*.ics.osaka-u.ac.jp のサーバはすべて helios のリージョンに属しているが、唯一 arpeggio のサーバ stelet だけは raicho のリージョンに属している。どのリージョンに属しているのか分からない場合はとりあえず a.b.c.d に属していることにする。以後は、a.b.c.d が処理を終えメッセージを送るのを待つ。

このようにデフォルトルールを加えることにより、すべてを列挙するときに比べエントリ数が減り、他リージョンのサーバの追加、削除には全く関与しなくてもよくなる。

最後に、このファイルは認証サーバだけが参照するファイルなので一般ユーザからは見ることができないようにしている。

5.6 コミュニティファイル

5.4.1章でクライアント、サーバは自分のコミュニティの認証サーバのある所を知っていることを前提とした。実際にはこれをコミュニティファイルというコンフィグレーションファイルとして与えている。その文法は

$$
\begin{aligned}
< Communityfile > &::= \{< entry >\} \\
< entry > &::= \text{コミュニティ名}\quad\text{認証サーバのポート番号} \\
&\quad\ < \text{マスタサーバ} > \quad < \text{スレーブサーバ} > \\
< \text{マスタサーバ} > &::= \text{マスタサーバが稼働しているホストアドレス} \\
< \text{スレーブサーバ} > &::= [\text{スレーブサーバが稼働しているホストアドレス}]
\end{aligned}
$$

と定めている。全てのクライアント、サーバプログラムは起動時にこのファイルを読むことで、接続すべき認証サーバのアドレス、ポート番号を得ることができる。このファイルは全てのユーザから見えなければならない、また、変更されても認証サーバには影響を及ぼさなので、全ての利用者が読むことができるようなパーミッションを設定してある。以下に例を示す。

```
genesis          2500          asama.rd.ecip.osaka-u.ac.jp
miya             2500          helios.ics.osaka-u.ac.jp
```

5.7　利用者、サーバの名前表現

コミュニティ内で各利用者を識別するために、利用者に次のような情報を持たせ、それによって識別する。

```
typedef struct {
  IPaddr          address;
  LoginName       name;
  char            host[64];
}ID;
```

*address*はクライアントプロセスを実行している計算機の IP アドレスであり、*name*は利用者のログイン名、*host*は利用者の属するリージョンの認証マスタサーバのアドレスである。クライアントプロセスはどの計算機で実行してもよいとするので、利用者のログイン名は、リージョン内で唯一であればよい。

一方、各サーバを識別するためにも上の構造体を用い、*address* はサーバプロセスを実行している計算機の IP アドレス、*name*はサーバ名、*host*はサーバの属するリージョンの認証マスタサーバのアドレスである。

5.8　暗号

5.8.1　利用者認証のための暗号

SPLICE/AS-II では利用者の認証をインターネット環境で用いることの拡張性を考えてパスワードで認証を行う方法を採用した。このためには 3章で述べたように一対一であり、かつ、容易に逆変換できてはならい暗号が必要である。この条件を満たす暗号としては UNIX の *crypt(3)*関数の暗号化が一般的である。これは全ての UNIX で標準で提供されており、かつ、現在のところ逆変換のアルゴリズムは発表されていない。よって SPLICE/AS-II ではこの暗号を用いることにする。

UNIX の`/etc/passwd` ファイルもこの関数で暗号化された情報を持っているが、このファイルは一般ユーザでも閲覧可能であるところに問題がある。暗号の逆変換はできないが、適当な単語を *crypt(3)*で暗号化し、それを`/etc/passwd` のエントリと見比べることによって他人のパスワードを知ることが可能である。筆者の経験でみると二ヵ月に一度はアイドルの名前をパスワードに使っている人を発見することができた。筆者が調べているユーザ数は 100 人前後であるが、この人数で、この確率で発見できるのだから、いかにパスワードが無防備であるかを知ることができる。このため、System V 系の UNIX では`/etc/passwd` のパスワード部分だけを一般ユーザからは見えないファイルに書く *shadow passward*を使うことが一般的となっている。SPLICE/AS-II でもパスワードファイルだけでなく、秘密鍵も管理しなければならないので、データベースファイルは一般ユーザからはアクセスできないようにする。

5.8.2 通信情報暗号化のための暗号

SPLICE/AS-II では RSA をソフトウェアで実現するため、その速度の問題から先で述べたブートストラップ暗号を用い、RSA の鍵は 680bit としている。これは 10 進数に直すと約 200 桁の数字となり、現在の暗号処理の速度と暗号の安全性を考えてこの大きさにした。

RSA の鍵生成はなるべくランダム性をユーザの意志に依存させるよう配慮した。そのための方法として利用者にキー入力をしてもらい、その入力した文字コードとキーストロークのインターバルを CPU タイムで計測し、それを乱数の種として利用している。鍵生成時の様子をリスト 1 に示す。ただしこの出力は鍵生成テストプログラムを実行しているので、生成した鍵が画面に表示されている (もちろん実際の鍵変更プログラムでは表示しない)。

これは SPARC station IPX(27MIPS Memory 13MB) での実行結果である。CPU タイムが 64.3(43.3 + 21.0) 秒であることを見れば鍵生成にかなり時間がかかることが分かる。鍵生成に時間がかかることは悪いことではない。なぜならば、手あたり次第に鍵を生成し、それを使って暗号文を復号化するといった攻撃をやりにくくするからである。

一方、認証後のクライアント・サーバ間の通信は DES を CFB モードで暗号化している。

```
nozomi$ time ./keygentest

Generating an RSA key with a 677-bit modulus...

We need to generate 93 random bytes.  This is done by measuring the
time intervals between your keystrokes.  Please enter some text on your
keyboard, at least 186 nonrepeating keystrokes, until you hear the beep:
................................................................
.................?..............................................
...................................................ooooOOO*
-Enough, thank you.
Ignored 1 keystrokes that were typed too fast.
....++++ .................................++++

SK.d:000000000000000003386380284234279582215194428534191234842375059388
85753013060976609224676180011063034644964745695393308206468667679623364
2594193504421966799484086213943529393126612142642845189649556313708817
SK.p:0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000110243427373687493705227 49
886166340928830755357242361990421960828318383045433997473845504107030 3
SK.q:0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000877063541951643825298159 75
2501763896786135963117675951496895173668051022984871089316411077696177
SK.u:0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000662953068107713216377005 37
5524857239121066548886159027589112681817701735826403712781314139891742
PK.e:0000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000017
PK.n:0000000000000000022201489602325587000263624812891964709915296637828
97799556515219950865695260737008390643952587182921326300488104644602241
10203121548276546744813450581718955834517715206401410788594030323051 99

       66.8 real         43.3 user         21.0 sys
```

リスト 1: 鍵生成の様子

第 6 章

応用

SPLICE/AS-II の認証機能を利用するために今回 telnet サービスを SPLICE/AS-II 対応に改造した。telnet のソースコードは BSD-tahoe のものを用いたが、システムコールを呼び出している箇所を以下のように書き直し、認証を行なうライブラリ関数をクライアント側、サーバ側にそれぞれ 1 行加えただけである。

ただし、read、recv、write、send はソケットを対象にしたシステムコールのみを書き換える。

このように比較的簡単に SPLICE 化が行え、実際に使ってみてもログイン時に 1 分ちかく要するが、それ以後は普通の telnet と変わらない使用感で安全な環境を提供することができた。

変更前	変更後
read	splice_client_read
write	splice_client_write
recv	splice_client_recv
send	splice_client_send

表 6.1: クライアント側の変更

変更前	変更後
read	splice_server_read
write	splice_server_write
recv	splice_server_recv
send	splice_server_send

表 6.2: サーバ側の変更

	関数名
クライアント側	splice_client_authentication
サーバ側	splice_server_authentication

表 6.3: 認証を行なうライブラリ関数

第 7 章

システムの評価、考察

7.1 性能

今回作成したシステムで実際の認証にどのくらい時間がかかるのか測ってみた。時間を計測するためには *getrusage(2)*を用い、認証を行う直前と直後にこのシステムコールを呼び出し、その差をとった (使用した計算機は SPARC station IPX)。その結果を表 7.1 に示す。

RSA をソフトウェアで実現しているため、このようにかなりの CPU 時間を必要としている。プログラムを UNIX の profile 機能を用いて調べたところ、暗号・復号処理部分、特に配列同士の乗算と剰余の計算ルーチンから呼び出される足し算、引き算ルーチンにかなりの CPU 時間が消費されていることが明らかになった。このルーチンだけを CPU 特性に合わせたコードを書くことによってより高速化が計れると思われる。

7.2 サーバの起動

また、現在のシステムではサーバが自動的に起動しない設計になっている。サーバプログラムを起動する際にはパスワード入力が必要であり、誰かが手で入力する必要がある。このことはサーバの管理者が管理するサーバの数だけパスワードを覚え、起動し、それを入力しなければならないことを意味する。これを煩わしいと思う人もいれば、セキュリティのためだから仕方がないと考える人もいると思われる。自動的に起動するほうがよいとは誰もが思うであろうが、この管理者が自分の管理しているサーバを責任を持って定期的にチェックすることによりプログラムの改変を発見することができるので、Kerberos

	認証サーバ (リージョン内認証)	認証サーバ (リージョン外認 証)	クライアント	サーバ
USER time	14.72	21.74	18.75	10.81
SYSTEM time	0.22	0.21	0.11	0.03

表 7.1: 認証過程で消費された CPU 時間

での問題であったプログラムの改ざんに対して無防備であるという問題が改善される。

7.3　システムの運用

一般にセキュリティを向上させるシステムを使用するにはどこからどこまでが secure なのかということを知っておかなければならない。SPLICE/AS-II は現システムの根底から改造したわけではなく、unsecure な環境の上に secure な環境を実現させただけであるので、どこかにその境界が存在するわけである。利用者はそれがどこなのかを理解してシステムを使用しなければ SPLICE/AS-II が意味のないものになってしまう。

第 8 章

おわりに

SPLICE/AS-II では他の認証システムにはないコミュニティという概念を導入し、柔軟なアクセス制御機能を提供し、また、コミュニティをリージョンに分けることにより広域分散ネットワークでの認証を行うことを可能とした。

本研究ではこのシステムのモデルの定義、プロトコルの設計、データベースの設計を行い、SPLICE/AS-II システムの実装を行った。暗号処理部は、既存ツールの RSA 暗号、DES 暗号ソフトウェアの一部変更を行いシステムに組み込んだ。そして、telnet の SPLICE 化を行い安全なリモートログイン環境を提供した。

セキュリティ問題はシステムを提供する側だけでなく、利用者側も十分理解しなければならないものである。一つの計算機が侵入されると、その計算機を足掛かりにして別の計算機へ次々と侵入を繰り返すことが可能となるので、ある計算機のセキュリティ管理を怠ることは、自分は構わないと思っていても他人の迷惑になりうる。しかし、現在セキュリティツールが身近なものになっていないせいかユーザのセキュリティ意識は浅い。

現在 SPLICE/AS-II では未実装の部分が幾つかあり、今後これを行い、さまざまなツールの SPLICE 化、そして普及に努めていく予定である。