國立交通大學

資訊科學與工程研究所

碩士論文

自動偵測認證協定漏洞系統

Automatically detecting flaws in authentication protocols

研究生：黃信達

指導教授：楊武　教授

中華民國　九十五　年六月

自 動 偵 測 認 證 協 定 漏 洞 系 統
Automatically detecting flaws in authentication protocols

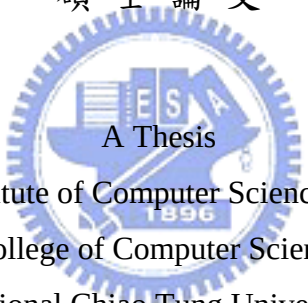
研 究 生：黃信達　　Student：Hsin-Ta Huang

指導教授：楊 武　　Advisor：Wuu Yung

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文



A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年六月

自動偵測認證協定漏洞系統

研究生：黃信達　　指導教授：楊武 博士

國立交通大學資訊科學與工程研究所

摘　　要

由於現今網路世界的發達，人們越來越喜歡透過網路交易、網路付費，因此認證協定是一個非常重要的一環，而他的安全性和正確性一直受到關注。在前人的研究中提出了各式各樣分析的方法，然而這些方法通常都需要一些假設，也因此只有很有經驗的人才能理解其中的奧妙。因此在本論文中，我們提出了一個不同於以往的想法來分析認證協定的安全性，而這個方法對於設計認證協定的設計者會是一個很好的小幫手。

Automatically detecting flaws in authentication protocols

Student: Hsin-Ta Huang **Advisor: Dr. Wuu Yang**

Institute of Computer Science and Engineering

National Chiao Tung University

Abstract

Because it is not very secure in the network environment, an authentication protocol plays a more and more important role under the current network environment. If the authentication protocol has flaw, it will be very serious. You cannot make sure whom you communicate with. The soundness of an authentication protocol has been drawing many spotlights since past years. Many logic systems have been proposed in plenty of papers. However, they are either hard to implement or need lots of experience. Then we suggest detection methods to reveal the potential vulnerabilities of an authentication protocol. We think that the proposed methods will be a helpful tool for an authentication designer.

致謝

首先我要謝謝我的指導老師 楊武教授，沒有老師不厭其煩的督促我和不斷的幫助和指導，我是沒辦法順利完成這篇論文。在此，向老師致上最深的謝意。

接下來，我要感謝實驗室的所有同學，大家的陪伴讓我在這兩年來的生活中感到許多溫暖。宸瑋是和我要一起畢業的同學，我們同舟共濟，互相鼓勵，還有女友的鼓勵、支持和學弟們的關心，讓我不會承受不了撰寫論文的壓力。

最後感謝我的家人，爸爸、媽媽、哥哥、姐姐，在我念研究所的日子裡不斷的關心和鼓勵我，尤其是在碩二撰寫論文時。對於他們的關心，讓我覺得很窩心、感動。最後僅以此論文獻給我最親的家人和朋友，由衷地謝謝他們。

目錄

圖目錄

第一章 序論

1.1 研究動機

在現今的社會中，由於網路的快速發展，使得電子商務、電子錢包、網路繳費、付費網站、網路商店等等各類的商機和應用如雨後春筍般出現在網路世界中。然而網路是一個虛擬的世界，而且在網路上傳送的資料任何人皆可以攔截、複製、和竄改，因此人與人之間如何透過網路來確認彼此的身分是一個非常重要的課題。認證協定就是用來達到此目的的一個確認彼此身份的方法，然而當一個認證協定不安全時，那麼受害者的資料將被利用來作爲欺騙別人的工具，就像現今的詐騙集團，不管被騙的是誰，有利的永遠是他們而且仍然可以逍遙法外。

1.2 研究目標

在這篇論文中，我們假設認證協定所使用的加密方法是完美沒有缺陷的，也就是說 Attacker 無法利用密碼學上的攻擊方式來攻擊認證協定，譬如像利用 Known plaintext、Chosen plaintext、Chosen ciphertext、Chosen text attacks 這些攻擊方法找出加密的 key 或是找出 hash function 的碰撞來攻擊。因此在這裡提到的漏洞指的是認證協定邏輯上的漏洞，譬如像利用 Man in the middle attack、Parallel session attack、Type flaw attack、Freshness attack 這些攻擊方法僞裝成別人欺騙對方，或是拿已經知道的資料當成 session key 來欺騙對方，而這些漏洞通常都是設計者沒有考慮的很詳細下所產生出來的邏輯上的漏洞。接下來在這篇論文中提到的漏洞指的都是邏輯上的漏洞。

由於以前的學者們設計出各式各樣的認證協定，然而這些認證協定在事後被證實有漏洞的存在，甚至連修改後的版本也仍然有漏洞存在。因此就有學者根據已經存在的漏洞來分析一個好的、安全的認證協定需要有哪些特性才能夠避免這些漏洞，但是如果根據檢查這些特性來決定認證協定是否安全沒有漏洞將會是一

件非常危險的事情，因爲如果今天有一個特性沒有被發現，那麼直到這個特性被發現之前永遠不會找到這個漏洞。因此就有學者設計驗證演算法來確認認證協定是否安全，而在分析認證協定安全性的演算法範疇中，最著名的莫過於 BAN logic[1]，BAN logic 是使用邏輯假設來推導認證協定是否安全，只是 BAN logic 需要一個很有經驗的人在分析前先列出需要的邏輯假設，但是只要邏輯假設錯誤那麼分析結果也會跟著錯誤，所以要用電腦程式來實作 BAN logic 是一件非常困難的工作，另外在[2, 10]中，也指出 BAN logic 不足的地方而且並不能適用於分析每一個認證協定。

因此我們希望能設計出一個容易理解和實作的演算法，而且不只是被動的找出是否擁有前人分析出來必須擁有的特性來判斷是否安全。除了一些已經知道的攻擊之後還希望能找出更多可能的攻擊方法。另外對一個安全分析人員來說，要知道一個系統有什麼漏洞，必須要先知道有哪些攻擊方法。也就是說要成爲一個安全防護人員之前要先成爲一個駭客。站在駭客的角度來看，知道有哪些攻擊之後，才能知道要如何去防護。所以這個演算法的中心概念就是站在駭客的角度來看如何攻擊認證協定。

因此整體而言總共有以下幾個目標：

- 從駭客的角度思考如何攻擊認證協定
- 設計一個容易理解和實作的演算法
- 不只是固定的找出特性而被攻擊的漏洞

1.3 相關研究

在[6, 9, 11]三篇文章中，他們皆是去分析認證協定邏輯上的漏洞，探討這些漏洞的原因、產生的原因，和防禦的方法。

- [9]一文中，將認證協定中各式各樣的漏洞做了一個整理，一一的分析探討這些漏洞的原理、如何產生，另外也說明了如果避免出現這些漏洞。

- [11]一文中探討了 SSL3.0 不安全的地方，包括 SSL 本身架構設計上所產生的漏洞，同時也說明 SSL 如何防禦一些邏輯上的漏洞。
- [6]一文中探討了 UMTS 中因爲要考慮向下相容的關係，導致 UMTS 和 GSM 用戶溝通時產生了 Man in the middle attack。

另外在[7]這篇文章中，作者設計一個用來確認認證協定是否安全的演算法，並詳細的說明他們提出的方法的原理、概念和模擬結果。

- [7]一文中將認證協定每一個訊息和他們彼此間的關係對應出來。譬如第一個訊息可以得到某筆資料，第二個訊息可以取得另一筆資料，而這兩個訊息間的關係是只要將第一個訊息的資料傳送給第二個訊息的接收者，Attacker 就可以取得第二個訊息的資料。而利用訊息彼此間的關係來找出攻擊認證協定的方法。

1.4 論文架構

在第二章我們會介紹認證協定基本概念和常用的標記，另外還有本文中會用到的標記和定義。第三章詳細的介紹我們演算法的想法、架構和可以找出哪些範圍的漏洞。第四章顯示程式的介面和分析模擬後的結果。第五章結論和提出未來可以改進的方向。

第二章 何謂認證協定

2.1 認證協定的基本概念

- 認證協定指的是至少兩個或兩個以上的參與者經過一連串的訊息傳遞後，可以確定彼此身份的一系列步驟。
- 認證協定必須由第一個訊息開始，由最後一個訊息結束，而且訊息必須從第一個依序到最後一個，其中任何一個訊息不能沒有執行到，否則認證協定就會中止。
- 認證協定中的參與者通常除了要溝通的雙方之外還會有一個受信任的第三者（通常用S來代表受信任的第三者）。
- 認證協定中所使用的加解密演算法有三種類型，第一類是對稱式加密演算法、第二類是非對稱式加密演算法，最後一類是前兩個類型交互使用。

2.2 認證協定常用的標記

- Na：由參與者 A 產生的亂數。
- Ta：由參與者 A 產生的 Timestamp。
- A：參與者 A 的 identity。
- Kas：代表只有參與者 A 和受信任的第三者 S 才擁有的 key。
- Kab：參與者 A 和參與者 B 在這次溝通中用來加解密的 Session key。
- {message}Kas：message 用 Kas 加密。

2.3 自訂的標記和定義

- Message：認證協定中的每一個訊息都是 Message，也就是說認證協定是由多個 Message 所組成的。如圖 2.1 總共有五個 Message。
- Item：以圖 2.1 的第一個 Message 來說，這個 Message 有三個 Item，分別是

A、B、Na。Item 可以是沒有加密過的，也可以是有加密過的，甚至於可以是多重加密的（例如{Na, A, {Na}Kas}Kas、{Na}Kas 都是 Item）。

- 資料數量：有多少個 Item。如果是要計算 Message 有多少個資料數量，那麼只要計算這個 Message 有多少 Item 即可，不必管 Item 是否有多重加密。如果是要計算一個 Item 有多少個資料數量，假設這個 Item 沒有加密，那麼資料數量只有一個，假設這個 Item 有加密，那麼只要計算被這個 Item 加密的 Item 有多少個，一樣不必管 Item 是否有多重加密。舉例來說，圖 2.1 的第二個 Message 只有一個資料數量（也就是{Na, B, Kab, {Kab, A}Kbs}Kas），而這個 Item 有四個資料數量（也就是 Na、B、Kab 和{Kab, A}Kbs）。
- 重要的 Item：顧名思義指的是 Item，而這個 Item 必須是有加密過的，並且還要跟接下來的 Message 比對才可判斷是否是重要的 Item。判斷方式詳見 **3.2.2** 節。
- 有意義的 Item：Attacker 可以製造出來或拿的到也看的懂的 Item，也就是說 Attacker 可以控制這個 Item。如果 Item 沒有加密，很顯然 Attacker 可以製造出這個 Item 而且也看的懂這個 Item 是什麼；如果 Item 有加密，Attacker 不一定可以製造的出來或看的懂這個加密的 Item 裡面的資料是什麼。
- 沒有意義的 Item：Attacker 無法製造出來或拿的到但看不懂的 Item。很明顯沒有意義的 Item 一定是有加密的 Item，因爲這樣 Attacker 才會看不懂是什麼資料。

1. A->S: A, B, Na
2. S->A:{Na, B, Kab, {Kab, A}Kbs}Kas
3. A->B:{Kab, A}Kbs
4. B->A:{Nb}Kab
5. A->B:{Nb-1}Kab

圖 2.1: Needham-Schroeder Protocol

第三章 系統設計與實作

在這一章節中，我們開始詳細的介紹我們提出的驗證演算法，在顯示這個演算法的虛擬碼時也同時說明這樣子做的原因。

3.1 Proposed 驗證認證協定演算法的介紹

要攻擊金融機構的安全系統，必須要先對他們的系統有深入的了解，剖析之後才能知道要怎麼攻擊這個系統。在法律方面，要鑽法律漏洞必須要先對法律條文滾瓜爛熟之後才能知道有哪些法律遺漏、沒有規範到的事情。即使在認證協定也是一樣的，要防止任何人入侵，破壞我們的系統之前，必須要先知道我們的系統有哪些漏洞可能會被攻擊，也就是說要防護之前必須要先知道怎麼破壞。因此爲了要找出認證協定的漏洞，我們希望能夠站在 Attacker 的角度上來剖析認證協定。當 Attacker 要攻擊認證協定的時候，他並不是想要攻擊就可以攻擊成功，而是會去把所有可以搜集的資料都搜集齊全，接著在尋找如何利用這些現有的資料去攻擊。這個想法就是我們演算法的中心概念。

以 Attacker 的想法來剖析認證協定之後，我們發現兩個重要的特性。

- **要攻擊的目標 Item (重要的 Item)**

Attacker 攻擊認證協定時，並不是真的盲目的用暴力法去攻擊整個認證協定，而是會去剖析認證協定找出認證協定中的缺陷，就像鑽法律漏洞一樣，另闢一條不正當的路徑來達到相同的目的。在我們看了很多認證協定後，我們發現認證協定中的 Item 可以分爲兩類，一類只是單純的訊息傳遞而已，另一類是用來確認身分的 Item。因此很明顯的 Attacker 要攻擊的 Item 就是認證協定中用來確認身分的 Item。只要 Attacker 能夠攻擊這些 Item 就可以成功的攻破認證協定。

- **可以利用的資源**

當 Attacker 決定了攻擊目標之後，不會就這樣一股腦的去攻擊目標，而是會盡量的去蒐集所有他可以到手的資料，然後想盡辦法去攻擊它。另外我們都知道網路是一個很不安全的地方，任何人都可以在網路上攔截封包，當然 Attacker 也不例外。因此很明顯的 Attacker 可以拿到要攻擊的認證協定的所有 Item，拿到這些 Item 之後，Attacker 會盡其所能的想辦法直接使用這些 Item 來當成要**攻擊的目標 Item(重要的 Item)**或是用來製造出要**攻擊的目標 Item(重要的 Item)**，以便來欺騙別人。

因此我們希望能找出所有**可以利用的資源**，並利用這些**可以利用的資源**來攻擊要**攻擊的目標 Item(重要的 Item)**。

3.2 Proposed 驗證認證協定演算法

在這一小節中，我們會先介紹我們設計用來儲存認證協定中的 Message、Item 這些資料的 Data Structure 並舉例說明如何將這些資料完美的連結起來。接下來我們會介紹如何找出 3.1 節裡提到的**攻擊的目標 Item(重要的 Item)**和**可以利用的資源**這兩個特性。接著再繼續介紹我們如何找出認證協定的漏洞。

3.2.1 Data Structure

在這一小節中，我們提出我們設計的 Structure Message 和 Structure Item，這兩個 Structure 分別是用來儲存認證協定中的 Message 和 Item。

```
struct Message
{
    String source;              //記錄 source
    String destination;         //記錄 destination
    Item item;                  //記錄 item;
    Message *next;              //記錄下一個 Message 的位置
}

struct Item
{
    String content;         //記錄 Item 的內容
    String key;             //記錄用來加密的 key
    int number;             //記錄總共有多少個 Item 數量
    int location;           //記錄 Item 是屬於認證協定的第幾個 Message
    Boolean important; //記錄 Item 是否是重要的 Item，預設是 False
    Boolean flag;           //記錄是否已經找到可以攻擊的 Item，預設是 False
    Item*next;              //記錄下一個 Item 的位置
    Item*include;           //記錄被加密的 Item 的位置
}
```

舉例說明：

假設有一個 Message 是 A -> B : A, {A, {Na}Kas}Kbs, Na。

1. 宣告一個 Message 變數是 message。

2. parsing 此 Message 後

   2.1 A 會記錄在 message.source

   2.2 B 會記錄在 message.destination

   2.3 而 Message 的 item（A, {A, {Na}Kas}Kbs, Na）經 parsing 後會記錄在 message.item

   2.4 *next 記錄下一個 Message 的位置。沒有下一個 Message 就是 NULL。

3. 首先分析第一個 item：A

   3.1 由於這個 item 沒有被加密，所以變數 content 會被記錄爲 A。

   3.2 由於這個 item 沒有被加密，所以 message.item.key 仍然是 NULL。

   3.3 由於這個 item 沒有被加密，所以資料數量只有一個，number 被記錄爲 1。

   3.4 假設此 Message 是認證協定的第一個 Message，則 location 會被記錄爲 1。

   3.5 當此 Item 被分析爲重要的 Item 時，important 才會被記錄爲 True。

   3.6 flag 記錄該 item 是否已經被找到可以用來代替自己的 item。

   3.7 *next 記錄下一個 item（{A, {Na}Kas}Kbs）的位置。

   3.8 由於這個 item 沒有被加密。因此變數 include 仍然是 NULL。

4. 接下分析第二個 item：{A, {Na}Kas}Kbs

   4.1 由於這個 item 有被加密，因此變數 content 仍然是 NULL。

   4.2 由於這個 item 有被加密，因此變數 key 會被記錄爲 Kbs。

   4.3 由於這個 item 有被加密，計算出資料數量爲二，因此 number 被記錄爲 2。

   4.4 假設此 Message 是認證協定的第一個 Message，則 location 會被記錄爲 1。

   4.5 當此 Item 被分析爲重要的 Item 時，important 才會被記錄爲 True。

   4.6 flag 記錄該 item 是否已經被找到可以用來代替自己的 item。

   4.7 *next 記錄下一個 item（Na）的位置。

   4.8 由於這個 item 有被加密，因此*include 會記錄被加密的第一個資料的位置。在這邊是記錄 A 的位置。接著會將 A、{Na}Kas 按照這個方式分析

5. 全部分析完畢後，會變成圖 3.1 這樣的結構。因爲 important 和 flag 這兩筆資料在分析重要的 Item 和尋找漏洞的時候才有可能改變，因此在圖 3.1 中我們就不顯示這兩筆資料的內容。

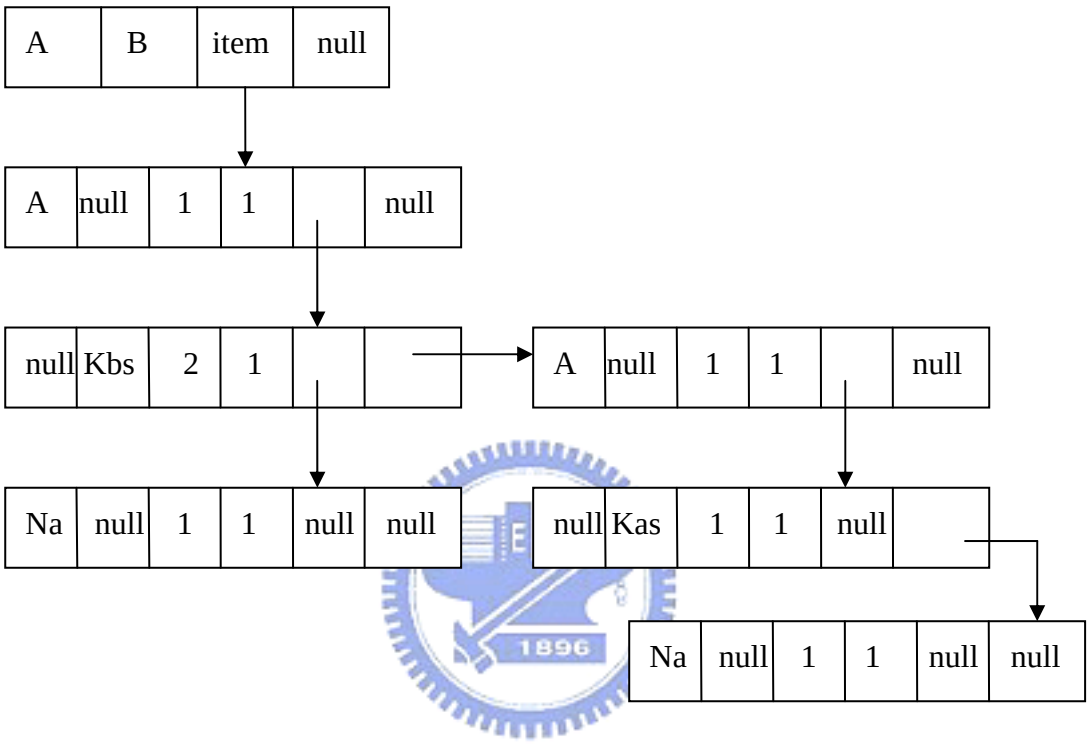


圖 3.1: Message & Item Linked list

3.2.2 尋找攻擊的目標 Item(重要的 Item)

在我們想好驗證認證協定演算法的設計概念之後，接著就開始設計我們的驗證認證協定演算法，首先就是要先找出 3.1 節裡提到的**攻擊的目標 Item(重要的 Item)**和找出**可以利用的資源**。只有找好這些資料後才可以來尋找認證協定是否存在漏洞。

我們先將認證協定裡的每一個 Message、每一個 Item 讀取、parsing 後建成 3.2.1 節中的 Data Structure 並且同時將這些資料一一的串接起來。接下來我們利用建好的 Structure 根據下面的方法（analyze 和 include 這兩個 function）來尋

找攻擊的目標 **Item(重要的 Item)**。

```
void analyze( Message protocol )
{     /*  參數 protocol 代表建好的認證協定 Message 和 Item 的資料串列  */
      int numberOfImportant = 0;                    //記錄找到多少個重要的 Item
      Item important;                               //Item is a linked list;
      Item nextItem;
      Item item_temp;
      Boolean flag = false;
      Item itemUsingKab;
      Message message_now = protocol;

      /*  找出最後一個 Message */
      while( message_now.next != null )
            message_now = message_now.next;
      item_temp = message_now.item;

      /*  處理最後一個 Message */
      while(item_temp != NULL )
      {
            if( item_temp is ciphertext )
            {
                  if( item_temp.key is not session key)
                  {
                        item_temp.important = true;
                        copy the item_temp to important;
```

```
//copy 代表將這個 item 複製到 important(不包含 Structure 裡的 next 欄位)。
                numberOfImportant++;
                flag = true;
            }
            else
                copy item_temp to itemUsingKab;
        }
        item_temp = item_temp.next;
}
if( flag == true && itemUsingKab != null)
{
        itemUsingKab.important = true;
        copy itemUsingKab to important;
        numberOfImportant++;
        flag = true;
}
/*  處理倒數第二個到第一個 Message */
while( message_now != protocol )
{
        /*  找出前一個 Message */
        Message Pre_message = protocol;
        if( Pre_message != message_now )
            while( Pre_message.next != message_now )
                Pre_message = Pre_message.next;
        message_now = Pre_message;
```

```
nextItem = message_now.next.item;
item_temp = message_now.item;
while( item_temp != NULL )
{
    boolean add = false;
    if( item_temp is ciphertext )
    {
        if( item_temp.key is not session key)
        {
          /*  將目前分析的 Message 跟他之後的所有 Message 比對  */
            if( nextItem 是目前分析的 Message 的下一個 Message)
            {
                if( include(nextItem, item_temp) == false )
                {
                    item_temp.important = true;
                    copy the item_temp to important;
//copy 代表將這個 item 複製到 important(不包含 Structure 裡的 next 欄位)。
                    numberOfImportant++;
                    add = true;
                }
            }
            if( include(nextItem, item_temp) == true )
            {
                if( the item which include this item_temp is the
                important item)
                {
```

```
                        item_temp.important = true;
                        copy the item_temp to important;
    //copy 代表將這個 item 複製到 important(不包含 Structure 裡的 next 欄位)。
                        numberOfImportant++;
                        add = true;
                    }
                }
            }
            else
                copy item_temp to itemUsingKab;
        }
        item_temp = item_temp.next;
    }
    if( flag == false && add == true )
    {
        itemUsingKab.important = true;
        copy itemUsingKab to important;
        numberOfImportant++;
        flag = true;
    }
}
if( numberOfImportant == 0 )
    The Protocol does not have any encrypted data。
}
```

Analyze function 的用意就是要找出**攻擊的目標 Item(重要的 Item)**，這個 function 會先找到認證協定的最後一個 Message，然後找出這個 Message 中所有

有加密而且用來加密的 key 不是 session key 的 Item，之後將這些 Item 記錄在 important 並同時遞增 numberOfImportant。如果一個 Message 中找到用 session key 加密的 Item，那麼是否要記錄下來？只有當此 Message 中有不是用 session key 加密的 Item 被記錄到 important 裡的時候才要記錄到 important 裡，否則不理會。

接著找出認證協定的倒數第二個 Message，將這個 Message 中所有有加密而且用來加密的 key 不是 session key 的 Item 找出來，接下來利用 include function 來判斷找到的這些 Item 是否出現在這個 Message 之後的 Message，如果沒有出現在下一個 Message 中就將 Item 記錄到 important 並同時遞增 numberOfImportant；如果有出現在下一個 Message 或是在下一個 Message 之後的 Message 中，就要看這些 Message 中包含這個 Item 的 Item 是否已經被記錄在 important 裡，如果已經在 important 中，那麼這個 Item 也要記錄在 important 裡並同時遞增 numberOfImportant，如果不在 important 裡，那麼這麼 Item 就不用記錄在 important 中。跟分析最後一個 Message 一樣，找出用 session key 加密的 Item，如果有不是用 session key 加密的 Item 被記錄到 important 裡，那麼這個 Item 也要記錄到 important 裡。倒數第二個 Message 分析完後就分析倒數第三個然後倒數第四個一直到第一個 Message 都分析完爲止。

最後如果 numberOfImportant 等於 0，那麼就代表這個認證協定沒有任何加密的資料，因爲都沒有加密的 Item，因此可以得知這個認證協定不安全。

```
boolean include(Item item1, Item item2)
{
    while(item1!= null )
    {
        if( item1.content.indexof(item2.content) != -1 )
//判斷是否包含 item2。如果 item1、item2 是密文，那麼 item1.content、item2.content
//代表加密的全部資料。
```

```
//indexof function 執行字串比對，如果在 item1.content 中找到 item2.content 就會
//回傳在 item1.content 的位置，如果找不到就回傳-1。
                return true;
            item1= item1.next;
        }
        return false;
}
```

include function 是用來找出參數 item1 是否包含 item2，包含的意思是指 item1 的 content 中有 item2 的 content 存在（item1 的內容和 item2 的內容是一樣的或 item2 的內容是 item1 的內容的一部分）。在這個 function 中要注意的地方：如果 item1、item2 是明文，那麼 if(item1.content.indexof(item2.content) != -1) 這個判斷式中的 item1.content、item2.content 很明顯指的就是儲存在 Structure Item 裡面的 content 變數中的值；但是如果 item1、item2 是密文，那麼 item1.content、item2.content 指的就是 item1、item2 的全部資料，雖然這時候 item1.content、item2.content 儲存的資料是 NULL（ex: 如果 item1 代表的資料內容是{A, Na, {Nb}Kbs}Kas，那麼判斷式裡的 item1.content 指的意義是{A, Na, {Nb}Kbs}Kas，雖然實際內容儲存的是 NULL，item2 也是如此）。

上面我們詳細的介紹了尋找**攻擊的目標 Item(重要的 Item)** 的流程，接下來我們開始說明爲什麼要這樣判斷。

首先先介紹一個觀念，在觀察很多認證協定之後，我們發現每一個 Item 可以分成兩類，而這兩類是互斥的，也就是每一個 Item 只會是其中一類，不會兩類都是或兩類都不是。

- Forward：第一類就是 Forward，顧名思義就是從某一個參與者中收到某個 Item 之後，不對它做任何處理而直接將它夾雜在之後的訊息中傳遞給其它參

與者。

- Produce：第二類就是 Produce，顧名思義就是某個 Item 是由參與者根據收到的資料、或是因爲一開始要送出什麼資料所產生出來要傳遞給其它參與者的 Item，而且其它參與者接收這個 Item 之後，對這個 Item 的處理不會是 Forward 的情況。

根據這兩類我們可以尋找我們要**攻擊的目標 Item(重要的 Item)**，首先不管是哪一類只要這個 Item 是沒有被加密的，那麼它就不會是我們要尋找的**攻擊的目標 Item(重要的 Item)**。接下來我們的說明都是假設 Item 是有加密的情況下，首先先說明 Produce 的情況，參與者根據收到的資料或一開始要送的資料所產生出來的 Item，那麼這個 Item 一定有某種意義（通常是用來之後確認身分），所以我們要尋找的**攻擊的目標 Item(重要的 Item)**會是 Produce 這種情況的 Item。那 Forward 只是用來轉送而已，所以他不會是我們要尋找的**攻擊的目標 Item(重要的 Item)**？錯了，因爲 Produce 情況中，產生出來的 Item 如果在之後被接收的參與者做 Forward 的動作，那麼這個 Item 就會被歸爲 Forward，雖然這個 Item 對接收的參與者沒用可是他會是之後的參與者需要用來確認身分的 Item，所以 Forward 在這種情況下的 Item 也是我們要尋找的**攻擊的目標 Item(重要的 Item)**。而這也是我們在 analyze function 中要從最後一個 Message 分析到第一個 Message 的原因，因爲只有這樣才可以找出 Item Produce 之後卻被接收的參與者做 Forward 的動作。

analyze function 中被我們認爲是**攻擊的目標 Item(重要的 Item)**而記錄下來的地方有三個

- 第一個是找出最後一個 Message 中有加密而且加密的 key 不是 session key 的 Item：不把加密的 key 是 session key 的 item 當成**攻擊的目標 Item(重要的 Item)**的原因是因爲他用 session key 加密代表之前的 Message 傳遞中參與者已經成功交換 session key，那麼我們攻擊這個用 session key 加密的 item

是沒用的（請記得我們只探討認證協定邏輯上的缺陷，而不探討加解密演算法的缺陷），但是我們發現用 session key 加密的 Item 有一種情況也是被用來確認身分的 Item，這種情況是當某個參與者收到資料取得 session key 之後，會用 session key 加密一個亂數或是其他資料傳送給對方，以此來確認對方真的擁有這個 session key，所以這種情況的 Item 也要記錄下來。

- 第二個是找出沒有出現在下一個 Message 中有加密而且加密的 key 不是 session key 的 Item。（也就是 Produce 的情況）：
- 第三個是找出有出現在下一個 Message 中有加密的 Item，而且下一個 Message 中包含這個 Item 的 Item 是**攻擊的目標 Item(重要的 Item)**。（也就是 Produce 但卻被接收的參與者做 Forward 的動作的情況下）

3.2.3 尋找可以利用的資源

在上一小節中我們介紹了如何尋找要**攻擊的目標 Item(重要的 Item)**，接下來在這一小節中，我們會說明如何尋找**可以利用的資源**。

在 3.1 節我們已經介紹過何謂**可以利用的資源**。根據 3.1 節的說明，我相信大部分的人的第一個想法是**可以利用的資源**就是認證協定的每一個 Item，因爲在網路上任何人（包括 Attacker）都可以攔截下任何一筆資料。當然認證協定的每一個 Item 都是**可以利用的資源**的這個想法沒錯，但是我們不僅會問這就是所有**可以利用的資源**嗎？因此我們希望能夠根據這些已知的**可以利用的資源**想辦法去得到更多更有用處的資源。接下來我們呈現了我們用來得到更多更有用的**可以利用的資源**的方法。

```
void pieceTogether( Message protocol, Item important )
{
    Item meaning = NULL;
```

```
Item unmeaning = NULL;
Boolean flag = true;
Message protocol_temp = protocol;

while( protocol_temp != null )
{
    Item item_temp = protocol_temp. item;
    while(item_temp != NULL )
    {
        if(item_temp is ciphertext )
            copy the item_temp to unmeaning;
//copy 代表將這個 item 複製到 unmeaning(不包含 Structure 裡的 next 欄位)。
        else
            copy the item_temp to meaning;
//copy 代表將這個 item 複製到 meaning(不包含 Structure 裡的 next 欄位)。
        item_temp = item_temp.next;
    }
    protocol_temp = protocol_temp.next;
}

while( flag == true )
{
    flag = false;
    Item unmeaning_temp = unmeaning;
    while( unmeaning_temp != NULL )
    {
```

```
                if( meaning include unmeaning_temp.key )
                {
                    copy the unmeaning_temp.include 中沒有加密的 item to
meaning;
                    copy the unmeaning_temp to meaning;
            //copy 代表將這個 item 複製到 meaning(不包含 Structure 裡的 next 欄位)。
                    remove the unmeaning_temp from unmeaning;
                    flag = true;
                }
                else
                {
                    Item hold = unmeaning_temp.include;
                    if( include(meaning, hold) )
                    {
                        copy the unmeaning_temp to meaning;
            //copy 代表將這個 item 複製到 meaning(不包含 Structure 裡的 next 欄位)。
                        remove the unmeaning_temp from unmeaning;
                        flag = true;
                    }
                }
                unmeaning_temp = unmeaning_temp.next;
            }
        }
}
```

pieceTogether function 的用意是要根據已知的**可以利用的資源**來找出更多更

有用處的資源。pieceTogether function 一開始會先將認證協定所有的 Item 分成兩類，一類是**“有意義的 Item”**，儲存沒有加密的 Item；另一類是**“沒有意義的 Item”**，儲存有加密的 Item。接下來的 while loop 是根據**“有意義的 Item”**爲已知的知識來判斷**“沒有意義的 Item”**中是否有 Item 可以從**“沒有意義的 Item”**變成**“有意義的 Item”**。

因爲一開始**“沒有意義的 Item”**裡面儲存的都是有加密的 Item，因此首先我們先判斷**“沒有意義的 Item**”裡面的每一筆資料中用來加密的 key 是否是**“有意義的 Item”**裡面的資料，如果是的話，就將目前分析的 Item 從**“沒有意義的 Item”**裡面移除，並同時將自己還有被自己加密的所有 Item 中沒有被加密的 Item 也都加到**“有意義的 Item”**裡面，而有加密的 Item 儲存到**“沒有意義的 Item”**裡面；如果不是的話，利用 **3.2.2** 節裡面介紹的 include function 來找出**“有意義的 Item”**中是否包含目前分析的 Item（這裡包含的意思是指目前分析的 Item 裡面所有的資料都要出現在**“有意義的 Item”**裡面），如果有包含就將目前分析的 Item 加到**“有意義的 Item”**並同時從**“沒有意義的 Item”**裡面移除，一直到把**“沒有意義的 Item”**裡面的所有資料都依據這方法分析完畢。如果分析完畢有**“沒有意義的 Item”**變成**“有意義的 Item”**那麼就要重新再分析一次，如果分析完畢沒有 Item 從**“沒有意義的 Item”**變成**“有意義的 Item”**那麼這個 while loop 就結束。這時候**“有意義的 Item”**就是我們依據已知的**可以利用的資源**找出更多更有用處的資源。

上面我們詳細的介紹了尋找出更多更有用處的**可以利用的資源**的流程，接下來我們開始說明爲什麼要這樣判斷。

在 pieceTogether function 中爲什麼一開始我們要先將認證協定中所有的 Item 分成兩類：第一類是沒有加密的 Item、第二類是有加密的 Item。

- 因爲第一類的 Item 是沒有加密過的所以 Attacker 可以對這些 Item 做任意的修改，甚至隨便拿一個 Item 來僞裝，換個角度講，這也是代表著 Attacker 可以隨意的控制這一類的 Item，而第二類的 Item 因爲是有加密過的，再加上 Attacker 不知道用來加密的 key，所以 Attacker 是無法控制住這一類的 Item。
- 將所有的 Item 分成兩類之後，接下來是把第一類的 Item 當成已知的知識來分析第二類的 Item。假設是第一次分析，那麼第一類的 Item 就是沒有加密的 Item，第二類的 Item 就是有加密的 Item。假設有 Item 從第二類的 Item 變成第一類的 Item，如果這個 Item 是因爲它用來加密的 key 有出現在第一類的 Item 裡面所以才從第二類的 Item 變成第一類的 Item，那麼很明顯的，Attacker 知道用來加密的 key，所以 Attacker 可以隨意的來控制這個 Item；如果這個 Item 是因爲它被加密的資料都有出現在第一類的 Item 裡面所以才從第二類的 Item 變成第一類的 Item，那麼我們可以知道這個 Item 是利用第一類 Item 裡的資料拼湊組合出來的，所以只要 Attacker 可以湊齊必要的 Item 之後就可以利用別人製造出這個 Item 了，因此 Attacker 也可以控制這一個 Item。經由這樣不斷的更新第一類的 Item 和第二類的 Item，最後我們就可以找出所有 Attacker 可以控制的 Item 了。

在這一小節中，我們找到了兩組可以利用的資源，爲了方便之後說明，我們把攔截下來認證協定的所有 Item 稱爲攔截直接使用的資源，而將 pieceTogether function 分析出來的可利用資源稱爲可控制的資源。

3.2.4 尋找認證協定的漏洞

根據 **3.2.2** 和 **3.2.3** 這兩節，我們找到了要**攻擊的目標 Item(重要的 Item)**和**可以利用的資源**，因此接下來我們開始介紹我們利用這兩筆資料如何找出認證

協定的漏洞。

我們利用攔截直接使用的資源和可控制的資源分別來搭配**攻擊的目標 Item(重要的 Item)**來找出認證協定的漏洞。

- 攔截直接使用的資源和**攻擊的目標 Item(重要的 Item)**
- 可控制的資源和**攻擊的目標 Item(重要的 Item)**

```
void DirectAttack(Item item_temp, Item important)
{
    Item important_temp = important;
    Item observation = NULL;
    /*  找出所有資料數量跟重要的 Item 一樣的 Item */
    while(important _temp != NULL )
    {
        int numberOfImportantTempItem = important_temp.number;
        while(item_temp!= NULL )
        {
            if(item_temp is ciphertext )
            {
                int numberOfItemTemp = item_temp.number;
                if( numberOfImportantTempItem == numberOfItemTemp )
                    copy the item_temp to observation;
    //不在意 item 在 observation 中的順序
    //copy 代表將這個 item 複製到 observation(不包含 Structure 裡的 next 欄位)。
            }
            item_temp = item_temp.next;
        }
```

```
        important _temp = important _temp.next;
}

/*  根據 content 和 key 相不相同來判斷是否可以攻擊  */
while( observation != NULL )
{
        if( observation.content == important.content )
        { /*  內容一樣  */
                if( observation.key == important.key )
                { /* content 一樣、key 一樣  */
                        if( important.content doesn't include Nonce or Timestamp )
                        {
                                important.flag = true;
                                int loc = important.location;
                                if(  找出 important 中 location 大於等於 loc 的 item)
                                        if(  找到的所有 item 的 flag == true )
                                                attacker 直接 replay 這個 message 即可攻破。
                        }
                }
                else /*  內容一樣，key 不一樣  */
                {
                        Item observation_temp = observation.include;
                        important_temp = important.include;
                        Boolean flag = true;
                        while( observati on_temp != null && important_temp!= null )
                        {
```

if(include(item_temp, observation_temp) == false ||

include(item_temp, important_temp) == false)

//如果 observation_temp 和 important_temp 有 Kab 這個 Item，不用判斷 item_temp 是否包含他。

flag = false;

}

if(flag == true)

假設 A 跟 B 要溝通，如果今天找到的 important_item 是由 A 發出的，不管 observation_temp 這個 item 是 B 還是 S 發出的(因爲用來加密的 key 如果是 S 的 key 那麼沒人可以解開，所以加密的 key 一定是 B 的 key)，只要 Attacker 新開一個 Run，僞裝成 C(C is anyone)要跟 A 溝通，這一個新的 Run 可以找到 important_item 是由 C 發出的，而 observation_temp 是由 A 或 S 發出的，而且這個新的 Run 的 observation_temp 會是一開始 A 跟 B 要溝通的 Run 的 important_item。所以可以用多個 Run 攻破此 Protocol。

}

}

else /* 內容不一樣 */

{

if(observation.key == important.key)

{ /* 內容不一樣，key 一樣 */

if(important 和 observation 不一樣的 content == Kab)

attacker 利用其他資料當作 Kab 即可來攻破。

}

}

}

}

DirectAttack function 的用意是找出直接利用參數 item_temp 當成 important 的攻擊，important 就是 **3.2.2** 節找到的要**攻擊的目標 Item(重要的 Item)**。

DirectAttack function 裡面是由兩個 while loop 組成的。

- 第一個 while loop 主要是找出資料數量跟**攻擊的目標 Item(重要的 Item)**一樣的 Item。
- 第二個 while loop 是將第一個 while loop 裡找到的 資料數量跟**攻擊的目標 Item(重要的 Item)**一樣的 Item 拿來繼續跟**攻擊的目標 Item(重要的 Item)**分析，分析的方式是比較 content 和 key 這兩筆資料相同不相同，這樣的比較方式可以得出四種結果
  - 第一種結果是 content 一樣、key 也一樣：在這種情況下，將 important 目前指到的 Item 的 flag 欄位設成 True，如果目前分析的 Item 沒有包含 Nonce、Timestamp 這些用來確保 Fresh 機制的資料，而且如果 important 中所有 location 欄位值大於等於目前分析的 Item，而這些 Item 的 flag 欄位如果都是 True，那麼 Attacker 可以直接 replay 這個 Item 當成**攻擊的目標 Item(重要的 Item)**來攻破此認證協定。（附錄一有範例詳細的說明）
  - 第二種結果是 content 一樣、key 不一樣：首先在這邊 content 一樣，除了內容都是一樣的情況下，還包括了格式一樣(都是 Nonce、Timestamp、identity)，譬如像 Na、Nb 不一樣可是他們的格式都是 Nonce，所以也算 content 一樣。另外在這種情況下，如果目前分析的 Item 和目前分析的重要的 Item 中被自己加密的資料有包含在 DirectAttack function 第一個參數的 Structure 中的話，那麼 Attacker 可以使用多個 Run 來攻破此認證協定。（附錄二有範例詳細的說明）
  - 第三種結果是 content 不一樣、key 一樣：在這種情況下，兩個 Item 的 content 不一樣，如果不一樣的 content 中就是 session key 的話，那麼

Attacker 可以 replay 這個 Item 當成**攻擊的目標 Item(重要的 Item)**來攻破此認證協定。（附錄三有範例詳細的說明）

- 第四種結果是 content 不一樣、key 也不一樣：在這種情況下，Attacker 無法攻破此認證協定。

上面我們詳細的介紹了找出認證協定漏洞的流程，接下來我們會說明爲什麼要這樣判斷。

首先先找出所有資料數量跟**攻擊的目標 Item(重要的 Item)**一樣的 Item，因爲只有資料數量一樣這個 Item 才有機會讓 Attacker 能拿來當成**攻擊的目標 Item(重要的 Item)**來攻破此認證協定。

接下來我們利用 content 和 key 這兩筆資料將找到資料數量和**攻擊的目標 Item(重要的 Item)**一樣的 Item 和**攻擊的目標 Item(重要的 Item)**來比較並分析出四種結果。接下來我們一一的來解釋這四種結果爲什麼可以攻破，爲什麼不可以攻破。

- 第一種結果 content 一樣、key 也一樣：在這種情況下，我們可以知道目前分析的 Item 跟**攻擊的目標 Item(重要的 Item)**其實可以說是同一個，所以只要分析這個 Item 是否包含 Nonce、Timestamp 這些用來確保 Fresh 的機制。如果這個 Item 沒有包含 Fresh 機制的資料，還不能說一定可以攻擊，因爲後面的 Item 可能也會有一些防護的機制，所以還要判斷 location 值大於等於目前這個 Item 的所有 Item 的 flag 是否都是 True 的話就可以知道 Attacker 是否能直接 replay 來攻破此認證協定。（附錄一有範例詳細的說明）
- 第二種結果 content 一樣、key 不一樣：首先先看這兩個 content 一樣的 Item 中，被他們加密的資料是否有出現在可以利用的資源中（當然如果加密的資料中有 session key 的話，當然就不尋找 session key 是否出現在可以利用的資源中），如果都有出現在可以利用的資源中，那麼就可以攻擊此認證協定。例如：假設 A 跟 B 要溝通，如果今天找到的**攻擊的目標 Item(重要的 Item)**

是由 A 發出的，不管目前分析的 item 是由 B 還是 S 發出的（因爲用來加密的 key 如果是 S 的 key 那麼沒人可以解開，所以加密的 key 一定是 B 的 key）。只要 Attacker 新開一個 Run，僞裝成 C（C is anyone）要跟 A 溝通，這一個新的 Run 可以找到**攻擊的目標 Item(重要的 Item)**是由 C 發出的，而目前分析的 Item 是由 A 或 S 發出的，而這個新的 Run 目前分析的 Item 會是一開始 A 跟 B 要溝通的 Run 的**攻擊的目標 Item(重要的 Item)**。所以可以用多個 Run 攻破此 Protocol。（附錄二有範例詳細的說明）

- 第三種結果 content 不一樣、key 一樣：目前分析的 Item 跟**攻擊的目標 Item(重要的 Item)**中 content 不一樣，而**攻擊的目標 Item(重要的 Item)**跟目前分析的 Item 不一樣的資料中如果剛好是 session key，那麼 Attacker 只要 replay 目前分析的 Item 就可以將目前分析的 Item 裡面某筆資料當成 session key 來攻破此認證協定。（附錄三有範例詳細的說明）
- 第四種結果 content 不一樣、key 也不一樣：很明顯的 Attacker 找不到可以當成**攻擊的目標 Item(重要的 Item)**的 Item 來攻破此認證協定，所以在這種情況下，Attacker 是無法攻破此認證協定。

最後還要判斷所有被認爲是**攻擊的目標 Item(重要的 Item)**的 Item 是否皆出現在可控制的資源中。如果有，那麼就代表 Attacker 可以控制這些被認爲是**攻擊的目標 Item(重要的 Item)**，也因此很明顯的，Attacker 自然可以攻破此認證協定。另外除了判斷是否出現在可控制的資源之外，還會判斷目前分析的 Item 的上一個 Message 中所有 Item 是否都有出現在可控制的資源中(第一個 Message 例外)，有的話也代表 Attacker 可以控制此 Item。爲了方便之後的說明，我們稱這個判斷是找出 Message 彼此間關係的漏洞。

3.3 Proposed 驗證認證協定演算法的防護範圍

在這一小節中，我們會分析我們 Proposed 的驗證認證協定演算法可以找出哪些漏洞。

從 **3.1** 節可以知道我們設計的演算法是站在 Attacker 的角度上來思考如何攻擊任何一個認證協定。而思考後的結果就是要找出**攻擊的目標 Item(重要的 Item)**和所有**可以利用的資源**，並利用可以利用的資源來攻擊要**攻擊的目標 Item(重要的 Item)**。

另外我們在 **3.2.4** 節中分別是用攔截直接使用的資源和可控制的資源跟**攻擊的目標 Item(重要的 Item)**搭配來尋找認證協定的漏洞。

很明顯的 Attacker 直接拿攔截直接使用的資源去攻擊要**攻擊的目標 Item(重要的 Item)**，也就是說 Attacker 只要找出認證協定中他想要**攻擊的目標 Item(重要的 Item)** 就可以不耗任何力量試著去攻擊認證協定了。因此利用攔截直接使用的資源跟**攻擊的目標 Item(重要的 Item)**搭配分析出來的結果，可以找出 Attacker 不需做任何事，而直接利用攔截直接使用的資源來攻擊認證協定的漏洞（包含 Freshness attack、Type flaw attack、Man-in-the-middle attack）。

而利用可控制的資源搭配**攻擊的目標 Item(重要的 Item)**來尋找認證協定的漏洞可以找到哪些類型的漏洞呢？首先在 **3.2.3** 節的最後我們提到了利用這個方法經由不斷的更新 **“有意義的 Item”** 和 **“沒有意義的 Item”**，我們可以找出所有 Attacker 可以控制的 Item。每一次的更新都是經由舊的 **“有意義的 Item”** 當已知的知識分析後所得到的，換句話說得到的這個 Item 可以經由舊的 **“有意義的 Item”** 組合起來，那也可以說是 Attacker 只要多一個 Run 就可以製造並得到

這個 Item。因此可控制的資源會是 Attacker 經由一個或一個以上的 Run 就可以取得的 Item。所以利用可控制的資源搭配**攻擊的目標 Item(重要的 Item)**來尋找認證協定的漏洞可以找出一個或一個以上的 Run 所取得的 Item 直接來攻擊認證協定的漏洞（同樣也包含 Freshness attack、Type flaw attack、Man-in-the-middle attack）。

另外我們在 **3.2.4** 節的最後一段說明了找出 Message 彼此間關係的漏洞的方法，前面提到的兩個方法都是判斷已經找到或是已經找好路徑的 Item 來攻擊的，而這個方法是先假設 Attacker 不知道路徑去得到**攻擊的目標 Item(重要的 Item)** 的所有 Item，然後利用可控制的資源來判斷是否可以找到路徑去取得這些 Item。

前面三段說明利用攔截直接使用的資源和可控制的資源分別來搭配**攻擊的目標 Item(重要的 Item)**還有 Message 彼此間的關係所能找出認證協定的漏洞的範圍。這三個方法找到的漏洞的範圍是有重疊的部份，首先第三個方法可能會找出已經被前兩個方法找到的路徑，但是經由判斷 Message 彼此間的關係卻能找出前兩個方法沒找到的攻擊路徑。另外第一個方法並不是第二個方法的特例，因爲攔截直接使用的資源不一定會是可控制的資源。如圖 3.2 就是他們的關係圖。

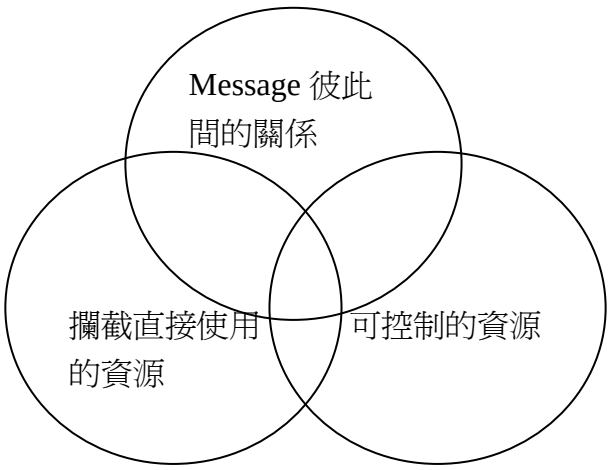


圖 3.2: 三種方法尋找到的漏洞範圍

第四章 系統介面與模擬結果

在這一章節，我們使用 Woo and Lam Authentication Protocol 當範例來說明我們這個系統的程式執行流程和系統分析模擬後的結果。

4.1 系統執行流程

我們以 Borland JBuilder 2006 爲開發此系統的平台。接著說明我們系統的執行流程。

1. 首先會先顯示出一個方便簡易的使用者介面(圖 4.1)。
2. 使用 MenuItem 或是 Button 來選擇要手動輸入認證協定或是讀取檔案裡的認證協定(圖 4.2~圖 4.5)
   2.1 選擇手動輸入，那麼會顯示一個簡單的輸入介面，並且顯示要輸入的認證協定格式(圖 4.6)。
   2.2 如果選擇讀取檔案裡的認證協定，那麼會顯示視窗讓使用者選擇檔案(圖 4.7)，當使用者輸入的檔案經分析後，發現認證協定格式有誤，那麼會將讀取的資料顯示在圖 4.6 的介面中讓使用者修改讀取到的認證協定，以便符合格式。
3. 當輸入正確的認證協定格式後，程式介面會顯示出使用者輸入的認證協定，並且根據輸入的認證協定來改變程式介面的 Agent 名稱(圖 4.8)。當輸入完畢後，可選擇模擬認證協定流程或模擬攻擊者行爲來找出認證協定的漏洞。
   3.1 選擇模擬認證協定流程時，會根據認證協定的各個 Message 而在相對應的 Agent 區域中顯示資訊，另外還可選擇上一步和下一步來觀看認證協定的流程(圖 4.9、圖 4.10)。
   3.2 選擇模擬攻擊者行爲時，會開始對輸入的認證協定進行分析。並且將分析後的結果顯示在 Hacker 區域裡(圖 4.11、圖 4.12)。結果會顯示 DirectAttack、pieceTogether 和 Relationship 這三類的訊息。DirectAttack 是圖 3.2 左邊的圓，pieceTogether 是右邊的圓，Relationship 是上方的圓。

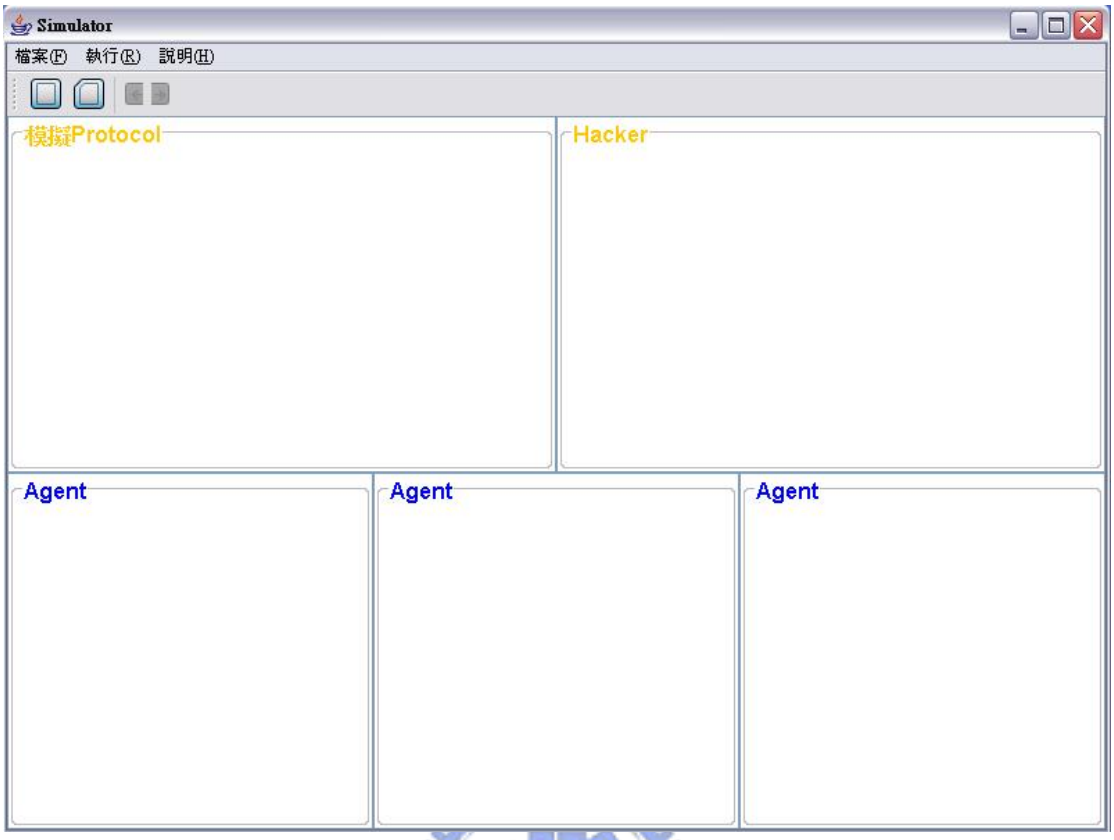


圖 4.1 程式初始介面

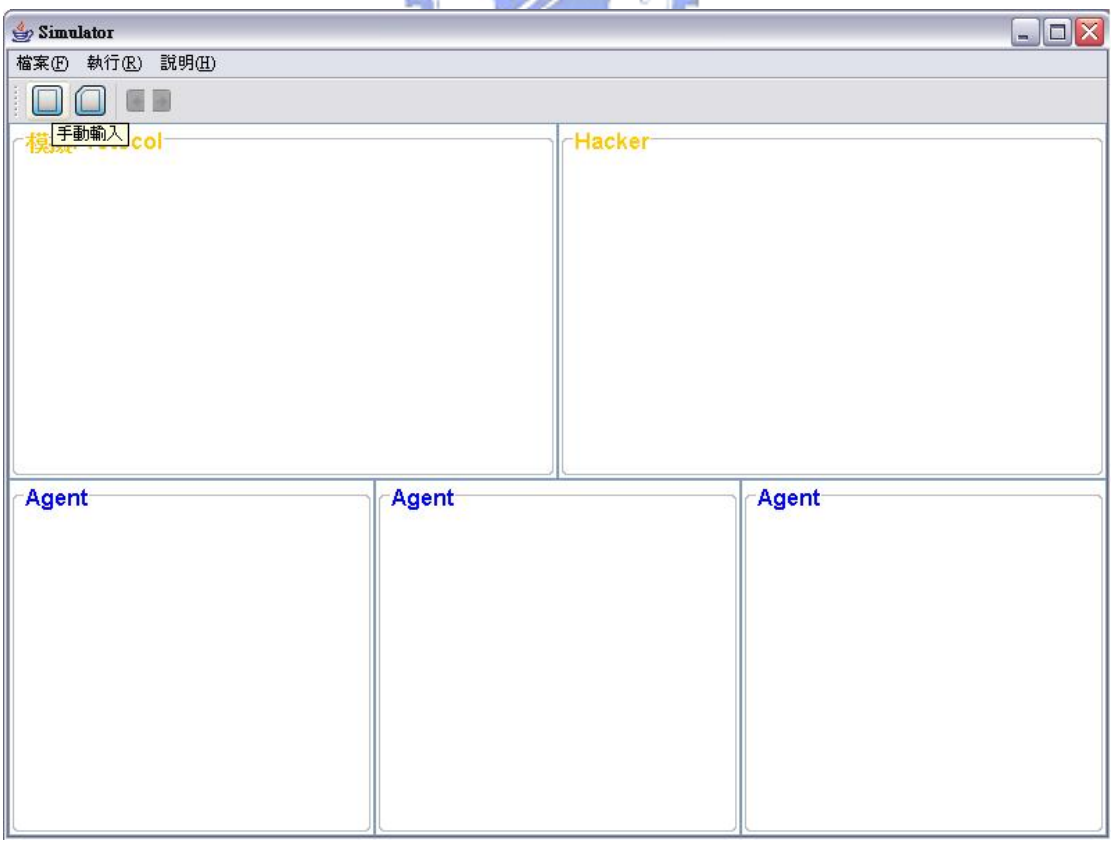


圖 4.2 程式中手動輸入認證協定

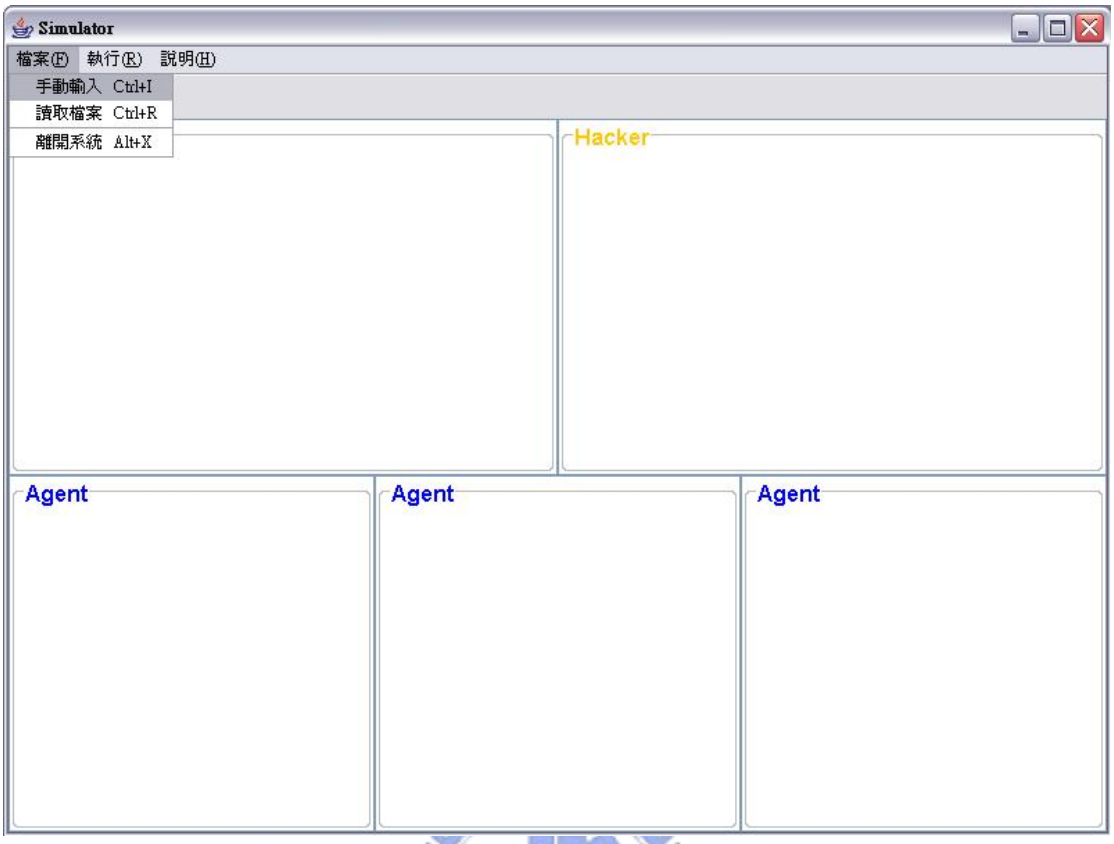


圖 4.3 程式中手動輸入認證協定

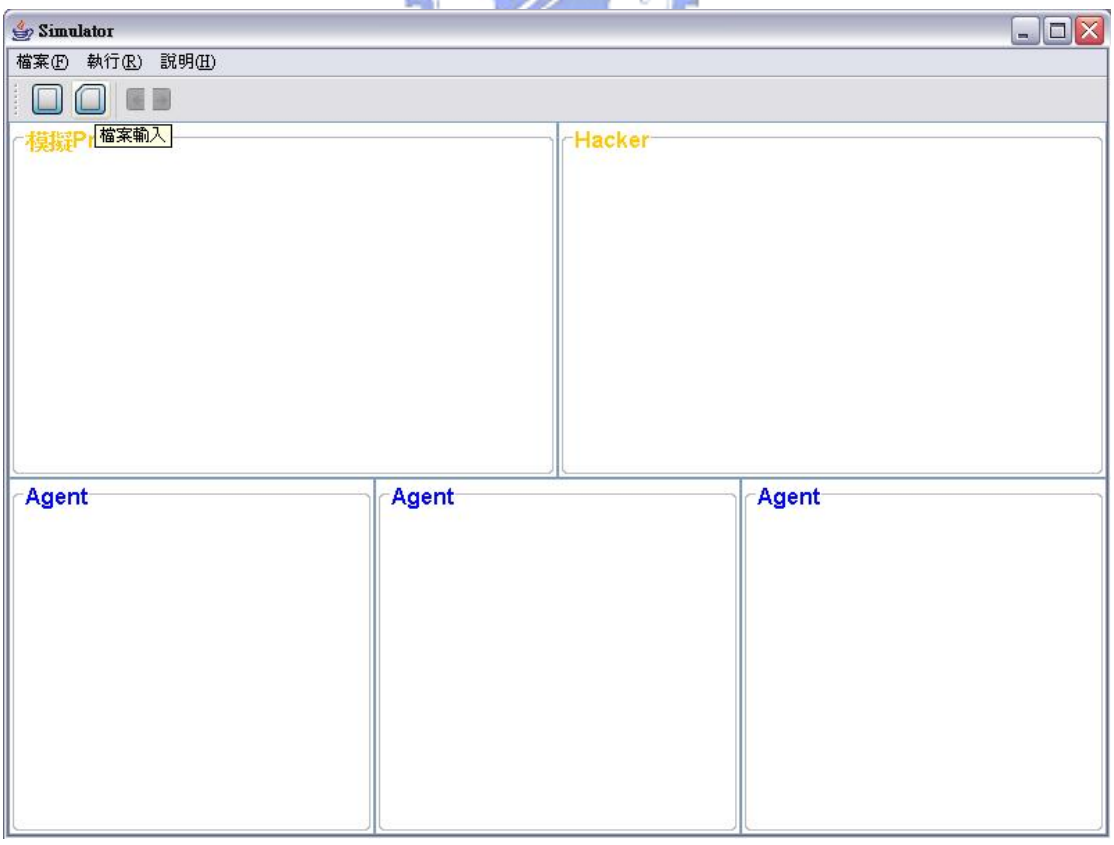


圖 4.4 程式中檔案輸入認證協定

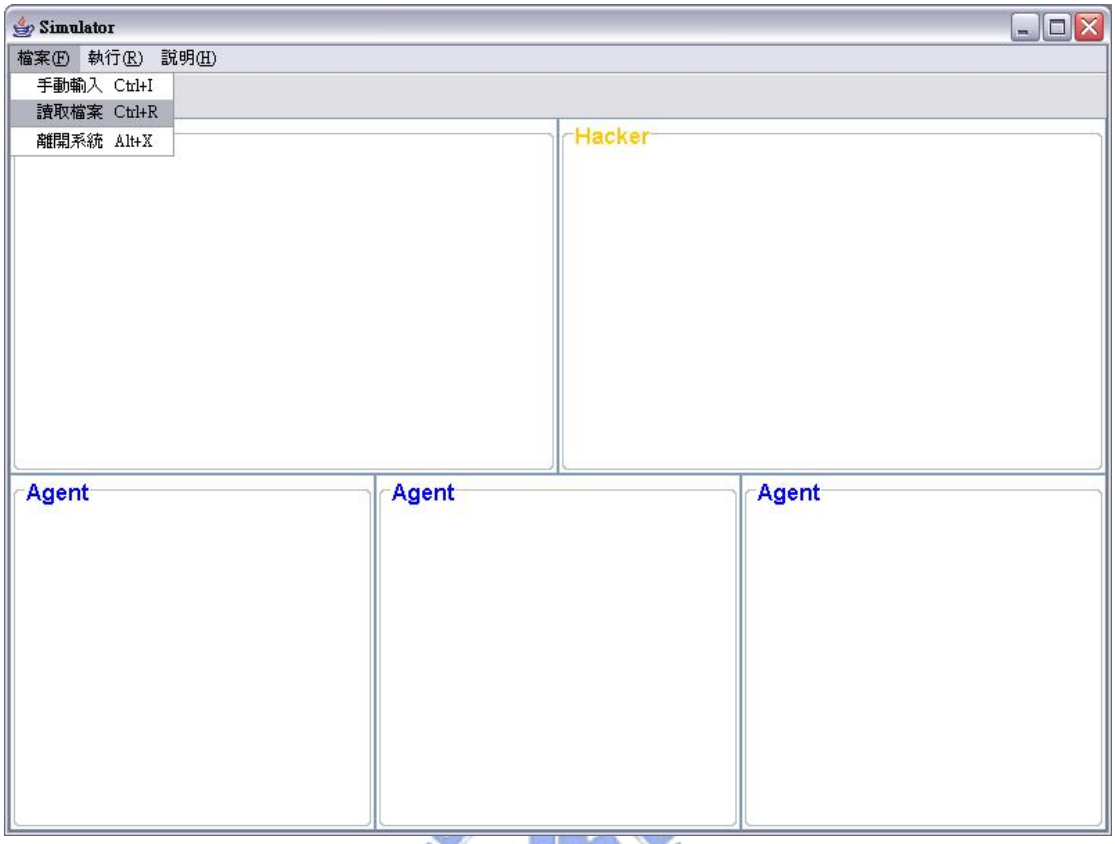


圖 4.5 程式中檔案輸入認證協定

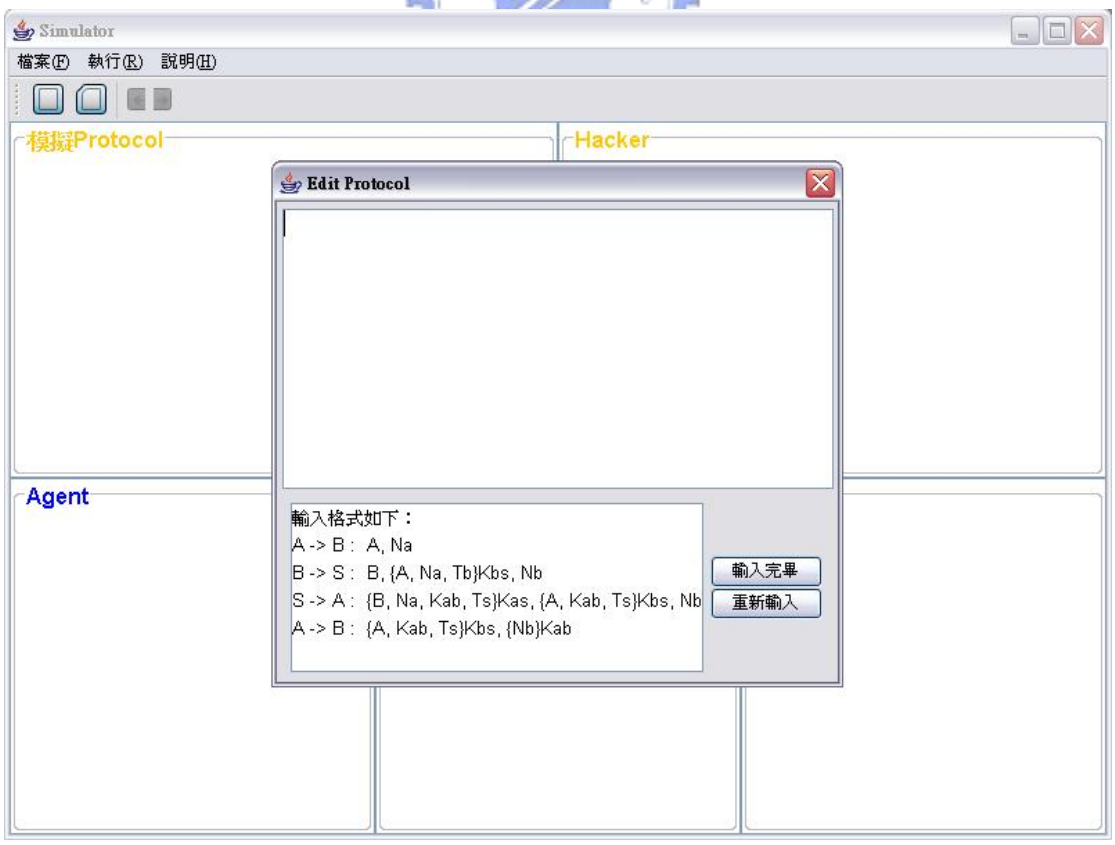


圖 4.6 手動輸入認證協定介面

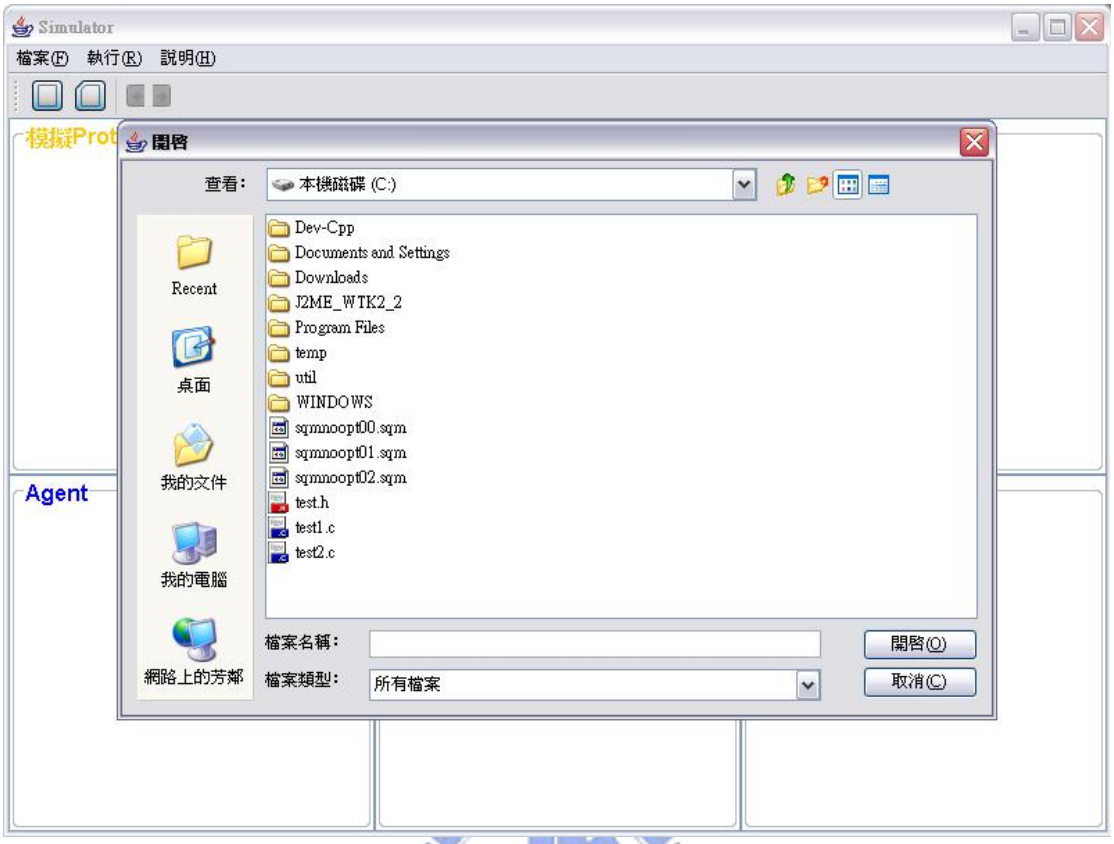


圖 4.7 選擇儲存有認證協定的檔案

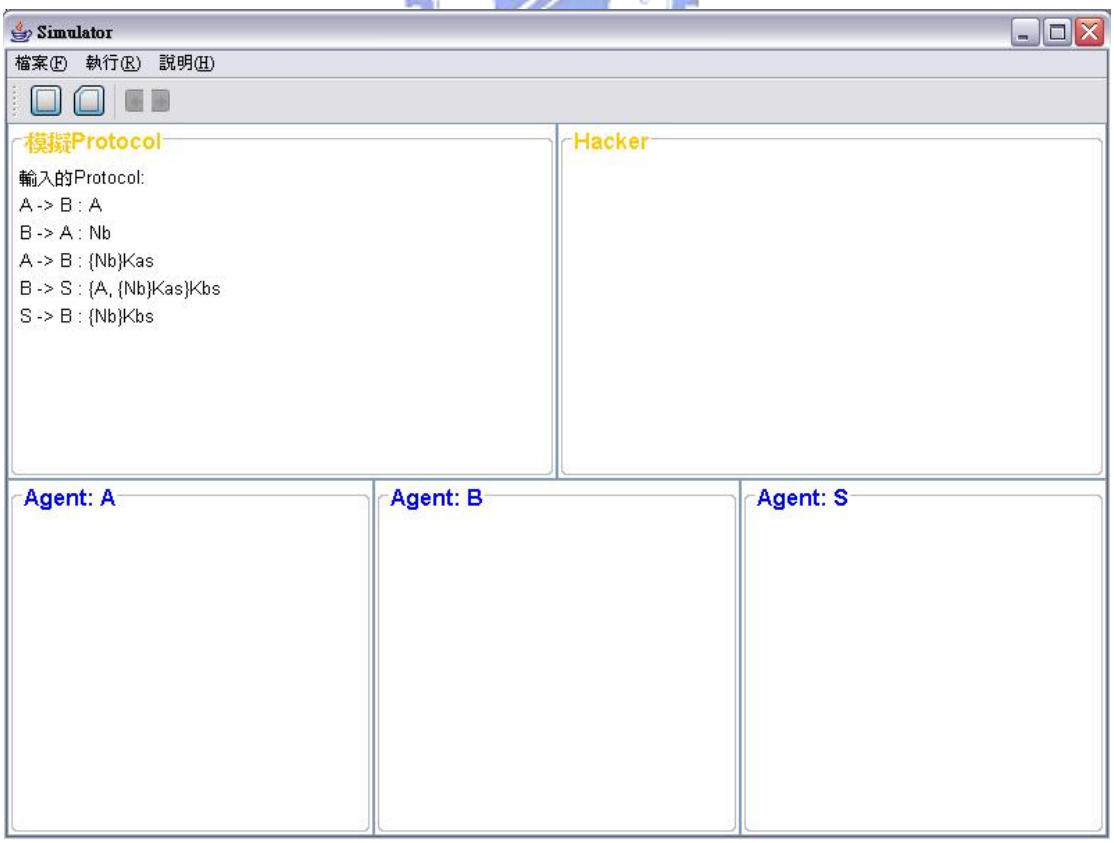


圖 4.8 輸入認證協定後的介面

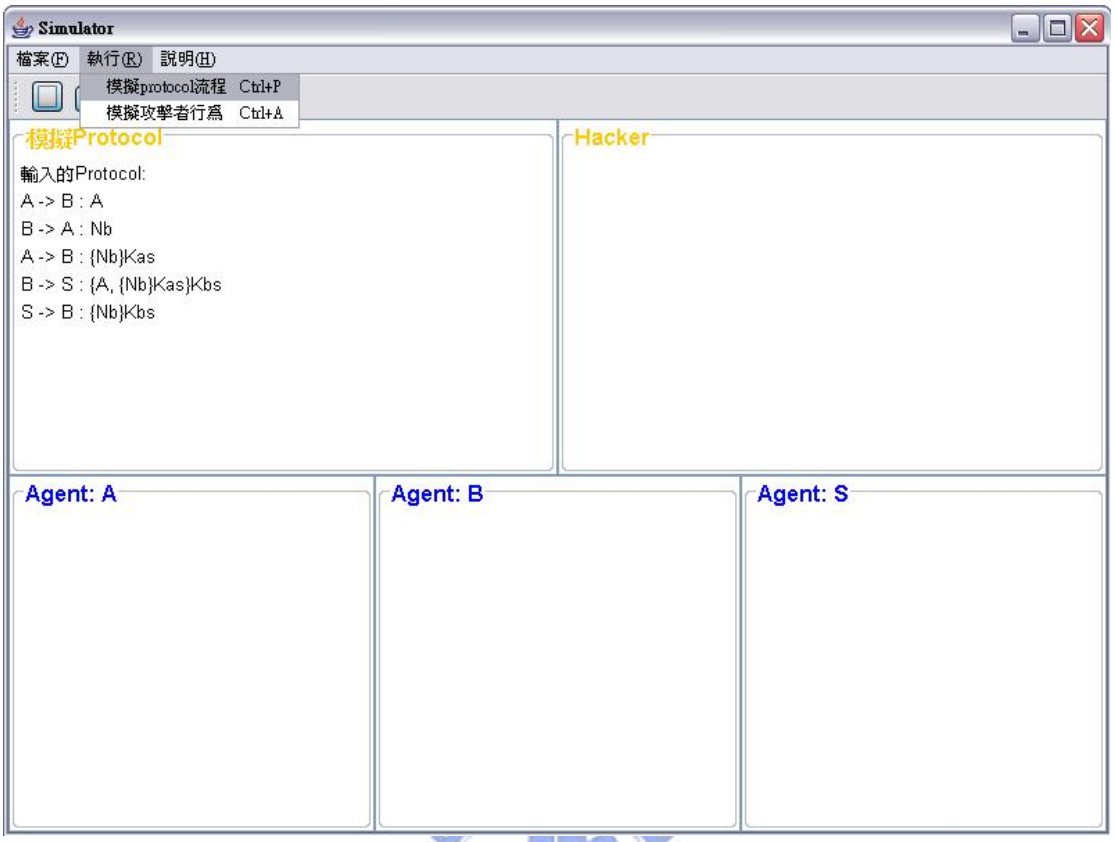


圖 4.9 選擇模擬認證協定流程

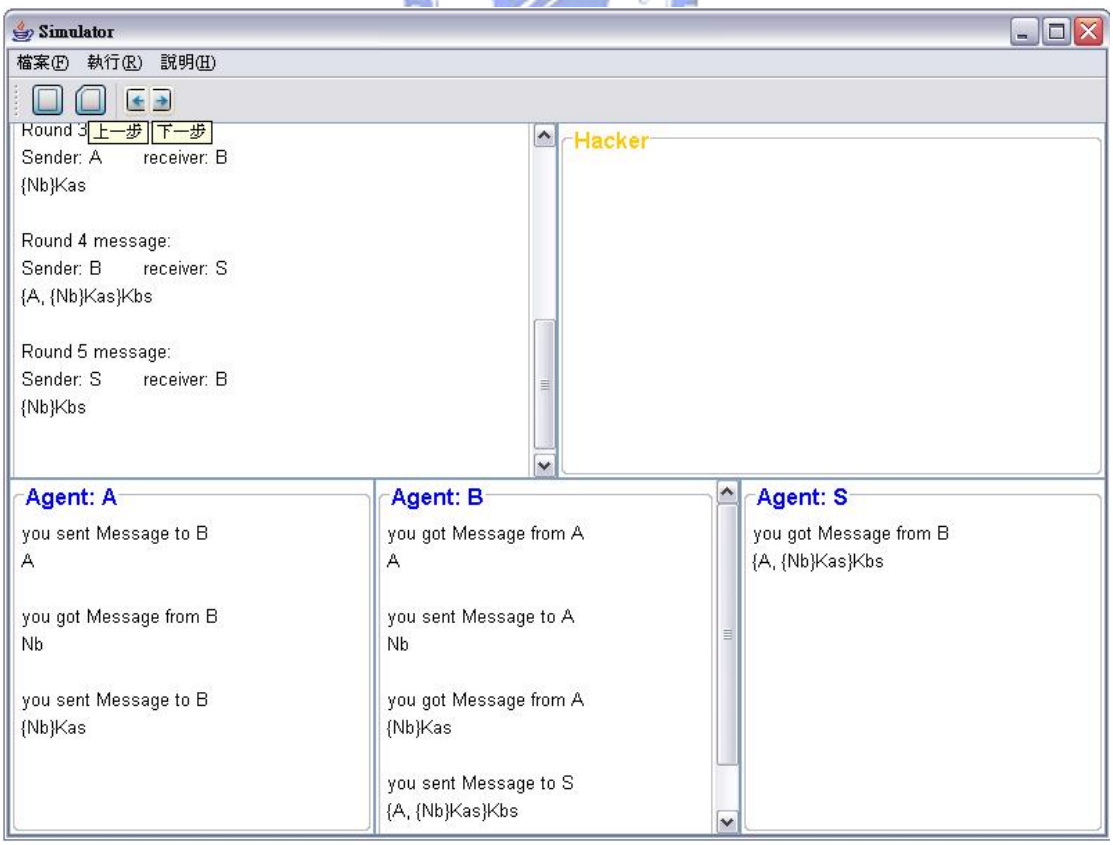


圖 4.10 一步一步的觀看認證協定流程

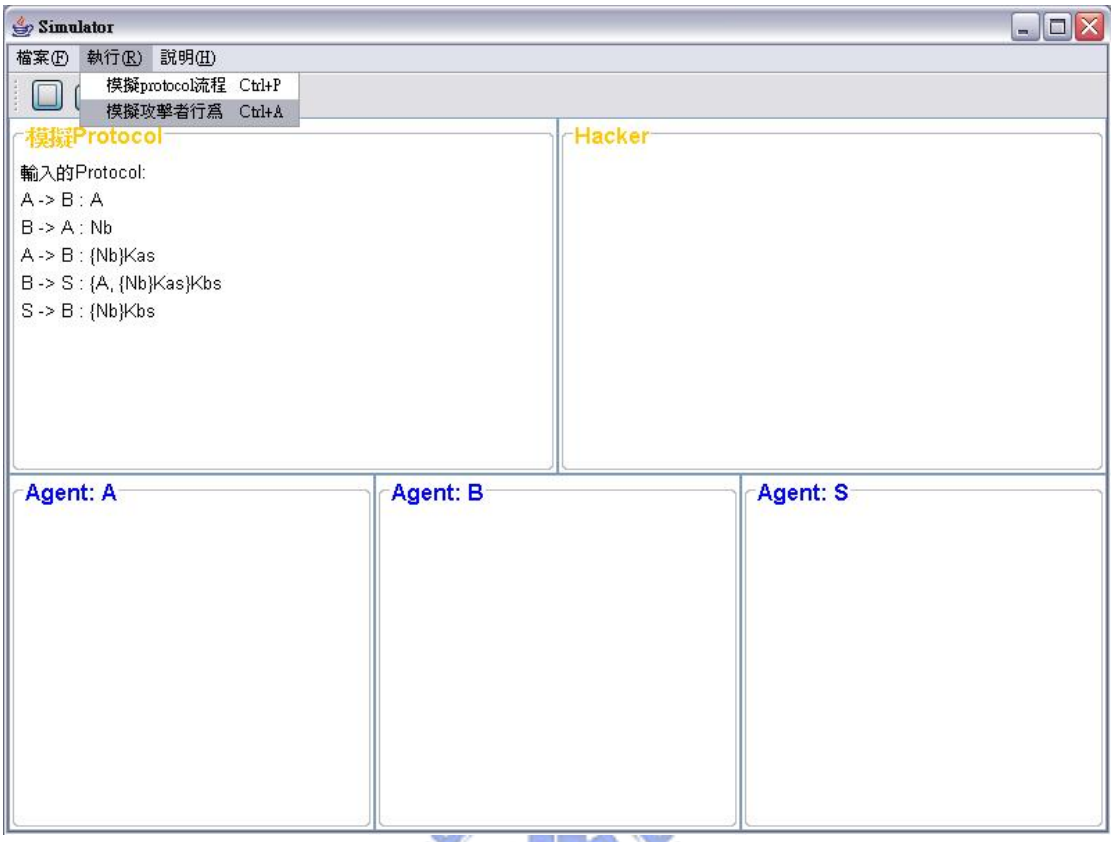


圖 4.11 選擇模擬攻擊者行爲找出認證協定漏洞

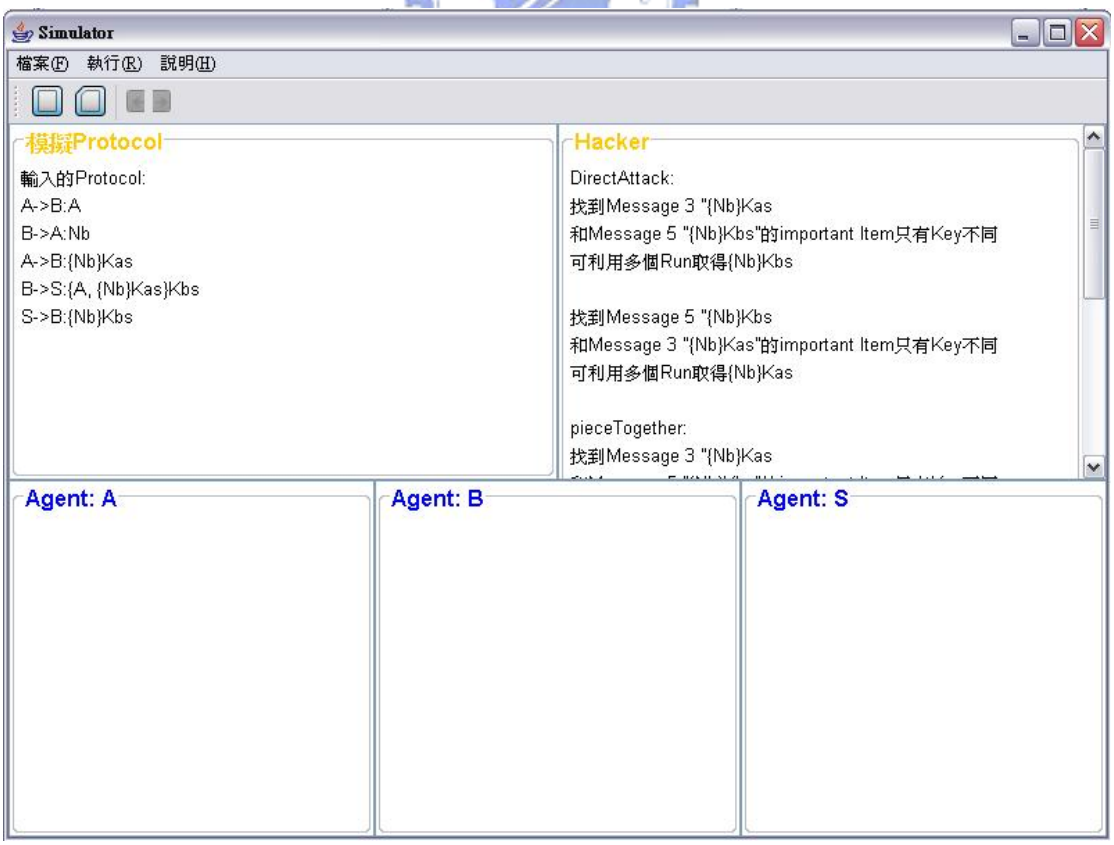


圖 4.12 系統偵測出認證協定的漏洞

4.2 系統分析後的結果

在 4.1 節中我們以圖解說明了我們這個系統分析 Woo and Lam Authentication Protocol 的執行過程。因此在這一小節中會探討我們系統分析後得到的資料究竟有什麼功用？

在圖 4.12 的 Hacker 區塊中有三類資訊。分別是 DirectAttack 的訊息、pieceTogether 的訊息和 Relationship 的訊息。在這個例子中，DirectAttack 和 pieceTogether 的訊息是一樣。

- DirectAttack 的訊息：

1. 第一個訊息說明系統找到第三個 Messgae 的{Nb}Kas 和第五個 Message 的{Nb}Kbs 只有加密的 key 不一樣，所以可以經由多個 Run 來取得 {Nb}Kbs。

流程如下：

1.1 I(A) -> B : A

1.2 B -> I(A) : Nb

1.3 I(A) -> B : {Nb}Kas

2.1 B -> I(C) : B

2.2 I(C) -> B : Nb

2.3 B -> I(C) : {Nb}Kbs

1.4 B -> I(S) : {A, {Nb}Kas}Kb

1.5 I(S) -> B : {Nb}Kbs

首先 Attack 僞裝成 A 說要跟 B 溝通，經由 1.1~1.3 這三個 Message 之後，Attack 可以拿到 1.2 裡 B 產生的 Nb 資料。

接著當 B 要跟其他人溝通的時候，Attacker 將 2.1 的資料攔截下來，然後僞裝成 B 要溝通的人傳送在 1.2 裡攔截下來的 Nb 給 B，接著 Attacker 將

B 在 2.3 裡要傳送給對方的{Nb}Kbs 攔截下來，這時候 Attacker 就已經成功的取得了{Nb}Kbs 了。

最後 Attacker 將 1.4 裡 B 要傳送給受信任的第三者的資料攔截下來，接著僞裝成受信任的第三者將 2.3 拿到的{Nb}Kbs 傳送給 B，而這時候 Attacker 也就成功的僞裝成 A 來跟 B 溝通了。

2. 第二個訊息是說找到第五個 Message 的{Nb}Kbs 和第三個 Message 的{Nb}Kas 只有加密的 key 不一樣，所以可以經由多個 Run 來取得{Nb}Kas。經由多個 Run 去取得{Nb}Kas 之後，認證協定也是會繼續執行到第五個 Message。另外因爲 Nb 是隨機的，所以第五個 Message 的{Nb}Kbs 是無法利用其他 Run 取得第三個 Message 的{Nb}Kas，況且既然是同一個 Run，Attacker 早在攔截下第三個 Message 的時候就已經成功取得{Nb}Kas 了。因此拿第五個 Message 的{Nb}Kbs 去取得第三個 Message 的{Nb}Kas 是一件非常沒有意義的事情，所以這段訊息就不加以理會。

- pieceTogether 的訊息：

1. 第一個訊息說明系統找到第三個 Messgae 的{Nb}Kas 和第五個 Message 的{Nb}Kbs 只有加密的 key 不一樣，所以可以經由多個 Run 來取得{Nb}Kbs。在 **4.1** 節我們提到 pieceTogether 代表的是圖 3.2 右邊的圓，而這個圓代表的意義在**3.3**節我們也提到用來分析的 Item 可以經由一個或一個以上的 Run 取得。因此第三個 Message 的{Nb}Kas 可以找出一個 Run 或一個 Run 以上來取得，當拿到之後再經由多個 Run 來成功取得第五個 Message 的{Nb}Kbs。

流程如下：

1.1 I(M) -> B : M　　M 代表任何人

1.2 B - > I(M) : Nb

1.3 I(M) -> B : anything

1.4 B -> I(S) : {m, anything}Kbs

2.1 N -> I(O) : N　　N, O 代表任何人

2.2 I(O) -> N : Nb

2.3 N -> I(O) : {Nb}Kns

3.1 I(N) -> B : N　　N 指的是 2.1 的 N

3.2 B -> I(N) : Nb'

3.3 I(N) -> B : {Nb}Kns

3.4 B -> I(S) : {N, {Nb}Kns}Kbs

3.5 S -> I(B) : {Nb}Kbs

1.5 I(S) -> B : {Nb}Kbs

首先假設 Attacker 要攻擊 B，一開始 Attacker 僞裝成任何一個人跟 B 溝通，當拿到 B 傳送過來的 Nb 後。接著只要有其他人(N)要跟別人(O)溝通的時候，就僞裝成別人(O)將 B 的 Nb 傳送回去藉此拿到{Nb}Kns。

接著 Attacker 僞裝成 N 跟 B 溝通，並且將 2.3 拿到的{Nb}Kns 在 3.3 的時候傳送給 B 一樣藉此拿到{Nb}Kbs。

最後將 3.5 拿到的{Nb}Kbs 回到原本的 Run 也就是 1.5 的時候傳送給 B，而此時 Attacker 也就成功的僞裝成他想僞裝的人來跟 B 溝通了。

在這個例子中，其實 2.1~2.3 和 3.1~3.5，Attacker 就已經可以成功的僞裝成 N 來跟 B 溝通了，可是因爲我們 pieceTogether 找出的是一個 Run 或一

個 Run 以上，所以才可以再加上 1.1~1.5 這樣多一個 Run 的攻擊方式。當然如果要在多加一個 Run 甚至多個 Run 也是可以的。

在[7]一文中，也有提到類似這樣的攻擊方法，不過他們找到的攻擊方法總共需要五個 Run 才可以攻破此認證協定，而我們的這個方法找到了只需要三個 Run 就可以攻破的攻擊流程。

2. 第二個訊息是說找到第五個 Message 的{Nb}Kbs 和第三個 Message 的{Nb}Kas 只有加密的 key 不一樣，所以可以經由多個 Run 來取得{Nb}Kas。經由多個 Run 去取得{Nb}Kas 之後，認證協定也是會繼續執行到第五個 Message。另外因爲 Nb 是隨機的，所以第五個 Message 的{Nb}Kbs 是無法利用其他 Run 取得第三個 Message 的{Nb}Kas，況且既然是同一個 Run，Attacker 早在攔截下第三個 Message 的時候就已經成功取得{Nb}Kas 了。因此拿第五個 Message 的{Nb}Kbs 去取得第三個 Message 的{Nb}Kas 是一件非常沒有意義的事情，所以這段訊息就不加以理會。

- Relationship 的訊息：
  1. 可以經由 Message 4 的資料取得 Message 5 的 "{Nb}Kbs" 這筆資料
  2. 可以控制 Message 5 的 "{Nb}Kbs" 這筆資料
  3. 可以經由 Message 3 的資料取得 Message 4 的 "{A, {Nb}Kas}Kbs" 這筆資料
  4. 可以控制 Message 4 的 "{A, {Nb}Kas}Kbs" 這筆資料
  5. 可以經由 Message 2 的資料取得 Message 3 的 "{Nb}Kas" 這筆資料
  6. 可以控制 Message 3 的 "{Nb}Kas" 這筆資料

從 2、4 和 6 我們可以知道這些資料都是可控制的資源，而從 1、3 和 5 可以知道經由上一個 Message 的所有 Item(因爲他們都是可控制的資源)可以取得該 Item。而且觀看認證協定我們可以發現，2、4 和 6 的這三個資料剛好是認證協定的第五、第四和第三個 Message，也就是說如果今天要攻擊第五個 Message 的"{Nb}Kas"，除了從攔截下來的所有 Item 和可控制的資源中找這個 Item 之外，只要取得第三個 Message 的"{Nb}Kbs"或第四個 Message 的"{A, {Nb}Kas}Kbs"就可以取得這個 Item 了，而這也就是 1、3 和 5 這三個訊息彼此間的關係。

4.3 分析其它認證協定

1. Needham-Schroeder protocol
   i. A -> S : A, B, Na
   ii. S -> A : {Na, B, Kab, {Kab, A}Kbs}Kas
   iii. A -> B : {Kab, A}Kbs
   iv. B -> A : {Nb}Kab
   v. A -> B : {Nb - 1}Kab

   利用我們的演算法分析後的結果如下：

   - DirectAttack :
     - 可以直接 replay Message 3 的"{Kab, A}Kbs"攻擊此 Protocol
       - 只要 Attacker 攔截下 Message 3 的"{Kab, A}Kbs"，之後不管什麼時候都可以冒充 A 來跟 B 溝通，而 B 都會認爲跟自己溝通的是 A。
   - pieceTogether :
     - 可以直接 replay Message 3 的"{Kab, A}Kbs" 攻擊此 Protocol
       - 同 DirectAttack 的說明。但是能用更多個 Run 來取得

Message 3 的"{Kab, A}Kbs"。

- Relationship :

  1. 可以控制 Message 3 的 "{Kab, A}Kbs" 這筆資料

  2. 可以經由 Message 1 的資料取得 Message 2 的"{Na, B, Kab, {Kab, A}Kbs}Kas"這筆資料

     - 從 2 我們可以發現，Attacker 可以隨意使用三個 Item 當成 Message 1 的 A、B 和 Na，來產生 Message 2 的"{Na, B, Kab, {Kab, A}Kbs}Kas"這筆資料。雖然這一筆資料是我們系統分析出來的**攻擊的目標 Item(重要的 Item)**，但是它不是可控制的資源之一。再加上 1 和 2 這兩點彼此沒有關係，因此即使我們僞裝成任何一個人送 A、B 和 Na 來取得 Message 2 的"{Na, B, Kab, {Kab, A}Kbs}Kas"也沒用，因爲我們沒有這個人的 key，所以無法攻擊此認證協定。

2. Neuman-Stubblebine protocol

   i. A -> B : A, Na

   ii. B -> S : B, {A, Na, Tb}kbs, Nb

   iii. S -> A : {B, Na, Kab, Tb}kas, {A, Kab, Tb}kbs, Nb

   iv. A -> B : {A, Kab, Tb}kbs, {Nb}Kab

利用我們的演算法分析後的結果如下：

- DirectAttack :

  - 可以 replay Message 2 的"{A, Na, Tb}kbs" 當成 Message 4 的 "{A, Kab, Tb}kbs" 來攻破此 Protocol

    - Attacker 攔截下 Message 2 的"{A, Na, Tb}kbs"這筆資料後，接著爲了不讓 A 收到訊息所以再把 Message 3 攔截下來，接著將 Message 2 的"{A, Na, Tb}kbs"這一個資料當

成 Message 4 的"{A, Kab, Tb}kbs"，當對方收到的時候就會認爲 Na 是 Kab，而此時{Nb}Kab 也就等於{Nb}Na。再加上 Na 是 plaintext，所以 Attacker 知道這個值。因此 Attacker 成功的攻破此認證協定。

- pieceTogether :
  - 沒找到
- Relationship :

  1. 可以控制 Message 4 的"{Nb}Kab"這筆資料
  2. 可以經由 Message 1 的資料取得 Message 2 的"{A, Na, Tb}kbs"這筆資料
     - 根據 1 和 2 可以發現這兩點彼此間沒有關係，所以無法找到其它攻擊路徑。

3. Otway-Rees protocol

   i. A -> B : Na, A, B, {Na, A, B}Kas

   ii. B -> S : Na, A, B, {Na, A, B}Kas, Nb, {Na, A, B}Kbs

   iii. S -> B : Na, {Na, Kab}Kas, {Nb, Kab}Kbs

   iv. B -> A : Na, {Na, Kab}Kas

利用我們的演算法分析後的結果如下：

- DirectAttack:
  - 找到 Message 3 的"{Na, Kab}Kas"和 Message 3 的"{Nb, Kab}Kbs"的 important Item 只有 Key 不同，可利用多個 Run 取得{Nb, Kab}Kbs
    - 詳細說明請參照附錄二
  - 找到 Message 3 的"{Nb, Kab}Kbs"和 Message 4 的"{Na, Kab}Kas"的 important Item 只有 Key 不同，可利用多個 Run 取

得{Na, Kab}Kas

- 詳細說明請參照附錄二

- 找到 Message 1 的"{Na, A, B}Kas"和 Message 2 的"{Na, A, B}Kbs"的 important Item 只有 Key 不同，可利用多個 Run 取得{Na, A, B}Kbs
  - 詳細說明請參照附錄二
- 找到 Message 2 的"{Na, A, B}Kbs"和 Message 2 的"{Na, A, B}Kas"的 important Item 只有 Key 不同，可利用多個 Run 取得{Na, A, B}Kas
  - 詳細說明請參照附錄二

- pieceTogether:
  - 找到 Message 3 的"{Na, Kab}Kas"和 Message 3 的"{Nb, Kab}Kbs"的 important Item 只有 Key 不同，可利用多個 Run 取得{Nb, Kab}Kbs
    - 詳細說明請參照附錄二
  - 找到 Message 3 的"{Nb, Kab}Kbs"和 Message 4 的"{Na, Kab}Kas"的 important Item 只有 Key 不同，可利用多個 Run 取得{Na, Kab}Kas
    - 詳細說明請參照附錄二
  - 找到 Message 1 的"{Na, A, B}Kas"和 Message 2 的"{Na, A, B}Kbs"的 important Item 只有 Key 不同，可利用多個 Run 取得{Na, A, B}Kbs
    - 詳細說明請參照附錄二
  - 找到 Message 2 的"{Na, A, B}Kbs"和 Message 2 的"{Na, A, B}Kas"的 important Item 只有 Key 不同，可利用多個 Run 取得{Na, A, B}Kas

- 詳細說明請參照附錄二

- Relationship:

  1. 可以經由 Message 3 的資料取得 Message 4 的"{Na, Kab}Kas"這筆資料
  2. 可以控制 Message 4 的"{Na, Kab}Kas"這筆資料
  3. 可以經由 Message 2 的資料取得 Message 3 的"{Nb, Kab}Kbs"這筆資料
  4. 可以控制 Message 3 的"{Nb, Kab}Kbs"這筆資料
  5. 可以經由 Message 1 的資料取得 Message 2 的"{Na, A, B}Kas"這筆資料
  6. 可以控制 Message 2 的"{Na, A, B}Kas"這筆資料
  7. 可以經由 Message 1 的資料取得 Message 2 的"{Na, A, B}Kbs"這筆資料
  8. 可以控制 Message 2 的"{Na, A, B}Kbs"這筆資料

  - 由 1 到 8，我們可以發現四個 Message 之間彼此都有關聯性，再加上記錄在**攻擊的目標 Item(重要的 Item)**的所有 Item 皆可控制，因此如果我們要攻擊 Message 4 的{Na, Kab}Kas，因爲這個 Item 是可控制的資源，因此除了原本已經找到的攻擊路徑之外，還可從 Message 1 來取得 Message 2 的{Na, A, B}Kas 和{Na, A, B}Kbs（也就是上面的 5 和 7 兩點）之後按照認證協定的流程一樣可以取得 Message 4 的{Na, Kab}Kas。當然也可從 Message 2 和 Message 3 來取得。因此可以找出更多樣化的攻擊路徑。

4. 我設計的認證協定(我認爲它是安全的) :

   i. A -> S : A, {A, Ta, B, Na}Kas

ii. S -> B : {A, Kab, Na, {Kab}Kas, Ts}Kb

iii. B -> A : {Nb, Na}Kab, {Kab}Kas

iv. A -> B : {Nb}Kab

利用我們的演算法分析後的結果如下：

- DirectAttack:
  - 沒找到
- pieceTogether:
  - 沒找到
- Relationship:
  1. 可以控制 Message 3 的"{Kab}Kas"這筆資料
  2. 可以取得 Message 1 的"{A, Ta, B, Na}Kas"這筆資料
     - 根據 1 和 2 可以發現這兩點彼此間沒有關係，所以無法找到其它攻擊路徑。

第五章 結論與未來展望

在這一章節中，我們將會對我們提出的演算法做一個簡單結論並且分析他的優缺點和探討未來可以改進的方向。

本文中，我們提出了一個驗證認證協定的演算法來找出攻擊認證協定中攻擊的目標 Item(重要的 Item)的方式，在這個演算法中由三個重要的角色所組成：攻擊的目標 Item(重要的 Item)、可以利用的資源、尋找漏洞。由於在我們的演算法中，必須要先找出認證協定中我們要攻擊的目標 Item，之後是尋找所有可以利用的資源，最後是找出漏洞。而在本文中，我們一律使用 Kas、Kbs、Kab 這種 symmetric key 的表示方法來說明我們的演算法，然而要修改成 asymmetric key 的表示方法只需要在 **3.2.3** 節一開始將所有 Item 分成兩類的時候，將 public key 加到“有意義的 Item”裡面，並且在“沒有意義的 Item”是否變成“有意義的 Item”的判斷時，只需要加上“沒有意義的 Item”中用 private key 加密的 Item 如果相對應的 public key 出現在“有意義的 Item”中即可變爲“有意義的 Item”，所以即使要修改成 asymmetric key 的表示方法也只需要多加一個 if 判斷式即可。

在我們系統的分析中，在分析攻擊行爲後會將可能的攻擊方式顯示在 Hacker 區塊的訊息中，而這個訊息分成 DirectAttack、pieceTogether 和 Relationship 三類。DirectAttack 只是提出在目前這個 Run，Attack 可以利用哪個 Item 攻擊哪個 Item。而 pieceTogether 一樣也是提出 Attacker 可以利用哪個 Item 攻擊哪個 Item，但是跟 DirectAttack 不一樣的是 pieceTogether 要利用的 Item 是經由一個 Run 或一個 Run 以上可以取得的，而 DirectAttack 卻是一個 Run 可以取得的。另外 Relationship 可以找出更多樣化路徑的攻擊方式。而這三類的訊息都只是提出一個攻擊的重點，而詳細的攻擊流程還是需要使用者去觀察設計出來。就像 **4.2** 和 **4.3** 節一樣，根據我們系統分析出來的攻擊重點，讓認證協定設計者可以經由這些重點更容易

的思考並改善他們設計出來的認證協定的漏洞。

在我們的演算法中，雖然說可以找出利用某個 Item 來攻擊的一個 Run 或多個 Run 的攻擊方式，但是我們的演算法沒辦法找出利用同樣的一個 Item 來攻擊的所有攻擊路徑，而這也是未來可以改進的方向，也將會使得同樣的漏洞但不只一種的攻擊方式越來越清晰明瞭。

參考文獻

[1] M. Burrows, M. Abadi, and R. Needham, “A logic of authentication”, *ACM Trans. Computer Systems*, Vol. 8, No. 1, pp. 18-36, 1990.

[2] D. Nessett, “A critique of the Burrows, Abadi and Needham logic”, *ACM Operating Systems Review*, pp.35-38, April 1990.

[3] Brackin, S.H. “Automatically detecting most vulnerabilities in cryptographic protocols**”,** IEEE Conference, DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings , Volume: 1 , 25-27 Jan. 2000, Pages:222 - 236 vol.1

[4] I.-Lung Kao, and Randy Chow, “An efficient and secure authentication protocol using uncertified keys”, *ACM SIGOPS Operating Systems Review*, Volume 29 , Issue 3 (July 1995), Pages: 14 – 21, Year of Publication: 1995

[5] W. Yang (2005), “Uncovering Attacks On Security Protocols”, in *Proceedings of International Conf. Information Technology and Applications*, Sydney, Australia, pp. 4-7, July 2005.

[6] U Meyer, S Wetzel, “A Man-in-the-Middle Attack on UMTS”, *Proceedings of the 2004 ACM workshop on Wireless secu*rity, 2004

[7] Debbabi, M. Mejri, M. Tawbi, N. and Yahmadi, I. “Formal automatic verification of authentication cryptographic protocols”, *Proceedings First IEEE International Conference Conference on Formal Engineering Methods*

[8] Guttman J.D.and Thayer, F.J. “Authentication tests”, *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*

[9] Carlsen, U. “Cryptographic Protocol Flaws: know your enemy”, *Ecole Nat. Superieure des Telecommun. de Bretagne, Cesson Sevigne, France*

[10] Paulson, L.C. "Proving security protocols correct", *Logic in Computer Science, 1999. Proceedings. 14th Symposium on*

[11] D. Wagner and B. Schneier. "Analysis of the SSL 3.0 protocol." *In Proceedings of the 2nd USENIX Workshop on Electronic Commerce (EC-96), pages 29 – 40, Berkeley, Nov. 18 – 21 1996. USENIX Association.*

附錄一

下面我們以 Needham-Schroeder protocol 爲例子來說明當 content 和 key 都一樣的時候是如何能成功攻擊認證協定。

Needham-Schroeder protocol

1. A -> S : A, B, Na
2. S -> A : {Na, B, Kab, {Kab, A}Kbs}Kas
3. A -> B : {Kab, A}Kbs
4. B -> A : {Nb}Kab
5. A -> B : {Nb - 1}Kab

將 Needham-Schroeder protocol 利用我們的演算法進行分析。分析結果如下:

- **攻擊的目標 Item(重要的 Item)**：{Kab, A}Kbs、{Na, B, Kab, {Kab, A}Kbs}Kas。
- 攔截直接使用的資源：A、B、Na、{Na, B, Kab, {Kab, A}Kbs}Kas、{Kab, A}Kbs、{Nb}Kab、{Nb - 1}Kab。
- 可控制的資源：A、B、Na、{Kab, A}Kbs。

尋找漏洞的過程中，我們會發現 Message 3 的{Kab, A}Kbs 跟**攻擊的目標 Item(重要的 Item)**中的{Kab, A}Kbs 一模一樣（也就是 content 和 key 都一樣）。

- 首先根據 **3.2.4** 節的判斷說明，先將此 Item 的 flag 值設成 True（代表找到可以替代的 Item），然後尋找此 Item 是否包含 Nonce、Timestamp 這些用來確保 Fresh 的資訊。如果有就無法攻擊，如果沒有就可能可以攻擊。此時我們可以發現{Kab, A}Kbs 沒有包含確保 Fresh 的資訊，因此可能會被攻擊。
- 接著我們會先找出被記錄在**攻擊的目標 Item(重要的 Item)**的所有 Item 中的 Location 值大於等於{Kab, A}Kbs 的 Location 值。找到之後判

斷這些 Item 的 flag 是否等於 True。如果都等於 True，那麼就確定可以攻擊（此時的狀況，只有找到{Kab, A}Kbs 的 Location 值大於等於自己的 Location 值，一開始我們已經將 flag 設成 True，因此確定可以攻擊）。

攻擊步驟：

1.1 A -> S : A, B, Na

1.2 S -> A : {Na, B, Kab, {Kab, A}Kbs}Kas

1.3 A -> I(B) : {Kab, A}Kbs

2.3 I(A) -> B : {Kab, A}Kbs

爲什麼還要判斷 Location 值大於等於目前分析的 Item 的 Location 值，首先我們將原本的認證協定修改一下：

1. A -> S : A, B, Na

2. S -> A : {Na, B, Kab, {Kab, A}Kbs}Kas

3. A -> B : {Kab, A}Kbs, {Na'}Kab

4. B -> A : {Na' - 1}Kab

而這認證協定的**攻擊的目標 Item(重要的 Item)**變爲：{Kab, A}Kbs、{Na'}Kab、{Na, B, Kab, {Kab, A}Kbs}Kas。在此我們先不管攔截直接使用的資源和可控制的資源爲何。一樣也會找到 Message 3 的{Kab, A}Kbs 和**攻擊的目標 Item(重要的 Item)**裡的{Kab, A}Kbs 一模一樣。同樣的此 Item 沒有包含確保 Fresh 的資訊，但是我們找到{Kab, A}Kbs 和{Na'}Kab 兩個 Item 的 Location 大於等於自己的 Location 值，同樣的{Kab, A}Kbs 的 flag 已經是 True 了，但是{Na'}Kab 的 flag 值還是 false，因此無法攻擊。

講了這麼多，主要是要判斷出 Item 沒有包含確保 Fresh 的資訊，即使沒有包含，但是跟此 Item 同樣 Message 或是之後的 Message 的 Item 有包含確保身分的資訊，那麼 Attacker 攻擊這個 Item 是無法成功的，因爲其他的 Item 會把這個攻擊阻擋下來了。

附錄二

下面我們以 Otway-Rees protocol 爲例子來說明當 content 一樣而 key 不一樣的時候是如何能成功攻擊認證協定。

Otway-Rees protocol

1. A -> B : Na, A, B, {Na, A, B}Kas
2. B -> S : Na, A, B, {Na, A, B}Kas, Nb, {Na, A, B}Kbs
3. S -> B : Na, {Na, Kab}Kas, {Nb, Kab}Kbs
4. B -> A : Na, {Na, Kab}Kas

將 Otway-Rees protocol 利用我們的演算法進行分析。分析結果如下:

- **攻擊的目標 Item(重要的 Item)**：{Na, Kab}Kas、{Nb, Kab}Kbs、{Na, A, B}Kas、{Na, A, B}Kbs。
- 攔截直接使用的資源：Na、A、B、{Na, A, B}Kas、Nb、{Na, A, B}Kbs、{Na, Kab}Kas、{Nb, Kab}Kbs。
- 可控制的資源：Na、A、B、Nb、{Na, A, B}Kas、{Na, A, B}Kbs、{Na, Kab}Kas、{Nb, Kab}Kbs。

在 **3.2.4** 節裡我們說只要多一個 Run 就可以成功攻擊。這裡就不說明其它一些資訊的判斷，而只顯示並說明如何多一個 Run 就可以成功攻擊。

攻擊步驟：

1.1 A -> B : Na, A, B, {Na, A, B}Kas

1.2 B -> S : Na, A, B, {Na, A, B}Kas, Nb, {Na, A, B}Kbs

2.1 I -> A : Ni, I, A, {Ni, I, A}Kis

2.2 A -> I(S) : Ni, I, A, {Ni, I, A}Kis, Na', {Ni, I, A}Kas

2.2 I(A) -> S : Ni, I, A, {Ni, I, A}Kis, Na, {Ni, I, A}Kas

2.3 S -> I(A) : Ni, {Ni, Kab'}Kis, {Na, Kab'}Kas

1.3 S -> I(B) : Na, {Na, Kab}Kas, {Nb, Kab}Kbs

1.4 I(B) -> A : Na, {Na, Kab'}Kas

根據 **3.2.4** 節裡我們說明過，假設今天是 A 要跟 B 溝通，我們的演算法進行分析的時候，假設目前分析的**攻擊的目標 Item(重要的 Item)**是{Na, Kab}Kas，而我們找到{Nb, Kab}Kbs 跟**攻擊的目標 Item(重要的 Item)**只有 key 不一樣。因爲這個情況下是 A 要跟 B 溝通，而我們找到的這個 content 一樣、key 不一樣的 Item 是由 B 產生的，因此 Attacker 只要自己去跟 A 溝通的時候，此時分析的**攻擊的目標 Item(重要的 Item)**是{Ni, Kab'}Kis，而我們可以找到{Na, Kab'}Kas 跟這個**攻擊的目標 Item(重要的 Item)**只有 key 不一樣。用文字說明比較難理解，上面顯示的攻擊步驟的流程就是我們這段文字說明的意義。

附錄三

下面我們以 Neuman-Stubblebine protocol 爲例子來說明當 content 不一樣而 key 一樣的時候是如何能成功攻擊認證協定。

Neuman-Stubblebine protocol

1. A -> B : A, Na
2. B -> S : B, {A, Na, Tb}Kbs, Nb
3. S -> A : {B, Na, Kab, Tb}Kas, {A, Kab, Tb}Kbs, Nb
4. A -> B : {A, Kab, Tb}Kbs, {Nb}Kab

將 Neuman-Stubblebine protocol 利用我們的演算法進行分析。分析結果如下:

- **攻擊的目標 Item(重要的 Item)**：{A, Kab, Tb}Kbs、{Nb}Kab、{B, Na, Kab, Tb}Kas、{A, Na, Tb}Kbs。
- 攔截直接使用的資源：A、Na、B、{A, Na, Tb}Kbs、Nb、{B, Na, Kab, Tb}Kas、{A, Kab, Tb}Kbs、{Nb}Kab。
- 可控制的資源：A、Na、B、Nb、{Nb}Kab。

以此認證協定來說，分析後的結果，我們可以發現{A, Na, Tb}Kbs 和**攻擊的目標 Item(重要的 Item)**的{A, Kab, Tb}Kbs 的關係是 content 不一樣，key 一樣。而在這裡我們會找出這兩個 Item 中彼此不一樣的 subitem 並分析是否其中一個是 session key，如果是的話就代表可以攻擊，因爲 Attacker 可以拿其他資料讓對方以爲是 session key 而開始進行溝通。以這個範例來說，就是以 Na 來讓對方以爲 Na 就是他們在這個 Run 用來加解密的 session key。