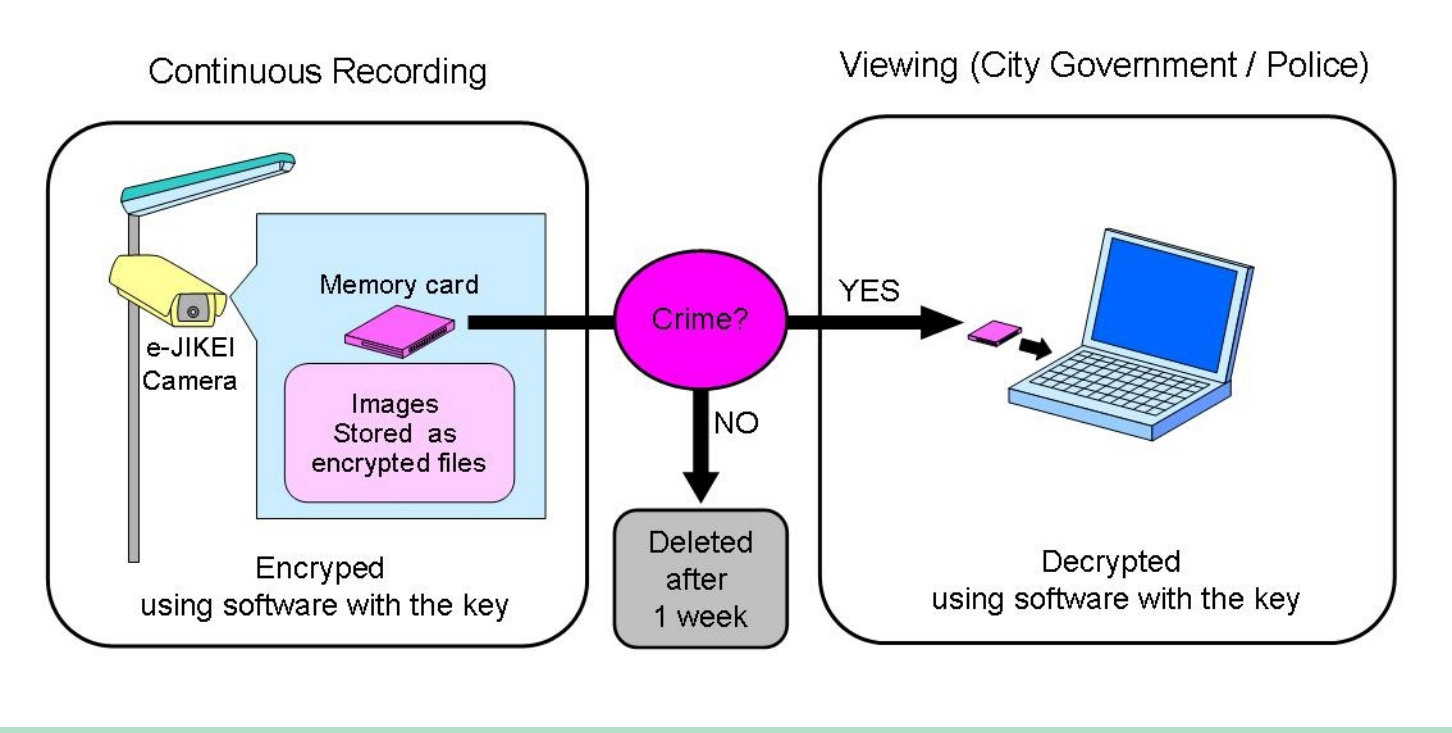
ISSN: 2153-1234 Vol.1, No.2, October 2010

Scientific Research

Journal of Information Security



ISSN: 2153-1234

www.scirp.org/journal/jis

Journal Editorial Board

ISSN: 2153-1234 (Print) ISSN: 2153-1242 (Online)

http://www.scirp.org/journal/jis

Journal of Information Security, **2010, 1, 45-94**
Published Online October 2010 in SciRes (http://www.SciRP.org/journal/jis/)

TABLE OF CONTENTS

Volume 1 Number 2 October 2010

The figure on the front cover is from the article published in Journal of Information Security , 2010,Vol.1, No.2, pp.68-73, by V. Prashyanusorn, Y. Fujii, S. Kaviya, S. Mitatha and P. P. Yupapin.

Journal of Information Security（JIS）
Journal Information

SUBSCRIPTIONS

The *Journal of Information Security* (Online at Scientific Research Publishing, www.SciRP.org) is published quarterly by Scientific Research Publishing, Inc., USA.

Subscription rates:

Print: $50 per issue.
To subscribe, please contact Journals Subscriptions Department, E-mail: sub@scirp.org

SERVICES

Advertisements
Advertisement Sales Department, E-mail: service@scirp.org

Reprints (minimum quantity 100 copies)
Reprints Co-ordinator, Scientific Research Publishing, Inc., USA.
E-mail: sub@scirp.org

COPYRIGHT

PRODUCTION INFORMATION

For manuscripts that have been accepted for publication, please contact:
E-mail: jis@scirp.org

Journal of Information Security, 2010, 1, 45-55
doi:10.4236/jis.2010.12006 Published Online October 2010 (http://www.SciRP.org/journal/jis)

Extending the Strand Space Method with Timestamps: Part I the Theory*

Yongjian Li[1,2], Jun Pang[3]
[1]*Chinese Academy of Sciences, Institute of Software, Laboratory of Computer Science, Beijing, China*
[2]*The State Key Laboratory of Information Security, Beijing, China*
[3]*University of Oldenburg, Department of Computer Science, Safety-critical Embedded Systems, Oldenburg, Germany*
E-mail: *lyj238@ios.ac.cn, jun.pang@informatik.uni-oldenburg.de*
Received June 23, 2010; *revised September* 14, 2010; *accepted July* 12, 2010

Abstract

In this paper, we present two extensions of the strand space method to model Kerberos V. First, we include time and timestamps to model security protocols with timestamps: we relate a key to a crack time and combine it with timestamps in order to define a notion of recency. Therefore, we can check replay attacks in this new framework. Second, we extend the classic strand space theory to model protocol mixture. The main idea is to introduce a new relation $\mapsto$ to model the causal relation between one primary protocol session and one of its following secondary protocol session. Accordingly, we also extend the definition of unsolicited authentication test.

Keywords: Strand Space, Kerberos V, Theorem Proving, Verification, Isabelle/HOL

1. Introduction

The strand space model [1] is a formal approach to reasoning about security protocols. For a legitimate regular participant, a strand s represents a sequence of messages that the participant would receive or send as part of a run as his/her role of the protocol. A typical message has the form of $\{|h|\}_K$ denoting the encryption of h using key K. An element of the set of messages is called a *term*. A term t' is a subterm of t is written as $t' \sqsubset t$. Usually, we call a strand element *node*. Nodes can be either positive, representing the transmission of a term, or negative, representing the reception of a term. For the penetrator, the strand represents atomic deductions. More complex deductions can be formed by connecting several penetrator strands. Hence, a strand space is simply a set of strands with a trace mapping. Two kinds of causal relation (arrow), $\rightarrow$ and $\Rightarrow$, are introduced to impose a graphic structure on the nodes of the space. The relation $\preceq$ is defined to be the reflexive and transitive closure of these two arrows, modelling the causal order of the events in the protocol execution. The formal analysis based on strand spaces can be carried on the notion of bundles. A bundle is a causally well-founded set of nodes and the two arrows, which sufficiently formalizes a session of a protocol. In a bundle, it must be ensured that a node is included only if all nodes that proceed it are already included. For the strand corresponding to a principal in a given protocol run, we construct all possible bundles containing nodes of the strand. In fact, this set of bundles encodes all possible interactions of the environment with that principal in the run. Normally, reasoning about the protocol takes place on this set of bundles.

However, the original strand space model has its semantical limitations to analyze the real-world protocols such as Kerbeoros protocols. First, it does not include timestamps as formalized message components, and therefore can not model security protocols with timestamps. In fact, the strand space model [1] as given by Thayer Fábrega, Herzog, and Guttman is only benchmarked on nonce-based protocols such as the Needham-Schroeder protocol and the Otway-Rees protocol. But many modern protocols use timestamps to prevent replay attacks, so this deficiency of the strand space theory makes it difficult to analyze these protocols. Second, it

*This is a revised and extended version of the homonymous paper appearing in the Proceedings the Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2007, IEEE Computer Society). The main modifications have been made on the presentation of the technical material, with the purpose of having full details. The first author is supported by grants (No.60496321, 60421001) from National Natural Science Foundation of China.

does not address issues of the protocol dependency when several protocols are mixed together. Many real-world protocols are divided into causally related multiple phases (or subprotocols), such as the Kerberos and Neuman-Stubblebine protocols. One phase may be used to retrieve a ticket from a key distribution center, while a second phase is used to present the ticket to a security-aware server. To make matters more complex, many protocols such as Kerbeors use timestamps to guarantee the recency of these tickets, that is, such tickets are only valid for an interval, and multiple sub-protocol sessions can start in parallel by the same agent using the same ticket if the ticket does not expire. Little work has been done to formalize the causal relation between protocols in a protocol mixture environment.

The aim of this paper is twofold. The first aim is to extend the strand space theory to cover the aforementioned two semantical features. Briefly, we include time and timestamps to model security protocols with timestamps: we relate a key to a crack time and combine it with timestamps in order to define a notion of recency. Therefore, we can check replay attacks in this new framework. We also extend the classic strand space theory to model protocol mixture: a new relation $\mapsto$ is introduced to model the causal relation between one primary protocol session and one of its following secondary protocol session. Despite the extensions, we hope that the extended theory still maintains the simple and powerful mechanism to reason about protocols. The second aim is practical. We hope to apply the extended theory to the analysis of some real-world protocols. Here we select Kerberos V as our case study. Kerberos V is appropriate because it covers both timestamps and protocol mixture semantical features.

2. Motivations

2.1. A Short Introduction to Kerberos V

The first version of Kerberos protocol was developed in the mid eighties as part of project Athena at MIT [2]. Over twenty years, different versions of Kerberos protocols have evolved. Kerberos V (**Figure 1** and **Figure 2**) is the latest version released by the Internet Engineering Task Force (IETF) [4]. It is a password-based system for authentication and authorization over local area networks. It is designed with the following aims: once a client authenticates himself to a network machine, the process of obtaining authorization to access another network service should be completely transparent to him. Namely, the client only needs enter his password once during the authentication phase. In order to access some network service, the client needs to communicate with two trusted

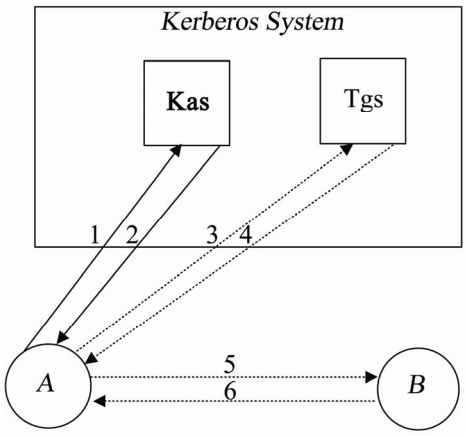


Figure 1. The layout of Kerberos V.

Authentication phase

$1.A \rightarrow Kas : \{\!|A, Tgs|\!\}$

$2.Kas \rightarrow A : \underbrace{\{\!|A, Tgs, authK, T_a|\!\}_{K_{Tgs}}}_{authTicket}, \{\!|A, Tgs, authK, T_a|\!\}_{K_A}\}$

Authorisation Phase

$3.A \rightarrow Tgs : \{\!|A, Tgs, authK, T_a|\!\}_{K_{Tgs}}, \{\!|A, t_2|\!\}_{authK}, B\}$

$4.Tgs \rightarrow A : \underbrace{\{\!|A, B, servK, T_S|\!\}_{K_B}}_{servTicket}, \{\!|A, B, servK, T_S|\!\}_{authK}\}$

Service Phase

$5.A \rightarrow B : \{\!|\{\!|A, B, servK, T_S|\!\}_{K_B}, \{\!|A, t_3|\!\}_{servK}\}$

$6.B \rightarrow A : \{\!|t_3|\!\}_{servK}$

Figure 2. Kerberos V: message exchanging.

servers **Kas** and *Tgs* . **Kas** is an authentication server (or the key distribution center) and it provides keys for communication between clients and ticket granting servers. *Tgs* is a ticket granting server and it provides keys for communication between clients and application servers. The full protocol has three phases each consisting of two messages between the client and one of the servers in turn. Messages 2 and 4 are different from those in Kerberos IV [2,4] in that nested encryption has been cancelled. Later we will show that this change does not affect goals of the protocol.

2.2. Timestamps

Timestamps are heavily used in the Kerberos protocols to guarantee the recency of messages. The strand space model cannot express security protocols with timestamps, although Guttman [5] provided a notion of recency and he used it to analyze replay attacks of a variant of the Yahalom protocol, it is still impossible to analyze secu-

rity protocols with timestamps. Timestamps are mainly used to avoid replay attacks in the literature of security protocols. Usually such attacks occur in protocols that involve a message encrypted by a session key, and the session key itself is sent as a part of a message which is encrypted by a long-term key. Although penetrators can never obtain a long-term key K if K is not sent as a part of a message, it is usually assumed that m will be obtained from $\{|m|\}_K$ via cryptanalysis by a penetrator after some time t, especially if a session key SK is a component of m, then it will be compromised after the time t. Here, we say that the time t is the crack time of K, and every key will be related to a crack time. Although the penetrator cannot obtain m from $\{|m|\}_K$ during a protocol session provided that $\{|m|\}_K$ did not occur in any old session and K's crack time is longer than the time of a session allowed, he still may replay stale messages and use the old compromised session keys to launch attacks if some message of the protocol does not contain necessary information to indicate its recency.

For example, in the Needham-Schroeder symmetric key protocols (see **Figure 3**), when B receives the third message $\{|A,K|\}_{K_B}$, although B can infer that it was generated by S, he is not certain of its recency because no such information is available. Perhaps $\{|A,K|\}_{K_B}$ has occurred in an old session, and a penetrator has cryptanalyzed the conversation to obtain the session K. In that case, the penetrator can start a session by resending $\{|A,K|\}_{K_B}$, and later return $\{|N_b+1|\}_K$. Denning and Sacco [6] pioneered the use of timestamps to fix the flaw of the protocol. A timestamp t, which is a number, is employed in the ticket $\{|A,K,t|\}_{K_B}$ by S to mark the time of issue, and will be compared with the current time by the receiver B to check whether the ticket is recent. In this paper, we will assume that all agents are synchronized via a global clock, so an agent knows the time when receiving or sending a message.

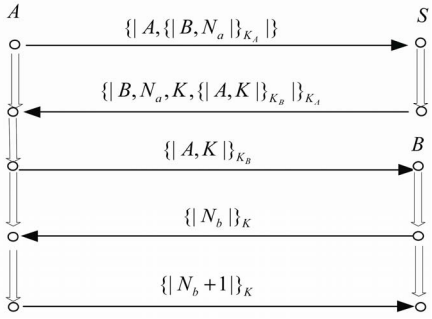


Figure 3. Needham-Schroeder symmetric key protocol.

[1]It is not the time to obtain K from $\{|m|\}_K$.

In this paper, we extend the strand space model with such features. A crack time is attached to every key. The crack-time of a key K is the time needed by a penetrator to break an encrypted message $\{|m|\}_K$.[1] We model a timestamp in the same way as atomic messages. A regular agent can attach a timestamp in a message to indicate when it sends the message, and check whether a received message encrypted by a key K is recent by comparing the timestamp in the message with the current time and the crack time of K. Once a message $\{|m|\}_K$ is no longer recent, a penetrator can break the message to obtain m.

2.3. Protocol Mixture

Another important feature of Kerberos, which is difficult to model in strand space, is protocol mixture. Kerberos protocol comprises three protocol phases: authentication, authorization, and service protocol phases. Once a client has passed an authentication phase and obtained an authentication ticket, then he can use the ticket to start multiple sessions of the authorization protocol phases in parallel to obtain different service tickets to access the services he needs provided that the authentication ticket does not expire. Similarly, once the client has gone through a session of the authorization phase, then he can use the service ticket obtained to access the service server for many times provided that the service ticket does not expire. Usually we refer to a protocol as one primary protocol, and the protocol following it as a secondary protocol. We note that other researchers have discussed the problem of protocols mixture [7,8], but they emphasized more on independency between two protocols. Namely, if they have disjoint encryption, then the first protocol is independent of the second. By this they mean that if the first protocol can achieve a security goal (either an authentication goal or a secrecy goal) when executed in isolation, then it still achieves the same security goal when executed in combination with the second protocol. In their theory, one primary and one secondary strands are rather independent of each other.

However, in Kerberos protocols, a secondary strand cannot be independent of its primary strand, and the events of a secondary strand has temporal relation with the events of the primary strand. For example, assuming that a client A runs a session s' of an authorization phase of Kerberos V, then he must have passed an authentication phase s. When A receives the second message in the session s', he must ensure that the current time should be before the ticket $\{|A,Tgs,authK,T_a|\}_{K_{Tgs}}$ expires, so A needs know the time T_a when the ticket is created, and checks how much time has elapsed until now. This side condition cannot be expressed without the semantical specification of s, because in the intended

case the ticket is a term encrypted with *Tgs* 's long-term key, which is unintelligible to A, A cannot know T_a from the ticket. Then A can only know the time T_a from the previous authentication phase s. Therefore, we need to formalize the facts that s' follows s, and A holds all the knowledge of s when he runs s', and there should be causal relation between events in s and those in s'. Such semantical features are not covered in [7,8].

In order to model the aforementioned causal relation between a primary strand and its following secondary strands, we introduce a new relation $\mapsto$ between strands. $s \mapsto s'$ holds if s is a primary protocol strand and s' is a subsequent secondary protocol strand. E.g., let s and s' be client strands in an authentication phase and authorization phase in Kerberos V respectively, $s \mapsto s'$ means that a client runs an authentication session s, and subsequently starts an authorization session s'. In practice, if $s \mapsto s'$, then s and s' may be two different processes started by the same client, and when the client starts s, he knows all the events which have occurred in s. This knowledge is useful for the client to perform actions in s'. E.g., when a client starts an authorization session, he uses an authentication ticket which is obtained in the preceding authentication session, and he knows the time when the ticket is created. So a causal relation should be imposed on two events which occur in a primary strand and its subsequent secondary strand.

Figure 4 illustrates a possible protocol execution of Kerberos V using the relation $\mapsto$. A client runs an instance in authentication phase, which is represented by the strand i_1. Following the primary protocol instance, the same client may run three authorisation subprotocol instances in parallel, which are showed in the strands i_{21}, i_{22}, and i_{23} respectively. Tr_{21} is a subtree which is a collection of client strands in the service phase. Tr_{22} and Tr_{23} are similar to Tr_{21}. Note that the semantics of the relation $\mapsto$ means that i_{21} and i_{22} and i_{23} inherits all the same knowledge from i_1, so they shares the same *authTicket*, *authK*, *Tgs*, T_a, *etc*. Therefore, if $term(i_1,1) = \left\{\left|authTicket, \left\{\left|A, Tgs, authK, T_a\right|\right\}_{K_A}\right|\right\}$ then then it must be the case that

$$term(i_{11},1) = \left\{\left|\left\{\left|authTicket, \left\{\left|A, t_1\right|\right\}_{authK}, B_1\right|\right\}\right|\right\}$$

and

$$term(i_{13},1) = \left\{\left|\left\{\left|authTicket, \left\{\left|A, t_2\right|\right\}_{authK}, B_2\right|\right\}\right|\right\}$$

for some t_1, t_2, B_1 and B_2. Here $t_1(B_1)$ can be different from $t_2(B_2)$. This means that the client use the same *authTicket* to obtain two different server tickets for accessing servers B_1 and B_2. Without the relation

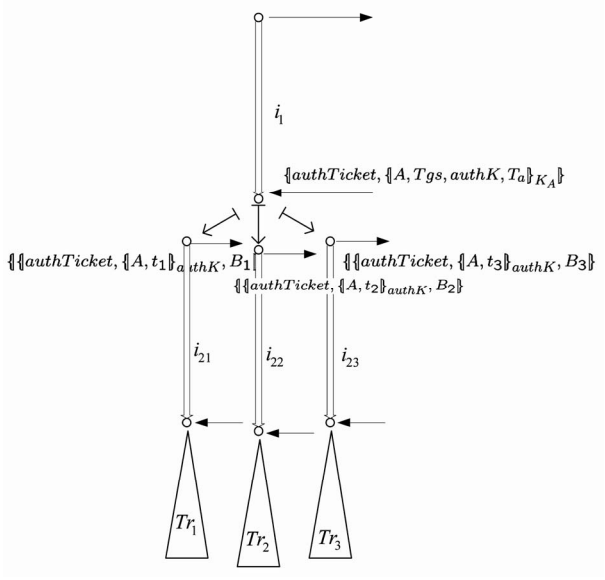


Figure 4. An illustration of protocol mixture.

$\mapsto$, i_{21} and i_1 are independent, therefore the knowledge inherence relation between them can not be imposed.

We extend the relation $\Rightarrow$ in the strand space model in the way that $n_1 \Rightarrow n_2$ holds if $n_1 = (s,i)$ and $n_2 = (s,i+1)$, or $n_1 = (s, length(tr(s))-1)$ and $n_2 = (s',0)$ and $s \mapsto s'$. Namely, the edge means either that n_1 is an immediate causal predecessor of n_2 on the same strand s or that n_1 is the last event in a primary strand s and n_2 is the first event in the subsequent secondary strand s'.

Structure of the Paper. In Section 3, we present the theory of the strand space method with our two extensions. We devote Section 5 to a new definition of unsolicited authentication test. We discuss related work and conclude the paper in Section 6.

3. Preliminaries

3.1. Messages and Actions

The set of messages is defined as the following BNF notation:

$$\begin{array}{rcl} h & ::= & \mathbf{name}(A) \mid \mathbf{nonce}(n) \\ & \mid & \mathbf{key}(K) \mid \mathbf{timestamp}(t) \\ & \mid & \{|h_1, h_2|\} \mid \mathbf{enc}(h, K) \end{array}$$

where A is an element from a set of agents, n from a set of nonces, K from a set of keys, and t from a set of times. Here we assume that **Time** is the set of all natural numbers. $t_1 < t_2$ means that the time t_1 is ear-

lier than t_2. We represent a timestamp by marking t as **timestamp**(t). Except this extension, the definitions of other kinds of messages are the same as those in the classic strand space theory. We call a key symmetric if $K^{-1} = K$. Otherwise, K is a public key and K^{-1} is private. For each K, we define *cracktime*(K) as the crack time of K. $\{|h_1, h_2|\}$ is called a composed message. We will write $\{|\{|h_1, h_2|\}, h_3|\}$ as $\{|h_1, h_2, h_3|\}$. $\{|h_1, h_2|\} = \{|h_1', h_2'|\}$ if and only if $h_1 = h_1'$ and $h_2 = h_2'$. We abbreviate **enc**(h, K) as $\{|h_K|\}$, denoting the encryption of h using key K. In our formulation, we use K_A to define a long-term key shared between an agent (also called a client) A and a server, and clients have distinct keys. An element of the set of messages is also called a *term*. Terms of the form **name**(A), **nonce**(n), **timestamp**(t), or **key**(K) are said to be atomic.[2] The set of all messages is denoted by **Message**. A message h is a text message if $h \neq K$ for any K. The set of all atomic text messages is denoted by T. We frequently need the subterm relation on messages. A term g' is a subterm of g is written as $g' \sqsubset g$.

Definition 1 *The subterm relation* $\sqsubset$ *is defined inductively as the smallest relation such that* $g \sqsubset g$, $g \sqsubset \{|h|\}_K$ *if* $g \sqsubset h$, *and* $g \sqsubset \{|h_1, h_2|\}$ *if* $g \sqsubset h_1$ *or* $g \sqsubset h_2$.

In our extended strand space model, we need to revise the definition of actions. The main point is to record the time when an action takes place. The transmission of a term g at time t is denoted by $(t, +, g)$, and the reception of a term g at t is denoted by $(t, -, g)$. Both are the possible actions that participants and a penetrator can take. We represent the set of finite sequences of actions by (**Time**, ±, **Message**)*.

3.2. Strands and Strand Spaces

A strand space Σ is a set of strands with a trace mapping $tr : \Sigma \to (\mathbf{Time}, \pm, \mathbf{Message})^*$. A strand element is called a node. (s, i) is the i-th node on strand s ($0 \le i < length(s)$). We use $n \in s$ to denote that a node n belongs to the strand s. The set of all the nodes is denoted by $\mathcal{N}$. If $n = (s, i)$ and $tr(s)_i = (t, \sigma, g)$, then we define $time(n)$ and $term(n)$ and $sign(n)$ to be the occurring time, the term and the sign of the node n, respectively. Namely, $time(n) = t$, $term(n) = g$, and $sign(n) = \sigma$. We call a node positive if its term has sign $+$, and negative if its term has sign $-$. A strand is a protocol history from the point of view of a single participant in a protocol run, so we explicitly define an attribute function $attr : \Sigma \to A$ to indicate which agent's peer a strand is. Namely, $attr(s) = a$ means that a is the agent who performs actions of the strand s in the run.

[2]For convenience, we often write A, n, K and t instead of **name** (A), **nonce** (n), **key** (K), and **timestamp** (t).

As mentioned in Section 2, we introduce a relation $\mapsto$ between strands to model protocol mixture, and $s \mapsto s'$ holds if s is a primary protocol strand, and s' is a subsequent secondary protocol strand. To make our theory sound, we also restrict the relation $\mapsto$ to be a tree-like one with the following principles. First, $\mapsto$ is irreflexive, *i.e.* $s \not\mapsto s$. Second, every strand has at most one $\mapsto$ predecessor, meaning if $s \mapsto s''$ and $s' \mapsto s''$, then $s = s'$. The two restrictions are consistent with our intuition on protocol mixture. The first principle says that one protocol session can not follow itself, this simply means that the primary protocol session and any one of its following secondary protocol sessions are different. The second principle shows that one secondary protocol session follows a unique primary protocol session.

Two kinds of causal relation (arrow), $\rightarrow$ and $\Rightarrow$, are introduced to impose a graph structure on the nodes of Σ. To be more precise, the relation $n \Rightarrow n'$ holds between nodes n and n' if $n = (s, i)$ and $n' = (s, i+1)$ and $time(n) \le time(n')$, or $n = (s, length(tr(s)) - 1)$ and $n' = (s', 0)$ and $s \mapsto s'$ and $time(n) \le time(n')$. This relation means that the event n' immediately follows n. On the other hand, the relation $n \rightarrow n'$ holds for nodes n and n' if $term(n) = term(n') = g$ for some term g, $sign(n) = +$ and $sign(n') = -$, and $time(n) \le time(n')$. This represents that n sends a message g and n' receives the message at a later time. Obviously, here we require that the two relations must respect the order of time. The relation $\preceq$ is defined to be the reflexive and transitive closure of $\rightarrow$ and $\Rightarrow$, modelling the causal order of the events in the protocol execution. We say that a term g *originates* at a node n if and only if n is positive, $g \sqsubset term(n)$, and there is no node n' such that $n' \Rightarrow^+ n$ and $g \sqsubset term(n')$; We say that g uniquely originates if and only if there exists an unique node n such that g originates from node n. Nonces and other recently generated terms such as session keys are usually uniquely originated.

3.3. Penetrator Strands

The symbol **Bad** is defined to denote the set of all the penetrators, and if an agent is not in **Bad**, then it is regular. There is a set of keys that are known initially to all the penetrators, denoted as $\mathbf{K}_{\mathcal{P}}$. $\mathbf{K}_{\mathcal{P}}$ usually contains all the public keys, all the private keys of all the penetrators, and all the symmetric keys initially shared between all the penetrators and principals playing by the protocol rules. It can also contain some keys to model known-key attacks. In this paper, we only need the fact that if an agent is not a penetrator then his shared key cannot be penetrated, which is formalized as follows.

Axiom 1 *If* $A \notin \mathbf{Bad}$, *then* $K_A \notin \mathbf{K}_{\mathcal{P}}$.

In the classic strand space theory, a penetrator can intercept messages, generate messages that are computable from its initial knowledge and the messages it intercepts. These actions are modelled by a set of penetrator strands, and they represent atomic deductions. More complex deduction actions can be formed by connecting several penetrator strands. In our extension, we assume that penetrators share their initial knowledge and can cooperate each other by composing their strands. Besides the behaviors inherited from classic strand space theory, a penetrator has the ability to crack an encrypted message once the message is no longer recent (see $KC_{K,h}$ strand).

Definition 2 *A penetrator's trace relative to* $\mathbf{K}_{\mathcal{P}}$ *is one of the following, where* $t,t_1,t_2,t_3 \in \mathbf{Time}$ *and* $t_1 \le t_2 \le t_3$:

- M_g (text message): $[(t,+,g)]$, where $g \in T$.
- K_K (key): $[(t,+,K)]$, where $K \in \mathbf{K}_{\mathcal{P}}$.
- C_{gh} (concatenation): $[(t_1,-,g),(t_2,-,h),(t_3,+,\{|g,h|\})]$.
- $S_{g,h}$ (separation): $[(t_1,-,\{|g,h|\}),(t_2,+,g),(t_3,+,h)]$.
- $E_{h,K}$ (encryption): $[(t_1,-,K),(t_2,-,h),(t_3,+,\{|h|\}_K)]$.
- $D_{h,K}$ (decryption): $[(t_1,-,K^{-1}),(t_2,-,\{|h|\}_K),(t_3,+,h)]$.
- $KC_{K,h}$ (key-crack): $[(t_1,-,\{|h|\}_K),(t_2,+,h)]$, where $t_1 + cracktime(K) < t_2$.

In our theory, if a strand s belongs to a penetrator, namely, $attr(s) \in \mathbf{Bad}$, then s must be a penetrator strand. If a strand is not a penetrator strand, then it is regular. A node is called *regular* if it is not in the penetrator strands. Except the key crack strand ($KC_{K,h}$), our penetrator model is similar to the one in [1]. Here M_g (or K_K) does not imply that a penetrator can issue any unguessable terms which are not in his initial knowledge such as nonces and session keys. Because when we introduce secrecy or authentication properties about an unguessable term t for all penetrators, we usually assume that t uniquely originates from a regular strand, and this implicitly eliminates the possibility that any penetrator can originate t. Intuitively, we use $\mapsto$ to model regular agents to start a primary protocol session and then starts multiple parallel secondary protocol sessions, so a penetrator strand cannot be mixed with any other strand. To be more precise, for all penetrator strands s and all strands s', we have that $s \not\mapsto s'$ and $s' \not\mapsto s$. This implies that a penetrator strand can only be composed with other strands by the relation $\rightarrow$.

3.4. Bundles

The formal analysis based on strand spaces is carried on the notion of bundles, which represents the protocol execution under some configuration. A bundle is a causally well-founded graph, which sufficiently formalizes a session of a protocol.

Definition 3 *Suppose* $\mathcal{B} = (N_{\mathcal{B}}, (\rightarrow_{\mathcal{B}} \cup \Rightarrow_{\mathcal{B}}))$, $\rightarrow_{\mathcal{B}} \subseteq \rightarrow$, *and* $\Rightarrow_{\mathcal{B}} \subseteq \Rightarrow$. $\mathcal{B}$ *is a bundle if*

- $N_{\mathcal{B}}$ and $\rightarrow_{\mathcal{B}}$ and $\Rightarrow_{\mathcal{B}}$ are finite;
- If the sign of a node n is $-$, and $n \in N_{\mathcal{B}}$, then there is a unique positive node n' such that $n' \in N_{\mathcal{B}}$ and $n' \rightarrow_{\mathcal{B}} n$;
- If $n' \Rightarrow n$ and $n \in N_{\mathcal{B}}$, then $n' \in N_{\mathcal{B}}$ and $n' \Rightarrow_{\mathcal{B}} n$;
- $\mathcal{B}$ is acyclic.

Suppose $\mathcal{B}$ is a bundle, we say $n \in \mathcal{B}$ if n is a node in $N_{\mathcal{B}}$, and use $\preceq_{\mathcal{B}}$ to denote the reflexive and transitive closure of the relation $\rightarrow$ and $\Rightarrow$ in $\mathcal{B}$. In a bundle, it must be ensured that a node is included only if all nodes that proceed it are already included. So a bundle $\mathcal{B}$ has the following properties:

Lemma 1 (Bundle well foundedness) *Let* $\mathcal{B}$ *be a bundle. Then* $\preceq_{\mathcal{B}}$ *is a partial order, i.e. a reflexive, antisymmetric, transitive relation. Every non-empty subset of the nodes in* $\mathcal{B}$ *has* $\preceq_{\mathcal{B}}$ *minimal members.*

We have formalized the above extended strand space theory in the theorem prover Isabelle/HOL [9]. See [10] for details.

4. Penetrator's Knowledge Closure Property

In this section, we will describe a useful property on penetrator strands. This property specifies what knowledge can be obtained from some special message set. First we need to define a key is regular w.r.t. a node m in a bundle.

Definition 4 *A key* K *is regular w.r.t. a node* m *in a bundle* $\mathcal{B}$, *denoted by* $regular(k,m,\mathcal{B})$, *if and only if the following condition holds: for any node* n *in* $\mathcal{B}$, *if* $term(n) = K$ *and* $time(n) \le time(m)$, *then* n *must be regular.*

This definition is about K's secrecy w.r.t. a node m in a bundle $\mathcal{B}$, which means that K cannot be penetrated before m in the bundle. In most of the cases, we only consider security properties for a protocol in a given bundle, so it is natural for us to just consider whether a key can potentially be penetrated in this bundle. Besides, we also need consider temporal restriction $time(n) \le time(m)$ because we discuss K's secrecy a timed framework.

Definition 5 *Let* m *be a node in a bundle* $\mathcal{B}$. *A message* t, *is a component w.r.t.* m *in bundle* $\mathcal{B}$, *denoted by* $component(t,m,\mathcal{B})$, *if*

1) $(\forall g \quad h.t \neq \{|g,h|\})$;

2) $\left(\forall kh.t = \{|h|\}_k \rightarrow \left(regular\left(k^{-1},m,\mathcal{B}\right)\right)\right)$

Intuitively, $component(t,m,\mathcal{B})$ means that t basic unit that can not be analyzed in $\mathcal{B}$ by penetrators. Namely, t can not be detached because t is not a

concatenated form; and if t is an encrypted form of $\{|h|\}_K$ t can not be decrypted before m in $\mathcal{B}$ because k^{-1} can not be penetrated before m.

Definition 6 *Let m be a node in a bundle $\mathcal{B}$. a is a message which uniquely originates at some node n. A message set M is a test suite for a w.r.t. m in $\mathcal{B}$, denoted by $suite(M,a,m,n,\mathcal{B})$ if*

1) $\forall t \in M.a \sqsubset t \rightarrow component(t,m,\mathcal{B})$

2) $\forall t \in M.a \sqsubset t \rightarrow (\forall k\ h.t = \{|h|\}_k \rightarrow time(m) \leq time(n) + cracktime(k))$

3) $\forall t.a \not\sqsubset t \rightarrow t \in M$;

Intuitively, $suite(M,a,m,n,\mathcal{B})$ means that for any $t \in M$ such that $a \sqsubset t$, t can not be detached or decrypted before m because such t is a component w.r.t. m in bundle $\mathcal{B}$; furthermore, if t contains a and is of the form $\{|h|\}_K$ for some k and h, t can not be cracked before m because the duration between m and n is less than k's crack time, and this is guaranteed by (2). Recall that $time(n)$ is the first time when a occurs because a uniquely originates at n.

Now we need introduce a function *synth* on a message set H, which captures the "building up" aspect of penetrator's ability [4,11]. $synth(H)$ is defined to be the least set that includes H, agents, timestamps and is closed under pairing, and encryption.

Definition 7 *Consider a message set H, $synth(H)$ is a message set which is defined inductively as follows*:

1) $A \in synth(H)$ *if* A *is an agent name*;

2) $t \in synth(H)$ *if* t *is a timestamp*;

3) $m \in synth(H)$ *if* $m \in H$;

4) $\{|h|\}_k \in synth(H)$, *if* $h \in synth(H)$ *and* $k \in H$;

5) $\{|g,h|\} \in synth(H)$, *if* $g \in synth(H)$ *and* $h \in synth(H)$.

In the context of this paper, we usually assume that a is an unguessable atomic message such as a nonce, which is uniquely originated from a regular strand and encrypted in a message. Let $M_0 = \{t \mid a \sqsubset t \wedge t \in M\}$, in later discussions we usually assume that M_0 is the set of messages which is emitted by some regular strands. f M is a test suite for a w.r.t. m in b, then the set $synth(M)$ is a knowledge closure which penetrators can synthesize in the bundle b from M. Namely, if the messages received in a penetror strand are in $synth(M)$, then the messages sent in the strand must still be in $synth(M)$.

Before we prove the closure property, we need two useful lemmas, as shown below:

Lemma 2 *If M is a test suite for a w.r.t. m in $\mathcal{B}$, and $\{|g,h|\} \in synth(M)$, then $g \in synth(M)$ and $h \in synth(M)$.*

Lemma 3 If $\{|h|\}_K \in synth(M)$, then $h \in synth(M)$ or $\{|h|\}_K \in M$.

Let a be an atomic message that uniquely originates at some node n, m be a positive penetrator node in a bundle $\mathcal{B}$ such that and $a \sqsubset term(m)$. Suppose M is a test suite for a w.r.t. m in the bundle $\mathcal{B}$, if any message that the penetrator can receive in the strand is in $synth(M)$, then the penetrator can only send a term which is still in $synth(M)$. **Figure 5** illustrates such behaviors of penetrators on knowledge, where (a) shows the cases for $C_{g,h}$, $E_{h,K}$, and $D_{h,K}$; (b) shows the case for $S_{g,h}$; and (c) shows the case for $KC_{K,h}$.

Lemma 4 *Let m be a positive penetrator node in a*

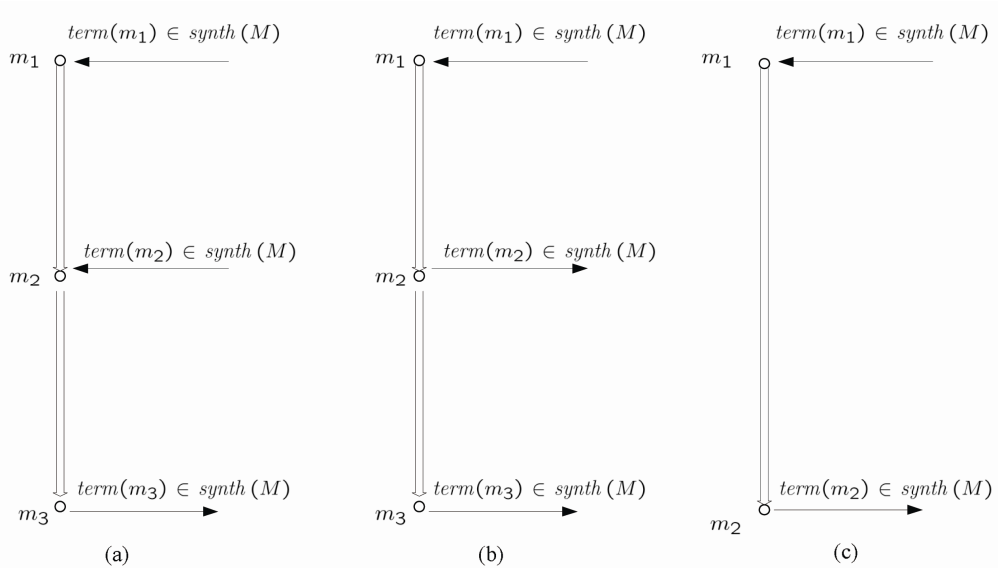


Figure 5. Penetrator's knowledge closure property.

bundle $\mathcal{B}$, *a be an atomic message that uniquely originates at a regular node* n, M *be a message set such that* $suite(M,a,m,n,\mathcal{B})$, *and* $term(m') \in synth(M)$ *for any node such that* $m' \Rightarrow^{+} m$, *then* $term(m) \in synth(M)$.

Proof. For convenience, the assumption that $term(m) \in synth(M)$ for any node such that $m \Rightarrow^{+} n$ is referred as (1) in the proof as follows.

By case analysis on the form of penetrator strand, we can easily exclude the cases when m is in a strand M_g, K_K. If thus, we can conclude that a originates at m. This contradicts with the fact that uniquely originates at a regular node n. Therefore, m is in a strand i such that i is $C_{g,h}$, $S_{g,h}$, $E_{h,K}$, $D_{h,K}$, or $KC_{K,h}$.

Case 1: i is in $C_{g,h}$, then $index(m) = 2$, $term(i,0) = g$, $term(i,1) = h$, and $term(m) = \{|g,h|\}$ for some g, h, and $sign(i,0) = -$, and $sign(i,1) = -$. From the assumption (1), we have $term(i,0) \in synth(M)$ and $term(i,1) \in synth(M)$, then $g \in synth(M)$ and $h \in synth(M)$; By the definition of *synth* operator, $\{|g,h|\} \in synth(M)$, then $term(m) \in synth(M)$.

Case 2: i is in $S_{g,h}$, then $index(m) = 1$, or $index(m) = 2$, $term(i,0) = \{|g,h|\}$, $term(i,1) = g$, and $term(m) = h$ for some g, h. From the assumption (1), we have $term(i,0) \in synth(M)$, $\{|g,h|\} \in synth(M)$, by Lemma 4, we have $g \in synth(M)$ and $h \in synth(M)$. So $term(m) \in synth(M)$.

Case 3: i is in $E_{h,K}$, then $index(m) = 2$, $term(i,0) = K$, $term(i,1) = h$, and $term(m') = \{|h|\}_K$ for some K, h, and $sign(i,0) = -$, and $sign(i,1) = -$. From the assumption (1), $term(i,0) \in synth(M)$ and $term(i,1) \in synth(M)$, then $K \in synth(M)$ and $h \in synth(M)$; by the definition of *synth*, we have $\{|h|\}_K \in synth(M)$, then $term(m) \in synth(M)$.

Case 4: i is in $D_{h,K}$, then $index(m) = 2$, $term(i,0) = K^{-1}$, $term(i,1) = \{|h|\}_K$, and $term(m) = h$ for some K, h, and $sign(i,0) = -$, and $sign(i,1) = -$. From the assumption (1), we have $term(i,0) \in synth(M)$ and $term(i,1) \in synth(M)$, therefore $K^{-1} \in synth(M)$ and $\{|h|\}_K \in synth(M)$, by Lemma 4, we have either (4-1) $term(m) = h \in synth(M)$ or (4-2) $\{|h|\}_K \in M$. From (4-1), the lemma can be proved at once. For the case (4-2), there are also two subcases, either (4-2-1) $a \sqsubset \{|h|\}_K$ or (4-2-2) $a \sqsubset \{|h|\}_{K^{-1}}$. From (4-2-1), we have $a \sqsubset h$, by M is a test suite for a in b, so $h \in M$, then $h \in$ **synth** M, then **term** $m' \in$ **synth** M. From (4-2-2), then by M is a test suite for a in b, we have **component** $\{|h|\}_K$ b, then we have $regular(K^{-1},m,\mathcal{B})$. From this and $(i,0) \in \mathcal{B}$ and $term(i,0) = K^{-1}$, then i is regular, but this contradicts with that m is in a penetrator strand.

Case 5: i is in $KC_{K,h}$, then $index(m) = 1$, $term(i,1) = h$, $term(i,0) = \{|h|\}_K$, (2) $term(i,0) + cracktime(K) < term(i,1)$. From the assumption (1), we have $\{|h|\}_K \in synth(M)$. From this, by Lemma 3, we have either (5-1) $h \in synth(M)$ or (5-2) $\{|h|\}_K \in M$. From (5-1), the lemma can be proved at once. For the case (5-2), there are also two subcases, either (5-2-1) $a \sqsubset \{|h|\}_K$ or (5-2-2) $a \sqsubset \{|h|\}_K$. From (5-2-1), we have $a \sqsubset h$, by the definition of $suite(M,a,m,n,\mathcal{B})$, so $h \in M$, then $h \in synth(M)$. From (5-2-2), then by the definition of $suite(M,a,m,n,\mathcal{B})$, we have (3) $time(m) \le time(n) + cracktime(k)$. From $a \sqsubset term(i,0)$, and a uniquely originates at n, we have $time(n) \le time(i,0)$. Then we have

$$time(n) + cracktime(k) \le time(i,0) + cracktime(k),$$

with (3), we have $time(m) \le time(i,0) + cracktime(k)$. But this contradicts with (2).

On the other side, a strand's receiving nodes get messages which are all in $synth(M)$, but a new message, which is not in $synth(M)$, is sent in the strand, then the strand must be regular because a penetrator strand can not create such a term. The result can be simply inferred from Lemma 4.

Lemma 5 *Let mbe a positive node in a bundle* $\mathcal{B}$, *a be an atomic message that uniquely originates at a regular node* n, M *be a message set such that* $suite(M,a,m,n,\mathcal{B})$, *and* $term(m') \in synth(M)$ *for any node such that* $m' \Rightarrow^{+} m$, *and* $term(m) \notin synth(M)$, *then* m *is regular*.

For Lemma 4 and 5, we have two comments:

1) Lemma 4 characterizes the knowledge closure properties of a penetrator's operations on messages. It says that if a penetrator only receives messages in $synth(M)$, where M is a test suite for some atomic message a, then the augmented knowledge of the penetrator is still in $synth(M)$ after the receiving actions.

2) Lemma 5 provides a key technique to prove the authentication guarantee that m is regular. Intuitively, condition (1) of *suite* requires the secrecy of the inverse key k^{-1} for any key k which is used to encrypt any message in M containing a; condition (2) of operator *suite* is a recency restriction that these encrypted messages containing a can not be cracked until m. Therefore this lemma provides a means of using secrecy and recency restriction to prove authentication guarantee. We will see this result is very useful for us to check whether a strand is regular in the next sections.

Note that the two lemmas relates the algebraic operator *synth* in trace theory [4,11] with penetrator's strand ability to deduce knowledge, which is the most important one which differs our work from the classical strand space theory. Such closure properties are not available in the classical strand space theory because message algebra operators such as *synth* are not formalized.

5. Unsolicited Tests

In [12] (Subsection 4.2.3), a negative node n is an unsolicited test for $\{|h|\}_K$, if $\{|h|\}_K$ is a *test component* for any atomic text a in n, and K cannot be penetrated in the strand space. Then an unsolicited test for $\{|h|\}_K$ in a bundle $\mathcal{B}$ can guarantee the existence of a positive regular node of which $\{|h|\}_K$ is a component. We simplify this definition of unsolicited tests by the following two aspects:

1) we consider a node n is an unsolicited test for $\{|h|\}_K$ in a bundle $\mathcal{B}$;

2) we only require that $\{|h|\}_K$ is a subterm of the term of n, and K is regular w.r.t. n in the bundle $\mathcal{B}$ instead of a strand space.

In our formulation, unsolicited authentication test is a kind of regularity about an encrypted term $\{|h|\}_K$, which is a subterm of a node n where K cannot be penetrated before n in a bundle $\mathcal{B}$. Then it can be ensured that there is a positive regular node m originating $\{|h|\}_K$ as a subterm, *i.e.*, m has $\{|h|\}_K$ as a subterm and it also holds that $\{|h|\}_K \sqsubset term(m')$ for any node $m' \preceq_{\mathcal{B}} m$. Intuitively, the reason why m must be regular lies in that K cannot be penetrated before m in $\mathcal{B}$. So the penetrator cannot create $\{|h|\}_K$ by encrypting h with K.

Definition 8 *Given a bundle $\mathcal{B}$. A node n in $\mathcal{B}$ is an unsolicited test for $\{|h|\}_K$ if $\{|h|\}_K \sqsubset term(n)$, and K is regular w.r.t. n in $\mathcal{B}$.*

Lemma 6 (Unsolicited authentication test) *$\mathcal{B}$ is a given bundle. Let n be an unsolicited test for $\{|h|\}_K$. Then there exists a positive regular node m in $\mathcal{B}$ such that $m \preceq_{\mathcal{B}} n$ and $\{|h|\}_K \sqsubset term(m)$ and $\{|h|\}_K \not\sqsubset term(m')$ for any node m' such that $m' \preceq_{\mathcal{B}} m$.*

Proof. Let $P =_{df} \{x \mid x \preceq_{\mathcal{B}} n \wedge \{|h|\}_K \sqsubset term(x)\}$. Obviously, $m \in P$. By Lemma 1, there exists a node m' such that m' is minimal in P, which means that $\{|h|\}_K \sqsubset term(m')$, $m' \preceq_{\mathcal{B}} n$, and for all y such that $y \preceq_{\mathcal{B}} m'$, $y \notin P$. Hence, $\{|h|\}_K \not\sqsubset term(y)$.

First, we prove that the sign of m' is positive by contradiction. If $sign(m') = -$, then by the upward-closed property of a bundle there must be another node m'' in $\mathcal{B}$ such that $sign(m'') = +$ and $m'' \rightarrow m'$. Then we have (a) $m'' \preceq_{\mathcal{B}} m'$ and (b) $term(m') = term(m'')$. By (a) and $m' \preceq_{\mathcal{B}} n$, we have $m'' \preceq_{\mathcal{B}} n$. By (b) and $\{|h|\}_K \sqsubset term(m')$, we have $\{|h|\}_K \sqsubset term(m'')$. Hence, $m'' \in P$ which contradicts with the minimality of m'.

Second, we prove that m' is regular. We show that a contradiction can be derived if m' is in a penetrator strand. Here, we only analyze cases when m' is in either $C_{g,g'}$ (concatenation strand), $E_{g,K'}$ (encryption strand), or $KC_{K',g}$ (key crack strand). Other cases are either straightforward or can be analyzed in a similar way.

- m' is in $i \in C_{g,g'}$.

By the form of the strand $C_{g,g'}$ and the fact that m' is a positive node, we have $m' = (i,2)$, $term(m') = \{|g, g'|\}$, $term(i,0) = g$, and $term(i,1) = g'$ for some g, g'. By the upwards-closed property of a bundle, we have that nodes $(i,0)$ and $(i,1)$ must be in $\mathcal{B}$. By $\{|h|\}_K \sqsubset \{|g, g'|\}$, we have either $\{|h|\}_K \sqsubset g$ or $\{|h|\}_K \sqsubset g'$, *i.e.* $\{|h|\}_K \sqsubset term(i,0)$ or $\{|h|\}_K \sqsubset term(i,1)$. So either node $(i,0) \in P$, or node $(i,1) \in P$. Both cases contradict with the minimality of m'.

- m' is in $i \in E_{g,K'}$.

By the form of the strand $E_{g,K'}$ and the fact that m' is a positive node, we have $m' = (i,2)$, $term(m') = \{|g|\}_{K'}$, $term(i,0) = K'$, and $term(i,1) = g$ for some g and K'. So $\{|h|\}_K \sqsubset \{|g|\}_{K'}$. Then it is straightforward that either (1) $\{|h|\}_K \sqsubset g$ or (2) $h = g$ and $K = K'$. For the first case, we have $\{|h|\}_K \sqsubset term(i,1)$. It is easy to derive a contradiction by the same argument as before. For the second case, by the definition of the relation $\Rightarrow$, we have (a) $time(i,0) \le time(i,2)$. And by definition of P, we also have (b) $time(m') \le time(n)$. Hence, $time(i,0) \le time(n)$. However, by the assumption that K must be regular w.r.t. n in $\mathcal{B}$, $term(i,0)$ must be regular, and this contradicts with the fact that i is a penetrator strand.

- m' is in $i \in KC_{K',g}$.

By the form of the strand $KC_{K',g}$, and the fact that m' is a positive node, we have $m' = (i,1)$, $term(m') = g$, $term(i,0) = \{|g|\}_{K'}$ for some g and K', and

$$time(i,0) + cracktime(K) < time(m').$$

By $\{|h|\}_K \sqsubset term(m') = g$, so $\{|h|\}_K \sqsubset term(i,0) = \{|g|\}_{K'}$. Obviously $(i,0) \preceq_{\mathcal{B}} m' \preceq_{\mathcal{B}} n$. So $(i,0) \in P$, which contradicts with the minimality of m'.

The proof totally depends on the well-founded induction principle on bundles, and we have formalized the proof of this lemma in Isabelle/HOL in our inductive strand space model, and the proof scripts are available at [10]. In fact, lemma 6 provides a useful proof method to reason about authentication properties basing on secrecy properties. Note that the premise that n is an unsolicited test for $\{|h|\}_K$ requires that K is regular w.r.t. n in $\mathcal{B}$, which is an assumption on the secrecy of K. And the conclusion is an authentication guarantee of the existence of a regular node m. Besides, compared with the original version of unsolicited test, our result also has two extensions that $m \preceq_{\mathcal{B}} n$ and m is minimal (*i.e.*, $\{|h|\}_K \not\sqsubset term(m')$ for any node m' such that $m' \preceq_{\mathcal{B}} m$). We find that the extended version of unsolicited authentication test is quite useful in many cases, especially in

the verification of authentication properties of symmetric key based protocols. In [13], we have used a version of unsolicited authentication test in the classical strand space theory to give new proofs of authentication properties of the Otway-Rees protocol. In this work, we have successfully applied unsolicited authentication test to our study of the Kerberos V protocol in the next paper.

6. Conclusions and related Work

This work is an extension of [14]. We have added two new semantical features in our new framework: timestamp and protocol mixture. In essence, our treatment of timestamps is to add a global clock to the underlying execution model, and to extend every action by a temporal annotation. This allows us to align the timestamps sent in the protocol messages with the actual occurrence times of the corresponding actions. Although it is quite straightforward, it gives a powerful mechanism to reason about recency of a message. For protocol mixture, we admit a realistic assumption that a regular agent can start multiple parallel secondary sessions once he has finished a primary protocol session, and he holds all the information of the primary protocol session when he begins a secondary protocol session. So we introduce a causal relation $\mapsto$ between strands to model the protocol dependency. The above two semantical features are seldom discussed in previous works of strand space literature.

Despite the aforementioned extensions in semantics, the definition of a bundle, which is the cornerstone of the strand space theory, remains unchanged. So the induction principle on the well-foundedness of a bundle is still effective in our model. Based on this principle, we have proved an extended result of the unsolicited authentication test.

In the literature, most of the existing approaches for protocol analysis have not concentrated on timestamps and replay attacks. These include the CSP model-checking approach [15], the rank functions [16], and the Multi-Set Rewriting formalism (MSR) [17]. Paulson and Bella's inductive method [4,11] is one exception. They not only have extended their method to model replay attacks, but also have succeeded in applying their method to the Yahalom protocol and the Kerberos IV protocol. Recently, Bozga *et al.* [18] proposed an approach based on timed automata, symbolic verification techniques and temporal logic to analyze security protocols with timestamps. But they haven't applied their approach to any real-world security protocols.

For protocol mixture, there have been a few works to reason rigorously about protocol interactions. For instance, Meadows studied the Internet Key Exchange protocol, emphasizing the potential interactions among its specific sub-protocols [19]. The analysis work was conducted in the NRL protocol analyzer. Recently, Cremers discussed the feasibility of multi-protocol attacks, and his work is done in the operational semantical framework which considers a so-called type flaw attacks [20]. All these works, including [7], focus on protocol interactions by message exchanging. Instead, our work emphasizes on the dependency between a primary protocol session and a secondary protocol session. Here we assume that when a regular agent starts a secondary protocol session, he should be aware that he has finished a corresponding primary protocol session, and he maintains all the information obtained in the primary protocol session, such as tickets and the creation time of the tickets. These modelling assumptions fit well with the real-world environments where the Kerberos protocols run.

7. References

[1] F. Javier Thayer, J. C. Herzog and J. D. Guttman, "Strand Spaces: Proving Security Protocols Correct," *Journal of Computer Security*, Vol. 7, No. 1, 1999, pp. 191-230.

[2] S. P. Miller, J. I. Neuman, J. I. Schiller and J. H. Saltzer, "Kerberos Authentication and Authorisation System," Technical Report, Technical Plan Section E.2.1, MIT, Athena, 1989.

[3] K. R. C. Neuman and S. Hartman, "The Kerberos Network Authentication Service (v5)," Technical report, Internet RFC 4120, July 2005.

[4] G. Bella, "Inductive Verification of Cryptographic Protocols," PhD thesis, Cambridge University Computer Laboratory, 2000.

[5] J. D. Guttman, "Key Compromise, Strand Spaces, and the Authentication Tests," *Proceedings of 7th Conference on the Mathematical Foundations of Programming Semantics*, ENTCS 45, 2001, pp. 1-21.

[6] D. Denning and G. Sacco, "Timestamps in Key Distribution Protocols," *Communications of the ACM*, Vol. 24, No. 8, 1981, pp. 533-536.

[7] F. Javier Thayer, J. C. Herzog and J. D. Guttman, "Mixed Strand Spaces," *Proceedings of* 12*th IEEE Computer Security Foundations Workshop*, 1999, pp. 72-82.

[8] J. D. Guttman and F. Javier Thayer, "Protocol Independence through Disjoint Encryption," *Proceedings of* 13*th IEEE Computer Security Foundations Workshop*, 2000, pp. 24-34.

[9] T. Nipkow, L. C. Paulson and M. Wenzel, "Isabelle/HOL—A Proof Assistant for Higher-Order Logic," LNCS 2283. Spinger, 2002.

[10] Y. Li, "Strand Space and Security Protocols". http://lcs.ios.ac.cn/~lyj238/strand.html

[11] L. C. Paulson, "The Inductive Approach to Verifying Cryptographic Protocols," *Journal of Computer Security*, Vol. 6, No. 1-2, 1998, pp. 85-128.

[12] J. D. Guttman and F. Javier Thayer, "Authentication

Tests and the Structure of Bundles," *Theoretical Computer Science*, Vol. 283, No. 2, 2002, pp. 333-380.

[13] Y. Li and J. Pang, "Generalized Unsolicited Tests for Authentication Protocol Analysis," *Proceedings of* 7*th Conference on Parallel and Distributed Computing*, 2006, pp. 509-514.

[14] Y. Li, "The Inductive Approach to Strand Space," *Proceedings of* 25*th IFIP Conference on Formal Techniques for Networked and Distributed Systems*, LNCS 3731, 2005, pp. 547-552.

[15] G. Lowe, "Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR," *Proceedings of* 2*nd International Conference on Tools and Algorithms for the* 10 *Construction and Analysis of Systems*, LNCS 1055, pages 147-166, 1996.

[16] J. Heather and S. A. Schneider, "Toward Automatic Verification of Authentication Protocols on an Unbounded Network," *Proceedings of* 13*th IEEE Computer Security Foundations Workshop*, 2000, pp. 132-143.

[17] F. Butler, I. Cervesato, A. Jaggard and A. Scedrov, "A Formal Analysis of Some Properties of Kerberos 5 Using MSR," *Proceedings of* 15*th IEEE Computer Security Foundations Workshop*, 2002, 175-190.

[18] L. Bozga, C. Ene and Y. Lakhnech, "A Symbolic Decision Procedure for Cryptographic Protocols with Time Stamps," *Journal of Logic and Algebraic Programming*, Vol. 65, No. 1, 2005, pp. 1-35.

[19] C. Meadows, "Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer," *Proceedings of* 12*th IEEE Computer Security Foundations Workshop*, 1999, pp. 216-231.

[20] C. J. F. Cremers, "Feasibility of Multi-Protocol Attacks," *Proceedings of* 1*st Conference on Availability, Reliability and Security*, 2006, pp. 287-294.

Journal of Information Security, 2010, 1, 56-67
doi:10.4236/jis.2010.12007 Published Online October 2010 (http://www.SciRP.org/journal/jis)

Extending the Strand Space Method with Timestamps: Part II Application to Kerberos V*

Yongjian Li[1,2], Jun Pang[3]
[1]*Chinese Academy of Sciences, Institute of Software Laboratory of Computer Science, Beijing, China*
[2]*The State Key Laboratory of Information Security, Beijing, China*
[3]*University of Oldenburg Department of Computer Science Safety-critical Embedded Systems, Oldenburg, Germany*
E-mail: *lyj238@ios.ac.cn, jun.pang@informatik.uni-oldenburg.de*
Received June 23, 2010; *revised September* 14, 2010; *accepted July* 12, 2010

Abstract

In this paper, we show how to use the novel extended strand space method to verify Kerberos V. First, we formally model novel semantical features in Kerberos V such as timestamps and protocol mixture in this new framework. Second, we apply unsolicited authentication test to prove its secrecy and authentication goals of Kerberos V. Our formalization and proof in this case study have been mechanized using Isabelle/HOL.

Keywords: Strand Space, Kerberos V, Theorem Proving, Verification, Isabelle/HOL

1. Introduction

The first version of Kerberos protocol was developed in the mid eighties as part of project Athena at MIT [1]. Over twenty years, different versions of Kerberos protocols have evolved. Kerberos V (**Figure 1** and **Figure 2**) is the latest version released by the Internet Engineering Task Force (IETF) [2]. It is a password-based system for authentication and authorization over local area networks. It is designed with the following aims: once a client authenticates himself to a network machine, the process of obtaining authorization to access another network service should be completely transparent to him. Namely, the client only needs enter his password once during the authentication phase.

As we introduced in the previous paper [3], there are two novel semantic features in Kerberos V protocol. First, it uses timestamps to prevent replay attacks, so this deficiency of the strand space theory makes it difficult to analyze these protocols. Second, it is divided into three causally related multiple phases: authentication, authorization, and service protocol phases. One phase may be used to retrieve a ticket from a key distribution center, while a second phase is used to present the ticket to a security-aware server. To make matters more complex, Kerbeors uses timestamps to guarantee the recency of these tickets, that is, such tickets are only valid for an interval, and multiple sub-protocol sessions can start in parallel by the same agent using the same ticket if the ticket does not expire. Little work has been done to formalize both the timestamps and protocol mixture in a semantic framework.

The aim of this paper is practical. We hope to apply the extended theory in [3] to the analysis of Kerberos V protocol. Kerberos V is appropriate as our case study because it covers both timestamps and protocol mixture semantical features.

Structure of the Paper: Section 2 briefly introduces the overview of Kerberos V. Section 3 presents the formalization of Kerberos V. Sections 4 and 5 prove its secrecy and authentication goals. We discuss related work and conclude the paper in Section 6.

2. An Overview of Kerberos V

The protocol's layout and its message exchanging are presented in **Figure 1** and **Figure 2** separately. In the infrastructure of the Kerberos V protocol, there is a unique authentication server, and some (not necessarily only one) ticket granting servers. The latter assumption is different from that in [4], where only a unique ticket granting server exists.

*This is a revised and extended version of the homonymous paper appearing in the Proceedings the Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2007, IEEE Computer Society). The main modifications have been made on the presentation of the technical material, with the purpose of having full details. The first author is supported by grants (No.60496321, 60421001) from National Natural Science Foundation of China.

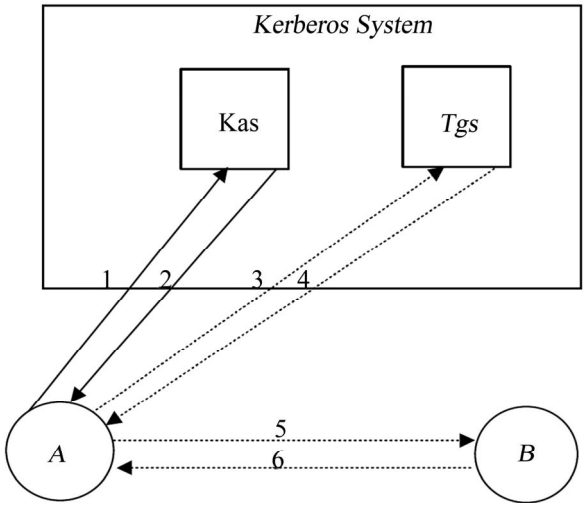


Figure 1. The layout of Kerberos V.

Authentication phase

1. $A \rightarrow Kas: \{|A, Tgs|\}$

2. $Kas \rightarrow A: \underbrace{\{|\{|A, Tgs, authK, Ta|\}_{K_{Tgs}}, \{|A, Tgs, authK, Ta|\}KA|\}}_{authTicket}$

Authorisation Phase

3. $A \rightarrow Tgs: \{|\{|A, Tgs, authK, Ta|\}_{K_{Tgs}}, \{|A, t_2|\}_{authK}, B|\}$

4. $Tgs \rightarrow A: \underbrace{\{|\{|A, B, seruK, T_S|\}_{K_B}, \{|A, B, seruK, T_S|\}_{authK}|\}}_{seruTicket}$

Service Phase

5. $A \rightarrow B: \{|A, B, seruK, T_S|\}_{K_B}, \{|A, t_3|\}_{seruK}|\}$

6. $B \rightarrow A: \{|t_3|\}_{seruK}$

Figure 2. Kerberos V: message exchanging.

In order to access some network service, the client needs to communicate with two trusted servers **Kas** and *Tgs*. **Kas** is an authentication server (or the key distribution center) and it provides keys for communication between clients and ticket granting servers. *Tgs* is a ticket granting server and it provides keys for communication between clients and application servers. The full protocol has three phases each consisting of two messages between the client and one of the servers in turn. Messages 2 and 4 are different from those in Kerberos IV [1,4] in that nested encryption has been cancelled. Later we will show that this change does not affect goals of the protocol.

Detailed explanation about Kerberos V is delayed to Section 2, where the protocol is formalized in strand space model with our extensions. Here we only give an overview of the general principles to guarantee recency, secrecy and authentication in the design of Kerberos V. For recency,

• A regular sender should attach a timestamp to indicate the time when the message is issued; usually such a message is of the form $\{|\ldots,t,\ldots|\}_K$, where t is the time, K may be either a session key or long-term key.

• When a regular receiver the message $\{|\ldots,t,\ldots|\}_K$ first he need be ensured of K's secrecy to guarantee that the message is not froged by the penetrator. Second he check the recency of the message by comparing the timestamp t with the reception time. More formally, if the receiving node is n, then $time(n)$ should be no later than $cracktime(K)+t$, meaning that this message cannot be cracked at $time(n)$, which in turn indicates that the message $\{|\ldots,t,\ldots|\}_K$ is recent.

For an encrypted message $\{|h|\}_K$, the secrecy of a part of the plain message h also comes from both the secrecy of K and the recency of the message $\{|h|\}_K$ itself. That is to say, when a regular receives $\{|h|\}_K$ at time t, it must be ensured that the aforementioned two conditions must be guaranteed until t. From this, we can see that recency and secrecy are closely related with each other in a timed protocol framework.

Unsolicited tests are the main mechanism to guarantee authentication. Because a guarantee of the existence of a regular node can be drawn from an unsolicited test, a regular agent uses unsolicited test to authenticate its regular protocol participant in Kerberos V.

Now let us briefly review the main theoretical results in [3], which will be used in this work. For interesting readers, refer to [3] for preliminary definitions.

If an agent is not a penetrator then his shared key cannot be penetrated, which is formalized as follows:

Axiom 1 *If* $A \notin$ **Bad**, *then* $K_A \notin \mathbf{K}_{\mathcal{P}}$.

Lemma 1 is the main technique used to reason about authentication guarantee of a node n which is an unsolicited test for an encrypted term of the form $\{|h|\}_K$ (e.g., the tickets $\{|A,Tgs,authK,T_a|\}_{K_A}$, $\{|A,t|\}_{authK}$, and so on). That is to say, regular agents can use an unsolicited test with other properties of the protocol to guarantee that the agent who originates the term $\{|h|\}_K$ should be an intended regular agent.

Lemma 1 (Unsolicited authentication test) *$\mathcal{B}$ is a given bundle. Let n be an unsolicited test for $\{|h|\}_K$. Then there exists a positive regular node m in $\mathcal{B}$ such that $m \preceq_{\mathcal{B}} n$ and $\{|h|\}_K \sqsubset term(m)$ and $\{|h|\}_K \not\sqsubset term(m')$ for any node m' such that $m' \preceq_{\mathcal{B}} m$.*

Let a be an atomic message that uniquely originates at some node n, m be a positive penetrator node in a bundle $\mathcal{B}$ such that and $a \sqsubset term(m)$. Suppose M is a test suite for a w.r.t. m in the bundle $\mathcal{B}$. A strand's

receiving nodes get messages which are all in $synth(M)$, but a new message, which is not in $synth(M)$, is sent in the strand, then the strand must be regular because a penetrator strand can not create such a term.

Lemma 2 *Let m be a positive node in a bundle $\mathcal{B}$, a be an atomic message that uniquely originates at a regular node n, M be a message set such that $suite(M, a, m, n, \mathcal{B})$, and $term(m') \in synth(M)$ for any node such that $m' \Rightarrow^{+} m$, and $term(m) \notin synth(M)$, then m is regular.*

We will illustrate these general principles in detail in the next sections when we formalize the semantics and prove secrecy properties of Kerberos V.

3. Formalizing Kerberos V

To model the time for a penetrator to break a message encrypted by a long-term shared key or a session key, we define two constants shrKcracktime and sessionKcrktime. The crack time of any regular agent's long-term shared key is the constant shrKcracktime,

Axiom 2 *$cracktime(K_A)$ = shrKcracktime, for any regular agent A in Kerberos V.*

The crack time of any session key originated by an authentication server is the constant sessionKcrktime.

Axiom 3 *$cracktime(authK)$ = sessionKcrktime, for any session key $authK$ originated by* **Kas**.

The trace tr specifications of the regular strands of Kerberos V (see **Figure 2**) are defined as predicates:[1]

1) Part I (Authentication Phase)

• **Ag-I** $[i_1, A, Tgs, authK, T_a, authTicket, t_0, t_1]$ iff

$$tr(i_1) = \begin{bmatrix} (t_0, +, \{|A, Tgs|\}), \\ (t_1, -, \{|authTicket, \\ \quad \{|A, Tgs, authK, T_a|\}_{K_A}|\}) \end{bmatrix}$$

where $Tgs \in$ **TGSs** and $t_1 - T_a \le$ shrKcracktime.

• **AS** $[as, A, Tgs, authK, t_0, t_1]$ iff

$$tr(as) = \begin{bmatrix} (t_0, -, \{|A, Tgs|\}), \\ (t_1, +, \{|\{|A, Tgs, authK, t_1|\}_{K_{Tgs}}, \\ \quad \{|A, Tgs, authK, t_1|\}_{K_A}|\}) \end{bmatrix}$$

where $Tgs \in$ **TGSs**.

In the first phase, when **Kas** issues the second message

$$\{|\{|A, Tgs, authK, T_a|\}_{K_{Tgs}}, \{|A, Tgs, authK, T_a|\}_{K_A}|\},$$

$authK$ is the session key that will be used for the client A to communicate with a ticket grant server Tgs, **Kas** attaches T_a with the message to indicate when this message is sent; if A receives this message at time t_1, A will check the condition $t_1 - T_a \le$ shrKcracktime to ensure the recency of this message. At the end of this phase, A obtains a ticket $authTicket$ and the session key $authK$ to communicate with Tgs.

2) Part II (Authorization Phase)

• **Ag-II** $[i_2, A, authK, authTicket, B, servK, T_S, servTicket, t_2, t_3]$ iff $\exists i_1, Tgs, T_a, t_0, t_1. i_1 \mapsto i_2 \wedge$ **Ag-I** $[i_1, A, Tgs, authK, T_a, authTicket, t_0, t_1] \wedge$

$$tr(i_2) = \begin{bmatrix} (t_2, +, \{|authTicket, \\ \quad \{|A, t_2|\}_{authK}, B), \\ (t_3, -, \{|servTicket, \\ \quad \{|A, B, servK, T_S|\}_{authK}) \end{bmatrix}$$

where $Tgs \in$ **TGSs** and $t_3 - T_a \le$ shrKcracktime and $t_3 - T_S \le$ sessionKcrktime.

• **TGS** $[tgs, A, Tgs, authK, servK, B, T_a, T_0, t_0, t_1]$ iff

$$tr(tgs) = \begin{bmatrix} (t_0, -, \{|\{|A, Tgs, authK, T_a|\}_{K_{Tgs}}, \\ \quad \{|A, T_{0_{authK}}, B|\}), \\ (t_1, +, \{|\{|A, B, servK, t_1|\}_{K_B}, \\ \quad \{|A, B, servK, t_1|\}_{authK}|\}) \end{bmatrix}$$

where $Tgs \in$ **TGSs,** $B \notin$ **TGSs,** $t_1 +$ sessionKcrktime $\le T_a +$ shrKcracktime.

In the second phase, the situation is more complex. Both Tgs and A need to check whether their received messages are recent by the same mechanism. Furthermore, Tgs also need ensure a side condition that

$$t_1 + \text{sessionKcrktime} \le T_a + \text{shrKcracktime}$$

to guarantee that the application server B only receives a recent service ticket. Informally speaking, this condition means that Tgs can guarantee any $authK$ that he receives can only be compromised later than $servK$ which is associated with the $authK$. We will comment this side condition in analysis in the third phase. At the end of this phase, A obtains a ticket $servTicket$ and the session key $servK$ to communicate with B.

[1]For simplicity, we assume any trace of a regular agent always respects the time order in Kerberos V protocol, and we do not include this side condition in the trace specifications.

3) Part III (Service Phase)

• **Ag-III** $[i_3, A, servK, servTicket, t_4, t_5]$ iff

$\exists i_1, Tgs, authK, T_a, t_0, t_1, i_2, authTicket, B, T_S, t_2, t_3$.

$$tr(i_3) = \left[\begin{array}{l}\left(t_4, +, \left\{\left|servTicket, \left\{\left|A, t_4\right|\right\}_{servK}\right.\right\}\right), \\ \left(t_5, -, \left\{\left|t_4\right|\right\}_{servK}\right)\end{array}\right] \wedge$$

Ag-I $[i_1, A, Tgs, authK, T_a, authTicket, t_0, t_1] \wedge$

Ag-II $[i_2, A, authK, authTicket, B, servK, T_S,$ $servTicket, t_2, t_3] \wedge i_1 \mapsto i_2 \wedge i_2 \mapsto i_3$

where $Tgs \in \mathbf{TGSs}$, $t_5 - T_a \le \text{shrKcracktime}$ and $t_5 - T_S \le \text{sessionKcrktime}$.

• **Apps** $[apps, A, B, servK, T_S, T_4, t_0, t_1]$ iff

$tr(apps) =$

$$\left[\begin{array}{l}\left(t_0, -, \left\{\left|\left\{\left|A, B, servK, T_S\right|\right\}_{K_B}, \left\{\left|A, T_4\right|\right\}_{servK}\right|\right\}\right), \\ \left(t_1, +, \left\{\left|t_1\right|\right\}_{servK}\right)\end{array}\right]$$

where $t_0 - T_S \le \text{sessionKcrktime}$.

In the last phase, it is subtle for the application server B to check the recency of the message $\left\{\left|\left\{\left|A, B, servK, T_S\right|\right\}_{K_B}, \left\{\left|A, T_4\right|\right\}_{servK}\right|\right\}$. From the ticket $\left\{\left|A, B, servK, T_S\right|\right\}_{K_B}$, B knows that Tgs must have issued $\left\{\left|\left\{\left|A, B, servK, T_S\right|\right\}_{K_B}, \left\{\left|A, B, T_4, servK, T_S\right|\right\}_{authK}\right|\right\}$ at time T_S. The potential compromise of $servK$ is from the message $\left\{\left|A, B, servK, T_S\right|\right\}_{authK}$. A penetrator can either directly break $\left\{\left|A, B, servK, T_S\right|\right\}_{authK}$ to obtain $servK$, or have $authK$ first then decrypt the message $\left\{\left|A, B, servK, T_S\right|\right\}_{authK}$ to obtain $servK$. Since $authK$ is also a session key which is originated by **Kas** in an earlier time than T_s, the guarantee for the confidentiality of $authK$ is of extreme importance. The corresponding ticket $\left\{\left|A, Tgs, authK, T_a\right|\right\}_{K_{Tgs}}$ is not available for B, B cannot know the creation time of $authK$. So B cannot directly check whether $authK$ has been compromised. Fortunately, if Tgs can guarantee that any $authK$ which it receives will be compromised later than $servK$, associated with the $authK$, then it is enough for B to check $t_0 - T_S \le \text{sessionKcrktime}$ to ensure that the $authK$ has not been compromised. At the end of this stage, A and B authenticate each other, and A can access the service provided by B.

The authentication server **Kas** must obey the following principles to generate a session key $authK$:

- $authK$ must never be known initially to a penetrator, *i.e.*, $authK \notin \mathbf{K}_{\mathcal{P}}$;
- $authK$ must be uniquely originated;
- $authK$ is a symmetric key;
- $authK$ must not be the same as an agent's long-term shared key.

We summarize these principles as the following axiom:

Axiom 4 *For any authentication server strand* as *such that* $\mathbf{AS}[as, A, Tgs, authK, t_0, t_1]$, *we have* $authK \notin \mathbf{K}_{\mathcal{P}}$, $authK$ *uniquely originates in* $(as, 1)$, $authK = authK^{-1}$, *and* $authK \neq K_B$ *for any agent* B.

A ticket grant server creates the session key $servK$ by three principles, which are similar to those which the authentication server obeys to create the session key $authK$.

Axiom 5 *For any ticket grant server strand* tgs *such that* $\mathbf{TGS}[tgs, A, Tgs, authK, servK, B, T_a, T_0, t_0, t_1]$, $servK \notin \mathbf{K}_{\mathcal{P}}$, $servK$ *uniquely originates in* $(tgs, 1)$, $servK = servK^{-1}$, *and* $servK \neq K_B$ *for any agent* B.

In the following two subsections, we verify the secrecy and authentication properties of Kerberos V. We use similar ways for representing these security properties as in [5]. However, we may need formulate secrecy properties with temporal restrictions when we discuss them in a timed framework. A value v is secret for a protocol if for every bundle $\mathcal{C}$ of the protocol the penetrator cannot receive v in cleartext until some time t; that is, there is no node n in $\mathcal{C}$ such that $term(n) = v$ and $time(n) \le t$. For Kerberos V, we mainly discuss the secrecy of a long-term key of a regular agent, and $authK$, $servK$ issued by servers. Authentication properties are specified as usual: for a participant B (e.g. acting as a responder), for a certain vector of parameters $\vec{x}$, if each time principal B completes a run of the protocol as a responder using $\vec{x}$ supposedly with A, then there is a run of the protocol with A acting as an initiator using $\vec{x}$ supposedly with B. And this is formalized as follows: there is a responder strand $Resp$ $(\vec{x})$ and the i-th node of the strand is in a bundle $\mathcal{C}$, then there is an initiator strand $Init$ $(\vec{x})$ and some j-th node of the initiator strand is in $\mathcal{C}$.

In order to prove the secrecy of a long-term key K_A, we only need use the well-founded induction principle on bundles. But the knowledge closure property on penetrators is needed when we prove the secrecy of some session key $authK$ or $servK$. For instance, in order to prove the secrecy of $authK$, we construct a set

$$M =_{df} \left\{\left\{\left|A, Tgs, authK, T_a\right|\right\}_{K_A}, \left\{\left|A, Tgs, authK, T_a\right|\right\}_{K_{Tgs}}\right\} \cup \left\{t \mid authK \sqsubset t\right\}.$$

We will show that for any node m in a Kerberos bundle $\mathcal{B}$, if $authK \sqsubset term(m)$ and $time(m) \leq$

$$T_a + \text{shrKcracktime},$$

then $term(m)$ must be in $synth(M)$. Intuitively, this fact holds because both the penetrator and regular strands can only emit a message which is in $synth(M)$. The penetrator can not decrypt or crack the messages

$$\{|A, Tgs, authK, T_a|\}_{K_A} \text{ and } \{|A, Tgs, authK, T_a|\}_{K_{Tgs}}$$

until time $T_a + \text{shrKcracktime}$, so it can only synthesize any messages which is in $synth(M)$; except a unique authentication server strand, any other regular strand can not emit any message which has $authK$ as a subterm until that time. But for the authentication server strand, he can only emit

$$\left\{\left|\{|A, Tgs, authK, t_1|\}_{K_{Tgs}}, \{|A, Tgs, authK, t_1|\}_{K_A}\right|\right\}$$

which is still in $synth(M)$. Our formal proof is by contradiction. If not so, by the well-founded induction principle on $\mathcal{B}$, we have a minimal element m such that $authK \sqsubset term(m)$ and $term(m) \notin synth(M)$. By the knowledge closure property, we can exclude the cases when m is in a penetrator strand. By case analysis on the form of the trace of regular strands, we can also exclude the case when m is in a regular strand. Thus, a contradiction is concluded.

In the following two sections, we give the detailed proof on the secrecy and authentication properties to show how to apply the proof techniques aforementioned. Note that we also have formalized all the proofs in Isabelle/HOL, and the proof scripts can be obtained at [6]. The paper proof here can be viewed as a text account of the mechanical proof scripts at [6].

4. Proving Secrecy Goals

In Kerberos V, a long-term key of a regular agent is never sent in the network, so it cannot be compromised. Let $\mathcal{B}$ be a bundle of Kerberos V. For any node in the bundle, the long-term key of a regular agent cannot be a part of the term of the node. In order to prove this lemma, we only need the well-founded induction principle on bundles.

Lemma 3 *Let* $n \in \mathcal{B}$. *If* $A \notin$ **Bad**, *then* $term(n) \neq K_A$.

Proof. Let

$$P =_{df} \{x \mid x \in \mathcal{B} \wedge K_A \sqsubset term(x)\}$$

We show that P is empty by contradiction. If there is a node $n' \in P$, then by the well-foundedness of a bundle, there exists a node m such that m is minimal in P. Namely, $m \in \mathcal{B}$, $K_A \sqsubset term(m)$, and for all $m' \in \mathcal{B}$, if $m' \preceq_{\mathcal{B}} m$ then $K_A \not\sqsubset term(m')$.

We prove that the sign of m is positive. If $sign(m) = -$, then by upward-closed property of a bundle there must be another node m'' in the bundle $\mathcal{B}$ such that $sign(m'') = +$ and $m'' \rightarrow m$. This contradicts with the minimality of m. Then m is either in a regular strand or in a penetrator strand.

• CASE 1: m is in a regular strand.

There are six cases. Here we only analyze the cases when m is in an authentication server strand $as \in$ **AS** $[as, A, Tgs, authK, t_0, t_1]$ or m is in a client strand $i \in$ **Ag-II** $[i, A, authK, authTicket, B, servK, T_S, servTicket, t_2, t_3]$. The other cases are either straightforward or can be analyzed in a similarly.

If m is in an authentication server strand such that $as \in$ **AS** $[as, A, Tgs, authK, t_0, t_1]$. By inspection on the trace form of the strand, we have $m = (as, 1)$, $K_A \sqsubset term(as, 1)$, and

$$term(as, 1) = \left\{\left|\{|A, Tgs, authK, t_1|\}_{K_{Tgs}}, \{|A, Tgs, authK, t_1|\}_{K_A}\right|\right\},$$

then

$$K_A \sqsubset \{|A, Tgs, authK, t_1|\}_{K_{Tgs}} \text{ or } K_A \sqsubset \{|A, Tgs, authK, t_1|\}_{K_A}.$$

In both cases, we can conclude that $K_A = authK$. But this contradicts with Axiom 4. If m is in a client strand such that $i \in$ **Ag** $-$ **II**$[i, A, authK, authTicket, B, servK, T_S, servTicket, t_2, t_3]$. By inspection on the trace form of the client strand, we have $m = (i, 0)$, $K_A \sqsubset term(i, 0)$, and

$$term(i, 0) = \left\{\left|authTicket, \{|A, t_2|\}_{authK}, B\right|\right\},$$

then $K_A \sqsubset authTicket$. But by the definition of the client strand, there exists some client strand i_1 suchthat $i_1 \mapsto i$ and **Ag-I** $[i_1, A, Tgs, authK, T_a, authTicket, t_0, t_1]$. From the definition of the strand , we have $authTicket \sqsubset term(i_1, 1)$. From this and $K_A \sqsubset authTicket$, we have (1) $K_A \sqsubset term(i_1, 1)$. From $i_1 \mapsto i$, we have (2) $(i_1, 1) \Rightarrow (i, 0)$. From (1) and (2), we can conclude that m is not minimal in P. This contradicts with the minimality of m.

• CASE 2: m is in a penetrator strand p.

Here we only analyze the cases when p is either K_K (key strand) or $C_{g,h}$ (concatenation). Other cases are either straightforward or can be analyzed in a similar way.

- p is K_K. We have $m = (p, 0)$ and $K_A \sqsubset K$. Then $K_A = K \in \mathbf{K}_P$. This contradicts with Axiom 1.
- p is $C_{g,h}$. We have $m' = (p, 2)$ and $K_A \sqsubset \{|g, h|\}$. By the definition of $\sqsubset$, we have $K_A \sqsubset g$, or $K_A \sqsubset h$. If $K_A \sqsubset g$, then $K_A \sqsubset term(p, 0)$. This contradicts with the minimality of m. The case when $K_A \sqsubset h$ can be analyzed similarly.

If an authentication ticket $\{|A,Tgs,authK,T_a|\}_{K_A}$ or $\{|A,Tgs,authK,T_a|\}_{K_{Tgs}}$ occurs as a subterm of a node in $\mathcal{B}$, A is not compromised, and Tgs is a ticket granting server, then it can be guaranteed that there must be an authentication server strand as in which $\{|A,Tgs,authK,T_a|\}_{K_A}$ and $\{|A,Tgs,authK,T_a|\}_{K_{Tgs}}$ originate at time T_a. Therefore, T_a is the earliest time when $\{|A,Tgs,authK,T_a|\}_{K_A}$ and $\{|A,Tgs,authK,T_a|\}_{K_{Tgs}}$ occur in $\mathcal{B}$. With the specification of the origination of the key $authK$ by **Kas** (formulated by Axiom 4), we also are ensured that T_a is the earliest time when $authK$ occurs in $\mathcal{B}$. The minimal property of T_a will be used in the proof of Lemma 5.

Lemma 4 *Let* $n \in \mathcal{B}$, $A \notin$ **Bad**, *and* $Tgs \in$ **TGSs**. *If* $\{|A,Tgs,authK,T_a|\}_{K_A}$ *or* $\{|A,Tgs,authK,T_a|\}_{K_A} \sqsubset term(n)$, *then there exists an authentication server strand* as *such that* **AS**$[as, A, Tgs, authK, t_0, T_a]$ *for some* t_0, $(as,1) \in \mathcal{B}$, *and* $T_a \le time(n)$.

Proof.

Here we only prove the case $\{|A,Tgs,authK,T_a|\}_{K_A} \sqsubset term(n)$. The other case can be proved in a similar way. First we prove that (1) n is an unsolicited test for the term $\{|A,Tgs,authK,T_a|\}_{K_A}$ We only need prove that K_A must be regular *w.r.t.* n. By Lemma 3, there is no node m in $\mathcal{B}$ such that $term(m) = K_A$, so K_A must be regular *w.r.t.* n.

From (1), by Lemma 1, there exists a positive regular node m in $\mathcal{B}$ such that $m \preceq_{\mathcal{B}} n$ and $\{|A,Tgs,authK,T_a|\}_{K_A} \sqsubset term(m)$ and $\{|A,Tgs,authK,T_a|\}_{K_A} \not\sqsubset term(m')$ for any node m' such that $m' \preceq_{\mathcal{B}} m$.

From $m \preceq_{\mathcal{B}} n$ and $\mathcal{B}$ is a bundle, we can easily conclude $time(m) \le time(n)$ and $m \in \mathcal{B}$.

Now we prove that m must be in an authentication server strand. From the fact that m is regular, then we have six cases, here we select two cases when m is in an authentication server strand as such that **AS** $[as, A', Tgs', authK', t_0, t_1]$ or in an ticket granting server strand tgs such that **TGS**$[tgs, A', Tg, authK', servK, B', T_a', T_0, t_0, t_1]$.

• m is in an authentication server strand as such that **AS** $[as, A', Tgs', authK', t_0, t_1]$. By inspection on the form of the strand, $m = (as,1)$ because m is positive. Obviously

$$term(m) = \left\{\left|\left\{|A', Tgs', authK', t_1|\right\}_{K_{Tgs'}}, \left\{|A', Tgs', authK', t_1|\right\}_{K_{A'}}\right|\right\}$$

By $\{|A,Tgs,authK,T_a|\}_{K_A} \sqsubset term(m)$, we have either (2) $\{|A,Tgs,authK,T_a|\}_{K_A} \sqsubset \{|A', Tgs', authK', t_1|\}_{K_{Tgs'}}$ or (3) $\{|A,Tgs,authK,T_a|\}_{K_A} \sqsubset \{|A', Tgs', authK', t_1|\}_{K_{A'}}$. From (2), we have $A = A'$ and $Tgs = Tgs'$ and $authK = authK'$ and $T_a = t_1$, so **AS**$[as, A, Tgs, authK, t_0, T_a]$. Case (3) can be prove similarly.

• m is in an ticket granting server strand such that **TGS** $[tgs, A', Tg, authK', servK, B', T_a', T_0, t_0, t_1]$. By inspection on the form of the strand, $m = (tgs,1)$ because m is a positive node. Obviously

$$term(m) = \left\{\left|\left\{|A', B', servK, t_1|\right\}_{K_{B'}}, \left\{|A', B', servK, t_1|\right\}_{authK'}\right|\right\}.$$

From $\{|A,Tgs,authK,T_a|\}_{K_A} \sqsubset term(m)$, we have either (2) $\{|A,Tgs,authK,T_a|\}_{K_A} \sqsubset \{|A', B', servK, t_1|\}_{K_{B'}}$ or (3) $\{|A,Tgs,authK,T_a|\}_{K_A} \sqsubset \{|A', B', servK, t_1|\}_{authK'}$. From (2), we can prove that $Tgs = B'$, then by the assumption $Tgs \in$ **TGSs**, we have $B' \in$ **TGSs**. But by the definition of the ticket granting server, we have $B' \notin$ **TGSs**. Therefore a contradiction is obtained. Case (3) can be proved similarly.

Once the authentication tickets $\{|A,Tgs,authK,T_a|\}_{K_A}$ or $\{|A,Tgs,authK,T_a|\}_{K_{Tgs}}$ are created by the authentication server **Kas** at T_a, then the session key $authK$ will be not compromised until the time $T_a + \text{shrKcracktime}$.

Lemma 5 *Let* $n \in \mathcal{B}$, $A \notin$ **Bad**, *and* $Tgs \in$ **TGSs**. *If*

$$\{|A,Tgs,authK,T_a|\}_{K_A} \text{ or}$$

$$\{|A,Tgs,authK,T_a|\}_{K_{Tgs}} \sqsubset term(n),$$

then for any node $m \in \mathcal{B}$ *such that* $time(m) \le T_a + \text{shrKcracktime}$, $term(m) \ne authK$.

Proof. First we define two sets.

$$M =_{df} \left\{\begin{matrix}\{|A,Tgs,authK,T_a|\}_{K_A},\\ \{|A,Tgs,authK,T_a|\}_{K_{Tgs}}\end{matrix}\right\} \cup \{t \mid authK \sqsubset t\}$$

$$P =_{df} \{m.m \in \mathcal{B} \wedge time(m) \le T_a + \text{shrKcracktime} \wedge term(m) \notin synth(M)\}.$$

We show that for any node $m \in \mathcal{B}$ such that $time(m) \le T_a + \text{shrKcracktime}$, $term(m) \in synth(M)$. In order to prove this, we only need show P is empty. We prove the assertion by contradiction. If P is not empty, then by the well-foundedness of a bundle, (1) there exists a positive node m such that $m \in \mathcal{B}$, $term(m) \notin synth(M)$, $time(m) \le T_a + \text{shrKcracktime}$, and for all $m' \in \mathcal{B}$, if $m' \preceq_{\mathcal{B}} m$ then $term(m) \in synth(M)$.

First from the fact that $\{|A,Tgs,authK,T_a|\}_{K_A}$ or $\{|A,Tgs,authK,T_a|\}_{K_{Tgs}} \sqsubset term(n)$ by the Lemma 4, then

there exists an authentication server strand as such that **AS**$[as, A, Tgs, authK, t_0, T_a]$ for some t_0, $(as,1) \in \mathcal{B}$, and $T_a \le time(n)$. From the definition of **AS** and Axiom 4, we have (2) $authK$ uniquely originates at $(as,1)$ and $time(as,1) = T_a$.

Next we prove that (3) **suite(**M, $authK$, m, $(as,1)$, $\mathcal{B}$). Here we need show both $\{|A, Tgs, authK, T_a|\}_{K_A}$ and $\{|A, Tgs, authK, T_a|\}_{K_{Tgs}}$ are components in $\mathcal{B}$. From $A \notin$ **Bad** and $Tgs \in$ **TGSs**, by Lemma 3, we have that neither K_A nor K_{Tgs} is compromised, and they are symmetry, therefore $regular(K_A^{-1}, m, \mathcal{B})$ and $regular(K_{Tgs}^{-1}, m, \mathcal{B})$; furthermore from $time(m) \le T_a +$ shrKcracktime, and by Axiom 2, $cracktime(K_A) =$ shrKcracktime, with (2), we have $time(m) \le time(as,1) + cracktime(K_A)$, similarly we have $time(m) \le time(as,1) + cracktime(K_{Tgs})$, so (3) is proved.

From (1), we have for any m' such that $m' \Rightarrow^+ m$, $term(m') \in synth(M)$. With (2)(3), by Lemma 2, we have m must be in a regular strand i, then there exist six cases. Here we analyze the cases when **AS** $[i, A', Tgs', authK', t_0', t_1]$, other cases are more simpler. If m is in an authentication server strand **AS** $[i, A', Tgs', authK', t_0', t_1]$. By inspection on the form of the strand, $m = (i,1)$ because m is positive. Obviously

$$term(m) = \left\{\left|\left\{|A', Tgs', authK', t_1|\right\}_{K_{Tgs'}}, \left\{|A', Tgs', authK', t_1|\right\}_{K_{A'}}\right|\right\}.$$

Obviously $authK \sqsubset term(m)$, otherwise $term(m) \in M$. Therefore $authK \sqsubset \{|A', Tgs', authK', t_1|\}_{K_{A'}}$ or $authK \sqsubset \{|A', Tgs', authK', t_1|\}_{K_{Tgs'}}$ then $authK = authK'$. From the definition of Axiom 4, we have $authK$ uniquely originates from the strand i. Combining with (2), we have $as = i$, then $A = A'$, $Tgs = Tgs'$, $t_1 = T_a$, so

$$term(m) = \left\{\left|\left\{|A, Tgs, authK, T_a|\right\}_{K_{Tgs}}, \left\{|A, Tgs, authK, T_a|\right\}_{K_A}\right|\right\} \in synth(M)$$

This contradicts with the fact $term(m) \notin synth(M)$.

Therefore for any node $m \in \mathcal{B}$ such that $time(m) \le T_a +$ shrKcracktime, $term(m) \in synth(M)$. Next we only need prove that $authK \notin synth(M)$. We prove by contradiction, if $authK \in synth(M)$, by the rule inversion of definition of $synth$, we have $authK \in M$, this contradicts with the definition of M.

In order to prove the conclusion of Lemma 5, we need the conclusion of Lemma 4, which ensures us that a penetrator cannot crack the term $\{|A, Tgs, authK, T_a|\}_{K_A}$ (or $\{|A, Tgs, authK, T_a|\}_{K_{Tgs}}$) to obtain $authK$. Because the earliest time when $authK$ occurs in $\mathcal{B}$ is T_a and $authK$ can only occur in $\{|A, Tgs, authK, T_a|\}_{K_A}$ (or $\{|A, Tgs, authK, T_a|\}_{K_{Tgs}}$) the penetrator cannot crack such a term until $T_a +$ shrKcracktime, and what he can only do is to synthesize some term from M. Therefore, $authK$ must be safe until that time. Furthermore, the intermediate result of this proof tells us that $term(m)$ must be in $synth(M)$ for any node $m \in \mathcal{B}$ such that $time(m) \le T_a +$ shrKcracktime.

If both the tickets

$$\{|A, B, servK, T_S|\}_{authK} \text{ and } \{|A, Tgs, authK, T_a|\}_{K_A}$$

occur as a part of the term of a node in $\mathcal{B}$, A and B are not compromised, and B is not a ticket grant server, and $authK$ is still not compromised at the time when the above two tickets occur, then it can be guaranteed that A must have passed the first and second phases of the protocol, and a ticket grant server strand tgs must exist in $\mathcal{B}$, where two tickets

$$\{|A, B, servK, T_S|\}_{K_B} \text{ and } \{|A, B, servK, T_S|\}_{authK}$$

are issued for some session key $authK$. Similar to Lemma 4, this lemma ensures us that T_S is the earliest time when $servK$ occurs in $\mathcal{B}$, and this minimal property is needed in the proof of Lemma 7.

Lemma 6 *Let* $m, n \in \mathcal{B}$, $A \notin$ **Bad**, $Tgs \in$ **TGSs**. *If both*

$$\{|A, Tgs, authK, T_a|\}_{K_A} \sqsubset term(m),$$

and

$$\{|A, B, servK, T_S|\}_{authK} \sqsubset term(n)$$

and $time(n) \le T_a +$ shrKcracktime, *then there exists a ticket granting server strand* tgs *such that* $TGS[tgs, A, Tgs, authK, servK, B, T_a, T_0, t_0, T_S]$ *for some* T_0, t_0, $(tgs,1) \in \mathcal{B}$ *and* $T_S \le time(n)$.

Here we only give the proof sketch of this lemma. First we need show that (1) n is an unsolicited test for $\{|A, B, servK, T_S|\}_{authK}$ in $\mathcal{B}$. We need prove $regular(authK, n, \mathcal{B})$. This can be ensured by Lemma 5. Because $time(n) \le T_a +$ shrKcracktime, we have $term(n') \ne authK$ for any node n' such that $time(n') \le time(n)$. From (1), we can show that there is a regular node n' such that $n' \preceq_{\mathcal{B}} n$ and

$$\{|A, B, servK, T_S|\}_{authK} \sqsubset term(n') \text{ and}$$

$\{|A, B, servK, T_S|\}_{authK} \not\sqsubset term(m')$ for any node m' such that $m' \preceq_{\mathcal{B}} n$. By the case analysis on the form of regular strands, we can prove that n must be in a ticket granting server strand *tgs* such that **TGS**[*tgs*, *A*, *Tgs*, *authK*, *servK*, *B*, T_a, T_0, t_0, T_S] for some T_0, t_0, $(tgs,1) = n$ and $T_S = time(n)$.

Moreover, by the fact a ticket $\{|A, B, servK, T_S|\}_{authK}$ is originated at time T_S, then the session key *servK* will not be compromised until the time

$$T_S + \text{sessionKcrktime}.$$

Because during the interval from T_S to

$$T_S + \text{sessionKcrktime},$$

neither $\{|A, B, servK, T_S|\}_{authK}$ will be cracked, nor the session key *authK* can be obtained by a penetrator to decrypt the ticket $\{|A, B, servK, T_S|\}_{authK}$.

Lemma 7 *Let* $m_0, n \in \mathcal{B}$, $A, B \notin$ **Bad**, $Tgs \in$ **TGSs**. *If both* $\{|A, Tgs, authK, T_a|\}_{K_A} \sqsubset term(m_0)$ *and*

$$\{|A, B, servK, T_S|\}_{authK} \sqsubset term(n),$$

and $time(n) \le T_a + \text{shrKcracktime}$, *then for any node* $m \in \mathcal{B}$ *such that* $time(m) \le T_S + \text{sessionKcr ktime}$, $term(m) \ne servK$.

Proof. First we define:

$$M =_{df} \left\{ \begin{matrix} \{|A, B, servK, T_S|\}_{K_B}, \\ \{|A, B, servK, T_S|\}_{authK} \end{matrix} \right\} \cup \{t \mid servK \sqsubset t\}.$$

We will show that for any node $m \in \mathcal{B}$ such that $time(m) \le T_S + \text{sessionKcrktime}$, $term(m) \in synth(M)$. We prove the assertion by contradiction.

Let

$P =_{df} \{m.\ m \in \mathcal{B} \wedge time(m) \le T_S + \text{sessionKcrktime} \wedge term(m) \notin synth(M)\}$. If P is not empty, then by the well-foundedness of a bundle, (1) there exists a positive node m such that $m \in \mathcal{B}$, $time(m) \le T_S + \text{sessionKcrktime}$, $term(m) \notin synth(M)$, and for all $m' \in \mathcal{B}$, if $m' \preceq_{\mathcal{B}} m$ then $term(m') \in synth(M)$.

From the fact $\{|A, Tgs, authK, T_a|\}_{K_A} \sqsubset term(m_0)$, by Lemma 4, there exists an authentication server strand *as* such that **AS**[*as*, *A*, *Tgs*, *authK*, t_0, T_a] for some t_0, $(as,1) \in \mathcal{B}$. From the definition of **AS**, we know (2) *authK* uniquely originates at $(as,1)$ and $time(as,1) = T_a$.

From the fact that $\{|A, Tgs, authK, T_a|\}_{K_A} \sqsubset term(m_0)$ and $\{|A, Tgs, authK, T_a|\}_{K_A} \sqsubset term(n)$, by Lemma 6, then there exists a ticket granting server strand *tgs* such that **TGS**[*tgs*, *A*, *Tgs*, *authK*, *servK*, *B*, T_a, T_0, t_0, T_S] for some T_0, t_0. From the definition of **TGS** and Axiom 5, we have (3) *servK* uniquely originates at $(tgs,1)$, $time(tgs,1) = T_S$,

$$\{|A, Tgs, authK, T_a|\}_{K_{Tgs}} \sqsubset term(tgs,0),$$

and $T_S + \text{sessionKcrktime} \le T_a + \text{shrKcracktime}$. From $\{|A, Tgs, authK, T_a|\}_{K_{Tgs}} \sqsubset term(tgs,0)$, by Lemma 4, we can easily conclude that $T_a \le time(tgs,0)$, then (4) $T_a \le time(tgs,1)$.

Next we prove that (5) $suite(M, servK, m, (tgs,1), \mathcal{B})$. Here we need show both $\{|A, Tgs, servK, T_a|\}_{K_B}$ and $\{|A, B, servK, T_S|\}_{authK}$ are components in $\mathcal{B}$. From $B \notin$ **Bad**, by Lemma 3, we have K_B are never compromised, similar to counterpart in Lemma 5, we can prove that $regular(K_B^{-1}, m, \mathcal{B})$. From

$$T_S + \text{sessionKcrktime} \le T_a + \text{shrKcracktime}$$

and $time(m) \le T_S + \text{sessionKcrktime}$, we have (6) $time(m) \le T_a + \text{shrKcracktime}$, with (4), we have $time(m) \le time(tgs,1) + cracktime(K_B)$. From (6) and $\{|A, Tgs, authK, T_a|\}_{K_A} \sqsubset term(m)$ for any node n' such that $time(n') \le time(m)$, we have

$$time(n') \le T_S + \text{sessionKcrktime},$$

then $time(n') \le T_a + \text{shrKcrktime}$, by Lemma 5, $term(n') \ne authK$; by Axiom 4, *authK* is symmetry, therefore $authK^{-1} = authK$, so $regular(authK^{-1}, m, \mathcal{B})$. From $time(m) \le T_S + \text{sessionKcrktime}$, and by Axiom 4 again, we have $cracktime(authK) = \text{sessionKcrktime}$, then $time(m) \le T_S + cracktime(authK)$, with (3), we have $time(m) \le time(tgs,1) + cracktime(authK)$. Therefore (5) is proved.

From (1), we have for any m' such that $m' \Rightarrow^{+} m$, $term(m') \in synth(M)$. With (2)(5), by Lemma 2, we have m must be in a regular strand i, then there exists six cases. Here we analyze the cases when **AS**[*i*, A', Tgs', $authK'$, t_0', t_1] or **TGS**[*i*, A', Tgs', $authK'$, $servK'$, B', T_a', T_0', t_0', T_S'], other cases are more simpler.

If **AS**[*i*, A', Tgs', $authK'$, t_0', t_1], then by inspection on the form of the strand, $m = (i,1)$ because m is positive. Obviously

$$term(m) = \left\{ \left| \{|A', Tgs', authK', t_1|\}_{K_{Tgs'}}, \{|A', Tgs', authK', t_1|\}_{K_{A'}} \right| \right\}.$$

Obviously $servK \sqsubset term(m)$, otherwise $term(m) \in M$. Therefore $servK \sqsubset \{|A', Tgs', authK', T_a|\}_{K_{A'}}$ or

$$servK \sqsubset \{|A', Tgs', authK', T_a|\}_{K_{Tgs'}}$$

then $servK = authK'$. From Axiom 5, we have *authK* uniquely originates from the strand i. Combining with (3), we can conclude i is both an authentication server strand and a ticket granting server

strand, obviously this is a contradiction.

If **TGS**$[i, A', Tgs', authK', servK', B', T_a', T_0', t_0', T_S']$, then by similar argument, we can prove that $servK' = servK$, from Axiom 5, we have $servK$ uniquely originates from i, with (3), we have $i = tgs$, we can prove that

$$term(m) = \left\{\left|\left\{\left|A, B, servK, T_S\right|\right\}_{K_B}, \left\{\left|A, B, servK, T_S\right|\right\}_{authK}\right|\right\}$$

then $term(m) \in synth(M)$ this contradicts with (1).

At last we only need prove that $term(m) \in synth(M)$ implies that $term(m) \neq servK$, this is similar to counterpart in Lemma 5.

Both Lemma 6 and Lemma 7 have the assumption that $\left\{\left|A, Tgs, authK, T_a\right|\right\}_{K_A}$ is a subterm of the node n, which can guarantee that *authK* must be a session key originated by an authentication server strand. The assumption $time(n) \leq T_a + \text{shrKcracktime}$ is used to guarantee that $authK$ is still safe at $time(n)$. Besides, the two terms $\left\{\left|A, B, servK, T_S\right|\right\}_{authK}$ and $\left\{\left|A, Tgs, authK, T_a\right|\right\}_{K_A}$ are intelligible for the client A, so these two lemmas are secrecy properties in the view of A.

In Lemmas 6 and 7, both $\left\{\left|A, B, servK, T_S\right|\right\}_{authK}$ and $\left\{\left|A, Tgs, authK, T_a\right|\right\}_{K_A}$ are unintelligible for an application server B because $authK$ and K_A cannot be known by B. So the two properties are not in B's view. B can only receive a message such as $\left\{\left|A, B, servK, T_S\right|\right\}_{K_B}$, can it be ensured that $servK$ is confidential when he receives the message $\left\{\left|A, B, servK, T_S\right|\right\}_{K_B}$? The following two lemmas are about the confidential information inferred from the message $\left\{\left|A, B, servK, T_S\right|\right\}_{K_B}$. They are secrecy properties in B's view.

Once a server ticket such as $\left\{\left|A, B, servK, T_S\right|\right\}_{K_B}$ occurs in a bundle, where A and B are not compromised, and B is not a ticket granting server, then conclusions similar to those in Lemma 6 and Lemma 7 can be drawn.

Lemma 8 *Let* $n \in \mathcal{B}$, $A, B \notin$ **Bad**, *and* $B \notin$ **TGSs**. *If* $\left\{\left|A, B, servK, T_S\right|\right\}_{K_B} \sqsubset term(n)$, *then there exists a ticket grant server strand tgs such that* $TGS[tgs, A, Tgs, authK, servK, B, T_a, T_0, t_0, T_S]$ *for some* Tgs, $authK$, T_a, T_0, t_0, $(tgs,1) \in \mathcal{B}$ *and* $T_S \leq time(n)$.

Lemma 9 *Let* $n \in \mathcal{B}$, $A, B \notin$ **Bad**, *and* $B \notin$ **TGSs**. *If* $\left\{\left|A, B, servK, T_S\right|\right\}_{K_B} \sqsubset term(n)$, *then for any node* $m \in \mathcal{B}$ *such that* $time(m) \leq T_S + \text{sessionKcr ktime}$, $term(m) \neq servK$.

Here we summarize the main ideas used in the above proof of secrecy properties.

• For a long-term key of a regular agent, its secrecy is easily inferred because it is never sent as a part of a message. We only need the well-founded induction principle on bundles to prove this.

• But for a short session key $authK$ or $servK$, the cases are more complex because they are sent as a part in a message such as

$$\left\{\left|A, Tgs, authK, T_a\right|\right\}_{K_A} \text{ or } \left\{\left|A, B, servK, T_S\right|\right\}_{authK}.$$

In kerberos V, a session key such as $authK$ ($servK$) occurs as a part of a term of node n which is of the form $\left\{\left|h\right|\right\}_K$, where K can be either a long-term key or another short session key, and h also contains a timestamp t such as $T_a(T_S)$, which indicates the time when $\left\{\left|h\right|\right\}_K$ is t. As mentioned before, both secrecy of K and recency of $\left\{\left|h\right|\right\}_K$ should be guaranteed. Secrecy of K can be directly drawn from other lemmas on K. But for recency checking, firstly we need prove that the timestamp t indeed indicates the time when $\left\{\left|h\right|\right\}_K$ is originated. Lemmas 4, 6, 9 play a role in guaranteeing that t is the first time when $authK$ ($servK$) is originated. From this and the assumption that $time(n) \leq t + \text{cracktime}(K)$, the recency of $\left\{\left|h\right|\right\}_K$ can be proved.

5. Proving Authentication Goals

For convenience, we call that a strand i uses a term $\left\{\left|h\right|\right\}_K$ as an unsolicited test if there is a node n is in the strand i and is an unsolicited test for $\left\{\left|h\right|\right\}_K$ in a bundle $\mathcal{B}$. Because a guarantee of the existence of a regular node can be drawn from an unsolicited test, a regular agent uses unsolicited test to authenticate its regular protocol participant in Kerberos V.

The client strand in the authentication phase receives $\left\{\left|A, Tgs, authK, T_a\right|\right\}_{K_A}$ as an unsolicited test that authenticates the positive node of the authentication server strand. The intuition behind this authentication is quite straightforward. By case analysis on the form of i, we have $\left\{\left|A, Tgs, authK, T_a\right|\right\}_{K_A} \sqsubset term(i,1)$, combining with the assumption that A is not compromised, by Lemma 4, we have $\left\{\left|A, Tgs, authK, T_a\right|\right\}_{K_A}$ can only be originated by an authentication server. For the sake of brevity, in the following discussion we use $P[x,*,\ldots,y]$ to denote $\exists x'.P[x,x',\ldots,y]$. $\mathcal{B}$ is a bundle of Kerberos V.

Lemma 10 *Let* $A \notin$ **Bad**. *If* i *is a client strand in the authentication phase such that* ***Ag-I***$[i, A, Tgs, authK, T_a, authTicket, t_0, t_1]$ and $(i,1) \in \mathcal{B}$, *then there exists an authentication server strand as such that* $AS[as, A, Tgs, authK, *, T_a]$, $(as,1) \in \mathcal{B}$.

The ticket grant server strand uses $\left\{\left|A, T_0\right|\right\}_{authK}$ as an unsolicited test to authenticate the client strand in the authorization phase. This guarantee is ensured from the secrecy of $authK$, which is in turn guaranteed by the ticket $\left\{\left|A, B, authK, T_a\right|\right\}_{K_{Tgs}}$. By the trace specification of a ticket grant server strand, we have that

$$\left\{\left|\left\{\left|A, B, authK, T_a\right|\right\}_{K_{Tgs}}, \left\{\left|A, T_0\right|\right\}_{authK}\right|\right\}$$

is received by *Tgs* earlier than the time

$$T_a + \text{sessionKcrktime},$$

by Lemma 5, $authK$ is safe at that time.

Lemma 11 *Let* $A \notin$ **Bad** , $Tgs \in$ **TGSs** . *If tgs is a ticket grant server strand such that* **TGS**[$tgs, A, Tgs, authK, servK, B, T_a, T_0, t_0, t_1$], *and* $(tgs,0) \in \mathcal{B}$, *then there exists a client strand* i *in the authorization phase such that* $(i,0) \in \mathcal{B}$ *and* **Ag-II**[$i, A, authK, *, *, *, *, *, *, T_0, *$] .

Proof. By analysis on the form of *tgs* strand, we have (1) $\left\{\left|A, T_0\right|\right\}_{authK} \sqsubset (tgs,0)$, (2) $\left\{\left|A, B, authK, T_a\right|\right\}_{K_{Tgs}} \sqsubset term(tgs,0)$ and $time(tgs,0) \le T_a$ + shrKcrackt ime. From (2), by Lemma 5, we have that $term(m) \neq authK$ for any $m \in \mathcal{B}$ such that $time(m) \le time(tgs,0)$, therefore $regular(authK, (tgs,0), \mathcal{B})$. With (1), by Lemma 1, we have (3) there is a positive regular node m such that $m \preceq_{\mathcal{B}} (tgs,0)$ and $\left\{\left|A, T_0\right|\right\}_{authK} \sqsubset term(n)$ and $\left\{\left|A, T_0\right|\right\}_{authK} \not\sqsubset term(m')$ for any node m' such that $m' \preceq_{\mathcal{B}} m$. Obviously $m \in \mathcal{B}$.

Now we need prove that m must be in a client strand i in the authorization phase. From the fact that m is regular, then we have six cases, here we select two cases when (4) *m* is in a strand *i* such that **Ag-II**[i, A', $authK'$, $authTicket'$, B', $servK'$, T_S', $servTicket'$, t_2, t_3] or (5) m is in a strand i such that **Ag_III**[$i, A', servK'$, $servTicket', t_4, t_5$]. Other cases are more simpler.

If (4) holds, then $m = (i,0)$ because m is positive. From $\left\{\left|A, T_0\right|\right\}_{authK} \sqsubset term(n)$ and

$$term(m) = \left|\left\{authTicket', \left\{\left|A, t_2\right|\right\}_{authK}, B\right\}\right|,$$

we have either (6) $\left\{\left|A, T_0\right|\right\}_{authK} \sqsubset authTicket'$ or (7)

$$\left\{\left|A, T_0\right|\right\}_{authK} \sqsubset \left\{\left|A', t_2\right|\right\}_{authK'} .$$

If (6) holds, then by the definition of the client strand, there exists some client strand i_1 such that $i_1 \mapsto i$ and **Ag-I**[i_1, A', Tgs', $authK'$, T_a', $authTicket'$, t_0', t_1']. From the definition of the strand, we have

$$term(i_1,1) = \left\{\left|authTicket', \left\{\left|A', Tgs', authK', T_a'\right|\right\}_{K_{A'}}\right|\right\}.$$

From this and (6), we have (8) $\left\{\left|A, T_0\right|\right\}_{authK} \sqsubset term(i_1,1)$. From $i_1 \mapsto i$, $(i_1,1) \Rightarrow (i,0)$, then (9) $(i_1,1) \preceq_{\mathcal{B}} (i,0)$. But (8) and (9) contradicts with (3). If (7) holds, then $A = A'$, $T_0 = t_2$, $authK = authK'$. So the conclusion is obtained.

If (5) holds, then similar to the counterpart of the argument for case (4), we have either (10) $\left\{\left|A, T_0\right|\right\}_{authK} \sqsubset servTicket'$ or (11) $\left\{\left|A, T_0\right|\right\}_{authK} \sqsubset \left\{\left|A', t_4\right|\right\}_{servK'}$. For case (10), its proof is similar to that of case (6). If (11) holds, then (12) $A = A'$, $T_0 = t_4$, $authK = servK'$. By the definition of **Ag_III** , (13) there is a client strand i_2, i_1 such that $i_1 \mapsto i_2$ and $i_2 \mapsto i$ and **Ag-II**[i_2, A', $authK'$, $authTicket'$, B', $servK'$, T_S', $servTicket'$, t_2, t_3] and **Ag-I** [i_1, A', Tgs', $authK'$, T_a', $authTicket'$, t_0', t_1'] for some $Tgs' \in$ **TGSs**. Obviously,

$$\left\{\left|A', Tgs', authK', T_a\right|\right\}_{K_{A'}} \sqsubset term(i_1,1),$$

$$\left\{\left|A', B', servK', T_S'\right|\right\}_{authK'} \sqsubset term(i_2,1),$$

and $time(i_2,1) \le T_a'$ + shrKcracktime. From $i_1 \mapsto i_2$ and $i_2 \mapsto i$ and $(i,0) \in \mathcal{B}$, we have $(i_1,1) \in \mathcal{B}$ and $(i_2,1) \in \mathcal{B}$. From (12), and the assumption $A \notin$ **Bad,** by Lemma 6, (14) there is a ticket granting server strand tgs' such that **TGS**[tgs', A', Tgs', $authK'$, $servK'$, B', T_a', T_0', t_0'', T_S'] for some T_0', t_0''. But from (2), by Lemma 4, we have (15) there is an authentication server as such that $AS[as, A, Tgs, authK, t_0', T_a]$ for some t_0'. But from (12) and Axioms 4,5, we have $tgs' = as$ because $authK$ ($servK'$) uniquely originates from a strand, obviously this is a contradiction.

A client strand i_2 in the authorization phase receives $\left\{\left|A, B, servK, T_S\right|\right\}_{authK}$ $\left\{\left|A, B, servK, T_S\right|\right\}_{authK}$ as an unsolicited test. Note that is received in the second node in the client strand; furthermore, from the definition of **Ag-II**, we have that there exists a client strand i_1 in the authentication phase such that $i_1 \mapsto i_2$, and the ticket $\left\{\left|A, Tgs, authK, T_a\right|\right\}_{K_A}$ must be received at the second node of i_1 ; from the definition of **Ag-II**, $\left\{\left|A, B, servK, T_S\right|\right\}_{authK}$ must have been received at an earlier time than T_a + shrKcracktime , then by Lemma 5, it can be guaranteed that $authK$ must be safe at the time when the client strand receives $\left\{\left|A, B, servK, T_S\right|\right\}_{authK}$.

Lemma 12 *Let* $A, B \notin$ **Bad** . *If* i *is a client strand in the authorization phase such that* **Ag-II** [$i, A, authK, T_a$, $auth\text{-}Ticket, B, servK, T_S, servTicket, t_0, t_1$] and $(i,1) \in \mathcal{B}$, *then there exists a client strand* i_0 *in the authentication phase, and a ticket grant server strand tgs, and some Tgs such that* $i_0 \mapsto i$ *and* **Ag-I** [$i_0, A, Tgs, authK, T_a$, $authTicket, *, *$] *and* **TGS**[$tgs, A, Tgs, authK, servK, B$, $T_a, *, *, T_s$] , *and* $(tgs,1) \in \mathcal{B}$, *and* $B \notin$ **TGSs** .

The application server B receives $\left\{\left|A, T_4\right|\right\}_{servK}$, which is an unsolicited test to guarantee that the first received message must be from a client strand in the service phase. This guarantee is ensured from the secrecy of $servK$, which is in turn guaranteed by the ticket $\left\{\left|A, B, servK, T_S\right|\right\}_{K_B}$. By the trace specification of an application server strand, we have that

$$\left\{\left|\left\{\left|A, B, servK, T_S\right|\right\}_{K_B}, \left\{\left|A, T\right|\right\}_{4servK}\right|\right\}$$

is received by B earlier than the time

$$T_S + \text{sessionKcrktime}.$$

By Lemma 9, $servK$ is safe at that time.

Lemma 13 *Let* $A, B \notin$ **Bad**, $B \notin$ **TGSs**. *If* b *is an application server strand such that* **Apps**$[b, A, B, servK, T_S, T_4, t_0, t_1]$, and $(b,0) \in \mathcal{B}$, *then there are two client strands* i_2, i_3 *and some servTicket such that* $i_2 \mapsto i_3$ *and* $(i_3,0) \in \mathcal{B}$ *and* **Ag_II**$[i_2, A, *, *, B, servK, T_S, servTicket, *, *]$ *and* **Ag_III**$[i_3, A, servK, servTicket, T_4, *]$.

The client strand in the service phase uses $\{|T_4|\}_{servK}$ as an unsolicited test to authenticate the application server strand. This guarantee is also ensured from the secrecy of $servK$, which is in turn guaranteed by the ticket $\{|A, B, servK, T_S|\}_{authK}$, and $\{|A, Tgs, authK, T_a|\}_{K_A}$. By the trace specification of an application server strand, we have that $\{|T_4|\}_{servK}$ is received by A earlier than the time $T_S + \text{sessionKcrktime}$ and $T_a + \text{shrKcracktime}$. By Lemma 7, $servK$ is safe at that time.

Lemma 14 *Let* $A, B \notin$ **Bad**, $B \notin$ **TGSs**. *If* i_3 *is a client strand in a service phase such that* **Agent_III**$[i_3, A, servK, servTicket, t_4, *]$, *and* $(i_3,1) \in \mathcal{B}$, *and* i_2 *is a client strand in the authorization phase such that* **Agent_II**$[i_2, A, authK, authTicket, B, servK, T_S, servTicket, t_2, t_3]$ *and* $i_2 \mapsto i_3$, *then there exists an application server strand* b *such that* **Apps**$[b, A, B, servK, T_S, t_4, , *, *]$ *and* $(b,1) \in \mathcal{B}$.

Proof. By analysis on the form of strand i_3 and i_2, we have (1) $term(i_3,1) = \{|t_4|\}_{servK}$ and (2) $\{|A, B, servK, Ts|\}_{authK} \sqsubset term(i_2,1)$. By unfolding the definition of **Agent_II,** there exists a client strand i_1 such that $i_1 \mapsto i_2$ and **Ag-I** $[i_1, A, Tgs, authK, T_a, authTicket, t_0, t_1]$ and $Tgs \in$ **TGSs** for some Tgs, T_a, t_0, t_1. Obviously, (3) $\{|A, Tgs, authK, T_a|\}_{K_A} \sqsubset term(i_1,1)$, $time(i_1,1) \leq T_a$ +shrKcracktime. From $i_1 \mapsto i_2$ and $i_2 \mapsto i_3$, we can easily conclude that $(i_1,1) \Rightarrow^+ (i_3,1)$ and $(i_2,1) \Rightarrow^+ (i_3,1)$. With $(i_3,1) \in \mathcal{B}$, we have $(i_2,1) \in \mathcal{B}$ and $(i_1,1) \in \mathcal{B}$. With (2)(3), by Lemma 7, (4) $term(m) \neq servK$ for any node m such that $time(m) \leq T_S$ +sessionKcrktime. By the definition of **Agent_III** we have (5) $time(i_3,1) \leq T_S$ +sessionKcrktime. From (4)(5), we have $term(m) \neq servK$ for any node m such that $time(m) \leq time(i_3,1)$, therefore $regular(servK, (i_3,1), \mathcal{B})$. So $(i_3,1)$ is an unsolicited test for $\{|T_4|\}_{servK}$ in $\mathcal{B}$. By Lemma 1, (5) there is a regular positive node m such that $m \preceq_{\mathcal{B}} (i_3,1)$ and $\{|t_4|\}_{servK} \sqsubset term(m)$ and $\{|t_4|\}_{authK} \not\sqsubset term(m')$ for any node m' such that $m' \preceq_{\mathcal{B}} m$. Obviously $m \in \mathcal{B}$. By simple case analysis, we have that m must be in an application server b such that **Apps**$[b, A', B', servK', T_S', t_4', *, *]$, $m = (b,1)$. By the definition of **Apps**, $term(b,1) = t_{4_{servK'}}'$. With $t_{4_{servK}} \sqsubset term(b,1)$, we have (6) $servK' = servK$ and $t_4' = t_4$.

Let $M =_{df} \left\{ \begin{matrix} \{|A, B, servK, T_S|\}_{K_B}, \\ \{|A, B, servK, T_S|\}_{authK} \end{matrix} \right\} \cup \{t \mid servK \not\sqsubset t\}$.

Obviously $time(b,0) \preceq_{\mathcal{B}} time(b,1) \preceq_{\mathcal{B}} time(i_3,1) \leq T_S$ + sessionKcrktime, by the proof of Lemma 7, we have $term(b,0) \in synth(M)$, *i.e.*,

$$\left\{|\{|A', B', servK', T_S'|\}_{K_{B'}}, \{|t_4'|\}_{servK'}|\right\} \in synth(M).$$

By the definition of $synth$, we have

$$\{|A', B', servK', T_S'|\}_{K_{B'}} \in synth(M),$$

then we have (7) $\{|A', B', servK', T_S'|\}_{K_{B'}} \in M$ or (8) $\{|A', B', servK', T_S'|\} \in synth(M)$. If (7) holds, from (6) (7) and the definition of M, we have

$$\{|A, B, servK, T_S|\} = \{|A', B', servK', T_S'|\},$$

then $A = A'$ and $B = B'$ and $T_S = T_S'$. Therefore, the conclusion holds. If (8) holds, by the definition of $synth$, we have $servK' \in synth(M)$, with (6), we have $servK \in synth(M)$, then by the definition of $synth$, we have $servK \in M$, but this contradicts with the definition of M.

Roughly speaking, we need two steps to prove an authentication goal that if there is a regular responder strand $Resp(r, \vec{x})$ and the k-th node of the strand is in a bundle $\mathcal{B}$, then there is an initiator strand $Init(i, \vec{x})$ and some j-th node of the initiator strand is in $\mathcal{B}$. First we prove that (r,k) is an unsolicited test for some encrypted term $\{|h|\}_K$ in $\mathcal{B}$, which requires the secrecy of K. This is can be easily proved by the secrecy results on keys in section 3. Therefore, we have that there exists some regular node m in $\mathcal{B}$ by Lemma 2. Second, we need prove that m indeed is the intended node (i, j). In order to prove this, we need do case analysis on the form of the strand which m possibly lies in. This proof needs unicity property of some session keys and the results of unsolicted tests, namely, the facts that $\{|h|\}_K \sqsubset term(m)$ and m is minimal.

6. Conclusions and Related Work

Our main aim is to extend and mechanize the strand space theory to analyze Kerberos V, since mechanization in a theorem prover not only helps us model protocols rigorously and specify protocol goals without any ambiguity, it also guarantees a formal proof. Besides the

essential inherence from the classic strand space method, our work is deeply inspired by Paulson and Bella's work. We have directly used their formalization of message algebra, and have learned a lot about the semantics of timestamps and replay attacks from [4]. However, we model and analyze protocols in strand space theory rather than in Paulson's trace theory. In detail, we model behaviors of all the agents by strands, and mainly use the well-founded induction principle to prove properties. So in our Isabelle formalization, main efforts have been devoted to definitions and lemmas about strand space theory. e.g., we formalize strands, bundles, unique originality, the well-founded principle on bundles, and use this principle to prove important results such as unsolicited authentication test and regularity of keys.

In [4], the ability of a penetrator to crack a stale encrypted message is modelled by the Oops rule in the inductive definition of a trace, and the trace definition depends on the protocol under study. However, in the strand space theory, a penetrator's abilities are modelled to be independent of the protocol, that is the main reason why we relate a key with a crack time, and model a penetrator's ability of cracking a stale encrypted message by a new key cracking strand. The advantage of our method is that modelling a penetrator's behavior remains independent and results such as the unsolicited authentication tests can be generalized.

Regarding verification of the Kerberos protocols, Mitchell *et al.* [7] analyzed a simplified version of the protocol by model checking, and Butler *et al.* [8] analyzed the Kerberos V protocol using MSR [9]. But they did not include timestamps and replay attacks in their model, in fact the former work ignored both nonces and timestamps, and the latter only considered the implementation of the Kerberos protocol basing on nonce.

7. References

[1] S. P. Miller, J. I. Neuman, J. I. Schiller and J. H. Saltzer, "Kerberos Authentication and Authorisation System," Technical Report, Technical Plan Section E.2.1, MIT, Athena, 1989.

[2] K. R. C. Neuman and S. Hartman, "The Kerberos Network Authentication Service (v5)," Technical report, Internet RFC 4120, July 2005.

[3] Y. L. and J. Pang, "Extending the Strand Space Method with Timestamps: Part I the Theory," *Journal of Information Security*, Vol. 1, No. 2, 2010, pp. 45-55.

[4] G. Bella, "Inductive Verification of Cryptographic Protocols," PhD Thesis, Cambridge University Computer Laboratory, 2000.

[5] F. Javier Thayer, J. C. Herzog and J. D. Guttman, "Strand Spaces: Proving Security Protocols Correct," *Journal of Computer Security*, Vol. 7, No. 1, 1999, pp. 191-230.

[6] Y. Li, "Strand Space and Security Protocols". http://lcs.ios.ac.cn/˜lyj238/strand.html.

[7] J. C. Mitchell, M. Mitchell and U. Stern, "Automated Analysis of Cryptographic Protocols Using Murphi," *Proceedings of* 18*th Symposium on Security and Privacy*, 1997, pp. 141-153.

[8] L. Bozga, C. Ene and Y. Lakhnech, "A Symbolic Decision Procedure for Cryptographic Protocols with Time Stamps," *Journal of Logic and Algebraic Programming*, Vol. 65, No. 1, 2005, pp. 1-35.

[9] F. Butler, I. Cervesato, A. Jaggard and A. Scedrov, "A formal Analysis of Some Properties of Kerberos 5 Using MSR," *Proceedings of* 15*th IEEE Computer Security Foundations Workshop*, 2002, pp. 175-190.

Journal of Information Security, 2010, 1, 68-73
doi:10.4236/jis.2010.12008 Published Online October 2010 (http://www.SciRP.org/journal/jis)

Sustainable Tourism Using Security Cameras with Privacy Protecting Ability

Vacharee Prashyanusorn[1], Yusaku Fuji[2], Somkuan Kaviya[1], Somsak Mitatha[3], Preecha Yupapin[3]
[1]*Innovative Communication Program, Krirk University, Bangkok, Thailand*
[2]*Gunma University, Kiryu, Japan*
[3]*King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand*
E-mail: *vacharee_prashyanusorn@hotmail.com, fujii@el.gunma-u.ac.jp, courting_*19*@hotmail.com, kmsomsak@kmitl.ac.th, kypreech@kmitl.ac.th*
Received September 26, 2010; *revised October* 12, 2010; *accepted October* 15, 2010

Abstract

For sustainable tourism, a novel method of security camera operation is proposed. In the method, security cameras, which encrypt the taken images and store them into the memory card inside, are used. Only when crimes occur, the memory cards are taken out from the cameras and the images are decrypted with the key and viewed by the city government and/or the police. When no crimes occur, images are overwritten by the new ones after a week automatically without being viewed by anyone. By using the stand-alone cameras without wiring to the control center, the installation cost and the operation cost are much lower than CCTV cameras. By using image encryption, the privacy of the tourists is protected. Using this system, high density installation of the security cameras with very low cost can be realized in encryption with image encryption privacy protection function.

Keywords: Innovative Communication, Security Camera, Privacy, Safety, Sustainable Tourism, Crime Prevention

1. Introduction

In the sightseeing places, security camera systems, such as Closed-circuit Television (CCTV) system, are now widely used and can be found in ordinary shops and citizens' houses. These systems sometimes play an important role in reducing crime and identifying suspects. However, many problems seem to arise with regard to such security camera systems because of the fact that they are introduced only for the benefit of the owners. One problem is that an expensive high-end security camera system is required for maintaining complete surveillance of an owner's property. The second problem is that a typical system usually keeps watch only inside the owner's property; therefore, it cannot be used for the overall safety of the community. The third problem is that if the system keeps a watch outside the owner's property, it could amount to invasion of the privacy of neighbour. We argue that these problems can be solved if the camera systems are introduced within an altruistic, community-minded framework.

Recently, many security camera systems have been installed in some countries such as the United Kingdom and the United States of America, by the national and the local governments. Although, it is difficult to evaluate the effectiveness of the security camera system in preenting crime [1,2], which are obvious that they can capture images of any person or car passing within their range. If a considerable number of security cameras are installed without any dead angles on every road, then every criminal who uses the roads can be captured and traced.

However, a center-controlled real-time monitoring system such as the typical systems costs a considerable amount of money and cannot be introduced everywhere without any dead angles. Therefore, we propose a new concept according to which a community can effectively prevent crime if some residents keep watch on what happens around their houses with the aid of their own home computers, cheap commercially available cameras, and free software. **Figure 1** shows the concept of the e-JIKEI Network.

Many types of software applications for capturing video images are available; however, we could not find a

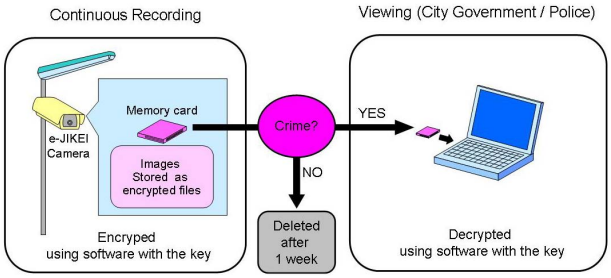


Figure 1. Concept of "e-JIKEI with privacy protection".

free one that could be used to implement our concept. Therefore, we have developed asoftware with the minimum necessary functions and distributed it free of charge through our website [3]. The software supports both English and Japanese languages. The software simply selects relevant pictures and saves them to the hard disk [4]. This concept has been discussed from the viewpoints of social science [5], homeland security [6] and e-Government [7].

2. Personal Computer (PC)-Based System Using Free Software

We have provided the first version of the free software "Dairy EYE standard." Its functions are very limited but essential. The major features of the software are as follows:

• High stability: It can be run continuously for more than 300 days.

• High operation of file storage: The file name and its path express time and location information.

• Minimum necessary storage: Simple picture selection software has been adapted. The software saves a picture only when the difference between two consecutive pictures exceeds the threshold.

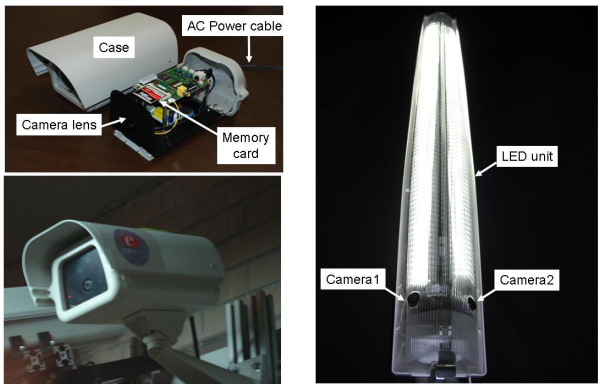
• Automatic delete: Folders that are older than the save period set by the owner are automatically deleted.

• Compatibility with many types of cameras: The software can operate in the VFW mode (PC cameras and USB video adapters) and the FTP mode (network cameras).

• Simultaneous operation: The software can operate several cameras connected to a PC.

• No Internet connection: Because of concerns related to privacy, the function of connection to the Internet was disabled in the distributed version of the software. Even in this case, the e-JIKEI Network can be formed, where the word "Network" refers not to the Internet but to the personal network of the residents.

We think that the e-JIKEI Network system should be easily installed in a D.I.Y. (Do It Yourself) manner at a low cost. **Figure 2** shows the examples of camera set-



(a) e-JIKEI Camera (b) e-JIKEI Light

Figure 2. Prototype of the e-JIKEI camera and e-JIKEI Light.

tings. In one case, an inexpensive network camera is installed outside a house. In the other case, an inexpensive USB PC camera is installed inside a house by using adhesive tape.

3. E-JIKEI with Privacy Protection

We propose a new concept regarding the management of security cameras, e-JIKEI with Privacy Protection, in which those who own and manage images (owners) and those who have the right to view these images (viewers) are separated by means of the encryption of the images [8]. On the basis of this concept, encrypted images are transferred from an owner to a viewer only when both the owner and the viewer consider it necessary, such as in the case of crimes; then, the encrypted images are restored for viewing by the viewer. By this method, the images can be viewed only when absolutely necessary. This concept has been proposed to prevent the risk of privacy violation, as well as to reduce the unnecessary psychological burden that third parties may be subjected to, with the aim of promoting the placement of security cameras throughout local communities.

By managing the security camera system using our concept, it is possible to markedly reduce the negative effects associated with the introduction of security cameras, such as concerns over the violation of privacy, without reducing the positive effects, such as crime prevention at places other than those requiring high-level security and constantly manned surveillance, i.e., most communities, while providing recorded images to investtigating authorities in the case of crime.

In a practical example carried out in Kiryu City, Gunma Prefecture, a PC-based security camera system is owned and managed by the owners of retail stores affiliated with the merchant association "Suehirocho Shotengai Shinkokyokai," and images are encrypted and stored

in the system. To view the stored images, special software installed in the PCs at the Police Department of Kiryu City must be used. Only when the owners of the retail stores and the police determine that it is necessary to view these images, are the stored images transferred from the owners of the retail stores to the police. Then, the stored images are viewed by the police and used as information for investigations. The encrypted images that are stored at retail stores are automatically deleted after 30 days if no incidents or accidents have occurred.

To prove that the software installed in the PC definitely encrypts the images with the cipher-key owned by only the police, a paper on which the owner states the purpose of the camera system and allows the investigations by the merchant association at any time is posted near the cameras. Because the owners of retail stores purely wish to safeguard their shopping street and the customers, and do not intend to violate the privacy of their customers, the installed system is ideal for them.

4. All-in-One System "E-JIKEI Camera"

In the experiments of the PC-based system, we have realized that the PC-based system is not very user-friendly since it is difficult for ordinary residents to maintain and operate PCs. In the near future, when home automation is widespread, this problem of PC operation will be solved. However, at this time, it is a serious obstacle for the widespread nationwide use of the e-JIKEI Network. Therefore, we decided to develop an all-in-one system without the use of a PC.

We have developed a prototype of security camera systems "e-JIKEI Camera," which can realize the concept of "e-JIKEI with Privacy Protection." **Figure 3** shows the prototype of the e-JIKEI Camera. It only requires an AC power supply and can be attached outdoors just like a streetlamp. If it is mass produced, the cost per camera will be less than 200 USD. The features of the developed camera are as follows:

1) It can realize the concept of "e-JIKEI with Privacy Protection."

2) All images are encrypted and stored in the memory.

3) To decrypt and view the image, both the special software and the secret key are required.

4) It has a card-type memory of 16 GB, in which the images for the last 1 week are recorded.

5) It can be placed outside.

6) It requires an AC power supply of only 100-240 ACV.

7) The price of the prototype, the first 1000 pieces, is 500 USD/piece.

There are many types of security camera systems available; however, a system with the above features does not exist, except for the newly developed e-JIKEI Camera.

The e-JIKEI Camera is used for realizing our concept of a security camera system in which those who own images (owners) and those who have the right to view the images (viewers) are separated by means of image encryption. This concept was suggested with the aim of preventing the risk of privacy violation, reducing the unnecessary psychological burden that third parties may experience, and promoting the placement of security cameras in local communities.

In Kiryu city, Japan, a social experiment has been conducted since 30 May 2009, in which eleven cameras are installed on the poles of the street lamps in a residential area, as illustrated in **Figure 3(b)**. **Figure 4** shows the location of the 11 e-JIKEI Cameras and the 411 street lamps in the area, where 2218 homes are located. In the experiment, the owner of the images is the PTA (Parent-Teacher Association) of the Higashi Elementary School, and the viewer is the Kiryu Police Station.

(a) Setting of e-JIKEI Camera (b) Setting of e-JIKEI Light

Figure 3. Examples of camera installation in walking street in Pattaya City.

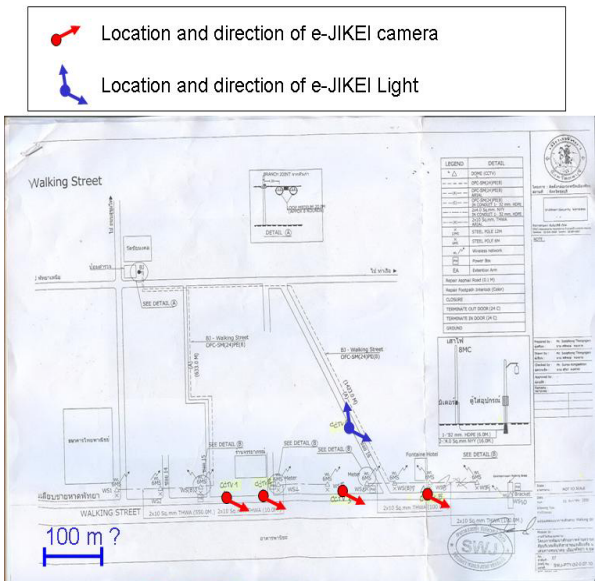


Figure 4. Locations of e-JIKEI cameras and street lamps.

Figure 5 shows the procedure for using the e-JIKEI camera in the experiment. Before the experiment, we explained the concept of e-JIKEI with Privacy Protection to the residents of all the 2218 homes by circulating a notice for the same and in an explanation meeting held at the community hall. Our proposal for this experiment was granted by the residents without any negative opinions. During the first six months of the experiment, three crimes were committed. In each case, the police asked the PTA to provide the images, and the PTA decided to grant the police request. During the experiment, many residents expressed their opinion that the e-JIKEI Cameras were very effective in improving the safety of the community but the number of cameras was still very small compared to the number of street lamps.

Recently, we held a discussion with the residents, PTA, and police. The residents and the PTA provided the following opinions about the installed system:

1) It seems very effective in improving the safety of the community.

2) Number of cameras is very small.

3) Privacy violation seems to be perfectly prevented.

4) The cost is comparable to that of the usual street lamps and therefore affordable.

(a) Picture taken by e-JIKEI Camera-1

(b) Picture taken by e-JIKEI Camera-5

Figure 5. Pictures taken by the cameras.

The police had the following opinions:

1) The reliability of the system is very high. (There has been no trouble for more than six months now.)

2) The quality of the images is acceptable but can be improved.

3) We hope this camera system spreads all over the city.

If our concept on the security camera system with privacy protection is accepted by society, then a considerable number of cameras, which is comparable to the number of streetlamps, will be introduced in communities throughout the country and the world. Then, every street will be watched by numerous cameras, and photographs of suspects can be provided to the police once a crime occurs in a community.

In the current all-in-one security camera in the e-JIKEI Network, the camera has to be opened to remove the memory card. However, this inconvenience is preferred from the viewpoint of privacy protection, especially in the initial stage of the society's gradual acceptance of our concept. However, in the near future, the cameras will be connected to the Internet after the information security system between the owners and the viewers is established. Thereafter, online operations of solving crime, such as the rescue of kidnapped child CCTV camera system [11,12] is suitable for the real time monitoring of the very important points. However, the cost of installation/maintenance/operation is high, Then the number of the cameras are strictly limited due to such costs.ren, can be implemented.

5. Discussions

Comparing to the existing the CCTV camera system in Pattaya City, the e-JIKEI Camera has the following features,

1) Low installation cost: The wiring to the control room and control room itself are not necessary. Only AC power supply is required.

2) Low maintenance/operation cost: The memory cards of the cameras are only taken, when the city government thinks that necessary.

3) Privacy Protection: Only crime occurs, only the certain officers of the city government can view the images.

In the case of the Pattaya City, we propose that the combination the existing CCTV system and the e-JIKEI Cameras. 300 pieces CCTV system watches for only the very busy points, and the huge number of the e-JIKEI Cameras watch the dead-angle of the CCTV in the busy area. In addition, if a huge number of the e-JIKEI Cameras are installed to the quiet residential area, the safety of the whole city will be increased significantly.

If the memory capacity is sufficiently large, the selection of images, in which only the images that are sufficiently different from the previous ones are saved, is not necessary. If the memory capacity is small and memory needs to be conserved, then the selection of images is useful. However, in general, there is no selection algorithm that has a zero failure rate with respect to the selection of necessary images. If all the images are saved without image selection, then the failure of saving a necessary image is prevented. In addition, without this selection, the CPU power can be saved.

At this moment, only the software and programmable stand-alone camera devices, which do not connect to the Internet, have been developed. If the system of security cameras connected to computers and to the Internet spreads nationwide, a very powerful and flexible social structure can be formed. In addition, the software installed in each system can be easily upgraded. This means that this social structure can lead to very interesting research subjects and applications for software research, such as research involving image processing, security systems, and artificial intelligence.

If the security cameras are to be connected to the Internet, the protection of the privacy of the ordinary citizen has to be considered very seriously. A different social structure, including increased social awareness and a revised legal system, will be required for the society; in this structure, every outdoor location will be monitored by security cameras, but the privacy of ordinary citizens will be highly protected, being understood and accepted.

If the appropriate legal, social, and administrative systems are established, most residents will allow appropriate third parties, such as the police department and the city hall, to access their PCs and the saved information through the Internet in the case of a community emergency. In such a case, it will be necessary to ensure that the access rights to the images saved on the PCs can be separately, strictly, and flexibly defined and given to the appropriate third parties by the owner of each system.

If the security cameras are connected to the Internet and can be accessed by the police in the case of serious crimes, the real-time chasing of criminals and rescue of kidnapped children will be possible. A single control station manned by the police, where many operators can access images from cameras spread throughout the nation, is required to realize such a social system.

6. Conclusions

We are asking citizens to compare the responsibility of watching what happens around their houses with the risk of violation of their privacy. In the meanwhile, we are trying to increase the advantages of the security camera such as crime prevention and identification of suspects and to reduce its disadvantages such as violation of privacy. We are now commencing tests to assess the true contribution of our concept toward the realization of a safer and more comfortable community.

7. Acknowledgements

The Japanese team was supported by the research aid fund of the Research Foundation for Safe Society and the Grant-in-Aid for Scientific Research (B) 21300268 (KAKENHI 21300268).

The authors would also like to give their acknowledgement to Pattaya City Council, Chonburi, Thailand for the research facility under the tourism with safety and privacy project.

8. References

[1] C. Welsh and D. Farrington, “Crime Prevention Effects of Closed Circuit Television: A Systematic Review,” Home Office Research Study, 252, 2002.

[2] M. Gill and A. Spriggs, “Assessing the Impact of CCTV,” Home Office Research Study, 292, 2005.

[3] NPO, The e-JIKEI Network Promotion Institute. http://www.e-jikei.org/index_e.htm/.

[4] Y. Fujii, N. Yoshiura and N. Ohta, “Community Security by Widely Available Information Technology,” JoCI 2005, 2. http://ci-journal.net/index.php/ciej/article/view/285/.

[5] Y. Fujii, N. Yoshiura and N. Ohta, “Creating a Worldwide Community Security Structure Using Individually Maintained Home Computers: The e-JIKEI Network Project,” *Soc. Sci. Comput. Rev.*, Vol. 23, 2005, pp. 250-258.

[6] N. Yoshiura, Y. Fujii and N. Ohta, “Using the Security Camera System Based On Individually Maintained Computers For Homeland Security: The e-JIKEI Network Project,” *Proc. IEEE IMTC*, Ottawa, Canada, 2005, pp. 101-105.

[7] H. Ueda, Y. Fujii, S. Kumakura, N. Yoshiura and N. Ohta, “e-JIKEI Network Project/Japan: Enhancing Community Security,” *eGov.*, Vol. 11, 2009, pp. 9-11.

[8] Y. Fujii, K. Maru, N. Yoshiura, N. Ohta, H. Ueda and Y. Sugita, “New Concept Regarding Management of Security Cameras,” JoCI 2008, 4. http://www.ci-journal.net/index.php/ciej/article/view/442/427/

[9] Y. Fujii, K. Maru, K. Kobayashi, N. Yoshiura, N. Ohta, H. Ueda and P. P. Yupapin, “e-JIKEI Network Using e-JIKEI Cameras: Community Security Using Considerable Number of Cheap Stand-Alone Cameras,” *Safety Science*, Vol. 48, No. 7, 2010, pp. 921-925.

[10] V. Prashyanusorn, S. Kaviya and P. P. Yupapin, “Surveillance System for Sustainable Tourism with Safety and Privacy Protection,” *Procedia—Social and Behavioral Sciences*, Vol. 2, 2020, pp. 74-78.

[11] M. Zhang, B. Yang, S. Zhu and W. Zhang, “Ordered

semiring-Based Trust Establish Model with Risk Evaluating," *International Journal of Network Security*, Vol. 8, No. 2, 2009, pp. 101-106.

[12] S. H. Chiu, C. P. Lu and C. Y. Wen, "A Motion Detection-Based Framework for Improving Image Quality of CCTV Security Systems," *Journal of Forensic Sciences*, Vol. 51, No. 5, 2006, pp. 1115-1119.

Journal of Information Security, **2010, 1, 74-87**
doi:10.4236/jis.2010.12009 Published Online October 2010 (http://www.SciRP.org/journal/jis)

iPhone Security Analysis

Vaibhav Ranchhoddas Pandya, Mark Stamp
Department of Computer Science, San Jose State University, San Jose, USA
E-mail: *stamp@cs.sjsu.edu*
Received August 10, 2010; *revised September* 21, 2010; *accepted October* 15, 2010

Abstract

The release of Apple's iPhone was one of the most intensively publicized product releases in the history of mobile devices. While the iPhone wowed users with its exciting design and features, it also angered many for not allowing installation of third party applications and for working exclusively with AT & T wireless services (in the US). Besides the US, iPhone was only sold only in a few other selected countries. Software attacks were developed to overcome both limitations. The development of those attacks and further evaluation revealed several vulnerabilities in iPhone security. In this paper, we examine some of the attacks developed for the iPhone as a way of investigating the iPhone's security structure. We also analyze the security holes that have been discovered and make suggestions for improving iPhone security.

Keywords: iPhone, Wireless, Mobile, Smartphone, Jailbreaking, Reverse Engineering

1. Introduction

The release of Apple's iPhone on June 29, 2007 was one of the most heavily publicized events in the history of mobile electronics devices. Thousands of people lined up outside Apple stores prior to its release. Approximately three and half million iPhones were sold within the first six months of its release in the U.S. alone [1]. By any measure, the iPhone has been a commercial success—in spite of being a first-timer in the smart phone industry, Apple immediately outpaced traditional cell phone giants like Nokia, Motorola, and LG. The iPhone is an all-in-one package including a cell phone, a digital music and video player, a camera, a digital photo, music, and video library, and more [2]. It has helpful widgets for maps, weather, in addition to email and other Internet capabilities [2].

1.1. Features

The iPhone confirms that Apple understands consumers' desires, not only in terms of functionality, but also in terms of appearance and style. While other smart phone companies have offered products that include features offered by the iPhone, none have approached the iPhone in terms of popularity and sales. Phone features include a soft keypad with the ability to easily merge calls and visually obtain voicemail information. Apple took advantage of iPod's popularity by including complete iPod functionality in the iPhone. A full-functional web browser with zoom in/out functionality made internet surfing experience on a mobile phone better than ever. The Multi-Touch touch screen display allows for gliding and scrolling besides zooming. The accelerometer detects the orientation of the phone. These features put iPhone above and beyond other smartphones such as Blackberry and Motorola Q.

1.2. Hardware

The iPhone uses the ARM 1176JZF-S processor, which offers good power management for superior battery life and powerful processing for 3D graphics. Further details regarding this processor are available on the ARM product website [3]. **Figure 1** shows how different functions within the iPhone interface with one another [4]. **Figure 2** shows an image of the board inside an iPhone.

2. Motivation

iPhones are supposed to only be used with AT & T wireless service (in the US). AT & T agreed to give a portion of its revenue to Apple per each new contract it signed with iPhone users. This agreement spawned outrage among users of other GSM-based wireless services such as T-Mobile since they could not offer services to iPhone customers. Many people viewed this as an "unfair" move by the two companies. People felt that they should be

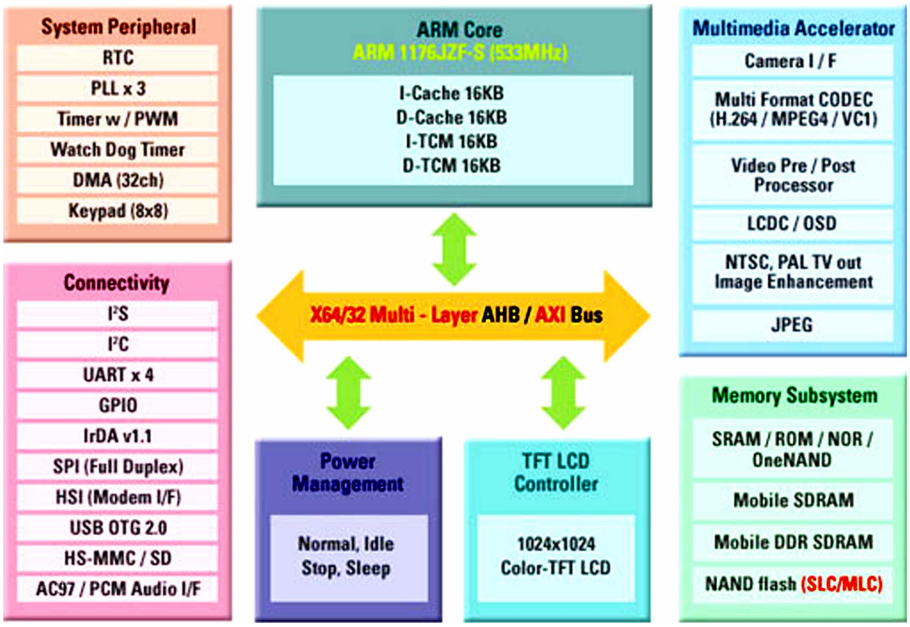


Figure 1. iPhone architecture from a high level [4].

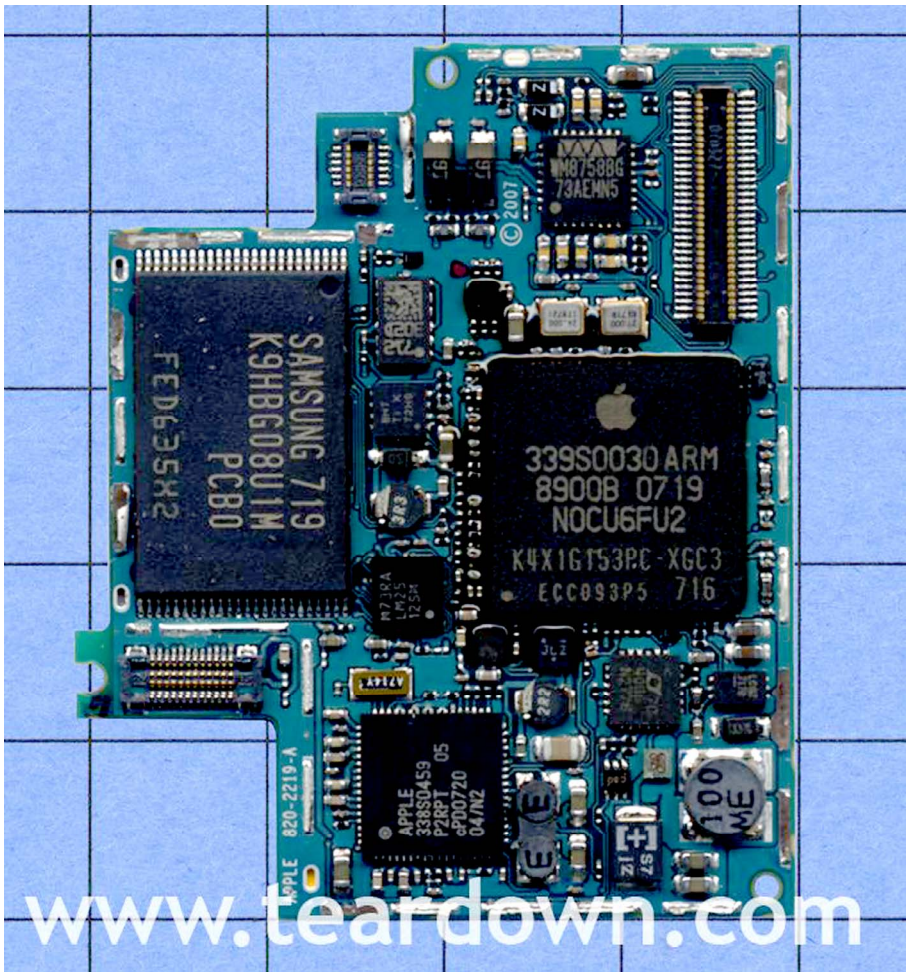


Figure 2. Board showing different parts in iPhone.

able to choose whatever wireless service they prefer and should not be forced to use a particular one.

There was another reason that some iPhone users became irritated. Apple designed iPhone as a closed system that does not allow installation of third-party applications. Users can only access a very small subset of the file system, a "sandbox" where they can add and remove music and other files via iTunes. Users who wanted to install third-party applications such as widgets and games were unable to do so.

These two limitations placed on iPhone users prompted a series of hack and attack efforts by iPhone enthuseasts and hackers. "Jailbreak" is an iPhone hack that permits the addition of third-party applications or gadgets on the iPhone by permitting read/write access to the root file system. Without "jailbreaking" an iPhone, a customer is limited to the factory-installed tools included with it. "Unlock" is an attack on iPhone that allows it to be used with any wireless service offering the GSM standard, not just AT & T. Without "unlocking" an iPhone, one can only use AT&T's wireless services. Perhaps surprisingly, jailbreaking is the more important of the two because it is the first step to unlocking. We look at a jailbreak attack in detail and also discuss different unlocking solutions.

Due to the commercial success of the iPhone, it makes a good candidate for security analysis. Having close to a million iPhones jailbroken and unlocked within first six months of its release, iPhone security obviously has had significant financial implications. In addition, with more millions of users worldwide, any security holes in iPhone can jeopardize the privacy of millions of people. We believe that these issues make the security analysis of iPhone a worthwhile and important topic.

3. Jailbreaking

The process of gaining root access to the iPhone so that third party tools can be installed is called Jailbreaking [5]. Without gaining read-write access to the root system, one cannot install third party applications. Note that this limitation prevents users from doing what they want to do with their iPhones—products that they own. This is somewhat analogous to buying a computer and not being allowed to install new programs on it. There are several websites (see, for example, [6]) that provide interesting gadgets and games for iPhone. Some of the most popular games are iSolitaire, iZoo, Tetris, iPhysics, and NOIZ2SA. Beyond providing access to such applications, jailbreaking is essential for another reason: it is the first step in unlocking.

Without jailbreaking, one cannot install the necessary application to use a wireless service other than AT & T. Close to a million new iPhones were not activated with AT & T in the first six months after its release [1]. Without jailbreaking, these iPhone owners would not be able to use the phone part of the iPhone unless they signed a contract with AT & T after switching from their existing GSM wireless service provider. Even for AT & T customers, jailbreaking is still necessary to enable the addition of third party applications to the iPhone.

3.1. Looking for Ideas

Immediately after its release, iPhone enthusiasts and hackers all around the world were looking for a way to gain root access. A feasible solution has to be reasonably easy to use and should not take several hours to complete. Hackers investigated various techniques for meeting these requirements. They evaluated existing hacks for other phones and devices and searched for similar vulnerabilities in the iPhone [7,8].

A previous hacker success was using buffer overflow techniques on the Sony PSP. By exploiting vulnerability in the Tag Image File Format (TIFF) library, libtiff, used for viewing TIFFs, hackers were able to hack PSP to run homebrew games, which was otherwise prohibited [9].

Hackers inspected Apple's MobileSafari web browser to see if it could be targeted for the same vulnerability. It turned out that for firmware version 1.1.1 of the iPhone, MobileSafari uses a vulnerable version of libtiff [10,11]. The exploitable vulnerability in libtiff is documented as entry CVE-2006-3459 in Commom Vulnerabilities and Exposures, a database tracking information security vulnerabilities and exposures [10]. This vulnerability is also documented and tracked in the U.S. National Vulnerability Database [12]. A malicious TIFF file can be created to include the desired rogue code. When attempting to view the malicious tiff file in a vulnerable version of MobileSafari, the vulnerabilities in libtiff are exploited to create a stack buffer overflow, and the malicious code is injected and executed.

3.2. Stack Buffer Overflow and Return-To-Libc Attacks

The attack we review, which exploits the libtiff vulnerability, uses a stack buffer overflow to inject code and the "return-to-libc" technique to execute it. To illustrate how a stack buffer overflow can be created and how a return-to-libc attack works, we first consider a generic example.

Consider the piece of code below [13]:

```
void func (char *passedStr) {
        char localStr[4]; // Note that only 4 bytes allocated
```



```
        strcpy(localStr, passedStr); // length of passedStr is not checked
}
int main (int argc, char **argv) {
        func(argv[1]);
}
```

Suppose that we have a program is called myprog. Now, let us look at a simplified representation of the stack when myprog is executed with "hi" as the input parameter—see **Table 1** below.

Now, consider the stack when myprog is executed with the string "goodsecurity."

As it is clear from the tables above(**Table 1, 2**), our program is only capable of handling a string with three characters plus NULL. When a string of more than three characters is passed, the extra characters cause stack buffer overflow and overwrite other sections of the stack [14]. Of course, the function func() should have performed a string length check on passedStr to ensure that it has three characters or fewer before the NULL. Any piece of code that makes a mistake similar to this is potentially vulnerable to a stack buffer overflow [14,15].

Instead of entering "good security," a carefully crafted string could be used. In the example above, suppose we replace "good security" with, say, "good secu\x12\x34\x56\x78." In little-endian, the last 4 bytes are 0x78563412, which might be the address of a function, say, system(). Then when the stack unwinds, instead of execution returning to the calling function, the pre-existing function indicated by the overwrite bytes will be executed—in this case, system(). Moreover, the stack could be overwritten so that desired parameter values are passed to a pre-existing function [16]. Such an attack is generally known as the return-to-libc attack. By discovering the address of such a desirable function, an attacker can potentially exploit a buffer overflow to execute the function and thereby achieve the desired behavior. Furthermore, by passing a carefully crafted malicious input that exploits a stack overflow, an attacker can even inject malicious code that results in a chain of calls to such pre-existing functions.

Table 1. Simplified stack representation with proper input.

Parent function's stack
Return address (4 bytes)
char* passedStr
hi\0 (4 bytes allocated for localStr. so String up to 3 characters is a good input)

Table 2. Simplified stack representation with corrupting input.

Parent function's stack
"rity" (return address overwritten)
"secu" (char* passedStr overwritten)
"good" (expected 3 characters + \0, got 12)

3.3. Libtiff Vulnerability

A vulnerability similar to that in the example above is found in libtiff version 3.8.1 and earlier—an area of memory is accessed without performing an out-of-bounds check. The vulnerability is in function TIFFFetchShortPair in the tif_dirread.c file [10]. That function fetches a pair of bytes or shorts, as the name implies. It should throw an error if the request is to fetch more than two bytes or shorts. Instead, it fetches any arbitrary number of bytes requested. This vulnerability was fixed in libtiff version 3.8.2. The source code for both versions of libtiff can be downloaded from the Maptools.org website [17]. Below we give excerpts of this function as it appears in libtiff versions 3.8.1 and 3.8.2. First, we look at the snippet from version 3.8.1:

```
static int
TIFFFetchShortPair(TIFF* tif, TIFFDirEntry* dir)
{
        switch (dir->tdir_type) {
                case TIFF_BYTE:
                case TIFF_SBYTE:
                        {
                        uint8 v[4];
                        return TIFFFetchByteArray(tif, dir, v)
                                && TIFFSetField(tif, dir->tdir_tag, v[0], v[1]);
                        }
                case TIFF_SHORT:
                case TIFF_SSHORT:
                        {
                        uint16 v[2];
                        return TIFFFetchShortArray(tif, dir, v)
                                && TIFFSetField(tif, dir->tdir_tag, v[0], v[1]);
                        }
                default:
                        return 0;
        }
}
```

Now, let us look at the snippet from version 3.8.2, which has the fix for the vulnerability. The fix is obvious from the developer's comments.

```
static int
TIFFFetchShortPair(TIFF* tif, TIFFDirEntry* dir)
{
```



```
        /*
         * Prevent overflowing the v stack arrays below by performing a sanity
         * check on tdir_count, this should never be greater than two.
         */
        if (dir->tdir_count > 2) {
            TIFFWarningExt(tif->tif_clientdata, tif->tif_name,
            "unexpected count for field \"%s\", %lu, expected 2; ignored",
                _TIFFFieldWithTag(tif, dir->tdir_tag)->field_name,
                dir->tdir_count);
            return 0;
        }

        switch (dir->tdir_type) {
            case TIFF_BYTE:
            case TIFF_SBYTE:
                {
                uint8 v[4];
                return TIFFFetchByteArray(tif, dir, v)
                    && TIFFSetField(tif, dir->tdir_tag, v[0], v[1]);
                }
            case TIFF_SHORT:
            case TIFF_SSHORT:
                {
                uint16 v[2];
                return TIFFFetchShortArray(tif, dir, v)
                    && TIFFSetField(tif, dir->tdir_tag, v[0], v[1]);
                }
            default:
                return 0;
        }
    }
```

To take advantage of the vulnerability in the TIFF library, a malicious TIFF file must be constructed. To accomplish that requires a reasonable working knowledge of the TIFF file format. There are two important objectives to keep in mind while constructing a malicious TIFF file: causing buffer overflow and injecting code. The iPhone is constructed around an ARM processor, thus some knowledge of it is required for successful code injection. Next, we discuss the TIFF format and give a brief overview of the ARM processor.

3.4. TIFF

The TIFF standard is owned and maintained by Adobe. It is tag-based format used primarily for scanned images [18]. A TIFF file has a header section and descriptive sections at the top of the file with offsets pointing to the actual pixel image data [19]. This means that a poorly constructed file may have tags pointing to incorrect offsets or offsets beyond the end of the file. Such aberrations can be used to exploit a buffer overflow in poorly written programs that read and manipulate tiff images [19]. Some examples of tags include image height, image width, planar configuration, and dot range. Different tags give necessary information about the image including color, compression, dimensions, and location of data. Below is an example of a tiff file ("value" column) with corresponding descriptions [18].

Offset (hex)	**Description**	**Value (numeric values are expressed in hexadecimal notation)**
Header:		
0000	Byte Order	4D4D
0002	42	002A
0004	1st IFD offset	00000014
IFD:		
0014	Number of Directory Entries	000C
0016	NewSubfileType	00FE 0004 00000001 00000000
0022	ImageWidth	0100 0004 00000001 000007D0
002E	ImageLength	0101 0004 00000001 00000BB8
003A	Compression	0103 0003 00000001 8005 0000
0046	PhotometricInterpretation	0106 0003 00000001 0001 0000
0052	StripOffsets	0111 0004 000000BC 000000B6
005E	RowsPerStrip	0116 0004 00000001 00000010
006A	StripByteCounts	0117 0003 000000BC 000003A6
0076	XResolution	011A 0005 00000001 00000696
0082	YResolution	011B 0005 00000001 0000069E
008E	Software	0131 0002 0000000E 000006A6

009A	DateTime	0132 0002 00000014 000006B6
00A6	Next IFD offset	00000000
Values longer than 4 bytes:		
00B6	StripOffsets	Offset0, Offset1, ... Offset187
03A6	StripByteCounts	Count0, Count1, ... Count187
0696	XResolution	0000012C 00000001
069E	YResolution	0000012C 00000001
06A6	Software	“PageMaker 4.0”
06B6	DateTime	“1988:02:18 13:59:59”
Image Data:		
00000700		Compressed data for strip 10
xxxxxxxx		Compressed data for strip 179
xxxxxxxx		Compressed data for strip 53
xxxxxxxx		Compressed data for strip 160 …

The first two bytes in an Image File Directory (IFD) represent the number of directory entries (14 in the example above). The IFD then consists of a sequence of tags, 12 bytes each, where the first two bytes identify the field, and the next two identify the field type: short int, long int, byte, or ASCII. The next four bytes specify the number of values, and the final four specify the value itself or an offset to the value [18]. Since TIFF files are not intended to be human-readable, their contents are best viewed in a hex editor.

3.5. Arm Processor

Since the ARM1176JZF-S processor is used in the iPhone, some working knowledge regarding its architecture and instruction set is required for this study. ARM is a RISC-based processor. **Figure 3** gives a high-level diagram of ARM1176JZF-S.

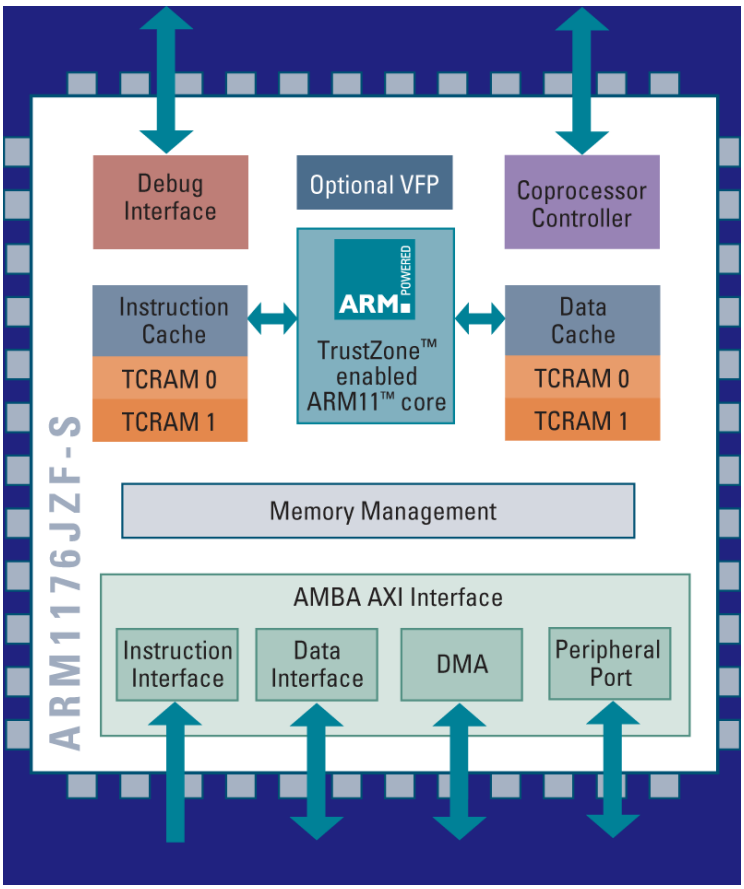


Figure 3. ARM 1176JFZ-S processor [3].

The ARM processor can be configured in either little- or big-endian modes to access its data [20]. The iPhone runs the ARM processor in little-endian mode. For example, if a value in a register is 0x12345678, in little-endian mode it appears in memory "byte-reversed", that is, as 0x78 0x56 0x34 0x12. This is illustrated in the **Figures 4** and **5** below.

The ARM processor can be configured in either little- or big-endian modes to access its data [20]. The iPhone runs the ARM processor in little-endian mode. For example, if a value in a register is 0x12345678, in little-endian mode it appears in memory "byte-reversed", that is, as 0x78 0x56 0x34 0x12. This is illustrated in the **Figures 4** and **5** below.

3.6. Dre And Niacin'S Tiff Exploit Jailbreak

We now have accumulated the background required to understand and reverse-engineer the libtiff exploit for jailbreaking developed by two teenagers known as Dre and Niacin. The source code for the attack is available on Dre and Niacin's website [23]. However, little explanation is provided, so we found it necessary to reverse engineer various aspects of the attack.

First, we verify and demonstrate the overflow problem. Though the exploit was created for the iPhone, we demonstrate the overflow on a Windows PC in cygwin to mimic a Unix-like environment. First the exploit source code was downloaded and compiled. Then, a malicious TIFF badDotRange.tiff was created.

An interesting outcome occurred when we attempted to create the code badDotRange.tiff. The file creation was blocked by Norton AntiVirus software running on the machine, and it claimed the file was "Bloodhound. Exploit.166" [24]. Further information on the vulnerability shows Norton characterizing badDotRange. tiff as a Trojan and a Virus, as shown in **Figure 6** [24].

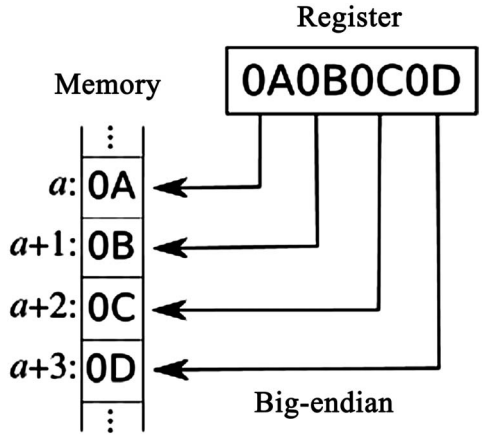


Figure 4. Big-endian [22].

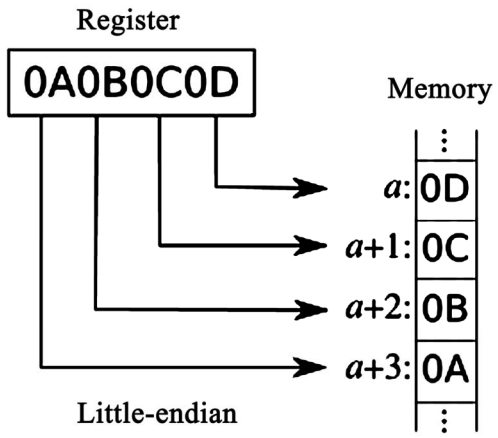


Figure 5. Little-endian [22].

Once the work area was put in the list of directories to be excluded by Norton AntiVirus, badDotRange.tiff was created; a hex editor view of the file is available in [25].

Next, we demonstrate the malicious TIFF file causing a buffer overflow in libtiff. We also show a well formed TIFF file being handled properly by libtiff. A program was written to simulate the stack buffer overflow. Below is a snippet from driver.cpp file.

```
int main() {
   cout << "Start!" << endl;
   TIFF* tif = TIFFOpen("c:/thesis/tiffExp/t1.tiff",
"r");
   if (tif) {
      cout << "Opened file successfully" << endl;
   } else {
      cout    << "FAILED to open tiff file" << endl;
   }
   TIFFClose(tif);
   cout << "End!" << endl;
   return 0;
}
```

Next, badDotRange.tiff is copied to t1.tiff and driver.cpp is compiled, linked with libtiff.a, and run, which results in a segmentation fault, as shown below.

```
$cp badDotRange.tiff t1.tiff
$g++ -I /usr/local/include –g driver.cpp –c
$g++ driver.o –L. –ltiff –o driver.exe
$./driver.exe
Start!
Segmentation fault <core dumped>
```

The program execution sequence is the following: TiffOpen() calls TIFFReadDirectory(), which upon encountering the DotRange tag calls TIFFFetchShortPair () as can be seen from the following snippet from tif_dirread.c.

```
case TIFFTAG_DOTRANGE:
        (void) TIFFFetchShortPair(tif, dp);
```

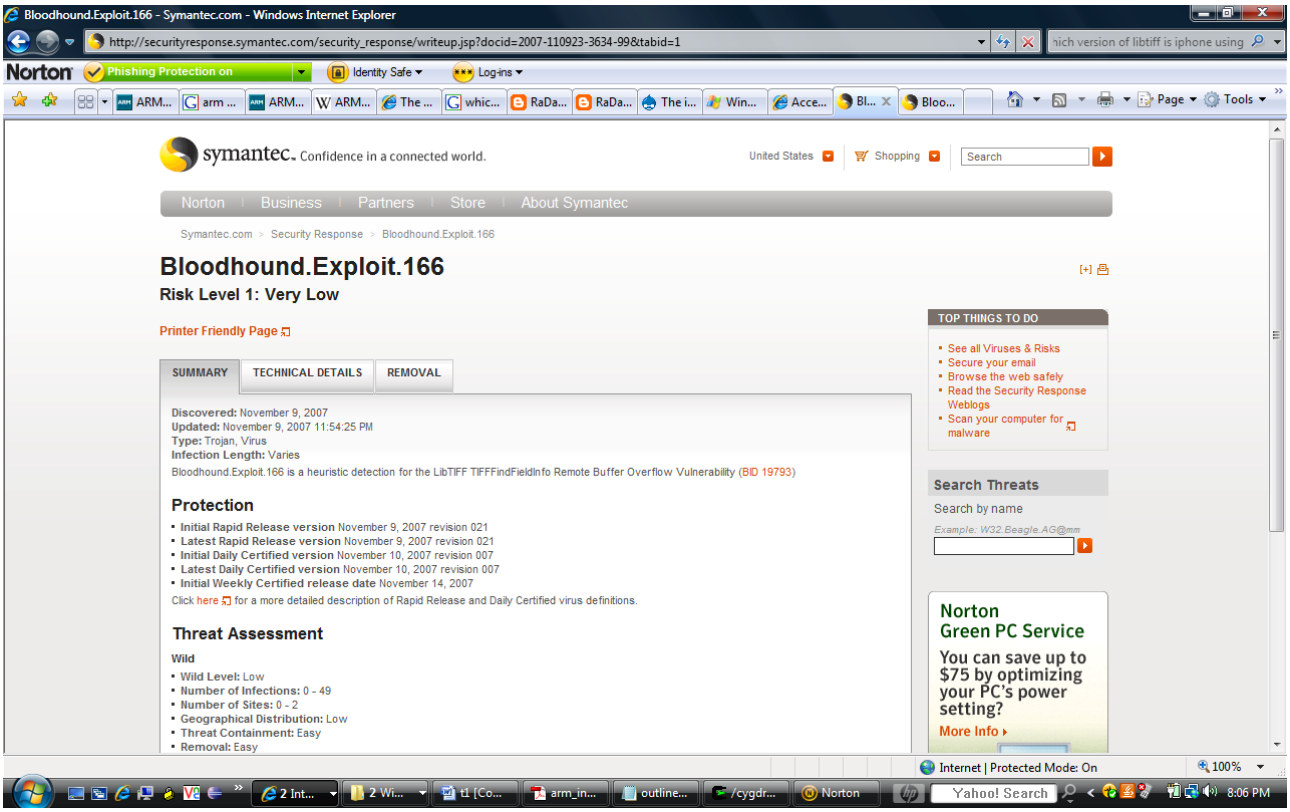



Figure 6. Bloodhound.Exploit.166 trojan [24].

break;
case TIFFTAG_REFERENCEBLACKWHITE: …

As seen earlier, that function allocates memory for two shorts, but instead receives the request to fetch 255 of them. Below is the corresponding line in the source code of the attack.

0x50,0x01,0x03,0x00,0xff,0x00,0x00,0x00,0x84,0x00,0x00,0x00,

Since we are assuming little-endian representation, the first two bytes become 0x0150, which represents the DotRange tag. The next two bytes give us the value 0x0003, which means the data type is SHORT. The next four bytes give us the number of different values for this tag, which is 0x000000ff or 255 in decimal. Finally, the final four bytes give us 0x00000084, which is the offset to the actual values for the tag [18].

By looking at the TIFF specification [18] and also looking at the code for the version of libtiff with sanity check [17], we see that the number of parameters expected by DotRange is two. As seen in the stack buffer overflow example, attempting to fetch 255 shorts causes a stack buffer overflow. In our example, the program overwrites the return value in the stack, changing it to some area in memory that is not accessible, resulting in a segmentation fault. Below, the line in badDotRange.tiff corresponding to the DotRange tag is shown, as it appears in Hex Editor. The twelve bytes corresponding to the DotRange tag appear from 0x74 to 0x7f.

0000070: 0100 0000 5001 0300 ff00 0000 8400 0000 ....P.....

Thus far, we have solved half of the problem of creating an attack by gaining control of the stack. Before we move on to injecting particular code and executing it, we first confirmed that a well-formed TIFF file is not recognized as a virus by Norton AntiVirus and does not cause a crash when opened with our program.

We now consider the code that provides root access to the iPhone and observe how it is executed. As mentioned earlier, this exploit uses the return-to-libc technique to execute a sequence of pre-existing functions. These pre-existing functions come from the dynamically loaded libSystem. dylib, which can be disassembled and searched for blocks of code that perform desired tasks [26]. The iPhone only allows access to a small section of the file system to add and remove music and other files. This “sandbox” area is the directory /var/root/Media. The algorithm used in the exploit renames /var/root/Media to /var/root/OldMedia. It then creates a symbolic link with /var/root/Media pointing to root, “/” and next it remounts root with the “MNT_UPDATE” flag to make it writable

[23]. The malicious tiff file is crafted skillfully to set up the stack to call the necessary functions from libSystem.dylib. Each of those functions must be studied carefully to discover how many values it reads from the stack and in what registers. The stack pointer must be set appropriately, and the link registers must be set properly for the next function call. With this method the exploit uses pre-existing functions to make the iPhone root writable—in other words, it "jailbreaks" the iPhone.

3.7. Summary of Jailbreaking

Let's recap the tools needed and the process taken for jailbreaking method used above. Vulnerability in tiff library was targeted to create a stack buffer overflow and inject desired code. Then return-to-libc technique was used to execute desired code to make the root directory of iPhone writeable – i.e. to jailbreak it. During the process knowledge of TIFF was necessary in order to construct a vulnerable TIFF file. Also, knowledge of ARM processor architecture and it's deficiencies were required to ensure the attack works consistently on any given iPhone. Furthermore, knowledge of ARM instructions was required to construct the code for the attack. In summary, a great deal of research and learning was required in order to pick up the necessary tools to successfully create the Jailbreak attack.

4. Unlocking

The iPhone is considered unlocked when it is able to use a cellular service other than that of AT & T. There are several free and paid software unlocking solutions available on the Internet including AnySIM, TurboSIM, and SimFree. Among these solutions, AnySIM seems to be quite popular, likely because it is free. It is developed by a group of people who call themselves the iPhone dev team.

AnySim works by patching the firmware on the baseband [27]. We can predict that somewhere in the baseband firmware, there is code that checks whether the SIM card being used is AT & T's. If the check passes, the baseband allows the phone part of the iPhone to work normally; conversely, if the check fails, the phone function does not work. AnySim performs a patch to the firmware so that it skips the above check and jumps to the section of code that executes when the check passes [27]. This procedure unlocks the iPhone because a SIM card from any GSM wireless carrier can then be used to make phone calls. If the baseband firmware is upgraded or downgraded, the iPhone gets "un-unlocked", as the patch that skips the check will almost certainly no longer be part of the code.

SimFree, also known as iPhone SimFree or IPSF, is unlocking software that currently sells for approximately $60, and at one point cost $99 [28]. Since it is a paid product, details about how it works are not revealed. It claims not to rely on firmware patching, so a phone unlocked with SimFree should remain unlocked even when a baseband upgrade is performed [27].

TurboSim is another paid solution for unlocking. It tricks the iPhone SIM card checking function into thinking it is an AT & T SIM card by providing an International Mobile Subscriber ID (IMSI) and an Integrated Circuit Card ID (ICC-ID)—also known as SIM Serial Number (SSN). For TurboSim to work, it must be programmed with a valid AT & T SIM, which it copies for later use [29].

Following table summarizes the above mentioned unlocking methods.

Unlocking Method	Technique used
AnySim	Patch the baseband to skip AT & T SIM card check.
SimFree	Proprietary software application that patches the iPhone firmware
TurboSim	Tricks iPhone into thinking that it's SIM card is an AT & T SIM card

5. Jailbreaking and Unlocking Newer Versions of Iphone

As mentioned earlier, for the purposes of this project, iPhone firmware version 1.1.1 and baseband bootloader version 3.9 are assumed. As of 2008, Apple had released versions 1.1.2, 1.1.3, and 1.1.4 of the firmware. Also, the baseband bootloader version is 4.6 in some of the phones. Can these phones be jailbroken and unlocked?

We use a simple approach: on newer versions of the iPhone, we downgrade the firmware to version 1.1.1 and the bootloader to version 3.9. Then we use the known attacks to jailbreak and unlock the iPhone. Several hacker websites, including iphone.unlock.no, offer instructions on how to downgrade the firmware and bootloader, and they also have different firmware files available for download [27].

Unlocking is not possible if the iPhone has version 4.6 or higher of bootloader because that version requires a secpack—a special password—to modify the baseband [30] and unlocking cannot be achieved without modifying the baseband. Since version 3.9 of the bootloader does not require any passwords, the baseband can be modified, and unlocking can be achieved. For that reason a "bootloader downgrader" tool *gbootloader* was developed by George Hotz and made available to iPhone users [31]. The tool downgrades the bootloader from version

4.6 to version 3.9 so that a patch to the baseband can be made and the iPhone can be unlocked.

Several other small utilities have been developed in addition to the ones mentioned here, which allows users to sort out different versions of firmware, baseband, and bootloader and make appropriate choices. Tools have been developed to upgrade the firmware on jailbroken phones to pick up some of the latest features developed by Apple for the iPhone.

6. Other Malicious Attacks

Attacks that we have examined so far do not carry the intention to be malicious, though the libtiff attack certainly could be malicious, depending on the type of code injected. For jailbreaking, the code injected was non-malicious—both behavior and intention-wise. However, using the libtiff vulnerability, malicious code could certainly be injected for a malicious attack. Now, let us examine a couple of malicious attacks created by a group of researchers at Independent Security Evaluators by exploiting other vulnerabilities; those attacks give us further insight into iPhone security. Details of the attacks discussed below are not revealed; the goal of the researchers was to make Apple aware of some of the issues and not to let the hackers find out the details of the vulnerabilities and the attacks. The attacks expose well-known security weaknesses in the OS X operating system used in the iPhone, including lack of address randomization and an executable heap [32].

The first attack consists of an exploit written to attack Safari on the iPhone. When a malicious HTML document was visited using MobileSafari, the iPhone was forced to make a connection to an outbound compromised server controlled by the attackers. The attackers were then secretly and automatically able to obtain personal data including contacts, call history, text message, and voice mail from the attacked iPhone. Attackers concluded that further personal information including passwords and emails could have been obtained had they chosen to do so [32]. What makes this attack even more dangerous is the ease with which it can be carried out. A link to a compromised website could be sent via email, and the iPhone owner could be lured into visiting it. That is all it would take to capture all of the personal data of the iPhone owner.

A second exploit was written to perform physical actions on the phone such as making a system sound and vibrating [32]. This exploit was run on the iPhone when another malicious HTML was viewed using Safari browser. To make matters worse, certain API functions discovered during this exploit could have allowed it to send text messages, dial phone numbers, or even record audio and transmit it over the network [32]. This vulnerability is particularly dangerous since the phone bill or text message bill could be increased by the attacker, which could cost the iPhone's owner a significant sum. The attacker could also send maliciously provocative messages to the owner's contacts, which could result in personal or professional relationship problems.

These malicious exploits are, collectively, comparable to having one's iPhone stolen. If attacks like these become widespread, there is a potential that customers would reconsider buying the iPhone.

While details of the attacks above were not disclosed, let us look at the high level approach used in the above MobileSafari attacks. This information could certainly be used as a guideline for the attacks above, provided one is able to write appropriate payloads. The iPhone uses Webkit, an open source web browser engine used by Mobile Safari [33], which in turn uses the Perl Compatible Regular Expression Library (PCRE). One of the first versions of iPhone used a version of PCRE that was more than a year old. Several versions of PCRE had been released with several bug fixes since the version used by iPhone. One of the bug fixes found in the change log of a newer version 6.7 [34] follows.

A valid (*though odd*) *pattern that looked like a POSIX character class but used an invalid character after* [(*for example* [[,*abc*,]]) *caused pcre_compile*() *to give the error "Failed*: *internal error*: *code overflow" or in some cases to crash with a glibc free*() *error*. *This could even happen if the pattern terminated after* [[*but there just happened to be a sequence of letters, a binary zero, and a closing*] *in the memory that followed.*

Now, one can review the bug fix and immediately get ideas for possible attacks on the iPhone. Attackers used the above vulnerability and constructed a regular expression in an HTML file that attacked the vulnerability when the file was viewed in Safari. The HTML document used was constructed as below [35]:

```
<SCRIPT LANGUAGE="JavaScript"><!--
var re = new RegExp("[[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]]
[[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]]
[[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]]
[[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]]
[[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]]
```



```
[[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]][[**]]
[[**]][[**]]ABCDEFGHIJKLMNOPQRSTUVWXYZAB-
CDEFG[\x01\x02\x03\x04\x05\x06\x07\x09\x0b\x0e\x0f\x11\x12\x13\x14\x15\x17\x19\x1b\x1c\x1d\x1f\x20\x21\x22
\x23\x25\x26\x27\x29\x2a\x2b\x2c\x2d\x2f\x30\x32\x33\x35\x37\x39\x3a\x3b
\x3c\x3e\x3f]XYZABCDEFGHIJKLMNOPQR");
</script>
```

To develop the exploit, attackers resorted to a technique called "fuzzing" [35], which involves passing different inputs that cause a given program to crash and then analyzing the crash to gain insight about the program. From the crash reports, they were able to get useful information such as the stack pointer and values in different registers. They then employed a technique to overwrite the return address on the stack to point to the heap area where shell code was injected [35]. The shell code then executed and did the job of stealing private information. The code consisted of typical socket connect, open, read, and write functions. The researchers have revealed some of the functions they used to perform physical actions on the phone including making a system sound, dialing phone calls, and sending SMS text messages. Those functions include AudioServicesPlaySystemSound from the Audio Toolbox library and CTCallDial, CTSMSMessageCreate, and CTSMSMessageSend from the Core Telephony library [35]. The purpose of each function is clear from its name.

To summarize, vulnerabilities in PCRE were targeted by creating a malicious HTML file to create a buffer overflow, which facilitated injection and execution of malicious code.

7. Security Analysis

Having briefly examined several vulnerabilities in the iPhone and attacks that exploit those vulnerabilities, we now analyze the iPhone security structure from a high level. What was the approach Apple took while designing the security architecture for the iPhone? Were there flaws in this philosophy? What high-level approaches can be used to exploit the security flaws? What are some of the ways that Apple can either fix some of the vulnerabilities or at least make it difficult for an attacker to exploit them? Let us try to answer some of these questions.

It is clear that iPhone is a vulnerable device with several security holes. The iPhone security philosophy itself has a signifcant flaw. Apple's approach to making the iPhone a secure device was to reduce "the attack surface of device" or "the device's exposure to vulnerabilities" [32]. To achieve this, Apple allowed write access only to a sandbox area in the file system and disallowed installation of third-party applications. Several features of Safari were removed in Mobile Safari, including the ability to use plug-ins like Flash and the ability to download certain file types. Mobile Safari was restricted to only execute Javascript code, and only do so in the sandbox area. In short, Apple's approach was to make a controlled, essentially closed-box device. Apple's security approach might be summed up by the following analogy: rather than teaching a child how to swim to prevent him from drowning, he is simply not allowed to jump in a lake.

While the security philosophy is debatable, the architecture has significant holes. Since Apple banked on preventing the iPhone from being compromised in the first place, it put very little effort into protecting different parts of the device individually. This conclusion is supported by the fact that all significant processes run as a super user or with administrative privileges—a major mistake from a security perspective. A result of this configuration is that an attacker is likely able to control the entire iPhone if he is able to exploit any vulnerability in any of its applications [32]. For example if Mobile Mail were compromised by an attack, the attacker could also gain access to contacts and pictures. In simple terms, the iPhone's security architecture looks like a home owner putting all effort for securing his or her home into buying a strong lock to stop an intruder from getting in. No effort is made to, say, secure individual room, to put valuables in a safe-deposit box, to use a home security system, etc. While it may be difficult to enter the house, if a thief can do so, he can easily steal all its contents.

A security hole is also created by the fact that the iPhone uses several applications including MobileSafari and MobileMail that are based on open source projects. While the use of open source is itself likely a good idea, using (and sharing) of open source projects with old and outdated versions of those projects is clearly a problem. Earlier we looked at examples of an old version of libtiff library facilitating the jailbreak attack, and an old version of the PCRE library allows another malicious attack. By using outdated versions of open source projects, Apple made it relatively easy for hackers to develop ideas and approaches for attacks attacks.

Apple also failed to make the exploitation of vulnerabilities challenging for hackers. By not utilizing common techniques such as Address Space Layout Randomization (ASLR) or non-executable heap in the version of OS X used for iPhone, Apple has not posed any

particular difficulties for hackers in the development and distribution of buffer overflow exploits [32].

The table below summarizes the attacks discussed in this paper.

Attack	Vulnerability targeted	Tools used	Effects
Jailbreaking	Vulnerable libtiff	TIFF, buffer overflow, return-to-libc, ARM architecture and instructions	Get root access
Unlocking	Jailbroken phones allow for installation of unauthorized applications	Installation of unauthorized application	Being able to use the iPhone with non-AT&T wireless services
Mobile safari (malicious)	Vulnerable PCRE	Malicious HTML, fuzzing, buffer overflow	Stolen personal data and other malicious effects

Apple did employ some good practices and has shown more effort recently in making the iPhone more secure. That has not stopped the hackers, however, as they have found solutions to the obstacles presented by Apple. For example, the stack is non-executable in the iPhone, so an attacker cannot simply add payload to the stack via a buffer overflow and execute it. However, a non-executable stack does not protect against the return-to-libc attack, which was employed in the jailbreaking attack, as we observed earlier. New versions of firmware have been released with certain vulnerabilities fixed to prevent jailbreaking. Unfortunately, these have been somewhat countered by the ability to downgrade the firmware. Apple also attempted to prevent unlocking by using a new version of the bootloader. That attempt failed because hackers found a way to downgrade the bootloader as well.

After evaluating Apple's security for the iPhone, one can safely conclude that overall the company failed to make the iPhone as secure as it could possibly have been. Looking at the security approach and the decisions the company made, it is no surprise that the initial iPhones were considered a fairly vulnerable device.

8. Analysis of Sample Decisions by Apple

Now that we have had a chance to analyze the iPhone's security structure, we can ask several questions regarding different choices Apple has made. Why are they using versions of open-source based packages that are about a year out of date? Why did they choose to have almost all important processes run as super user? Why did they not use ASLR? Why did they use a vulnerable version of the tiff library? This final question is particularly important because even after three new versions of firmware and a new version of the bootloader, Apple was still paying for this mistake.

It seems implausible that Apple had no knowledge of the vulnerability in libtiff that causes buffer overflow, since this vulnerability is well known in the hacking community and other mobile devices including Sony's PSP had been hacked using it. We can only speculate as to why Apple used the vulnerable version of libtiff. Perhaps there was an existing version of Safari with the vulnerable version of libtiff ready to be used with iPhone. One can certainly see that there is some cost involved in using a new version of libtiff in Safari, which would have to be thoroughly tested prior to being deployed in a new version for iPhone. Perhaps Apple found that there were other known vulnerabilities in the version used anyway. Perhaps Apple performed a cost analysis of losses suffered by delaying the new version of firmware versus losses due to the number of people who would hack the iPhone to jailbreak it and eventually unlock it and use a wireless service other than that of AT & T. Such a decision would express disregard for consumer security, since the same vulnerability could be also used to perform truly malicious acts.

From a short-term perspective, it is hard to argue with the success of the iPhone. However, from the consumer confidence or reputation perspective, the situation is not so clear. Apple is generally regarded as a company that delivers secure and robust products. They may have lost some of that sheen with the iPhone.

9. Suggestions to Improve Security Structure

We have pinpointed several flaws in the initial iPhone security structure. A large security hole would have been filled if most of the processes were not run with administrative privileges, or as the super user. This would generally make it more difficult for an attacker to gain full control of an iPhone.

While using open-source based applications is a good idea, Apple needs to be more cognizant about using versions that do not have serious known bugs. Apple should also use a technique such as ASLR for heap and stack address randomization to make it more difficult for hackers to develop stable attacks and distribute them [32]. Moreover, it could develop a mechanism that prohibits both writing to and executing an area of the heap. Some attacks copy the exploit payload into the heap area that is both writeable and executable, and they execute it there. If an area in heap was not both writeable and executable,

such attacks would be thwarted. Also, if ASLR were employed, even if an attacker could successfully write an attack that relies on an address in the stack or heap, distribution of the attack would be difficult, as the target address is unreliable due to randomization.

10. Conclusions

In this paper, we considered the iPhone security structure and its vulnerabilities. The Jailbreaking attack analyzed here relied on a known vulnerability in the TIFF library. The analysis of the attack required some knowledge of the ARM architecture and the TIFF file format. We showed that using a vulnerable version of the TIFF library proved costly for Apple, in the sense that updates could not easily prevent "rollback" attacks. Interestingly, hackers found ways to jailbreak later iPhone without even losing the new features introduced in newer versions. Perhaps predictably, the attacks on the iPhone and the countermeasures by Apple quickly devolved into a cat and mouse game.

The security problems discussed here have resulted in financial losses for both Apple and AT&T and, arguably, a reputation loss for Apple. For each iPhone that was unlocked to access an alternate wireless carrier, AT & T stood to lose about $1500 in revenue for the two-year contract period. As we noted earlier, the number of unlocked iPhones was estimated at nearly a million in just its first six months [1]. Apple too missed out on some gains, as it receives a certain amount from AT & T for each iPhone activated with AT&T. The security vulnerabilities of the iPhone have also affected Apple's reputation as a company, as it had been generally believed to deliver relatively secure products. While Apple's exclusive deal with AT & T and its decision to use a closed system undoubtedly increased the motivation to attack the iPhone.

We have also explained that malicious attacks can be created for the iPhone. However, the significant attacks have not been malicious, but were instead focused on enabling people more freedom to do what they want with their telephone product.

We conclude that Apple's initial effort in making the iPhone a secure device was somewhat disappointing. While Apple worked to improve iPhone security, the initial release unnecessarily gave hackers the upper hand, which, to some extent, has continued to this day.

10. References

[1] C. Maxcer, "Apple Minus AT&T Equals Lots of iPhones Somewhere Else," Mac News World. http://www.macnewsworld.com/story/61389.html?welcome=1209968031

[2] iPhone, Apple–iPhone. http://www.apple.com/iphone/

[3] ARM, ARM1176 Processor. http://www.arm.com/products/CPUs/ARM1176.html

[4] A. L. Shimpi, "Apple's iPhone Dissected: We did it, so you don't have to," Anandtech, 29 June 2007. http://www.anandtech.com/mac/showdoc.aspx?i=3026&p=3

[5] In brief, Network Security, Vol. 2009, No. 7, July 2009, pp. 3.

[6] Best iPhone Apps. http://www.Installerapps.com

[7] K Dunham, "Mobile Malware Attacks and Defense," Elsevier 2009, pp. 197-265.

[8] B. Haines, "Seven Deadliest Wireless Technologies Attacks," Syngress, 2010.

[9] Max Console. http://www.maxconsole.net/?mode=news&newsid= 9516

[10] Common Vulnerabilities and Exposures, 2006. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3459

[11] TIFF Library and Utilities, 15 January 2008. http://www.libtiff.org/

[12] National Vulnerability Database, 2006. http://nvd.nist.gov/nvd.cfm?cvename=CVE-2006-3459

[13] "Stack buffer overflow," Wikipedia. http://en.wikipedia.org/wiki/Stack_buffer_overflow

[14] M. Stamp, "Information Security: Principles and Practice," Wiley 2005.

[15] C. Cowan, *et al.*, "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks," *Proceedings of the 7th USENIX Security Symposium*, San Antonio, Texas, January 26-29, 1998.

[16] "Return-to-libc," Wikipedia. http://en.wikipedia.org/wiki/Return-to-libc

[17] Maptools, 15 January 2008. http://dl.maptools.org/dl/libtiff/

[18] Adobe Developers Association, TIFF Revision 6.0 Final, 3 June 1992. http://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf

[19] "Tagged Image File Format," Wikipedia. http://en.wikipedia.org/wiki/TIFF

[20] Simple Machines, The ARM instruction set. http://www.simplemachines.it/doc/arm_inst.pdf

[21] "1176JZF-S Technical Reference Manual Revision r0p7," ARM. http://infocenter.arm.com/help/topic/com.arm.doc.ddi0301g/DDI0301G_arm1176jzfs_r0p7_trm.pdf

[22] "Little-endian," Wikipedia. http://en.wikipedia.org/wiki/Little_endian

[23] Toc2rta, TIFF exploit. http://www.toc2rta.com/files/itiff_exploit.cpp

[24] "Bloodhound.Exploit.166 Technical Details," Symantec, 9 November 2007.

[25] V. Pandya., IPhone security analysis, Masters Thesis, Department of Computer Science, San Jose State University, 2008. http://www.cs.sjsu.edu/faculty/stamp/students/pandya_vaibhav.pdf

[26] Metasploit. http://www.metasploit.com

[27] iPhone UnlockUSA.com. http://iphone.unlock.no

[28] iPhone Sim Free. http://www.iphonesimfree.com

[29] Hackintosh, Turbosim Technical Background. http://hackint0sh.org/forum/showthread.php?t=18048

[30] Hackintosh, iPhone. http://www.hackint0sh.org

[31] G. Hotz, "On the iPhone," 15 February 2008. http://iphonejtag.blogspot.com/

[32] C. Miller, J. Honoroff and J. Mason, "Security Evaluation of Apple's iPhone," Independent Security Evaluators, 19 July 2007. http://securityevaluators.com/files/papers/exploitingiphone.pdf

[33] The Webkit Open Source Project. http://webkit.org/

[34] Perl Compatible Regular Expressions, Change log. http://www.pcre.org/changelog.txt

[35] C. Miller, "Hacking Leopard: Tools and Techniques for Attacking the Newest Mac OS X," Black Hat Media Archives, 2 August 2007. https://www.blackhat.com/presentations/bh-usa-07/Miller/Presentation/bh-usa-07-miller.pdf

Journal of Information Security, 2010, 1, 88-94
doi:10.4236/jis.2010.12010 Published Online October 2010 (http://www.SciRP.org/journal/jis)

Denial of Service Due to Direct and Indirect ARP Storm Attacks in LAN Environment*

Sanjeev Kumar, Orifiel Gomez
Department of Electrical/Computer Engineering, University of Texas—PanAm, Edinburg, USA
Email: sjk@utpa.edu, sanjeevk@utpa.edu
Received October 3, 2010; *revised October* 16, 2010; *accepted October* 19, 2010

Abstract

ARP-based Distributed Denial of Service (DDoS) attacks due to ARP-storms can happen in local area networks where many computer systems are infected by worms such as Code Red or by DDoS agents. In ARP attack, the DDoS agents constantly send a barrage of ARP requests to the gateway, or to a victim computer within the same sub-network, and tie up the resource of attacked gateway or host. In this paper, we set to measure the impact of ARP-attack on resource exhaustion of computers in a local area network. Based on attack experiments, we measure the exhaustion of processing and memory resources of a victim computer and also other computers, which are located on the same network as the victim computer. Interestingly enough, it is observed that an ARP-attack not only exhausts resource of the victim computer but also significantly exhausts processing resource of other non-victim computers, which happen to be located on the same local area network as the victim computer.

Keywords: ARP Attack, Computer Network Security, Computer Systems, Direct Attack, Distributed Denial of Service Attacks (DDoS), Indirect Attack, Local Area Networks

1. Introduction

A Distributed Denial of Service (DDoS) attack [1,2] involves multiple DoS agents configured to send attack traffic to a single victim computer. DDoS is a deliberate act that significantly degrades the quality and/or availability of services offered by a computer system by consuming its bandwidth and/or processing time. As a result, legitimate users are unable to have full quality access to a web service or services. A Denial of Service attack consumes a victim's system resource such as network bandwidth, CPU time and memory. This may also include data structures such as open file handles, Transmission Control Blocks (TCBs), process slots etc. Because of packet flooding in a DDoS attack that typically strives to deplete available bandwidth and/or processing resources, the degree of resource depletion depends on the traffic type, volume of the attack traffic, and the processing power of the victim computer.

According to Computer Emergency Response Team Coordination Center (CERT/CC) [3], there has been an increase in use of Multiple Windows-based DDoS agents. There has been a significant shift from Unix to Windows as an actively used host platform for DDoS agents. Furthermore, there has been an increased targeting of Windows end-users and servers. To raise awareness of such vulnerabilities, the CERT/CC published a tech tip entitled "Home Network Security" in July of 2001 [4]. According to the CERT/CC [3], there is a perception that Windows end-users are generally less security conscious, and less likely to be protected against or prepared to respond to attacks compared to professional industrial systems and network administrators. Furthermore, large populations of Windows end-users of an Internet Service Provider are relatively easy to identify and hence the attackers or intruders are leveraging easily identifiable network blocks to selectively target and exploit Windows end -user servers and computer systems.

In this paper, we consider a Distributed Denial of Service (DDoS) attack that can be caused by a barrage of ARP-requests sent to a victim computer. In order to understand the intensity of the attack, we conduct experiments in a controlled lab environment to measure the

*Work of Dr. Kumar is supported in part by funding from CITeC, FRC, FDC, OBRR/NIH, digital-X Inc, and US National Science Foundation.

availability of the processing power and memory resources of the victim computer during an ARP-attack. Since, windows based servers are very commonly deployed, we consider a Window-XP server with a 3.06 GHz Pentium-IV processor and 512 Mbytes of RAM to be used as the victim computer in the ARP-attack experiments. Section II presents a background on ARP and how it is used to exploit vulnerability of a computer system; Section III presents detail on use of ARP requests, ARP format, types of ARP-request traffic, and the processing that needs to be done for ARP-request messages; Section IV presents the experimental-setup, systems configuration for DDoS attacks in the controlled lab environment, and attack measurement results under direct ARP attack traffic and Indirect ARP attack traffic; and Section V provides discussion on detection and prevention schemes for ARP storm attacks, Section VI concludes the paper.

2. Arp-As an Attack Bullet

The Address Resolution Protocol (ARP) requests are legitimate and essential for the operation of the network. However, ARP can be used in more than one way to exploit the vulnerability of a computer system or a network. Some of the security attacks involving ARP can cause Denial of Service (DoS) attack by sending a massive amount of ARP requests to a victim computer and tying up its resource [5]. ARP can also be used to create Denial of Service attack by sending a victim computer's outgoing data to a sink by the technique of ARP cache poisoning. Other ARP based attacks can result in unauthorized sniffing of packets, or hijacking of secured Internet sessions. The Denial of Service attacks due to ARP storms can also be caused by worms such as code red due to their rapid scanning activity [6,7]. The worm initiated ARP storms have been commonly found in networks with high numbers of infected and active computers and servers. In ARP storm, an attacked victim (the gateway or a server) may receive a constant barrage of ARP requests from attacking computers in the same sub-network, and this ties up not only the network bandwidth but also the processing resource of the victim computer.

The worm Code-Red's rapid scanning activity can result in a denial-of-service attack against a Windows NT 4.0 IIS 4.0 server with URL redirection enabled [6]. The worm Code-Red can easily spread to new vulnerable systems, and there is a patch available for this vulnerability. Applying the patch can keep a server from being infected by the worm Code-Red. Nevertheless, it is still possible for the worm in other infected computers on the network to attack the same chain of IP addresses over and over again. This can generate a high-traffic overload due to massive amount of ARP requests generated in the network, which in turn can still affect the server's performance (despite the patch).

In this paper, we investigate the brute force of ARP attack where a constant barrage of ARP requests is directed to a victim computer. In this experiment, we set out to measure how bad the effect of the ARP attack was on the victim computer. Furthermore, we also measure the extent of resource exhaustion due to the ARP attack traffic on other computers located on the same LAN segment as the victim computer. To understand the degree of resource exhaustion, we measure performance in terms of processor exhaustion, occupancy of systems' memory and the page-file size. Since Microsoft Windows-XP based computers and servers with high performance Pentium-IV processors are becoming quite affordable and popular with small businesses, we use a Windows-XP based computer as a victim computer to be stress-tested for the extent of resource exhaustion under the ARP attack.

3. Processing an Arp-Request Message

3.1. Use of ARP-Request Message

A gateway or a host on a local area network uses ARP request broadcast messages [8] for IP and hardware address bindings. The ARP message contains the IP address of the host for which a hardware address needs to be resolved (**Figure 2**). All computers on a network receive ARP message and only the matching computer responds by sending a reply that contains the needed hardware address.

3.2. ARP Message Format

ARP is used for a variety of network technologies. The ARP packet format varies depending on the type of network being used. The ARP packet format used in Ethernet is shown in **Figure 1**. While resolving IP protocol address, the Ethernet hardware uses 28-octet ARP message format [8]. The ARP message format contains fields to hold sender's hardware address and IP address, shown as SENDER-HA and SENDER-IP fields in **Figure 1**. It also has fields for the target computer's hardware and IP address, which is shown as TARGET-HA and TARGET-IP fields in **Figure 1**. When making an ARP request, the sender supplies the target IP address, and leaves the field for the target hardware address empty (which is to be filled by the target computer).

In the broadcasted ARP request message, the sender also supplies its own hardware and IP addresses for the target computer to update its ARP cache table for future

correspondence with the sender. Other fields in the ARP packet format in **Figure 2** are HARDWARE TYPE of 2 Bytes (shown as 2B in **Figure 1**), which specifies the type of network being used such as Ethernet in this case. The PROTOCOL TYPE field of 2 Bytes specifies the high-level protocols address used such as the IP addresses. The fields HLEN and PLEN of one Byte each specify the length of hardware address and high-protocol address, in the case of ARP protocol use in the arbitrary networks. The OPERATION field of 2 Bytes specifies if the message is one of the four possible types *i.e.* 1 for ARP-request, 2 for ARP-reply, 3 for RARP-request and 4 for RARP-reply.

4. Types of ARP-Request Traffic on a LAN

A computer on the LAN will receive two different types of ARP-request packets from the network. The first type of ARP request packets can be named as the *direct ARP request traffic* where the IP address in the ARP request packet matches the local IP address of the computer *Pi*. The second type of ARP request traffic that is received by the computer on a LAN can be named as *indirect ARP request traffic* where the IP address in the ARP request packets doesn't match the local IP address of the computer P_i.

In other words, a computer *i* on a LAN with IP address

Harware Type (2b)		Protocol Type (2b)
Hlen (1b)	Plen (1b)	Operation (2b)
Sender Ha (Octets 0-3)		
Sender Ha (Octets 4-5)		Sender Ip (Octets 0-1)
Sender Ip (Octets 2-3)		Target Ha (Octets 0-1)
Target Ha (Octets 2-5)		
Target Ip (Octets 0-3)		

Figure 1. ARP mssage frmat

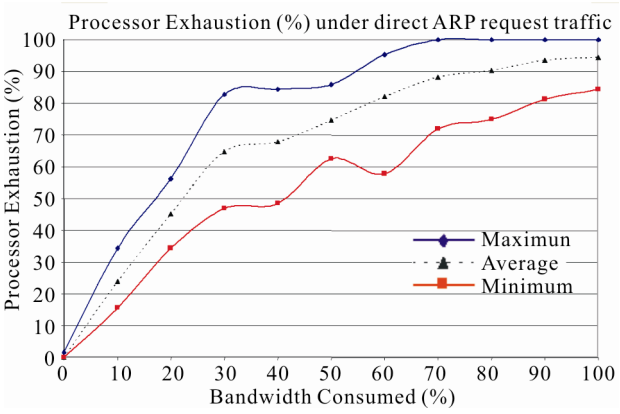


Figure 2. Processor exhaustion under direct ARP- attack traffic with IP address = $\{\chi \mid \chi = P_i\}$.

of P_i may receive one of the two possible types of ARP request traffic during an ARP-attack –

a) *Direct ARP traffic* – it is a traffic comprising of ARP request messages with IP address = $\{\chi \mid \chi = P_i\}$

b) *Indirect ARP traffic* – it is a traffic comprising of ARP request messages with IP address = $\{\chi \mid \chi \neq P_i\}$

The target or victim computer will primarily be inundated with the direct-ARP attack traffic, whereas the other computers (non-victim computers) located on the same LAN segment will be inundated with the indirect ARP-attack traffic.

The main task of the processor in the target computer after receiving the ARP request message is to make sure the ARP request message is for it. In the case of direct ARP frames, the processor proceeds to fill in the missing hardware-address in the ARP request format-header, swaps the target and sender hardware & IP address pair, and changes the ARP-request operation to an ARP-reply. Thus the ARP reply carries the IP and hardware addresses of both, the sender and the target computers. Unlike the ARP request message, the ARP replies are directed to just the sender computer and it is not broadcasted. In the case of indirect ARP frames received, the computer still does some processing to determine if the ARP request message is for the local computer. In this case, once it is determined that the frame is not for the local computer, the indirect ARP message is simply dropped.

The processing needed for an ARP-request message is fairly simple, however there is more processing involved when direct ARP request frames are received by a victim computer, compared to that of the indirect ARP-request frames received by non-victim computers present on the same LAN. Even though, there is comparatively less processing involved when an indirect ARP request message is received, a barrage of such requests can still exhaust the processing power of a non-victim computer just because it happens to be sitting on the same LAN segment as the victim computer or server. The degree of processor exhaustion for a given computer will of course depend on the processor speed and the bandwidth consumed by ARP-request messages. In the following sections, we discuss our experiment to measure the extent of resource exhaustion of two different types of computers on a LAN under an ARP attack – the first type of computer, being the victim computer, which is inundated with direct ARP-request frames. We also measure the computing resource exhaustion of the second type of computers (the non-victim computers, which happen to be on the same LAN as the victim computer or server), when inundated with indirect ARP-request frames.

5. Performance Evaluation

5.1. Experimental Setup

In this experiment, an ARP-storm was generated in a controlled environment of the network security research lab at the UTPA by having different computers send a barrage of ARP-request messages to a victim computer on the same local area network. A Windows-XP based computer was used as the attack target of the ARP-storm. The computer under attack deployed a Pentium IV processor with a speed of 3.06 GHz with 533 MHz Bus, 512 kb Cache, a physical memory of 512 Mbytes (RAM), and a NIC card from 3 Com. Furthermore, other computers (that received indirect ARP request traffic) on the LAN deployed exactly the same resources as the victim computer on the LAN. Computers under attack on the LAN deployed Windows-XP Service-Pack 2 (SPK 2). We also used the network observer software to collect traffic detail and the applied load on the LAN.

This experimental setup and results obtained in this paper are much more detailed compared to the one presented in [9] where a different system was used for the victim computer, which deployed a Pentium-4 processor with a speed of 2.66 GHz. Furthermore, the NIC card used in [9] were the Intel's NIC card, which could not support full speed of 100 Mbps of network traffic. Whereas, in this experiment, the 3 Com's NIC card was used that supported full speed of 100 Mbps. Furthermore, in [9] only the effect of direct ARP traffic was measured and no indirect ARP traffic was considered.

5.2. Attack Measurements

Parameters of performance evaluation considered for this attack experiment were the applied load of the ARP-attack traffic, processor exhaustion during the attack and memory occupied while processing the attack traffic by the target computer. The DDoS attack was simulated as ARP packets coming from multiple different attacking-computers at a maximum aggregate speed of 100 Mbps towards the target server. The attack traffic (while simulating ARP storm) load was started with 0 Mbps (the background condition) and was increased by 10 Mbps *i.e.* from 0% load to 100% load (= 100 Mbps). In the ARP-storm experiment, the attacked target computer continued to receive a barrage of ARP-requests for a period of 60 minutes for a given load, and was obligated to process them by creating an ARP-reply. In this experiment, a total of 10 different loads were generated, *i.e.* 10% - 100%. A total of 10 hours of ARP attack traffic were experienced at the victim computer and another non-victim computer on the local network. The CPU time is termed as processor exhaustion in these measurements, which gives an indication of the rate of processor exhaustion for a given bandwidth consumed by the attack-traffic during the ARP storm. It is observed that as the network bandwidth is increasingly consumed by the ARP-attack traffic, the processor is exhausted at a much faster rate, and hence this type of attack can be classified under computing-resource starvation attack.

5.3. Resource Exhaustion of the Victim Computer Due to Direct-ARP Request Traffic

Direct ARP request traffic comprises of ARP-request frames that have

$$\text{IP address} = \{\chi \mid \chi = P_i\}$$

In this experiment, we measure, processor exhaustion, memory used and the page file size under direct-ARP request traffic. Page file size gives indication of virtual memory activity, if any, during the attack.

Figure 2 shows minimum and maximum CPU time observed (called processor exhaustion in the attack experiments) for a given load of the direct ARP-attack traffic. Average CPU time is also shown in the graph so that we can get an idea if the majority of observations are closer to the maximum CPU time or closer to the minimum CPU time. It can be seen that a bandwidth consumption of 40% by direct ARP-attack traffic in a fast Ethernet environment exhausts a Pentium-IV processor to up to 85% of its 3.06 GHz processing capacity. Due to the processing of a barrage of ARP-requests the CPU resource is easily consumed and this in turn can degrade the quality and availability of associated web services.

Furthermore, it is obvious that if such servers are operated in a Gigabit network deploying higher interfaces such as 1 Gbps then it will be easier for such CPU of 3.06 GHz to be completely consumed by the Gigabit-flood of ARP-attack traffic, and attacks in such Gigabit environment can completely stall the system. Complete stalling of system means that one cannot even move the cursor on the attacked computer, let alone running the security diagnostics. It is also obvious from this experiment that a lower capacity (< 3.06 GHz) processor can easily be frozen (consumed 100%) by this type of ARP-storm in commonly available fast Ethernet environment of local area networks.

Figure 3 shows the memory-usage of the victim computer under direct ARP-attack traffic, as the network bandwidth is increasingly consumed by the ARP-storm. The memory consumed due to direct ARP attack traffic is observed to be within a range of 6 Mbytes, which seems to be not much of an issue for a 3.06 GHz processor with 512 Mbytes of RAM. However, for a slower

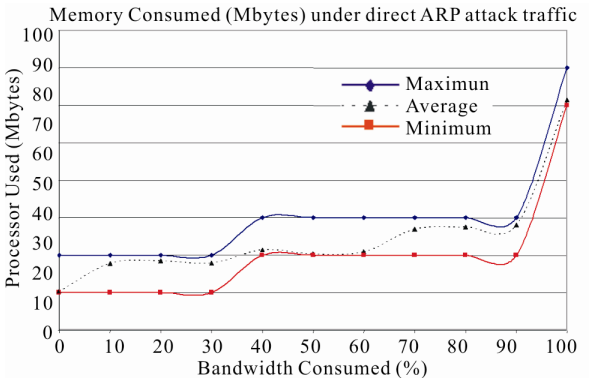


Figure 3. Memory usage under the direct ARP-attack traffic with IP address = $\{\chi \mid \chi = P_i\}$.

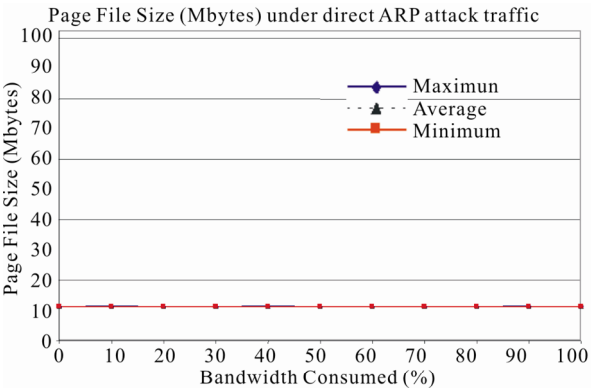


Figure 4. Page-file size is visibly unaffected under direct-ARP-attack traffic with IP address = $\{\chi \mid \chi = P_i\}$.

processor with processing power less than 3.06 GHz, a greater amount of computer's memory resource can be wasted. Slower processing power in the fast Ethernet environment can cause the queue of ARP packets to build up waiting for address resolution and computer's response. Hence a slower processor will exhaust a relatively greater amount of memory resource of the victim computer under ARP storm. In any case, the memory usage is so insignificant that it is not really a problem in these ARP attacks.

Another parameter of interest is the Page file size. Page File size is the current number of bytes that the active processes have used in the paging file(s). We measure the page file size during the attack to observe for activities in the virtual memory.

Figure 4 shows that there is no change in the page-file size before and during the direct ARP attack. Page-file size measurement at 0% load mainly provides the size due to the background processes running in the computer in the absence of any ARP request traffic. Furthermore, as the load of incoming direct ARP traffic is increased, there is really no impact on the virtual memory of the computer.

5.4. Resource Exhaustion of a (Non-Victim) Computer Receiving Indirect Frames

If i^{th} computer in the broadcast domain has an IP address of P_i then the indirect ARP-request frames arriving to the computer can be described as the frames with

$$\text{IP addresses} = \{\chi \mid \chi \neq P_i\}$$

Figure 5 shows minimum, maximum and average value for the processor exhaustion for a given load of the indirect ARP-attack traffic. It can be seen that a bandwidth consumption of 40% by indirect ARP-attack traffic in a fast Ethernet environment exhausts a Pentium-IV based non-victim computer to up to 55% of its 3.06 GHz processing capacity. Indirect ARP requests are still being processed by the computers on the network even though they are not directed towards them. Due to the processing of a barrage of indirect ARP-request messages, the CPU resource is still getting significantly consumed, however the processor exhaustion rate is relatively less intense compared to the one under direct ARP attack traffic. This is understandable as there is relatively more processing involved in direct ARP attack traffic compared to that of indirect ARP attack traffic.

Figure 6 shows the memory-usage of a non-victim computer, which is located on the same LAN segment as that of the victim computer, as the network bandwidth is increasingly consumed by the indirect ARP attack traffic. The memory consumed due to such indirect-ARP attack traffic is observed to be within 3 Mbytes, which is comparatively less than that consumed by the direct ARP attack traffic in **Figure 3.** Consumption of physical memory in the range of 3 Mbytes is not much of an issue for a 3.06 Hz computer with 512 Mbytes of RAM.

In this experiment, we also measure the page-file size before the onset of indirect ARP attack, and during the indirect ARP attack (**Figure 7**). The page-file size at 0% ARP traffic indicates the page-file size before the onset

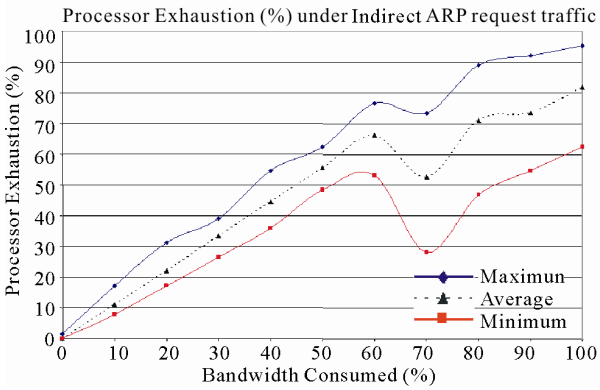


Figure 5. Processor exhaustion under the indirect ARP-attack traffic with IP address = $\{\chi \mid \chi \neq P_i\}$.

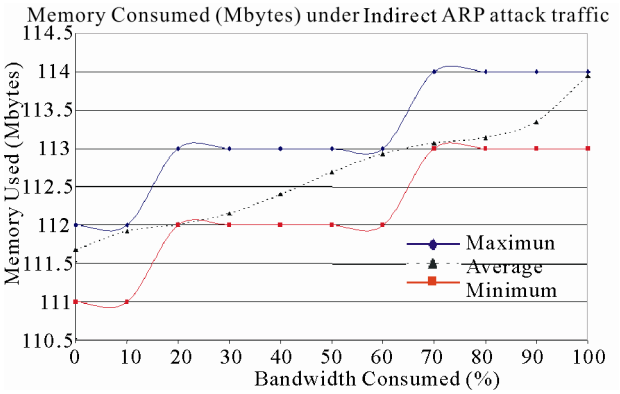


Figure 6. Occupancy of the computer's memory under indirect ARP-attack traffic with IP address = $\{\chi \mid \chi \neq P_i\}$.

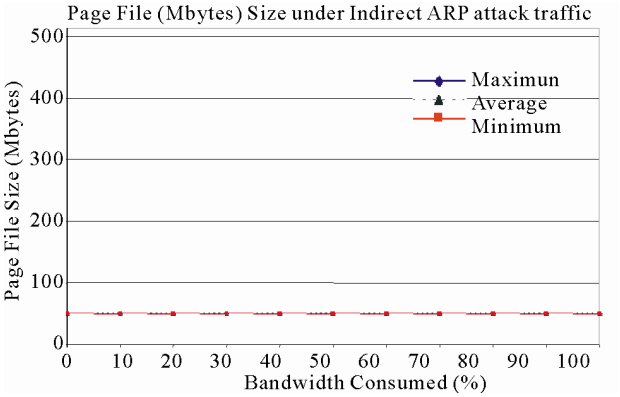


Figure 7. Page-file size is visibly unaffected under the indirect ARP-attack traffic with IP address = $\{\chi \mid \chi \neq P_i\}$.

of the ARP-attack, which is mainly due to the processes running in the background. The page-file size of the victim computer is measured as the network bandwidth is increasingly consumed by the indirect ARP-attack traffic. **Figure 7** shows that the page-file size is not affected by the indirect ARP attack traffic, as it stays the same before and after the onset of the indirect ARP attack. This is obviously due to the fact that the memory-consumed by the indirect ARP traffic (**Figure 6**) is quite minimal, and stays within the range of 3 Mbytes (out of a total 512 Mbytes) of RAM space, and hence no incoming ARP messages spill to the page-file.

6. Detection and Prevention

It can be seen from the prior experiments that the ARP storms can consume the computing resources rapidly for all the computers on the affected LAN segment. Hence it is important to detect the ARP storms immediately and raise alarm for its possible prevention before the entire LAN segment is brought down by such ARP storms. In order to detect these types of ARP attacks, it is important to monitor the ARP traffic on each LAN segments. Programs such as ARPwatch [10] can be used to monitor ARP traffic on each LAN segments and raise alarm when ARP storms or ARP poisoning tools are detected. One can also use SNMP to monitor changes in ARP table in routers and switches to raise alarm for onset of such ARP attacks.

One way to prevent ARP storm is to involve layer-2 switches in controlling the ARP broadcast floods at the source where the storm starts building up. This can be achieved by allowing for threshold limits for broadcast/multicast traffic on a per-port basis. Furthermore, these thresholds per-port basis should be set up by limiting the bandwidth consumed by ARP broadcasts on a switch port.

In order to support multiple layers of prevention, the routers can also be used in controlling ARP storm from spreading to others LAN segments. A network manager can configure the router (using its control policy) to impose a limit on the rate of ARP requests that can be allowed for the associated LAN segments. When the imposed threshold for the ARP requests is exceeded then the ARP request packets are dropped by the router. The router hardware should be fast enough to examine and drop the ARP request packets that exceed the imposed threshold, otherwise it is possible for the router to crash or experience slowdown of its operation and itself become a bottleneck resulting in eventual denial of service (DoS).

7. Conclusions

According to Computer Emergency Response Team (CERT/CC), there has been an increased targeting of Windows end-users' computer systems and servers for security attacks. Distributed Denial of Service (DDoS) attacks due to ARP-storms can be found in local area networks where many computer systems are infected by worms such as Code Red or by DDoS agents. In this paper, we present results of our experiments to measure the impact of ARP-storms on systems resource exhaustion of a Window-XP based computer system deploying a high performance Pentium-IV processor. It is observed that ARP-storms not only waste the communication bandwidth but also exhaust a processor's resource of a victim computer even more rapidly by forcing it to reply to a barrage of ARP-request messages. It is also observed that when the network bandwidth is consumed 40% by the ARP-attack traffic in a fast Ethernet environment, a computer system with a high-performance Pentium-IV processor of 3.06 GHz speed wastes up to 85% of its (victim computer) raw CPU-time in processing direct

ARP attack traffic and 55% of its (non-victim computers) raw CPU-time in processing indirect ARP attack traffic. This attack is found to be more processor intensive which means that it exhausts processor resource more rapidly than other computing resources such as memory. The memory exhaustion is found to be not significant when compared with the corresponding processor exhaustion. Memory usage is observed to be quite insignificant compared to the memory resource deployed in the system. The virtual memory or the page file of the victim computer is observed to be completely unaffected. Based on these experimental results, the ARP-attack can be categorized as the attack that causes computing resource starvation more rapidly than the bandwidth starvation, especially that of the processor of the victim and non-victim computer systems on the affected network. It is interesting to notice the collateral damage done by this attack on a given LAN, according to which it not only exhausts the resource of the victim computer but also exhausts computing resource of other non-victim computers present on a given LAN where the victim computer resides. The rate of resource exhaustion in this type of experiment can help network security engineers design efficient flow-control and threshold based attack prevention schemes at the switches and routers used in the LAN.

8. Acknowledgements

The authors would like to thank Uriel Ramirez and Sumanth Avirneni for equipment support, data collection and verification efforts in the Network Security Research Lab (NSRL) at UTPA. The work in this paper is supported in part by funding from US National Science Foundation under grant # 0521585.

9. References

[1] L. Gerber, "Denial of Service Attacks Rip the Internet," *IEEE Computer*, April 2000.

[2] P. G. Neumann, "Denial-of-Service Attacks," *ACM Communications*, Vol. 43. No. 4, April 2000, p. 136.

[3] K. J. Houle and G. M. Weaver, "Trends in Denial of Service Attack Technology," Computer Emergency Response Team (CERT)® Coordination Center, V1.0, October 2001.

[4] Computer Emergency Response Team (CERT)® Advisory, "Home Network Security," CA-2001-20. http://www.cert.org/tech_tips/home_networks.html

[5] A. Householder, A. Manion, L. Pesante and G. M. Weaver, "Managing the Threat of Denial-of-Service Attacks," CERT Coordination Center, October 2001.

[6] CERT® Incident Note IN-2001-10, "Code-Red Worm Crashes IIS 4.0 Servers with URL Redirection Enabled," CERT Coordination Center, August 2001. http://www.cert.org/incident_notes/IN-2001-10.html

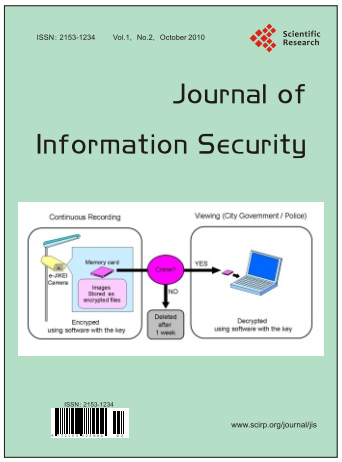
[7] Cisco Security Advisory, "Code-Red Worm—Customer Impact," Cisco Networks, July 2001. http://www.cisco.com/warp/public/707/cisco-code-red-worm-pub.shtml

[8] D. C. Plummer, "Ethernet Address Resolution Protocol," IETF Network Working Group, RFC-826, November 1982.

[9] S. Kumar, "Impact of a Distributed Denial of Service (DDoS) Attack Due to ARP Storm," *International Conference on Networking*, to be published in *Lecture Notes in Computer Science (LNCS)*, April 2005.

[10] ARPwatch. http://en.wikipedia.org/wiki/Arpwatch

Call for Papers



Journal of Information Security

ISSN 2153-1234 (Print) ISSN 2153-1242 (Online)
http://www.scirp.org/journal/jis

JIS, a quarterly journal, publishes research and review articles in all important aspects of information security. Both experimental and theoretical papers are acceptable provided they report important findings, novel insights, or useful techniques in these areas.

Editor-in-Chief

Executive Editor in Chief

Editorial Board

Subject Coverage

JIS aims to provide a platform for scientists and academicians all over the world to promote, share, and discuss various new issues and developments in different areas of information security. All manuscripts submitted to JIS must be previously unpublished and may not be considered for publication elsewhere at any time during JIS's review period. Additionally, accepted ones will immediately appear online followed by printed in hard copy. The topics to be covered by Journal of Information Security include, but are not limited to:

Access Control and Anonymity	Grid Security
Anti-Virus and Anti-Worms	Information Hiding and Watermarking
Authentication and Authorization	Intellectual Property Protection
Biometric Security	Intrusion Detection
Data and System Integrity	Key Management and Key Recovery
Database Security	Language-Based Security
Distributed Systems Security	Network Security
Electronic Commerce Security	Operating System Security
Fraud Control	Security Models

We are also interested in short papers (letters) that clearly address a specific problem, and short survey or position papers that sketch the results or problems on a specific topic. Authors of selected short papers would be invited to write a regular paper on the same topic for future issues of the ***JIS***.

Notes for Intending Authors

Submitted papers should not have been previously published nor be currently under consideration for publication elsewhere. Paper submission will be handled electronically through the website. All papers are refereed through a peer review process. For more details about the submissions, please access the website.

Website and E-Mail

http://www.scirp.org/journal/jis **E-mail: jis@scirp.org**

Journal of Information Security, **2010, 1, 45-94**
Published Online October 2010 in SciRes (http://www.SciRP.org/journal/jis/).

TABLE OF CONTENTS

Volume 1 Number 2 **October 2010**

