Introducing Orthus a Dual-Headed Authentication Protocol

Dean Rogers
Napier University, Edinburgh, UK
Email: dean.rogers.de@ieee.org

***Abstract*—This document describes a messaging architecture and internal message components of an authentication protocol which has been called 'Orthus'. For insecure closed LAN networks Kerberos is the most popular authentication protocol, currently in official release Version V [1]. Kerberos' objectives include protecting the privacy of message transfers necessary to achieve authentication, together with safe-guards against replay and man-in-the-middle, MitM, attacks. Orthus is intended to operate precisely this environment, here however, the Authentication Server, instead of delivering a ticket to the Client for use with the Ticket Granting Server, delivers that ticket directly to the TGS, and the TGS then delivers service granting tickets directly to the client, offering a simpler message flow, therefore providing fewer opportunities for message corruption or interception.**

***Index Terms*—authentication, authorisation, identity management, kerberos, orthus, single-sign-on**

I. Introduction

The two most common authentication protocols, Kerberos [2] and Radius [3], are well established; and between them cover most of the common requirements. Kerberos provides authentication services over closed LANs known as realms, involving a complex sequence of message transfers between all four entities concerned.

Other authentications protocols are encapsulated within more general networking protocols, such as Point-to-Point, PPP, which may incorporate Challenge Handshake Access Protocols, CHAP [4].

As seen from Fig. 1 we see the client A is involved in message-response pairs exchanged with two entities, the authentication service, AS, and the Ticket Granting Service, TGS, collectively known as the KDC (Key Distribution Center), before being assigned a ticket which will eventually grant access to the target server service, SS. The KDC does not negotiate directly with the SS; a legacy of the Needham-Schroeder protocol [5] upon which Kerberos is based.

The rational behind Kerberos being named after the ancient Greek mythological three headed hound is clear from Fig. 1. Namely, A, negotiates with three entities, or heads, to achieve authentication in the desired realm and access to its target service.

Manuscript received July 26, 2014; revised October 23, 2014.

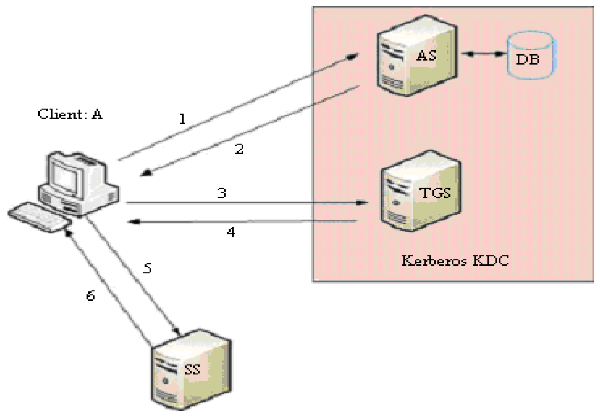

Figure 1. Kerberos message exchange

II. Problem Statement

Originally Kerberos was designed to run over closed insecure networks for the verification of a users' ID, and via inbuilt encryption systems to provide secure authentication services. Optimisation of the internal message structures of Kerberos when run on IPSec secured networks was covered in an earlier paper [6]. The essential message exchange workflow of Kerberos can however appear overly complex, involving six transactions to successfully access the desired Service. Could simplifying the structure from the clients' perspective maintain single sign-on, SSO, functionality, and security?

III. Literature Review: How Kerberos V5 Message Exchange Functions

Preliminary setup: a realm administrator sets a first time use password for a new User, who changes it via an integrated password changing protocol which produces a hash to be stored as a key in the KDC database.

Step one: from Client machine A, a User logs in with their established Username and password. Client side software hashes the password, and then sends a message to the Authentication Server, AS, including the Users ID, ID of the Ticket Granting Server, TGS, and a request for a timestamp, Fig. 2, thus protecting the Users password by not transmitting it. The use of colour coding below indicates the elements of message exchanges illustrating their relationship to each other during their passage through the message interchange process.

$$A \rightarrow AS\text{: } A+R_A+TGS+N_1+Tf$$

here: the '+' sign indicates concatenation
R_A this is the Realm of the client
N_1 is a nonce
Tf are client options, from (start) till (expiration or time-to-live) rtime (renew till time request)

By including the ID of TGS the AS recognises that the client requests authentication (there is only one TGS within the realm, and the AS would know this.) The client can determine the ID-TGS by various means outside this discussion, for example, from DNS information. A Timestamp establishes that the message is not a replay of an earlier one. Note that at this stage the SS-ID has not been indicated.

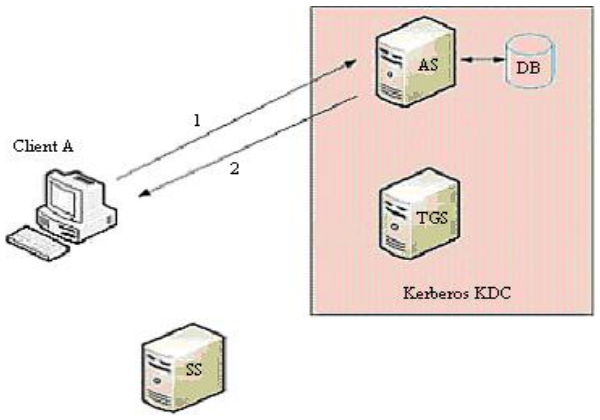

Figure 2. Client with authentication service

Step two: AS reply: after seeking the User ID in its database, it checks that the password hash stored matches that received. The user has authenticated to this stage of the procedure upon success; however, use of network resources is not yet permitted. To enable this, the AS returns to the client a message for further processing. This message contains two sets of encrypted data; a ticket for presentation to the TGS (Including additional flags not relevant here: and omitted for simplicity), the other is session information validating communication between the client and TGS.

$$AS \rightarrow A\text{: } A+R_A+K_{TGS}(K_{ATGS}+R_A+A+IP_A+Tf)+ K_A(K_{ATGS}+R_{TGS}+TGS+N_1+Tf)$$

here: K_{TGS} is the encryption key of TGS
R_{TGS} is the realm of TGS
K_{ATGS} is a session key for A and TGS use
IP_A is the IP address of A in any format consistent with the realm
K_A is the clients' encryption key as previously discussed

The TGT is a remit from the AS authorising the TGS to issue a ticket to A granting access to SS's. It includes duration stipulation, IP address, realm identifier, ID of client, along with a unique session key specific to A and TGS communications; all encrypted with the TGS key (in practice a hash of a value provided by the realm administrator). Facilitating SSO, the TGT is reusable by specifying different SS-IDs. Session information includes the session key (large random number generated by the relevant host), duration stipulation, realm and ID of the TGS (A's matching realm), and a nonce (if repeated in a later ticket it would be discarded); all encrypted with the hash of the clients password.

Step three: Using its own key the client can decrypt and retrieve the contents of the session information, and the data for connecting to the TGS. The TGT itself is stored for forwarding, Fig. 3.

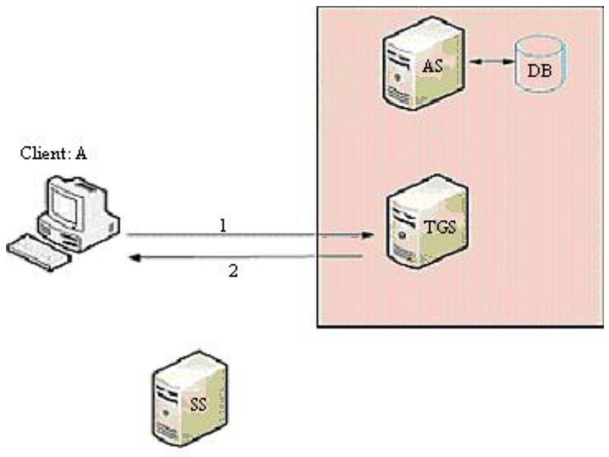

Figure 3. Client with ticket granting service

$$A \rightarrow TGS\text{: } SS+Tf+N_2+K_{TGS}(K_{ATGS}+R_A+A+IP_A+Tf)+ K_{ATGS}(A+R_A+T_{S1})$$

here: N2 is a second nonce in the protocol exchange
T_{S1} is a timestamp

The client is requests access of the TGS to an SS. It is of no concern at this point how A, in extensive networks, determines the ID of the desired SS from possibly hundreds; selected from a list, pre-configured in a profile, or via DNS. It sends a message consisting of the session duration, SS-ID, a new nonce, the ticket granting ticket previously saved, and an 'authenticator,' certifying the clients identity (effectively this means the client is certifying its own ID, but only they could). The authenticator consists of client-ID, and clients' realm, together with a time-stamp; all encrypted with the client-TGS session key recovered from the session information that the AS had encrypted with the clients key, retrieved from its DB.

Step four: (TGS reply) the TGS issues a ticket, for the client to present to the SS verifying to the SS's satisfaction that the client has been authenticated at realm level, Fig. 4.

$$TGS \rightarrow A\text{: } A+R_A+K_{SS}(K_{ASS}+R_A+A+IP_A+Tf)+ K_{ATGS}(K_{ASS}+Tf+N_2+R_{SS}+SS)$$

here: K_{SS} is the SS secret key
K_{ASS} is the session key generated by the TGS for the sole use of A and SS

The pink block represents the Service Granting Ticket SGT

The message returned by the TGS to A is similar in structure to that returned earlier by the AS, it consists of

ID and realm of the client, along with a re-usable ticket SGT facilitating SSO procedures to verify authentication of the client, encrypted with the SS secret key which the TGS holds; along with an encryption of the session information relevant to the client and SS communication using the client-TGS session key previously issued by the AS. This session information includes a client-SS session key, K_{ASS}, the realm, ID, and IP address of the client, and duration stipulations.

Step five: A stores the authentication ticket for presentation to the SS, and creates a new authenticator which it encrypts with the client-SS session key previously retrieved. It contains the clients' realm and ID, a new time-stamp, an optional Sequence Number SN to detect replays; and an optional session sub-key for the subsequent client SS communications.

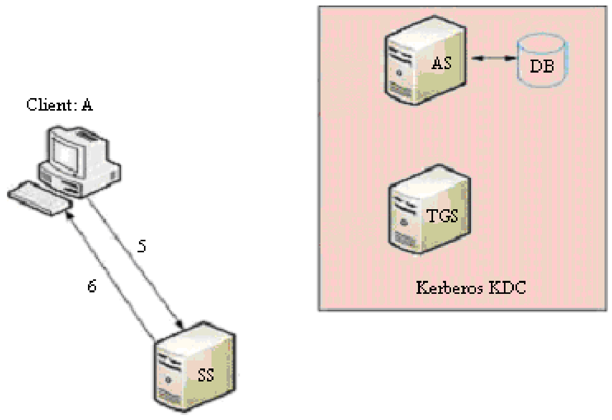

Figure 4. Client with server service

$$A \rightarrow SS: K_{SS}(K_{ASS}+R_A+A+IP_A+Tf)+K_{ATGS}(A+R_A+T_{S1})+O$$

here: O refers to optional fields that A can request

An optional field indicates that the client can request that the SS verify its identity, proving that it really is the SS the client intended to communicate with, thus insisting on mutual authentication.

Note that although A presents an authenticated SGT to the SS in question, the TGS does not perform a match of ID_A to ID_{SS}. Meaning that the SS can later return a message to A indicating that despite authentication, A is not authorised to use that SS. The usual means of achieving this is by means of Access Control Lists, which does not concern us here [7].

Step six: the final authentication message returned by the SS to the client is of the form,

$$SS \rightarrow A: K_{ASS}(T_{S2}+S_K+SQ_{N0})$$

here: T_{S2} is a new timestamp
SQ_{NO} is a sequence number
S_K is a flag requesting a new sub-key

The SS decrypts the SGT using its key, then maps the ID and IP of the User's workstation and the ID of the SS. Confirmation of SS ID constitutes, where applicable, the sub-key (if S_K is absent $K_{ASS,}$ the previous client SS session key is used) and sequence numbers, a new timestamp T_{S2} (an attacker cannot re-construct this massage without prior knowledge of the session key $K_{ASS,}$ and so T_{S2} can safely be returned without modification), all encrypted with client-SS session key. The inclusion of successively incremented sequence numbers, upon iteration of message exchanges, prevents replay attacks within the session. Only A and the correct SS have the session key, in this way, the SS has authenticated itself to the client.

A one time use only nonce is generated. For security reasons, to prevent guessing its value, it should be created by a random number generator. Further, an SQn is incremented upon each iteration; the original could be a random number.

It may be noted that LDAP has been misconceived in some quarters as an authentication protocol [8], however the background integral 'Bind' operation is usually reliant on Kerberos (or some other service), and so cannot in itself be considered a full authentication protocol. Further, the message transfers involve TLS encryption services to protect them.

Kerberos is undergoing further development as new applications are sought to fulfill the rising needs of new technological innovation [9].

IV. Kerberos Overview

To recap, assessing how the message interchanges relate, taking Diagram 1 above,

1. The first message, simply requests realm authentication.

2. The message returned from the AS is a permit to apply to the TGS for a ticket to access Services. This facilitates single-sign-on, as it is reusable.

3. Message three, to the TGS, specifies the ID of the SS which the client wishes to access.

4. The fourth message returns a ticket to A which, when presented to the SS in question will be accepted allowing access: note the encryption with the SS secret key, which the TGS holds.

5. The fifth message is the one that is finally sent to the SS requesting access.

6. A sixth and last message confirms to the client that they are conversing with the correct SS.

It follows from the above,

a. The client is effectively accepted into the realm upon transmission of message 2.

b. The SS required by A is first mentioned in message 3.

c. A database look-up of the SS ID is performed by the TGS before message 4 in order to retrieve its secret key before return to the client.

d. The SS accepts the SGT presented, allowing access. The TGS function is merely to retrieve the SS key – its database contains no indication whether the relevant SS will accept the Client. This is achieved by a process of 'Authorisation,' which the SS maintains. Simply, this could be on a scale of 0 to 9, with zero indicating no authorisation and so no access despite realm authentication.

e. If A requires access to another service, SS2, steps 3 to 6 above are repeated for each service.

V. Orthus

The Orthus authentication protocol involves similar concepts and terminology to those depicting Kerberos and for brevity a repetition will be omitted here.

Where the AS could return a reusable ticket to A in a universally applicable form, a UT, which additionally, for the purpose of gaining access to other SS's, facilitates SSO. What functionality may have been lost? Formerly the TGS had the role of recovering the specific SS secret key and providing it to the client, otherwise information relating to the existence of SS simply passed through it in encrypted form. We have now lost the ability to encrypt the SGT with this exact SS-key. Here, session information contained in message 4 is eliminated. For a well setup realm, evaluation indicates little disruption to functionality, assuming that the SS is preconfigured with the Realm Authority, and that the R_{SS} is redundant.

In a scenario whereby A transmits an initial request for ream Authentication to the AS, and subsequently the AS passes a 'success' Ticket directly to the TGS, informing it of the Clients ID. The TGS can now send a UT to A, enabling realm access, and A can then determine which SS it wishes to access. A retains the UT while sending a copy; together with the name of the SS it wishes to access, back to the TGS. The TGS returns a Ticket to A for access to a specific SS. Should A later wish to access another SS, it sends the same UT back to the TGS, but with the name of the new SS.

The benefit here was that Client A still exchanges a total of six messages before gaining access to SS1. Furthermore, under the Kerberos scheme, when calculated to include access to a second SS2, the total is ten messages, which is also so under this scheme. Thus, the only gain was a simplified UT compared to the previous, and a head reduction to two.

A radical solution may be obtained when the UT returned from the TGS contains sufficient information to use for approaching various SS's without the need to further retrieve appropriate SS keys from the TGS. Such a scheme, see Fig. 5, would reduce the number of transactions needed to access a given SS to four, and for a further SS2 to six. Consider, Kerberos would require six and ten respectively (when accessing several SS's Kerberos starts to appear more like a Hydra protocol).

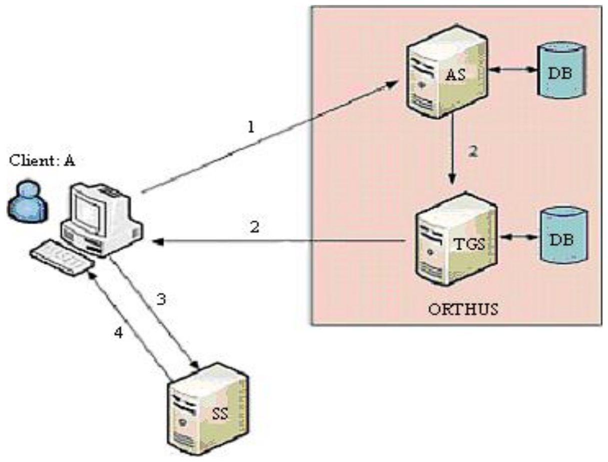

Figure 5. Orthus message exchange flow

As indicated in note e. above, the TGS holds no information as to the acceptability of the relevant client by its desired SS (just the SS key, ensuring that it only receives communication from authenticated clients) – can security be maintained in this scenario?

Here, client A needs to request the desired SS key, in what is now message 4, because the TGS cannot simply provide in bulk the keys of all SS's in its realm. Here, the client requests the SS Key directly from the SS itself, which is returned encrypted in message 5; which is now processed by the SS for mutual authentication and session key. The alternative of A knowing in advance the ID of which SS he needs at the time of login is not practical.

The Orthus protocol message structures are:

1. A $\rightarrow$ AS: A+R_A+TGS+N_1+Tf

The initial challenge 'here I am, and I'd like to join' generated by the client remains unchanged.

2. AS $\rightarrow$ TGS: A+R_A+K_{TGS}(K$_A$+R_A+A+IP_A+Tf)+ K_A(R_{TGS}+TGS+N_1+Tf) **[**K_{TGS} pre-shared]

The response in this case, however, is not back to the client. The AS notifies the TGS directly of a successful realm authentication request by the client, by sending a Ticket Granting Permit and client session information. Thus negating the need to transmit a client-TGS session-key across the network, albeit protected by A's encryption key. A reduction in the number messages transmitted should also enhance security.

3. TGS$\rightarrow$A: A+R_A+K_A(K_{ASS}+R_A+A+IP_A+R_{TGS}+TGS+N_1+Tf+K_{SSU})

The TGS notifies the client of successful realm authentication, and includes within the session info returned to the client the Kssu Universal SS key, all encrypted with the clients key. Possession of the Kssu key is now critical. Although potentially this seems to allow access to many SS's in the same realm its security is assured by encryption with K_A. Albeit placing increased reliance on the encryption algorithms protecting it, and the enforcement of strong password complexity polices.

As indicated above increased reliance on each SS's internal authorisation systems is now paramount, least a rogue client gain access to e.g. a database system where they have no business. This is not so different from the situation in Kerberos where the TGS has no prior knowledge whether the relevant SS will authorise the client once they have gained access to it. With Orthus this effect is multiplied, but the principle remains the same.

4. A$\rightarrow$SS: K_{SSU}(K_{ASS}+R_A+A+IP_A+Tf_1)+O

The client can now approach the SS, by using the Kssu, in a way that the SS understands as confirmation of the clients' realm membership, and indicates a session key for mutual exclusive use Kass. To prevent replay attacks based on this message it includes a new Tf_1 specification, which is now protected within the encrypted portion.

Note that the Kssu is never stored on the client side and as such presents no greater security risk than similar situations within the Kerberos environment, for example Kerberos message 2 from above. In fact in Kerberos the K_{ATGS} is retained in temporary memory for re-use in massage 4.

5. SS → A: **K_{ASS}(T_{S2}+S_K+SQ_{N0})**

Already possessing the Kssu, the SS can decrypt the message and indicate acceptance of the request by using the suggested session key to encrypt security data in a message returned to the client. T_{S2} derives from Tf_1 for replay protection.

As with Kerberos timestamps and nonces are used where appropriate throughout the protocol to circumvent MitM and replay attacks.

The defining feature of Orthus is that the client negotiates only once with the KDC before obtaining access to its target SS. As with Kerberos, Orthus provides no indication as to how the client obtains SS-ID information, except that this is only possible by post-authentication mechanisms.

VI. Implementation

Orthus specifies an alternative and coincidentally incompatible infrastructure when compared with Kerberos. Therefore, a complete rollout of all Orthus components as indicated would be necessary to achieve a functioning system. Indeed, Orthus is intended to be instigated the same environments where Kerberos would usually operate. However, in situations where Kerberos compatibility is not a requirement a complete installation could be effective immediately, such as with embedded card-pass systems.

As with Kerberos it is assumed that initial authentication material are transmitted to the client via out of band means.

A possible implementation scenario would be in networks consisting of a single service where concerns regarding the universal nature of the service granting ticket are mute.

VII. Conclusions

A party wishing to access network services needs to authenticate itself to the authoritative body responsible for the realm in which those services reside. Further, this needs to be as seamless as possible for the client in question, and secure for both parties. Each message exchange increases the potential for error and interception. Orthus reduces the number of entities with which the client needs to negotiate, and thus the total number of exchanges required to access the desired service, while maintaining security. Thus it appears to solve to the problem stated earlier.

Indeed, a side-effect of Orthus is that transmitting messages between the AS and TGS services may in one respect improve security, where they reside on the same server by reducing network transmissions.

It can be observed in note **c** from section III, which the TGS performs no validity check before retrieving the SS key from its database and issuing to the client in question. Thus a client could serially obtain the keys of all SS's in the realm simply upon repeated request. In effect this constitutes little higher security than the Universal ticket employed in Orthus.

With Orthus, security depends on whether it is possible to determine the SS verification information encrypted by the K_{SSU} from the two messages where it is used. In message 3 above, as long as K_A remains secure it cannot be determined, and in message 4 the author is unaware of any successful method to reduce a strong encryption key from the encrypted message. However it is recommended that provision is made for the necessity of upgrading the relevant algorithm by implementing the code modular in modular form.

As mentioned in Orthus message exchange 3 above, the security of the realm services relies more heavily than it does under Kerberos on the individual authorisation mechanism present on each Server Service.

In anticipation of 'authentication as a service,' it could be that more elegant solutions exist.

Appendix A Related Research

Various alternative authentication protocols exist, and are in continual development. One such is known as Central Authentication Service [10], CAS, and was developed at Yale University by Shawn Bayern in 2004, the specification being published in 2005. This, however, is browser based and relies on use of the https protocol to secure message transfers. Another is RADIUS, which is undergoing continual development by an IETF (Internet engineering Task Force) working group [11].

Appendix B Suggestions for Further Research

Firstly, time and resources permitting a practical implementation such as laboratory test environments in order to monitor and assess performance, resilience, and reliability of the Orthus system proposed here is needed.

Secondly, as proposed Orthus requires not only client processing power, but also temporary storage capability for store-and-forward functionality. Where authentication can be dynamically embedded in a Token, a User could commence using a computer by simply waving a Token past a sensor we would achieve truly user-friendly authentication. Such a system might be acceptable in lower 'physical security' environments, or as part of a layered Identity Management [12] scheme may enhance security by embedding complex passwords, the end-user would no longer need to memorise, especially in environments with multiple passwords.

Thirdly, as noted under the Conclusion Section Orthus provides authentication at the realm level, while subjugating meaningful access to services to relevant authorisation services. Benefits may be obtained from the development of more tightly integrated realm wide authorisation and authentication systems further enhancing Orthus security.

Finally, because the AS/TGS combination effectively forms a boundary, this suggests further applications of the Orthus protocol, and may lead to adaptations for uses other than as indicated above. With the increased usage of cloud computing and the virtualisation technologies, new Authentication models should be investigated to meet the coming challenges.

ACKNOWLEDGMENT

The inspiration for this concept arose from course-work set during the MSc in computing at the Liverpool John Moores University, UK. The author wishes to thank the staff of the Computing Department for their patience and guidance during his attendance.

REFERENCES

[1] B. C. Neuman, T. Yu, S. Hartman, and K. Raeburn. (Jul. 2005). The kerberos network authentication service (V5). [Online]. Available: http://tools.ietf.org/html/rfc4120

[2] J. Steiner, C. Neuman, and J. Schiller, "Kerberos: An authentication service for open network systems," in *Proc. Usenix Conference*, Dallas, Texas, Feb. 1988, pp. 191-202.

[3] C. Rigney, A. Rubens, W. Simpson, and S. Willens, *RFC 2058 - Remote Authentication Dial in User Service (RADIUS)*, The Internet Society, 2000.

[4] G. Zorn. (Jan. 2000). Microsoft PPP V2. [Online]. Available: http://tools.ietf.org/html/rfc2759

[5] R. M. Needham and M. D. Schroeder, "Authentication in large networks of computers," *ACM*, vol. 21 no 12, Dec. 1978.

[6] D. Rogers, "Proposals for a revision of kerberos when run in conjunction with the IPSec protocol suite," *International Journal of Electrical Energy*, vol. 1, no. 4, pp. 228-233, Dec. 2013.

[7] R. S. Sandhu and P. Samarati, "Access control: Principles and practice," *IEEE Communication Magazine*, vol. 32, no. 9, pp. 40-48, Sep. 1994.

[8] J. Garman, *Kerberos: The Definitive Guide*, O'Reilly Media, 2003.

[9] S. Hartman, K. Raeburn, and L. Zhu, *RFC 6806 - Kerberos Principal Name Canonicalization and Cross-Realm Referrals*, Encryption for RADIUS**,** Internet Engineering Task Force (IETF), Nov. 2012.

[10] D. Mazurek, *et al.* (May 2005). CAS protocol. [Online]. Available: http://www.jasig.org/cas/protocol

[11] S. Winter, M. McCauley, S. Venaas, and K. Wierenga, *Transport Layer Transport Layer Security (TLS) Encryption for RADIUS***,** Internet Engineering Task Force (IETF), May 2012.

[12] I. M. Milenkovic, O. Latinovic, and D. Simic, "Using kerberos protocol for single sign-on in identity management systems," *Journal of Information Technology and Applications*, vol. 3, no. 1, pp. 27-33, Jun. 2013.

Dean Rogers MSc, was educated at Porth Grammar Technical School, South Wales, Kellogg College Oxford, and in 2013 attained an MSc in Computing from Liverpool John Moores University, UK.
He has worked as a DBA and as a network administrator for some household names, and is currently employed with IT consultancy firm CGI.