Towards a distributed authentication system in Coordinated Mobile Virtual Organizations

Genge Bela[1], Haller Piroska[2],

Abstract — **Authentication in Virtual Organizations is an acute problem, when the system does not want to use a centralized, open to failures authentication model. In this paper we outline the specifications of a Coordinated Mobile Virtual Organization, where the coordinators are responsible only for the "introduction" of the new mobile nodes, which can move freely (but continuously) from one member to another. The system allows the use of a random number of coordinators, each of them having a random number of directly connected neighbors. We also propose a distributed authentication protocol based on the "Wide-Mouth Frog" and "Neumann-Stubblebine" protocols.**

Keywords — Distributed Authentication, Protocol Formalization, Virtual Organizations.

I. INTRODUCTION

In today's Internet the possibilities of being the subject of an attack are growing each day. This is why companies restrict access to they're stations by creating Virtual Private Networks (VPN) or by using proxies and secured gateways.

The problem with VPN is that once a user was authenticated, he can then have access to the entire network. The other problem is that VPN's are created statically, the entire communication being routed through a single point of access.

This paper deals with a Mobile Virtual Organization type, where a node may move around, and may access any other node in the system, previously being authenticated by a third, trusted party. There have been a number of proposals for the administration and authentication in Virtual Organizations ([1], [2], [3], [4]) but the systems described consider a global access and authentication point, meaning that every node will have to consult a central authority when accepting a connection from a specific host and each client will need to have a password with each node he will access.

[1] Genge Bela is with the Faculty of Engineering, "Petru Maior" University of Targu Mures, Romania; bgenge@upm.ro
[2] dr. Haller Piroska is with the Faculty of Engineering, "Petru Maior" University of Targu Mures, Romania; phaller@upm.ro

These assumptions are perfectly reasonable if we consider systems that are static or protected by physical devices (routers, proxys) [4]. But today, the mobile world is emerging and VO's may accept each day new members who can offer new services.

Therefore, we propose an organization type having multiple points of access, where each node, modeled as an Agent, can become a possible authenticator. The protocol used for authenticating users is formalized and transformed into CSP specification [15] using the Casper compiler developed by Gavin Lowe [5].

The system model proposed in this paper is only a starting point for future development and simulations that concern distributed authentication protocols in mobile environments.

The paper is structured as follows. In Section 2 we introduce our system, describing Virtual Organizations in general, and then specifying the properties of the CMVO (Coordinated Mobile Virtual Organization). In section 3 we describe the protocol for the authentication of the parties. We end with a conclusion and a specification of future work.

II. COORDINATED MOBILE VIRTUAL ORGANIZATIONS

A Virtual Organization (VO) is a set of entities (nodes), that we call Agents, each of them having a set of resources that may be used by a specific client, or other Agent. We consider nodes as being Agents because they can be modeled as acting on they're own environment independently. Also, Agents are not restricted to one point, they can move from one node to another to satisfy they're goals. Examples of autonomous agent systems may be found at [1], [6], [7], [8].

The purpose of this paper is not to answer the question "why is a VO created?" or "how does it share resources?". These questions are covered in detail in [1]. Instead, the questions that may find answers here, look like "is he safe to join the VO?" , "am I talking to the right person?" or "are you qualified to become an authenticator?".

1. System architecture

The system is composed of three kinds of Agents: Coordinator (C), Authenticator (A) and Requester (R), as shown in figure 1. These are connected through network lines that may be not permanent, or may be wireless connections, and more important: they are not safe: messages may be loosed, spoofed, replicated or created using old discovered passwords.

When a client (Requester Agent-R) wants to join the VO, to have access to the resources provided by an

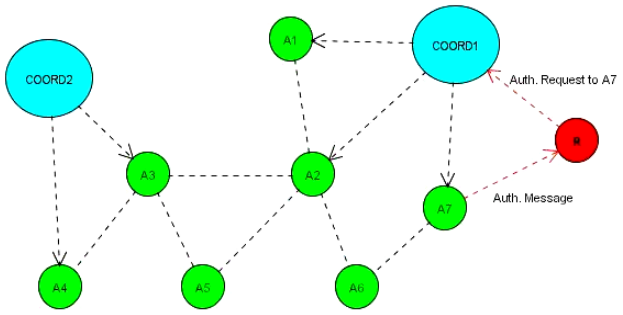

Figure 1. Agent R's entry in the VO, using COORD1 as access point

Authenticator, he presents himself to one of the Coordinators to which is registered. R cannot access the nodes from the system, without registration. The Coordinators must not be connected with all nodes in the VO. They have role only at the beginning of the authentication process, and can migrate to the adjacent nodes after that.

2. Registration

The process of registration can take any form, through a web page, e-mail, the important thing is that at the end of it, the user will have a certificate containing personal data (supported algorithms, password and random number generation capabilities, encryption, decryption speeds) signed by a Certification Authority (CA). In the proposed system we will make the simplifying assumption that there is only one kind of authority for each access point (Coordinator) that can do this.

In this version we assume that there is a stored password in a secure database that is used by the coordinator, but in the future we will include some sort of key exchange, to allow the entry of "unknown" users in the VO.

3. The Agents

The **Coordinator** Agent's role is mainly for primary authentication. We say mainly, because it will also have other purposes in the future, like the centralization of node behavior and key lifetime assertion. The system allows the access of two kinds of Agents:

- Authenticator
- Requester

If a new node wants to become an Authenticator ($A_i, i = N_A, N_A \geq 0$), he must first authenticate himself at one of the Coordinators ($C_j, 1 \leq j \leq N_C$), using his secret password $K_{A_i C_j}$ and presenting the certificate given by the CA. After the verification of the password and the certificate, C_j will engage in a provocation conversation with A_i, testing the "knowledge" of the new Agent, so A_i can prove that it is capable of authenticating other users.

The conversation between the two parts consists of a sequence of question/response (X/Y) type messages as those described in the process of argumentation in Letia [8], the difference being that this conversation is based on challenging the opponent, existing only one proponent, the Coordinator:

- $Q'_i, i \geq 0$
- $R'_j, j = i+1$
- $Q'_i \in X$
- $R'_j \in Y$

The proposed verifications include:

- Random number generation (rand)
- Supported algorithms (alg)
- Proposed message encryption speed (enc)
- Password generation (pwd)

Having these analyzed, a normalized quality function of the agent is constructed:

$$Qag(rand,alg,enc,pwd) = (f(rand) + f(alg) + f(enc) + f(pwd))/4 \quad (1)$$

where $f : R \rightarrow [0,1]$. The result is compared with the one in the database. The accepted tolerance function allowed, that is, the allowable difference between the value stored in the database at registration and the computed value is the following:

$$\eta = |Qag - D_Q| \leq \alpha \quad (2)$$

where D_Q is the stored (registered) value of the quality of the Agent, and α is the allowed tolerance level.

After the authentication process has been completed, the A_i agent is allowed to connect to the next Authenticator node, requesting COORD to initiate the authentication algorithm described in the next section.

The **Authenticator** Agent is responsible for the future introduction of "new" nodes, which want to migrate to other Authenticators. Every new node sends initially to an Authenticator his requirements. These include a set of values $\{L_i, I_i, K_i\}$, where L_i is the lifetime of the new session key, I_i is the requirement of the new agent, to authenticate other agents, and K_i is the capability of the agent (password generation, algorithms, …).

The last agent is the **Requestor** agent, which may correspond to any entity that wants to request an authentication from the organization. If a newly authenticated entity has the possibility to authenticate other nodes, he will be an Authenticator. Else, he will remain in the state of Requestor.

This chain of authentication allows the system to be scalable and not to depend on the functionality of the coordinators.

III. AUTHENTICATION PROTOCOL

In the described system, there was a need for an authentication protocol that satisfied the following:

a) Support protocol runs in insecure environments
b) Support message loss
c) Support creation of session key by a capable third party
d) Minimize the contribution of random keys from the new entity
e) Use only symmetric algorithms
f) Minimize DoS and Replay attack possibility

To satisfy these needs, a number of existing protocols have been studied: "Wide-Mouth Frog" [9], Yahalom [9], Needham-Schroeder [10], Otway-Rees [11], Neumann-Stubblebine [12] and Kerberos version 5, as evaluated in [13] and all of them shortly presented in [14]. From these, only Kerberos satisfies almost all needs, the rest having the big problem that they are all open to DoS attacks because any user can initiate the authentication mechanism.

Kerberos could not be used in our system, because of the following. Firstly, it is too complex, relying on two authentication servers and timer synchronization, or in our distributed system, the only servers that need to communicate are adjacent and they may not have they're clocks synchronized. Secondly, because the key generation in Kerberos happens on every query for a TGT message, and if we consider a system where messages may be loosed, a new key will be generated even if the client did not get the last one. Also, a third party authentication protocol is more preferred because the "Man in the middle" attack may be harder to create if messages do not travel on the same line.

A. The Casper formalization language

This section briefly introduces the Casper security protocol specification language. For more information, the reader should consult [5].

The Casper project, developed by *Gavin Lowe* is composed of a specification language and a compiler. The Casper protocol specification language is simple and clear, making it possible to describe a protocol in a few minutes. The goal of the project was to offer a simple language, similar to the "usual" way of specifying protocols that would allow (using the compiler) transforming a protocol specification into a more complex, CSP description.

Next, we will briefly describe the syntax of the Casper language.

A Casper specification of a protocol is structured in sections. Because of space considerations we can not offer a full-description of each section, therefore we will focus our attention upon the following sections:

#Free variables
#Protocol description
#Actual variables
#System
#Intruder Information

In the '*#Free variables*' section, the user may specify the types of participants to the protocol (Agents, Servers), Key types (SessionKeys of ServerKeys), timestamp variables and so on. The naming of the variables chosen in this section is used in the '*#Protocol description*'.

The '*#Actual variables*' section contains the actual participants that take part to the run of the protocol. Using these variables, the System is specified in the '*#System*' section, stating the roles that each variable will take.

Finally, an intruder information is provided in the '*#Intruder Information*' section so that the protocol may be verified against the knowledge of the intruder.

The steps of a protocol are numbered according to the specification, larger messages may be broken into sub-steps by using letters:

2.a 2.b

Sending an encrypted message is possible using the following statement:

A -> B : { X, Y, Z }{ kab }

which means that A sends to B an encrypted message with the key 'kab' that is composed of 3 parts: X, Y and Z.

If a party does not need to understand a certain message, it must be specified with the special operator '%' meaning that the message is stored in a variable and sent to the destination principal later:

Store message into 'Va':

A -> B : { kab, Rs1, Ts }{ k } % Va

Send it to other principal:

B -> C : Va % { kab, Rs1, Ts }{ k }

B. The proposed protocol

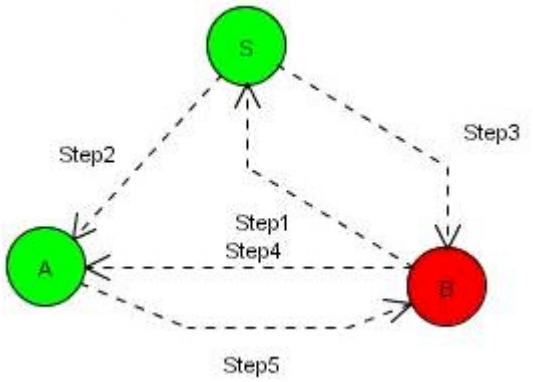

Figure 2. The algorithm steps for authenticating the new agent B

The '*#Free variables*' section:

A, B	*: Agent*
S	*: Server*
SKey	*: Agent -> ServerKey*
kab	*: SessionKey*
Ts, Tb	*: TimeStamp*
L, Rb, Rs, Rs1	*: Nonce*
InverseKeys	*: (SKey, SKey)*

The '*#Protocol description*' section:

0. -> B : A
1. B -> S : {A, Tb, Rb }{ SKey(B) }
2a. S -> A : { B, Rs, kab, L, Ts }{ SKey(A) }
2b. S -> A : { kab, Rs1, Ts }{ SKey(B) } % Va
3a. S -> B : { A, Rs1, kab, L, Ts }{ SKey(B) }
3b. S -> B : { Rb - 1 }{ kab }
3c. S -> B : { kab, Rs, Ts }{ SKey(A) } % Vb
4. B -> A : { B, Vb % { kab, Rs, Ts }{ SKey(A) } }{ kab }
5. A -> B : { A, Va % { kab, Rs1, Ts }{ SKey(B) } }{ kab }

The '*#Actual variables*' section:

Alice, Bob, Mallory	: Agent
Sam	: Server
Kab	: SessionKey
TS, TB	: TimeStamp
Life, RB, RS, RS1	: Nonce

The '*#System*' section:

INITIATOR(Bob, Sam, TB, RB)
RESPONDER(Alice)
SERVER(Sam, TS, Life, RS, RS1, Kab)

The '*#Intruder Information*' section:

Intruder = Mallory
IntruderKnowledge = {Alice, Bob, Mallory, Sam, SKey(Mallory)}

C. Protocol analysis

The protocol is straightforward, being initiated by agent Bob (the Requestor), who wants to authenticate himself to Alice (node A). S plays the third-party server.

The protocol assumes the following:

i. A and S share a secret key SKey(A)
ii. B and S share a secret key SKey(B)

Although these keys are in fact session keys (established at the beginning, when the user authenticates himself to one of the Coordinators) we consider them server keys because of the roles they play.

To protect against replay attacks, the protocol makes use of timestamps. The clocks of communicating neighbors do not have to be synchronized because the timestamp is used only as a Nonce [16], [17], [18], [19] ("Number once used"). This way, the receiving entities will not have to store a list of random nonces and verify them against incoming messages, but check only the timestamp of the latest package.

The protocol is started by B who wants to authenticate himself to A (step 0).

0. -> B : A

In step 1,B informes S that it wants to be authenticated to A:

1. B -> S : {A, Tb, Rb }{ SKey(B) }

B sends to S this message, composed of the name of A, a Timestamp Tb and a random number Rb, all encrypted with the key he shares with the server. The timestamp is sent to ensure S that this is a fresh message. The Rb is used to hide the contents of the message so that the protocol is well protected against offline-dictionary attacks.

Receiving this message, the server checks the timestamp and generates two messages, one of which is sent to A and the other one is sent to B. This way, A may present to B a proof that he is "known" by S and B may present to A a proof that he is also "known" by S. The word "known" is used to state that S has authenticated the parties.

The message sent back to A is decomposed in two parts for clarity. The first part:

2a. S -> A : { B, Rs, kab, L, Ts }{ SKey(A) }

informs A about the session key 'kab' that the server has generated. It also specifies a random number Rs that will be used by A to authenticate B. The package includes also a Lifetime for the key.

The second part:

2b. S -> A : { kab, Rs1, Ts }{ SKey(B) } % Va

is a message that A does not understand, beeing encrypted with B's server key. It is used only to prove that A knows the key and he got it from S. Also, in this package, the server ties the session key to the Rs1 random number so that the package may be authenticated with the random number sent back to B in step 3a.

The messages sent from B by S is composed of 3 parts. The first part:

3a. S -> B : { A, Rs1, kab, L, Ts }{ SKey(B) }

is a message similar to 2a, only addressed to B.

The second part:

3b. S -> B : { Rb - 1 }{ kab }

is used to ensure B that the server has produced the key and it is a response to the challenge sent by B in step 1.

The third part:

3c. S -> B : { kab, Rs, Ts }{ SKey(A) } % Vb

is similar to the message 2b.

After receiving these messages, the two parties now exchange the messages that will confirm them that the other side has received exactly the same password in the same run of the protocol. This is done in steps 4 and 5.

The great thing about this protocol is that it minimizes the use of B's capabilities in generating passwords and random numbers. The authentication process is grouped in sessions, the B agent having the capability of generating new authentication sessions if it wants to re-generate a password, or re-authenticate himself. If a message is loosed, B will not have to create a new authentication session, but he will only send a message for the same session, the keys being re-sent and not re-generated this way. The initial random number Rb is only used so that on response the client knows for which session was the authentication process started.

In the future, we will offer a formalization of the protocol using Typed-MSR where we will model and analyze the users and the possible intruders as specified in [16], [17], [18], and using the Typed SPI-Calculus [19] that will allow us the verification of the protocol.

IV. CONCLUSION

The described system allows the distributed authentication of users that are previously registered at a coordinator. The system does not differ very much from other authenticated virtual organizations when we are looking at the way entities join the system. The main difference is that users are not required to have a password with any of the entities in the VO and still be able to authenticate themselves properly using a third-party authentication protocol, as the one described in section III.

This system is recommended for use in mobile networks where agents (nodes) move around continuously, collecting information and then leaving the organization.

The protocol described in section III allows not only a third-party authentication, but was designed for the tolerance of message loss, and for use in environments that are not message-secure.

In this paper, we have presented a system that offers a decentralized authentication protocol where each node may become an authenticator. As future work we plan to construct a simulation model for the system that will allow us to detect and correct the possible faults. Also, we will have to investigate on the possibilities of evaluating the security "fingerprint" (password and random number generation capabilities) of a specified user so that the password will only become a back-up security element and not the primary means of authentication.

In these kinds of systems, nodes may misbehave, may malfunction as the result of hardware/software error, generating random data in a Byzantine manner. This is why, we propose also as a future work, the introduction of behavior lists for each Authenticator Agent so they may be excluded from the system in proper time.

REFERENCES

[1] Timothy J. Norman, Alun Preece, Stuart Chalmers, Nicholas R. Jennings, Michael Luck, Viet D. Dang, Thuc D. Nguyen, Vikas Deora, Jianhua Shao, W. Alex Gray, Nick J. Fiddian: "Agent-based formation of virtual organizations". KBS, 2004

[2] Roberto Alfieri, Roberto Cecchini, Vincenzo Ciaschini, Luca dell'Agnello, Ákos Frohner, Alberto Gianoli, Károly Lörentey, Fabio Spataro: "VOMS, an Authorization System for Virtual Organizations". European Across Grids Conference 2003: 33-40

[3] Michael Kaminsky, George Savvides, David Mazière, M. Frans Kaashoek: "Decentralized user authentication in a global file system". SOSP 2003: 60-73

[4] Mark L. Green, Steven M. Gallo, Russ Miller: "Grid-Enabled Virtual Organization Based Dynamic Firewall". GRID 2004, Pittsburg, PA, USA: 208-216

[5] Gavin Lowe: "Casper: A compiler for the Analysis of Security Protocols". In Proc. CSFW '97, Rockport. IEEE, 1997

[6] Ana L. C. Bazzan: "A distributed approach for coordination of traffic signal agents". AAMAS, 2005: 131-164

[7] N. Haque, N. R. Jennings, L. Moreau: "Resource allocation in communication networks using market-based agents". KBS, 2005

[8] Ioan Alfred Letia: "Gradually intrusive argumentative agents for diagnosis". Muti-Agent Systems for Medicine, Computational Biology and Bioinformatics, 2005

[9] M. Burrows, M. Abadi, R. Needham: "A Logic of Authentication". ACM Transactions on Computer Systems, Feb 1990, pp. 18-36

[10] R.M. Needham and M.D. Schroeder: "Using Encryption for Authentication in Large Networks of Computers". Communication for the ACM, Dec 1978, pp 993-999

[11] D. Otway and O. Rees: "Efficient and Timely Mutual Authentication". Operating Systems Review, 1987, pp. 8-10

[12] A. Kehne, J. Schonwalder, H. Langendorfer: "A Nonce-Based Protocol for Multiple Authentications". Operating Systems Review, Oct 1992, pp. 84-89

[13] B.C. Neuman and T. Ts'o: "Kerberos: An Authentication Service for Computer Networks". IEEE Communications Magazine, Sep 1994, pp 33-38

[14] Bruce Schneier: "Applied Cryptography". John Wiley & Sons, 1996

[15] C. A. R. Hoare: "Communicating Sequential Processes". Prentice Hall. April 1985.

[16] Balopoulos T. Gritzalis S. Katsikas S., "An Extension of Typed MSR for specifying Esoteric Protocols and their Dolev-Yao Intruder", in Proceedings of the CMS'2004 IFIP TC6/TC11 International Conference on Communications and Multimedia Security. D. Chadwick (Ed.), September 2004, Salford, UK, Kluwer Academic Publishers

[17] C.J.F. Cremers, S. Mauw & E.P. de Vink: "Formal Methods for Security Protocols: Three Examples of the Black-Box Approach". NVTI Newsletter 7, 2003.

[18] Iliano Cervesato: "Typed Multiset Rewriting Specifications of Security Protocols". Electr. Notes Theor. Comput. Sci. 40, 2000

[19] A. D. Gordon, A. S. A. Jeffrey: "Authenticity by Typing for Security Protocols", In J. Computer Security. 11 (4). 2003. pp. 451-521