A P2P Single Sign-On which remains Single on Smart Spaces [⋆]

Enrique Soriano
Francisco J. Ballesteros
Gorka Guardiola

Laboratorio de Sistemas
DITTE, Universidad Rey Juan Carlos de Madrid
{esoriano,nemo,paurea}@lsub.org

Abstract. Most single sign-on (SSO) schemes do not handle well scenarios where users work simultaneously with more than one device because their agents do not cooperate. In these scenarios, most SSO systems are really *per-machine* single sign-on. Besides, most authentication services usually depend on centralized services, may be hard to administer, and do not work well for highly dynamic and partitionable systems. In this work we present the Plan B's Peer-to-Peer security agent, SHAD. This agent offers SSO even when handling multiple machines and takes advantage of a context infrastructure. In addition, SHAD does not depend on any centralized service. We are using SHAD daily in our smart space and offices for accessing services without authentication obstruction.

1 Introduction

Most Single Sign-On schemes work well when the principal is using a single workstation and accessing classic client/server services through network connections. The problem arises when the user needs to use several devices simultaneously.

Nowadays, principals own several devices to perform different tasks on each one. As a consequence, users have to log on and authenticate for each device. These SSO systems do not work well in this case, because the user has to type at least one password per machine. Therefore they become multi-sign-on when used in a pervasive environment, and **there is not a single sign-on** in practice.

Several authentication devices have been proposed in order to reduce the level of obtrusiveness even when using multiple machines, see for example biometric sensors, Smartcards, Ibuttons, and USB Tokens. They alleviate the problem, but are still obtrusive. Most of them depend on specific hardware (generally readers) that may not be available in some situations, specially on mobile devices. A user would rather not be inserting cards in readers or pressing his fingerprint onto biometric devices all the time.

[⋆] Research work supported by the Spanish Ministry of Science and Technology project TIN2004-07474-C02-02.

Systems like Kerberos[20] and Sesame[12] are based on centralized servers that authenticate users and manage the access control policies. In these systems, the user depends on a central server to be authenticated. A principal can only be authenticated if he is on-line with the authentication server. What would then happen when a user needs access to a service and both the user and the service are disconnected from the centralized authentication service?

We argue that explicit authentication is an obstacle to make the machines vanish and allow the environment to become pervasive, and propose SHAD to overcome this obstacle.

The main contribution of this paper is an architecture that avoids explicit authentication and dependency of centralized authentication servers to access services inside and outside of the smart space. The scheme is based on the ideas described in [18], and has been integrated in the Plan B operating system [9, 8].

2 Introducing the SHAD SSO approach

SHAD offers **Single Sign-On for an ubiquitous environment** we are using. The user has to authenticate himself only once in order to access any service from any of his machines. Otherwise, users would not keep the illusion that *the ubiquitous environment is a unique system and not a set of interconnected systems*. A SHAD main agent serves the secrets (keys, passwords, etc.) to other machines that belong to the user. The main agent is designed for running in a mobile device that can be always carried by the used. Nevertheless, any machine owned by the user may run the SHAD main agent.

Different users can set different policies in order to distribute their secrets. SHAD does not try to emulate human trust relationships or to define any security policy. It only offers mechanisms to permit users to set their own policies. These policies are based on attributes that are assigned to each secret. Therefore, the security level of the secrets is up to the user. This will be thoroughly described in section 5. Using a mobile device in order to authenticate the user raises some issues that will be discussed in section 7.

3 Elaborating the Problem

3.1 The problem we had

In our smart space, we work at least with four machines concurrently: three PCs equipped with large displays running Plan B and its graphical environment, Omero [7], and a laptop PC, commonly running another operating system. Figure 1 depicts the environment. We use all these machines as an unique system: they share the keyboard, the mouse, the file system, the graphical interface, the voice device, and so on. But we need a SSO system for all machines to work with them as a single system, because programs must be authenticated constantly in

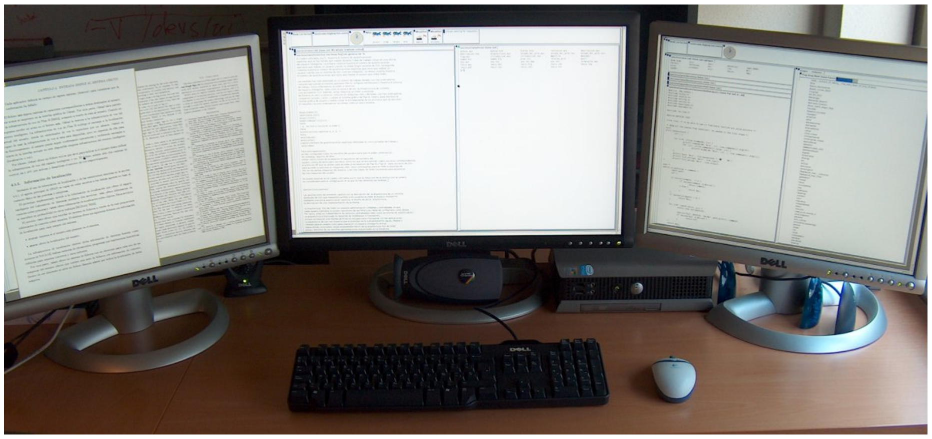

Fig. 1. A typical Plan B desktop.

order to mount the file systems, import services for location and context, etc. **Using this system without SSO would be painful**.

When we arrive to the office in the morning, we have to boot four machines. Using other SSO systems, we would have to authenticate ourselves for at least four times in order to set up the office environment. That is because we would be using a *per-machine* SSO scheme to sign onto a pervasive environment. **The point is clear: there is not a real single sign-on system!**

3.2 How SHAD fixes this problem

The SHAD SSO scheme is simple. Each machine runs a SHAD agent that offers authentication to the applications in two ways: (i) executing the authentication logic, or (ii) providing the secret to the application, which executes its own authentication logic.

We run a main SHAD agent in one machine, commonly a Pocket PC we always carry along[1]. The main agent obtains all the secrets of the user from a secure store. The secure store is a strongly encrypted file obtained from a SD Card inserted in the device, or from a centralized service. Note that in the second case we have to be on-line with the centralized service when we start the SHAD main agent for retrieving the secrets store. Once the store is in main memory, we do not need to be on-line with this server anymore. The main agent decrypts the file using a passphrase inserted by the user (the unique explicit authentication!). Then, it reads all the secrets (passwords, keys, etc.). Later, the secrets are stored in the main agent's process memory (which is not accessible from other processes or machines). We can also add new secrets to the agent if needed.

[1] Note that the main agent can run at any machine, although it is intended to be run in a mobile device

All other machines run plain SHAD agents. These plain agents try to retrieve the secrets from the main agent when necessary using a symmetric key secure protocol[2].

We can define policies to set the availability of our secrets: maybe some secrets might not be sent to a plain agent without an explicit confirmation from the user, other secrets might not be sent if the user is at a particular location. The examples are too numerous.

Last but not least, secrets may[3] be shared between plain agents when the main agent is gone. In order to support disconnections of the main agent, plain agents can cooperate and exchange secrets when needed using a peer to peer protocol. Of course, plain agents are not allowed to share secrets the user dislikes to share.

There is a trade-off between security and comfort, and SHAD SSO offers mechanisms to permit the user to set up policies and adjust his own level of both properties.

4 Architecture

The **main agent** role is adopted by the agent providing secrets to other agents. A SHAD agent becomes a main agent if: (i) it cannot discover another main agent; (ii) it is able to access the secrets store of the user (locally or remotely); (iii) the user provides the passphrase to decrypt the secrets store.

Plain agents try to discover their main agent and to establish a session with it. To do that, the plain agent broadcasts a look-up message ciphered with a private key assigned to the machine in which it's running. In order to support our architecture, this secret key must be kept stored in each device. This secret key is created and stored together with the identification name of the owner of the device at configuration time. Usually, this is done when the device is first installed. This kind of secret is named **SHAD terminal's secret**. It is also kept within the secrets store. In this way, the main agent is able to authenticate the machines owned by the user.

To set up a session, the main agent provides three data items to the plain agent: (i) a new session key for this plain agent in order to request secrets; (ii) a long number to identify the main agent's incarnation; and (iii) an incarnation key that is shared among all plain agents that started their session within this incarnation of the main agent.

When an application running in a machine needs authentication, it asks the local SHAD agent. The local agent needs to know an specified secret to authenticate the application. For example, to authenticate a SSH connection, the agent must know the user's password for the SSH server. If it has the secret, the authentication is done just like in a per-machine SSO scheme. But, if the local agent does not know the secret needed, then it will ask to its main agent.

[2] Due to space limit, we are not able to describe the protocol in this paper.
[3] Note, not "must".

The main agent evaluates the restrictions assigned to this specific secret, and proceeds in accordance.

When the local agent is not able to contact the main agent, it tries to get the required secret from any other plain agent. It broadcasts a request using a secure protocol relying on the incarnation key and the incarnation id. If any other plain agent receives the broadcast, and it has the required key, and the key restrictions allow to share it, the agent responds to the broadcast with an unicast message to share its secret.

When a user needs to add a new device to the environment he does not have to contact any administrator. Instead, he must create a key for the new device and store it in his secrets store. Second, he must assign the key to the device, for example, by writing it in NVRAM within the device.

We assume that an attacker does not have physical access to the hardware. Physical access to the hardware implies a great risk, because the secret stored in NVRAM could be compromised and the attacker could impersonate this machine. Therefore the attacker could gain access to the owner's secrets shared with this machine.

But note that when an attacker has physical access to the hardware, securing it is pointless. This principle is also known as *The Big Stick Principle* [19, Chapter 4]. In this case, the adversary can manipulate it, steal the disks containing all the information, modify the software and the operating system to be malicious or literally burn and destroy the machine. Other systems are based on the same assumption, see for example the GAIA security architecture[15].

5 Restrictions and Countermeasures

The secrets stored in the secured repository have some attributes that are used to set access control policies and restrictions. We describe them here:

- `noremoteaccess` means that the secret cannot be sent outside the local agent in any case.
- `nopeeraccess` means that this secret can only be sent by the main agent to plain agents; but plain agents cannot share these secrets among them.
- `needconfirm` forces explicit confirmation in order to share the secret.The user can confirm operations just by pressing a button on the machine running the main agent.
- `userlocation=`*location* means that the secret can only be sent if the user is at the specified location. Note that if the `needconfirm` is set, the operation also needs to be confirmed by the user. This location information is provided by a external context infrastructure[4]. If the context infrastructure is not available, the secret cannot be sent.

[4] Securing the context infrastructure is out of the scope of this paper. We assume that context data is correct and reliable, but not fault tolerant. If a user does not trust this infrastructure, it suffices to avoid the *location attributes. The trade-off, as we said, is made by the user.

- `clientlocation=`*location* works in a similar way, but taking into account of the location of the machine running the plain SHAD agent, that is requesting the secret.
- `samelocation` requires the user and the client machine to be at the same physical location.
- `accesiblefrom=`*machine* makes the secret available only from a plain agent running on the specified machine.

All these attributes can be combined in order to restrict the access to the secrets. The user must evaluate the trade-off between comfort level and security level from his own perspective and set the attributes according to it.

When the main agent sends a secret (with or without user confirmation), it always prints a warning about it and all these messages are saved in a log file.

The user can set a timeout to block the main agent if the machine in which it runs is idle for a given period of time. When the timeout expires, the secrets stored in main memory are deleted and the agent shuts down. In order to activate the main agent, the user must introduce the passphrase again. Then, the secrets are retrieved from the secret store and decrypted. Therefore, if the user loses the machine running the main agent, there is a customizable window of vulnerability.

6 Implementation, Integration and Performance

SHAD OSS is a Factotum[10] derivative. Factotum is the Plan 9 [14] security agent, which provides *per-machine* SSO through a virtual file server interface. We have added all the mechanisms to permit the agents running in different machines to cooperate.

Due to its file system interface, applications using SHAD do not depend on any kind of middleware or framework. To add secrets, the user only has to write a textual tuple in a virtual file. The tuples contain all attributes for the secret, including the SHAD attributes described in section 5.

A SHAD graphical front-end has been built for Omero, the Plan B's pervasive graphical system[7]. This graphical front-end uses the Plan B's voice service to warn the user about some situations. SHAD also takes advantage of our context infrastructure, in which users are located through ultrasonic transceivers and X10 sensors [6].

The prototype uses the AES symmetric algorithm in CBC mode in order to encrypt the communication between agents. In addition, it uses SHA-1 in order to assure message's integrity.

As we stated before, the SHAD terminal's secret is stored in the NVRAM of the machine. We are looking for hardware alternatives to the NVRAM. A solution would be to use any tamper resistant hardware using challenge-response methods in order to authenticate machines.

The prototype is fast enough from the point of view of user's experience as shown in Table 1. It shows the mean time to connect to a SSH server from a client using Factotum and SHAD in two cases: (i) retrieving the key from the main agent, and (ii) retrieving the key from the other plain agents (P2P protocol). The experiment has been performed with four PCs connected by a fast-ethernet network.

	Factotum	SHAD	SHAD P2P
$Time(sec)$	0.12	0.14	4.06

Table 1. Average time for a SSH connection.

Note that the P2P case would be needed once per-machine and per-service because SHAD will remember the secret for the next time. The P2P mean time is highly under the influence of the main agent discovery's timeout.

Table 2 shows the overload caused by the use of a real context infrastructure in order to avoid confirmations. First column shows the average time to mount a Plan B volume when the required secret does not have any restriction. Second column shows the average time when using the location information provided by the context architecture to avoid the confirmation.

	no restrictions	`userlocation`
Time (sec)	0.15	0.17

Table 2. Average time for mounting a Plan B volume.

The overload caused by the use of the context infrastructure is barely perceptible.

We would like to show some measures about the number of explicit authentications that must be performed in the scenario illustrated in section 3.1. Table 3 shows the number of explicit authentications required during five working days.

	No SSO	Per-machine SSO	SHAD
Machine #1	129	10	10
Machine #2	73	10	0
Machine #3	70	16	0
Total	272	36	10

Table 3. Number of authentications required by the system in a working week.

Daily work consists of numerous tasks, such as programming, compiling, editing LaTeX, and so on. Note that authentication is needed to perform frequent tasks, for example to redirect the mouse and the keyboard. *Machine #1* represents the primary terminal in which both the keyboard and the mouse are attached. This terminal runs the main agent. The two other terminals are represented as *machine #2* and *machine #3*. First column shows the number of explicit authentications that should be performed when avoiding any SSO system. Second column shows the number when using a *per-machine* SSO system. Third column shows the number when using SHAD SSO.

We can observe that SHAD reduces considerably the obtrusiveness of the system, even when using only three terminals. Note that the number of required authentications measured when using SHAD is higher than usual, due to the tasks that were performed this week. These tasks included kernel debugging, that forced several system reboots. In the common case, the user has only to authenticate himself once per day.

7 Discussion and Related Work

The problem of using a personal server to authenticate users is that the security data depends on a physical object, and it might be stolen or lost. Nevertheless, in real life humans already depend on physical objects that might be stolen or lost. For example, credit cards and official documents like the driving license or the social security card. When humans lose these very important physical objects, they immediately call the bank office or the police. Likewise, if a user loses the device with his secrets, he should revoke them.

In general, security is a trade-off between safety and costs[17]. Users have to accept the trade-off between comfort and the risk of losing the physical object that contains their secrets. In other words, users must face the risk of losing the device that provides the non-obtrusive authentication. SHAD implements some countermeasures against the possible security risk derived from losing this device, such as confirmations and timeouts.

The system is highly configurable and flexible. Users preferring comfort to higher security can customize SHAD in order to serve all passwords and seldom ask for confirmation. On the other hand, users preferring extra security (instead of more comfort) can disable the sharing for the most important passwords and require confirmation for all requests. In real life, users prefer a combination of comfort and security and configure SHAD according to their own perspective.

In SHAD, a user might make a wrong choice, but note that most systems are vulnerable if the user makes wrong choices (e.g. password based systems are vulnerable if lazy users set a blank password in order not to type at log in).

There are password tools [16] that permit the user to copy his passwords and paste them in the applications. These tools make authentication more comfortable than password typing, but they are still obtrusive. SHAD is not, because it can provide secrets without human interaction.

Simple SSO agents permit single sign-on for a single service, for example the SSH agent. Web browsers remember passwords and provide SSO for web applications. Other SSO systems provide single sign-on for different services[1, 2, 10]. These systems do not work well in scenarios like the one presented in section 3. Moreover, most of them depend on centralized servers.

In CorSSO [11], when a client needs to be authenticated by the application server, it has to retrieve a subset of keys from a set of authentication servers. It solves the problem of centralization by using multiple authentication servers. However, it does not solve the problem of authentication of devices at isolated locations where they are disconnected from the rest of the system. In addition, CorSSO's scheme requires changes in both client and server applications. Most important, as in most other SSO systems, the user has to authenticate himself at least **once per machine**. SHAD gives real SSO.

Classic security systems like Kerberos[20] and Sesame[12] provide authentication, but they depend on centralized services and are hard to administer and are *per-machine* single sign-on.

The use of authentication devices, such as Smart Cards or iButtons, has been proposed in several previous works, but they are normally used to authenticate users for specific and *ad-hoc* purposes. They require the authentication hardware and are not general purpose authentication mechanisms that could be used for any service in the system. Many schemes based on authentication devices are obtrusive and offer *per-machine* SSO. For example, CryptoToken[13] is a USB device that holds the secrets of the user. When the user needs to work with a machine, he connects the CryptoToken to it. This approach is only a bit less obtrusive that typing passwords, but it is still a burden for the user. In addition, not every device has USB ports. Finally, it does not support concurrent authentications in different machines, because the user only has one CryptoToken. However, we do.

Other security schemes have been proposed for *ubiquitous environments*. Most of them depend on complex middleware architectures and are highly centralized. In general, middleware based architectures make developers depend on specific platforms or languages. Some works [3–5] related to GAIA are heavily based in centralized security schemes like Sesame and Kerberos and therefore have the same problems cited above. We do not.

8 Conclusion

The main contributions of this paper are (i) a way to provide **real Single Sign-On** for ubiquitous environments where users work with several machines at same time, and (ii) a short description of a prototype implementation of the architecture.

SHAD does not require complex administration, and it is independent of centralized services. Last but not least, it does not depend on any kind of middleware or framework, because it relies on distributed file system technology and can be used through a virtual file system interface. As far as we know, all these properties make SHAD distinct and unique from other systems presented in the literature.

References

1. *SecureLogin Single Sign-On White Paper*. Protocom Development Systems, 2003. http://www.protocom.com/html/whitepapers/.
2. *Focal Point Evaluator's Guide*, 2004. http://www.okiok.com.
3. J. Al-Muhtadi, M. Anand, N. D. Mickunas, and R. Campbell. Secure Smart Homes using Jini and Sesame. In *Proceedings of the 16th Annual Computer Security Applications Conference*, 2000.
4. J. Al-Muhtadi, A. Ranganathan, R. Campbell, and N. D. Mickunas. A Flexible, Privacy-Preserving Authentication Framework for Ubiquitous Computing Environments. In *Proceedings of IWSAEC 2002*, 2002.
5. J. Al-Muhtadi, A. Ranganathan, R. Campbell, and N. D. Mickunas. Cerberus: A Context-Aware Security Scheme for Smart Spaces. *IEEE International Conference on Pervasive Computing and Communications*, 2003.
6. F. J. Ballesteros, G. Guardiola, E. Soriano, and K. Leal. Traditional Systems can work well for Pervasive Applications. A Case Study: Plan 9 from Bell Labs becomes Ubiquitous. In *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications*, 2005.
7. F. J. Ballesteros, K. Leal, E. Soriano, and G. Guardiola. Omero: Ubiquitous User Interfaces in the Plan B Operating System. In *Proceedings of the IEEE International Conference on Pervasive Services 2006*, 2006.
8. F. J. Ballesteros, E. Soriano, G. Guardiola, and K. Leal. Plan B: A pervasive Computing Environment Without Middleware Designed for Programmers. *To appear.*
9. F. J. Ballesteros, E. Soriano, K. Leal, and G. Guardiola. Plan B: An Operating System for Ubiquitous Computing Environments. In *Proceedings of the IEEE International Conference on Pervasive Services 2006*, 2006.
10. R. Cox, E. Grosse, R. Pike, D. Presotto, and S. Quinlan. Security in Plan 9. In *In Proceedings of the 11th USENIX Security Symposium*, 2002.
11. W. Josephson, E. G. Sirer, and F. B. Schneider. Peer-to-peer Authentication with a Distributed Single Sign-on Service. In *Proceedings of the International Workshop on Peer-to-Peer Systems*, 2004.
12. P. Kaijser, T. Parker, and D. Pinkas. Sesame: The Solution to Security for Open Distributed Systems. *Computer Communications, vol. 17, pp. 501-518*, 1994.
13. H. Kopp, U. Lucke, and D. Tavangarian. Security Architecture for Service-Based Mobile Environments. In *Proceedings of the 2nd International Workshop on Middleware Support for Pervasive Computing, IEEE PerCom 2005*, 2005.
14. R. Pike, D. Presotto, K. Thompson, and H. Trickey. Plan 9 from Bell Labs. In *In Proceedings of the Summer 1990 UKUUG Conference*, 1990.
15. G. Sampemane, P. Naldurg, and R. Campbell. Access Control for Active Spaces. *In Annual Computer Security Applications Conference (ACSAC2002)*, 2002.
16. B. Schneier. Password Safe. http://www.counterpane.com/passsafe.html.
17. B. Schneier. *Beyond Fear: Thinking Sensibly about Security in an Uncertain World.* Copernicus Books, New York, NY, 2003.
18. E. Soriano. Shad: A Human Centered Security Architecture for Partitionable, Dynamic and Heterogeneous Distributed Systems. In *Proceedings of the 5th ACM/IFIP/USENIX International Middleware Workshops.* 2004.
19. F. Stajano. *Security for Ubiquitous Computing.* John Wiley and Sons, 2002.
20. J. G. Steiner, B. C. Neuman, and J. I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Proceedings of the Winter 1988 USENIX Conference*, 1988.