Community-Centric Vanilla-Rollback Access, or: How I Stopped Worrying and Learned to Love My Computer

Mike Burmester, Breno de Medeiros, and Alec Yasinsac

Department of Computer Science, Florida State University,
Tallahassee, FL 32306, USA
{burmester,breno,yasinsac}@cs.fsu.edu

Abstract. We propose a new framework for authentication mechanisms that seek to interact with users in a friendlier way. Human or community-centric authentication supports *vanilla access* to users who fail an initial attempt to identify themselves. This limited access enables them to communicate with their peer community to achieve authentication. The actions of users with vanilla access can be *rolled back* in case they do not progress to full authentication status.

This mechanism is supported by a peer community trust infrastructure that exploits the effectiveness that humans have in understanding their communal roles in order to mitigate their lesser skill in remembering passwords or pins. The techniques involved essentially implement a human-centric key escrow and recovery mechanism.

1 Introduction

The research and practice of user authentication techniques often contradicts human nature. Indeed, most tasks required for user authentication, such as remembering a password or a pin, are much more effectively done by machines. While machines have no difficulties in storing and perfectly reproducing any amount of data, we humans are used to committing to memory only a few private authentication values (such as an identification card number or social security number). The real security requirements of password-based authentication (frequently changeable, long and not easily guessable passwords) are not truly compatible with human behavior, if acceptable to perhaps a minority of us. Other mechanisms that underlie strong authentication protocols, such as secure tokens, often work less well in practice than might be expected because of the human tendency to think of security measures as hindrances to be tolerated as necessary evils, or because they require extra investment in hardware and system management which may not be available.

On the other hand, humans are very effective at understanding the roles they play in their community, at organizing access control to common resources and at protecting private property. This indicates that there is no intrinsic human inability to deal with the authentication problem, only human ineffectiveness at

B. Christianson et al. (Eds.): Security Protocols 2005, LNCS 4631, pp. 228–237, 2007.

dealing with machine-friendly protocols. The goal of this research is to introduce a human-centric – and since authorization happens in the context of the human role within society – a community-centric approach to access control.

The premise that humans are effective at comprehending their roles is a broad one and we believe could lead to new directions and ways of thinking about security. In this position paper, we illustrate how a broad principle of community-centric security can be applied to the design of securing a particular task, namely remote login. We then broaden our scope and speculate how this approach can be used to tackle other unwieldy areas where security technology and human-behavior intersect.

Previous efforts to tailor protocols for human capabilities include the human-computer authentication techniques by Hopper and Bloom [1,2]. Our approach here differs in that it considers not the exact user-login protocol used (in particular, Hopper-Bloom protocols could be employed for that end) but an architecture to provide for alternative means to faithfully authenticate users when they have forgotten their authenticating secrets.

Our mechanism is conceptually a human-centric escrow key recovery mechanism [3,4]. Humans who forget their password can still get "vanilla" access to network services. By interacting with other users, vanilla users can leverage the peer community trust infrastructure to achieve full access to services —provided they can convince others of their true identity. Otherwise, any transactions accomplished with vanilla-access can be rolled back.

2 A Toy Example

Consider the following use-case scenario: Alice moved to a new department in her company, one that adopts a community-centric access control system. As she starts to work using her new account, she notices few differences from her previous experience, so she does not endeavor to read the tutorial recommended by her system administrator. She next proceeds to import her "address book," and she finds that the address cards display differently in the new system. Indeed, each address card has options related to being added to a *trusted peer community*. She decides to add her office-mate Marla to such peer community, which she fairly unimaginatively names "office-mate community."

The following day the most disruptive event happens. Alice forgets her new password, what with the stress of the new position. Now Alice must contact the system administrator, get him to reset her password, and then remember a new password. In a last desperate attempt to save her pride, Alice tries to login a third time. Voilà, success, or maybe not? (This is vanilla access.)

All her files and folders seem to be nowhere in sight. Her mailbox (with her name on it, for sure) seems to be empty of e-mail; quite tragic considering all the requests her new boss had sent her yesterday. On the verge of tears, Alice sends a quick e-mail to her friend Marla, asking what should she do now? A minute later, Marla sends her a reply. As Alice opens the e-mail, the most amazing thing happens –suddenly all her files are back, her e-mail, everything. Did Marla set

her up on some practical joke, she wonders, but really Marla is not the type for that. As she ponders the mystery of it all, Marla steps in and says, "I am glad you trusted me in a peer community. That's how I was able to authenticate you by e-mail today." "You authenticated me by e-mail?" Alice asks. Marla responds, "That's just it. Our e-mail system uses some secret sharing technique —or maybe threshold trust, one of these buzzwords of techno-geek. What that means is that you made me a fully trusted reference by creating a peer community with only myself in it. By the way, I suggest that you add another person to the same community you added me. I don't want to be solely responsible for authenticating you, as that also means I am solely to blame if someone manages to fool me into authenticating herself as you —not very likely, but you never know. You can add our boss, he is a stickler for procedure and would never reply to an authentication request without triple checking it was indeed you beforehand." After Alice plays the tutorial, she understands the notion of a trusted peer community. Essentially all her files are encrypted under her password-derived master key. Nonetheless, when she adds persons to a trusted peer community, they automatically receive "shares" of her password-derived key by secure e-mail. If she forgets her password, she may send e-mails to all the members of one of the trusted communities and she may log in! Of course, they will have to reply and they are supposed to check to ensure that she is indeed Alice. Today this saved her from looking like a fool to the systems administrator. She was rescued by her company's community-centric vanilla-rollback access system.

3 Analyzing Community-Centric Vanilla-Rollback

A community-centric access control system recognizes that authentication is scalar rather than Boolean. For example, it assumes that when Alice cannot remember her password, she may not be Mallory, and as likely as not, just Alice on a bad day (Murphy-Alice), so it allows a limited login to occur anyway. In fact, it even allows Alice write[1] access to her file system and e-mail accounts. We call this *vanilla* access, because it provides a modest level of access. For instance, it allows vanilla-Alice to accomplish low-risk tasks, e.g. to send e-mails. [2] (The sending mail server modifies sender information to indicate that it may come from someone impersonating Alice.) In addition, if vanilla-Alice sends e-mail to Marla, a member of one of Alice's trusted-peer communities, Marla is notified that she may provide Alice with authentication credentials if she believes

[1] Write-only access can be quite dangerous. For instance, vanilla-Mallory might add invisible executable files to Alice's account and cause mayhem when the real Alice comes in. That's where the rollback feature plays an important part. The write access only creates "evidentiary" files that are not committed to the Alice's file system if the vanilla-logged user leaves the system without being promoted to full access status. For details, see section §5.

[2] Simple precautions would prevent spammers from using vanilla access to relay e-mail. One possible measure is to prevent sending of e-mail to a recipient outside the peer community.

vanilla-Alice is indeed the real Alice. If Marla decides to do so, a share of Alice's key is securely transmitted in the response, and in combination with other shares, it is sufficient to reconstruct Alice's key (which protects her file-encrypting key). Alice can then recover her files and other access privileges, and change her password. Conversely, if an intruder is impersonating Alice and she fails to complete the authentication, the system will roll back to its previous safe state. We call this process *Vanilla-Rollback-Access* (VRA).

The vignette in the previous section illustrates how VRA leverages the scalar properties of community-centric authentication for rollback access control. An essential notion that is a feature of human identity but that Boolean authentication overlooks, is that identity is naturally difficult or impossible to verify (or deny) conclusively. Rather, we gain confidence in someone's identity through evidence that may take many forms with widely variable strength. VRA recognizes this inconclusiveness and balances the potential damage against the level of identity confidence that the system has acquired. For example, while the password may be incorrect, the person at the keyboard at least knew their user name. Not a great source of identity confidence, but slightly greater than zero. Additional confidence may be gained if VRA can determine behavioral aspects of users, such as typing speed or other keyboarding characteristics. These offer additional confidence in the identity (or identity denial) of the user under evaluation. Beyond intrinsic identity characteristics, external socio-contextual trust evidence may also be gathered, as in the example given in Section 2.

4 Community-Centric Authentication Sequence

In order to enable sharing of user authentication tokens, it is convenient to model the authentication mechanism as a *key re-construction* step. In systems that support encrypted file systems, this key may indeed be used as a cryptographic encrypting key to protect the user's data. In regular systems, the key may be used to simply encrypt (or key-hash) a particular system object. The result is then verified against an authenticator value stored in the user database to verify the user's identity.

In the non-exceptional case – i.e., when the user remembers his password, or is in possession of the authentication tokens – the password and other authentication information is combined with user's information stored in the system database (here abstracted as a *salt*) to derive the authentication key. When the user misses authentication tokens/information, he may instead use vanilla access to the system's communication resources to request that his peers release to the system the key shares that will enable reconstruction of a *pre-key*. This pre-key can then be combined with a second pre-key from the system itself to reconstruct the user authentication key, as shown in Figure 1.

The use of a system pre-key is required to ensure that the set of shares entrusted to the community is semantically independent from any user password-derived value. This eliminates the possibility that the peer community may collude to reconstruct a password-derived value and use that to attack the user password

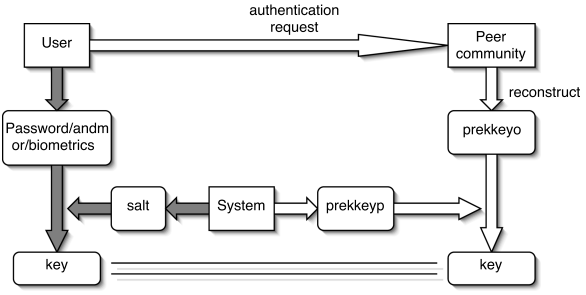

Fig. 1. Login sequence: The shaded arrows refer to the non-exceptional case. The light arrows refer to a request initiated by the user with only vanilla access.

(e.g., via off-line dictionary or brute-force attacks). A separate advantage to the use of pre-keys is that it facilitates peer community update. Whenever an update takes place, the pre-keys are also changed, ensuring that old shares from an earlier community cannot be combined with new shares from a current community to achieve user authentication in violation of the current trust infrastructure. In this fashion it is possible, for instance, to support peer community disenrollment without requiring users to update passwords, by simply distributing new key shares.

A more sophisticated authentication sequence is shown in Figure 2. Here, an additional step is introduced – the system must interact with the user to obtain additional information in order to recover the authentication key. A typical example is to require the user to answer a set of questions and verify the answers against a database. A different approach would be to use the answers directly to obtain key-like information, as in personal entropy systems [5,6].

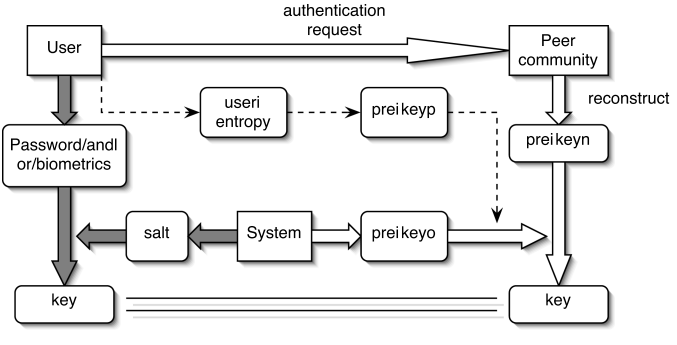

Fig. 2. Another login sequence: The dotted arrows refer to the use of a personal entropy system

The basic tools required to implement the above described authentication sequence are standard off-the-shelf tools. We require secure (private and authenticated) point-to-point channels, so that the shares of authentication keys do not

leak, and techniques to derive keys from strings. In case of encrypted file systems, we may require techniques such as password-based encryption. References to all these mechanisms may be found in [9]. As for the sharing the file-encrypting master keys, there are several practical secret sharing mechanisms that can be readily used [7,8]. Finally, the peer community trust infrastructure is essentially the access control structure of a secret sharing scheme; this can be based on an unstructured trust infrastructure, such as PGP [10].

5 Vanilla-Rollback: A State of Suspension

An important enabling feature of vanilla-rollback[3] is that if a session ends before authentication is complete, the session manifestation may be suspended; neither committed nor discarded, neither alive nor dead. If the questionable session is later authenticated, manifestations can be triggered and the system state updated as though the actions were taken at the time they were initiated by vanilla-Alice. Conversely, if an impersonation attempt is recognized, Vanilla-rollback can revert the system to its original state, essentially rolling back changes that vanilla-Alice performed.

Suspended transactions are not uncommon in today's systems. For instance, journaled file systems delay the commitment of file changes for various reasons, such as reducing disk-write latency and guaranteeing atomicity of file system changes [11]. Another prevalent illustration of suspended transactions include credit card accounting. On most online credit card status systems, recent transactions reflect a temporary status. Most of these are quickly confirmed or withdrawn, but occasionally temporary transactions linger, awaiting supplementary action to become final.

5.1 Rollback Technology

The most important element of an effective rollback mechanism is partitioning. Many actions cannot be rolled back; for example, once someone has viewed a data item, we cannot make them forget it. Consequently, vanilla-access cannot grant read access, except to already world-readable data. However, there are *many* actions that every user takes that can be safely, universally allowed: Document creation and accessing some web sites are two examples of commonly allowed actions.

To protect against inappropriate access, each action taken must be identified as either: (1) Fully vanilla; (2) Vanilla-Rollback; (3) Neither. Actions in the first class need not be monitored beyond their original recognition. They pose no threat,[4] divulge no sensitive information, nor change managed state as defined by

[3] Rollback in SQL cancels proposed changes in a pending database transaction. In this paper we use *rollback* as both an enabling and disabling procedure.

[4] These actions potentially facilitate a denial-of-service attack by exhaustion of available resources by vanilla-users. This threat can be mitigated by restricting the number of vanilla instances of the same user within the system, and by imposing more restrictive quotas on such users.

the prevailing security policy. Opening a browser, reading a public newsgroup, or starting an SSH session are actions that fall into this class in many organizations.

Items in the second class pose no immediate threat and accomplish only actions that can be reversed. Creating a document and adding a record to a database are potential examples of vanilla-reversible actions.

The third class encompasses all actions that are not in the first two classes. At a minimum these include any action that (a) necessarily requires authenticated access (is not fully vanilla) and (b) is not reversible. These are sensitive actions that cannot be rolled back and are always prohibited to vanilla users.

5.2 Rollback Application

The reference monitor (RM) [12] is the foundation of many access control systems and can implement VRA. Action partitions must be complete and applications must be engineered to provide VRA, essentially requiring them to incorporate temporary (commit-confirm) transaction processing. Applications without such capabilities may utilize VRA, but only if they are fully vanilla. As we mentioned earlier, many computing actions are universally safe, others are somewhat dangerous, and still others are always risky. Write-only access can be quite dangerous. For instance, vanilla-Mallory might add invisible executable files to Alice's

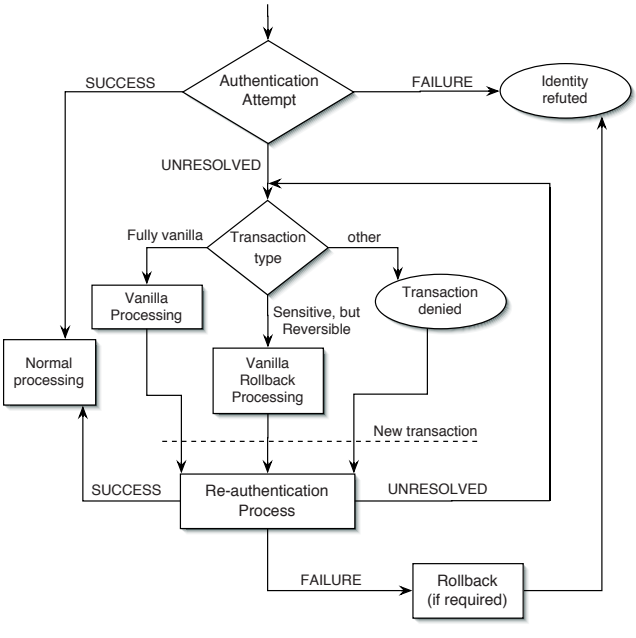

Fig. 3. Vanilla-Rollback Workflow. If the initial authentication attempt leads to an inconclusive state, transactions are either rejected (if they require authentication in an essential way), fully processed (if its execution does not require an authenticated state) or processed but not committed (to be reversed if final attempt at authentication fails).

account and cause mayhem when the real Alice comes in. Clearly, any executables installed during a vanilla session should be controlled appropriately—for instance, their execution by real users could be denied. Only after the questionable vanilla section is authenticated might such programs be executable at higher levels of privilege.

Transaction commit-confirm processing is one approach to rollback processing. An essential element for any rollback mechanism in VRA is containment. Temporary transactions must be separate from permanent state. This is essential to ensure that a vanilla process does not accomplish non-vanilla actions. For example, write access to vanilla users should create temporary files that are not committed to the user's file system if the vanilla logged user leaves the system without being promoted to full access status. This temporary file may provide forensic evidence if authentication is not completed. Whether a vanilla session was authenticated or not, the authenticated user could receive notification of the vanilla session, and could be given a choice of options -authenticate, force logout, or track the impostor, flagging her to the system administrator. Finally, once a session is purged as illegitimate, its contents are not discarded, but logged and used as evidentiary information for forensic analysis of intrusion attempts –perhaps even leading to the intruder's capture. We illustrate VRA in Figure 3.

6 Extensions

There are several ways in which our model for community-centric rollback can be extended. In this section we discuss some of them.

In our basic model, rollback is managed by the trust infrastructure of peer communities. Alice's trusted friends (e.g. Marla) have shares of her secret key, and use these to authenticate Alice, if they are convinced that she is the real Alice. Their decisions will be based on several events that partially identify Alice. For instance, Marla may remember that Alice was late and was wearing a blue sweater, and that she told her that she also loved cats. Our trust model is dynamic and context-driven: it is based on the augmentation of the (essentially static) infrastructure of the peer communities by incorporation of socio-contextual information to support trust dynamically. This information is out-of-band information and volatile. It is the kind of information that humans are ideally suited to manage and machines cannot, at least in any effective way.

A first and natural extension (from a deployment perspective) would be to use other communication media that are potentially more effective at quickly capturing socio-contextual trust data. Example of such mechanisms are video or audio chat tools. These are, in many respects, ideally suited to capture trust data which machines cannot recognize.

Another extension would be to involve enlarging the peer community, to include the system itself, with regards to the management of rollback. The system may be enabled with capabilities to monitor (human) user behavior and to develop biometric profiles of users. Profiling is a popular mechanism for supporting

authentication, and has been proven to be a very effective tool. User profiling may be based on: face or voice recognition, or the recognition of keyboard stroke and other biometric patterns.

Recognition can be done by the system locally (at the host) or remotely. Both in the cases of local and remote authentication loco-temporal information can enhance the reliability of biometric profiling. For instance, vanilla-Alice is unlikely to be Mallory sitting at Alice's workstation in the mid-morning of a business day. Similarly, if the system dials Alice's home/cell phone and recognizes her voice requesting a login from home/airport —that is contextual evidence of such a nature as to be nearly impossible to forge.

7 Towards a Threat Model for Community-Centric Vanilla-Rollback Access

There is an inherent conflict between facilitating access to, and providing security for, protected computing resources. Thus, it is not surprising that an access control system with vanilla-rollback may offer intruders a fertile target for identity theft and system intrusion.

We distinguish between two types of intruders: authorized users, called *insiders*, and unauthorized users. Insiders (Mallory-Marlas) are the worst kind of intruders. In community-centric rollback systems they possess some share of the password-derived key of another user, say Alice, and will try to fool her community peers to get other shares so that they can impersonate Alice. Here we focus on such threats, since protection from these will also protect from the other threats.

Protection of a community-centric vanilla-rollback access system is achieved by: (1) Requiring that entities distribute shares of their password-derived key to community peers; (2) Allowing entities to choose their own secret sharing access infrastructure, and; (3) Supporting the access infrastructure with out-of-band socio-contextual trust information, optionally augmented by biometric-profiling information available to the system.

The first two protection items deal with secure key recovery in the traditional model [4]. The third deals with insiders, who may try to trick Alice's trusted community peers into revealing their shares, so as to impersonate Alice. In traditional authentication systems widely prevalent today, insiders can use their knowledge of facts about Alice to social-engineer their way through, or otherwise circumvent access controls. (In other words, they may know Alice's mother's maiden name.) However, that is a less significant advantage if the intruder's task is to fool Alice's trusted circle of humans. The third item in this access-control paradigm is an essential component of a hardened (and yet more human-friendly) security environment. In conjunction with the other features it shows that the access control approach based on rollback and community-centric trust infrastructures is robust against both insider and outsider attacks.

References

1. Hopper, N., Bloom, M.: A secure human-computer authentication scheme. Technical Report CMU-CS-00-139, Carnegie Mellon University (2000)
2. Hopper, N., Bloom, M.: Secure human identification protocols. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 52–66. Springer, Heidelberg (2001)
3. Denning, D., Branstad, D.: A taxonomy of key escrow encryption. Comm. of the ACM 39, 34–40 (1996)
4. Fouquè, P., Poupard, G., Stern, J.: Recovering keys in open networks. In: ITW 1999. Proc. IEEE Information Theory and Communications Workshop, IEEE Computer Society Press, Los Alamitos (1999)
5. Ellison, C., Hall, C., Milbert, R., Schneier, B.: Protecting secret keys with personal entropy. J. of Future Generation Computer Systems 16, 311–318 (2000)
6. Frykholm, N., Juels, A.: Error-tolerant password recovery. In: Proc. of the 8th ACM Conference on Computer and Communications Security, pp. 1–9. ACM Press, New York (2001)
7. Blakley, G.R.: Safeguarding cryptographic keys. In: Proc. of the National Computer Conference, vol. 48, pp. 242–268 (1979)
8. Shamir, A.: How to share a secret. Comm. of the ACM 22, 612–613 (1979)
9. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton, USA (1997)
10. Zimmermann, P.: The Official PGP Guide. MIT Press, Cambridge, MA, USA (1995)
11. Seltzer, M.I., Granger, G.R., McKusick, M.K., Smith, K.A., Soules, C.A.N., Stein, C.A.: Journaling versus soft updates: Asynchronous meta-data protection in file systems. In: Proc. of the 2000 USENIX Annual Conference, General Session, USENIX, the Advanced Computer Systems Association (2000)
12. Anderson, J.P.: Computer security technology planning study. Technical Report ESD-TR-73-51, Air Force Electronic Systems Division, Hanscom AFB, Bedford, MA (1972)