Using *Continuous* Biometric Verification to Protect Interactive Login Sessions

Sandeep Kumar Terence Sim Rajkumar Janakiraman Sheng Zhang

School of Computing, National University of Singapore
3 Science Drive 2, Singapore 117543
{skumar, tsim, janakira, zhangshe}@comp.nus.edu.sg

Abstract

In this paper we describe the theory, architecture, implementation, and performance of a multi-modal passive biometric verification system that continually verifies the presence/participation of a logged-in user. We assume that the user logged in using strong authentication prior to the starting of the continuous verification process. While the implementation described in the paper combines a digital camera-based face verification with a mouse-based fingerprint reader, the architecture is generic enough to accommodate additional biometric devices with different accuracy of classifying a given user from an imposter. The main thrust of our work is to build a multi-modal biometric feedback mechanism into the operating system so that verification failure can automatically lock up the computer within some estimate of the time it takes to subvert the computer. This must be done with low false positives in order to realize a usable system. We show through experimental results that combining multiple suitably chosen modalities in our theoretical framework can effectively do that with currently available off-the-shelf components.

1. Introduction

By continuous verification we mean that the identity of the human operating the computer is continually *verified*. Verification is computationally simpler than identification and attempts to determine how "close" an observation is to a known value, rather than finding the closest match in a set of known values. Verification is a realistic operation in the normal usage of a computer system because we can assume that the user's identity has been incontrovertibly established by a preceding strong authentication mechanism. It is also appealing because it can conceivably be offloaded to a hardware device that is properly initialized with user specific data upon successful login.

The sense in which we are using identity verification is weaker than the ultimate aim of techniques such as intrusion detection [4] which even attempt to detect misuse by the authorized user who would clearly pass the biometric verification test. However, host-based intrusion detection has not quite been successful in practice, either because of the computational requirements of handling voluminous amounts of low level trace, or because of the large number of false positives that result from an attempt to *sharply characterize* user behavior based on observed low-level traces. We believe that continuous verification, if realized efficiently with low false positives, can be important in high risk environments where the cost of unauthorized use is high. This can be true for computer driven airline cockpit control, computers in banks, defense establishments, and other areas whose use directly affects the security and safety of human lives.

Biometric verification is appealing because several of them that are easy to incorporate in ordinary computer use are *passive*, and they obviate the need to carry extra devices for authentication. In a sense, they are always on one's "person", and perhaps a little safer than using external devices which can be *separated* from their carrier more easily. However, biometric verification can be construed as a matching problem and usually makes a probabilistic judgment in its classification. This makes it error prone. Furthermore, when used passively like we are attempting to do, it can result in time periods with no samples or poor quality samples; for example, when the user is not looking directly into the camera, or when the surrounding light is poor. To avoid both these pitfalls, researchers have used multiple modalities, say, fingerprint and face images simultaneously. This makes classification more robust and is also the approach that we have taken in this work. Even when some modalities may be very accurate, they might be inherently limited in their sampling rate, so combining them with faster (albeit less accurate) modalities helps to fill gaps between successive samples of the better modality. However, the use of multiple modalities presupposes independent sampling so that not all modalities fail to generate a

valid sample at the same time.[1]

Building an effective *reactive* biometric verification system consists of many aspects. Not only must the verification results be integrated into the operating system, it can be critical to balance several conflicting metrics: namely, accuracy of detection, system overhead incurred during the verification, and reaction time i.e., the vulnerability window within which the system must respond when it detects that the authorized user is no longer present. This relationship is especially important when all these aspects are performed in software on the same machine that is being protected from unauthorized use.

In the rest of the paper we describe the theoretical underpinnings of our multi-modal biometric verification system, our implementation architecture, the OS kernel changes needed to make the system reactive to verification failures, and the performance impact of such a system on ordinary computer use. The goal is to render a computer system ineffective within a certain time period of verification failure. This time should be a conservative estimate of the time it would take someone to cause information loss (confidentiality, integrity, or availability [11]) on the system.

2. Biometrics in Brief

We begin with a brief introduction of some of the important concepts in biometrics and verification. Readers familiar with these concepts may skip ahead; while readers wanting more details can refer to [5].

2.1. Basic concepts

Biometrics is generally taken to mean the measurement of some *physical characteristic* of the human body for the purpose of identifying the person. Common types of biometrics include fingerprint, face image, and iris/retina pattern. A more inclusive notion of biometrics also includes the *behavioral characteristics*, such as gait, speech pattern, and keyboard typing dynamics.

When a biometric is used to verify a person, the typical process is as shown in Figure 1. The user first presents her biometric (e.g. the thumb) to the sensor device, which captures it as raw biometric data (for example a fingerprint image). This data is then preprocessed to reduce noise, enhance image contrast, etc. Features are then extracted from the raw data. In the case of fingerprints, these would typically be minutiae and bifurcations in the ridge patterns. These features are then used to match against the corresponding user's features taken from the database (retrieved based on the claimed identity of the user). The result of the match is called a *Score*, S, typically a real number between 0 and 1, where 0 means "most dissimilar" and 1 means "most similar". The final step is to compare S to a predefined threshold T, and output:

- a decision of "Accept" (when $S \geq T$), meaning the Verifier considers the user as legitimate, or
- "Reject" (when $S < T$), meaning the Verifier thinks that the user is an imposter.

Some verification systems also output "Unsure", to indicate that the sample cannot be reliably classified one way or the other. In this case, the user may be asked to re-present her biometric.

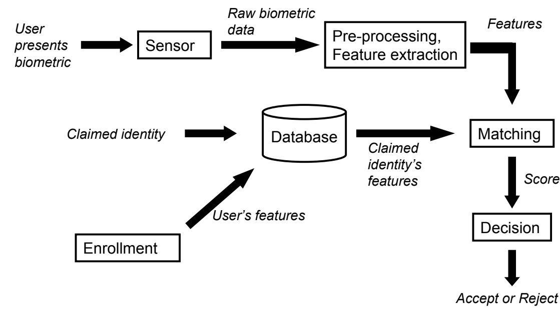


Figure 1. A typical biometric verification process.

Of course, the user's biometric features must first be entered into the database. This is done in an earlier one-off phase called *enrollment*. The process of enrollment is usually similar, consisting also of biometric data capture, preprocessing, and feature extraction. However, to increase robustness, multiple biometric samples are usually acquired (e.g. multiple images of the same finger), so that the verifier can "learn" the natural variation present in the user's biometric.

How accurate is biometric verification? There are two types of errors that a Verifier can make: a *False Accept*, or a *False Reject*. The False Accept Rate (*FAR*) is the probability that the Verifier incorrectly classifies an imposter as a legitimate user. This is a security breach. On the other hand, the False Reject Rate (*FRR*) is the probability that the Verifier incorrectly decides that the true user is an imposter. This is an inconvenience to the user, since she must usually resort to another means of verifying herself. In general, while a small *FRR* can be accepted as an inconvenience, a large *FRR* value can impact availability and may be construed as indirectly impacting the security of the system [11].

In an ideal Verifier, both the *FAR* and *FRR* are zero. In practice, there is usually a tradeoff between the *FAR* and *FRR* : a lower rate for one type of error is achievable only at the expense of a higher rate for the other. This tradeoff is usually described using the Receiver Operating Charac-

[1] Face and fingerprint may not be totally uncorrelated in that sense. However, that's not the thrust of this paper; rather this paper focuses on integrating multiple biometrics within an OS.

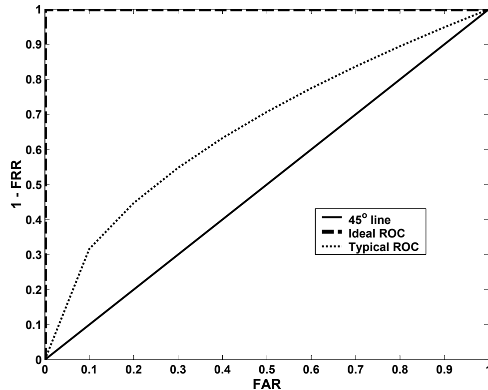


Figure 2. ROC Curves.

teristic (ROC) Curve (Figure 2), which plots $(1 - FRR)$ versus *FAR*. For any given Verifier, one can determine its ROC simply by varying its decision threshold T, running the Verifier on test data, and calculating the *FAR* and *FRR* for that value of T. The Ideal Verifier has an inverted-L shaped ROC curve, while an imperfect Verifier has a curve lying somewhere between the Ideal curve and the $45°$ line.[2] The *Power* of a Verifier is defined as the area under its ROC curve, and that is a useful measure of the Verifier's overall accuracy in a way that combines both its *FAR* and *FRR*. The greater the area, the better the Verifier. In general, fingerprint verification is considered more *power*ful than face verification.

When using multiple biometrics, individual classification results must be combined into a composite result. Computational overhead related to biometric processing must be balanced to get the desired tradeoff between usability, security and remaining computational power available for useful work. We describe these issues next.

2.2. Operational issues

① Computational overhead. Generally speaking, for all biometrics there is a tradeoff between computation and the Verifier's *Power*. For biometrics with weaker accuracy (less *Power*), multiple samples can often be combined to yield a more accurate composite assessment for that biometric[3]. But this requires more computation for a single assessment output, and for continuous verification it can add a factor to the computational load. An effective system must strike a balance between load and accuracy, especially when all biometric related computation is done in software on the same machine that is used for computing needs.

For example, in one set of measurements that we took for face verification, the CPU needed for our operating environment was nearly .2s per image, mostly incurred in locating the face in the whole image. This figure could be reduced to .1s by employing heuristics such as remembering the location of the face in the image, and using that as the starting point of face detection for the next image. The upshot is that processing about 10 frames per second would saturate the CPU. Adding multiple samples to increase accuracy of this biometric would seriously impact performance (about 10% for each extra frame rate). The alternative is to combine face verification with another, different biometric which has much higher accuracy.

② Usability versus Security. We consider the *FRR* of a biometric system as a measure of the system's *usability*, and its *FAR* as a measure of its *security*. With a higher false reject rate, the verification system deduces more frequently (but incorrectly) that the system is under attack and reacts by freezing or delaying the currently logged-in user's processes. This would unnecessarily delay the user's time-to-completion of ordinary tasks and may make the system frustrating to use. There is evidence that system response time is correlated to user productivity [7].

False rejects can be reduced by adjusting the decision threshold T of a biometric Verifier, but with a concomitant increase in the false accept rate. This could be disastrous from a security perspective. A usable system must balance its *FAR* against its *FRR*. Using at least one biometric with high accuracy can sharply distinguish a valid user from an imposter and can strike a good balance between the two choices. Higher accuracy can also be achieved at the cost of more samples but that increases the computational overhead, which impacts usability.

③ Choice of biometrics. For our design objective we need biometrics that are both passive and accurate. Passive biometrics do not require active participation by the user, (as opposed to active ones, such as those that use speech) and therefore do not intrude into the normal activity of the user by requiring them to periodically perform biometric related tasks that are not part of their normal activity. Such a requirement can be distracting and result in low system usability. Recently available computer peripherals such as the Secugen mouse [15] incorporates an optical fingerprint scanner at the place

Figure 3. SecureGen mouse.

[2]If a Verifier has an ROC curve below the $45°$ line, simply swap its "Accept" and "Reject" decisions and the ROC curve will move above this line.

[3]There are a plethora of techniques of combining them for e.g., using the sum, product, minimum, median, and maximum rules [6]. Other researchers have used decision trees and linear-discriminant based methods [13].

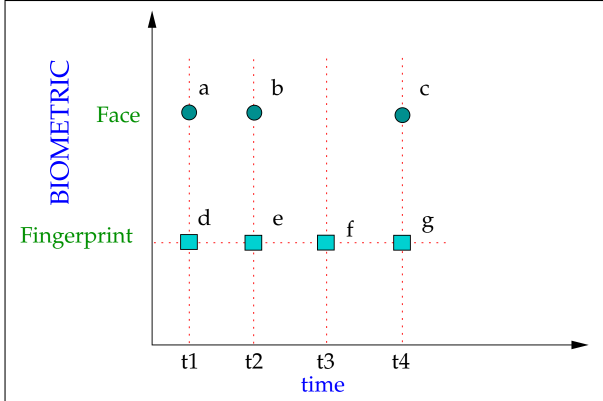


Figure 4. Combining multiple biometric modalities.

where a user would normally place their thumb (Figure 3). This device effectively turns fingerprint, a normally active biometric, into a passive one. Our other passive biometric is the face image, which can be acquired at a distance without the user's active cooperation.

④ Using multiple modalities. There is general agreement in the biometric research community, also supported by theory, see for example [13], that using multiple types (modalities) of biometrics (with an appropriate combination rule) can yield a higher classification accuracy than using only a single modality. In the context of our work here, combining face and fingerprint modalities is useful because there are frequent situations in which one modality is missing, e.g. when the user is looking away from the camera, or when the user is not using the mouse. Finally, attempting to thwart a multi-modal system is a much harder task than fooling a single-modality system.

There are two general ways of combining biometric data samples that are coming from different biometric modalities at different times [1]:

1. (Time-first) Combining samples of each modality first across time, and then combining them across modalities. In Figure 4, this scheme would first combine samples $a, b, c\ (= u)$ for face, and $d, e, f, g\ (= v)$ for fingerprint, then combine u and v.
2. (Modality-first) Combinining across modality first, then across time. This would first combine samples in the order a, d at the end of t_1, b, e at the end of t_2 etc., and then combine across the different times.

Recently we proposed a technique that combines the two approaches in whatever order the biometric data is made available [19]. This paper presents performance results using that technique of multi-modal fusion. The technique is based on Bayesian probability (see Section 3.3) and models the computer system as being in one of two states: *Safe* or

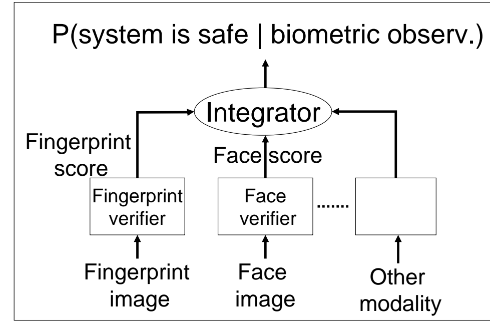


Figure 5. Integration scheme

Attacked. A *Safe* state implies that the logged-in user is still present at the computer console, while *Attacked* means that an imposter has taken over control.[4] The result of the fusion is the calculation of P_{safe}, the probability that the system is still in the *Safe* state. This value can then be compared to a pre-defined threshold T_{safe} set by the security administrator, below which appropriate action may be taken. A key feature of our method is that we can compute P_{safe} at any point in time, whether or not there are biometric observations. In the absence of observations, there is a built-in mechanism to decay P_{safe} reflecting the increasing uncertainty that the system is still *Safe*.

In the following section we describe our use of face and fingerprint biometrics in detail, as well as our technique for combining them.

3. Multimodal Biometrics

We use two modalities of observations: fingerprint and face images. The challenge is to integrate these observations across modality and over time. To do this, we devised the integration scheme shown in Figure 5. Our system currently uses the face verifier and a fingerprint verifier; other modalities are possible in the future. Each verifier computes a score from its input biometric data (fingerprint or face images), which is then integrated (fused) by the Integrator. In the following sections, we describe in turn how we compute the score for each modality and how we fuse them into a single estimate.

3.1. Fingerprint Verifier

We acquire fingerprint images using the SecureGen™ mouse (Figure 3). The mouse comes with a software development kit (SDK) that matches fingerprints, i.e., given two images, it computes a similarity score between 0 (very

[4] There is a possible *Absent* state, to model the situation in which the user has left the console but has not logged out. Because we are assuming a high-risk environment, it is justifiable to make *Absent* $\equiv$ *Attacked*.

dissimilar) and 199 (identical). Unfortunately, the matching algorithm is proprietary and is not disclosed by the vendor. Nevertheless, we've obtained good results using the score generated by this algorithm.

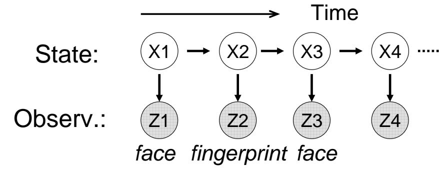
First we collect 1000 training fingerprint images from each of four users. Then, for each user we divide the training images into two sets: those belonging to the user (intra-class images), and those belonging to others (inter-class images). For each set, we calculate the pairwise image similarity using the proprietary algorithm, and determine the histogram of the resulting scores. That is, for each user, we compute two probability density functions (pdf) – the intra-class and inter-class pdfs (represented by histograms). Figure 6(a) shows the pairwise pdfs for a typical user. If we denote the similarity score by s, the intra-class set by Ω_U, and the inter-class set by Ω_I, then these pdfs are $P(s \mid \Omega_U)$ and $P(s \mid \Omega_I)$. Note that the pdfs do not overlap much, indicating that fingerprint verification is reliable (high verification accuracy).

Given a new fingerprint image and a claimed identity, the image is matched against the claimed identity's template (captured at enrollment time) to produce a score s. From this we compute $P(s \mid \Omega_U)$ and $P(s \mid \Omega_I)$. These values are then used by the Integrator to arrive at the overall decision. Section 3.3 has more details.

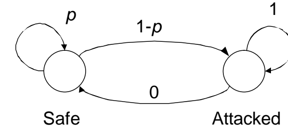
3.2. Face Verification

To train the face Verifier, we first capture 500 images of each of the four users under different head poses using a Canon VCC4 video camera and applying the Viola-Jones face detector on the image [18]. About 1200 face images are also collected of sundry students on campus to model as imposters. For each user, we construct training images from two sets: those belonging to the user, and those belonging to the imposter. All face images are resized to 28×35 pixels. For each set we calculate the pairwise image distance using the L_p norm (described below). This constitutes the biometric feature that we extract from the image and is similar to the ARENA method [14]. If we denote the similarity score by s, the set of legitimate users by Ω_U, and the set of imposters by Ω_I, then these pdfs are $P(s \mid \Omega_U)$ and $P(s \mid \Omega_I)$. We can now determine the histogram of the resulting scores. Figure 6(b) shows a pair of pdfs for one user.

The L_p norm is defined as $L_p(\mathbf{a}) \equiv (\sum |a_i|^p)^{\frac{1}{p}}$, where the sum is taken over all pixels of the image $\mathbf{a}$. Thus the distance between images $\mathbf{u}$ and $\mathbf{v}$ is $L_p(\mathbf{u}-\mathbf{v})$. As in ARENA, we found that $p = 0.5$ works better than $p = 2$ (Euclidean). Given a new face image and a claimed identity, we compute the smallest L_p distance between the image and the intra-class set of the claimed identity. This distance is then used as a score s to compute $P(s \mid \Omega_U)$ and $P(s \mid \Omega_I)$, which in turn is used in the holistic fusion step.



(a)



(b)

Figure 7. Holistic integration: (a) Hidden Markov Model; (b) State transition model.

3.3. Holistic Fusion

The heart of our technique is in the integration of biometric observations across modalities and time. This is done using a Hidden Markov Model (HMM) (Figure 7 (a)), which is a sequence of states x_t that "emit" observations z_t (face or fingerprint), for time $t = 1, 2, \ldots$ Each state can assume one of two values: $\{$*Safe*, *Attacked*$\}$. The goal is now to infer the state from the observations.

Let $\mathcal{Z}_t = \{z_1, \ldots, z_t\}$ denote the history of observations up to time t. From a Bayesian perspective, we want to determine the state x_t that maximizes the posterior probability $P(x_t \mid \mathcal{Z}_t)$. Our decision is the greater of $P(x_t = Safe \mid \mathcal{Z}_t)$ and $P(x_t = Attacked \mid \mathcal{Z}_t)$. Using a little algebra, we may write:

$$P(x_t|\mathcal{Z}_t) \propto P(z_t|x_t, \mathcal{Z}_{t-1}) \cdot P(x_t|\mathcal{Z}_{t-1}) \quad (1)$$

and

$$P(x_t|\mathcal{Z}_{t-1}) = \sum_{x_{t-1}} P(x_t|x_{t-1}, \mathcal{Z}_{t-1}) \cdot P(x_{t-1}|\mathcal{Z}_{t-1}) \quad (2)$$

This is a recursive formulation that leads to efficient computation. The base case is $P(x_0 = \textit{Safe}) = 1$, because we know that the system is *Safe* immediately upon successful login. Observe that the state variable x_t has the effect of summarizing all previous observations. Because of our Markov assumptions, we note that $P(z_t \mid x_t, \mathcal{Z}_{t-1}) =$

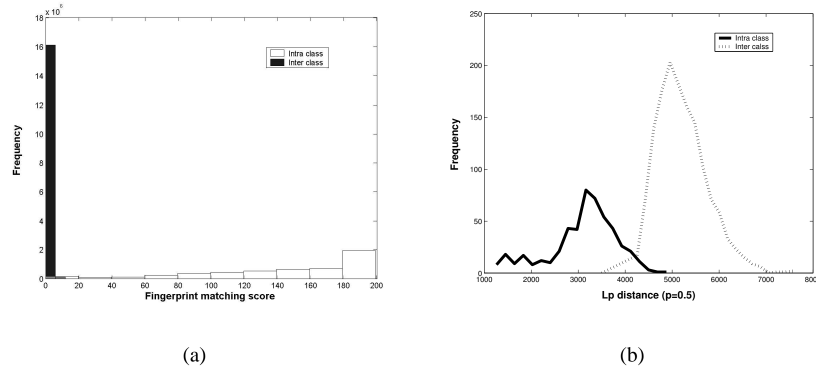


Figure 6. (a) Fingerprint intra-class and inter-class histograms for a typical user. (b) Face intra-class and inter-class histograms for a typical user. There is greater overlap in these histograms than in fingerprint, indicating that face verification is less reliable than fingerprint verification.

$P(z_t \mid x_t)$, and that $P(x_t \mid x_{t-1}, \mathcal{Z}_{t-1}) = P(x_t \mid x_{t-1})$. Also, $P(z_t \mid x_t)$ is determined from the pdf-pair (Figure 6(b) for face, and an analogous one for fingerprint). As for $P(x_t \mid x_{t-1})$, this is described by the state transition model shown in Figure 7 (b). In the *Safe* state, the probability of staying put is p, while the probability of transition to *Attacked* is $(1 - p)$. Once in the *Attacked* state, however, the system remains there and never transitions back to *Safe*. Finally, note that Eq. 1 is used to compute P_{safe} when there is a biometric observation, while Eq. 2 is used when there is no observation.

The value of p is governed by domain knowledge: if there is no observation for a long period of time, we would like p to be small, indicating that we are less certain that the user is still safe (and thus more likely to have been attacked). To achieve this effect, we define $p = e^{k\Delta t}$, where Δt is the time interval between the current time and the last observation, and k is a free parameter that controls the *rate of decay*, which the security administrator can define. In general, any decay function may be used to specify p, with a suitable rate of decay.

4. Integrating biometric feedback into the OS

Having considered some issues in the use of biometrics for security, we now consider design issues relating to its integration into the operating system to make the whole system *reactive*. We consider two mechanisms for reaction: *delaying* processes when $P_{\text{safe}} < T_{\text{safe}}$, or *suspending* them entirely, as in Somayaji's work [16].

Our model of protection is intended for single computer use to which users login through a bitmapped display (usually the console) that is directly connected to it. We also assume that biometric sensors feed data directly to the computer thereby insuring the integrity of both capturing and forwarding of the data for processing. Our current design affords continuous authentication protection to the "interactive" processes started by the user *after logging in*. This allows processes started upon system boot to be *exempt* from monitoring, and for privileged processes started after user login (such as executing setuid programs) to remain within the purview of continuous authentication.

We consider the following design choices.

① Identifying interactive sessions. For us, an interactive session consists of all processes derived from the initial console login. In Unix-based operating systems, there is usually a focal point in the form of a display manager (akin to getty), such as the KDE kdm program, that collects the user name and password for authentication before starting the user's X session. By tagging this process and every process derived from it through a fork()-like inheritance mechanism, we can tag all processes belonging to a session.

However, it is possible for the same user to login more than once (at different times, therefore different sessions) and still have processes from an earlier session running, so we must decide whether later logins also authenticate processes started in earlier interactive sessions, or whether each login session is considered as distinct. The former choice can be easily implemented by using a *user id*-based mechanism for process monitoring. In such a mechanism, only a process's *uid* field is examined to determine whether it is subject to continuous authentication. This would necessitate that the same *uid* not be used for both login sessions and for doing background activity because user logout would result in delaying or freezing such processes. An example of

a useful service that would be impacted is the use of *cron* and *at* job processing which may happen at any time, even when the user is not currently logged in.

A more general approach would be to identify the entire process tree derived from the initial display manager as belonging to a session. This would enable daemons such as *cron* and *at* to work without being subjected to continuous authentication.

② When to enforce verification on processes. A simple way to implement this is to make the check upon system call entry. However, for compute bound tasks that do not make frequent system calls, it might be better to also check them before starting their time slice. It is not immediately clear whether the latter mechanism would be useful in practice, and that the former would not suffice.

③ Policies for controlling the monitored processes. How should we penalize processes when $P_{\text{safe}} < T_{\text{safe}}$? Do we delay or freeze a process, and if we delay it, for how long? Freezing a process may be considered as the extreme form of penalty. In some sense, the penalty charged to a process should depend on the "severity" of the action i.e., the potential damage that can result if the action was permitted. The study of Bernaschi et. al [2] is useful in that determination at the syscall level. They divide the Linux 2.2 system call set into four threat levels with level one being the most critical and level four being harmless. One possibility is to use their classification to assign penalty to a system call. Our implementation (Section 5) provides a mechanism that permits this specification for individual system calls.

For non critical actions, we could delay processes. The delay added to a system call ought to be a (inverse) function of the probability that the correct user is present at the console. But this number is readily available: we can simply treat the Score S (see Section 2.1) as the required probability. Recall that in making a verification decision, the Score S is compared against the threshold T (in our case, $S = P_{\text{safe}}$, $T = T_{\text{safe}}$). We can turn this into a formula for calculating delay:

$$\delta(S) = \begin{cases} 0, & S \geq T \\ e^{\left(\frac{1}{S} - \frac{1}{T}\right)} - 1, & S < T \end{cases}$$

where e is the exponential function. δ imposes exponential delay on the calling process as the probability of classification is further away from the decision threshold T. Different functions will result in different tradeoffs between security and usability. A function that changes more rapidly as a function of $(T - S)$ will provide better security but likely be less usable because *FRR* errors (incorrect classifications) will impose heavy delays on processes.

In general, it is conceivable that process penalty is not just a function of the system call but is rather a more general function of the state of the system, the state and history of the calling process, and the arguments of the system call. Prior work on sandboxing, for example, that of Niels Provos [12] is work in that direction. For our purpose, however, the simpler mechanism of delaying or freezing processes at system call entry suffices.

5. Implementation Architecture

Figure 8 depicts the various elements of our implementation and how they are integrated into the operating system. We have implemented this architecture on the Linux 2.4.26 kernel [17] with the KDE graphical environment running on the Redhat 9.0 distribution. For face image capture, we use the Euresys Picolo capture card and the Canon VCC4 camera. The captured images have a resolution of 768×576 pixels and are 24-bit deep. The fingerprint images are captured using the Secugen OptiMouse III. All experiments were performed on an Intel Pentium 2.4 Ghz machine with 512MB RAM. The details of the various elements of the architecture are described below under task-related groupings for ease of understanding.

① Starting continuous verification. When a user logs in at the console using the kdm session manager [3], kdm authenticates the user using a password. Additionally, it starts the face and fingerprint verifiers and initializes the *monitor* with the user-id of the user that has logged in. We achieve this non-invasively by using PAM [10] to realize the side effect. To do this we added an entry in /etc/pam.d/kde of the form

```
session  optional pam_contauth.so
```

which is invoked during the kdm execution. kdm, being PAM aware, calls the PAM login authentication routine. This results in calling pam_contauth which starts the face, fingerprint and monitor components of Figure 8, and sets the session number of the kdm process to be the value of a kernel maintained integer ca_global_session. This is done through a newly added system call. A "session" conceptualizes an interactive login session, and in order to tag all the processes started by the user in a given session, we maintain an integer variable in every process's task_struct that denotes its *session*. Because all the components of the K Desktop Environment are forked off kdm, the value of this variable is automatically inherited across process forks and remains intact across execs. The ca_global_session is a counter in the kernel that is incremented after every successful kdm login.

Once the monitor has the user-id of the logged in user, it loads the biometric *profile* (the biometric features to be used for verification) corresponding to the user and starts biometric data capture using the video and fingerprint boxes in Figure 8. The arrows in the diagram denote the direction of

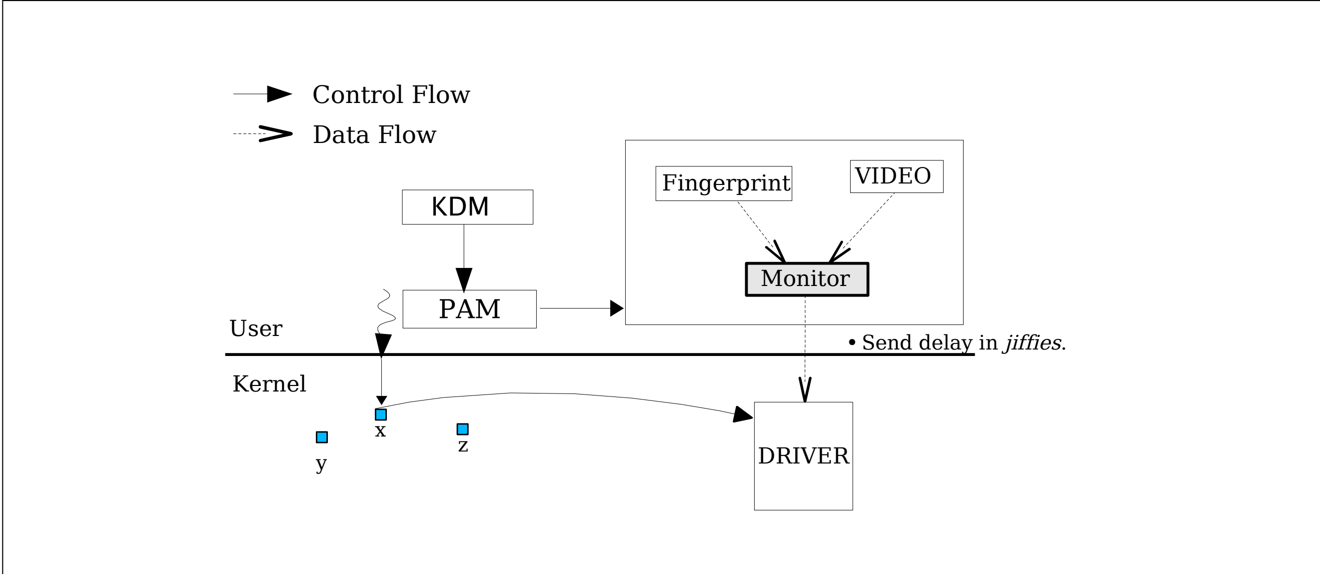


Figure 8. Architecture of a face verification system integrated with the operating system.

data flow. The monitor is the central coordinating entity in the architecture that performs the following tasks:

1. it controls the rate at which biometric data is captured by querying each biometric device and runs the modality-specific verifier for that sample (Section 3.2).
2. it combines the verification results from different modalities obtained at different times into P_{safe}, the probability that the computer system is still *Safe* (Section 3.3).
3. it periodically communicates P_{safe} (indirectly, it actually computes and communicates the delay value in jiffies[5]) to the kernel so that the kernel can appropriately freeze or delay processes.

② Controlling processes. To support the controlling of processes, we modified the Linux kernel as depicted in Figure 8. When a user process makes a system call, it traps into the OS kernel and eventually executes the code that implements the system call [8]. We introduced control processing just before the system call is *dispatched*. To do this, we add a kernel global variable (contauth_cb), which is a function pointer to code that implements the processing. This allows the processing code to be dynamically added to a running kernel and also serves to localize kernel changes. This function is invoked for *every* process on *every* system call.

The total change in the Linux kernel amounts to three lines of assembly code in arch/i386/kernel/entry.S, about 100 lines in a newly added C file contauth.c and miscellaneous code including adding system calls to get and set the kernel variable ca_global_session, and to set a process's session_id adding to another 50 lines. Currently we have only one callback point in the kernel and that is where a system call is dispatched. In the future we will probably add more callback points in the kernel for finer process control, for example at the point where a process is context switched. The performance impact of this change is described in our micro benchmarks in Section 6.1.

The overall pseudo code of the kernel control processing is as follows.

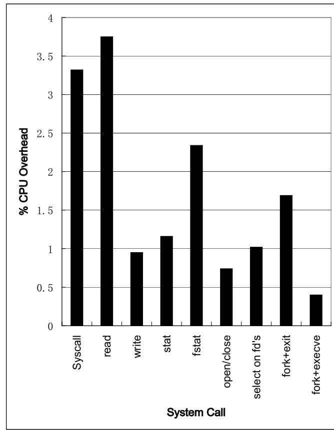
```
1  double x = current_biometric_classification;
2  boolean below_thresh = (x < threshold);
3
4  if(current->ca_sessid == 0)
5    do_nothing;
6  else if(current->ca_sessid == ca_global_session)
7  {
8    if(syscall is critical && below_thresh)
9      freeze yourself;
10   else if(syscall is !critical && below_thresh)
11     delay yourself by [e^(1/S-1/T) - 1] jiffies
12
13   //!below_thresh ⇒ do_nothing;
14 }
15 else if(current->ca_sessid < ca_global_session)
16   unconditionally freeze yourself;
```

As used in line 4, each process has a "session id" in its task_struct denoted by the field ca_sessid. A value of 0 means that the process is *not* rooted at *any* interactive session. Such processes are not controlled in any way as specified by the action in line 5. In the kernel, the variable ca_global_session identifies the session id of the *current*

[5] A unit of kernel time used by many kernel functions in the Linux operating system.

interactive session if it is in progress, or the session id of the *next* interactive session if none is in progress. Line 6 tests whether the progress belongs to the current interactive session and if it does, its activity is controlled. Otherwise, it may belong to a prior interactive session (line 15) in which case it is frozen. A utility program similar to ps can conceivably be written that examines every task_struct and sends a signal to each process whose session id corresponds to a prior interactive session. This would clean out frozen processes belonging to an earlier interactive session that will never be executed.

Lines 8-11 specify that *critical* [2] system calls be frozen, non-critical ones be left free while the remaining ones be delayed. The delay value is an exponential function of how far the current probability estimate of user presence is from some suitable threshold.



(a) Micro benchmarks

	Real	User	Sys
without contauth verification	276	258	16
with contauth verification	346	263	17
Overhead	$\approx 25\%$		

(b) Macro benchmarks

Figure 9. Performance benchmarks.

When P_{safe} exceeds T_{safe}, all frozen processes in the current interactive session are "unfrozen", and delayed processes are made runnable. This is practically important and affects system usability because if the user looks away from the camera and does not have his finger against the mouse, the system may start delaying his processes. But as soon as a good sample is obtained, the system ought not to penalize processes that are currently being delayed and wait until their duration of delay has ended. Because the exponential function can produce very large delay values as $P_{\text{safe}} \to 0$; to ensure a rapid recovery once the monitor regains confidence in user presence, the driver issues a wakeup call to all processes that were delayed.

6. Performance

We describe results of both micro and macro benchmarks.

6.1. Micro benchmarks

To assess the performance impact of our Linux kernel changes, we ran the lmbench [9] suite to determine the overhead introduced in the system call path. The results are shown in Figure 9.

The percent overhead on the y-axis is the percent increase in time for executing a system call with our modifications for stopping and delaying processes when compared with a standard 2.4.26 Linux kernel that can be downloaded from www.kernel.org. The overhead is dependent on the system call exercised. The overhead is as low as $.4\%$ for the fork+execve combination to a 3.75% overhead for read. We believe this to be acceptable.

6.2. Macro benchmarks

For macro benchmark tests we assessed the performance impact on compiling the Linux (2.4.26) kernel. The compilation generates about 1200 object files. We chose the Linux kernel compilation because it pollutes the cache and its processor utilization is significant. The face biometric is sampled twice per second while the fingerprint biometric is sampled once in two seconds. The numbers in Figure 9 are averages over three runs. The overhead is about 25% for our operating environment.

7. Usability

A standard metric for assessing the usability of a biometric is its *FRR*. In our system, false rejects result in process delays, so one way to measure usability is the delay that ordinary tasks suffer in their time-to-completion. If the overhead (reflected as *delay*) introduced by the normal use of biometrics is $x\%$ (see Section 6), then we are interested in determining how much further ordinary tasks are delayed under normal use of the system. We ran some simple operations that ordinary users might perform in their use of a computer to assess this difference.

1. ls -R /usr/src/linux-2.4.26 results in a "real" time overhead of 36%, about an 11% increase.

2. ls -R /usr/local results in a "real" time overhead of 37%.
3. grep -R <key> /usr/src/linux-2.4.26 results in a "real" time overhead of 44%.

All times are averages of 5 runs. So the impact on usability of using the system in practice is an extra 10-20$\%$ degradation. While the biometric verification can conceivably be offloaded to extra hardware, the delays resulting from *FRR* errors cannot.

For our operating environment, our security goals seem to be met although that is a qualitative judgment at this point. We have tried to switch users suddenly and execute rm /tmp/foo, but the system freezes before the command is fully typed. A caveat is that key strokes by the imposter may not be delivered to the application (shell) but only because it is not executing. When the correct user comes back, these key strokes would be delivered and damaging action performed. To be totally secure, the tty/pty driver or the X server must somehow be made to discard all user input when a process is delayed or frozen.

8. Conclusion and Future Work

We believe that the reactive system that we set out to build works reasonably well at this point. Biometric verification is the main bottleneck in the computation and we are looking into how to offload that into an FPGA-based implementation. We are also investigating how to derive a mathematical basis for computing the "sweet spot" of the system that maximizes a utility function, such as $U(u) + S(s)$ given the various parameters of the system. u is the raw fractional delay overhead in using the system and $U(\cdot)$ maps it to a utility value. Similarly s is a security metric, e.g., the *FAR* of the system, and $S(\cdot)$ maps it to a utility value. u and s in turn are functions of the biometric modalities, their ROC curves, the number of samples used for each biometric decision, and the multi-modality fusion method.

The thrust of this paper is less towards biometrics per se, although our multi-modal combination technique is new; rather it is about how to integrate biometrics as a useful general abstraction into the operating system so that all processes can gain from it, with the aim of enhancing the security of the system. Now that newer biometric devices are commonly appearing that can permit passive biometrics to be integrated into normal computer use, such abstractions can be useful to investigate at a lower layer so that computer response can be provided in a more general and encompassing manner.

Acknowledgements

This work was funded by the National University of Singapore, project no. R-252-146-112. The anonymous reviewers gave excellent feedback that has helped improve the presentation of the paper.

References

[1] A. Altinok and M. Turk. Temporal Integration for Continuous Multimodal Biometrics. *Proceedings of the Workshop on Multimodal User Authentication*, December 2003.

[2] M. Bernaschi, E. Gabrielli, and L. V. Mancini. REMUS: A Security-Enhanced Operating System. *ACM Transactions on Information and System Security*, 5(1):36–61, 2002.

[3] N. Crook. The kdm Handbook. Available at http://docs.kde.org/en/3.1/kdebase/kdm/.

[4] D. E. Denning. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, 13(2), February 1987.

[5] R. Duda, P. Hart, and D. Stork. *Pattern Classification, 2nd edition*. John Wiley and Sons, 2000.

[6] J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas. On combining classifiers. *IEEE Trans. on PAMI*, 20(3):226–239, Mar. 1998.

[7] G. N. Lambert. A comparative study of system response time on program developer productivity. *IBM Systems Journal*, 23(1):36–43, 1984.

[8] R. Love. *Linux Kernel Development*. SAMS, 2003.

[9] L. McVoy and C. Staelin. *lmbench*: Portable Tools for Performance Analysis. *USENIX 1996 Annual Technical Conference*, January 1996.

[10] A. G. Morgan. The Linux-PAM System Administrators' Guide. Documentation distributed with Linux-PAM. Available at http://www.kernel.org/pub/linux/libs/pam/pre/library/.

[11] C. P. Pfleeger. *Security in Computing*. Prentice Hall, 2nd edition, 1996.

[12] N. Provos. Improving Host Security with System Call Policies. *12th USENIX Security Symposium*, August 2003.

[13] A. Ross and A. K. Jain. Information fusion in biometrics. *Pattern Recognition Letters*, 24(13):2115–2125, 2003.

[14] T. Sim, R. Sukthankar, M. Mullin, and S. Baluja. Memory-based Face Recognition for Visitor Identification. In *Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition*, 2000.

[15] S. B. Solutions. Secugen optimouse iii. http://www.secugen.com/products/po.htm.

[16] A. Somayaji. *Operating System Stability and Security through Process Homeostasis*. PhD thesis, University of New Mexico, Department of Computer Science, July 2002.

[17] The Linux Kernel Archives. http://www.kernel.org/.

[18] P. Viola and M. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 2002.

[19] S. Zhang, R. Janakiraman, T. Sim, and S. Kumar. Continuous Verification Using Multimodal Biometrics. In *The 2nd International Conference on Biometrics*, 2006.