ATTESTATION-BASED REMOTE BIOMETRIC AUTHENTICATION

Thomas Polon
Clemson University, tpolon@alumni.clemson.edu

ATTESTATION-BASED REMOTE BIOMETRIC AUTHENTICATION

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Engineering

by
Thomas Polon
May 2007

ABSTRACT

Migration from password and token-based authentication in distributed systems requires fundamental changes to the authentication process. A person's biometric data is not a secret, which presents a fundamental difference with other authentication methods. Matching a sample with a database template is secondary to establishing trust in the integrity of the sample. The process is similar to establishing a chain of custody for judicial evidence. In computer systems this is accomplished using attestation architectures. In this paper, a design for a secure remote biometric login system based on an attestation architecture is analyzed. The system uses a commercially available Trusted Platform Module (TPM) to authenticate the platform during the boot process and perform trusted private-key functions to participate in a challenge/response between the client and a remote biometric matcher. The result is a system that can provide higher assurance than current systems in an economically and administratively feasible system.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER ONE
INTRODUCTION

Using biometric authentication as a primary means for system authentication is a viable alternative to using password authentication. However, this technology is still in its relative infancy, and issues concerning secure implementation of biometrics remain. Aside from assessing and improving the performance of specific biometrics, most current and past research in biometric system security revolves around protection of the biometric template. In this research, an attestation architecture is used in conjunction with biometric middleware to establish authenticity in the biometric capture process, resulting in a trusted path for secure biometric authentication.

In most current authentication systems, the primary means for authentication is by password. However, most passwords are simple enough that they can be easily guessed or broken through simple dictionary attacks [6]. Thus, the system is only as secure as the password used for authentication. Simple passwords are easy to crack and compromise security. However, complex passwords are difficult to remember and are expensive to maintain. Adding to the problem, most users use the same password for different authentications/applications and therefore, if a password is compromised, it gives an attacker the full privileges assigned to the user.

Password limitations can be improved by incorporating other methods of user authentication. One such method is biometric authentication. Biometric authentication refers to verifying individuals based on their physiological or behavioral characteristics. This can include such things as fingerprints, hand geometry, retina, and voice. Biometric authentication is potentially more reliable than password-based authentication, since biometric characteristics cannot be lost or stolen, they are difficult to copy, and they require presence at the point of authentication. Consequently, biometric authentication has the potential to become the primary authentication method in computing systems, but technical advances are needed in the three areas listed below to improve security.

- *Performance* – reliable, unique measurement of biometric for widest set of users
- *Liveness testing* – detection of physical spoofing
- *Sample authenticity* – establishment of sample collection process

Performance and liveness testing involve modality-specific improvements, and these metrics are important to assess before considering the use a particular biometric. This paper focuses on the third issue: sample authenticity, which current remote authentication systems give little attention to.

Matsumoto and others successfully used fake fingerprints to achieve a biometric match [5]. Improved liveness testing can prevent physical attacks like these that do not compromise the system, but signal injection attacks will continue to be possible because sensors do not bind the sample to the identity of the sensor or the time that the sample was taken. For instance, retina scanning, which would be very difficult to spoof through physical means, can still be recorded and replayed at the sensor level. The use of templates protects privacy and prevents reuse of biometric samples between systems [4, 16], but there is little to prevent a dishonest party from capturing a biometric in raw form to be used as a live sample in another system. As biometric identification becomes widespread, people will become accustomed to frequently providing biometric samples, and the possibility of a Trojan sample acquisition or a dishonest administrator is realistic. The result is similar to using the same password for multiple systems. When personal biometrics are no longer a secret, then how can an automated system be sure that a biometric sample is coming from a biometric device and not recorded data? Consequently, biometric samples must be treated more like evidence and less like a password.

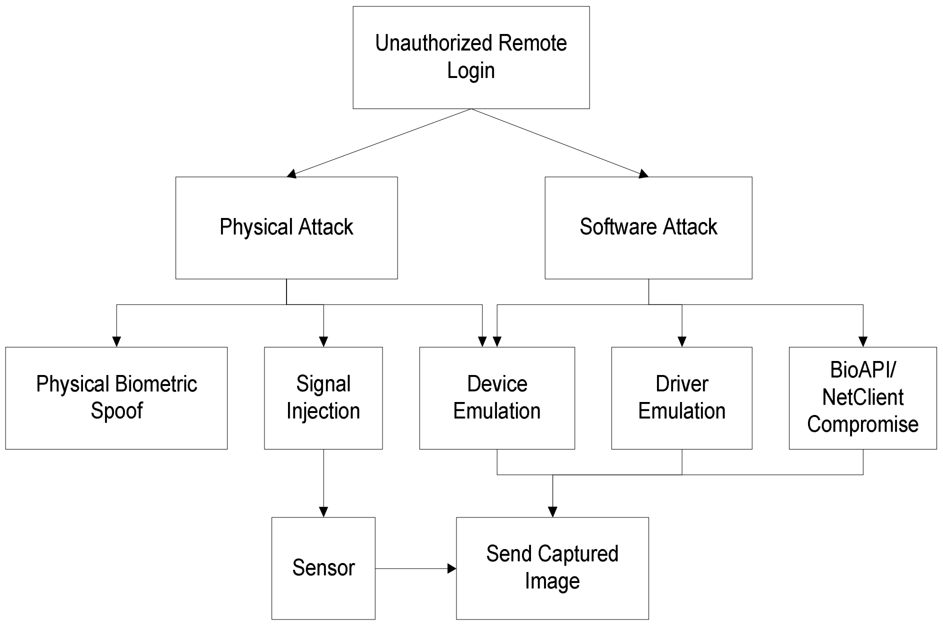


Figure 1: Attack Tree For Generic System.

Consider the attack tree in Figure 1, which represents a standard remote biometric login without trusted hardware. The system can be attacked through software or hardware (locally), and there is little to prevent an attacker from inserting a biometric sample from anywhere in the network, perhaps from an untraceable location. The addition of a local private key provides minimal additional protection, because an attacker can use a virus or Trojan horse to retrieve keystores, capture biometric samples, and keystrokes from an authorized user's workstation.

From the example in Figure 1, it is clear that adding a biometric to an insecure system provides little benefit, and there is no distinction between a valid platform and an attacker's platform that can provide the same credentials. The opportunity for an attacker to insert or capture a live sample starts with the end sensor and continues up the chain until a trusted entity can sign or encrypt the sample. Ideally this would occur in the sensor itself, but this would require key management at the device level. Providing tamper resistance, enrollment and revocation at the device level would be ideal, but it is not considered in this paper due to the large administrative burden.

Instead, it is more reasonable to establish a chain of integrity within the platform and sign the sample by the local root of trust. The level of trust in the signature depends on the ability to protect the key and the ability to authenticate requests to use the key. If the key is protected by tamper resistant hardware, and if the authenticity of the software is verified prior to signing, then this establishes a trusted relationship between the platform and the live sample. This can be accomplished using a commercially available attestation architecture, where components vouch for the validity of information by signing with a private key.

In this research, the Trusted Computing Group's (TCG) Trusted Platform Module (TPM) [14] is used in the design of a secure remote biometric login system. The process involves a chain of several components, and each verifies the next component. The system remains independent of specific biometric technologies using the BioAPI specification [2]. A remote matching service is used to reduce the physical exposure of the final decision point, and the remote match is performed as a system independent step that can provide authentication credentials for multiple systems.

In the following chapter, the background on which this research was built upon is presented. First, the related work will be covered. It can be noted that most work related to biometric security revolves around template security, as mentioned earlier. During this section, reasons why template protection alone is not enough for biometric security will be presented. Following this, an overview of bioAPI as well as an overview of our attestation device will be given. In relation to bioAPI, reasons why it is not secure in its current form will be presented.

In the final chapter, the research methodology will be presented. First, an overview of the actual hardware and software used will be given. Second, an overview of the code used in this research is given. Next, the results are provided, which show how the work provides added security to a biometric system. Finally, conclusions are stated along with possible work in the future which can solidify a fully trusted path.

CHAPTER TWO
BACKGROUND

Given the assumption that raw biometric data cannot be considered a secret as discussed in the previous section, approaches that create secret keys or unlock secret keys using biometric data are incomplete without integrity verification. This is still true if the algorithms used to achieve these ends remain a secret, since raw data can be injected to replace sensor data. Creation of keys from biometric data removes the need to trust the matching algorithm, but this is not useful if the biometric sample is not a secret or is not combined with other secrets.

Using a local biometric match to unlock a key, which is used as the network login credential, can be even less secure than using a password. A biometric match is typically a Boolean value (after thresholding), providing no protection beyond code obfuscation, which is difficult to quantify. Password attacks at least have a measure of protection through repeated hashing to slow down a brute force attack. Even if sufficient protection is provided to enforce that a valid live sample is used to unlock the key, what mechanism prevents an attacker from using an electronic copy of a valid live sample to the matching algorithm?

2.1 Related Work

Most work related to the protection of biometric data centers around template protection. This is quite important as biometric data spends most of its time in template form. However, if the biometric is not a secret, how important and how protected is the template if the raw data itself can in no way be legitimized?

Sun et al. [10] proposed a template called KMT, or Key-Mixed Template. The idea is to use the template in conjunction with a secret key to create a new template. This mixture between the secret key and the original template is done on the user end and is matched on the server side with the database. By having a separate secret key per authentication system, having a template compromised does not necessarily mean the attacker can gain access to all systems that use that biometric template. This new template can help to prevent backend attacks, snooping, and tamper attacks without a performance hit.

Sutcu et. al. [12] proposed a method for protecting minutiae-based fingerprint templates using a geometric transformation. The solution creates a two-dimensional vector from a set of minutiae points. A centroid point is calculated from this minutiae set and a circle is drawn around it. For every pair of minutiae points in the set that are greater than a pre-defined threshold, a line is drawn through them and the two points of intersection of the line and circle are determined. The intersection points are then organized into bins according to their position in the circle and the number of points in each bin are concatenated together to create the fingerprint code. This sounds fine in theory, but no rigorous security testing has been done as of yet, so the robustness of the code is not known. Furthermore, like other work which only deals with template protection, this work would not be able to protect against a raw biometric data compromise.

Sutcu et. al. [11] also proposed a system to protect biometric templates by using *secure sketch*, an error-tolerant cryptographic primitive. They use feature vector extraction on biometric samples, apply user-specific random mapping on these vectors, and then secure them using the sketch. However, they note that the security measure in terms of entropy loss may not be sufficient since false accept rate and false reject rate should also be considered in practical systems.

Uludag et. al. [16] proposed to generate a cryptographic key using biometric information. Their solution proposed to hide a cryptographic key in the user's biometric template itself. It relies heavily on the robustness of the key hiding and retrieval algorithms, which could be placed within the biometric reader device, requiring controlled biometric devices. However, this does not really help to add security if the biometric itself is not a secret.

Jain et. al. [4] constructed a list of possible biometric attacks which would leave biometrics vulnerable. From this list, they determined that it is of high importance to make sure the template is secure and cannot be compromised. Possible techniques to increase security within templates include watermarking and steganography. Also, by using smartcards to store biometric templates, the chance of an attacker stealing the template is reduced considerably.

While all these systems help to protect the template, this still does not protect or authenticate the raw biometric capture data (before it is put into a template), and it also does not provide a trusted path from the capture device to the biometric server. While the biometric data does exist mostly in template form, and template protection is important, the raw data cannot be ignored. If an attacker were to steal the raw biometric data from one system and use it with a modified device in another to gain access, how could this be prevented? A protected template means nothing if it protects raw data that is easily stolen from another system and injected in the present one. For a system to fully trust that a user is who he or she says, the full path from the device to the authenticator needs to be trusted.

Using a TPM to enhance the security of a biometric login is not a new idea, and there are multiple commercial solutions and research concepts using this widely available security tool [1, 3]. However, current systems only provide a wrapper around legacy password based systems (a successful biometric match unlocks a password cache). This places full trust in an end device that could be subjected to a rigorous offline attack. Current systems also do not provide sufficient authentication of the software on the platform used to collect the biometric sample. For instance, current systems do not detect if the device driver has been altered to send a pre-captured sample rather than read from the device.

Ratha et. al. proposed a challenge/response system to alleviate the problem with resubmission attacks on biometric devices [7]. In their proposed method, the server would generate a random challenge and the sensor would compute a response to the challenge. Their research involves custom sensor devices, which use sensors that are able to integrate logic to participate in a challenge/response system. The research in this paper assumes no changes are made to the biometric sensor device.

Chen and others proposed an attestation based system to provide for the integrity of the biometric sample [3]. Their work required a custom, trusted biometric reader device and a modified TPM capable of performing a biometric match, which may not be practical. The research in this paper assumes that an unaltered TPM and a standard biometric reader will be used, albeit at the cost of less assurance in the capture process. Additionally, rather than performing a local match and releasing a key that provides for a login credential, this paper uses a local key to attest that a live sample was collected from what appeared to be a valid device and then the signed sample is sent to a biometric matching server. A remote matching server provides additional security measures: it removes the capacity for offline attacks of the matching process, it provides increased integrity, and it provides better auditing and management of the authentication process.

2.2 Attestation Overview

This system uses an attestation device to derive a trusted platform, which extends work by Safford [8] to include verification of the network client, BioAPI, and the biometric device driver. The research system uses the TCG Trusted Platform Module (TPM) Specification version 1.1b [14]. However, a newer version of this specification is currently available [15], which may be used in future development as resources permit. The TPM chip provides several security features to help make a system more secure [8].

The first is a boot-time authentication system. The TPM has the ability to measure the system as it is booted and as software is loaded. This creates a profile of the system's operating software. This profile is based on a hash computed by the TPM from code as it is loaded. The TPM has sixteen Platform Configuration Registers (PCRs) that can be used to store the hashes of the software boot chain. This checks the integrity of the BIOS, the MBR, trusted GRUB, Linux kernel, the initial Ramdisk image, the net client, BioAPI, and the biometric device driver. An example of this can be seen in Figure 2. In this system, each layer authenticates the next layer. This ensures integrity as the next layer is not loaded unless the previous layer has been approved. By doing this, we can provide evidence that all pertinent software components on the machine are unaltered.

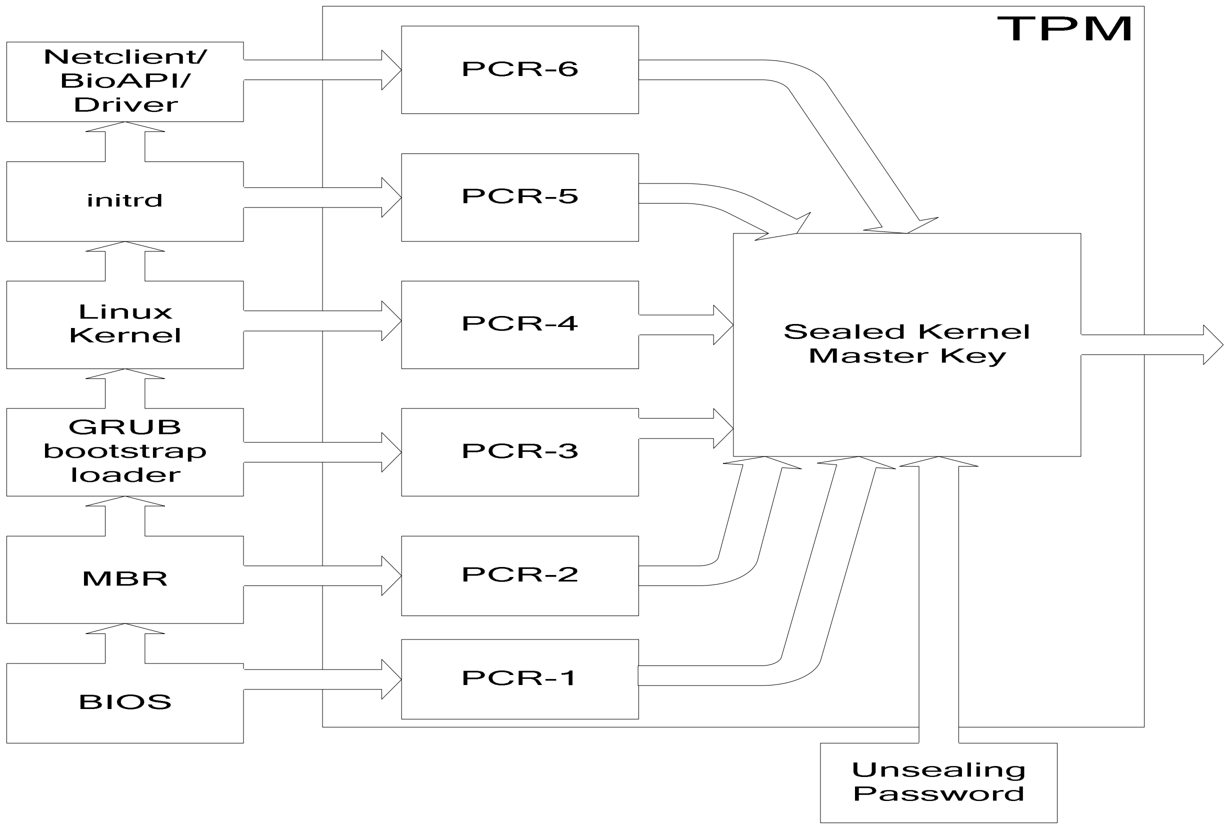


Figure 2: Block Diagram For Boot Sequence.

The contents of the TPM's Platform Configuration Registers can be compared to reference values at any time. This means that not only will all pertinent software be checked during the boot sequence, but if the software is reloaded at any time while the computer is on, the PCR will recheck the value with the reference value. If the values differ, an appropriate action (determined by the user) can be taken. Examples could be automatically denying access or prompting the user for a decision. The advantage to this is in situations when the intruder attempts to bypass the boot check by injecting a virus/worm after the computer has already booted.

The TPM can also provide additional security by signing the hash value stored in the PCR. The signed value serves as proof of the state of the remote machine. For a server to only read the hash value of a remote machine does not provide assurance that the machine should in fact have access to the network. This is because stored, unsigned hash values can be stolen from one machine and used in another for an attack. Another possibility is an attacker changing the values during transmission. By signing the values using a trusted key inside the TPM, the remote party can be assured that the hash is legit.

The TPM also provides hardware based public key management and authentication. The private keys for these services cannot be stolen by software attacks as they are generated and kept on-chip. Finally, it provides secure storage of data. This can include keys, which the TPM maintains over reboots.

2.3 BioAPI overview

The purpose of BioAPI [2] was to provide an API that could serve the various biometric languages. The BioAPI Consortium was formed in April of 1998, with a multi-level API architecture developed later in that year. In December of 1998, I/O Software joined and the BAPI specification was integrated as the lower level of the BioAPI specification. In March 2000, Version 1.0 of the BioAPI Specification was released.

BioAPI was originally developed to be a multi-level API. The "high level" would include basic calls such as enroll, verify, and identify. The lower level would address increasing control, detail, sophistication, and technology dependence. However, it was decided that the lower level proposals were not broad enough to justify inclusion into the API. Therefore, when Version 1.0 was published in 2000, it was done so as a single layer API.

Features

The API consists of three high-level abstraction functions: Enroll, Verify, and Identify. During Enroll, samples are captured from the biometric device, processed into a template, and then sent to the application. The Verify function captures one or more samples from the device, processes them into a template, and then matches the template against an input template. The results of this comparison are returned to the application. The Identify function is similar to the Verify in that a comparison takes place. However, instead of matching the captured sample template against a single input template that the user claims to be, the captured sample template is matched against a set of templates and a list is returned showing how close the sample compares against the top candidates in the set.

The processing of biometric data from raw samples to template to matching against a template can be accomplished in many stages. However, this usually involves much CPU-intensive processing. The API has been defined to allow the developer as much freedom as needed in the placement of processing involved. It also allows the processing to be shared between the original client machine and possibly a server machine. Furthermore, it allows for devices that can do all the processing internally.

There are other reasons as to why processing and matching are a good idea to do on a server. First, the algorithms will execute in a more secure and trusted environment. Second, the client PC may not have enough horsepower to run the algorithms as well, or not have the ability to support a large local database, both of which can be provided by a server. Third, the user database and resources may already be on the server, so this makes it more convenient. Finally, identification over large populations can only reasonably be done on a server. However, the developer needs to be very careful. Transmitting raw biometric data over a network between a client and a server could leave the data vulnerable to being captured by an attacker. This problem can be alleviated by creating the template on the client system, but again, this is up to the administrator as the client system might not be very powerful and take time to create the template.

There are two methods provided by the API which support client/server processing. The first is the use of *primitive functions*. There are four primitive functions in the API which can accomplish the same result as high-level abstractions. They are Capture, Process, Match, and CreateTemplate.

Capture is the process of capturing biometric data. This is always performed on the client machine. Doing so on a server (which would not have a biometric device attached) would return an error. The samples are acquired, done for the three basic functions mentioned earlier (Enroll, Verify, or Identify), and processing on the samples are then done. The *Capture* function is capable of performing all the processing on a sample and it is up to the developer as to how much is actually done on the client.

The *Process* function is used to process samples necessary for verification and identification, but not enrollment. These algorithms must be available on the server; however, they can also be available on the client. On the server, processing will always be completed. On the client however, the application can defer processing to the server. Also, in an effort to save bandwidth or server horsepower, the client can opt to finish the processing itself.

The *Match* function performs the actual comparison between the templates. It can be done between two templates (VerifyMatch) or a set of templates in a database

(IdentifyMatch). The match functions are always available on the server, and much like the process function, they may also be available on the client.

The last function included in the primitive functions is the *CreateTemplate* function. It is provided to perform the processing of samples to form a template. Optionally, the function can also take an existing template and create a new template from it, using new samples acquired. Another option is the ability to allow the application to provide a "payload" to wrap inside the new template. An example of a primitive function implementation is shown in Figure 3.

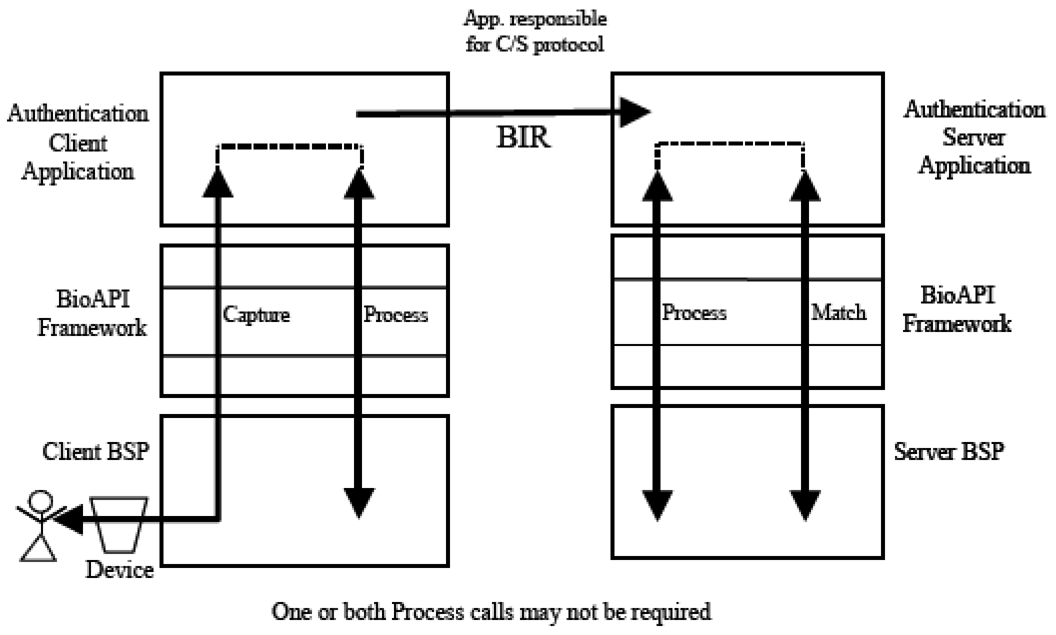

App. responsible for C/S protocol
Authentication Client Application
BIR
Authentication Server Application
BioAPI Framework
Capture
Process
Process
Match
BioAPI Framework
Client BSP
Server BSP
Device
One or both Process calls may not be required

Figure 3: Client/Server Implementation Using Primitive Functions (Taken From The BioAPI Specification, Version 1.1 [2]).

The second method provided by the API to allow client/server processing support is called *streaming callback*. In this method, the application is responsible for providing a streaming interface to transfer the samples and return the results. Also, the application does not need to use the primitive functions described earlier. The Verify, Identify, and

Enroll functions use the streaming interface to split the functions between client and server. If there are GUI callbacks set, the client will call them when needed to allow the application to control the look and feel of the UI.

Streaming Callback leaves the decision making to the application. The client/server application decides whether authentication is driven by the client or the server. First, the driving component sets a Streaming Callback interface for the biometric service provider (BSP). Not only does this tell the BSP that it will operate in client/server mode, but also provides an interface for communication. The application then calls the appropriate high-level function, and the BSP calls the Streaming Callback to initiate the BSP-to-BSP protocol.

The Streaming Callback is only used by the driving BSP. Whenever it is in control and has a message to deliver to its partner, it calls the interface to send the message, and receives an answer on return. A StreamInputOutput function is used by the non-driving application in order to deliver messages to the driving BSP. The driving application sends a return message by returning from the Streaming Callback. Figure 4 and Figure 5 show examples of a streaming callback implementation.

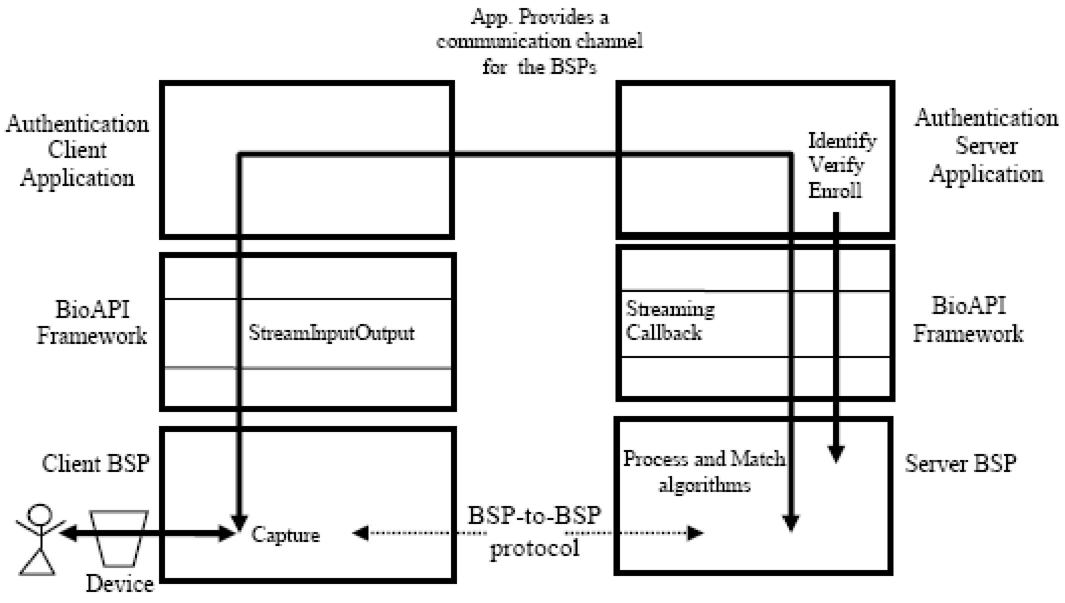

App. Provides a communication channel for the BSPs
Authentication Client Application
Identify
Verify
Enroll
Authentication Server Application
BioAPI Framework
StreamInputOutput
Streaming Callback
BioAPI Framework
Client BSP
Process and Match algorithms
Server BSP
Capture
BSP-to-BSP protocol
Device

Figure 4: Client/Server Implementation Using Streaming Callback - Server Initiated Operation (Taken From The BioAPI Specification, Version 1.1 [2]).

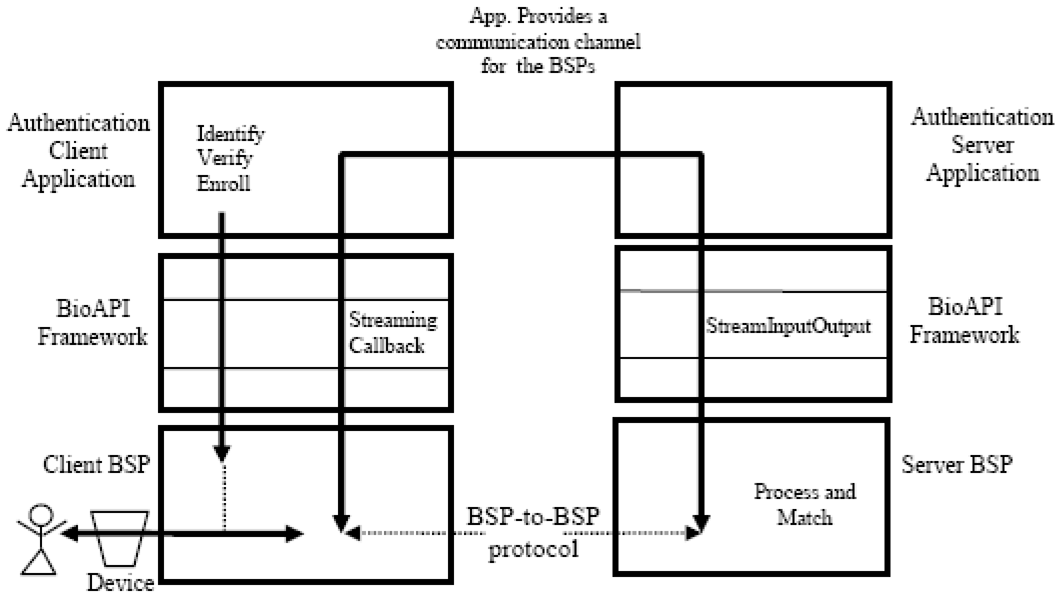


Figure 5: Client/Server Implementation Using Streaming Callbacks - Client Initiated Operation (Taken From The BioAPI Specification, Version 1.1 [2]).

Security Risks

As discussed earlier, there are some possible security risks associated with having an open system using biometrics and BioAPI. The most common types of attacks might be in the form of signal injection, a Trojan horse, or software alteration. All of these attacks might give an intruder the ability to spoof an allowed user to gain access into the system.

The first possible attack covered is a signal injection. Suppose an attacker was able to acquire a user's biometric data from some other system. The way it was acquired is not really known but that is not important. The attacker has the raw data of an allowed user and can therefore possibly inject this data into the system. As long as the data is injected before the step where the raw data is converted into a template, the system will have no idea that the sample was injected and not captured from the biometric device. If this attack were to occur, the attacker would gain access and the system would be none the wiser.

A second common attack would be a Trojan horse. This type of attack could be used in conjunction with the previous attack. For instance, the malicious script would extract the raw sample from the sequence chain (before it was converted to a template). This might happen on another system which is not as secure and therefore the attacker has access too. Or, it could be on the same system where the attacker has limited rights and is trying to gain access as a user with more rights on the system. An attacker can use a virus or Trojan horse to retrieve keystores, capture biometric samples, and keystrokes from an authorized user's workstation.

The third common attack discussed is software alteration. With an open system that has modifiable software and drivers, it is important that the software is not altered maliciously to allow attackers access. In the case of biometrics, some software might be altered to increase the false acceptance rate (FAR). Furthermore, BioAPI might be altered to pull templates or samples from locations other than the biometric sensor. It is important, therefore, to make sure that after the BioAPI software is installed, as well as the other pertinent biometric software, it is not being changed or altered. The system needs to be in the same working state that it was in when it was originally installed.

CHAPTER THREE
METHODOLOGY

The research system uses platform authentication and remote matching to create a medium-grade authentication using a single-factor biometric authentication (which would be backed up by a password and token for users that are unable to reliably use the system). The hardware of the user workstation consists of a biometric device (any device with a Linux BioAPI device driver) and a complete commodity workstation equipped with an embedded TCG compliant TPM. The major software components of the user workstation consist of a trusted boot loader, operating system, network client, BioAPI, and a biometric device driver. By using a TPM in a bootstrap process [7], there is a measure of assurance that the platform has not been altered and that the proper driver is used to collect the sample.

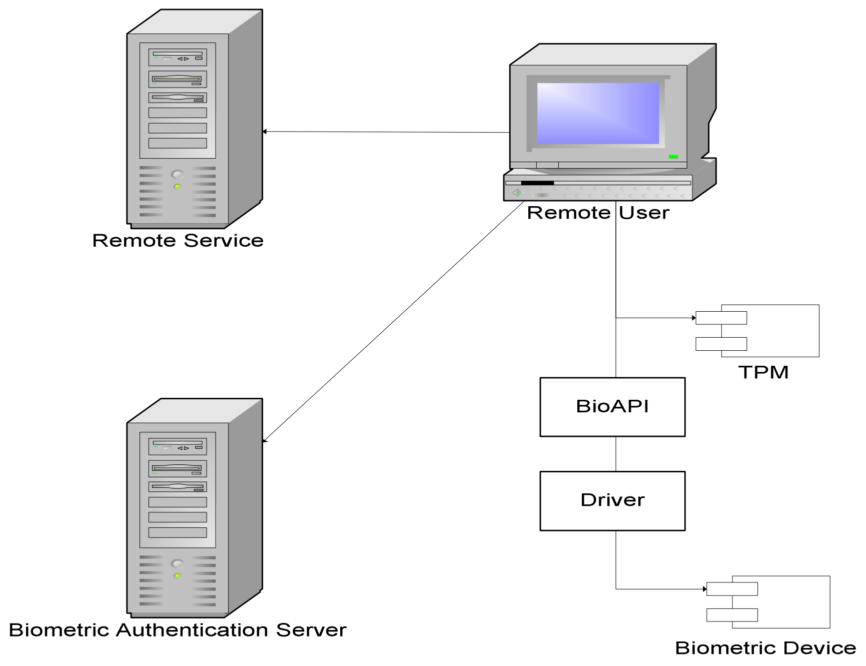


Figure 6: Client/Server Overview.

A Biometric Authentication Server (BAS), shown in Figure 6, is used to evaluate the validity of a biometric sample using evidence provided by the user workstation as well as match the sample with a known template. Notice how the BAS is completely a completely separate entity from the login/remote service. To enable validation, a root of trust acknowledged by the BAS must enroll each user workstation. Given a successful validation and match, the BAS returns the requested login credentials – ideally in the form of signed challenges. The BAS is similar to the user workstation with the exception of the additional BAS service software and a reduced set of user applications. Like the user workstation, the BAS is designed to use BioAPI to be independent of biometric technologies. Additionally, the BAS enables independence from network services enabling a one-time login and removal of the requirement for network services to be biometric-aware. A remote biometric match also enables the following security features.

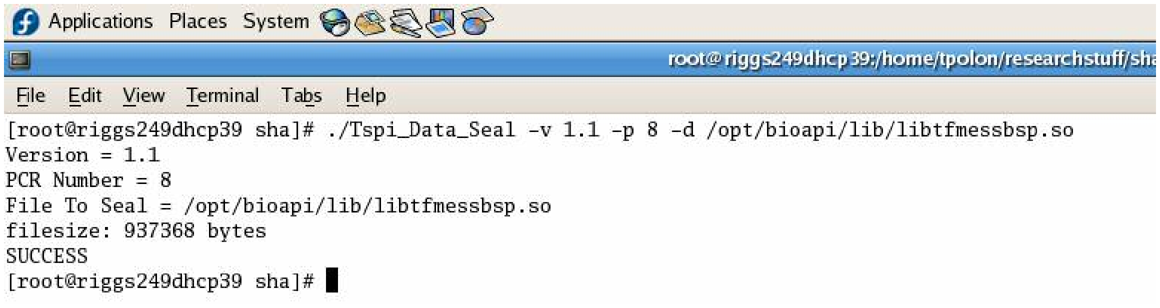
- User revocation
- Workstation revocation
- Multiple attempt lockout
- Convenient algorithm updates
- Reliable auditing
- Intrusion prevention possibilities

These features prevent an attacker with physical access to an approved platform from physically spoofing a valid biometric.

Our research platform is described in more detail as follows. The user workstation is an IBM ThinkPad z60t, outfitted with a TCG TPM (Version 1.1b) [9] and a built in UPEK fingerprint reader device. The operating system is the Red Hat Fedora Core 5 with Linux kernel version 2.6.15. The Linux fingerprint driver provided by UPEK is invoked through BioAPI [2] to collect the fingerprint sample from the reader. Trousers, the open-source TCG Software Stack, is used to interact with the TPM [8]. Currently, a small program that requests a biometric sample through BioAPI is used to emulate the network client.

A seal program needs to be used on the platform to seal all the data important to keeping the integrity of the system. When the system is in a known good state, the administrator will run a seal on all the data to be unsealed when the user attempts to log on at a later time.

An example of the seal being run is shown in Figure 7. From the command line, the seal executable is run with three parameters. The first parameter is the TPM version, the second parameter is the PCR register to hold the system state for when the unseal is called. The last parameter is the file to be sealed. This command needs to be run for every file required to establish system integrity. In the figure below, the file being sealed is the biometric device driver.


Applications Places System
root@riggs249dhcp39:/home/tpolon/researchstuff/sha
File Edit View Terminal Tabs Help

```
[root@riggs249dhcp39 sha]# ./Tspi_Data_Seal -v 1.1 -p 8 -d /opt/bioapi/lib/libtfmessbsp.so
Version = 1.1
PCR Number = 8
File To Seal = /opt/bioapi/lib/libtfmessbsp.so
filesize: 937368 bytes
SUCCESS
[root@riggs249dhcp39 sha]# █
```

Figure 7: Example Seal.

The protocol for the login sequence is shown in Figure 8. All communications between the local client and the remote service use 2-way authenticated SSL. Both the local machine and the remote service will contain private keys protected in hardware. When the local machine sends a login request to the remote service, the remote service will return an authentication challenge. The local client will then forward the challenge to the BioAPI, which will request a live sample from the biometric device. However, before the live sample is requested, the TPM will check the biometric driver during the BioAPI Init call. If the biometric driver has been compromised, the TPM will not allow BioAPI to finish initializing and the authentication will halt. If the biometric driver is unaltered, the TPM will allow the authentication to continue and the live sample to be requested. This and subsequent steps involving BioAPI interacting with the BAS and the TPM represent new functionality that is proposed as an enhancement to the current BioAPI or a new layer on top of the current BioAPI. Before taking the sample, and while BioAPI is initializing, the TPM will check the platform to make sure it has not been compromised. If the machine is in an approved state, the TPM will allow the authentication to continue. The TPM will then allow BioAPI to take the live sample, which it will forward to the BAS. The BAS will then check for a match. If a match is found the BAS will send a response for the authentication challenge back to the local client. The local client will send this response to the remote service, where it will be verified. A result is then sent back to the local client, either allowing or refusing a login.

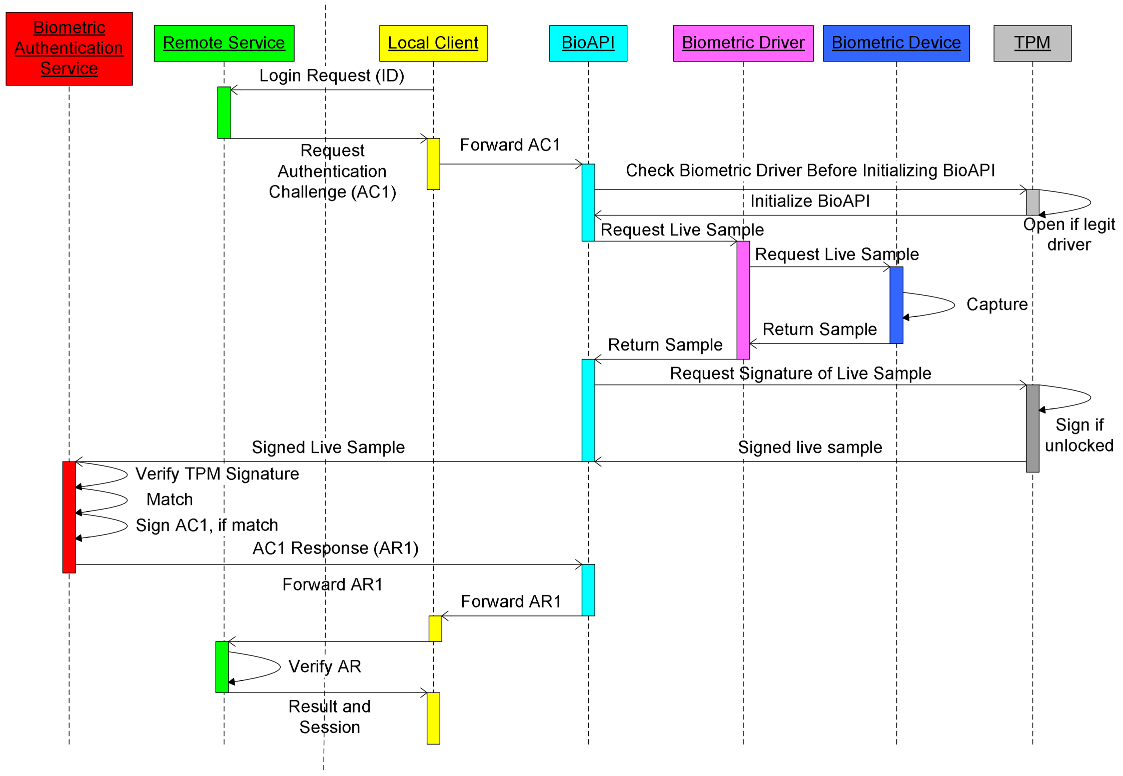


Figure 8: Sequence Diagram For Authentication Protocol.

Below are example pseudo equations for the TPM functions used in this research. The name of the driver as well as the PCR used for this specific research are included in the equations for clarity.

- Seal Operation
    - SealedDataBlob=Kroot(SHA1(libtfmessbsp.so),PCR8)
- Unseal Operation
    - UnsealData=Kroot(SealedDataBlob,PCR8)
    - UnsealData?=SHA1(libtfmessbsp.so)
- Verify Operation
    - CurrentConfig?=PCR8

The equations are explained as follows. For the seal function, the TPM root key (denoted by Kroot) is used to seal a SHA-1 hash of the driver, as well as store the current configuration at the time of seal in PCR-8. This sealed blob is then stored to a file. When the unseal is performed, the TPM will again use the root key to unseal this blob. At the time of unseal, the current value in PCR-8 will be checked against the value at the time of seal. If they do not match the unseal will fail. After the data is unsealed, the unsealed data will be checked against a SHA-1 hash of the current driver. If they differ by even one bit, the operation fails. Finally, during the verify phase of the login sequence, the current client configuration is compared against the reference value to make sure that no tampering has occurred while the biometric sample was being acquired.

3.1 Code Overview

The code written consists of two main files, the seal function and the unseal function. The seal function receives a PCR number to use as well as the file to be sealed. Within the program, all the TPM handles are called and the file is sealed. However, before the seal function is called, a SHA-1 hash of the file is taken. This is done for two reasons: to make the file harder to infiltrate by attackers and to make the file sealable (the seal function has an upper file limit of 150 bytes). After the file is sealed, the data blob is outputted to a file.

The unseal function, for the most part, looks very similar to the seal function. The same TPM handles are called and a file is passed in to compare with the unsealed data. Within the program, the blob is also passed in to be unsealed. After the unseal, the data, which will still be a SHA-1 hash, will be compared to a hash of the file being passed in.

If both files match, the unseal function will return success and the initialize will continue. Otherwise it will return a failure and the initialize will stop and BioAPI will not start.

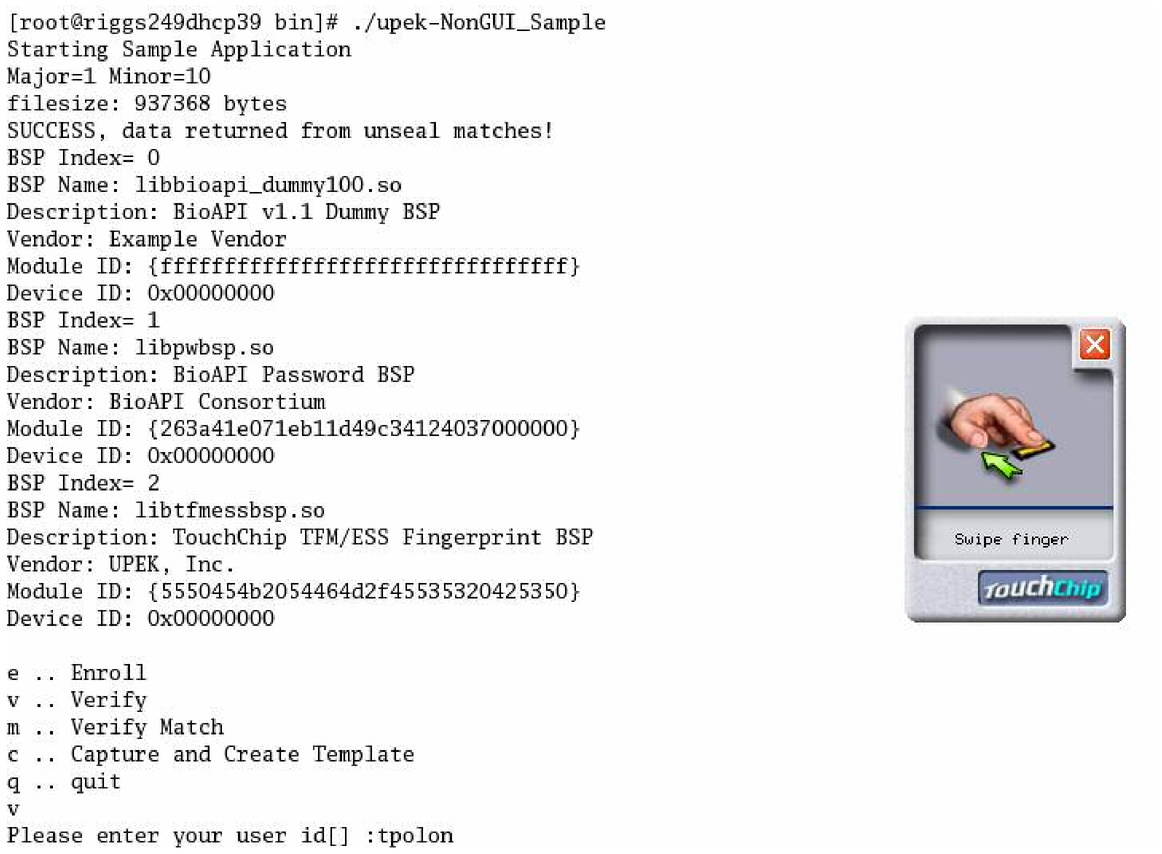
The last written code was a modification to the BioAPI source code. The file modified was the manage_interface.c file. In this file, the unseal function mentioned earlier was added to the BioAPI initialization function. The unseal function is called and depending on the result, the initialization will continue or the API will completely stop loading.

Finally, the original script used to install BioAPI and the driver was modified to copy over the unseal function, the hash function, and a new Makefile in the h_layer folder of the BioAPI source code that includes these functions. This Makefile has been modified to include the unseal function and the hash function as sources, creates object files from them, and adds them to the library.

3.2 Results

The objective of the research system is to prevent most, if not all attacks that do not exploit the capture device. By using an attestation device as well as a challenge/response system between the authentication service and the biometric authentication service, we can assure that most attacks will be prevented. The result is a system that can prevent many more attacks than current systems, while using commodity hardware and not requiring proprietary components.

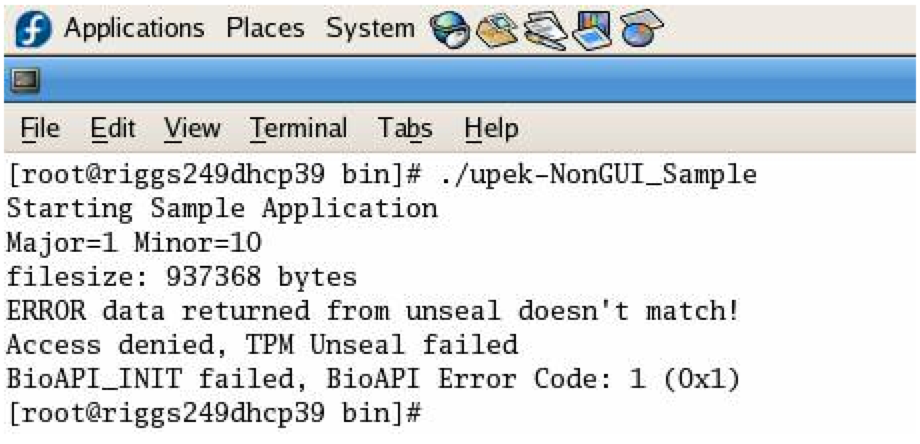
Below are screenshots showing how the system works. When the BioAPI is initialized, it calls a TPM Unseal function which checks all the pertinent files to make sure they have not been modified. For testing purposes, we only used the biometric driver to seal and unseal. In Figure 9, a sample biometric program is run and started without problems. When the biometric driver was modified by only one bit, the same sample biometric program did not start at all, as shown in Figure 10.



```
[root@riggs249dhcp39 bin]# ./upek-NonGUI_Sample
Starting Sample Application
Major=1 Minor=10
filesize: 937368 bytes
SUCCESS, data returned from unseal matches!
BSP Index= 0
BSP Name: libbioapi_dummy100.so
Description: BioAPI v1.1 Dummy BSP
Vendor: Example Vendor
Module ID: {ffffffffffffffffffffffffffffffff}
Device ID: 0x00000000
BSP Index= 1
BSP Name: libpwbsp.so
Description: BioAPI Password BSP
Vendor: BioAPI Consortium
Module ID: {263a41e071eb11d49c34124037000000}
Device ID: 0x00000000
BSP Index= 2
BSP Name: libtfmessbsp.so
Description: TouchChip TFM/ESS Fingerprint BSP
Vendor: UPEK, Inc.
Module ID: {5550454b2054464d2f45535320425350}
Device ID: 0x00000000

e .. Enroll
v .. Verify
m .. Verify Match
c .. Capture and Create Template
q .. quit
v
Please enter your user id[] :tpolon
```

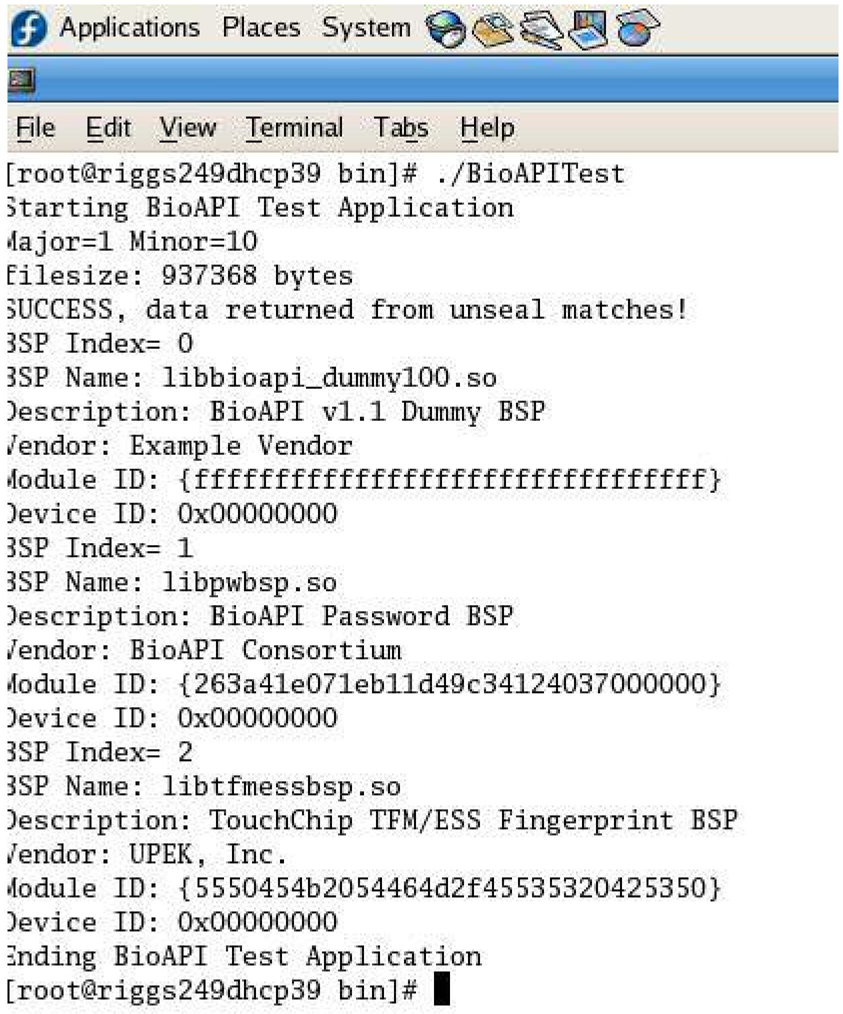
Figure 9: Successful Sample Program Initialization.



```
[root@riggs249dhcp39 bin]# ./upek-NonGUI_Sample
Starting Sample Application
Major=1 Minor=10
filesize: 937368 bytes
ERROR data returned from unseal doesn't match!
Access denied, TPM Unseal failed
BioAPI_INIT failed, BioAPI Error Code: 1 (0x1)
[root@riggs249dhcp39 bin]#
```

Figure 10: Failed Startup By Slightly Modifying The Driver.

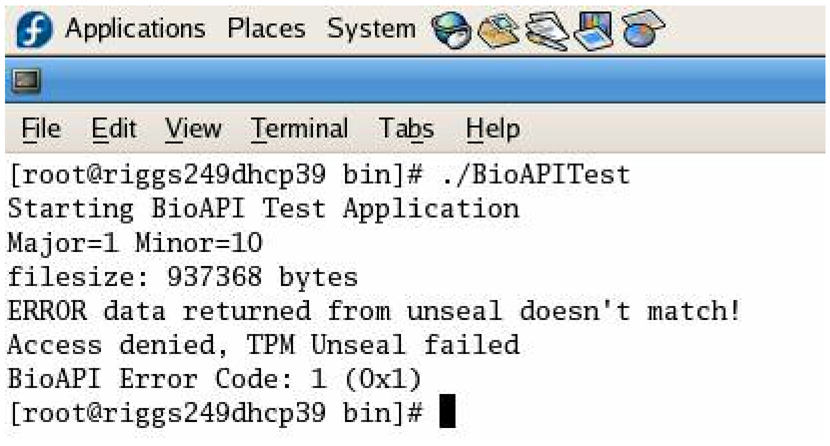
Below, in Figure 11 and Figure 12, a BioAPI test is run on the device to make sure that the device driver is operating and that it is communicating correctly. In the first example, the test is run successfully. However, in Figure 12, it can be observed that even a simple test will fail if the required files have been changed. Like stated earlier, this is because every time BioAPI is called at all, the TPM function will be called during the initialization function and check the known good data against the current data files.



```
[root@riggs249dhcp39 bin]# ./BioAPITest
Starting BioAPI Test Application
Major=1 Minor=10
Filesize: 937368 bytes
SUCCESS, data returned from unseal matches!
BSP Index= 0
BSP Name: libbioapi_dummy100.so
Description: BioAPI v1.1 Dummy BSP
Vendor: Example Vendor
Module ID: {ffffffffffffffffffffffffffffffff}
Device ID: 0x00000000
BSP Index= 1
BSP Name: libpwbsp.so
Description: BioAPI Password BSP
Vendor: BioAPI Consortium
Module ID: {263a41e071eb11d49c34124037000000}
Device ID: 0x00000000
BSP Index= 2
BSP Name: libtfmessbsp.so
Description: TouchChip TFM/ESS Fingerprint BSP
Vendor: UPEK, Inc.
Module ID: {5550454b2054464d2f45535320425350}
Device ID: 0x00000000
Ending BioAPI Test Application
[root@riggs249dhcp39 bin]# █
```

Figure 11: Successful BioAPI Device Test.

This added protection helps to make sure that there are no modified or corrupted files that would compromise the system. From the previous figures, we have proven that any type of change will be picked up before BioAPI even finishes initializing. By doing this so early in the authentication process, the system avoids any type of infiltration by the malicious code.



```
[root@riggs249dhcp39 bin]# ./BioAPITest
Starting BioAPI Test Application
Major=1 Minor=10
filesize: 937368 bytes
ERROR data returned from unseal doesn't match!
Access denied, TPM Unseal failed
BioAPI Error Code: 1 (0x1)
[root@riggs249dhcp39 bin]#
```

Figure 12: Failed Device Test.

However, this extra protection does come at a price. First, in order to use this system, a computer with a trusted platform module already installed would need to be used. While computers outfitted with a TPM are become more and more available, it is not yet a standard and therefore must be specified when purchasing. Cost is not the only hurdle. The time it takes for the TPM to initiate the seal and unseal commands is not instantaneous. In our tests, the unseal function only took a fraction of a second. But what if the administrator wanted to check the integrity of a dozen files, or even more? The unseal process might take several seconds. Ultimately, it is up to the stakeholders to decide whether the added protection is worth the several extra seconds it may take to authenticate the system. If the data is more important than an extra small period of time, then it should not be an issue.

3.3 Threat Assessment

A new attack tree model for the proposed system is shown in Figure 13. The most obvious difference is the separation of machines into approved and rogue. The rogue machine side of the tree is fairly simple. The only way an outside machine can attack the system is through a PKI attack. This relies only on the strength of the cryptography. If a rogue machine is able to gain access, the cryptography will need to be strengthened. However, with a strong enough cryptography, the rogue machine has very little chance of access. Without a signature from the TPM, the BAS will not be able to verify and thus the authentication response will fail. Therefore, it is difficult for a host that is not approved to infiltrate the network.

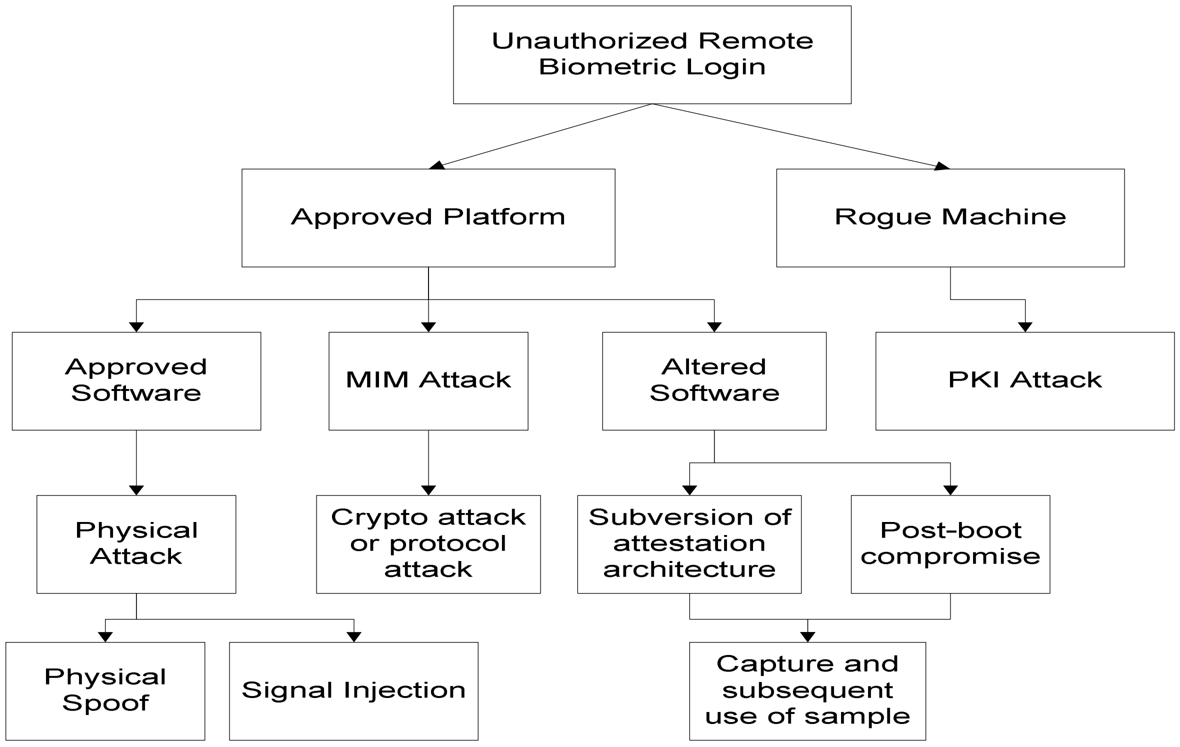


Figure 13: Attack Tree For The Proposed System.

An attack using an approved platform is more involved. While the platform itself can be approved, the device itself might become compromised at some point, as well as the software that is used for authentication (BioAPI, net client, or the device driver), but this would require finding vulnerabilities in these components while still maintaining the appearance of system integrity from the view of the preceding component.

A man in the middle attack (MIM) would have to rely on a cryptographic attack or a protocol attack. As stated earlier, the system will use a 2-way authenticated SSL with private keys. Therefore, it is difficult for a MIM attack to occur with this system.

In the event of software compromise, the attestation mechanism will prevent access in both cases. In one case, the software might be compromised before boot-up. In this case, the software boot chain will fail as the hash check will fail when the compromised software is checked. In the second case, post-boot compromise, the attestation mechanism will check the current state of the software with the hashes held in its registers. If any of the software fails, the attestation mechanism will not sign the live sample and the authentication will fail.

The final case would be the event where nothing has been changed, but instead a physical attack on the device is made. The current system cannot prevent a physical attack on the biometric device. If an attacker was somehow able to physically spoof the biometric on an approved machine running approved software, he or she would be able to gain access. Also, if a device was somehow modified, such as the thermal reader on a fingerprint reader being modified or removed and replaced, there is nothing to prevent an attacker from injecting a signal into the system. However, methods such as additional authentication factors, liveness testing, multiple attempt lockout, passwords, tokens, monitoring, and the use of anti-tamper hardware can be used to prevent such attacks.

3.4 Conclusions

In the beginning of this paper, we explained the potential weaknesses with using a biometric authentication system. We have proposed a solution that will alleviate problems created by a lack of secrets by adding verification to the sample collection process. The research system involves adding an attestation device, a Biometric Authentication Server, and biometric independent software. The system is independent of the biometric technology used and does not require single-source or proprietary products.

3.5 Future Work

Currently, the system only works on a single independent platform. Since the TPM Unseal calls have been added to the source code, adding a BAS would be fairly simple as the source code is already written and just needs to be implemented. Also, the SSL sign on needs to be implemented as well. These additions should not take long to add and while important to the system, a remote biometric server and a secure login are not new research ideas and could possibly even be pulled from open source code elsewhere.

In terms of where the future of a secure biometric login lies, this requires help from the OS developers as well as the biometric manufacturers. For instance, there is no type of real TPM integration within either Windows or Linux. Also, to a lesser extent, the modifications made to BioAPI to include TPM support have not been integrated. Even for this research, a software stack was needed in order to manipulate the TPM through Linux. From a hardware perspective, the next step would be to standardize the software and possibly integrate it within the actual device. By doing this, we can make the path even more secure as the sample can be checked for authenticity before it even leaves the device. When we start working towards all these solutions, and when developers and manufacturers begin to embrace the TPM in conjunction with a biometric device, we can finally produce a full trusted path.

APPENDIX

Software Dependencies

Trousers:

Automake > 1.4
Autoconf > 1.4
Pkgconfig
Libtool
Gtk2-devel
Openssl >= 0.9.7
Openssl-devel >= 0.9.7
Pthreads library (glibc-devel)

Tpm-tools:

Automake
Autoconf
Libtool
Gettext
Gettext-devel
Trousers
Trousers-devel

Installation Instructions

As stated earlier, the platform used for this system is Fedora Core 5. If you are using a different Linux distro, the installation procedure might be different. Please visit http://fedora.redhat.com for more information on how to download a free version of this OS.

To install the BioAPI software and driver for a UPEK fingerprint reader (which was used in this system), the full installation directions can be found at http://www.thinkwiki.org/wiki/How_to_enable_the_fingerprint_reader. To make things easier, if you are running a Fedora Core system, there is a script to completely install everything, including “bioapi_pam”, which adds fingerprint authentication to the Linux PAM (Pluggable Authentication Modules) framework. For this research, the script has been modified to include the TPM functions and the new Makefile described earlier in the code overview section. However, in doing so, the PAM installation will no longer work. This is not a problem though as the PAM framework is not required for the purpose of this research. This script can be downloaded at the site listed below. Again, this can only be used if you are using a Fedora Core distro.

When installing the system on a new machine, the TPM code, modified BioAPI file, and modified Makefile must be placed in the directory used in the script in order for it to work properly. Alternatively, this code may be placed in another location so long as the script is modified to look for the files in said location.

The installation of the TPM is a little trickier. When installing Fedora Core, I did not see an option to enable the TPM in Linux. Therefore, after the system was installed I downloaded the kernel. After extracting, “make menuconfig” must be run. Within the menu, module support for the TPM must be enabled. After doing so, the modules need to be copied by using “make modules_install” and the kernel needs to be compiled.

When the new kernel is compiled with TPM support, Trousers can be installed. Trousers is used to interact with the TPM and is therefore required. Before installing Trousers, make sure all the dependencies are already installed on the system. The actual installation of Trousers is very straightforward and is included in a README file with the source code. The same goes for tpm-tools, an extension of Trousers, which also needs to be installed. When all these files are correctly installed, the computer is ready for TPM use.

Software

Fedora Core:	http://fedora.redhat.com/download/mirrors.html
BioAPI:	http://www.bioapi.org/
Trousers:	http://trousers.sourceforge.net/

Code Written

Makefile: http://www.clemson.edu/~tpolon/Makefile

Tspi_Data_Seal: http://www.clemson.edu/~tpolon/Tspi_Data_Seal.c

Tspi_Data_Unseal: http://www.clemson.edu/~tpolon/Tspi_Data_Unseal.c

Manage_Interface: http://www.clemson.edu/~tpolon/manage_interface.c

Installation Script: http://www.clemson.edu/~tpolon/enable-fingerprint-reader.sh

REFERENCES

[1] Anthony Allan, “Security special report: Fingertip security” ComputerWeekly.com. April 26, 2006.

[2] BioAPI Specification, Version 1.1, March 16, 2001. http://www.bioapi.org/New%20Downloads%20(Add%20to%20Site)/BioAPI%201.1.pdf

[3] Liqun Chen, Siani Pearson, Athanasios Vamvakas “A Trusted Biometric System” HP Laboratories Technical Report HPL-2002-185. July 15th, 2002. http://www.hpl.hp.com/techreports/2002/HPL-2002-185.pdf

[4] A. Jain, A. Ross, and U. Uludag, “Biometric Template Security: Challenges and Solutions,” http://biometrics.cse.msu.edu/Publications/SecureBiometrics/JainRossUludag_TemplateSecurity_EUSIPCO05.pdf

[5] T. Matsumoto, H. Matsumoto, K. Yamada, S. Hoshino, “Impact of Artificial ‘Gummy’ Fingers on Fingerprint Systems”, *Proceedings of SPIE*, vol. 4677, January, 2002

[6] R. Morris and K. Thompson, "*Password Security: A Case History*", Communications of the ACM, Vol.22, No.11, November, 1979, pp.594-597.

[7] N. K. Ratha, J. H. Connell, R. M. Bolle, “A biometrics-based secure authentication system,” http://www.research.ibm.com/ecvg/pubs/ratha-chall.pdf

[8] D. Safford and M. Zohar, “A Trusted Linux Client (TLC),” http://www.research.ibm.com/gsal/tcpa/tlc.pdf

[9] R. Sailer, L. Van Doorn, and J.P. Ward, “IBM Research Report, The Role of TPM in Enterprise Security,” https://www.trustedcomputinggroup.org/news/articles/rc23363.pdf

[10] S. Sun, C Lu, and P. Chang, “Biometric Template Protection: A Key-Mixed Template Approach,” http://vaplab.ee.ncu.edu.tw/english/pcchang/pdf/c75.pdf

[11] S. Sutcu, Q Li, and N Memon, “Protecting Biometric Templates with Sketch: Theory and Practice,” http://isis.poly.edu/~qiming/publications/tifs07.pdf

[12] S. Sutcu, H. T. Sencar, and N. Memon, "A Geometric Transformation to Protect Minutiae-Based Fingerprint Templates," http://isis.poly.edu/~biomet/yagiz07geometric.pdf

[13] Trousers, the open-source TCG Software Stack. http://trousers.sourceforge.net

[14] Trusted Computing Platform Alliance (TCPA), Main Specification Version 1.1b, (2003). https://www.trustedcomputinggroup.org/specs/TPM/TCPA_Main_TCG_Architecture_v1_1b.pdf

[15] Trusted Computing Group, TPM v1.2 Specification Changes, (2003). https://www.trustedcomputinggroup.org/groups/tpm/TPM_1_2_Changes_final.pdf

[16] U. Uludag, S. Pankanti, S. Prabhakar, and A. K. Jain, "Biometric Cryptosystems: Issues and Challenges," *Proceedings of the IEEE*, vol. 92, no. 6, pp. 948-960, June 2004.