System and Data Security - A CS Systems Prelim

Andy Sayler
University of Colorado Boulder, Colorado
andy.sayler@colorado.edu

ABSTRACT

Security is a core component of modern computer systems. From protecting our data to securing our communications, security across the computing spectrum is fundamental to the manner in which we leverage and trust computers. But the security of modern computing systems has not come easily: it has been improved slowly over many years, sometimes at the cost of painful lessons. Furthermore, the modern state of the art in computing security still leaves much to be desired. In this paper, I explore the development of the current state of the art in computer security focusing on four core components: cryptography, access control, file systems, and usability. Computer security is an inherently large topic, but these core topics provide a reasonable basis on which a discussion the modern state of computer security can be built. In particular, I seek to answer the question: "How can we secure our systems and data in a robust, comprehensive, and easy-to-use manner?". This question in examined from a historical perspective, as well as the perspective of a modern user with modern use cases. This paper builds on the background work completed in my Master's Thesis [12], further extending my analysis of the current state of the art and hypothesizing on future extensions to this state.

1. INTRODUCTION

We have reached the age of ubiquitous computing. There is not a facet of our lives that is not heavily integrated with the vast computing networks we have built and continue to expand. From the cell phones we use to communicate to the web sites on which we mange our life's savings to the hard drives filled with our photos and personal documents, computers are not only everywhere, but often at the heart of our most intimate interactions. As such, the security of our computing infrastructure is a foremost concern in modern computing system design. But what is the state of the art in computing security? And how have we arrived at this state? These are question I address in this paper through the exploration and analysis of ten significant publications in the field. In particular, I break my analysis of the state of the art in computer security into four related topics: cryptography, access control, file storage, and usability. These topics provide the underpinnings of the bulk of the modern state of the art of computer security.

On the topic of cryptography (Section 2), I present the basics of modern asymmetric cryptographic systems [3], extensions to these systems to accommodate the diversification of trust [13], and the manner in which these core concepts can be leverage in access control applications [1]. On the topic of access control (Section 3), I present the basics of modern access control models [11], the ways in which these models can incorporate cryptography to avoid the need for a trusted compute base [1], and the manner in which various access control schemes have been applied to modern file systems [7]. On the topic of file systems (Section 4), I present an effort to support file system distribution with minimal trust [6], an overview of the security mechanism employed by a range of modern file system [5], and the manners in which modern file system implement access control [7]. Finally, on the topic of usability (Section 5), I present an early effort to standardize the basic system security primitives [10], techniques for making security more robust and simpler for the end user [2], and efforts to unify security primitives across multiple administrative domains [8]. These ten papers are by no means the complete body of prior art, but they do elucidate the core concepts relevant to the question at the core of this paper. References to other relevant works will be provided where appropriate, but the bulk of my analyses will focus on the papers listed above.

In addition to exploring the historic contributions of the work mentioned above and the current state of the art they represent, I also suggest possible future expansions of this state. At it's core, this involves looking at the existing answers to the question question "How can we secure our systems and data in a robust, comprehensive, and easy-to-use manner?", as well as proposing potential new answers. This hypothesizing is addressed with respect to the four core topics mentioned above within each of the relevant sections and is intended to inform potential future research paths and projects.

2. CRYPTOGRAPHY

Cryptography is the basis of much of the modern computer security landscape. This is largely because it represent a security primitive that does not rely on trusting specific people, platforms, or systems in order to securely function. Instead, it requires that we place our trust in only one thing:

the underlying math. This has led to the proliferation of cryptography as the security primitive on which many other security features are built.

2.1 History

Cryptography has a very long history: there is evidence of societies employing basic cryptographic systems in order to "secure" writing and messages dating back to thousands of years BCE. These early forays into cryptography, however, lacked the sound grounding in mathematical theory that makes cryptography so appealing today. I will thus skip over the bulk of cryptographic history involving them.

Modern cryptography has its roots in the field of information theory that begin to develop during WWII and advanced quickly in the post war years. Much of information theory laid the basis for our ability to prove that a given cryptographic algorithm requires a certain amount of effort to crack in the absence of the "key". This led to the rise of mathematically grounded symmetric encryption algorithms, designed for use with the growing availability of computers, by the early 1970s.

Symmetric cryptography algorithms function on the principle that a single "key" is used to both encrypt and decrypt a message. This key must be securely stored, or if shared, securely exchanged between parties. Anyone with the key can decrypt the corresponding ciphertext the key was used to create. The security of a symmetric encryption cipher tends to be directly related to the length of the encryption key: the longer the key, the more secure the data encrypted with it is.

While symmetric cryptography algorithms are useful in situations where a single actor will be both encrypting and decrypting a piece of data (and thus can hold the required key personally), they pose a major challenge it situations where multiple parties wish to communicate securely. In this situation, the parties must find a way to securely communicate the required symmetric key. In the absence of cryptographic methods, the only way to securely exchange a key while avoiding both eavesdroppers and interlopers is to meet in person and exchange the key manually. The tediousness and lack of practicality of this task, especially in modern digital communication systems where multiple actors may be continents apart, led researchers to seek a better method for secure data exchange in the absence of an inherently secure communication channel.

The major breakthrough in solving this challenge came in 1976 with Diffe and Hellman's publication of "New Directions in Cryptography" [3]. Diffe and Hellman proposed a system for asymmetric cryptography: a cryptography system in which one key is used for encryption while a second related key is used for decryption. When properly designed, it is computationally unfeasible to derive one of the keys in an asymmetric cryptography system form the other, allowing a user to publish one of their keys for the public to consume while keeping their other key private. A member of the public can then use the user's public key to encrypt a message that only the holder of the private key will be able to decrypt. If all members of the public maintain such public/private key pairs, it becomes possible for any user to send any other user a message that only the recipient an read without requiring any form of secure communication channel.

Asymmetric cryptography relies on the existence of "trapdoor" functions in order to operate. These functions can be quickly solved in one direction, but are computationally difficult to reverse without a special piece of information (e.g. the 'key'). Factoring large numbers is a classic example of a trapdoor function (and the method on which many modern public key encryption systems are based). Factoring large numbers is computationally difficult in cases where some piece of secret information (e.g. one of the factors) is not known.

Diffie and Hellman proposed a potential implementation of a public key cryptography system, although the first practical public key crypto system came a few years latter with the invention of the RSA [9] algorithm. In addition to public/private key systems, Diffie and Hellman also proposed a system for joint key generation where two parties can negotiate a secrete key across an insecure connection. Like asymmetric cryptography, such a system can be used to bootstrap secure communications across an insecure connection by allowing two parties to derive a secret key that can then be used to facilitate further secret communication using a symmetric encryption algorithm.

Diffie and Hellman also introduce the concept that asymmetric encryption can be used to build the two additional core cryptographic primitives we have come to rely upon: cryptographic verification and cryptographic authentication. Cryptographic verification (also called a cryptographic "signature") is essentially the reverse of asymmetric encryption: instead of a member of the public using another party's public key to encrypt a message that only the target party can read, the target party uses their private key to encrypt a message that the public can then decrypt using the target's public key. Since only the target has access to the private key, and is thus capable of generating such a message, the target can "prove" that a given message comes from them and that it has not been altered in transit. Just as asymmetric encryption gives rise to cryptographically secure signatures, cryptographically secure signatures can give rise to cryptographically secure authentication. If a user generates a signed message saying "I am John" and sends it to an authentication server, the server can verify that the message signature is valid by checking it using John's public key, and thus authenticating John in the process. The server need only have a list of public keys for each user. It can then leverage the assertion that only the indented user has access to the corresponding private key for each of the server's public keys, and is thus the only one capable of generating a signed message on the user's behalf, as the basis of user authentication.

Beyond the rise of public key cryptography, one of the other major cryptographic breakthroughs of the last fifty years was the invention of cryptographically secure secret sharing schemes. In particular, Adi Shamir (the 'S' form "RSA") proposed a practical and robust secret sharing scheme in his 1979 paper "How to share a secret" [13]. In this work, Shamir lays out the basics of what has come to be known as Shamir Secret Sharing: a method for splitting a piece of information up into two or more pieces in a manner such that holders of any subset of the pieces cannot infer any information about the pieces they do not hold or the original information block as a whole. Shamir Secret Sharing allows a user to divide a piece of D data into N pieces of which K or more pieces can be used to recompute the original value of D. A user with fewer than K pieces, however, has no more information about the value of D than a user with no pieces. This system provides a highly useful method for distributing information amongst multiple parties or systems in situations where no single party or system can be fully trusted.

Shamir Secret Sharing, unlike all known asymmetric encryption techniques, does not rely on computational complexity as the basis of its security. Instead, it is fundamentally secure based on information theory principles. Thus, unlike computationally secure systems such as RSA, Shamir Secret Sharing can not be broken regardless of the amount of computational power one posses. Shamir Secret Sharing functions on the basis of defining a polynomial of degree (K-1) over a finite field with the D data encoded as the first order-zero term. N points are then selected from this polynomial and distributed to the participants. Since K points (but no fewer) will uniquely identify the original polynomial, and thus allow the derivation of D, K users must combine their pieces in order to re-compute D.

Shamir Secret Sharing (and related systems) are useful in a wide range of situations where one needs to distribute trust across multiple entities. In particular, secret sharing techniques are leveraged in some cryptographically-based access control systems like that described in [4]. Such systems will be discussed further in Section 3.

2.2 State of the Art

Both symmetric and asymmetric encryption systems have a place in the modern security landscape: symmetric systems for their performance and resistance to cryptanalysis and asymmetric systems for their avoidance of the key exchange problem. Often symmetric and asymmetric cryptography are used together, each system playing to its strength. Symmetric systems are good at quickly and securely encryption data, making them appropriate for the core of an encryption system. Symmetric ciphers, however, suffer from a lack of natively secure method for exchanging the required encryption key. This is where asymmetric cryptography and related secure key exchange systems come in handy. These systems provide the basis for securely exchanging data over insecure channels and they can be thus used to bootstrap symmetric encryption systems by facilitating the secure exchange of a symmetric encryption key which can then be used to encrypt the underlying data. Such systems are common in many modern protocols like SSL, TLS, PGP, and SSH.

Modern symmetric encryption ciphers like AES (Rijndael), Twofish, or Camellia are well-established, fast, and secure methods for encrypting data. Symmetric encryption systems are the preferred means of encrypting files, hard disks, and other large chunks of data due to their speed and relative simplicity of implementation. They are also useful for encrypting streams of data in communication protocols like TLS/SSL or SSH. They tend to be well understood, and are generally considered highly secure (at least the well vetted ones). The strength of a given symmetric cipher is directly related to the length of the associated encryption key. Common key lengths generally considered secure today include 128-bit keys, 256-bit keys, 384-bit keys, and 512-bit keys.

Modern asymmetric encryption systems include systems like RSA, ElGamal, ECDH, ECDSA, or ECIES. These systems are all built atop various one-way trapdoor functions. RSA is based on prime-factorization functions, ElGamal is based on discrete logarithm functions, and ECDH, ECDSA, and ECIES are all based on various elliptic curve functions. It is often useful to mix cipher suites relying on different trapdoor functions as a hedge against an efficient solution to any of the underlying function families being discovered, thus rendering the encryption using such a family obsolete. Within a given family the security of an asymmetric cipher is related to the length of the keys in a key pair: longer keys are more secure. Standard key lengths for asymmetric keys depend of the family of functions be used. In prime-factorization based system like RSA or discreet-log systems like ElGamal, key lengths of 1024-bits, 2048-bits, and 4096-bits are all common (with 2048 considered to be best practice for general data and 4096 considered to be best practice for highly sensitive data). Elliptic curve based systems tend to use shorter keys: recommended sizes range from 160-bits to 512-bits (similar to symmetric key lengths).

2.3 Future Extensions

While both symmetric and asymmetric cryptography are largely settled art at this point, there are a variety of challenges yet to be solved in the cryptography research realm. One of the constant threats to the settled cryptographic art is that of a major breakthrough in solving the currently "hard" problems used as the basis for a variety of trapdoor functions. One of the proposed methods for solving existing trapdoor functions more quickly than currently possible is to use quantum computing approaches. While no practical attacks on existing cryptographic systems using quantum approaches are known, there are certainly researchers working on such approaches. There is also work being done in the opposite direction to try to leverage quantum techniques to build cryptographic systems that would be effectively unbreakable due to the underlying quantum limitations of our universe. Such techniques often involve leveraging the quantum "observer" effect to detect eavesdropping on a communication channel, allowing the users to regenerate keys until they complete the negotiation unobserved. Again, however, there are no known practical implementations of quantum cryptography at this time.

One of the other major challenges to cryptography today is the reliance on good sources of randomness to generate the secure cryptographic keys used in both symmetric and asymmetric encryption systems. Finding and harnessing good randomness is challenging since it depends on having access to an sufficient amount of entropy. Traditionally, computing systems leverage user interactions (e.g. mouse movements, key strokes, network interrupts, etc) as the sources for OS-maintained entropy pools. The rise of cloud computing systems, however, posses challenges to using such sources: in the cloud, systems often run in highly homogeneous environments with few "random" user interactions. Coming up with secure ways to derive entropy and provide good randomness is such environments is a topic of active study. The importance of randomness to modern computing systems requires us to come up with secure ways of gathering and distributing entropy. Potential solutions range from building secure Entropy-as-a-Service systems that gather entropy from sources where it is abundant and redistribute it to source where it is scarce to using entropy-generating hardware that relies on unpredictable physical phenomena (e.g. atomic decay events) to derive randomness.

Finally, while strong cryptographic systems are well understood, there are a myriad of situations where cryptographic best practices are ignored or where cryptographic systems are unpractical to use, leading to security failings. Many of these issues can be linked to usability challenges within cryptographic systems: if a system is difficult to use or challenging to deploy, it will often go unused or wind up misconfigured and insecure. One of the major usability challenges present in all current cryptography systems is the management of private/secret keys. Keeping such keys secure while also enabling their use across a range of modern multi-user, multi-device, ephemeral resource use cases is extremely challenging and lacks a standard solution at this time. My previous master's thesis work, Custos [12], proposes a potential solution to the secret management problem and remains an area of active research. Other approaches to alleviating usability issues are discussed in Section 5.

3. ACCESS CONTROL

Over the years, we have developed a range of access control techniques. All of these techniques share a common goal: controlling access to a specific system, resource, or piece of data. Must access control models have two key components: authentication and authorization. Authentication is used to establish the identity of an actor. Authorization then leverages this identification as the basis of granting or denying specific permissions to the actor. This section will discuss historic access control techniques, the current state of the access control art, and potential future access control additions.

3.1 History

Computer-based access control systems have been with us since the earliest multi-user (e.g. time sharing) operating systems became popular in the 1970s and 1980s. Early access control systems were primarily focused around the Unix model of access control: users, groups, and read/write/execute file-level permissions. Authentication in these early systems was generally limited to username:password combinations, the mechanisms of which were hard coded into the `login` program. Each user was a member of one or more groups and each file had a owner and group. The three file permissions, read, write, and execute, were granted on the basis of a user's relationship to a given file: either the user was the file owner, the user was a member of the file group, or the user was neither of these things. This model is fairly flexible, and continues to be used today as the core access control model in many Unix-like operating systems (e.g. BSD, Linux, OSX, etc).

Access Control List (ACL) based schemes gained prominence in 1990s and were popularized by the Windows NT family of operating systems. ACLs extend the permission model beyond the basic Unix file permissions to include a wider range of file (e.g. read, write, delete, create, etc) and system-level (e.g. shutdown, connect to network, etc) permissions. ACLs are associated with specific system objects (e.g. files, folders, OS subsystems, etc) and map a user or group to a list of permission that user or group possess. They generalize the Unix access model to accommodate a wider range of permissions and mappings between permissions and actors. ACL-based systems have been integrated into many modern Unix-like operating systems as an optional extension beyond the tradition Unix permissioning scheme.

Exiting access control schemes are often grouped into one of two classes: Mandatory Access Control (MAC) systems or Discretionary Access Control (DAC) Systems. While the lines between these two approaches are occasionally blurred, the basic difference between the two lies in which actors within a system have the ability to grant/extend permissions to other actors. In MAC system, all permissions are set by the system administrator and users have no ability to change these permissions themselves or transfer permissions to other users. DAC systems, in contrast, give users the ability to set their own permissions on objects they own or create, and to transfer these permissions to other users. A MAC-based system can be thought of as similar to a DAC system where the system administrator owns all files and never transfer this ownership to any other user. Traditional Unix access control systems as well ACL access control systems can generally be used in either MAC or DAC based systems. MAC systems are generally preferred in high security environments where the centralized management models they offer lead to tighter control over data. DAC systems are more common in general purpose systems where the extra flexibility they offer reduces the administrative burden. Most Unix-like systems are DAC systems by design, but extensions (e.g. SELinux) can be used to add MAC properties to these systems.

Many of the early access control systems pose a host of manageability challenges. How do you coordinate the permissions of thousands of users across millions of objects? How do you revoke permissions for a defunct user? Or add a new user? To cope with many of these challenges Sandhu, et. al. proposed the concept of Role-Based Access Control Models in their 1996 paper by the same name [11]. Role-Based Access Control (RBAC) inserts an additional layer of indirection between users and permissions. In an RBAC system, users are assigned to one or more roles. Each role is then assigned one or more permissions. This model simplifies management by separating permission assignment from specific users. RBAC permissions are assigned assigned on the basis of specific positions or duties within an organization and mapped to specific roles. Users are then assigned to these roles on the basis of whether or not they hold a specific positions or are required to perform a specific duty. Thus, adding or removing users does not require any modification to permission mappings, only role mappings. Likewise, adding or removing permissions does not require modifying user mappings, only role mappings. [11] describe 4 classes of RBAC systems: $RBAC_0$ (the base model), $RBAC_1$ (the base model with the addition of role hierarchies and inheritance), $RBAC_2$ (the base model with the addition of constraints), and $RBAC_3$ (the combination of $RBAC_1$ and $RBAC_2$).

The primary limitation of all of the access control models mentioned thus far is their reliance on a trusted arbiter for enforcement: generally this trusted arbiter is the operating system or some other underlying program in charge of enforcing the access control system. This means that the security of any of these access control systems is only as good as the security of the system enforcing them (e.g. the security of the OS). Thus, if the underlying OS or program is compromised, the access control system falls apart. Likewise, anyone in control of the underlying OS or program (e.g. an administrator) automatically gains full control over the access control system. This is an acceptable limitations in many situations, especially those based on a centrally managed system with existing physical security and administrative safeguards in place. But in distributed systems or other systems where physical and management control in not guaranteed (e.g. the cloud), a more robust system that lacks this "trusted arbiter" requirement is desirable.

To overcome the need for a trusted enforcement mechanism in access control systems, researchers have turned to cryptographically-based access control systems. As mentioned in Section 2, cryptographic-based system require no trust in external systems, only in the underlying math. Sahaie, Waters, et. al. propose several cryptographically-based access control systems in their 2006 paper [4] and its 2007 follow-up [1]. These two systems are based on the concept of Attribute-Based Encryption (ABE) schemes. ABE schemes allow a user to encrypt a document in a manner such that the access control rules associated with the document are part of the encryption process itself. Thus, in order to decrypt/access a document, a user must satisfy one or more cryptographically guaranteed access control attributes. [4] allows user to encrypt documents that can only be decrypted by users possessing specific attribute polices encoded in their keys. [1] extended this concept to allow documents to be encrypted with a full access control policy tree embedded in the encryption processed file directly allowing only users who's private keys meet a generalized set of requirements to access the documents. Both these systems allow the construction of access control systems that do not require any trusted arbiter to regulate access to objects. Instead, the regulation in enforced by the underlying encryption backed by the associated math.

3.2 State of the Art

Today, the state of the are in access control differs widely based on the system and application. As mentioned, many Linux and Unix-like systems still use basic Unix access control primitives largely unchanged over the last 20 to 30 years. This is largely due to the fact that these systems are "good enough" for many single-user and small group environments and the administrative burden of shifting to a more advanced system is simply not worth the effort. Most Linux systems today do support extended file ACLs, as well as system-level ACLs exposed by systems like PolicyKit. These systems allow users to move beyond the limitations of a pure Unix-like, file-centric access control scheme. That said, many users never need to deal with these more advanced systems for the majority of day-to-day use cases.

Windows-based operating systems make extensive use of ACL-based access control schemes. While, these schemes are useful in the Windows-domain environments used by most large corporations, many home and leisure Windows users never have any need to interact with file or system level ACLs. Modern Windows systems combine RBAC concepts with ACL concepts by allowing administrators to define role-based "groups" that can then be used with specific permission assignments. Most stand-alone access control systems bundled with specific applications (e.g. content management systems, etc) also take cues from both RBAC and ACL access control models. While these system do help to reduce the management burden of large systems, they are often prone to misconfiguration, the occurrences of which lead to many of the security breaches that happen today.

Cryptographically-based access control schemes remain largely an academic novelty at this point. I am aware of no commercially or generally deployed software that leverage ABE or similar cryptographically-based access control schemes as the basis of their access control models. None the less, the need for robust access control schemes that can be used across a range of untrusted infrastructure is only going to increase in our modern cloud-based computing world. Thus, it is possible that we will see an increase in the practical deployment of these systems in the near future.

Outside of permission-side access control models, there have also been advances in the authentication side of access control. Most modern access control systems support authorization primitives far more complex than basic username:password combinations. Several such systems are discussed in more detail in Section 5. None the less, the vast majority of user authorization systems are still password based. To overcome the well known security deficiencies of user passwords, multi-factor authentication schemes are staring to gain prominence on high value target systems (e.g. email accounts, bank accounts), but such systems are most often optional and are not in wide use amongst the general population.

3.3 Future Extensions

One of the major limitations to most access control system today is their lack of global name space or rules: access control rules are currently scoped to the system or administrative domain in which they are created, with little to no support for wider, globally-enforceable rules. This creates a host of challenges implementing robust access control schemes across our modern multi-user, multi-device, loosely managed environments. As a basic example, traditional Unix file permissions are useless when used on portable media like USB flash drives since user IDs, groups, and permissions are all scoped to the local machine and do not extend to other machines on which you might wish to access the portable media. Cryptographically-based access control schemes like ABE are a step toward solving this problem by removing the need for a trusted host system. Still, such systems still pose many of the same challenges other cryptographic systems have: namely how to manage and control access to the underlying private keys in a secure yet globally accessible manner. In addition, a variety of system-specific distributed access control schemes will be discussed in the context of the file systems to which they apply in Section 4. But few of these systems are generalized enough for use on a multi-application scale. Providing secure, manageable, and usable access control systems that can operate on a global scale across a variety of distributed devices largely remains an open problem, and finding a solution will continue to increase in importance as our use cases continue to demand an increasingly global and distributed perspective.

Beyond globally usable access control schemes, most access control system today have a fairly clear delineation of authentication and authorization. While this division makes since from a separation of duties standpoint and fits well into tradition access control schemes, it can also limit the expressiveness of an access control system. For example, strictly separating authentication and authorization makes it difficult to set up access control rules that depend on more than a single user's identity (e.g. time dependent, etc) and also make it difficult to operate in situations where the concept of a single "user" is not well defined (e.g. anonymous systems). In [12], I explored relaxing the separation between authentication and authorization to build a more expressive access control scheme. Schemes like ABE also blur the lines between authentication and authorization in the name of increased expressiveness. How to strike a correct balance between a proper separation of authentication and authorization duties and a high level of expressiveness remains an open problem. Building expressive systems that are also easy to reason about, manage, and maintain is a relevant topic to future access control advances.

4. FILE SYSTEMS

Much of the work we perform on computers today is highly data-centric. As such, the protection and control of our data is a core goal within the realm of computer security. The previous two sections explored ways to protect data cryptographically and via various generalized access control models. In this section, I'll look at data protection schemes built into storage systems directly.

4.1 History

Early storage and file system technologies often simply neglected security, lacking robust encryption and access control primitives. As mentioned in Section 3, the rise of multi-user operating systems like Unix mandated the creation of basic file-system access control schemes. Thus we gained the previously mentioned Unix file access control and permissioning scheme as part of the virtual file system (VFS) abstraction inherent in all legacy and modern Unix-like operating systems. As previously stated, however, this system has a number of limitations: it supports only a single, basic access control model (owner, group, R/W/E permissions), it requires a trusted system for enforcement, and it is strongly coupled to a local system. Systems like NFS attempt to extend Unix file security semantics beyond the local machine allowing remote sharing of files, but even these systems are limited to singular administrative domains and trusted systems.

The Windows NT file system access control model (implemented via the NTFS file system) extends the flexibility of the traditional Unix model by adding support for more expressive ACLs. These both allow the control of additional permissions (e.g. delete, create directory, etc) as well as more expressive user to permission mappings beyond the basic owner/group/other Unix model. Furthermore, the Windows NT model has the ability to delegate user authentication to a local Domain Controller (DC) capable of centrally managing all users from a single location. This expands the ability to control file access beyond the users associated with the local system to the users associated with an entire administrative domain. Still, this system still has many of the same limitations as the Unix model: the requirement for a trusted system for enforcement and the tight coupling to the local administrative domain.

The rise of the Internet as a reliable and high speed system for connecting multiple machines across the world as well as the move toward cloud computing models where computational resources are outsourced to dedicated providers has increased the demand for secure storage systems capable of spanning multiple systems and domains. In order to overcome the limitations posed by traditional file system security models and accommodate modern multi-user, multi-system use cases, researchers have proposed a number of newer systems. These systems try to address one or more of the limitations mentioned above. Some of them employ cryptographic security models to overcome the need for a trusted enforcement system. Others are designed to extend access control semantics beyond the local machine to large networks or even the global internet. Still others explore the use of novel access control models more expressive then Unix permissions or Windows NT ACLs. Many system combine more than one of these approaches to build a fully featured next generation secure storage system.

Kher and Kim provide a survey of the various approaches to securing distributed storage systems in their 2005 paper "Securing Distributed Storage: Challenges, Techniques, and Systems" [5]. In it, they discuss the security models of various storage systems, sorting such systems into basic networked file systems, single-user cryptographic file systems, and multi-user cryptographic file systems. As previously mentioned, basic networked file systems rely on trusted systems and administrators for the enforcement of security rules. Examples of such systems include the Sun Network File System (NFS), the Andrew File Systems (AFS), and the Common Internet File System (CIFS/SMB). All of these systems are designed for use within local administrative domains and do not scale well to global, loosely-coupled distributed systems. To deal with the scalability issues, researchers have built system like SFS (discussed below) or OceanStore which aim to reduce the administrative burden of large scale distributed file systems. All of these systems, however, rely on some degree of system or administrator trust. In order to accommodate situations where users do not wish to place trust on the underlying system or remote servers, Kher and Kim discuss a handful of cryptographically-secure file systems. The best of these systems offer end-to-end cryptography, meaning that data is encrypted and decrypted on the client side and the server never has access to the unencrypted data. Systems like the Cryptographic File Systems (CFS) provide basic single-user end-to-end file encryption. While end-to-end encryption is a powerful security model for enabling secure storage atop untrusted systems, it does pose challenges with respect to multi-user, multi-device use cases since it generally requires that all clients have access to private cryptographic credentials in order to effectively read or write files. In order to support both end-to-end encryption and multi-user scenarios, researchers have proposed multi-user cryptographic storage systems like SiRiUS, Cephus, or Plutus.

Miltchev, et. al. provide a survey of access control techniques across a variety of distributed file systems in their 2008 paper "Decentralized Access Control in Distributed File Systems" [7]. They present a framework for analyzing the suitability of various distributed file systems for modern multi-user, multi-domain use cases by analyzing five underlying file system qualities: authentication, authorization, granularity, delegation, and revocation. Miltchev, et. al. suggest that any secure large scale file system must successfully address the functionality of all five of these factors across multiple administrative domains in order to be an effective multi-user, multi-domain file system. In addition to the systems discussed in [5], Miltchev, et. al. also discuss systems like Truffles, Bayou, WebFS, CapaFS, DisCFS, WebDAVA, and Fileteller as examples of systems that attempt to support multi-domain, multi-user, globally-accessible use cases. Miltchev, et. al. reach the following conclusions regrading successful secure multi-user, multi-domain file systems: the use of public-key cryptography for user authentication is an effective way to support autonomous delegation, capability-based access control systems tend to lack support for auditing and accountability, ACL-based access control systems pose scalability challenges when used across administrative domains, and revocation and user autonomy are often at odds.

A good example of a modern multi-user distributed file system with many of the desirable qualities discussed previously is SFS, described by Mazières, et. al. in their 1999 paper "Separating Key Management form File System Security" [6]. In this paper, they describe the design and implementation of the SFS file system. SFS is unique in that it is designed for global-scale file system operations and leverages basic cryptographic security while avoiding the need for tightly specified key management infrastructure. Such infrastructure often adds a large administrative burden, limiting file system expansion beyond a single administrative domain on the basis of management complexity alone. SFS achieves its goals through the use of self-certifying file names: file names that encode the public key of remote a file server into the file path itself. Self-certifying path names allow SFS clients to bootstrap a cryptographically secure communication channel to any remote server without requiring any large scale key-management infrastructure. SFS leverages authentication servers to verify users on the basis of asymmetric key pairs and uses security agents to help reduce the usability burden this might otherwise impose (see Section 5 for more details on agents). SFS is built atop the NFS distributed file system, and thus uses an a Unix-based access control model to regulate file and directory access. SFS's cryptographic capabilities help it to overcome the need for a fully trusted compute base for the enforcement of its security model. Extensions to SFS like GSFS help further move SFS beyond the limitations of a single administrative domain to accommodate a more general global file system. SFS does not however provide full end-to-end encryption, so it still requires some trust of the underlying infrastructure and administration.

4.2 State of the Art

As mentioned in Section 3, the basic Unix and Windows NT file system security models are by far the most widely deployed file system security approaches. Where distributed file systems are concerned, NFS, AFS, and CIFS are the de facto standards. As discussed, all of these systems lack support for end-to-end cryptographic security, global access, and inter-domain sharing. They are the standard not because of their rich, modern feature sets (which they lack), but simply because they are adequate for many of the standard single user or single domain use cases still deployed today. Replacing these systems would often be a larger burden than it is worth. These are not ideal systems, but they are "good enough" for many users, and thus remain the standard.

In many instances, the traditional systems listed above are not so much threatened by the more advanced research systems previously discussed (e.g. SFS, Bayou, Plutus, etc) but by the growing proliferation of cloud-based distributed storage service like Dropbox or Google Drive. Today, when users wishes to share files across administrative boundaries (or even within these boundaries) or wishes to sync files across multiple devises, they often do so using a dedicated file sharing/syncing service based in the cloud. These services tend to have highly polished user interfaces and overcome many of the single-domain, single-system restrictions of more traditional approaches like NFS or CIFS. While polished and easy to use, these systems do require placing full trust in the cloud providers who operate them. While most users seem willing to trade trust and privacy for the features these service provide, there is a growing understanding (a la Edward Snowden and our friends at the National Security Agency) of the privacy risk offloading data to the cloud involves. While these risks don't seem to deter a majority of users (at least this time), they do make such service strictly off limits within certain high security and regulatory realms. To date, none of the major cloud storage services offer access to end-to-end encryption schemes that would allow users to store their data in the cloud while also minimizing their need to trust a given cloud service provider. The reasons behind the lack of a cryptographically secure cloud service seems to be a combination of lack of user demand, usability challenges, and the conflicts of interested between privacy-minded user and data-mining based cloud business models.

While cryptographically secure distributed or cloud-based file systems are not in wide spread use, there are a growing number of cryptographically secure local file systems worth mentioning. On Linux, systems like eCryptfs provide an overlay file system capable of performing transparent encryption on a subset of the system's file tree. Alternatively, systems like dm-crypt provide block-level encryption suitable for full-disk encryption schemes. Using a local disk encryption system helps to guard against data loss or compromise in the event that a storage device falls into an advisories hands. Such systems are particularly useful on mobile computing devices like laptops or tablets since these devices tend to be more prone to theft or loss. Systems like BitLocker on Windows or FileVault on OSX can be used to provide similar features. Even most modern SSD-based hard disks tend to offer on-board encryption features to help protect them in the event that they are lost. All of these systems tend to be fairly user friendly and/or transparent. They often tie encryption keys to a user's login password, allowing a user to encrypt their system without requiring any more effort than they would exert during a normal password-based login process. This approach, however, does mean that such systems are not useful for protecting files that can be stolen while a user is actively logged into a system (e.g. via a malicious program). They only provide protection when a system is powered off, locked, or similarly put into a non-active state.

4.3 Future Extensions

There is a lot of work to be done in order to make cryptographically secure, distributed file systems a day to day reality for most users. While theoretical research systems exist that can provide many benefits over the current status quo, many of these systems introduce usability challenges that make them unsuitable for the average user. As mentioned in previous sections, many of these usability challenges can be directly linked to the problems that arise managing private cryptographic keys across our modern multi-device, multi-user use cases. Usability is the major hurdle preventing cryptographically strong distributed storage systems form entering the mainstream. Providing systems that can offer cryptographic security while also achieving the kind of intuitive usability provided by service like Dropbox or Google

Drive needs to be a major area of research if we wish for such systems to provide a benefit other than supporting tenure-track progress within the ivory tower.

The rise of cloud-based services is only going to increase the demand for secure storage systems that can support multi-user, multi-device, multi-domain user cases while also minimizing the need to trust cloud providers. Providing such a system will open up the growing range of cloud service to an even wider audience, and will help to keep users in control of their data even as we increasingly outsource our computing resources. Custos [12] was one attempt at building a component of such a system. Other attempts and further research are required if we're serious about closing the cloud vs trust divide.

5. USABILITY

There's little use in having highly secure systems that are impossible to manage/use. Unfortunately, manageability and ease of use is a major challenge on many secure systems in use today. Multiple research efforts have shown that one of the most common causes of security failures is misconfiguration. Likewise, there have been multiple studies showing the unwillingness of users to adopt new security practices if such practices increase the usage burden of a system. In this section, I'll discuss how secure systems can be made more usable across developer, administrative, and end-user use cases.

5.1 History

The usability of security has been a concern since the early days of thinking about computer security. Unfortunately, it hasn't always been a priority. There are multiple facets of usability with respect to security. In particular, it is useful to analyze the usability of a system from the frames of reference of various stake holders. There are three core stake holders who are affected by the usability of a security scheme: developers, administrators, and end-users. Developers care about the programmatic usability of a system: e.g. how easily can a security scheme be integrated with other systems? Administrators care about the management usability of a system: e.g. How easily can a security scheme be installed, configured, and managed? Finally end-users care about the consumer usability of a system: which is really to say, they'd generally prefer not to have to care about a secure system at all. In all these cases, usability is of the utmost importance. Security is generally a secondary usage goal: it's not the primary feature a system is trying to implement. Instead, it's a means to an end. Thus, if security systems become too burdensome, they will simply be ignored or removed. Security that gets in the way of the user instead of naturally accommodating the user is security that will go unused.

As mentioned in previous sections, early computer security implementations often involved hard coding security primitives directly into the program that used them. Unfortunately, this practice quickly leads to developer usability challenges, tightly coupling security policy with security mechanism and making it difficult to update one without changing the other. To deal with issues like this, Vipin Samar invented the Pluggable Authentication Modules (PAM) system and presented it in his 1996 paper [10]. The PAM system was designed to provide a flexible interface for user authentication on Unix (and later Linux) systems. It strips the `login` program of its internal authentication mechanism code, instead providing it with an interface into an extensible authentication plugin stack. Like the GSS-API system that compliments it, PAM is designed to provide a generic security mechanism (in this case, a generic authentication mechanism) so that individual programs do not have to hard-code their authentication algorithms. In this manner, PAM systems ease developer usability by freeing programmers from the burden of having to code an authentication stack directly into their applications. This also avoids the need to constantly upgrade applications simply to support new authentication mechanisms. Furthermore, PAM expands the administrator's ability to control how authentication mechanisms are used within each application. It allows administrators to control exactly which authentication primitives get used by each application on a system, even allowing them to specific multiple authentication primitives where required (e.g. to provide multi factor authentication). The manner in which PAM does this is fairly straightforward, ideally promoting administrative usability in addition to developer usability.

Beyond the developer and administrative usability of authentication systems lies a whole suite of usability challenges when it comes to having users managing passwords and authentication credentials (e.g. keys) for a range of services. Users are often asked to remember a wide range of passwords for a variety of services. When using cryptographically based authentication schemes (e.g. public/private keys/certificates), users must also keep track of all the necessary private keys so that they can use them when required. Thus, the rather mundane task of password or certificate based authentication leads to a large end-user burden, encouraging the creation of simple, repetitive passwords or discouraging the use of cryptographic certificates. Cox, et. al present a potential solution to these kinds of problems in the 2002 paper on "Security in Plan 9" [2]. Cox, et. al. propose using a security "agent" to manage the complexity of maintaining multiple authentication secrets (e.g. passwords or keys) on behalf of a user. They build such an agent for the Plan 9 operating system called Factotum. The idea of having a dedicated "agent" manage credentials on the user's behalf predates Factotum (i.e. the SSH agent), but Factotum generalizes the concept to make it an application-agnostic credential manager. A security agent like Factotum locally stores and tracks all credentials for a given user. These credentials are encrypted, generally using the user's login password, ensuring that the agent has access to them when the user is logged in, but that they remain secure when the user is offline. When the user attempts to authenticate to a service that requires credentials, Factotum looks them up and provides them on the user's behalf. If Factotum lacks the necessary credentials, it prompts the user for them, and then stores them for future use. In this manner, Factotum can make authentication to a range of disparate services a largely seamless task for the user, greatly easing end-user usability and encouraging the use of strong password or cryptographic authentication techniques that would otherwise be too burdensome to leverage.

Morgan, et. al. present an alternate approach to managing the authentication across a range of service in their 2004 paper "Federated Security: The Shibboleth Approach" [8]. Where as agent-based system like Factotum deal with the multitude of user credentials on the client side, Shibboleth attempts to deal with this problem from the server side. Shibboleth is a from of advanced Single-Sign-On (SSO) system that allows users to use a single set of credentials to authenticate to a range of Shibboleth-backed sites. In contrast to more traditionally SSO systems, Shibboleth offers support for federations: the ability to share identify information across administrative domains. In Shibboleth, each users is associated with an Identity provider (IdP) server. This server maintains identity and authentication information about a user. When a user wishes to authenticate to a third party website, that website looks up and contacts the user's IdP server. The user's session is passed to the IdP server where user completes the authentication process, after which they are passed back to the original website with a token (i.e. assertion) stating whether or not their authentication attempt was successful and what access attributes they should be granted. The original site then grants access based on the validity of this token and the associated attributes it provides. Shibboleth leverages the Security Assertion Markup Language (SAML) on the back-end as a standardize format for passing security assertions between an IdP and a third party site. Since Shibboleth only requires that a user maintain and provide a single set of authentication credentials regardless of the number of disparate sites or services they need to access, it can greatly reduce the user's security burden and encourage the use of more secure (albeit harder to remember) credentials. It also eases developer usability by relieving services of the need to build their own authentication systems (similar to PAM) and administrative usability by providing a centralized point at which all user attributes and permissions can be managed.

5.2 State of the Art

Today, a wide range of security management systems designed to increase various usability aspects are in common use. PAM is still the de facto standard for managing authentication on Unix and Linux systems. Today, PAM supports a wide range of authentication primitives, from passwords, to multi-factor devices, to hardware-based SmartCards. PAM support authentication for a range of Unix and Linux subsystems including `login`, SSH, FTP, etc. Windows provides similar pluggable authentication interfaces. All of these systems encourage the rapid development of authentication primitives without unnecessarily tying them to the associated program leveraging them. They also expose a lot of administrative flexibility when it comes to configuring authentication across a range of services.

While the concept of an OS-managed general security agent such as Factotum hasn't really gone mainstream, specific systems make use of agents for credential management, namely SSH and GnuPG. Password mangers, however, have become increasingly mainstream. These system specialize in the storage of passwords and other secrets, and while they often lack the fully integrated nature of an authentication agent, they do tend to support rudimentary autocompletion of user secrets where required. Since the bulk of day-today user authentication today takes place on the web, most password managers are browser based. In many ways, the browser in the modern "OS" for many user activities. In that manner, browser-based password managers such as LastPass or 1Password fulfill many of the same goals as Factotum: they ease the end-user's security burden and encourage better security practices in the process. Most security experts recommend the use of a password manager for all users today. The fact that many users use the same password across multiple sites and choose essentially weak passwords is a major barrier to internet security. Password managers help counter this weakness by focusing all of the user's memorization efforts in a single, strong master password while encourage the use of long, random passwords between the password management software and the third part web site or service.

The Shibboleth system is deployed and in use across a range of academic and Internet2 infrastructure. Similar federated identity protocols such as OpenID or Persona have become common across the wider internet. Large cloud providers like Google or Facebook often act as identify providers for users who already have accounts with them, allowing these users to authenticate to other websites without creating additional accounts. Identity provision is quickly becoming a core role of large cloud service providers, and options like "Login with Facebook" or "Login with Google" are fairly ubiquitous across web services. These systems benefit users by reducing the number of credentials they must remember. When coupled with password managers and/or multi-factor authentication systems, they provide the basis for a much stronger web authentication framework than forcing users to remember a different password for each web site they utilize would allow.

5.3 Future Extensions

Usability across administrative, developer, and end user domains remains a challenge in the security realm. Building secure systems that are easy to use, easy to manage, and easy to build is a prerequisite to having any security system gain wide spread adoption. Many otherwise suitable security approaches have been doomed by the fact that they posed serious usability issues for one or more of the aforementioned groups.

One of the major remaining usability challenges is deciding on the best manner in which to manage user secrets and credentials across the multitude of devices today's user expect to use. Traditional agent programs generally fail in multi-device use cases since these system keep only a local cache of user secretes that is useless if the user is trying to authenticate from a system other than the one on which they originally provided their credentials. This calls for a form of agent program capable of syncing data across multiple devices, while also retaining the security of the data should one of the devices become compromised.

Systems like browser-based password managers or federated identity services overcome the multi-device issue by providing a central, often cloud based, repository of user credentials or identity data. But this introduces a new issue: trust. Do we really want to be trusting a single third party cloud provider with all of our credentials, or with our identity and it's accompanying metadata? While convenient, such trust does seed user privacy and control, and potentially increases the risk of user data exposure should a third party lack scruples, fall to an attack, or be legally compelled to provide user data.

The best manner in which to provide the kind of convenience security agents, password managers, and identity providers offer while also supporting multi-device use cases and minimizing third part trust requirements remains an open question. One potential approach to building such a system would be to leverage one of the aforementioned cryptographically secure storage technologies as the basis of multi-device secret manager. Doing this in a manner that does not increase user burden (by forcing them to operate their own storage infrastructure) will not be easy. Peer-to-peer systems like BitTorrent Sync might provide avenues toward building such a system, but such a system has yet to be seriously proposed or prototyped. Furthermore, most cryptographically secure storage systems require some form of private key management, which in and of itself can pose usability challenges. This creates a bit of a "chicken and the egg" problem where technologies to provide security in the absence of third party trust require just such systems to reduce the end-user usage burden. The solution to this cyclical usability vs trust problem is a potential area of active research (possible avenues for with are discussed in [12]).

6. CONCLUSION

Computer security has been said to be like insurance: no one wants to deal with it until they need it, and by then it's too late. Ensuring that we can build secure computing systems is going to be the cornerstone or computing's success in the future. If computers and the service we build atop them can not be trusted to remain secure, they will be abandoned as tools of serious work. In order to advance the state of the art in computer security, I propose focusing on the following problems. These topics cut across the previously discussed security realms and help to elucidate the originally posed question, "How can we secure our systems and data in a robust, comprehensive, and easy-to-use manner?"

Multi-User|Multi-Device|Multi-Domain:
: Most dominant use cases today require support for all of the "multi-*" scenarios listed above. Users expect the ability to share and collaborate with other users, users expect system to work across a range of personal computing devices, and users do not wish to be burdened by arbitrary delineations like a specific administrative domain. Any successful security design must accommodate all of these desires gracefully if it is to succeed.

Control of Trust:
: The rise of the cloud has provided numerous benefits and conveniences to most end users. Unfortunately, it often does so at the expense of allowing users to decide whom to trust and to place in control of their data. Successful security systems must grant users the leeway to pick and chose whom to trust and how their data can be used.

Usability:
: Any security system that significantly (or for that matter, even moderately) increases the effort a user must expend to complete a task is doomed to fail. End-users, developers, and administrators do not generally enjoy having to think about security. Successful security systems must be easy to use and avoid subjecting users to onerous burdens if they are to be adopted.

These goals are lofty, but I believe they are attainable. Future research work focused around accomplishing one or more of them will provide the basis of the future of computing security, and by extension, the future of computing in general. Working toward the betterment of computer security is an important pursuit and one that must be undertaken to ensure the long term viability of modern computing practices.

7. REFERENCES

[1] Bethencourt, J., Sahai, A., and Waters, B. Ciphertext-Policy Attribute-Based Encryption. In *IEEE Symposium on Security and Privacy, 2007* (May 2007), IEEE, pp. 321–334.

[2] Cox, R., Grosse, E., Pike, R., Presotto, D., and Quinlan, S. Security in Plan 9. In *USENIX Security* (2002), pp. 3–16.

[3] Diffie, W., and Hellman, M. E. New directions in cryptography. *IEEE Transactions on Information Theory 22*, 6 (Nov. 1976), 644–654.

[4] Goyal, V., Pandey, O., Sahai, A., and Waters, B. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security - CCS '06* (New York, New York, USA, 2006), ACM Press, p. 89.

[5] Kher, V., and Kim, Y. Securing distributed storage: challenges, techniques, and systems. In *Proceedings of the 2005 ACM workshop on Storage security and survivability* (New York, New York, USA, 2005), ACM Press, p. 9.

[6] Mazières, D., Kaminsky, M., Kaashoek, M. F., and Witchel, E. Separating key management from file system security. *ACM SIGOPS Operating Systems Review 33*, 5 (Dec. 1999), 124–139.

[7] Miltchev, S., Smith, J. M., Prevelakis, V., Keromytis, A., and Ioannidis, S. Decentralized access control in distributed file systems. *ACM Computing Surveys 40*, 3 (Aug. 2008), 1–30.

[8] Morgan, R. L. B., Cantor, S., Carmody, S., Hoehn, W., and Klingenstein, K. Federated Security: The Shibboleth Approach. *Educause Quarterly 27*, 4 (2004), 12–17.

[9] Rivest, R. L., Shamir, A., and Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21*, 2 (1978), 120–126.

[10] Samar, V. Unified login with pluggable authentication modules (PAM). In *Proceedings of the 3rd ACM Conference on Computer and Communications Security* (New York, New York, USA, 1996), ACM Press, pp. 1–10.

[11] Sandhu, R. S., Coynek, E. J., Feinstein, H. L., and Youman, C. E. Role-Based Access Control Models. *IEEE Computer 29*, 2 (1996), 38–47.

[12] Sayler, A. Custos: A Flexibly Secure Key-Value Storage Platform. Master's thesis, University of Colorado Boulder, December 2013.

[13] Shamir, A. How to share a secret. *Communications of the ACM 22*, 11 (Nov. 1979), 612–613.