RESEARCH ARTICLE OPEN ACCESS

Resist Dictionary Attacks Using Password Based Protocols For Authenticated Key Exchange

NagamaniAbirami[1] ,Jagapriya R[2], Kavipriya R[3] , Nithya A[4]
[1]Assistant Professor , Department of Computer Science, Manakula Vinayagar Institute of Technology, Pondicherry university, India.
[2 , 3 , 4]B.Tech. ,Department of Computer Science, Manakula Vinayagar Institute of Technology, Pondicherry university, India.

ABSTRACT
A parallel file system is a type of distributed file system that distributes file data across multiple servers and provides for concurrent access by multiple tasks of a parallel application. In many to many communications or multiple tasks, key establishments are a major problem in parallel file system. So we propose a variety of authenticated key exchange protocols that are designed to address the above issue. In this paper, we also study the password-based protocols for authenticated key exchange (AKE) to resist dictionary attacks. Password-based protocols for authenticated key exchange (AKE) are designed to work to resist the use of passwords drawn from a space so small that attacker might well specify, off line, all possible passwords. While many such protocols have been suggested, the elemental theory has been lagging. We commence by interpreting a model for this problem, to approach password guessing, forward secrecy, server compromise, and loss of session keys.

I. INTRODUCTION

Both parallel and distributed systems can be defined as a collection of processing elements that communicate and cooperate to achieve a common goal. Processor technology using parallelism at all levels: within each CPU by executing multiple instructions from different threads of control simultaneously (simultaneous multithreading); by introducing multiple cores in a single chip (chip multiprocessors); by using multiple chips to form multiprocessors; or via multiple networked nodes to form a cluster; making parallel systems increasingly ubiquitous.

Simultaneously, advances in networking technology have created an explosion of distributed applications, making distributed computing an inherent fabric in our day-to-day lives. This course will focus on the principles of parallel and distributed systems and the implementation and performance issues associated with them. We will examine programming models/interfaces to parallel and distributed computing, interprocess communication, synchronization and consistency models, fault tolerance and reliability, distributed process management, parallel machine architectures, parallel program optimization, and the interaction of the compiler, run-time, and machine architecture.

Difference between distributed system and parallel system are given below:

- Distributed Operating systems are also referred to as Loosely Coupled systems whereas parallel processing systems are referred to as loosely coupled systems.
- A Loosely coupled system is one in which the processors do not share memory and each processor has its own local memory whereas in a tightly coupled system there is a single system wide primary memory shared by all the processors.
- The processors of distributed operating systems can be placed far away from each other to cover a wider geographic area which is not the case with parallel processing systems.
- The no. of processors that can be usefully deployed is very small in a parallel processing operating system whereas for a distributed operating system a larger no. of processors can be usefully deployed.

1.1 Distributed Computing

A distributed system is a network of autonomous computers that communicate with each other in order to achieve a goal. The computers in a distributed system are independent and do not physically share memory or processors. They communicate with each other using *messages*, pieces of information transferred from one computer to another over a network. Messages can communicate many things: computers can tell other computers to execute a procedures with particular arguments, they can send and receive packets of data, or send signals that tell other computers to behave a certain way.

Computers in a distributed system can have different roles. A computer's role depends on the goal of the system and the computer's own hardware and software properties. There are two predominant ways of organizing computers in a distributed system. The first is the client-server architecture, and the second is the peer-to-peer architecture.

1.2 Parallel Computing

Computers get faster and faster every year. In 1965, Intel co-founder Gordon Moore made a prediction about how much faster computers would get with time. Based on only five data points, he extrapolated that the number of transistors that could inexpensively be fit onto a chip would double every two years. Almost 50 years later, his prediction, now called Moore's law, remains startlingly accurate.

Despite this explosion in speed, computers aren't able to keep up with the scale of data becoming available. By some estimates, advances in gene sequencing technology will make gene-sequence data available more quickly than processors are getting faster. In other words, for genetic data, computers are become less and less able to cope with the scale of processing problems each year, even though the computers themselves are getting faster.

To circumvent physical and mechanical constraints on individual processor speed, manufacturers are turning to another solution: multiple processors. If two, or three, or more processors are available, then many programs can be executed more quickly. While one processor is doing one aspect of some computation, others can work on another. All of them can share the same data, but the work will proceed in parallel.

In order to be able to work together, multiple processors need to be able to share information with each other. This is accomplished using a shared-memory environment. The variables, objects, and data structures in that environment are accessible to all the processes. The role of a processor in computation is to carry out the evaluation and execution rules of a programming language. In a shared memory model, different processes may execute different statements, but any statement can affect the shared environment.

II. RELATED WORK

Some of the earliest work in securing large-scale distributed file systems, for example [1], [2], have already employed Kerberos for performing authentication and enforcing access control. Kerberos, being based on mostly symmetric key techniques in its early deployment, was generally believed to be more suitable for rather closed, well-connected distributed environments.

On the other hand, data grids and file systems such as, OceanStore [3], LegionFS and FARSITE [7], make use of public key cryptographic techniques and public key infrastructure (PKI) to perform cross-domain user authentication. Independently, SFS, also based on public key cryptographic techniques, was designed to enable inter-operability of different key management schemes. Each user of these systems is assumed to possess a certified public/private key pair. However, these systems were not designed specifically with scalability and parallel access in mind.

With the increasing deployment of highly distributed and network-attached storage systems, subsequent work, such as [8], focussed on scalable security. Nevertheless, these proposals assumed that a metadata server shares a group secret key with each distributed storage device. The group key is used to produce capabilities in the form of message authentication codes. However, compromise of the metadata server or any storage device allows the adversary to impersonate the server to any other entities in the file system. This issue can be alleviated by requiring that each storage device shares a different secret key with the metadata server. Nevertheless, such an approach restricts a capability to authorising I/O on only a single device, rather than larger groups of blocks or objects which may reside on multiple storage devices.

More recent proposals, which adopted a hybrid symmetric key and asymmetric key method, allow a capability to span any number of storage devices, while maintaining a reasonable efficiency-security. For example, Maat encompasses a set of protocols that facilitate (i) authenticated key establishment between clients and storage devices, (ii) capability issuance and renewal, and (iii) delegation between two clients. The authenticated key establishment protocol allows a client to establish and re-use a shared (session) key with a storage device. However, Maat and other recent proposals do not come with rigorous security analysis.

As with NFS, authentication in Hadoop Distributed File System (HDFS) is also based on Kerberos via GSS-API. Each HDFS client obtains a TGT that lasts for 10 hours and renewable for 7 days by default; and access control is based on the Unix-style ACLs. However, HDFS makes use of the Simple Authentication and Security Layer (SASL), a framework for providing a structured interface between connection-oriented protocols

and replaceable mechanisms. In order to improve the performance of the KDC, the developers of HDFS chose to use a number of tokens for communication secured with an RPC digest scheme. The Hadoop security design makes use of Delegation Tokens, Job Tokens, and Block Access Tokens. Each of these tokens is similar in structure and based on HMAC-SHA1. Delegation Tokens are used for clients to communicate with the Name Node in order to gain access to HDFS data; while Block Access Tokens are used to secure communication between the Name Node and Data Nodes and to enforce HDFS file system permissions. On the other hand, the Job Token is used to secure communication between the MapReduce engine Task Tracker and individual tasks. Note that the RPC digest scheme uses symmetric encryption and depending upon the token type, the shared key may be distributed to hundreds or even thousands of hosts.

III. SYSTEM ANALYSIS

3.1. Existing System

In this work, we investigate the problem of secure many to-many communications in large-scale network file systems that support parallel access to multiple storage devices. That is, we consider a communication model where there are a large number of clients accessing multiple remote and distributed storage devices in parallel. Particularly, we focus on how to exchange key materials and establish parallel secure sessions between the clients and the storage devices in the parallel Network File System. Our review of the existing protocol shows that it has a number of limitations. A metadata server facilitating key exchange between the clients and the storage devices has heavy workload that restricts the scalability of the protocol. The protocol does not provide forward secrecy. The metadata server generates itself all the session keys that are used between the clients and storage devices, and this inherently leads to key escrow.

3.2 Problem Definition

- Heavy Workload
- Scalability is restricted
- Does not provide forward secrecy
- Key escrow
- Security is low

3.3 Proposed System

In this paper, we propose a variety of authenticated key exchange protocols that are designed to address existing issues. We also study the password-based protocols for authenticated key exchange (AKE) to resist dictionary attacks. Password-based protocols for authenticated key exchange (AKE) are designed to work to resist the use of passwords drawn from a space so small that an attacker might well enumerate, off line, all possible passwords. While several such protocols have been suggested, the underlying theory has been lagging. We begin by defining a model for this problem, one rich enough to deal with password guessing, forward secrecy, server compromise, and loss of session keys.

3.4 Advantage in proposed system

- Scalability achieved
- Collusion avoided
- Guarantee the security of past session keys
- Reduced workload
- High Security
- Resist dictionary attacks

IV. SYSTEM DESIGN

Systems design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development.

4.2 System Architecture

System architecture is the conceptual model that defines the structure, behaviour, and more views of a system. And then the architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviours of the system.

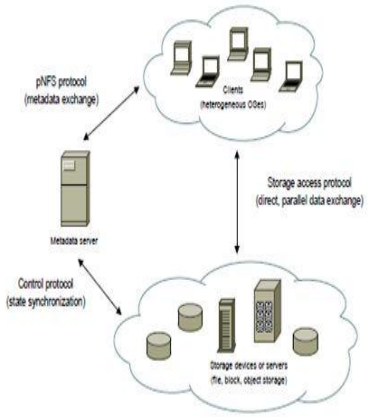

Fig 4.2 System architecture

V. ALGORITHM

5.1 Encrypted Key- Exchange(EKE) protocol

Encrypted Key Exchange (also known as EKE) is a family of password-authenticated key agreement methods described by Steven M. Bellovin and Michael Merritt. Although several of the forms of EKE were later found to be flawed, the surviving, refined, and enhanced forms of EKE effectively make this the first method to amplify a shared password into a shared key, where the shared key may subsequently be used to provide a zero-knowledge password proof or other functions. In the most general form of EKE, at least one party encrypts an ephemeral (one-time) public key using a password, and sends it to a second party, who decrypts it and uses it to negotiate a shared key with the first party.

Augmented methods have the added goal of ensuring that password verification data stolen from a server cannot be used by an attacker to masquerade as the client, unless the attacker first determines the password.

5.2 Executing the protocol: Formally, a protocol is just a probabilistic algorithm taking strings to strings. This algorithm determines how instances of the principals behave in response to signals (messages) from their environment.

Our communications model places the adversary at the centre of the universe. The adversary A can make queries to any instance: she has an endless supply of Π^i_U oracles (U ϵ ID and i ϵ N). There are all together six types of queries that A can make. The responses to these queries are specified in Figure 5.2. We now explain the capability that each kind of query captures.

(1) Send (U, *i*, M) | This sends message M to oracle Π^i_U. The oracle computes what the protocol says to, and sends back the response. Should the oracle accept, this fact, as well as the SID and PID, will be made visible to the adversary. Should the oracle terminate, this too will be made visible to the adversary. To initiate the protocol with client A trying to enter into an exchange with server B the adversary should send message M = B to an unused instance of A. A Send-query models the real-world possibility of an adversary A causing an instance to come into existence, for that instance to receive communications fabricated by A, and for that instance to respond in the manner prescribed by the protocol.

Algorithm

Initialization()
{ h ←R Ω h pwA, pwBi A∈Client,B∈Server ←R PWh ()
for i ∈ N and U ∈ ID do
statei U ← ready acci U ← termi U ← usedi U ← false
sidi U ← pidi U ← ski U ← false }
Send(U, i, M)
{ usedi U ← true if termi U = true then return invalid
hmsg-out, acc, termi U , sid, pid, sk, statei U i ←
P h (hU, pwU , statei U , Mi)
if (acc = true and ¬acci U = true) then
sidi U ← sid; pidi U ← pid; ski U ← sk; acci U ← true
return hmsg-out, sid, pid, acc, termi U i }
Reveal(U, i)
{ return ski U }
Execute(A, i, B, j)
{ if A ∈/ Client or B ∈/ Server or usedi A = true or usedj B = true
then return invalid
msg-in ← B
for t ← 1 to ∞ do
hmsg-out, sid, pid, acc, termAi ←R Send(A, i, msg-in)
αt ← hmsg-out, sid, pid, acc, termAi
if termA and termB then return hα1, β1, α2, β2, ..., αti
hmsg-out, sid, pid, acc, termBi ←R Send(B, j, msg-in)
βt ← hmsg-out, sid, pid, acc, termBi
if termA and termB then return hα1, β1, α2, β2, ..., αt, βti }
Oracle(M)
{ return h(M) }

(2) Reveal (U, *i*) - If oracle Π^i_U has accepted, holding some session key *sk*, then this query returns *sk* to the adversary. This query models the idea (going back to Denning and Sacco) that loss of a session key shouldn't be damaging to other sessions. A session key might be lost for a variety of reasons, including hacking, cryptanalysis, and the prescribed-release of that session key when the session is torn down.

(3) Corrupt (U, *pw*) - The adversary obtains pw_U and the states of all instances of U. This query models the possibility of subverting a principal by, for example, witnessing a user type in his password, installing a "Trojan horse" on his machine, or hacking into a machine. Obviously this is a very damaging type of query. Allowing it lets us deal with forward secrecy and the extent of damage which can be done by breaking into a server. A Corrupt query directed against a client U may also be used to replace the value of pw_B[U]

used by server B. This is the role of the second argument to Corrupt. Including this capability allows a dishonest client A to try to defeat protocol aims by installing a strange string as a server B's transformed password pw_B [A].

(4) Execute (A, *i*, B, *j*) - Assuming that client oracle Π^i_A and server oracle Π^j_B have not been used; this call carries out an honest execution of the protocol between these oracles, returning a transcript of that execution. This query may at first seem useless since, using Send queries; the adversary already has the ability to carry out an honest execution between two oracles. Yet the query is essential for properly dealing with dictionary attacks. In modeling such attacks the adversary should be granted access to plenty of honest executions, since collecting these involves just passive eavesdropping. The adversary is comparatively constrained in its ability to actively manipulate °ows to the principals, since bogus flows can be auditied and punative measures taken should there be too many.

(5) Test (U, *i*) - If Π^i_U has accepted, holding a session key *sk*, then the following happens. A coin b is flipped. If it lands b = 0, then sk is returned to the adversary. If it lands b = 1, then a random session key, drawn from the distribution from which session keys are supposed to be drawn, is returned. This type of query is only used to measure adversarial success it does not correspond to any actual adversarial ability. You should think of the adversary asking this query just once.

(6) Oracle (M) - Finally, we give the adversary oracle access to a function h, which is selected at random from some probability space Ω. As already remarked, not only the adversary, but the protocol and the LL-key generator may depend on h. The choice of - determines if we are Woking in the standard model, ideal-hash model, or ideal-cipher model.

VI. PERFORMANCE EVALUATION

6.1 Communication overhead

Assuming fresh session keys are used to secure communications between the client and multiple storage devices, clearly all our protocols have reduced bandwidth requirements. This is because during each access request, the client does not need to fetch the required authentication token set from M. Hence, the reduction in bandwidth consumption is approximately the size of n authentication tokens. The total delay can be calculated as

$$(n-2)\ell + s/b$$

Where *n* is the number of messages sent; *l* is the latency in seconds; *s* is the total size of all messages; and *b* is the bandwidth in bytes per second.

6.2 Key Storage

We note that the key storage requirements for Kerberos pNFS and all our described protocols are roughly similar from the client's perspective. For each access request, the client needs to store N or N + 1 key materials (either in the form of symmetric keys or Diffie-Hellman components) in their internal states. However, the key storage requirements for each storage device is higher in pNFS-AKE-III since the storage device has to store some key material for each client in their internal state. This is in contrast to Kerberos-pNFS, pNFS-AKE-I and pNFS-AKE-II that are not required to maintain any client key information.

6.3 Security

First, whenever it happens we have a way to \embed" instances of the DH problem into the protocol so that adversarial success leads to our obtaining a solution to the DH problem. Second, absence of the bad event leads to an inability of the adversary to obtain information about the password at a better rate than eliminating one password per reveal or test query to a manipulated oracle. Bounding the probability of the bad event involves a \simulation" argument as we attempt to \plant" DH problem instances in the protocol. Bounding adversarial success under the assumption the bad event does not happen is an information-theoretic argument. Indeed, the difficulty of the proof is in choosing the bad event so that one can split the analysis into an information-theoretic component and a computational component in this way.

VII. CONCLUSION

We proposed password-based protocols for authenticated key exchange (AKE) to resist dictionary attacks for parallel network file system (pNFS). Password-based protocols for authenticated key exchange (AKE) are designed to work despite the use of passwords drawn from a space so small that an adversary might well enumerate, off line, all possible passwords. While several such protocols have been suggested, the underlying theory has been lagging. We begin by defining a model for this problem, to approach password guessing, forward secrecy, server compromise, and loss of session keys.

REFERENCE

[1]. J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West. Scale and performance in a distributed file system. ACM Transactions on Computer Systems (TOCS), 6(1):51–81. ACM Press, Feb 1988.

[2]. G.A. Gibson, D.F. Nagle, K. Amiri, J. Butler, F.W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A costeffective, high-bandwidth storage architecture. ACM SIGPLAN Notices, 33(11):92–103. ACM Press, Nov 1998.

[3]. J. Kubiatowicz, D. Bindel, Y. Chen, S.E. Czerwinski, P.R. Eaton, D. Geels, R. Gummadi, S.C. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B.Y. Zhao. OceanStore: An architecture for global-scale persistent storage. In Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 190–201. ACM Press, Nov 2000.

[4]. Hadoop Wiki. http://wiki.apache.org/hadoop/PoweredBy

[5]. F. Hupfeld, T. Cortes, B. Kolbeck, J. Stender, E. Focht, M. Hess, J. Malo, J. Marti, and E. Cesario. The XtreemFS architecture – a case for objectbased file systems in grids. Concurrency and Computation: Practice and Experience (CCPE), 20(17):2049–2060. Wiley, Dec 2008.

[6]. Hadoop distributed file system. http://hadoop.apache.org/hdfs/.

[7]. A. Adya, W.J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J.R. Douceur, J. Howell, J.R. Lorch, M. Theimer, and R. Wattenhofer. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In Proceedings of the 5th Symposium on Operating System Design and Implementation (OSDI). USENIX Association, Dec 2002.

[8]. M.K. Aguilera, M. Ji, M. Lillibridge, J. MacCormick, E. Oertli, D.G. Andersen, M. Burrows, T. Mann, and C.A. Thekkath. Blocklevel security for network-attached disks. In Proceedings of the 2nd International Conference on File and Storage Technologies (FAST). USENIX Association, Mar 2003.

[9]. M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. Communications of the ACM, 53(4):50–58. ACM Press, Apr 2010.

[10]. Amazon simple storage service (Amazon S3). http://aws.amazon.com/ s3/.

[11]. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In Advances in Cryptology – Proceedings of EUROCRYPT, pages 139–155. Springer LNCS 1807, May 2000.

[12]. D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In Advances in Cryptology – Proceedings of CRYPTO, pages 258–275. Springer LNCS 3621, Aug 2005.