Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

A SECURE DOMAIN NAME SYSTEM BASED ON INTRUSION TOLERANCE

WEI ZHOU[1, 2], LIU CHEN[3, 4]

[1]School of Computer, Wuhan University, Wuhan 430072, China
[2]Department of computer science, Huazhong Normal University, Wuhan 430079, China
[3]Department of Electronic and Information Engineering, Huazhong Normal University, Wuhan 430079, China
(2013)

Abstract:

DNS was not designed to be secure. The biggest security hole in DNS is the lack of support for data integrity authentication, source authentication, and authorization. In this paper, a secure DNS scheme based on intrusion tolerance is proposed. This secure DNS is intrusion-tolerant by using Byzantine intrusion tolerant technique and voting mechanism. The scheme provides high integrity, robustness, and availability of service in the presence of arbitrary failures, including failures due to malicious attacks. The proposed scheme consists of 3f+1 tightly coupled replicas per name server and guarantees safety and liveness properties of the system assuming no more than f replicas are faulty. By adding authentification of client and using symmetric key cryptography, the system guarantees a secure communication mechanism by providing a way to detect whether DNS data has been corrupted during communication over the Internet. Experimental results show that the scheme can provide a much higher degree of security and reliability, as well or even better than an implementation of the DNS security extension.

Keywords:

DNS; Intrusion tolerance; Byzantine fault tolerance; Voting

1. Introduction

DNS (Domain Name System) is an important part of Internet, which uses caching technology, distributed databases, and multi-level structure to realize the conversion between domain name and IP. At the beginning of being designed, DNS did not take into account security issues[1]. DNS system also has shown some flaws in application, such as vulnerability to phishing attacks. One of the reasons is that DNS server and client communicate by connectionless UDP, and there is no mechanism to judge whether the data packets are received from legitimate sources or not, which exposed DNS to the danger of various attacks. Furthermore, instead of verifying legitimacy of the responses from DNS server, DNS client will take by default that these responses are correct and are sent by legitimate DNS server. This also adds the dangers of DNS being attacked.

In order to solve the security problems of DNS, in 1997 IETF proposed Security Extensions for DNS[2][3], i.e. DNSSEC. The purpose of DNSSEC is to help the domain name resolving system of the client to find out whether the domain name it resolved is indeed valid or not, so that user can avoid the disclosure of his important information to attackers on phishing webs. As the security extensions based on current DNS, DNSSEC can effectively prevent DNS spoofing and secure legitimacy of the DNS records the client received.

However, as the entire DNS system on the Internet is already a very large distributed database of domain name records, the full realization of the conversion to DNSSEC requires a lot of time and workload. In addition, DNSSEC only provides a verification of the authenticity of DNS records, which can only provide a limited guarantee for the user's communications security. On the other hand, DNSSEC adds digital signatures to DNS requests and responses, which increases the communication traffic and complexity. This may lead to a new denial of service attacks based on encryption algorithm. So DNSSEC system is far from prevalent in reality till now.

Intrusion tolerance[4][5] is a new method that has evolved in recent years in the field of network security. Unlike traditional security methods, intrusion tolerance[6] assumes that there are unknown vulnerabilities in the system. Its basic idea is not to try to ward off invasion, but to tolerate the invasion when it happens, and continue providing services[7][8] with even lower performance. In this paper, a DNS system scheme based on intrusion tolerance is proposed. This DNS system is able to tolerate intrusion using redundant servers, Byzantine intrusion tolerant technique and voting mechanism. And a DNS system named as SDNS based on intrusion tolerance is designed according this scheme.

2. SDNS Model

The entire SDNS system is composed of proxy server group, DNS name server group, a voting server and a configuration management server, as shown in Figure 1.

User's request gets to a proxy server through outer defense. The proxy server transfers the request to the DNS name servers. The entrance of the whole system is the proxy server, which first analyzes the legitimacy of the request, then sends verified legitimate request to the primary one of DNS name servers. The communication and management between DNS servers uses improved Byzantine fault tolerant algorithm based on state machine replication. DNS name servers send the result to the voting server. Then the voting server sends back its trusted result based on voting mechanism to the proxy server, which transmits it to the client.

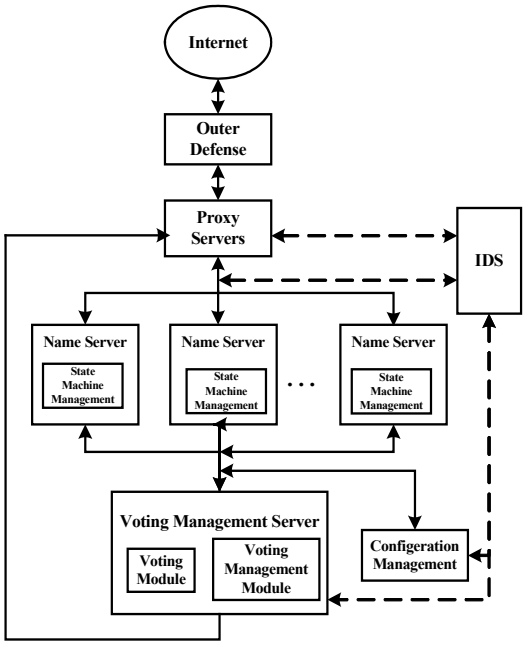

Figure1 System architecture of SDNS

As the entrance, proxy server is the bottleneck of the whole system and may also become easy targets of attack. In the proxy server group，one of the proxy server is assigned to act as primary proxy server, and others are assigned to act as secondary proxy servers, which together constitute a proxy server group. When the system is running, the primary proxy server transfers data between users and the system, while the secondary proxy servers monitoring the operation of the primary proxy server. Once the primary proxy server fails, one of the secondary proxy servers will be selected to be a new primary proxy server. Thus the balance of the load is achieved and the reliability of the system becomes better.

In order to keep proxy servers and DNS servers operating normally, IDS is deployed in the system. IDS detects attacks from outside and monitors the running condition of the servers. When abnormality happens, IDS will trigger configuration management module to reconfigure the system.

DNS name server group consists of several name servers, of which one is primary name server and others are secondary name servers. We use an improved Byzantine fault tolerant algorithm to ensure correct name service. The algorithm is described in 3.1.

The voting management server consists of voting management module and voting module. The main functions of voting management module are as follows: (1) it receives information from DNS servers, votes according voting algorithm and sends the voting result to proxy servers, analyzes all the information to distinguish the faulty servers from the normal ones; (2) it can change some of the parameters of the voting module (for example the threshold of the voting module) to improve the performance of voting algorithms.

The main function of configuration management is to reconfigure the system according the requests of other parts of the system, such as voting management server and DNS servers. The reconfiguration may be dynamic or static, which makes the system has better adaptability after its failure or being invaded, and adds difficulty to the attackers' consecutive attacks.

3. Design for SDNS

3.1. Byzantine Fault Tolerant Algorithm

In the current DNS system, there is one primary name server and a number of secondary name servers for a zone. The primary name server reads the zone data form a file on its host. The secondary name servers get the zone data from the primary. When a new host is added to a zone, or some records need to be changed, the administrator will modify the database in the primary server. The secondary server queries the primary server and if the primary server contains new data, the secondary obtains the new data and updates its database. This is referred as a zone transfer. DNS uses this simple replica mode to improve availability. If one name server is down, resolvers can still use other name servers. The DNS generally can only tolerant fail-stop failures. But fail-stop tolerance is not strong enough for the DNS, because malicious attacks are becoming increasingly common. This kind of attack can cause faulty nodes to act arbitrarily, which are far beyond the fail-stop failure model. If a DNS can tolerate Byzantine failures[9][10], it will be a

hacker-tolerant system, i.e., it will continue to provide correct service even when some of its components are controlled by an attacker.

Under a Byzantine fault model, a faulty node may act arbitrarily, this is in contrast to the more restrictive fail-stop model. Although only f + 1 replicas are required to tolerate f Byzantine faults if communication is assumed to be synchronous and replicas can authenticate messages, this synchrony assumption is not practical for environments like the Internet. In BFT(Byzantine failure tolerance) methods, most approaches use replicas. The basic idea is to make replicas of the process that we want to protect, run them on different hosts, and synchronize their behaviors with appropriate messages exchanged between them. We assume that at most f hosts may be attacked when there are at least 3f+1 replicas. Even if some of the hosts, not more than f, are hijacked and the replicas running on them are controlled by a cracker, other replicas maintain their original behavior, and we recognize f+1 behaviors that are the same as the original from 2f+1 results of replicas. Here, we can always expect only 2f+1 results because controlled agents may not indefinitely give results. Castro and Liskov's method is also based on this replica system. Castro–Liskov Practical Byzantine Fault Tolerance (CLBFT)[11][12] is a BFT state-machine-replication protocol and library implementation that can be used to create client–server Byzantine-fault-tolerant applications. Unfortunately, at present, their method assumes some structures in communication between processes, such as the server client model, where replication is applied only to servers, and clients have to be reliable. But clients may be hackers, they can implement man-in-middle attacks. In this paper, we propose an improved CLBFT method for DNS, in which the DNS server will authentificate the client.

The method uses a primary-backup mechanism where replicas move through a succession of configurations called views[5]. In a view, one replica is designated as the primary and the others are backups. The algorithm chooses the primary p of a view[12] v such that p= v mod |R| where views are numbered consecutively. View changes are carried out to provide liveness by allowing the system to make progress when the current primary fails.

In our algorithm, we call name servers of the DNS name servers group as replicas. The primary name server is called the primary replica, other name servers are called backup servers.

Communication between client, proxy server, voting server and replica is as follow:

(1)START phase

A client c makes a resolving request by sending K_p(Request, t, c, m) message to the primary proxy server. Timestamp t is used to ensure exactly-once semantics for the execution of client requests. Timestamps for c's requests are totally ordered such that later requests have higher timestamps than earlier ones. K_p is the public key of the primary proxy server. c is the client number. m is the signature for the request of client c. As the primary proxy server receive K_p(Request, t, c, m),it verifies the signature of client c. If the signature is correct, the primary proxy server transfers the request to the primary replica by sending K_r(Request, t, c) message. K_r is the public key of the primary replica. If the signature is wrong, the primary proxy server drops the message.

(2) REQUEST phase

As the primary replica receive the message K_r(Request, t, c) from the primary proxy server, it decrypts the message and extract the request. The primary server assigns a sequence number n to the request and sends n to the primary proxy server. The sequence number n is progressive increasing and not repetitive. The primary proxy server generates a session K_s and sends the encrypted K_s with the client c's public key K_c to c. After c receives the encrypted K_s, it gets K_s. The primary proxy server stores the data (c, n, K_s) into its local database.

(3)REPLY and END phase

The primary replica broadcasts the client c's request to all backup replicas. After three communication stages of PRE-PREPARE, PREPARE and COMMIT among replicas, the results from the replicas will be sent to the voting server. The voting server votes for the results from the replicas with voting algorithm. If the result is trusted, the voting server sends the result message (Reply, t, c, n) back to the primary proxy server. The primary proxy server gets the session key K_s from local database according to c and n, then send the encrypted results with K_s to the client c. After receiving the result encrypted with K_s from the primary proxy server, the client c decrypts the message and get the trusted result. If there is no trusted result when voting, the client c will send the request to the primary proxy server again. The primary proxy server broadcasts the request directly to all replicas, and REPLY and END phase will start again.

In this paper we use session key K_s encrypts the message between the client and the proxy server with symmetric encryption, which is much shorter and less costly than a public key

Communication among replicas is as follow:

(1) After receiving a request, the primary assigns a sequence number to it and to all of the replicas broadcasts a PRE-PREPARE message including the request and the number. The message format is PRE-PREPARE(v, n, m) where v is a number indicating the current primary, n is a sequence number and m is a request message. We omit the formats of the protocol messages introduced below.

(2) When a replica receives a PRE-PREPARE message, it broadcasts a PREPARE message with essentially the same content as the PRE-PREPARE message.

(3) When a replica receives the same 2f+1 valid PREPARE messages from different replicas, the replica enters a prepared state for that request and sequence number. Then the replica broadcasts a COMMIT message with the same content as the PREPARE message.

(4) When a replica receives the same 2f+1 valid COMMIT messages from different replicas, it enters a committed state for that request and sequence number. Then the replica executes the request and returns the result to the client.

In Figure2, there is 3f+1=4 replicas where f=1.Replica1 is the primary replica, the others are backup replicas. Replica4 has Byzantine fault. As is show in the figure, after three phase of START,REQUEST,REPLY and END, the client gets the trusted result.

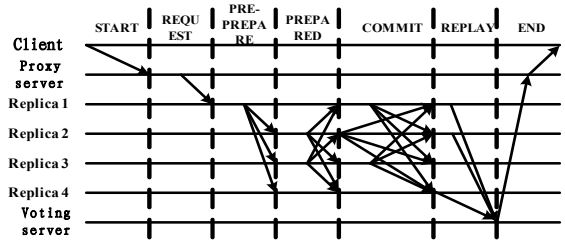

Figure2. Result producing process

3.2. Voting algorithm

Voting algorithm[13][14] is key for voting server. In the design of SDNS, we use formalized large number voting algorithm. The algorithm is as follow:

Let $X = (x_1 , x_2 , x_3 , ..., x_N)$ denotes the set of N replicas' results.

(1) Construct a partition $V_1, V_2, ..., V_k$ of X where for each i the set V_i is maximal with respect to the property that for any x and y in V_i, $d(x, y) \leqslant a$. where a is the voting threshold.

(2) Let V be the set in the partition $V_1, ... , V_k$ of the largest cardinality.

(3) If $|V| \geqslant f+1$, then select any value from V as voting output, where $|V|$ denotes the cardinality of the set V, f denotes the max number of faulty replicas.

The main idea of the algorithm is to select the largest set V whose cardinality $\geqslant$ f+1 from N replicas' results. As f is the max number of faulty replicas, if the cardinality of V is larger than f, any element of V is a trusted result.

4. Experiment and results

According to the scheme above, we realized SDNS prototype system, including client resolve program-the resolver and DNS service program-name server and agent service program-agent server and service voting program-voting server, and all the programs were implemented using c language. The communications between reclicas relied on CLBFT Replication Base. The security comparison of SDNS, DNSSEC and traditional DNS is shown in table 1. The response time an server time comparisons of SDNS, DNSSEC and traditional DNS are shown in table 2 and table 3, taking the most simple A and NS record resolving as an example. The response time is the response latency observed by the client from the time the client sends a request to the time it accepts a result and successfully interprets it. The server time is measured from the time the server receives a request to the time the result is ready to be sent.

Table 1 Security comparison of DNS,SNSSEC and SDNS

System	*Authentication*	*Security*
Traditional DNS	none	low
DNSSEC	Sign RR with zone private key	medium，but can not tolerate Byzantine fault
SDNS	Signature among client ,proxy and DNS servers	high，can tolerate Byzantine fault

Table 2 Performance comparison for the Resolving for A Record

System	*Response time（ms）*	*Server time（ms）*
Traditional DNS	0.254	0.012
DNSSEC	0.623	0.023
SDNS	0.517	0.031

Table 3 Performance comparison for the Resolving for NS Record

System	*Response time（ms）*	*Server time（ms）*
Traditional DNS	0.654	0.028
DNSSEC	0.826	0.062
SDNS	1.217	0.110

The results show that the response time and server time of SNDS are a little larger than traditional DNS and DNSSEC, but the security of SDNS is superior to traditional DNS and DNSSEC.

5. Conclusions

In this paper, we have proposed a secure domain name system (SDNS). The system provides high integrity, robustness, and availability of service in the presence of arbitrary failures, including failures due to malicious attacks. The proposed system consists of 3f+1 tightly

coupled replicas per name server and guarantees safety and liveness properties of the system assuming no more than f replicas are faulty.

By incorporating an improved CLBFT algorithm and voting mechanism in our solution, we not only make the system intrusion-tolerant, but also ensure correct name service even when a fraction of the replicas fail. Unlike the current insecure DNS and its security extension, our system tolerates server failures due to benign faults and malicious attacks.

Symmetric cryptography requires that every communication pair share a session key, which could lead to a huge number of keys servers to be remembered. We plan to add session key cache mechanism to the system.

References

[1] HUANG Jian-ye, WANG Feng, "The DNS Security and Solution Polices", Journal of Kunming University, Vol 17,No.4, pp.41-43,Jul.2006.

[2] D.Eastlake, "Secure domain name system dynamic update", Technical Report DARPA-Internet RFC 2137,CyberCash Inc., April 1997.

[3] D.Eastlake, "Domain name system security extensions", Technical Report DARPA-Internet RFC 2535,IBM, March 1999.

[4] F.Wang, "SITAR:A Scalable Intrusion Tolerance Archittecture for Distributed Service", Proceeding of the IEEE Workshop on Information Assurance and Security, New York, pp.38-45,June 2001.

[5] S.Brant, F.Wang, "Design and Implementation of Adaptive Reconfiguration for Intrusion Tolerant Systems", International Conference on Dependable Systems and Networks, Washington D.C.,pp.253-306, June 2002.

[6] Luenam, P. and P. Liu, "The Design of an Adaptive Intrusion Tolerant Database System", Foundations of Intrusion Tolerant Systems: IEEE Computer Society Press, 2003.

[7] Partha P.Pal,Franklin Webber, "Intrusion Tolerant Systems",Proceding of the IEEE Information Survivability Workshop,Boston,MA,pp.24-26,October 2000.

[8] Wu T,Malkin M,and Boneh D, "Building Intrusion Tolerant Applications", Proceeding of the 8th USENIX Security Symopsium, Washington D.C,pp.7-91,August 1999.

[9] L. Lamport, R. Shostak, M. Pease, "The Byzantine Generals Problem", ACM Transactions on Programming Languages and System, Vol 4, No.3, pp.382-401, Jul.1982.

[10] M. Castro and B. Liskov, "A Correctness Proof for a Practical Byzantine-Fault-Tolerant Replication Algorithm", Technical Memo MIT/LCS/TM-590, MIT Laboratory for Computer Science, 1999.

[11] M. Castro, "Practical Byzantine Fault Tolerance", Ph. D. Thesis, Massachusetts Institute of Technology, MA,Jan.2000.

[12] Miguel Castro and Barbara Liskov, "Practical Byzantine Fault Tolerance", Proceedings of the Third Symposium on Operating Systems Design and Implementation , New Orleans, pp. 173-186, February 1999.

[13] [Ben Hardekopf,Kevin Kwiat, "Secure and Fault-Tolerant Voting in Distributed Systems",Proceeding of IEEE Aerospace Conference, Montana, pp.1117-1127,March 2001.

[14] YU Yan-ping. "The research and design of intrusion tolerant system based on voting mechanism", Master. Thesis, Xidian University, Xi'an, May.2005.