Denial of Service Attacks and Defenses in Decentralized Trust Management

Jiangtao Li[1] Ninghui Li[1] XiaoFeng Wang[2] Ting Yu[3]

[1]Department of Computer Science, Purdue University, {jtli,ninghui}@cs.purdue.edu
[2]School of Informatics, Indiana University, xw7@indiana.edu
[3]Department of Computer Science, North Carolina State University, tyu@unity.ncsu.edu

Abstract

Trust management is an approach to scalable and flexible access control in decentralized systems. In trust management, a server often needs to evaluate a chain of credentials submitted by a client; this requires the server to perform multiple expensive digital signature verifications. In this paper, we study low-bandwidth Denial-of-Service (DoS) attacks that exploit the existence of trust management systems to deplete server resources. Although the threat of DoS attacks has been studied for some application-level protocols such as authentication protocols, we show that it is especially destructive for trust management systems. Exploiting the delegation feature in trust management languages, an attacker can forge a long credential chain to force a server to consume a large amount of computing resource. Using game theory as an analytic tool, we demonstrate that unprotected trust management servers will easily fall prey to a witty attacker who moves smartly. We report our empirical study of existing trust management systems, which manifests the gravity of this threat. We also propose a defense technique using credential caching, and show that it is effective in the presence of intelligent attackers.

1 Introduction

As the world is increasingly connected by the Internet, the need for entities from different security domains to dynamically collaborate, share resources and conduct sensitive transactions grows more and more important. Trust management (TM) [4, 5, 6, 9, 11, 14, 17, 20, 21, 22, 24] is an approach to enable such collaboration and resource sharing. In TM systems, an entity's privilege is based on its attributes instead of its domain-specific identities. An entity's attributes are demonstrated through digitally signed credentials. Delegation is an important mechanism for scalable and flexible trust management. Instead of relying on one or a few commonly trusted parties (e.g., certificate authorities), delegation allows each domain to autonomously determine who can access its resources and how such trust decisions can be propagated to entities from other domains; this nicely models complicated trust relationships between collaborating parties. Thus, in order to access a local resource, an entity from other domains submits a chain of credentials (or a set of credentials that form a graph) to prove its privileges. A TM server will check the authenticity of the submitted credentials and determine whether these credentials form the right proof of the legitimacy of the entity's access request.

In this paper, we study denial-of-service (DoS) attacks in TM systems. Recently, DoS attacks that exploit application-level vulnerabilities have become an increasing concern for Internet applications. Different from network-level DoS attacks, these attacks usually consume a small amount of bandwidth and therefore are more difficult to detect. For example, an attack that exploits the fact that an authentication server needs to perform expensive RSA decryption operations needs only a bandwidth of several Megabits/second to bring down a website [8]. Such an attack rate could easily be mingled into normal traffic. Compared with authentication systems, TM systems are even more susceptible to this type of DoS attacks. Like authentication systems, TM systems require a server to verify signatures, a computation-intensive operation. Most common secure sites cannot sustain more than 4,000 DSA verifications per second, even specially-designed hardware servers cannot perform more than 18,700 DSA verifications per second [2]. *Unlike authentication systems in which every service request leads to only a single or at most a few verification operations, TM systems allow delegation, which opens the door for an attacker to force a server to perform tens or even hundreds of verification operations with a single request.* The length of a credential chain can be arbitrarily large in theory. Because many credentials are not secret and are often sent in the clear, an attacker can obtain credentials both from public data and by eavesdropping. An attacker can also generate new public/private keys and create new credentials. Therefore, an attacker has a lot of freedom in carefully crafting credential chains to most effectively consume server resources. In addition, credential-chain based DoS is also stealthier than authentication based DoS. While 4000 concurrent SSL sessions may trigger the alarm on a website, 80 trust management sessions, each asking the server to verify 50 credentials, could appear less suspicious.

Automated trust negotiation (ATN) [28, 29, 30] adopts the basic TM approach but considers the fact that credentials may contain sensitive information and need protection just as resources do. ATN techniques enable strangers to establish trust in each other through cautious, iterative, bilateral disclosure of credentials and policies. Because ATN adopts the TM approach, it is subject to the same kind of DoS attacks as TM systems. The issue of DoS attacks in ATN systems has been discussed in a previous paper [25]; however, the gravity of the problem was not analyzed. The approach proposed in [25] is for the server to abort once it receives an irrelevant credential, and after that to ban the client at the firewall. This approach requires protection from external mechanisms such as a firewall. Even with this protection, a single malicious client can still force a server to spend significant resources, and a small number of coordinating malicious clients can bring down the server.

In this paper, we present the first systematic study of the problem of DoS attacks in TM systems. There are three types of resources involved in the interaction between the server and the clients, CPU, memory and bandwidth. Among them, CPU resource for signature verification is usually the bottleneck for TM systems, and thus becomes the focus of this research.

We illustrate that DoS attacks are a serious concern in two steps. We first present a qualitative study that identify the DoS vulnerabilities in existing trust management systems. We choose KeyNote [4] and TrustBuilder [29] as the two sample systems to study. The former is the most mature and efficient publicly available implementation of TM systems that we are able to find, while the latter is the only trust negotiation prototype system that we have access to. Similar vulnerabilities likely exist in other TM systems as well. In the second step, we build a multi-threaded server that uses KeyNote, and successfully launch a DoS attacks to bring down the server. In fact, the attack would have succeeded if we chose any other TM systems that we have looked at, as they have worse performance than KeyNote.

We analyze the credential verification strategies attackers and servers can use in DoS attacks against TM systems. The server's goal is to identify an attacker as quickly as possible; and the attacker's goal is to make the server verify as many as possible. Using game theory, we show that the equilibrium strategy pair is randomized, and the expected number of credentials that need to be verified is on average half of the length of the credential chain. This means that, even if a TM server adopts the best verification strategy, if the attacker moves intelligently, it becomes impossible for the server to determine the legitimacy of a service request before committing a substantial amount of computing resources. This illustrates the need to develop additional countermeasures defending against potential DoS attacks.

We propose the first countermeasure to this threat, credential caching with a caching strategy tailored specifically to defend against DoS attacks. Credential caching allows the server to cache valid credentials that have appeared before to reduce the cost of credential chain verifications. This approach has several interesting features. Credential caching improves the performance even when no DoS attacks are being carried out. In addition, credential caching implicitly enables legitimate users to help a trust management server defend against DoS attacks. The more legitimate users a trust management interacts with, the more valid and relevant credentials will be cached by the server, which will in turn significantly reduce the verification cost even if attackers submit forged credential chains. This feature is not available in existing countermeasures against DoS attacks. We empirically evaluate the effectiveness of this approach.

One may wonder that since TM systems have not been widely deployed yet in today's decentralized applications, whether it is worthwhile to investigate DoS attacks against them. We argue that DoS resilience is an important security requirement for decentralized authorization. The risk of DoS attacks needs to be carefully analyzed. Effective countermeasures should be studied and seamlessly integrated with the design of TM systems. Ignoring its potential vulnerabilities and waiting attackers to exploit them first will only incur much higher costs to fix the problem.

The rest of the paper is organized as follows. In section 2, we review existing work on TM systems and DoS attacks. In section 3, we conduct case studies of two existing trust management systems, and qualitatively analyze their vulnerability to DoS attacks. Our attack models and assumptions for DoS in trust management are presented in section 4. In section 5, we use game theory as a tool to analyze the gravity of DoS attacks against trust management. We propose in section 6 credential caching as a countermeasure. An empirical evaluation of the effectiveness of credential caching against DoS attacks is presented in section 7. We conclude the paper in section 8.

2 Related Work

A large amount of work has been done on trust management. The term trust management was coined by Blaze, Feigenbaum, and Lacy, in [5]. They also presented PolicyMaker, the first design of a trust management system. Blaze et al. later introduced KeyNote [4], which includes a well-defined format and semantics for credentials and policies. SDSI (Simple Distributed Security Infrastructure) [24] and SPKI (Simple Public Key Infrastructure) were two public key infrastructures which also support delegations. They were later merged into a unified framework [6, 11]. Li et al. introduced Delegation Logic [20], a logic-based trust management language, and RT [21], a family of Role-based Trust-management languages that combine features from trust management and role-based access control. Other trust management languages that have appeared over the years include the Query Certificate Managers [14], the Secure Dynamically Distributed Datalog language [17], and the Binder language [9].

All the above work assumes uni-directional trust establishment, i.e., the service provider is trusted, and only the client has to show its credentials to prove its privilege. In the approach of automated trust negotiation [28, 29], mechanisms for mutual trust establishment were proposed, which support the protection of the contents of the client's credentials as well. Hess et al. [16] proposed the Trust Negotiation in TLS (TNT) protocol to integrate trust negotiation into the SSL/TLS handshake protocol.

Denial-of-service attacks and defense have been studied for two decades. Most existing work, however, focuses on the high-bandwidth network DoS, in which attackers produce a large volume of attack traffic to saturate the victim's links. By comparison, low-bandwidth DoS attacks are easier to launch and less visible to the victim, but equally destructive. Most of these attacks exploit application-level vulnerabilities. For example, stack smashing[1] and the ping-of-death attack[2] crash an Internet server by overflowing vulnerable buffers inside the server software. Some attacks can exploit algorithmic weaknesses: Crosby and Wallach's work [7] shows that carefully crafted inputs could degrade hash tables to linked lists, and thus force a web proxy to run at its worse-case performance.

An important type of low-bandwidth DoS attacks targets at authentication protocols. Authentication relies on resource-consuming public-key decryption. Attackers can send a large number of messages with bogus signatures to deplete an authentication server's CPU cycles. Meadows [23], Aura et al [3] and Dean and Stubblefield [8] have pointed out this problem.

A potential defense against authentication-based DoS is incremental authentication which requires a weak but high-speed authentication first and a stronger authentication later [23]. An alternative is client puzzles which ask the client to solve a puzzle and prove to the server its work before authentication begins. An inherent weakness of many Internet applications is that attackers may consume significant server resources at little cost. Client puzzles strive to improve this situation: the client is required to commit resources before the server does. This technique has been used to mitigate DoS threats to network protocols [18, 26, 27]. Aura et al. first introduces it to authentication protocols [3] and its effectiveness has been empirically evaluated by Dean and Stubblefield [8], using TLS as an example.

To the best of our knowledge, the only work discussing DoS in trust management is by Ryutov et al. [25]. Several heuristics, such as the server's load, the relevancy of credentials, and the number of rounds of credential exchanges, have been used to identify potential DoS attacks as well as other abnormal activities. They, however, do not provide a systematic analysis of the problem. Also, there are no evaluations of the gravity of DoS attacks and the effectiveness of their proposed heuristics.

3 Denial of Service Vulnerabilities in Trust Management Systems

In this section, we analyze the denial of service vulnerabilities in existing trust management systems. After comparing with several candidates, we choose to analyze KeyNote [4] and TrustBuilder [29]. The former is the most mature and efficient publicly available implementation of trust management that we are able to find, while the latter is the only trust negotiation prototype system that we have access to. Our analysis in this section is qualitative. In Section 7, we build a multi-threaded server that runs the KeyNote program, and successfully launch a DoS attacks to disable the server.

3.1 Case Study: KeyNote

KeyNote [4] is a simple and flexible trust management system designed to work well for a variety of Internet-based applications. It has been published as Internet RFC 2704 in 1999. In KeyNote, both policies and credentials are modeled as assertions, which contain predicates that describe the trusted actions allowed by the key holders. KeyNote credentials have the same syntax as KeyNote policies, but are signed by the principal delegating the trust.

The latest implementation of KeyNote, version 2.3 [19], contains a command line tool and a reference library. There are four basic functions in the KeyNote library: key generation, signature generation, signature verification, and request evaluation. The key generation function can be used to generate a pair of public and private keys. The signature generation and verification functions are used to sign cre-

[1] `http://www.phrack.org/show.php?p=49&a=14`

[2] `http://www.insecure.org/sploits/ping-o-death.html`

dentials and verify credentials. The request evaluation function are used to determine whether an action request should be granted or denied, given a set of assertions (i.e., policies and credentials). The KeyNote reference library is written in C, and is very efficient. In our experiments on a 2.53GHz Intel Pentium 4 machine with 384MB RAM running RedHat Linux 9.0, the speed of a credential verification is about 3.5ms and 0.1 ms for DSA signature algorithm and RSA signature algorithm, respectively, with 1024 bit key length. In the DSA setting, verifying a credential chain length of 10, 100, and 1000 takes about 35ms, 354ms, and 3.77s, respectively. In the RSA setting, verifying a credential chain length of 10, 100, and 1000 takes about 1.2ms, 9.7ms, and 101ms, respectively. We do realize that RSA verification is much faster than DSA verification (about 20-40 times faster depending on the settings [1]). Nevertheless, DSA, a United States Federal Government standard for digital signatures, is still popular among many security services.

In a typical setup of KeyNote, whenever a client wants to access a resource, it needs to connect to an authorization server and submits an action request along with a set of credentials. The authorization server will evaluate the legitimacy of the client's request according to the submitted credentials and the server's policies using the request evaluation function provided by the KeyNote API.

We observe the following DoS vulnerabilities in authorization servers using KeyNote:

- *No upper bound on the number of credentials*. Since KeyNote only provides a reference library instead of a complete software package for the implementation of an authorization server, the request evaluation function itself does not impose any upper bounds on the number of credentials that it will accept. If an authorization server uses the library as it is without considering possible DoS attacks, then an attacker can send an arbitrarily large number of credentials to the server, exhausting its computational resources.

- *Not fail-stop during signature verification*. In fail-stop model [13], whenever a party detects any deviation from the protocol by the other participant, it terminates the communication immediately. The KeyNote implementation does not adopt the fail-stop strategy when verifying credential signatures. If a client sends a set of credentials along with an action request, the request evaluation function will verify the signatures of all credentials even when some invalid signatures have been detected. The reason of this design may be, according to the trust management semantics, even if one credential fails to verify, as long as among all credentials submitted there exist a valid chain, the authorization should still be allowed. However, this design, while logically correct, may be exploited by malicious clients to launch DoS attacks, even if it does not possess any valid credentials.

- *Asymmetric computational load*. The computational cost for the server is much higher than the cost of a client, as the server needs to perform signature verifications whereas the client simply needs to send credentials to the server. An attacker may continuously send action requests and credentials to the server, exhausting its computational resources.

  Note that this property is not specific to servers using KeyNote. Instead, it holds for any credential-based authorization server. However, since KeyNote request evaluation function does not use the fail-stop strategy, the vulnerability of servers using KeyNote is particularly severe.

Note that it is possible for the authorization server to verify the signatures of the credentials first before submitting them to the KeyNote engine. Then the server can decide whether to stop the authorization process if it receives a failed credential. However, as the KeyNote engine has already provided the functionality of credential verification, it would be intuitive for users to rely on the KeyNote engine to perform both credential verification and policy evaluation, which however may be subject to DoS attacks. Further, even if credential verification is done by applications and fail-stop is adopted, a malicious client can send valid but irrelevant credentials to the server to launch a new DoS attack. Therefore, it is desired to integrate the DoS defense mechanism directly into the KeyNote engine.

3.2 Case Study: TrustBuilder

TrustBuilder [29] is the first implementation of ATN. It was designed and developed by researchers at Internet Security Research Lab at Brigham Young University. TrustBuilder was developed in Java, and uses X.509v3 certificates. It adopts the policy language and policy compliance checker in the IBM Research's Trust Establishment system [15]. We ran our experiments on TrustBuilder on a 1.70GHz Intel Pentium M processor with 768MB RAM running Windows XP Professional. Each X.509 credential is signed using the RSA signature algorithm with 1024-bit key length. In our experiments, a client and a server, both running TrustBuilder, communicate with each other using a TCP socket. A trust negotiation involving the verification of a credential chain with 28 credentials takes about 4.5 seconds.

TrustBuilder has an upper-bound on the number of credentials received each round, e.g., the server can receive at most 30 credentials from the client at each step of the negotiation.

The latest version of TrustBuilder is vulnerable to DoS attacks in the following perspectives:

- *Verification of irrelevant credentials*. A client's credential is useful for a negotiation session only when it is relevant to the server's access control policy. In the current implementation of TrustBuilder, a server first verifies all the credentials received from the client, whether they are relevant to the negotiation or not. An attacker is thus able to send unrelated credentials to the server, making the server verify all of them. Note that this attack does not even require the attacker to forge credentials, since the attacker can always assume new identities by creating new public/private key pairs, and issue irrelevant credentials by using these identities.

- *Not fail-stop*. Same as the Keynote system, TrustBuilder does not use the fail-stop strategy. When the server receives an invalid credential, the server ignores the credential and continues the negotiation.

From the above two case studies we see that the current design of trust management systems does not take application-level DoS attacks into consideration. We show in Section 7 that, with the current design of trust management systems, DoS attacks can be easily launched to effectively deny the service of an authorization server.

4 Model and Assumptions in Analyzing DoS Attacks

We now describe the model and assumptions for analyzing DoS attacks in trust management. One important feature of TM and ATN systems is that trust is distributed among multiple principals and can be delegated from one principal to another. Here, we present a simple model to capture the delegation feature. In the model, there are multiple principals, each of which has a unique public-private key pair. Every policy is described by a trust tree. The trust tree corresponding to a server's policy will have the server as the root of the tree. A node of the tree represents a principal, and an edge represents a digital credential. Given an edge $v \rightarrow w$ in the tree, the corresponding credential means that the parent node v delegates the trust to the child node w. The credential is signed by the private key of v. The leaf nodes in the tree represent the principals who do not have delegation right, whereas the non-leaf nodes represent principals who can delegate trust to others.

For example, Alice is a student at StateU. She has a student credential from College of Science (CoS), which has a credential issued by StateU. StateU is certified by Accreditation Board for Engineering and Technology (ABET). The credential chain to prove that Alice is a valid student takes the form $\mathrm{root} \rightarrow \mathrm{ABET} \rightarrow \mathrm{StateU} \rightarrow \mathrm{CoS} \rightarrow \mathrm{Alice}$. There are four credentials associated with this chain. Suppose the server's policy is that students can access the resource. Alice could show the credential chain to the server, proving that she satisfies the policy.

When a client and a server begin interaction, they first establish a secure communication channel so that they can verify each other's identity. Such a channel can be established using, for example, TLS/SSL [10] with self-signed certificates. After such a channel is established, both client and server are certain that the other party holds the private key corresponding to the claimed identity (i.e., public key). After the session is established, the server can determine whether the client possess credentials that satisfy the server's policy for the requested resource.

Attackers Assumptions and Strategies There are two kinds of attackers: insiders and outsiders. An insider attacker is someone who has a valid credential chain and can legally access the server. An outsider attacker does not have a valid credential chain to gain access, and aims at bringing down the server. Within the insider attackers, some of the insiders has right to delegate (i.e., can issue arbitrary new valid credential chains), others do not have such rights (e.g., insiders that belong to the leaf nodes of the credential tree). The first type of insider attackers are rare and are hard to detect, we do not consider them in this paper and leave it as future work. For the second type, an insider attacker may constantly connect and disconnect from the server, making the server verify its credential chain again and again. It can behave exactly as a regular principal within each session. Insider attackers can be addressed in several ways, such as session caching, limiting the number of sessions, and deterrence measures such as revoking access once DoS behavior is detected. Because the number of insiders is limited and the possibility of using deterrence measures, we believe that outsiders are more serious concerns in DoS attacks. We thus focus our analysis on outsiders. We note that the credential caching techniques we propose in Section 6 can be used in defending against insider attackers as well as outsider attackers.

As an outsider attacker does not satisfy the server's policy, it does not control any node in the trust tree corresponding to the server's policy. However, the attacker can collect a large number of credentials in the trust tree and can forge arbitrary new credentials. An outside attacker can carry out the following attacks.

1. Sending a large number of valid, but irrelevant credentials.
2. Sending a valid credential chain of another principal.
3. Sending a credential chain from the trust root to itself; however, some credentials in the chain are faked, i.e., have signatures that do not verify. We observe that an attacker only needs to have one faked credential in such a chain: The attacker first collects a valid credential chain $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k$ where v_0 is the trust

root principal and v_k is principal who has authorized access. The attacker can then generate $n-k+1$ new principals $v'_k, \ldots, v'_n$, and creates a credential chain $v'_k \rightarrow v'_{k+1} \rightarrow \cdots \rightarrow v'_n$; however, to link the above two chains together, the attacker has to fake the credential $v_{k-1} \rightarrow v'_k$, as it does not control the principal v_{k-1}. Note that all the credentials except the one from v_{k-1} to v'_k in the chain could be valid. We also observe that in this case, the credential chain can be arbitrarily long.

Server Strategies: We now describe the strategies a TM server should use when facing the threat of DoS attacks.

1. The server does not verify any credential until it receives a complete credential chain that connects to the trust root at one end and to the client identity (that has been authenticated during the establishment of the communication channel) at the other end. In other words, the server starts verifying credentials in a chain only after it is certain that if all credentials in the chain are valid, then the client will gain access. This prevents against attacker strategies such as sending irrelevant credentials and sending other principals' valid credential chains.

2. The server sets an upper bound for the length of a credential chain. If the server receives a credential chain longer than that limit, it refuses to process the chain. Because the delegation feature in trust management allows local autonomy, the server may not know the length of all valid chains. Choosing the right bound becomes a tradeoff between DoS threats and cutting off some legitimate principals.

3. The server uses fail-stop model [13] for credential chain verification, that is, when the server finds one invalid credential, it stops verifying the chain and terminates the communication session.

4. Given a credential chain, the server may not necessarily verify the chain from the beginning to the end. We will further discuss this in next section.

By using the above strategies, the only attacker strategy that remains effective is the one of sending a chain that connects the trust root to the attacker, but has one faked credential in the chain. We analyze this situation in the next section.

5 A Game Theoretical Analysis on Credential Chain Verification

Using the model in the previous section, we study the following problem: the attacker tries to deplete the server's CPU cycles by having the server verify a long credential chain faked by the attacker. This problem, which we call the *credential chain verification problem*, can be specified as follows. The attacker sends to the server a credential chain of length n, in which one credential is invalid (i.e., with invalid signature). Assuming the attacker can place the invalid credential in any position of the chain, it has to determine where to place it so that the number of signature verifications performed by the server is maximized. On the other hand, given a set of credentials, the server intends to find an optimal strategy to verify the credential chain such that the number of signature verifications is minimized. This game can be modeled as a two-player zero-sum game (see Definition 1), as the interests of the attacker and the server are diametrically opposite.

Definition 1 (Two-player zero-sum game) *In the two-player zero-sum game (also called matrix game), there are two players A and B. Player A has n strategies and player B has m strategies. The strategies chosen by the two players determine the outcome of the game. Each possible outcome has two payoffs, one for each player; and the sum of the payoffs is always zero. We use an $n \times m$ matrix to represent the payoffs of A for each of the $n \cdot m$ possible outcomes (the payoffs of B are the opposite of A's, thus are not presented in the matrix).*

We assume that the server fixes its upper bound on the length of acceptable credential chains to n and the attacker also knows this bound. In the matrix game for our credential verification problem, the attacker has n possible strategies, the ith strategy placing the invalid credential to the ith position of the chain. The server has $n!$ possible strategies, each of which corresponds to a unique verification order (or a permutation of $\{1, \ldots, n\}$). For example, the server's strategy $(x_1, \ldots, x_n)$ represents a verification order, i.e., the server first verifies the x_1th position, then verifies the x_2th position, and so on. The payoff for the attacker is the number of signature verifications the server performs before it detects the invalid credential, which the attacker tries to maximize while the server tries to minimize. This game is called "verification game". Figure 1 describes a verification game where the length of the credential chain is 3.

The above game has a "solution" in which both players' strategies are optimal. This has been formally described in the following minimax theorem.

Theorem 1 (Minimax Theorem) *[12] Every $n \times m$ matrix game has a solution. that is, there is a value of the game v, and there are optimal strategies for players A and B such that (1) if A plays its optimal strategies, A's expected payoff will be $\geq v$, no matter what B does, and (2) if B plays its optimal strategies, A's expected payoff will be $\leq v$, no matter what A does.*

	(123)	(132)	(213)	(231)	(312)	(321)
1	1	1	2	3	2	3
2	2	3	1	1	3	2
3	3	2	3	2	1	1

Figure 1. The verification game with $n = 3$. The rows in the matrix stand for the attacker's strategy and the columns in the matrix stand for the server's strategy. Each entry in the matrix stands of the payoff of the attacker, i.e., the number of signature verification before the server detects the invalid credential.

In the above game, no matter which strategy the attacker chooses, there is always a corresponding strategy for the server such that the payoff for the attacker is minimized (i.e., to 1); similarly, given any strategy chosen by the server, there is always a strategy for the attacker to maximize its payoff (i.e., to n). This suggests that an optimal strategy here must be probabilistic, in the sense that either player will randomly choose their strategies according to some probability distribution. Such a probabilistic strategy is also called "mixed strategy" in game theory.

In the following proposition, we show that the verification game has a simple solution.

Proposition 2 *In the verification game, an optimal mixed strategy for the attacker is to play each possible strategy with $1/n$ probability, an optimal mixed strategy for the server is to play the strategy $(y_1, \ldots, y_n)$ with the probability of $1/2$ and $(y_n, \ldots, y_1)$ with the probability of $1/2$, where $(y_1, \ldots, y_n)$ can be any permutation to array $(1, \ldots, n)$.*

The proof of the above proposition is omitted due to the space limit. We will give the proof in the full version of this paper.

The above proposition shows that an optimal strategy for the attacker is to randomly pick a position between 1 to n to place the invalid credential. An optimal strategy for the server is to first determine an order of all the credentials, and then choose to verify the credential chain by that order or by the reverse order with the same probability. Note that the preceding optimal strategy is not the only possible optimal strategy. For example, the mixed strategy that the server uniformly randomly picks a strategy from the $n!$ possible strategy is also optimal.

Conclusion of this game Our analysis shows that if a server sets its upper bound to n, then a strategic attacker can make the server verify $(n + 1)/2$ credentials on the average, no matter what verification strategy the server uses. This demonstrates that, without further protection mechanisms, trust management systems are vulnerable to denial of server attacks.

6 Defense: Caching Verified Credentials

In this section, we present a simple defense mechanism against DoS attacks in TM systems: the server caches a credential in the memory once it verifies the signature successfully. When a server needs to verify a credential, it first checks whether the credential has been cached. If so, it does not need to perform the signature verification computation. To make this defense mechanism effective, we have to answer the following questions: What strategy the server should use to verify a credential chain when a cache exists? When the cache is full, which credential should be discarded? When answering this question, we have to keep in mind that the attacker will try to decrease the effectiveness of caching by cleverly ordering the credentials to be presented.

```
Credential chain verification with unlimited cache
Input:
    $c_1, \ldots, c_n$: the credential chain in order.
    $D$: the database of cached credentials.
Output:
    true or false: the result of the verification.
    $D$: the updated credential database.
Procedure:
    For $i = 1, \ldots, n$
        Compute $h_i = hash(c_i)$,
        If $h_i \notin D$, verify the signature of $c_i$
            If $c_i$ is invalid, return false,
            Otherwise $D = D \cup h_i$,
    Return true.
```

Figure 2. Pseudocode for credential chain verification using cache

To save memory, the server stores the hashes of the credentials rather than the credentials themselves. We use D to denote the database of cached credentials. The algorithm for credential chain verification is presented in Figure 2. Note that in the algorithm the server always verifies the credential chain from the trust root to the client. This is different from the optimal strategy we derived in the previous section. The presence of a cache changed the optimal strategy. There are two reasons that one should verify the credential chain in order. First, if the server verifies the credential chain from the trust root, only credentials in the trust tree can be added into the database. Recall that the attacker can create new principals and issues credentials between these principals. Since such credentials do not belong to the trust tree, they will not be cached in the credential database. An attacker thus cannot fill the database with irrelevant credentials. Second, when the server caches enough valid credentials, the number of credential verifications needed is minimized when the server verifies the credential chain starting from the trust root.

We observe that if all credentials in the trust tree that have been collected by an attacker is cached, then the server can detect any faked credential chain within one signature verification. Given a credential chain $c_1, \ldots, c_n$ created by the attacker, there exists a $k \in [1..n]$ such that c_k is an invalid credential, $c_1, \ldots, c_{k-1}$ are valid credentials in S, and $c_{k+1}, \ldots, c_n$ are valid credentials but not in S. Clearly, when the server verifies the chain from the beginning, $c_1, \ldots, c_{k-1}$ are all cached and do not need to be verified. Therefore, the server only needs to verify the signature of c_k before it detects the faked chain.

In many scenarios, the server does not have enough memory to store all the credentials in the trust tree. It thus needs to decide which cached credentials to be replaced when the cache is full. One naive approach is to use well-known memory caching replacement strategies, such as LFU and LRU. However, these caching strategies may not be optimal in our model, as the traditional caching strategies focus on overall performance for normal users, whereas in our model we need to consider the worst case scenario launched by a sophisticated attacker.

To design a caching strategy that makes DoS attacks less effective, we first give some theoretical analysis, assuming that we know T, the trust tree corresponding to the server's policy. Let S be the set of all credentials in the tree. Let m be size of the server's cache, i.e., the server can store at most m credentials. Given an edge $u \rightarrow v$ in the tree, let c be the corresponding credential. We use $h(c)$ to denote the height of c, which is the distance between v and the farthest descendant of v. In our strategy, the server only keeps credentials whose height is *larger* than some threshold. More specifically, let $S_i = \{c \in S \mid h(c) \geq i\}$. Clearly, $S_0 = S$ and $S_{i+1} \subseteq S_i$ for any positive i. Let σ be the threshold such that $|S_\sigma| \leq m$ and $|S_{\sigma-1}| > m$. The best caching strategy for the server is to cache all the credentials in S_σ and as many credentials in $S_{\sigma-1}$ as one can. We now prove that this strategy is optimal for the server.

Claim 1 *If the server caches S_σ, then no matter how the attacker chooses its strategy, the number of credential verifications needed by the server is bounded by $\sigma + 1$.*

Proof. Recall that the attacker creates a faked credential chain and sends it to the server for verification. Suppose the attacker collects all the credentials in S. To fake a credential chain of length n, the attacker chooses a node v_k in the tree with depth k. Let $c_1, \ldots, c_k$ denote the credentials in the path from the root v_0 to v_k. The attacker then fakes a credential c_{k+1} and creates $n - k - 1$ valid credentials $c_{k+2}, \ldots, c_n$. Observe that, for $1 \leq i \leq k$, $d(c_i)$ is greater than or equal to $k - i$. Therefore $c_1, \ldots, c_{k-\sigma} \in S_\sigma$. When the server verifies the credential chain, the server does not need to verify $c_1, \ldots, c_{k-\sigma}$, as they are already stored in the cache. The only credentials potentially need to be verified by the server are $c_{k-\sigma+1}, \ldots, c_{k+1}$. Thus, the number of credential verifications is bounded by $\sigma + 1$. ∎

Claim 2 *Suppose $|S_\sigma| = m$. If the server caches any credentials not in S_σ, then it is possible for an attacker to construct a credential chain such that the server has to perform more than $\sigma + 1$ credential verification.*

Proof. Since $|S_\sigma| = m$, if the server caches any credentials not in S_σ, then at least on credential in S_σ is not cached. Let c be such a credential. Since $d(c) \geq \sigma$, there exists a valid credential chain $c_1, \ldots, c_i = c, \ldots, c_k$ such that $k - i \geq \sigma$. Then if the attacker submits $c_1, \ldots, c_k, c'$, where c' is a credential with a forged signature, then the server at least has to verify the validity of credentials $c_i, \ldots, c_k, c'$, which involves more than $\sigma + 1$ signature verifications. ∎

The above two claims show the optimality of caching S_σ. When the server knows the trust tree, it can statically compute S_σ and achieve this optimality easily. However, in reality, the server does not know the whole trust tree, since a large part of the tree is constructed by other domains which directly or indirectly get delegation from the server. Thus, it cannot pre-compute S_σ. We need to let the server dynamically adjust the cached credentials so that the database can approximate S_σ.

To achieve this goal, we propose a new caching strategy where the server keeps the credential hash along with its height. If the cache becomes full, the server replaces the credential that has the smallest height. If the height of a newly verified credential is smaller than the height of any existing credential in the cache, then no replacement will take place. The caching algorithm is presented in Figure 3.

The caching algorithm in Figure 3 takes two steps. In the first step, the server finds the credential that is invalid by going through the credentials in the chain one by one. In the second step, if the credential c_i is already in the database, the server updates its height (if needed) by setting the $e(c_i) = \max(e(c_i), k - i)$. If the credential c_i is not in the database, the server inserts the credential hash in the database along with the height $e(c_i)$. The server can implement this caching algorithm using a hash table for the credential hashes and a priority queue for the heights. We shall discuss the implementation issues in details in next section.

Benefits and Limitations of Credential Caching The credential caching approach has the following advantages.

- Credential caching is beneficial even when there are no DoS attacks. Many legitimate credential chains share

```
Credential chain verification with limited cache
Input:
    $c_1, \ldots, c_n$: the credential chain in order.
    $D$: the database of cached credentials.
Output:
    true or false: the result of the verification.
    $D$: the updated credential database.
Procedure:
    Set $k = n$,
    For $i = 1, \ldots, n$
        Compute $h_i = hash(c_i)$,
        If $h_i \notin D$, verify the signature of $c_i$,
            If $c_i$ is invalid, $k = i - 1$, break;
    For $i = 1, \ldots, k$
        If there exists an entry $\langle h_i, e \rangle \in D$, update
            the entry with $\langle h_i, \max(e, k - i) \rangle$,
        Otherwise, insert $\langle h_i, k - i \rangle$ into $D$;
        If $|D| > m$, remove those tuples from $D$
            that has the smallest height.
    If $k = n$, return true, otherwise return false.
```

Figure 3. Pseudocode for credential chain verification with cache replacement strategy

common credentials. For example, consider the credential chains proving university students, if the credentials for all universities and colleges are cached, then verifying a student's credential chain only requires one more signature verification.

- Credential caching implicitly has legitimate users involved in defending against DoS attacks. Once a trust management server interacts with a legitimate user, all the user's valid credentials will be cached. These credentials will then not be helpful to attackers. Thus, the more legitimate users the server serves, the more resilient the server is against DoS attacks. This property is not observed in other countermeasures against DoS attacks.
- Credential caching allows the server to set a larger upper-bound on the length of credential chains that can be accepted. This allows the server to be able to handle unusually long valid credential chains. With credential caching, the upper bound does not affect the number of credentials to be verified by the server. This is because even when an attacker presents a very long chain, only those that are in the trust tree with at most one additional credential will be verified. This bound can thus be set based on other resource limitations, such as bandwidth concerns, and result in a larger bound.

We point out that credential caching cannot completely eliminate the DoS threat to a TM system. This is because a TM server will still have to verify at least one credential from the client. However, as we will discuss in the follow-up section, caching helps greatly mitigate the threat of DoS attacks, making them more difficult to happen. We believe that an effective combination of credential caching with other existing DoS countermeasures such as puzzles [3, 8] will make a TM system robust against DoS attacks.

Note that credentials can be expired or revoked. It is useful in practice to keep the expiration date along with the credential hash in the database. The server can then remove the credentials that have been expired from the database periodically. Given a credential, the server can easily lookup the database and check whether it has been cached or not. If the server wants to check whether the credential has been revoked, it may need to query the certificate revocation list with additional costs. We stress that the purpose of credential caching is to save the server from expensive signature verifications.

7 Experiment Results

In this section, we present the results of our empirical study on DoS attacks and defenses in trust management and negotiation systems. We first use KeyNote [4] as an example to demonstrate that a DoS attack can easily paralyze a trust management server (Section 7.1). Then, we evaluate the effectiveness of an implementation of credential caching system in mitigation of DoS threats (Section 7.2).

7.1 Multi-Client KeyNote Server

KeyNote is an open-source library for the KeyNote trust management system. To study the DoS vulnerability of KeyNote, we first built a multi-client KeyNote server using C, and then launched a DoS attack on that server. In our implementation, a client connects to the KeyNote server through TCP sockets. Each connection to the server is handled by a different thread. Upon receiving the public key and credentials from the client, the server calls KeyNote API to verify the credentials and to further check whether the whole credential chain satisfies the server's policy.

We carried out the experiments on a 2.53GHz Intel Pentium 4 machine with 384MB of RAM running RedHat Linux 9.0. We use 1024-bit DSA algorithm as the key generation and signature algorithm. The key file and credential file are stored in base64 format. The size of a credential is about 1.6KB. As discussed in Section 3, the KeyNote program is able to verify a credential in 3.5ms. In theory, the server can be paralyzed by less than 457KBps of traffic. In our experiments, a KeyNote client kept sending credential chains of length 20 to the server at the rate of 40 requests per second. Each request was about 32KB in size. As shown in Figure 4, the number of requests in the server's queue kept increasing until the server reached its limit on the number of allowed socket connections. The server had been completely disabled after 280 requests (7 seconds). Figure 5 illustrates the latency experienced by legitimate users who tried to connect to the server during and after a DoS attack. We assume that legitimate users connected to the server at a constant rate of 2 requests per second for 50 seconds. The DoS attack was launched at the first 10 seconds. From Figure 5, we can see that even after the DoS attack finished for 5 seconds, the latency for a legitimate request was still around 4 seconds. This is because the server was still busy at processing the pending credentials received from the attacker. These results demonstrate that an unprotected KeyNote server is indeed vulnerable to DoS attacks.

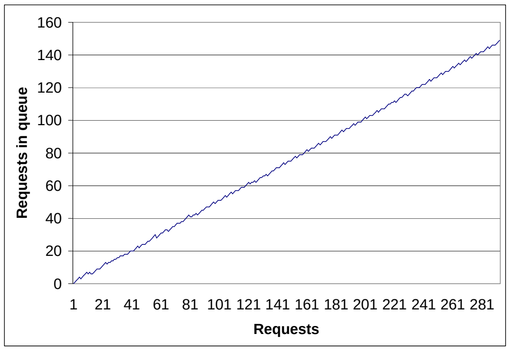


Figure 4. Number of requests in server's queue at each request during attack

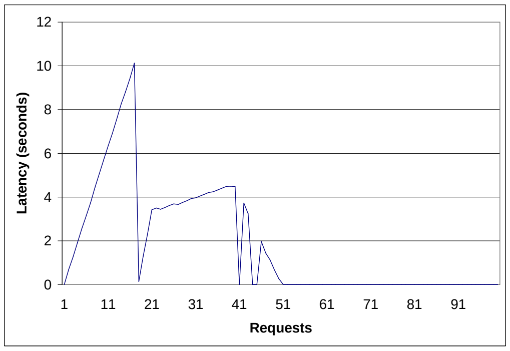


Figure 5. Latency for legitimate keynote clients during and after attack. The DoS attack was launched during the first 20 requests.

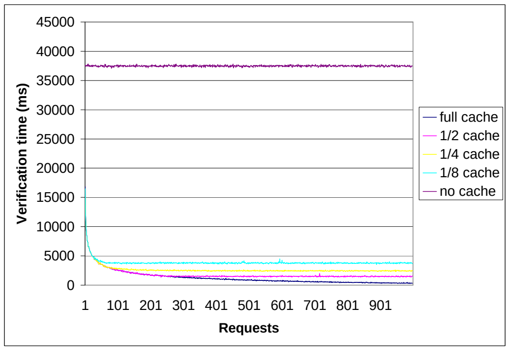
7.2 Credential Cache System

We implemented a credential cache system based on Figure 3 using C++. Our system maintains three data structures: a credential cache, a hash table, and a binary heap table. The credential cache is an array, each element of which stores the message digest of a credential that has been successfully verified. In addition, each element also stores a 16-bit number that is the height of the credential and two integers that index into the credential cache array. These indexes enable elements to form linked lists. The hash table is used to quickly determine whether the message digest of a credential is in the cache. Let m be the size of the cache, i.e., maximum number of credentials that can be cached, and ℓ be the length of binary representation of m. The size of the hash table is 2^ℓ, i.e., the size of the hash table is the smallest power of 2 that is greater than or equal to m. Given the message digest of a credential, the last ℓ bits of the message digest are used as a hash value that indexes into the hash table. All credential digests with the same hash value are organized into a linked list. The binary heap table is used to implement a priority queue, so that we can efficiently remove the credential with the smallest height from the cache when the cache is full. Each element in this table contains an index to the credential cache and a copy of the height value of the credential. This copy is used to improve access time of the heap. Assuming the message digest is 20 bytes long, then each element in the cache takes 30 bytes and each element in the heap takes 6 bytes. To cache 1M credentials, the credential cache and the heap takes 36M memory. The hash table has size less than 2M, and each element in the table takes 4 bytes. Therefore, we can cache 1M credentials using no more than 44MB memory.

When using caching, one should always verify a credential chain from the trust root to the leaf. Before actually verifying a credential's signature, one should first check whether the credential's message digest is in the cache.

In order to measure the effectiveness of the cache system, we simulate a trust tree. We create a tree with 500k node. With the probability of 0.5, each node has either 0 child or 1-8 children. The maximum depth of the tree is 16. For a randomly chosen node in the tree, the average distance to the root is 13.3. We create a credential for each edge of the tree.

Our experiments were carried out on a 2.53GHz Intel Pentium 4 machine with 384MB of RAM running RedHat Linux 9.0. Using OpenSSL's DSA implementation and benchmarks, the server can perform a 1024-bit signature verification in 2.81ms. In our experiment, we generated 1M requests for credential chain verification. For each request, we randomly picked a node from the tree, and passed the corresponding credential chain (from the root to the node) to the verification program. On average, it took 2.8ms to verify a non-cached credential and took less than 0.01ms to verify a cached credential. For every 1000 requests, we recorded the total verification time and the number of signature verification performed. It took about 37 seconds and 13.3k signature verifications to process 1000 requests without cache support. With the support of the caching system, the processing time dropped quickly. For example, in the case that the cache size was equal to the size of the tree, after processed 500k requests, the server only needed to verify 299 credentials out of 13266 credentials and hence the hitting rate was 97.7%. If the cache size was 1/8 of the tree size, the hitting rate after 500k requests also became more than 90.3%. A performance chart of the cache system is given in Figure 6. In the figure, "full cache" refers to the setting in which the cache size is the same as the tree size; similarly, "1/2 cache" means the cache size is half of the tree size. Note that the cost of the cache maintenance operation (e.g., insert, delete, update) is negligible comparing with the cost of credential verification.



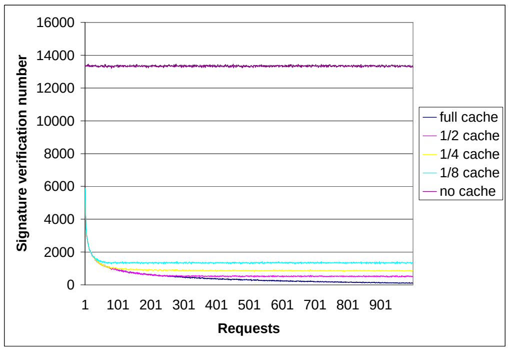


Figure 6. Credential verification time and numbers of signature verifications per each 1000 requests

8 Conclusion and Future Work

We analyzed the vulnerabilities of DoS attacks in trust management systems, and proposed credential caching as a countermeasure to such attacks. Empirical studies have showed that credential caching is an effective means of mitigating DoS attacks against TM servers. Future work includes integrating the credential caching system with KeyNote and TrustBuilder. As ATN protocols are more complicated than a single credential chain verification, future work also includes studying other DoS vulnerabilities in ATN systems and providing corresponding defense mechanisms.

Acknowledgement

This work is supported by NSF IIS-0430274, NSF CCR-0325951, and sponsors of CERIAS. We thank the anonymous reviewers for their helpful comments.

References

[1] Crypto benchmarks. *http://www.eskimo.com/∼weidai/benchmarks.html*.

[2] Server's benchmarks. *http://www.sun.com/servers/coolthreads/t1000/benchmarks.jsp*.

[3] T. Aura, P. Nikander, and J. Leiwo. Dos-resistant authentication with client puzzles. In *Proceedings of the Cambridge Security Protocols Workshop 2000*. LNCS, Springer-Verlag, 2000.

[4] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The KeyNote trust-management system, version 2. IETF RFC 2704, Sept. 1999.

[5] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, May 1996.

[6] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.

[7] S. A. Crosby and D. S. Wallach. Denial of service via algorithmic complexity attacks. *USENIX Security*, 2003.

[8] D. Dean and A. Stubblefield. Using client puzzles to protect tls. In *Proceedings of the 10th USENIX Security Symposium*. USENIX, Aug. 2001.

[9] J. DeTreville. Binder, a logic-based security language. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 105–113. IEEE Computer Society Press, May 2002.

[10] T. Dierks and C. Allen. The TLS Protocol Version 1.0, Jan. 1999. *http://www.ietf.org/rfc/rfc2246.txt*.

[11] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory. IETF RFC 2693, Sept. 1999.

[12] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 1991.

[13] L. Gong and P. Syverson. Fail-stop protocols: An approach to designing secure protocols. In *Proceedings of the 5th International Working Conference on Dependable Computing for Critical Applications*, September 1995.

[14] C. A. Gunter and T. Jim. Policy-directed certificate retrieval. *Software: Practice & Experience*, 30(15):1609–1640, Sept. 2000.

[15] A. Herzberg, Y. Mass, J. Mihaeli, D. Naor, and Y. Ravid. Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 2–14. IEEE Computer Society Press, May 2000.

[16] A. Hess, J. Jacobson, H. Mills, R. Wamsley, K. E. Seamons, and B. Smith. Advanced client/server authentication in TLS. In *Network and Distributed System Security Symposium*, pages 203–214, Feb. 2002.

[17] T. Jim. SD3: A trust management system with certified evaluation. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 106–115. IEEE Computer Society Press, May 2001.

[18] A. Juels and J. Brainard. Client puzzles: A cryptographic defense against connection depletion attacks. In *Proceedings of the 1999 Network and Distributed System Security Symposium*, February 1999.

[19] A. D. Keromytis. The KeyNote trust-management system. *http://www.cis.upenn.edu/∼angelos/keynote.html*.

[20] N. Li, B. N. Grosof, and J. Feigenbaum. Delegation Logic: A logic-based approach to distributed authorization. *ACM Transaction on Information and System Security*, 6(1):128–171, Feb. 2003.

[21] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.

[22] N. Li, W. H. Winsborough, and J. C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, Feb. 2003.

[23] C. Meadows. A Cost-Based Framework for Analysis of Denial of Service Networks. *Journal of Computer Security*, 9:143–164, 2001.

[24] R. L. Rivest and B. Lampson. SDSI — A simple distributed security infrastructure, Oct. 1996. *http://theory.lcs.mit.edu/∼rivest/sdsi11.html*.

[25] T. Ryutov, L. Zhou, C. Neuman, T. Leithead, and K. E. Seamons. Adaptive trust negotiation and access control. In *Proceedings of the tenth ACM symposium on Access control models and technologies (SACMT)*, pages 139–146, 2005.

[26] X. Wang and M. Reiter. Defending against denial-of-service attacks with puzzle auction. In *IEEE Symposium on Security and Privacy*, May 2003.

[27] X. Wang and M. Reiter. Mitigating bandwidth-exhaustion attacks using congestion puzzles. In *Proceedings of the 11th ACM conference on Computer and Communication Security*, November 2004.

[28] W. H. Winsborough, K. E. Seamons, and V. E. Jones. Automated trust negotiation. In *DARPA Information Survivability Conference and Exposition*, volume I, pages 88–102. IEEE Press, Jan. 2000.

[29] M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu. Negotiating trust on the web. *IEEE Internet Computing*, 6(6):30–37, November/December 2002.

[30] T. Yu, M. Winslett, and K. E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security*, 6(1):1–42, Feb. 2003.