Assignment 7.1: Kerberos Authentication

Joshua A. Wortz

University of San Diego

CSOL-510

Prof. Hoffman

06/22/2026

Kerberos Authentication Issues

Kerberos is a robust network authentication protocol, but its implementation in modern Active Directory (AD) environments suffers from critical vulnerabilities, primarily due to legacy configuration support and its inherent statelessness (Aksüt, 2024). Because Kerberos does not track which accounts need access to which services, an attacker with access to any low-privilege domain user account can request access rights to nearly any service (Aksüt, 2024). This structural flaw enables "Kerberoasting," an attack where an adversary requests service tickets for accounts with weak Service Principal Names (SPNs) and extracts the encrypted tickets from memory (Aksüt, 2024). If the ticket is encrypted using obsolete, backward-compatible cipher suites such as RC4, an attacker can crack the password offline without interacting with the target system or triggering traditional security alarms (Aksüt, 2024; Noureen, 2026). Furthermore, the protocol's reliance on a centralized Key Distribution Center (KDC) poses a severe risk; if an attacker compromises the KDC's krbtgt account hash, they can forge their own tickets to impersonate any user or service, leading to complete domain compromise (Aksüt, 2024; Park et al., n.d.).

Kerberos Authentication Flow Algorithm

The Kerberos protocol leverages a trusted third-party Key Distribution Center (KDC), which houses an Authentication Server (AS) and a Ticket Granting Server (TGS), to facilitate mutual authentication (Aksüt, 2024; Ram et al., 2024). The algorithmic flow consists of the following exchanges:

1. **Authentication Service Request (AS-REQ):**

$$\text{Client} \longrightarrow \text{AS} : \{C, S_{TGS}, T_{pre}\}_{K_C}$$

The client transmits its user identity (C), the requested Ticket-Granting Server ID (S_{TGS}), and a pre-authentication timestamp (T_{pre}) encrypted with the client's secret master key (K_C), derived from the user's password.

2. **Authentication Service Response (AS-REP):**

$$\text{AS} \longrightarrow \text{Client} : \{K_{C,TGS}\}_{K_C} \parallel \text{TGT}$$

$$\text{where TGT} = \{C, S_{TGS}, T_{expire}, K_{C,TGS}\}_{K_{KDC}}$$

The AS decrypts and verifies the timestamp. Upon validation, it generates a unique, short-lived TGS session key ($K_{C,TGS}$) encrypted with the client's master key, and appends the Ticket-Granting Ticket (TGT), which contains the client identity and the session key encrypted with the KDC's master key (K_{KDC}).

3. **Ticket-Granting Service Request (TGS-REQ):**

$$\text{Client} \longrightarrow \text{TGS} : \text{TGT} \parallel S_{App} \parallel \text{Authen}_C$$

$$\text{where Authen}_C = \{C, T_{stamp}\}_{K_{C,TGS}}$$

To access an application server (S_{App}), the client sends the encrypted TGT along with a newly generated **Authenticator** (Authen_C) containing the client ID and a current timestamp, encrypted using the fresh TGS session key ($K_{C,TGS}$).

4. **Ticket-Granting Service Response (TGS-REP):**

$$\text{TGS} \longrightarrow \text{Client} : \{K_{C,App}\}_{K_{C,TGS}} \parallel \text{ST}_{App}$$

$$\text{where } \text{ST}_{App} = \{C, S_{App}, T_{expire}, K_{C,App}\}_{K_{App}}$$

The TGS decrypts the TGT using K_{KDC}, extracts $K_{C,TGS}$, and uses it to decrypt the Authenticator to verify that the requesting principal matches the identity in the TGT. It then generates an application session key ($K_{C,App}$) encrypted with the TGS session key, alongside a Service Ticket (ST_{App}) encrypted with the application server's unique secret key (K_{App}).

5. **Application Request (AP-REQ):**

$$\text{Client} \longrightarrow \text{Application Server} : \text{ST}_{App} \parallel \text{Authen}_{App}$$

$$\text{where } \text{Authen}_{App} = \{C, T_{stamp2}\}_{K_{C,App}}$$

The client transmits the encrypted Service Ticket and a fresh **Authenticator** (Authen_{App}) encrypted with the application session key to prove real-time possession of the key.

6. **Application Response (AP-REP):**

$$\text{Application Server} \longrightarrow \text{Client} : \{T_{stamp2} + 1\}_{K_{C,App}}$$

The resource server decrypts the Service Ticket using its own key (K_{App}) to extract the session key ($K_{C,App}$). It decrypts the Authenticator, verifies the timestamp, increments it by one, encrypts it back with $K_{C,App}$, and returns it to prove its own identity to the client, concluding the handshake and establishing the secure channel.

Solutions with Real-Life Examples

To address Kerberos vulnerabilities, security teams deploy a mix of strict cryptographic enforcement and behavioral detection. A prominent real-world example of Kerberoasting occurred during the "Solarigate" (SolarWinds) intrusion and the Carbon Spider ransomware attacks, where threat actors used the technique to escalate privileges across compromised networks quietly (Aksüt, 2024).

One critical solution to prevent offline password cracking is the complete deprecation of weak legacy cryptography. For example, in 2026, Microsoft rolled out a mandatory update to eliminate RC4 encryption from the Kerberos protocol path in response to CVE-2026-20833, forcing domain controllers to default to stronger AES-SHA1 or AES-SHA256 encryption types (Noureen, 2026). Additionally, to detect attacks such as Kerberoasting that blend into normal traffic, organizations are implementing Supervised Machine Learning algorithms (such as Logistic Regression and Random Forest Classifiers) to analyze Event ID 4769 logs and detect anomalous spikes in Service Ticket requests (Aksüt, 2024).

Can Kerberos guarantee the integrity of data exchanged between principals?

Yes. Kerberos guarantees data integrity during the initial exchange. However, its ability to enforce data integrity for subsequent application traffic depends on specific cryptographic mechanisms and is constrained by architectural limitations.

- **Role of Session Keys and Authenticators:** During the authentication phase, integrity and anti-tampering are enforced through short-lived session keys generated by the KDC. Because network data can be intercepted, the protocol requires the client to generate an Authenticator—a packet containing identity

metadata and a precise timestamp encrypted with that specific session key. The server's ability to successfully decrypt this Authenticator and match it against the timestamp in the ticket mathematically proves that the data has not been modified or replayed in transit by an adversary. Formal mathematical verifications using tools such as AVISPA and the Otway-Rees Formal Model of Communication (OFMC) confirm that this combined architecture successfully mitigates Man-in-the-Middle (MIM) and Replay attacks.

- **Protocol Limitations:** A critical protocol limitation is that Kerberos is fundamentally an authentication architecture, not a persistent data-transit encapsulation layer. While it injects a secure session key into both principals, Kerberos does not automatically sign or encrypt the payload of all subsequent application data out of the box. To maintain integrity post-authentication, the application layer must explicitly utilize the established session keys via integration frameworks such as the Generic Security Services Application Program Interface (GSS-API) or native Kerberos messaging variants like KRB_SAFE (which appends data integrity checksums) and KRB_PRIV (which enforces full confidentiality).

Can it protect application server resources against unauthorized access?

Yes. Yes, it protects application server resources, but it does so by strictly serving as a reliable root of trust for identity validation, which must be clearly contextualized by separating authentication from authorization, alongside its inherent architectural limitations.

- **Authentication vs. Authorization:** To analyze how resources are protected, a firm boundary must be drawn between identity verification and access

enforcement. Kerberos is inherently an authentication protocol designed to prove who an entity is, rather than an authorization mechanism that dictates what that entity is permitted to do in an enterprise ecosystem such as Active Directory. Authorization data is handled by embedding a Privilege Attribute Certificate (PAC) in the service ticket issued by the KDC. This PAC contains the user's security identifiers (SIDs) and group memberships, signed cryptographically by the KDC.

- **Resource Protection & Limitations:** Resource protection is achieved because the application server refuses to accept a client's self-asserted identity. Instead, the client must present a Service Ticket encrypted with a secret key known only to the target server and the TGS. Successful decryption proves that the KDC validated the user. However, the core limitation of this model is that Kerberos merely serves as a secure transit utility for authorization metadata; the ultimate burden of resource protection falls entirely on the application server itself. The application must read the unpacked PAC data and map it against its local Access Control Lists (ACLs). If an application server is misconfigured, fails to properly evaluate PAC signatures, or has a local privilege escalation flaw, Kerberos cannot inherently prevent unauthorized access to those application resources.

References

Aksüt, Y. (2024). An analysis of Kerberoasting attack and detection with supervised machine learning algorithms (Master's thesis), Department of Cyber Security.

Noureen, R. (2026, January 27). Microsoft begins phasing out RC4 in Kerberos to block credential-cracking attacks. Petri IT Knowledgebase.

Park, A. T., Peck, N., Dill, R., Hodson, D. D., Grimaila, M. R., & Henry, W. C. (n.d.). Quantifying DDS-cerberus network control overhead. AFIT Scholar.

Ram, A., & Swarnendu. (2024). An authentication mechanism to prevent various security threats in software-defined networking by using AVISPA. Journal of Scientific & Industrial Research, 83(09), 977–988.