Attacking and fixing the Microsoft Windows Kerberos login service

Tommaso Malgherini and Riccardo Focardi

Università Ca' Foscari, Venezia

Abstract

We implement and test a recent attack called *pass-the-ticket* [2, 3] on various real Kerberos implementations. The attack allows a malicious user to physically login as a different one on a target host, under the assumption he is able to mount a man-in-the-middle attack between the attacked host and the Kerberos server. Our results are that all recent Microsoft Windows operating systems are vulnerable to the attack while the MIT Kerberos implementation version 1.6.3, tested on Linux, is not. We have reported through CERT [4] the vulnerability to Microsoft that will fix it in the next service pack.

NOTE: this paper reports the work developed by Tommaso Malgherini in his BSc thesis [6] under the supervision of Prof. Riccardo Focardi.

1 Introduction

The Kerberos authentication system was developed at MIT as part of the ATHENA project [7, 14]. It has become widely used after its adoption from Microsoft as the default network authentication protocol, from Windows 2000 onward. In the course of time, the protocol has been subjected to extensive review and analysis leading to the addition of new features and the corrections of various weaknesses (see for example [1]). Nevertheless, some issues regarding the protocol implementation for specific application or types of services still remain open. In particular, in [2, 3] the author presents a new attack technique, called *pass-the-ticket*, which combines previous well-known approaches to obtain complete authentication using fake credentials.

We have implemented the pass-the-ticket attack and tested it on real implementations of the protocol. Our findings are that all the recent versions of the Microsoft Windows systems in use, including Windows XP, Windows Vista Enterprise and Windows 7 Professional, are vulnerable to the attack. As a consequence, an attacker with physical access to a target machine and able to mount a man-in-the-middle attack between such a machine and the Kerberos authentication server, can login as any other principal using an

arbitrary password of his choice. This is particularly disturbing given that Kerberos is the default network authentication protocol for such systems. The flaw is due to an excessive simplification of the protocol in the Microsoft implementation of the Kerberos login service. We have also tested that the MIT Kerberos implementation version 1.6.3 for Linux is instead immune to the attack.

The vulnerabilities we have found have been immediately reported to Microsoft through CERT [4] and have been acknowledged and confirmed. They will be fixed in the next service pack.

The paper is organized as follows: in section 2 we give a quick review of the Kerberos protocol; section 3 describes known attacks and countermeasures, and discusses under which circumstances such attacks can still be considered a threat; section 4 reports the results of our vulnerability tests on adopted Kerberos implementations; we also illustrate why the MIT implementation is more robust than the Microsoft one, which suggests how to fix the latter; in section 5 we draw some concluding remarks.

2 The Kerberos Protocol

Kerberos is an authentication and key distribution protocol which uses a trusted third party, also called Key Distribution Service or KDC, to communicate the necessary cryptographic credentials to the parties involved in the authentication process. The scheme is entirely based on symmetric encryption (support for asymmetric cryptography was added later) and described in detail in [9]. The parties involved in the process are the following:

C, Client: the entity to be authenticated for a certain service;

AS, Authentication Service: the trusted third party authenticating the client to the TGS (below).

TGS, Ticket Granting Service: the trusted third party authenticating the client to the final service V (below). This phase is based on previous authentication from the AS.

V, Verifier: the final service, also called the Verifier.

It should be noted that while AS and TGS are conceptually distinct services, they are typically both implemented as functions of the KDC.

The protocol is composed of three request/response exchanges illustrated in figure 1 and described below. We write $\mathsf{E}_{\mathsf{K}}(\mathsf{x})$ to note message x encrypted with a symmetric cipher (including an integrity check) under key K; N_{C} represents a *nonce* chosen by C, T_{C} a timestamp from C clock; K_{U} is the personal key of the entity U, while k_{n} is a session key. L indicates a *lifetime*, the validity period of a specific ticket; a ticket

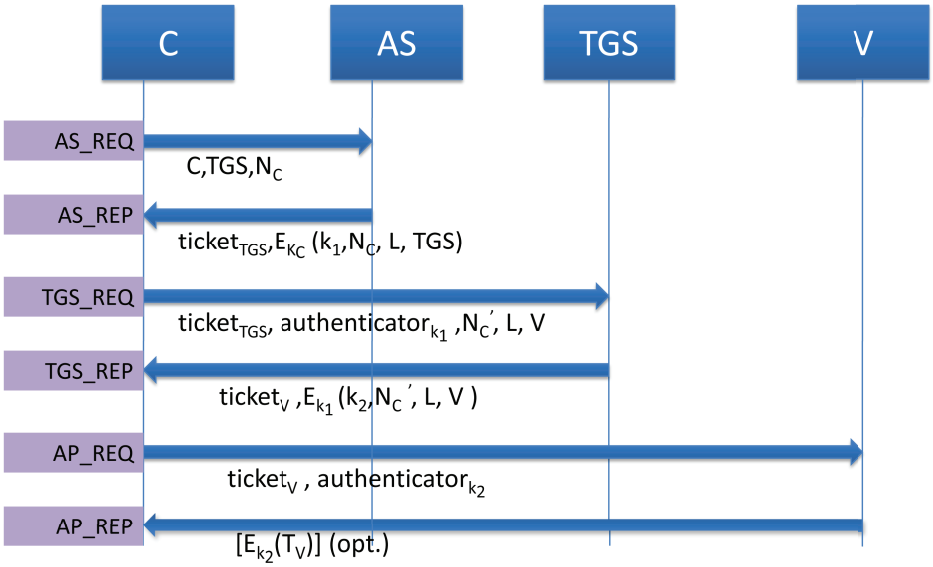


Figure 1: The Kerberos V protocol

is defined as $\mathsf{ticket}_U := \mathsf{E}_{K_U}(k_n, C, L)$, while an authenticator is noted as $\mathsf{authenticator}_{k_n} := (C, T_C, [k_n^1])$. Key k_n^1 is a sub-session key not chosen by the KDC which the parties can optionally agree on; as this component is irrelevant for the attacks presented, we will always omit it.

The three protocol exchanges work as follows:

AS_REQ/AS_REP: the client C contacts AS, from which she receives a *ticket* for the TGS, that will be used in the next phase to request a ticket for a service V. This ticket has the form $\mathsf{E}_{K_{TGS}}(k_1, C, L)$, where k_1 is a fresh session key to be used for subsequent communication between C and the TGS. This key is also sent to C encrypted under the client personal key K_C, which is derived from the user password through a key derivation function (for details on this and all cryptographic-related functions of the Kerberos system see [11]). Note that this ticket, like all tickets emitted, contains a lifetime value L, which limits its usage to a certain time window (typically 24 hours, for the TGS ticket). The (random) nonce N_C is checked by C so to prevent replays of old responses from the TGS that might fake C into re-accepting an old, possibly broken, session key;

TGS_REQ/TGS_REP: the client C forwards the received ticket to the TGS, together with an *authenticator* of the form $\mathsf{E}_{k_1}(C, T_C)$, containing a timestamp T_C. The TGS then decrypts the ticket, extracts the session key and if it can successfully decrypt the authenticator and

the timestamp is inside a certain time windows of acceptability, it assumes the client to be legitimate (since she knows the session key $\mathsf{k_1}$). The TGS sends, as a reply, a ticket for the specific service requested, together with a new session key $\mathsf{k_2}$ encrypted with $\mathsf{k_1}$. Even in this case, nonce $\mathsf{N'_C}$ prevents replays of old sessions.

Note that if the client needs access to a certain service and already possesses a valid TGS ticket, she can skip the first phase and just send a TGS_REQ, thus avoiding password reinsertion (Single-Sign-On), needed in the first phase for AS_REP decryption;

AP_REQ/AP_REQ: As in the previous step, the ticket received is sent with a new authenticator (this time encrypted with $\mathsf{k_2}$) to the service for which authentication is needed. If both these elements are verified as correct, the client is considered authenticated. If mutual authentication is requested, the service sends his own authenticator using the extracted key $\mathsf{k_2}$.

3 Attacks and countermeasures

Kerberos has been a publicly available protocol since 1989. A lot of weakness and limitations have been discussed in the course of years and, many of them, resolved or eliminated (see, e.g. [1]). There are two well-known attacks which, while not being a threat to the protocol itself, can represent, under specific circumstances, a real risk: *KDC Spoofing* (section 3.1) and *Replay* (section 3.2). A more elaborated attack, called *pass-the-ticket* [2, 3], is described in section 3.3. We have implemented and tested the pass-the-ticket attack on real systems, finding that it can lead to a complete authentication bypass on Windows operating systems (cf. section 4).

3.1 KDC Spoofing

This technique gained widespread knowledge after the release of the proof of concept code by Dug Song [13]. The attack is based on the possibility for a malicious user to spoof KDC responses. Since the entire Kerberos protocol has been designed to stand against an insecure network, this would not normally be a threat to the protocol. It can be seen, in fact, that while a KDC response message does not contain any data proving its authenticity to the client, only the true KDC can emit a ticket which will decrypt correctly with the service key, thus making the attack useless.

Nonetheless, there are cases under which such an attack can be of practical use to an intruder: the physical *login service*, for example, is a particular type of 'kerberized' service which is typically targeted by this attack technique. Kerberos services are usually daemons running on remote machines,

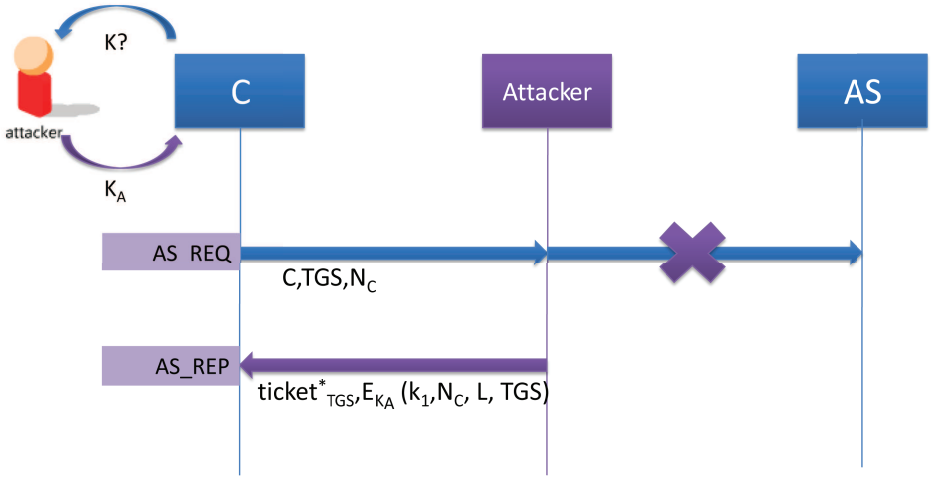


Figure 2: KDC Spoofing

requiring the client to prove her identity before accepting commands. Login service is different as there is no remote machine to send credentials to.

Various approaches has been followed to implement this service through Kerberos. On Unix/Linux systems, for example, the first versions of the pam_krb5 module used a shortcut: when the user entered her password, an AS_REQ request was made to the KDC, and when a AS_REP was received, PAM tried the decryption of the non-ticket part of the message with the key derived from user password. If the decryption and the nonce check were successful, the system assumed the password was correct and permitted the access. It can be seen that this procedure leads to a security breach, in which an attacker with physical access to the machine and the ability to manipulate network traffic inserts an arbitrary password, blocks the KDC reply, and insert his own AS_REP message encrypted with a key derived from the previously inserted password. The attack is depicted in figure 2. Notice that $\mathsf{ticket}^*_{\mathsf{TGS}}$ is a fake TGS ticket forged by the attacker and is just ignored by the client.

The obvious solution, which is what Windows and later versions of the PAM module do, is to implement also the next step of the protocol. A fictitious service for every machine in the network is created on the KDC, and the service key is imported on the corresponding machine. After the AS_REQ/AS_REP step, the client machine requests a ticket for the service represented by itself to the TGS. If a ticket which decrypts correctly with the service key is received, the authentication is considered completed, since only the KDC and the client machine have knowledge of the key. Given that $\mathsf{ticket}^*_{\mathsf{TGS}}$ has been forged by the attacker, TGS will refuse to provide the ticket for the service and authentication will consequently fail.

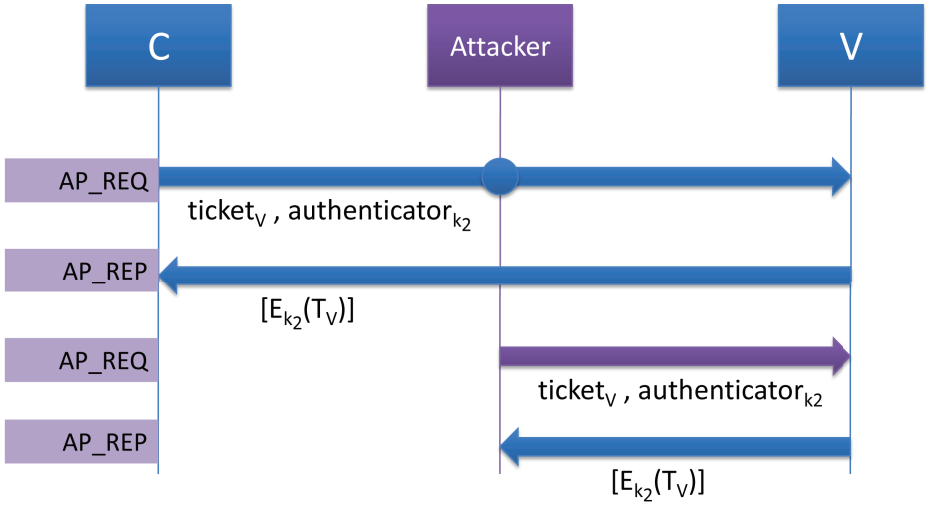


Figure 3: Replay Attack

3.2 Replay attacks

By observing the protocol scheme of figure 1, it can be seen that the server decides whether or not to allow access to the client by considering only the validity of the AP_REQ message. The protocol should thus prevent the possibility for an attacker to replay a previously sniffed AP_REQ message and thus impersonate the client on a specific service, as illustrated in figure 3.

This is a very old idea, already publicly discussed in [1] for Kerberos IV, and many countermeasures were thought to solve the problem, which are extensively described in the RFCs (see also [3] on this)

Timestamps: As discussed in section 2, the authenticator contains a timestamp T_C emitted during its creation. When received, the service verifies that the T_C falls inside a certain *time window* from its clock, which is typically 5 minutes. If this is not the case, the authentication request is refused.

Authenticators cache: In [10] it is explicitly indicated that verifiers should reject an already received authenticator. This is normally done by caching previous authenticators for a certain amount of time, which must be at least equal to the time window for acceptability. This makes it impossible to reuse any valid authenticator.

IP address field inside ticket: The KDC can optionally insert into tickets a list of IP addresses for which the ticket is valid ("address-full tickets"). This poses some difficulties mounting a replay attack as it becomes necessary to steal in some way the victim's IP address. However, in

practice, very few services actually check such addresses: according to [3], none in Microsoft Kerberos, and only KDC services in the MIT implementation.

The simple replay technique of figure 3 is clearly prevented by the aforementioned countermeasures. However, if the attacker is able to realize a man-in-the-middle (MITM), variants of such an attack can be successfully put into practice:

- Once the MITM is active, instead of letting the user authenticate, the attacker can block the communication, capturing the AP_REQ request and reusing it for himself before the expiration of the time window. It should be noted that the use of address-full ticket would make this attack useless without stealing in some way the victim's IP. As mentioned above, however, the check on IP addresses contained in the ticket is not usually implemented: in [5] this particular attack is tested on a Windows 2000 SP3 workstation, authenticating on a SMB server. The attack proves to be successful even in presence of address-full tickets, thus apparently confirming that Windows 2000 does not verify the list of allowed IP addressed contained in the ticket.

  Notice also that the session key k_2 exchanged between the client C and the service V is not leaked. In case k_2 is used to protect the confidentiality/integrity of subsequent message exchanges between C and V, the above attack becomes useless. This, however, depends on the specific implementation of each service.

- Kerberos heavily depends on time synchronization for checking the validity of timestamps, but it is not its responsibility to provide such a synchronization. The system normally employed to do so is *Network Time Protocol* (NTP). Even if this protocol provides a secure way to keep the clients in sync, such a feature is rarely used, and the server response containing the exact time can be modified by an intruder in the middle. The resulting attack is not immediate but makes it possible to capture an AP_REQ and reuse it even after its validity time has expired. It can be successfully mounted by (*i*) performing a MITM on the NTP server/verifier link, (*ii*) modifying the NTP response, and (*iii*) sending the captured AP_REQ (possibly awaiting for it to be deleted from the authenticators cache), thus obtaining access.

  As in the previous case, IP verification by using address-full tickets, if implemented, would easily block the attack in case the attacker is not able to steal the victim's IP. The adoption of a secure time synchronization protocol would also be desirable to prevent the attack.

3.3 *Pass-the-ticket* Attack

A more sophisticated attack, combining the above ideas, has been recently discussed [2, 3]. As in previous cases, the attack does not represent a threat for a correct implementation but we will show (in section 4) that in practice it can lead to a complete authentication bypass on real operating systems. This attack is designed to target the already discussed login service, when the extended login procedure presented at the end of section 3.1 is used. Recall that this extended procedure also contains the TGS_REQ/TGS_REP message exchange in order to prevent standard TGS spoofing attacks.

Requirements for this new attack are similar to the ones for KDC Spoofing: the attacker must have physical access to the target machine and also be able to falsify KDC messages. A MITM between the target and the KDC is also needed, in practice, since real KDC messages could easily hinder the attack, would they reach the target before the attacker's fake responses. Using a MITM the attacker can easily block unwanted messages.

The attack is composed of two phases: first, the attacker intercepts a valid login communication on the target machine and captures the ticket contained in the valid TGS_REP message emitted by the KDC; then, the spoofing attack is performed and the previously stolen ticket is used to forge a TGS_REP which will decrypt correctly on the target machine. Let us see the attack in detail:

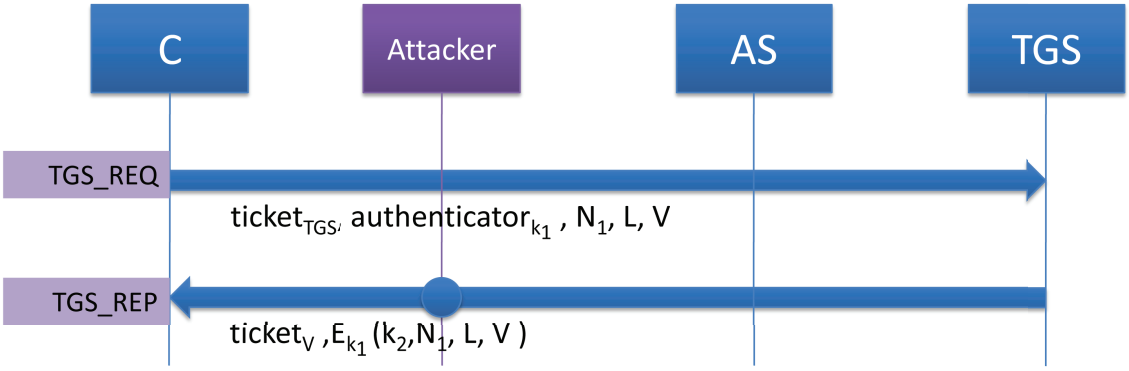
Phase 1: Sniffing (figure 4a)[1]

The legitimate user starts the login procedure on the machine. In the second part of the authentication, the client machine asks for the service ticket and the KDC replies with the expected TGS_REP message. The attacker eavesdrops the message and saves $\mathsf{ticket_V}$. Notice that the login procedure is allowed to be completed.

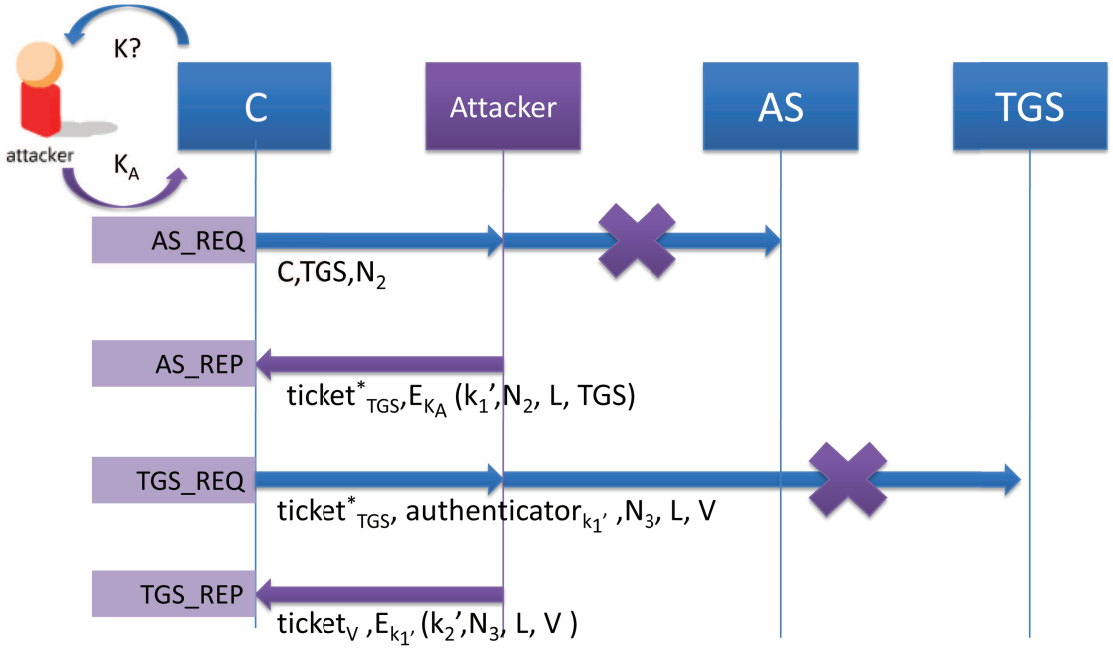
Phase 2: Spoofing and replay (figure 4b)

The attacker physically accesses the target machine, inserts an already known principal name and an arbitrary password, from which the machine derives a key which is considered as the user personal key. The AS_REQ/AS_REP exchange proceed as shown for the KDC spoofing attack. Then, the client uses the received fake TGS ticket $\mathsf{ticket^*_{TGS}}$ to generate a TGS_REQ and sends it to the TGS. This is where the standard KDC spoofing attack fails as the TGS would refuse the fake ticket. At this point, the attacker (in the middle) blocks the request and sends a fake TGS_REP with the previously captured $\mathsf{ticket_V}$. The client machine decrypts the message encrypted under k'_2 and uses the key contained in his keytab to decrypt the received ticket for the fic-

[1] From now on we will drop the N_C^n notation in favor of the simpler (although less precise) N_n



(a) Phase 1



(b) Phase 2

Figure 4: Pass-the-ticket attack

Client workstation	KDC workstation	Vulnerable
Windows XP SP2	Windows Server 2008	✓
Windows Vista Enterprise SP1	Windows Server 2008	✓
Windows 7 Professional	Windows Server 2008	✓
Gentoo MIT Kerb. 1.6.3-r6	Debian4.0 MIT Kerb. 1.6.3	

Table 1: Pass-the-ticket: summary of experiments

titious service V, and since they both decrypt correctly, the attacker gets authenticated.

It can be seen that the attack remains feasible for as long as the stolen ticket remains valid, which is typically 24 hours.

4 *Pass-the-ticket* at work

We have implemented the attack described in section 3.3 and tested it against both MIT and Microsoft implementations of Kerberos. Table 1 summarizes our findings. We have discovered that MIT version is immune to the attack while Microsoft implementation is vulnerable.

Microsoft Windows systems We have tested the following Microsoft Windows operating systems:

- Windows XP Service Pack 2
- Windows Vista Enterprise Service Pack 1
- Windows 7 Professional

The systems were tested both with the out-of-the-box configurations and with all updates installed.[2] As KDC, we have used a Windows Server 2008 machine with default services.

All the given configurations have been found vulnerable to the attack.

MIT Kerberos on Linux systems Tested configuration consisted in a KDC running Debian 4.0 with MIT Kerberos 1.6.3 and a client workstation running Gentoo Linux and MIT Kerberos 1.6.3-r6. The configuration was found not vulnerable to the attack. As described below, the MIT implementation of the login service uses and extended procedure that prevents the attack.

[2]Last update on 01/22/2010.

4.1 Preventing the attack: the MIT solution

Since the MIT implementation is released as open-source software [8], we have inspected it to see how the attack is prevented. This has not been immediate given the size (∼10MB) of the source code.

The important thing to notice is that in Windows systems the attack is successful since, in the second phase, it is only verified the validity of the service ticket. This is not enough to prove that the sender actually knows the key and is, in fact, the TGS. In the MIT Kerberos, instead, all three phases are carried out. The third one, AP_REQ/AP_REP, is executed 'internally', in a simulated way.

Let us review the messages involved in the second phase of the attack to see how this extension prevents it. We write A(U) to note the attacker impersonating U.

$$
\begin{array}{lcll}
\mathrm{C} & \rightarrow & \mathrm{A(AS)} & : \quad \mathsf{C}, \mathsf{TGS}, \mathsf{N}_2 \\
\mathrm{C} & \leftarrow & \mathrm{A(AS)} & : \quad \mathsf{ticket}^*_{\mathsf{TGS}}, \mathsf{E}_{\mathsf{K}_\mathsf{A}}(\mathsf{k}'_1, \mathsf{N}_2, \mathsf{L}, \mathsf{TGS}) \\
\mathrm{C} & \rightarrow & \mathrm{A(TGS)} & : \quad \mathsf{ticket}^*_{\mathsf{TGS}}, \mathsf{authenticator}_{\mathsf{k}'_1}, \mathsf{N}_3, \mathsf{L}, \mathsf{V} \\
\mathrm{C} & \leftarrow & \mathrm{A(TGS)} & : \quad \mathsf{ticket}_\mathsf{V}, \mathsf{E}_{\mathsf{k}'_1}(\mathsf{k}'_2, \mathsf{N}_3, \mathsf{L}, \mathsf{V}) \\
\mathrm{C} & \rightarrow & \mathrm{C} & : \quad \mathsf{ticket}_\mathsf{V}, \mathsf{authenticator}_{\mathsf{k}'_2}
\end{array}
$$

In the fourth message, the client C decrypts the fake TGS response, extracts the session key k'_2 and uses it to create an authenticator, as in a normal Kerberos authentication. Then, C "sends" it to herself (in a simulated way, i.e., no message is actually passing on the network) and, posing as the verifier, decrypts the ticket and uses the key inside it to verify the authenticator.

Recall that the ticket is reused from a previous session and it contains an old session key k_2. The authenticator, instead, is constructed by the client using the key obtained from the TGS, impersonated by the attacker, who does not know what is inside the ticket. Such session key is, in fact, a different key k'_2. In the last phase, the client will then try to decrypt the authenticator with a key different from the one used to encrypt it, thus raising an error condition and terminating the login process.

To conclude, this procedure avoids the pass-the-ticket attack and, at the same time, makes the login service more adherent to the Kerberos specification, since it performs all the protocol steps and verifies all received credentials. The MIT solution, in fact, also offers a clear fix to the Microsoft Windows implementation. Figure 5 summarizes the MIT implementation of the login process and how it prevents the pass-the-ticket attack.

5 Conclusions

Kerberos provides network authentication in a very secure and scalable way, while putting very little burden on the user and being completely cross-

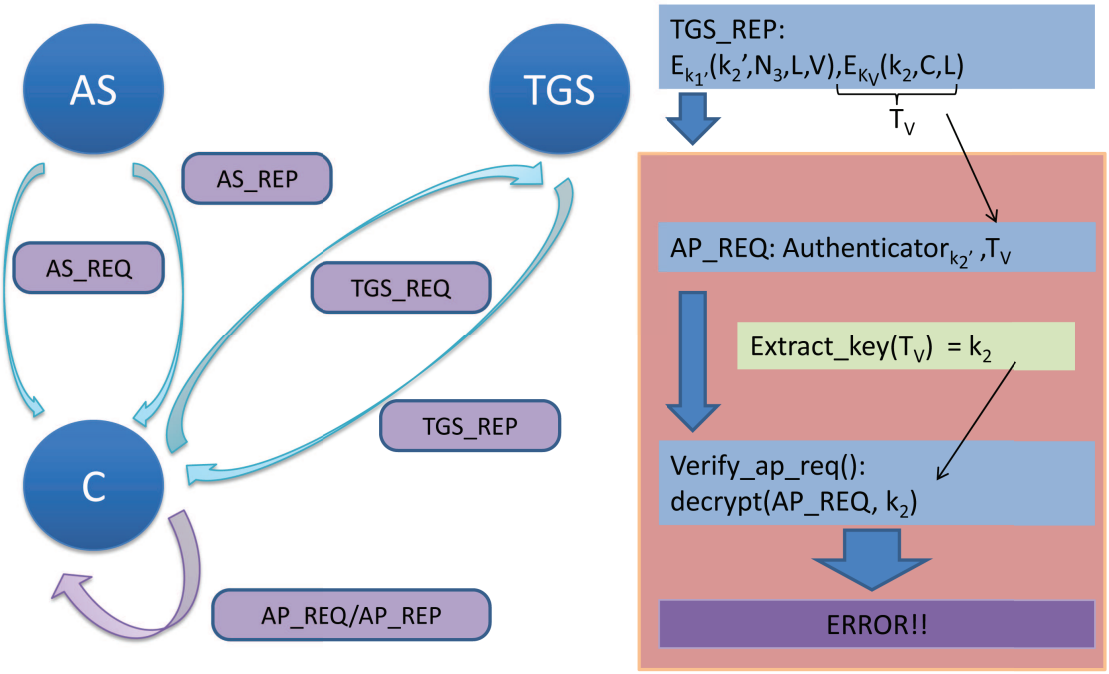


Figure 5: The login process in MIT Kerberos

platform. It has critical importance in Microsoft Windows systems since it is their default network authentication protocol. However, particular care is needed in its implementation and configuration. In fact, many attacks exist that, while not being a threat to the protocol itself, can be successfully exploited by an attacker if the target network makes use of incorrect configurations or implementations deviating from the specifications.

To support this point, we have implemented a recent attack technique targeting physical login on Windows systems, which combines both spoofing and credentials replay techniques to completely falsify authentication. The attack exploits an excessive simplification of the protocol when adapting it to a particular situation that probably Kerberos was not designed to handle: the login process.

We have tested that all recent Microsoft Windows systems, from Windows XP on, are vulnerable to the attack. A very clever solution is provided by the MIT Kerberos code which makes the software much more adherent to the RFC by internally simulating the last message exchange. We have tested that this prevents the attack on a Linux system running MIT Kerberos 1.6.3. Given that the attack is prevented at the protocol level, we expect that any porting of such MIT Kerberos version should be resistant to the attack, independently of the underlying operating system.

References

[1] Bellovin, S. M., and Merritt, M. Limitations of the Kerberos authentication system. In *Usenix Proceedings* (Dallas, TX, 1991).

[2] Bouillon, E. Gaining access through Kerberos. In *PacSec* (2008).

[3] Bouillon, E. Taming the beast: Assess Kerberos-protected networks. In *Black Hat EU* (2009).

[4] CERT. Carnegie Mellon University's Computer Emergency Response Team. `http://www.cert.org/`.

[5] Kasslin, K., and Tikkanen, A. Replay attack on Kerberos V and SMB. `http://users.tkk.fi/autikkan/kerberos/docs/phase1/pdf/LATEST_replay_attack.pdf`, 2003. Vulnerability report.

[6] Malgherini, T. Attacchi sull'autenticazione in reti Kerberos-based. BSc Thesis, Università Ca' Foscari, Venezia. April 2010. (In Italian.).

[7] MIT. ATHENA Project. `http://web.mit.edu/acs/athena.html`.

[8] MIT. Kerberos: The network authentication protocol. `http://web.mit.edu/Kerberos/`.

[9] Neuman, C., and Ts'o, T. Kerberos: An authentication service for computer networks. Tech. Rep. ISI/RS-94-399, USC/ISI, 1994.

[10] Neuman, C., Yu, T., Hartman, S., and Raeburn, K. *RFC 4120: The Kerberos Network Authentication Service (V5)*, 2005.

[11] Raeburn, K. *RFC 3961: Encryption and Checksum Specifications for Kerberos 5*, 2005.

[12] Scapy. `http://www.secdev.org/projects/scapy/`.

[13] Song, D. `http://www.monkey.org/~dugsong/kdcspoof.tar.gz`.

[14] Steiner, J. G., Neuman, C., and Schiller, J. I. Kerberos: An authentication service for open network systems. In *Usenix Proceedings* (Dallas, TX, 1988).

Appendix: kdcreplay.py

We have developed a simple proof-of-concept software to test some of the described techniques, using the scapy [12] library for packets manipulations. The tool can reproduce both the KDC-spoofing and the pass-the-ticket attack, and works in two phases:

1. It sniffs network traffic capturing passing TGS_REP responses, optionally storing it in a pcap file;

2. It performs a KDC-spoofing and replies to subsequent TGS_REQ requests by inserting the ticket for the requested service (typically the only service for which this makes sense will be the login service, with a principal in the form of host/machinename@realm).

It is also possible to skip the sniffing part and replay with a ticket obtained from a previous saved pcap file, or skip sniffing and replay completely and perform a simple KDC-spoofing.

Supported encryption algorithms are Triple Des, RC4 (as implemented by Microsoft, using NTLM hashes as keys) and AES. The tool requires the two included libraries Krb5Packets and Krb5Crypto, to respectively manipulate the desired packets with Scapy and bind the necessary cryptographic functions from the Kerberos API.

```
# python kdcreplay.py -h
"Pass the Ticket" attack
 (a m0t Studios production)
Usage:
kdcreplay.py [opts]
Options:
    -t <target>         set target machine ip
    -k <kdc>            set kdc ip
    -p <pcapfile>       skip service ticket sniffing
  and load TGS_REP from file
    -d <dumpfile>       save sniffed TGS_REP to file
    -r <realm>          set realm name
    -S                  skip tickets replay (kdc spoofing
attack)
    -e <3des|rc4win|aes>        set encryption type
(for rc4win binary key has
to be set, default:3des )
    -D                  lots of debug printing
    -s                  skip spoofing and replay
(for debug purposes)
    -h                  read this
```