Development of Techniques to Remove Kerberos Credentials from Windows Systems

Steffan Roobol
steffan.roobol@os3.nl

Nick Offerman
nick.offerman@os3.nl

July 21, 2019

Assessor:
Prof. dr. ir. Cees de Laat
University of Amsterdam

Supervisors:
Dima van de Wouw, M.Sc.
& Arris Huijgen, M.Sc.
Deloitte
Abstract

Since the release of Windows 2000, all Windows Operating Systems (OS) use the Kerberos protocol to authenticate users to a domain. The Kerberos credentials used for this authentication are stored in the Local Security Authority Subsystem Service (LSASS) process memory for Single Sign-On purposes. Subsequently, they can be extracted from the LSASS memory space using an open-source tool called Mimikatz. Microsoft has released multiple security patches to prevent the system from storing clear-text passwords in memory or prevent access to the LSASS process, but only a reboot of the OS can remove the credentials that are already stored in the LSASS process memory. Unfortunately, some systems, like power generation systems, telecommunication systems, and other Industrial Control Systems, can’t be rebooted due to high availability standards. Therefore, we investigated how Kerberos credentials can be removed in a safe manner from Windows Operating Systems without rebooting the system. To determine this, we investigated how Mimikatz reads out Kerberos credentials from memory. Then, we researched where the Windows native klist command retrieves and removes Kerberos credentials from. Finally, we studied how the process of completely removing Kerberos credentials from a Windows operating system can be automated. We discovered that both Mimikatz and klist look at different Kerberos credentials: Mimikatz extracts credentials from the LSASS memory, and klist enumerates Kerberos credentials from a memory location not belonging to LSASS. In addition to this, we developed a proof of concept tool that removes Kerberos authentication credentials from Windows 7, without rebooting or crashing the operating system.

Keywords: Kerberos, Windows, LSASS Purge, Credential caching, Mimikatz, Klist
1 Introduction

Kerberos is an open-source authentication protocol used by Microsoft in their Windows Operating System (OS) since the release of Windows 2000 [1]. It is used in the Windows OS to allow a device and its users to communicate over a network, by providing the device with credentials in the form of Kerberos tickets. A user authenticates to Kerberos to obtain a Ticket Granting Ticket (TGT) by logging in with his password, which is converted by Kerberos to an encryption key [2]. With this TGT, users can obtain session tickets to access network resources without constantly re-entering their credentials. To provide this Single Sign-On (SSO) feature, all Windows credentials are stored in memory in the Local Security Authority Subsystem Service (LSASS) process [3]. LSASS is a crucial process that handles user authentication on Windows systems. Among others, Windows credentials include both the Kerberos tickets and the encrypted passwords that are used to authenticate to the Kerberos Domain Controller (KDC). The passwords that are stored in memory are encrypted using the native Windows function LsaProtectMemory [4].

As the credentials are stored in memory, they can be read out by a tool called Mimikatz [5]. Mimikatz is a popular open-source post-exploitation tool that allows users to extract all credentials stored in the LSASS memory [6]. Mimikatz also automatically converts the encrypted passwords used for Kerberos back to plain text, using the native Windows function LsaUnprotectMemory. To protect from attacks on the LSASS process, Operating Systems from Windows 8.1 and Windows Server 2012 R2 onwards do not store Kerberos credentials in memory [7]. Windows 10 even introduced Windows Credential Guard, a virtualized container that isolates the LSASS process from the rest of the OS [8]. Still, Mimikatz stays an effective tool for Windows 8 and below [7].

According to the Microsoft documentation, the Windows command klist purge can be used to delete all Kerberos tickets for a specified logon session [9]. However, running the klist purge command for all logon sessions appears to only partially counteract Mimikatz. Before this research, our preliminary experiments showed that the Kerberos credentials of the current logon session could no longer be enumerated. However, credentials from prior sessions could still be read out, even after logging off and logging on again. Therefore, it can be stated that the klist purge command is not able to remove all Kerberos credentials from the LSASS memory.

To truly get rid of data in memory, you can either reboot the system or overwrite the data. Rebooting a device can be a good solution but is not practical for all systems: some systems, like Industrial Control Systems (ICSs), cannot be rebooted as rebooting the system would severely disrupt its operation and the services it provides. In addition to this, overwriting the Kerberos credentials in the LSASS process needs to be done without corrupting, breaking or restarting the LSASS process, as this would prevent further authentication to the machine, force it to reboot or might even crash it [10]. To the best of our knowledge, there is no previous research that attempts to overwrite the Kerberos credentials in the LSASS process memory. Consequently, these credentials remain unaddressed.

Therefore, the main research question of this paper is:

How can Kerberos credentials be completely removed from a Windows Operating System in a safe manner without rebooting the system?

To answer this question, we investigated the following sub-questions:

1. How does Mimikatz read out Kerberos credentials from memory?
2. Where does the klist command retrieve and remove Kerberos credentials from?
3. How can the process of completely removing Kerberos credentials from a Windows Operating Systems be automated?

2 Related Work

Delpy created an open source hacking tool, Mimikatz, to read out credentials from the LSASS process [5]. The tool, written in C, can extract plaintext passwords, NTLM hashes, and Kerberos
tickets. It also has modules for performing pass-the-hash or pass-the-ticket attacks and can create Kerberos tickets. Lastly, the tool contains a module that accurately reads out Kerberos credentials.

Multiple security blog posts from e.g., Medium and Windows OS Hub, address mitigating Mimikatz attacks [11][12]. Suggested protection methods include removing debug privileges from the domain. However, after disabling debug privileges, it is still possible to gain these privileges [13]. Further protection methods include protecting the LSASS memory using a registry key or Windows Credential Guard (Windows 10), or preventing the system from caching credentials using a registry key. Protecting the LSASS memory does not adequately protect against Mimikatz though, as it is possible to dump the process memory with system level privileges and read from that dump. Preventing credential caching does work against Mimikatz, but it only prevents new logon credentials from being cached. Old credentials are still stored in the LSASS memory, so a reboot or overwrite is still necessary.

Loftus and Zismer looked at Kerberos credential extraction on Linux/GNU systems [14]. They tested the MIT Kerberos V5 implementation of the protocol and were successful in stealing Kerberos credentials from a GNU/Linux machine and subsequently reusing them from an attacking device. They retrieved these credentials from the Kerberos credential cache stored in the file system of the machine and did not investigate the Kerberos memory cache of the machine. Although this research is not directly applicable to our research because they work with Linux systems and do not look at the Kerberos memory cache, their test set up provided us with insights on how to perform our experiments.

3 Methods

Our research focused solely on removing Kerberos credentials on client-side operating systems. Figure 1 shows the test environment used for this research. The experiments were performed on a single physical machine, containing an Intel Xeon E3-1200 processor, 16GB RAM, and a 240GB SSD. The OS of the machine was Linux Ubuntu version 18.04.2 LTS, with a KVM hypervisor version 1.5.1 installed on top of it. A virtual machine (VM) with Windows 2008 Datacenter functioned as a server to provide Kerberos Authentication, a Key Distribution Center (KDC), and to serve as the Domain Controller (DC). Several Windows OSs were installed on separate VMs that functioned as client OS for two users, to simulate a domain where multiple users could access the same machine. Table 1 shows the OSs, including their usage worldwide (market share) regarding different Windows versions used for these experiments. The KDC granted Kerberos tickets to the users, so the users could authenticate to the DC, which acts as the Kerberos service.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Edition</th>
<th>NT Version</th>
<th>Buildnumber</th>
<th>Global Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 10</td>
<td>Pro</td>
<td>10.0</td>
<td>10240</td>
<td>58.21%</td>
</tr>
<tr>
<td>Windows 8.1</td>
<td>Professional</td>
<td>6.3</td>
<td>9600</td>
<td>5.75%</td>
</tr>
<tr>
<td>Windows 8</td>
<td>Datacenter</td>
<td>6.2</td>
<td>9200</td>
<td>1.74%</td>
</tr>
<tr>
<td>Windows 7</td>
<td>Ultimate SP1</td>
<td>6.1</td>
<td>7601</td>
<td>31.96%</td>
</tr>
</tbody>
</table>

Table 1: Operating systems used as clients in our test set-up.
The software analysis of Mimikatz was done by hand, inspecting its source code in Visual Studio 2017. The module we investigated was the \texttt{sekurlsa} module, as this is the part of the program specifically used to retrieve passwords from the LSASS process. This is done with the command \texttt{sekurlsa::logonPasswords}, or \texttt{sekurlsa::kerberos} if focusing solely on Kerberos credentials. To remove the Kerberos credentials from memory, a Mimikatz-like tool was constructed, called ‘LSASS Purge’. To reuse as much of Mimikatz’ code as possible, this tool was developed in C. The goal of LSASS Purge was to search for the location and size of the memory that holds Kerberos credentials within the LSASS process, and to overwrite that part of memory with zeroes.

To analyze the behavior of Kerberos credentials in Windows systems, we ran the available credential enumerating commands provided by Windows and Mimikatz, both before and after removing Kerberos credentials using either \texttt{klist purge} or LSASS Purge. These enumerating commands are Windows’ \texttt{klist} and Mimikatz’ \texttt{kerberos::list} and \texttt{sekurlsa::kerberos}. To analyze the \texttt{klist} command further, we disassembled its executable file using IDA version 7.0.190307, and x64dbg version 2019-07-02_16-06. To run \texttt{klist purge} for all logon sessions instead of just the current one, we used a PowerShell script developed by Jared Poeppelman \cite{16}.

4 Results

4.1 Mimikatz analysis

Analyzing the Mimikatz source code yielded the following results. Figure 2 shows the steps taken by the \texttt{sekurlsa::kerberos} command to retrieve the credentials stored in the LSASS process on a Windows 7 OS.

![Diagram showing the steps taken by the \texttt{sekurlsa::kerberos} command](image)

Figure 2: Program flowchart of the steps taken by the \texttt{sekurlsa::kerberos} command on a Windows 7 OS.

When the \texttt{sekurlsa::kerberos} command is executed, a handle is opened to the LSASS process using the native Windows command \texttt{OpenProcess} with read permissions. To do so, Mimikatz needs to have debug privileges, which it acquires using the \texttt{privilege::debug} command. After getting the handle to the LSASS process, Mimikatz reads out the Process Environment Block (PEB) loader data. Then, the program searches the remaining LSASS memory for the piece of memory that
holds the credentials. This piece of memory is subsequently copied to a local buffer. In the LSASS memory space, the memory blob holding the credentials is divided per logon session, including both active and inactive logon sessions. The credentials from each separate logon session are enumerated by Mimikatz using specific memory offsets for Kerberos, converted to strings and printed to the terminal.

A second command we investigated was the kerberos::list command. This command is called to enumerate all Kerberos tickets in the system belonging to the current logon session. The kerberos::list command uses the native Windows LsaCallAuthenticationPackage function, to retrieve all tickets for the current logon ID. This behavior mimics the functionality of klist.

### 4.2 LSASS Kerberos credentials removal

By analyzing Mimikatz, we were able to determine the point where the credentials are read out from LSASS. We copied this code, writing to the piece of memory instead of reading from it [17]. This change is shown in Listings [1.2] and [1.2]

Listing 1: Snippet of original code.

```c
BOOL kull_m_memory_copy(OUT PKULL_M_MEMORY_ADDRESS Destination,
    IN PKULL_M_MEMORY_ADDRESS Source, IN SIZE_T Length)
{
    BOOL status = FALSE;
    KULL_M_MEMORY_ADDRESS aBuffer = {NULL, &KULL_M_MEMORY_GLOBAL_OWNER_HANDLE};
    status = ReadProcessMemory( Source−hMemory−pHandleProcess−hProcess ,
        Source−address, Destination−address, Source−Address, Length,
        NULL);
    return status;
}
```

Listing 2: Snippet of changed code.

Listing [1.2] shows the code that enables Mimikatz to read from the LSASS process. Its inputs are a pointer to a source memory address and the length in bytes that needs to be read, and its output is a destination memory address. The function is called with the memory address within LSASS where the credential blob starts, and the size of this credential blob. It then calls the Windows function ReadProcessMemory to copy the credential blob to a local buffer. Listing [1.2] shows our altered code: we changed the incoming source memory address to an outgoing destination memory address, and instead of calling ReadProcessMemory, we called the Windows function WriteProcessMemory, writing from an empty local memory buffer to the LSASS credential blob.

We tested the tool, LSASS Purge, as described in Section [3]. Before running LSASS Purge, we could use Mimikatz to read out all Kerberos credentials, as seen in Figure [3]. Figure [4] shows the output of the sekurlsa::kerberos command after running LSASS Purge.
Table 2 shows the results of running LSASS Purge on different Windows OSs. After running LSASS Purge, we were still able to use Windows’ klist and Mimikatz’ kerberos::list commands to enumerate Kerberos tickets. In addition to that, we observed that it was no longer possible to enumerate Kerberos credentials using sekurlsa::kerberos on Windows 7. Attempting to run LSASS Purge on Windows 8 yielded no results; thus, we were not able to remove the credentials. Lastly, both before and after running LSASS Purge, sekurlsa::kerberos could not enumerate any Kerberos credentials.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>7</th>
<th>8</th>
<th>8.1</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Before LSASS Purge</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>klist</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>kerberos::list</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sekurlsa::kerberos</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>After LSASS Purge</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>klist</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>kerberos::list</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sekurlsa::kerberos</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 2: Results of retrieving passwords using klist, kerberos::list and sekurlsa::logonPasswords on Windows 7, 8, 8.1 and 10 before and after running LSASS Purge.

As seen by our experiments, Mimikatz is not able to read out any credentials from LSASS after running LSASS Purge on Windows 7. We did observe one situation, which we have verified multiple times, where Mimikatz was still able to read out the credentials; when sekurlsa::kerberos had been run both before and after LSASS Purge, and Mimikatz had not been closed. In addition to this, we were able to access the network share without reauthenticating. Lastly, we observed one situation that forced the Windows 7 OS to reboot: running the PowerShell command Get-WmiObject to enumerate Win32_LogonSession objects.

4.3 Klist analysis

Table 2 shows the results of analyzing the klist purge command on different Windows OSs. Our analysis shows that even before running LSASS Purge it is not possible to list or enumerate Kerberos tickets on Windows 8.1 and higher using either kerberos::list or sekurlsa::kerberos. Furthermore, it shows that running klist purge stops klist from enumerating Kerberos tickets on all tested OSs, and Mimikatz’ kerberos::list from reading out Kerberos tickets on Windows 7 SP1. However, the credentials stored in LSASS can still be extracted, even after running the klist purge command. Lastly, on Windows 8, we were able to list Kerberos tickets using klist, but not using kerberos::list.

The klist.exe executable is located in the C:\Windows\System32 directory. By using IDA and x64dbg, we were able
to disassemble the executable, as shown in Figure 5. The executable uses the Windows function LsaCallAuthenticationPackage in the secur32.dll for all its Kerberos calls. This function uses an unknown function, which is called from inside the advapi32.dll. From here, the disassembly only returned an export table and a function declaration, so further investigation was no longer possible.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>7</th>
<th>8</th>
<th>8.1</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before klist purge</td>
<td>klist</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>kerberos::list</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sekurlsa::kerberos</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>After klist purge</td>
<td>klist</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>kerberos::list</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sekurlsa::kerberos</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 3: Results of retrieving passwords using klist, kerberos::list and sekurlsa::logonPasswords on Windows 7, 8, 8.1 and 10 before and after running klist purge.

4.4 Kerberos credentials removal

Figure 6 shows the results of our Kerberos credentials removal. Both klist and kerberos::list were able to read out credentials after LSASS Purge had been run, and sekurlsa::kerberos was able to read out credentials after klist purge had been run. Therefore, the commands klist, klist purge and kerberos::list do not touch upon the LSASS memory, and sekurlsa::kerberos and LSASS Purge have no influence over the apparent other location storing Kerberos credentials. To remove all Kerberos credentials from Windows systems, we wrote a script using Windows PowerShell that runs both klist purge and LSASS Purge [17]. Table 4 shows the results of that script.

Running either klist, kerberos::list or sekurlsa::kerberos after the PowerShell script, resulted in the credentials being removed from the Windows 7 machine. Again, on Windows 8, we were not able to run LSASS Purge, but the klist purge was carried out fine. Additionally, as with our previous results, both before and after attempting to remove them, sekurlsa::kerberos could not enumerate any Kerberos credentials in Windows 8.1 or Windows 10.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>7</th>
<th>8</th>
<th>8.1</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before removal script</td>
<td>klist</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>kerberos::list</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sekurlsa::kerberos</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>After removal script</td>
<td>klist</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>kerberos::list</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sekurlsa::kerberos</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 4: Results of retrieving passwords using klist, kerberos::list and sekurlsa::logonPasswords on Windows 7, 8, 8.1 and 10 before and after running both klist purge and LSASS Purge.
5 Discussion

This paper set out to determine whether it is possible to completely remove Kerberos credentials from Windows systems. Our results show that, for Windows 7 systems, it is indeed possible to do so, using both native Windows command and a tool based on Mimikatz.

After analyzing Mimikatz’ source code, we developed a tool called LSASS Purge to overwrite all credentials stored in the LSASS process. LSASS Purge does not remove just the Kerberos credentials from memory, and therefore, it could have many unintentional side effects. On Windows 7 systems, LSASS purge is able to remove the credentials without immediately crashing the system. We did notice that calling a `winlogon.exe` function to enumerate logon sessions caused the OS to reboot itself. Additionally, on the Windows 7 OS, we observed that Mimikatz was still able to read out Kerberos credentials from LSASS if Mimikatz had been run and not closed before executing our tool. This was to be expected, as Mimikatz saves the LSASS credentials it reads out to a local buffer. If the executable is not terminated, any subsequent readings are done from that local buffer as well.

A limitation of our research is that we were not able to run LSASS Purge on Windows 8. We expect that LSASS Purge could work on Windows 8, as Mimikatz is able to read out credentials on that OS without any problems. Besides, our tool did not generate any output, such as error messages, which were put in place to detect problems at run time. These observations lead us to believe the problem lies with the configuration of the executable, as it might not have been compiled or ran against the right Windows SDK. Otherwise, we would have gotten feedback from the tool as to what went wrong during its execution. Furthermore, we were not able to test LSASS Purge on Windows 8.1 and Windows 10. As LSASS Purge is based on Mimikatz, and Mimikatz is not able to read out credentials in these OSs, we are not able to test if we truly overwrote the Kerberos credentials.

By analyzing the behavior of Windows’ `klist` command and Mimikatz’ `sekurlsa::kerberos` and `kerberos::list` commands both before and after running either `klist purge` or LSASS Purge, we showed that both Kerberos enumerating commands retrieve their input from different locations. Notably, on Windows 8, we were able to enumerate tickets using `klist`, but not using `kerberos::list`. This could be due to the `LsaCallAuthenticationPackage` call in Mimikatz. From Windows 8 and higher, to perform a Kerberos ticket request using this function, an executable needs to be in the Trusted Computer Base, which Mimikatz is not. During our analysis of the `klist.exe` executable, we were able to trace its calls back to both `secur32.dll` and `advapi32.dll`, but no further. Our results could have been limited by the tooling, but can also be explained by potential obfuscation of the source code of Windows system executables.

Our analysis of the `klist purge` command and its execution stack reached a point where we could not investigate any further. Therefore, our approach to removing Kerberos credentials from a Windows OS is based on the Mimikatz tool alone. As to our knowledge, there have not been any attempts to clear the LSASS memory of its stored Kerberos credentials. In addition, prior efforts by Microsoft to protect the credentials stored in the LSASS memory have only focused on preventing access to the credentials, or on preventing the storage of those credentials in the LSASS process. Such an approach can only be effective as long as an OS is allowed to reboot for clearing the credentials already cached in the LSASS memory, or after an update of the OS. If there is a more robust approach to removing Kerberos credentials already stored in Windows memory, such as Windows function calls, that approach would be preferred to LSASS Purge. However, as of now, LSASS Purge can serve as a proof concept and can be used as a stepping stone for future development.
6 Conclusion

This paper set out to research whether it is possible to completely remove Kerberos credentials from a Windows OS in a safe manner without rebooting the system. It did so by investigating the native Windows executable `klist` and the open source post-exploitation tool Mimikatz. This showed that the Kerberos credentials are stored in two separate memory locations. Furthermore, as a proof of concept, our research produced a tool called LSASS Purge that can be used to overwrite all credentials in the LSASS memory on a Windows 7 OS, including the Kerberos credentials. This tool was able to do so without forcing the OS to reboot or crash, save for one occasion. Using both LSASS Purge and the native Windows command `klist purge` on a Windows 7 OS, we were able to remove the Kerberos credentials from the LSASS memory and let `klist purge` remove the Kerberos credentials from the other memory location.

Based on our results, it can be concluded that on a Windows 7 operating system, using both `klist purge` and LSASS Purge, all Kerberos credentials can be removed from the system without rebooting it.

7 Future Work

As our investigation of the `klist` command stopped at the call to `advapi32.dll`, future work could try to discover where the Kerberos credentials are actually stored. This could solidify and expand the understanding of the `klist` executable, as well as the functions it calls in the process of enumerating or removing Kerberos tickets.

LSASS Purge safely removes all credentials from memory in a Windows 7 system. Removing all credentials could have many unexpected and unwanted side effects, such as forcing a reboot after enumerating `Win32 LogonSession objects`. Therefore, future research could focus on fine-tuning LSASS Purge to more specifically wipe the memory regions containing the Kerberos credential material, to avoid breaking the structures of the data so it won’t affect the OS’ stability. Additionally, in our research, LSASS Purge has only been tested properly on Windows 7. Therefore, future research could focus on reproducing and verifying our results in different Windows Operating Systems. Both earlier versions like Windows XP and Vista as later versions like Windows 8, 8.1 and 10 could be tested to broaden the compatibility of LSASS Purge.