University of Amsterdam

M.Sc. Security and Network Engineering

Research Project 1

# Development of Techniques to Remove Kerberos Credentials from Windows Systems

Steffan Roobol
steffan.roobol@os3.nl

Nick Offerman
nick.offerman@os3.nl

August 18, 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 live or from memory dumps by a tool called Mimikatz on Windows XP, Windows 7, Windows Vista, Windows 8, Windows 8.1 and Windows 10 [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].

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. As users and administrators keep logging onto ICS systems that hardly ever reboot, their credentials keep building up in memory over time. Therefore, when such a system is hacked or otherwise compromised, an attacker could use Mimikatz to obtain a large number of domain credentials. Using these, an attacker could further exploit the domain of the company, especially when the memory of the machine in question contains credentials to privileged user accounts. In addition to not all systems being able to reboot, 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 [9]. According to the Microsoft documentation, the Windows command `klist purge` can be used to delete all Kerberos tickets for a specified logon session [10]. To the best of our knowledge, there is no previous research that attempts to overwrite the Kerberos credentials in the LSASS process memory or verify the effectiveness of `klist purge` with regards to Mimikatz. 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. Prior credentials are still stored in the LSASS memory, so a reboot or overwrite is still necessary to clear credentials of e.g. system administrators on ICS systems, which make exploitation by hackers on the domain of the company less likely.

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.

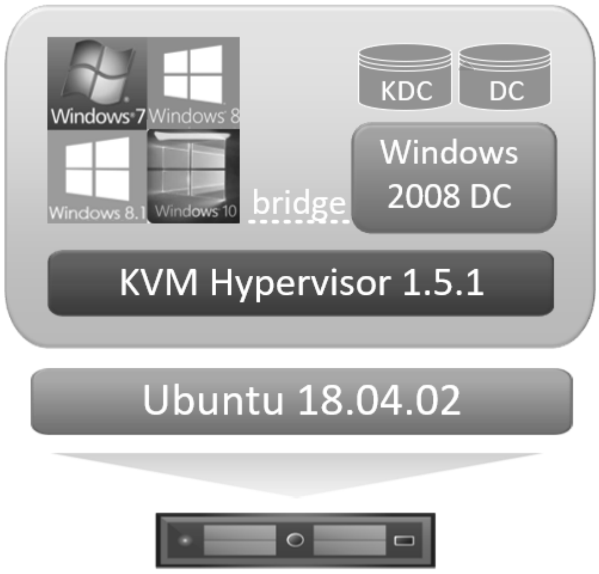


Figure 1: Test environment.

The software analysis of Mimikatz was done by hand, inspecting its source code in Visual Studio 2017. The module we investigated was the `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 `sekurlsa::logonPasswords`, or `sekurlsa::kerberos` if focusing solely on Kerberos credentials. To

Table 1: Operating systems used as clients in our test set-up.

| Operating System | Edition | NT Version | Buildnumber | Global Usage [15] |
|---|---|---|---|---|
| Windows 10 | Pro | 10.0 | 10240 | 58.21% |
| Windows 8.1 | Professional | 6.3 | 9600 | 5.75% |
| Windows 8 | Datacenter | 6.2 | 9200 | 1.74% |
| Windows 7 | Ultimate SP1 | 6.1 | 7601 | 31.96% |

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 `klist purge` or LSASS Purge. These enumerating commands are Windows' `klist` and Mimikatz' `kerberos::list` and `sekurlsa::kerberos`. To analyze the `klist` command further, we disassembled its executable file using IDA version 7.0.190307, and x64dbg version 2019-07-02_16-06. To run `klist purge` for all logon sessions instead of just the current one, we used a PowerShell script developed by Jared Poeppelman [16].

## 4 Results

### 4.1 Mimikatz analysis

Analyzing the Mimikatz source code yielded the following results. Figure 2 shows the steps taken by the `sekurlsa::kerberos` command to retrieve the credentials stored in the LSASS process on a Windows 7 OS. When the `sekurlsa::kerberos` command is executed, a handle is opened to the LSASS process using the native Windows command `OpenProcess` with read permissions. To do so, Mimikatz needs to have debug privileges, which it acquires using the `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. Mimikatz' creator, who reverse engineered the LSASS process, hard-coded the length of the credential block for Windows XP, Windows 7, Windows Vista, Windows 8, Windows 8.1 and Windows 10 in Mimikatz' source code.

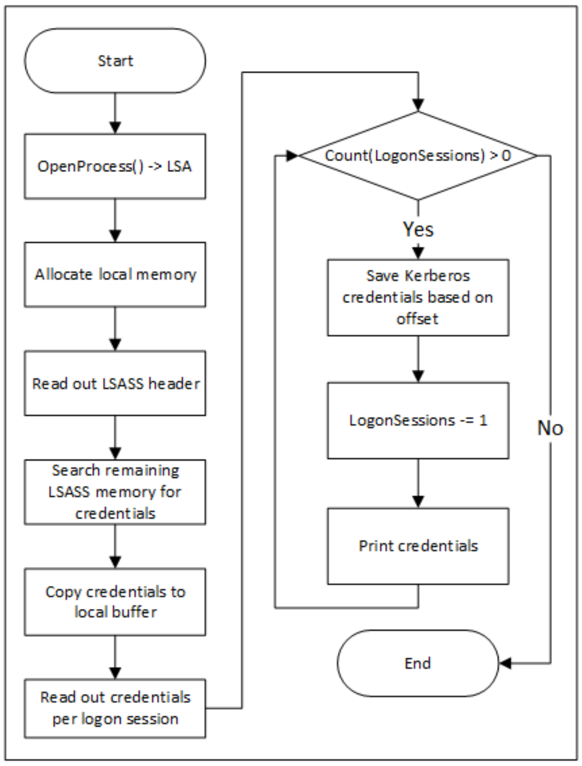

Start
OpenProcess() -> LSA
Allocate local memory
Read out LSASS header
Search remaining LSASS memory for credentials
Copy credentials to local buffer
Read out credentials per logon session
Count(LogonSessions) > 0
Yes
Save Kerberos credentials based on offset
LogonSessions -= 1
Print credentials
No
End

Figure 2: Program flowchart of the steps taken by the `sekurlsa::kerberos` command on a Windows 7 OS.

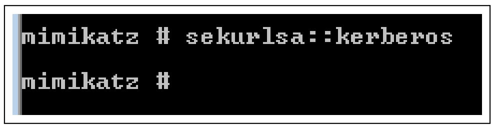
A second command we investigated was the `kerberos::list` command. This command is called to enumerate all Kerberos tickets in the system belonging to the current logon session. The `kerberos::list` command uses the native Windows `LsaCallAuthenticationPackage` function, to retrieve all tickets for the 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 and 2 in the Appendix.

Listing 1 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 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 6 in the Appendix. Figure 3 shows the output of the `sekurlsa::kerberos` command after running LSASS Purge.



```
mimikatz # sekurlsa::kerberos

mimikatz #
```

Figure 3: 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 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. Red results denote a changed observation.

| | **Experiment** | **7** | **8** | **8.1** | **10** |
|---|---|---|---|---|---|
| **Before LSASS Purge** | `klist` | Yes | Yes | Yes | Yes |
| | `kerberos::list` | Yes | No | No | No |
| | `sekurlsa::kerberos` | Yes | Yes | No | No |
| **After LSASS Purge** | `klist` | Yes | Yes | Yes | Yes |
| | `kerberos::list` | Yes | No | No | No |
| | `sekurlsa::kerberos` | **No** | Yes | No | No |

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 by running the PowerShell command `Get-WmiObject` to enumerate `Win32_LogonSession` objects [18, 19]. The reboot happened consistently after we verified this situation multiple times.

## 4.3 Klist analysis

Table 3 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. Additionaly, on Windows 8, we were able to list Kerberos tickets using `klist`, but not using `kerberos::list`. Lastly, observing the LSASS memory space before and after running `klist purge`, showed no changes except for the removal of semaphore variables.

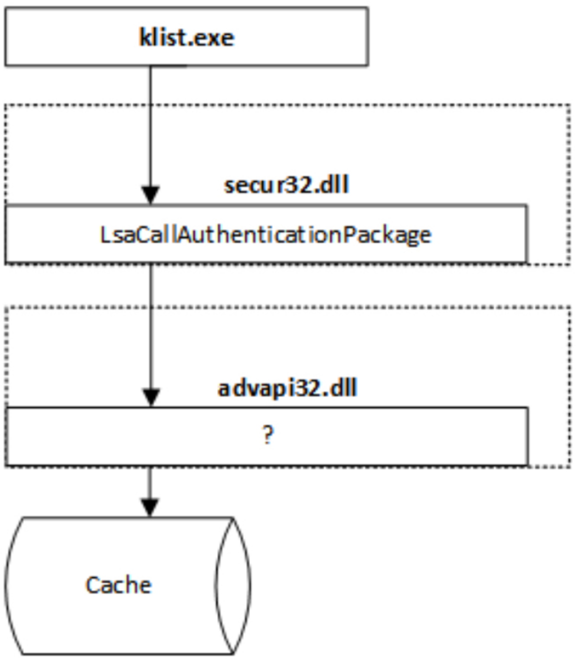


Figure 4: Results of disassembly analysis of `Klist` using IDA.

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 4. 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 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`. Red results denote a changed observation.

| | **Experiment** | **7** | **8** | **8.1** | **10** |
|---|---|---|---|---|---|
| **Before klist purge** | `klist` | Yes | Yes | Yes | Yes |
| | `kerberos::list` | Yes | No | No | No |
| | `sekurlsa::kerberos` | Yes | Yes | No | No |
| **After klist purge** | `klist` | **No** | **No** | **No** | **No** |
| | `kerberos::list` | **No** | No | No | No |
| | `sekurlsa::kerberos` | Yes | Yes | No | No |

## 4.4 Kerberos credentials removal

Figure 5 shows the results of our Kerberos credentials removal, in which a solid arrow indicates that the credentials could be retrieved successfully, and a broken arrow means they could not be retrieved. 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. This means there is probably a second location in Windows Operating Systems where Kerberos credentials are stored, separate from the LSASS process. 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. It is important to note that LSASS Purge could not run correctly on Windows 8, so the results of `sekurlsa::kerberos` after running the removal script were the same as before running it.

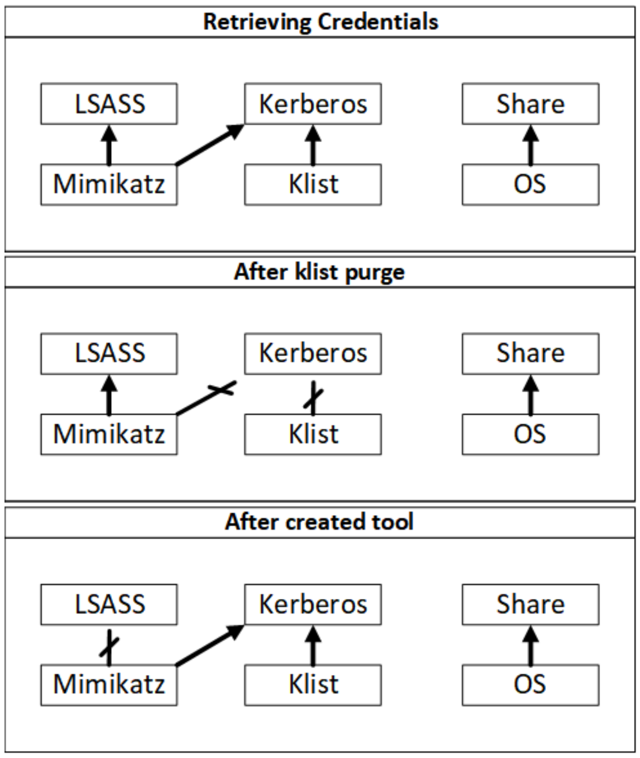


Figure 5: Analysis of retrieving Kerberos credentials with Mimikatz and klist, before and after executing klist purge and LSASS Purge.

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 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. Red results denote a changed observation.

| | **Experiment** | **7** | **8** | **8.1** | **10** |
|---|---|---|---|---|---|
| **Before removal script** | `klist` | Yes | Yes | Yes | Yes |
| | `kerberos::list` | Yes | No | No | No |
| | `sekurlsa::kerberos` | Yes | Yes | No | No |
| **After removal script** | `klist` | **No** | **No** | **No** | **No** |
| | `kerberos::list` | **No** | No | No | No |
| | `sekurlsa::kerberos` | **No** | Yes | No | No |

## 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. In addition to this, our results show that the `klist purge` command does not use or alter the Kerberos credentials stored in the LSASS memory, instead focusing on a different part of the OS.

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 (`Get-WmiObject Win32_LogonSession`) caused the OS to reboot itself consistently, which would lead LSASS Purge to be an unsafe way of removing Kerberos credentials from the memory of a Windows 7 system. 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. Furthermore, we noticed that by running `klist purge`, the LSASS memory space did not change except for the disappearance of some semaphore variables. As semaphores are not actual content but merely set in place to prevent race conditions, this further solidifies our results that `klist purge` does not remove Kerberos credentials from LSASS, but from another location in the OS.

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 [7]. Another limitation of this study could be the use of Windows 2008 Server as server OS, as this is a much older version than the currently available Windows 2019 Server, the latter of which could be more secure and robust. However, as all our experiments were performed on the client Operating Systems, we do not think this affected our outcomes.

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 [20]. 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.

Since Windows 10, Windows Credential Manager stores credentials that are protected by Windows Defender Credential Guard [21]. Stored credentials can't be decrypted, because Virtualization-Based Security (VBS) provides isolation between the secure kernel and the OS. This isolation is further enforced by use of a Trusted Platform Module (TPM) [21]. This implementation of the LSASS process could explain why both Mimikatz and LSASS Purge can not gain the necessary privileges to access the credentials in the LSASS process in Windows 10 systems.

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 [22, 23]. 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 immediately rebooting or crashing the OS. Using PowerShell to call the GetWmiObject function to enumerate `Win32_LogonSession objects` did force the system to reboot consistently. 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 immediately crashing it, forcing it to reboot, or disrupting further authentication.

## 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 removes all credentials from memory in a Windows 7 system in an unsafe manner. 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.

## References

[1] Maples W. *Kerberos and Windows 2000.* URL: `http://techgenix.com/kerberosandwindows2000/` (visited on 03/06/2019).

[2] Microsoft. *Authentication Service Exchange - Windows applications.* URL: `https://docs.microsoft.com/en-us/windows/desktop/secauthn/authentication-service-exchange` (visited on 03/06/2019).

[3] Microsoft. *Cached and stored credentials technical overview — Microsoft Docs.* URL: `https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/hh994565(v%5C%3Dws.11)?WT.mc_id=twitter` (visited on 03/06/2019).

[4] Massachusetts Institute of Technology. *Encryption types - MIT Kerberos Documentation.* URL: https://web.mit.edu/kerberos/krb5-devel/doc/admin/enctypes.html (visited on 03/06/2019).

[5] Delpy B. *Mimikatz - a little tool to play with Windows security.* URL: https://github.com/gentilkiwi/mimikatz (visited on 03/06/2019).

[6] Metcalf S. *Mimikatz - Active Directory Security.* URL: https://adsecurity.org/?page_id=1821 (visited on 03/06/2019).

[7] AD Security. *Mimikatz and Active Directory Kerberos Attacks.* URL: https://adsecurity.org/?p=556 (visited on 09/07/2019).

[8] Microsoft. *Protect derived domain credentials with Windows Defender Credential Guard.* URL: https://docs.microsoft.com/en-us/windows/security/identity-protection/credential-guard/credential-guard (visited on 19/07/2019).

[9] HowToGeek. *Local Security Authentication Server.* URL: https://www.howtogeek.com/79792/local-security-authentication-server/ (visited on 07/06/2019).

[10] Microsoft. *klist — Microsoft Docs.* URL: https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/klist (visited on 03/06/2019).

[11] Medium. *Preventing Mimikatz Attacks.* URL: https://medium.com/blue-team/preventing-mimikatz-attacks-ed283e7ebdd5 (visited on 09/07/2019).

[12] Windows OS Hub. *Defending Windows Domain Against Mimikatz Attacks.* URL: http://woshub.com/defending-windows-domain-against-mimikatz-attacks/ (visited on 11/07/2019).

[13] Windows OS Hub. *How to Obtain SeDebugPrivilege when Debug Program Policy is Enabled.* URL: http://woshub.com/obtain-sedebugprivilege-debug-program-policy-enabled/ (visited on 11/07/2019).

[14] Loftus R and Zismer A. “Kerberos Credential Thievery (GNU/Linux)”. In: *M.Sc. thesis, University of Amsterdam* (2017).

[15] Statcounter. *Desktop Windows Version Market Share Worldwide.* URL: http://gs.statcounter.com/windows-version-market-share/desktop/worldwide/#monthly-201601-201907 (visited on 15/07/2019).

[16] Poeppelman J. *Purge the Kerberos client ticket cache for all sessions (Powershell one-liner).* URL: https://gallery.technet.microsoft.com/scriptcenter/Purge-the-Kerberos-client-b56987bf (visited on 05/06/2019).

[17] Offerman N and Roobol S. *Gitlab repository for LSASS Purge tool.* URL: https://gitlab.os3.nl/sroobol/lsass-purge-tool (visited on 13/07/2019).

[18] Microsoft. *Win32$_L$ogonSessionclass.* URL: https://docs.microsoft.com/en-us/windows/win32/cimwin32prov/win32-logonsession (visited on 18/08/2019).

[19] Microsoft. *IWbemServices::GetObject method.* URL: https://docs.microsoft.com/en-us/windows/win32/api/wbemcli/nf-wbemcli-iwbemservices-getobject (visited on 18/08/2019).

[20] Microsoft. *LsaApCallPackageUntrusted function.* URL: https://docs.microsoft.com/en-us/windows/win32/api/ntsecpkg/nc-ntsecpkg-lsa_ap_call_package (visited on 11/07/2019).

[21] Microsoft. *Windows Defender Credential Guard: Requirements.* URL: https://docs.microsoft.com/en-us/windows/security/identity-protection/credential-guard/credential-guard-requirements (visited on 18/08/2019).

[22] How To Geek. *Why does Windows Wants To Reboot So Often?* URL: https://www.howtogeek.com/182817/htg-explains-why-does-windows-want-to-reboot-so-often/ (visited on 09/07/2019).

[23] Microsoft. *Microsoft Security Advisory 2871997.* URL: https://docs.microsoft.com/en-us/security-updates/SecurityAdvisories/2016/2871997 (visited on 09/07/2019).

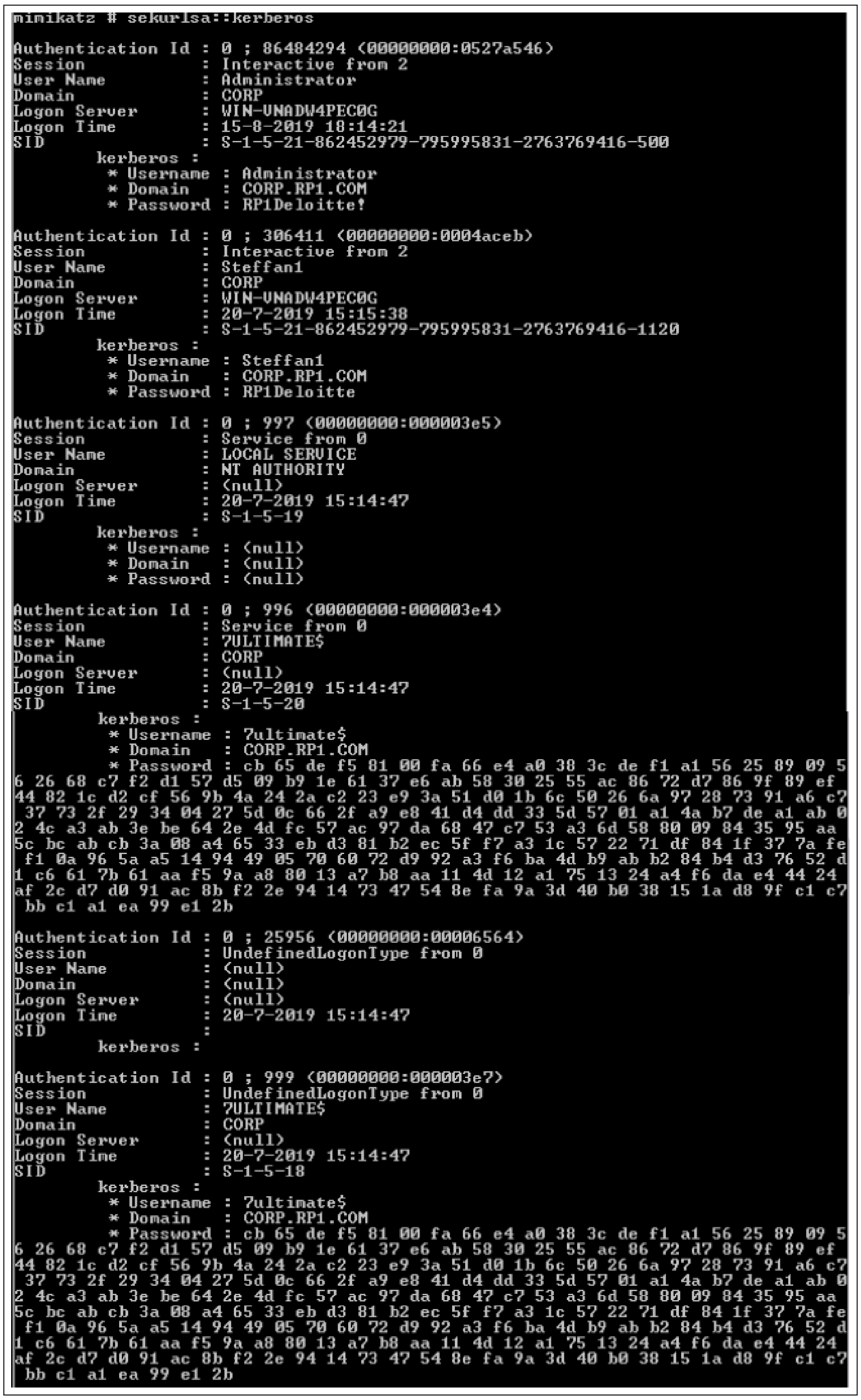
## Appendix

```
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_OWN_HANDLE};

    status = ReadProcessMemory(Source->hMemory->pHandleProcess->hProcess,
        Source->address, Destination->address, Source->Address, Length,
        NULL);

    return status;
}

```

Listing 1: Snippet of Mimikatz' original code.

```
BOOL memory_remove(OUT PLSASS_MEMORY_ADDRESS Destination, IN SIZE_T Length)
{
    BOOL status = FALSE;

    //Initializes a buffer the size of the piece of memory of the credentials
    LSASS_MEMORY_ADDRESS buffer;
    buffer.address = LocalAlloc(LPTR, Length);

    //Writes the buffer to the credential memory
    status = WriteProcessMemory(Destination->hMemory->pHandleProcess->hProcess,
        Destination->address, buffer.address, Length, NULL);

    //Frees the buffer and returns the output of WriteProcessMemory
    LocalFree(buffer.address);
    return status;
}

```

Listing 2: Snippet of LSASS Purge's changed code.



```
mimikatz # sekurlsa::kerberos

Authentication Id : 0 ; 86484294 (00000000:0527a546)
Session           : Interactive from 2
User Name         : Administrator
Domain            : CORP
Logon Server      : WIN-UNADW4PEC0G
Logon Time        : 15-8-2019 18:14:21
SID               : S-1-5-21-862452979-795995831-2763769416-500
        kerberos :
         * Username : Administrator
         * Domain   : CORP.RP1.COM
         * Password : RP1Deloitte!

Authentication Id : 0 ; 306411 (00000000:0004aceb)
Session           : Interactive from 2
User Name         : Steffan1
Domain            : CORP
Logon Server      : WIN-UNADW4PEC0G
Logon Time        : 20-7-2019 15:15:38
SID               : S-1-5-21-862452979-795995831-2763769416-1120
        kerberos :
         * Username : Steffan1
         * Domain   : CORP.RP1.COM
         * Password : RP1Deloitte

Authentication Id : 0 ; 997 (00000000:000003e5)
Session           : Service from 0
User Name         : LOCAL SERVICE
Domain            : NT AUTHORITY
Logon Server      : (null)
Logon Time        : 20-7-2019 15:14:47
SID               : S-1-5-19
        kerberos :
         * Username : (null)
         * Domain   : (null)
         * Password : (null)

Authentication Id : 0 ; 996 (00000000:000003e4)
Session           : Service from 0
User Name         : 7ULTIMATE$
Domain            : CORP
Logon Server      : (null)
Logon Time        : 20-7-2019 15:14:47
SID               : S-1-5-20
        kerberos :
         * Username : 7ultimate$
         * Domain   : CORP.RP1.COM
         * Password : cb 65 de f5 81 00 fa 66 e4 a0 38 3c de f1 a1 56 25 89 09 5
6 26 68 c7 f2 d1 57 d5 09 b9 1e 61 37 e6 ab 58 30 25 55 ac 86 72 d7 86 9f 89 ef
44 82 1c d2 cf 56 9b 4a 24 2a c2 23 e9 3a 51 d0 1b 6c 50 26 6a 97 28 73 91 a6 c7
 37 73 2f 29 34 04 27 5d 0c 66 2f a9 e8 41 d4 dd 33 5d 57 01 a1 4a b7 de a1 ab 0
2 4c a3 ab 3e be 64 2e 4d fc 57 ac 97 da 68 47 c7 53 a3 6d 58 80 09 84 35 95 aa
5c bc ab cb 3a 08 a4 65 33 eb d3 81 b2 ec 5f f7 a3 1c 57 22 71 df 84 1f 37 7a fe
 f1 0a 96 5a a5 14 94 49 05 70 60 72 d9 92 a3 f6 ba 4d b9 ab b2 84 b4 d3 76 52 d
1 c6 61 7b 61 aa f5 9a a8 80 13 a7 b8 aa 11 4d 12 a1 75 13 24 a4 f6 da e4 44 24
af 2c d7 d0 91 ac 8b f2 2e 94 14 73 47 54 8e fa 9a 3d 40 b0 38 15 1a d8 9f c1 c7
 bb c1 a1 ea 99 e1 2b

Authentication Id : 0 ; 25956 (00000000:00006564)
Session           : UndefinedLogonType from 0
User Name         : (null)
Domain            : (null)
Logon Server      : (null)
Logon Time        : 20-7-2019 15:14:47
SID               :
        kerberos :

Authentication Id : 0 ; 999 (00000000:000003e7)
Session           : UndefinedLogonType from 0
User Name         : 7ULTIMATE$
Domain            : CORP
Logon Server      : (null)
Logon Time        : 20-7-2019 15:14:47
SID               : S-1-5-18
        kerberos :
         * Username : 7ultimate$
         * Domain   : CORP.RP1.COM
         * Password : cb 65 de f5 81 00 fa 66 e4 a0 38 3c de f1 a1 56 25 89 09 5
6 26 68 c7 f2 d1 57 d5 09 b9 1e 61 37 e6 ab 58 30 25 55 ac 86 72 d7 86 9f 89 ef
44 82 1c d2 cf 56 9b 4a 24 2a c2 23 e9 3a 51 d0 1b 6c 50 26 6a 97 28 73 91 a6 c7
 37 73 2f 29 34 04 27 5d 0c 66 2f a9 e8 41 d4 dd 33 5d 57 01 a1 4a b7 de a1 ab 0
2 4c a3 ab 3e be 64 2e 4d fc 57 ac 97 da 68 47 c7 53 a3 6d 58 80 09 84 35 95 aa
5c bc ab cb 3a 08 a4 65 33 eb d3 81 b2 ec 5f f7 a3 1c 57 22 71 df 84 1f 37 7a fe
 f1 0a 96 5a a5 14 94 49 05 70 60 72 d9 92 a3 f6 ba 4d b9 ab b2 84 b4 d3 76 52 d
1 c6 61 7b 61 aa f5 9a a8 80 13 a7 b8 aa 11 4d 12 a1 75 13 24 a4 f6 da e4 44 24
af 2c d7 d0 91 ac 8b f2 2e 94 14 73 47 54 8e fa 9a 3d 40 b0 38 15 1a d8 9f c1 c7
 bb c1 a1 ea 99 e1 2b
```

Figure 6: Full output of the `sekurlsa::kerberos` command from Mimikatz.