Defeating Pass-the-Hash

Separation of Powers

Baris Saydag, Microsoft

Seth Moore, Microsoft

Abstract

Pass-the-Hash is but one of a family of credential-theft techniques attackers use in order to impersonate users. Once credentials are obtained, attackers use them to infiltrate and take over entire networks. One source of credentials is the operating system itself, as it provides single sign-on behavior for the user. In this paper, we present a technique for defeating theft of single sign-on credentials.

Credential Theft

Stealing credentials and reusing them is a common technique used to infiltrate computer networks. Pass-the-Hash, often shortened as PtH, is one of many well-understood avenues to steal credentials. With PtH, password hashes are stolen from OS memory and reused. Other, similar techniques are Pass-the-Pass and Pass-the-Ticket, in which case passwords and Kerberos tickets, respectively, are replayed.

These attacks are all made possible by a feature of the operating system called single sign-on, or SSO. With SSO, the computer system itself caches data in order to act on behalf of the signed-on user without challenging the user again for authentication. Since the data used for SSO is in RAM, an attacker who may read that RAM is able to read, and replay, the data.

Once an attacker has credentials for one user, the next stage begins which is lateral traversal. Lateral traversal means movement across a network between systems as the same user identity. The movement is considered lateral, as the attacker has not gained any new, elevated privileges yet. However, the attacker can read the memory on any system which their stolen user identity has administrative privileges. Attackers will then harvest more credentials from these systems, and repeat the process until they gain higher privileges. Given time and persistence, an attacker can often move deeply into a network and gain highly-privileged accounts.

One possible solution to stopping this problem is to simply disable SSO. This solution is viewed as not feasible as users can still be tricked into authenticating, legacy protocols still have credential-equivalent artifacts, and it's a poor user experience.

Another solution, which is implemented in Windows 10, is a feature called Credential Guard. It blocks administrator-level attackers from reading the memory which contains credentials. Credential Guard itself is built atop a new architecture which defines a mechanism by which secrets can truly be kept hidden from all users, even those with system privileges.

Keeping Secrets

Introduction

Once upon a time, an emperor told his army to get ready for a big military campaign, but he didn't share the target country even with his generals because he feared information leakage. One of his generals insisted on knowing the destination, and told him that he can trust him. Emperor asked him if he knows how to keep a secret. When the general got excited and said "yes", emperor said, "So do I".

Keeping secrets is easy, because nothing needs to be done, you just keep quiet. Of course, just keeping secrets is not a goal. In the above story, of course, emperor would have preferred to be able to share the name of the target country so that his army can prepare better. We want to be able to share the secrets constructively without a fear of information leakage.

Need for separation of powers

Ensuring an environment of trust, which is also free of abuse starts with distribution of powers. Modern governments achieve this by separating powers. Simply,

- Legislative branch passes the laws
- Executive branch carries out the tasks, enforces laws
- Judicial branch, arguably the biggest differentiator of modern times, makes sure that everyone obeys the constitution, even the legislative branch.

It is reasonable to think of a modern OS as a modern government.

- Administrators → The Legislative Power, decides who can run what, and to what degree.
- Kernel / System Services / Drivers → Executive Power, carries out the requests of user mode apps, which was decided by admins.
- Trusted Computing Base (TCB) → Judicial Power, makes sure everyone obeys the constitution, even the administrators.

A secure OS design necessitates a separation between these.

Admins are human and humans err. Data shows: > 90% of Windows users run as some sort of administrator. Admin being the ultimate power would mean a total loss of system when a malicious attachment is run by the administrator. Furthermore, we can't simply assume that administrators are always trusted; what if the administrator is malicious? Administrators should not have total control on the machine. Good examples are games: we want to prevent piracy and cheating. Another example is multi-tenant scenarios. An administrator is not necessarily the only administrator on an OS image. One OS image is not necessarily the only compute domain on a physical machine with VMs.

We can't simply trust the kernel, either. Because of thousands of system calls and IOCTLs, kernel has a very big attack surface. Furthermore, the diverse ecosystem of many 3rd party drivers with different quality assurance standards makes the security inconsistent.

Past solutions

Microsoft has attempted solving this problem in the past. Authenticode / Kernel Mode Code Signing (KMCS) is designed to put reputation of an authenticated identity on the line. The goal is negatively impacting exploit economics by increasing cost and traceability. The big problem is that strong verification of publishers by CAs is questionable at best and recalls are hard and slow.

Another mechanism is "Protected Process" (PP) / "Protected Process Light" (PPL) that were introduced in Windows Vista. These technologies isolate sensitive user mode processes from others (including administrators) by preventing injection of threads, memory access and debugging. Since these processes are not protected from kernel, and since kernel cannot be trusted, this mitigation is not enough.

Finally, patch-guard is designed to protect kernel by limiting what kernel mode code can do. However, parts of it are heuristic-based and therefore not failsafe.

All these technologies are good for what they are, but they all have an inherent limitation: They are all software based. Ideally, we want to root security on HW.

HW protections

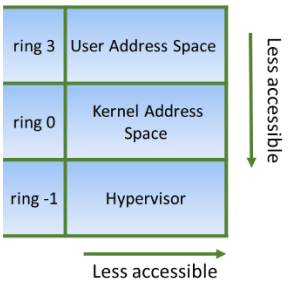
Historically, Intel architecture systems used "rings" to protect various levels of OS. Ring 0 commanded the entire physical address space, therefore it commanded the entire machine. This confused the protection between kernel (because kernel runs in Ring 0) and TCB (because ring 0, therefore kernel, can access all memory).

With the introduction of Hypervisor, a new abstraction layer was introduced. Hypervisor didn't rely on rings, it had its own privileged CPU instructions. But since it brings restrictions to what ring 0 can do, and therefore it is below ring 0, it is nicknamed "ring -1". Hypervisor is consequently protected from kernel and ring 0. Also, hypervisor is small compared to rest of the system, therefore easier to test and verify. And as mentioned above, it roots its security on HW. Therefore, we can consider hypervisor as true TCB.

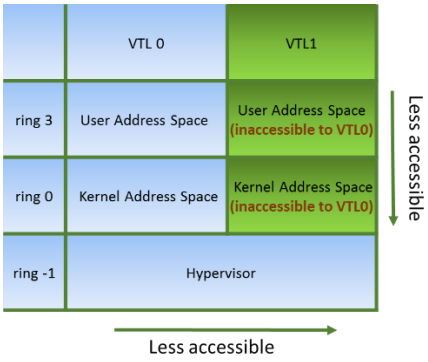
Hypervisor would be a good place to keep and process secrets, but that would make it bigger and more complex, which is detrimental to its security. It is clear that there is a need for hypervisor kind of isolation without cluttering hypervisor.

Virtual Trust Levels - VTLs

Windows 10 introduces a new concept called Virtual Trust Levels. Historically, access layers grew vertically. VTLs allow growing horizontally. Here is the legacy architecture:



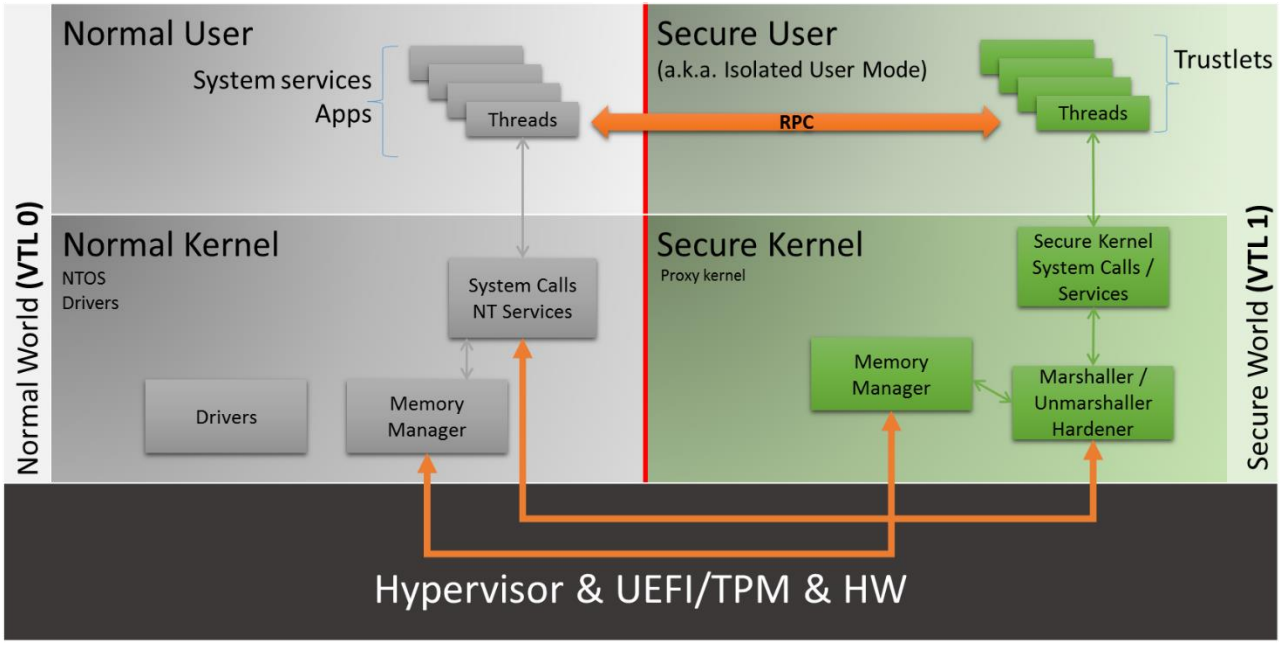
Here is the architecture with VTLs:



Above, regular Windows, now called "Normal World" runs in VTL0. This is mostly business as usual. A new, "secure world" runs in VTL1, which is only selectively accessible to VTL 0. It is important to note that access restrictions between VTL1 and VTL0 is enforced even if VTL0 happens to be running in ring 0.

Such horizontal expansion was achieved utilizing Second Level Address Translation (SLAT) technologies that recently became available in modern CPUs. This virtualization based technology allows a very efficient mapping between guest virtual, guest physical and system physical pages. Therefore, sections of memory can be efficiently access protected and isolated from each other. Access configuration is done by Hypervisor (and as delegated to VTL1), hence making VTL 1 less accessible than VTL 0.

Here is normal world and secure world side by side:



Here are a few important aspects to note:

- User mode applications are called "trustlets" in the secure user mode.
- Since VTL1 is mostly (not entirely, because VTL1 deliberately shares some data with VTL0 as needed) inaccessible to VTL0, all communication happens only through supported secure channels. User mode trustlets talk to their counterparts using RPC. Normal and Secure Kernel modes talk to their counterparts through a specific system call channel.
- Since VTL1 doesn't trust VTL 0, both channels (RPC and system call) are hardened against a possibly rogue VTL0.

Properties of Secure World

Secure world is invisible. There is no user interaction or UI. Since this technology is planned to be deployed to both servers and workstations, performance was a key concern. Several optimizations both in SW and HW made a low performance impact possible. Virtualization based security has minimal impact on perf, typically less than 5%.

Secure world has much tighter control to ensure proper testing and verification. No 3rd party code will be allowed in the secure kernel. Furthermore, trustlets are isolated from each other, ensuring that a compromised trustlet doesn't automatically compromise the rest of the system. Finally, trustlets are limited in number and they are all purpose built – therefore they are much smaller and easier to protect.

World is small... Secure world is smaller. If secure mode is not available, a trustlet can run as a normal mode process, ensuring code reuse, therefore code reduction. Secure world relies on enlightened normal world / NTOS for many things such as scheduling, most of memory management and synchronization. Secure kernel only does the bare minimum, such as configuring SLAT as applicable and encrypting pages before being paged out (normal kernel still has all the logic for paging out, which allows for code reduction, encryption ensures secrecy).

Prior Windows State of the Art

The state of Windows credentials in memory, up until Windows 10, is best described by the following Figure 1, which demonstrates the basic architecture. In this diagram, Alice's laptop is shown with the Windows security

subsystem, LSASS, hosting authentications protocols. In this case, the most interesting protocols are highlighted: NTLM and Kerberos. These are the primary protocols used for Windows Active Directory authentication.

As illustrated in Figure 1, an admin-level attacker may read all the memory. The attacker may also read Alice's password, as it is entered into memory on logon. With any of these items, the attacker may impersonate the user on the network, gaining access to resources. With Credential Guard, these credentials may be better protected, preventing an admin-level attacker from stealing them. Credentials, and equivalent artifacts, are also shown in Figure 1. The primary credentials targeted are typically the user's password, the NTOWF, or the TGT key.

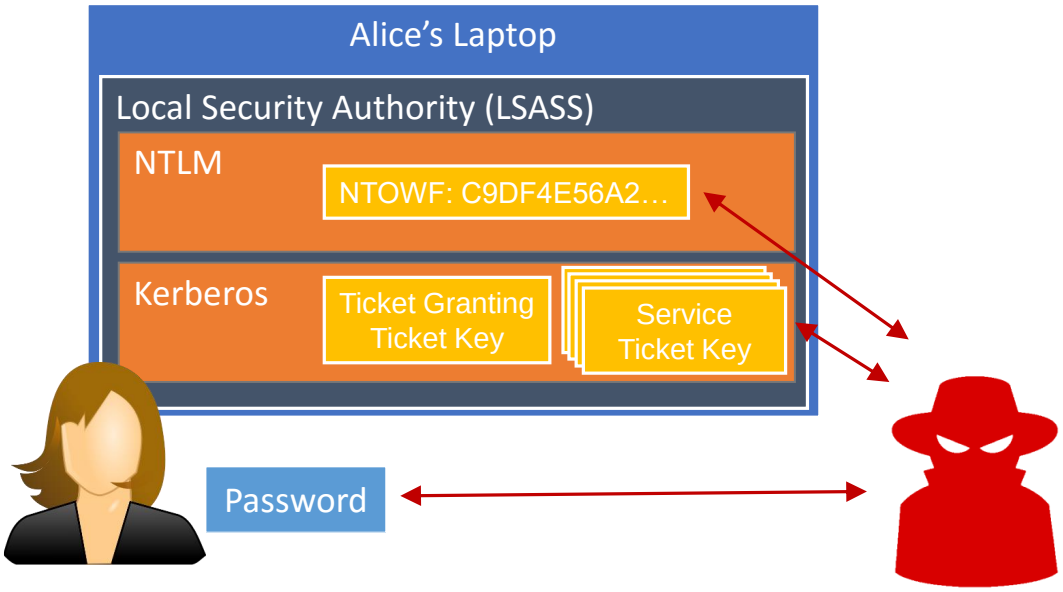


FIGURE 1

Password

The password is the primary credential entered by the user to authenticate themselves. An attacker who harvests this may impersonate the user in many contexts and via virtually any protocol. This makes the password one of the highest-value credentials that may be stolen.

NTOWF

An abbreviation of "Windows NT One Way Function", the NTOWF is the MD4 hash of the user's password. The NTOWF is used to authenticate users via the Windows NT LAN Manager protocol, often abbreviated as NTLM. Though an old protocol, NTLM is used heavily in legacy applications and hardware.

TGT Key

The Ticket-Granting-Ticket, abbreviated TGT, is an artifact of Kerberos authentication. When a user authenticates to a Windows domain, the Kerberos protocol is used. After authentication, the domain provides the user's system a TGT along with a corresponding key. An attacker who is able to harvest both the TGT and TGT key is able to impersonate the user in the context of Kerberos authentication.

Strengthening Primary Credentials

The first step required to prevent credential theft is to move to stronger credentials. A weak credentials is defined as one which is easily stolen and reused. A stronger credential is one that is resistant to theft, and may not simply be taken and reused on its own.

Examples of credential strengthening are the use of key-based authenticators or multi-factor authentication. One such strong credential is a smart card which is unlocked with a PIN. An attacker may steal a user's PIN, but it is not possible to steal the key contained within the smart card. This means there is nothing for the attacker to pass to another system in order to laterally traverse the network.

Multi-factor authentication is a different sort of credential strengthening. Instead of fundamentally changing the primary credential it is augmented with a second factor, such as a code generator. This mitigates credential theft, as the attacker must steal more than one piece of information in order to impersonate the user.

In order to provide any guarantees of credential secrecy, users must use strong credentials. Weak ones, such as passwords, are so easily stolen that an attacker may bypass any mitigations and impersonate the user with their primary credential.

Windows Legacy Compatibility

Even with strong credentials, there are credential-equivalent artifacts on the system which may be stolen. These are first shown in Figure 1: the NTOWF and TGT Key. Figure 2 demonstrates how these artifacts are obtained by the system.

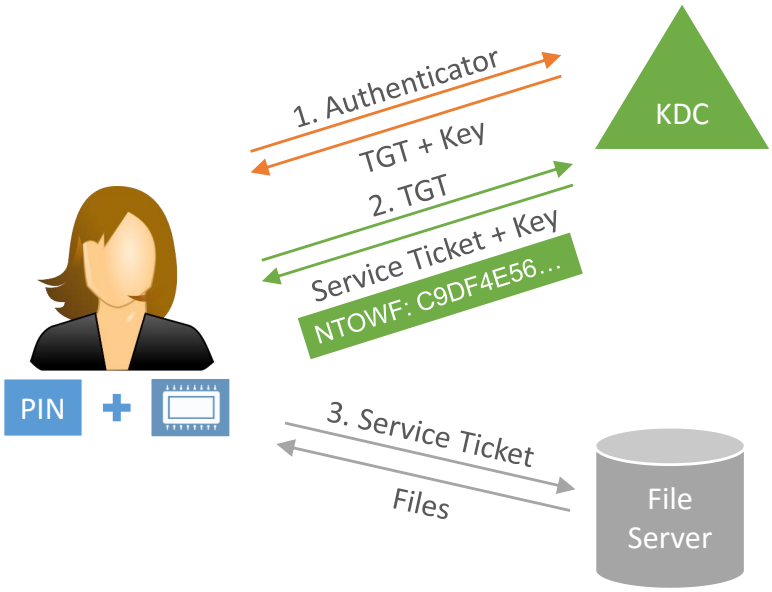


FIGURE 2

In the first phase, Alice presents her proof of smart card position via an authenticator. This is accepted by the Key Distribution Center, or KDC, who is the authenticating party. The KDC returns the TGT plus key. Later, Alice uses her TGT to get a ticket for accessing a file share. At this point, Alice has a ticket she may use to access the file server.

In addition to the service ticket for accessing the file server, the KDC also returns an NTOWF for legacy NTLM support. This is required to access resources when the user doesn't know their password.

In this authentication flow, Alice's initial authentication is safe, as an attacker may not steal the smart card key. However, the attacker may steal the TGT Key or NTOWF out of system memory. This is where the new feature, Credential Guard, comes into play. It ensure in-memory credentials and equivalents are unreadable by any user.

Credential Guard

Design

The solution to credential theft, in Windows 10, is a new authentication component called Credential Guard. With Credential Guard, a new, isolated process is the only process on the system allowed to use the clear credentials. These credentials are never revealed to memory an attacker may read. Credential Guard is predominantly stateless, and thus most secrets are encrypted so that they may be managed by the Normal User mode LSASS process.

The bulk of protocol logic remains within LSASS, while Credential Guard provides only the required processing for the protocols to use their secrets. Examples of such operations are generation of response messages or proof-of-possession of keys. In these cases, LSASS gives the encrypted secret to Credential Guard and receives the output of the operation. This division of effort is depicted in Figure 3.

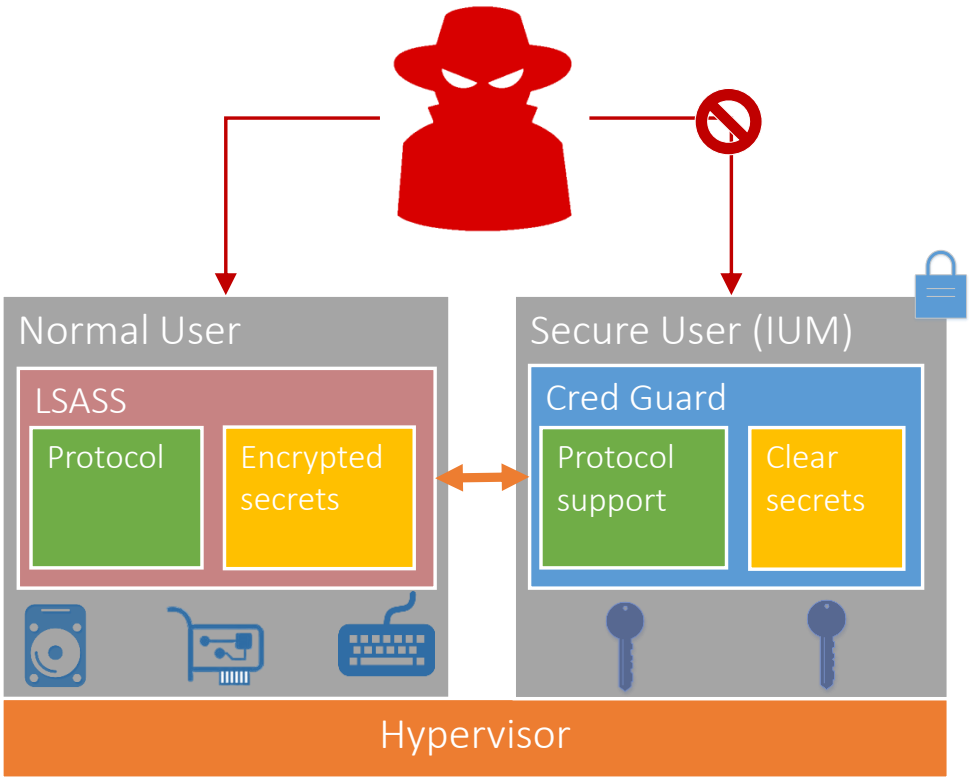


FIGURE 3

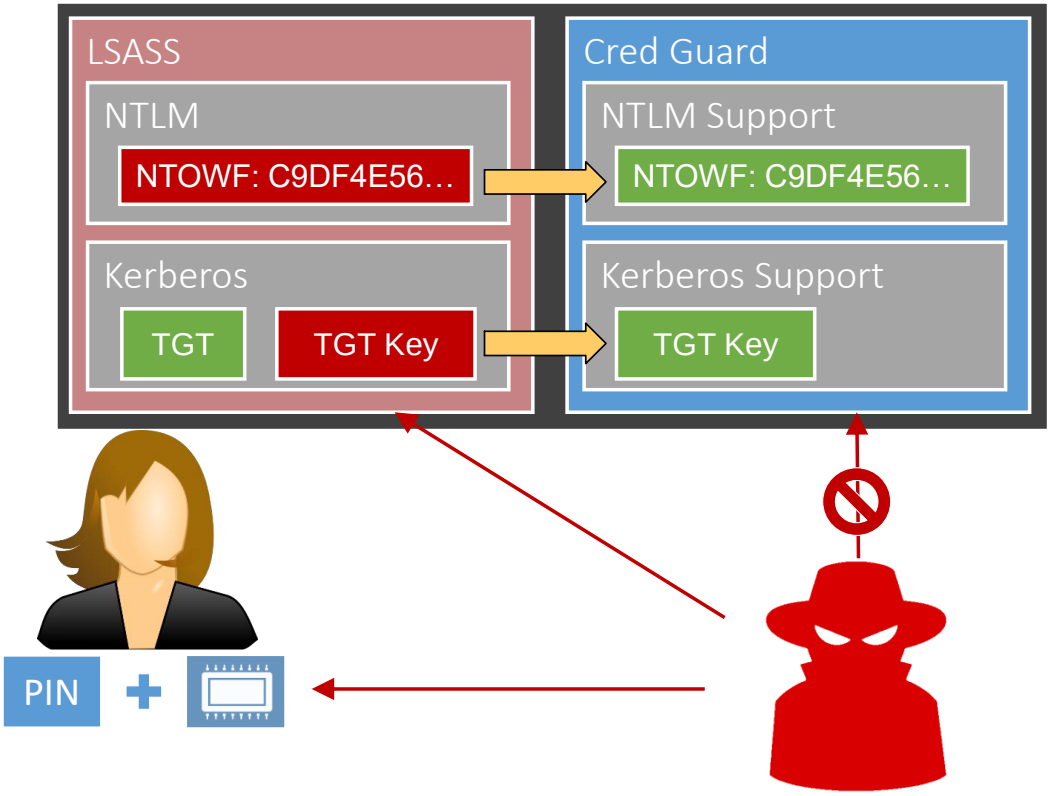


FIGURE 4

The specific credentials which are protected is shown in Figure 4. Here, it is illustrated that Credential Guard hosts the NTOWF and TGT. They no longer exist in LSASS memory, even in a transient state. These credentials are thus protected from attacks which read LSASS memory.

Binding Users to Systems

With Credential Guard and strong primary credentials, Alice's identity is strongly protected. Attackers have no access to data that can be stolen and used for lateral traversal. However, there's nothing preventing a determined attacker from turning off Credential Guard. There's also nothing preventing an attacker from keylogging Alice's PIN and using the smart card. In order to protect against these attacks, a Windows Server 2012 R2 feature, called Authentication Policies, is required.

Authentication Policies allow designating which systems a user may authenticate from. The KDC enforces this binding by requiring a cryptographic proof in the form of an "armor key". This armor is defined as part of RFC 6113: A Generalized Framework for Kerberos Pre-Authentication.

When a user authenticates from a system using armoring, the user's credential is combined with the machine credential, forming a sort of proof-of-origin key. Only with this proof-of-origin can the reply from the KDC be decrypted. This results in a chain of identification, starting with the machine's credential and ending with the user TGT Key, illustrated in Figure 5.

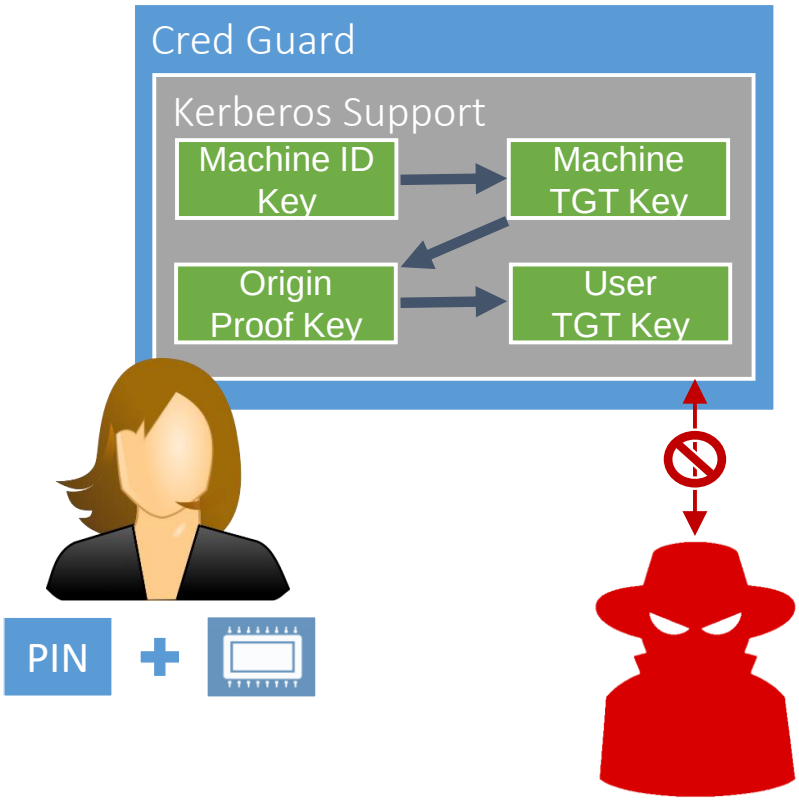


FIGURE 5

So long as the machine's identity key is strongly protected by Credential Guard, the attacker may not disrupt this chain of identification and gain any passable data from the user's session. The attacker may not steal the primary smart card key, the machine identity key, or any derived keys from it. They all reside inside of Credential Guard.

The Path to Secured Users

Windows 10 provides strong mitigations against the theft of credentials and credential equivalents. Users with compliant hardware, Credential Guard, strong authenticators, and a well-defined authentication policy can be certain that even the most highly privileged malware is unable to steal credentials. These protections form a strong, first-line defense against attackers looking for a credentials to being infiltrating other systems.