Bypassing Local Windows Authentication to Defeat Full Disk Encryption

Ian Haken (ian.haken@synopsys.com)

November 12, 2015

Full disk encryption is a defensive measure in which all data stored on a physical disk or volume is encrypted, therefore protecting any data stored on a device such as saved passwords, emails, session tokens, and intellectual property. Full disk encryption protects data at rest, assuring confidentiality even when an attacker has physical access such as when a device is lost or stolen. BitLocker is Microsoft's full disk encryption solution included with certain versions of Windows, first introduced in 2007.

This paper describes an attack which is able to bypass Windows authentication, even in the presence of BitLocker full disk encryption, and thus allows an attacker to access a user's data or install software. On systems effected this attack therefore bypasses all of the protections offered by BitLocker.

1 Introduction

In 2007, starting with Windows Vista, Microsoft began including a full disk encryption feature named BitLocker with professional and enterprise versions of Windows. Full disk encryption helps protect users from threats with physical access to the device or disk. This can, for example, prevent the exposure of proprietary information and saved account credentials if a laptop is lost, stolen, or even left temporarily accessible to an attacker (such when left unattended).

Under the hood, BitLocker's preferred method of operation utilizes a system's Trusted Platform Module (TPM) to store the secret key used for full disk encryption, and is able to use the features of the TPM to safely provide transparent, passwordless decryption of the disk on boot. Because BitLocker can work in a way that's completely transparent—without any extra passwords or prompts on boot—many corporate settings have opted to enable this form of full disk encryption as a part of their data loss prevention strategy.

However, this paper presents an attack which takes advantage of physical access to bypass local Windows authentication. When BitLocker is enabled without any form of pre-boot authentication by using the TPM (which is Microsoft's recommended deployment strategy for BitLocker [1]), this would allow an attacker to access a user's data even though the disk is fully encrypted. Unlike other attacks that have been considered against full disk encryption generally or BitLocker specifically, this attack is completely reliable on systems effected, is a software-only attack, is fast (there is no brute-forcing of keys and only takes seconds to execute), and does not require a sophisticated attacker (only requiring standard open source tools and a few commands).

This paper describes details of BitLocker's operation and Windows authentication in order to illuminate both the countermeasures in place and how the attack described is able to operate despite them. Notwithstanding the level of detail discussed in the first few sections, the attack in section 5 is fairly straightforward and non-technical, so eager readers are encouraged to skip straight to that section, referring back to the earlier sections if greater insight is desired.

In section 2 the motivation and goals of full disk encryption schemes are described. The next section then discusses the relevant functionality of the Trusted Platform Module, specifically how it is utilized by Microsoft BitLocker to provide transparent full disk encryption. Section 4 describes aspects of Windows domain authentication and relevant parts of the Kerberos protocol. The final two sections then describe the attack itself and the impact and mitigation of this attack.

2 Full Disk Encryption

Full Disk Encryption (FDE) is a technique of securing data at rest by encrypting all data before it is written to a disk (or, depending on the implementation, a particular volume/partition). FDE avoids the problem of needing to selectively specify what data should be considered sensitive by protecting all data on the system. This avoids accidental leakage of data, such as leaving an unencrypted working copy in a temporary directory, a swap partition, or a page file.

FDE is intended to mitigate the impact of a malicious actor who has physical access to the device containing the data. FDE will generally offer no defense against a remote attack since a system that is fully operational and allowing remote access should already be transparently decrypting any data from the disk. Therefore, FDE should be viewed as a mitigation against attacks which include physical access, such as when a device is lost, stolen, or left unattended. In is therefore in the context of physical access that we entertain the attack in section 5.

Depending on the implementation FDE can, besides offering confidentiality of data at rest, also offer an assurance of system integrity when the operating system and applications are encrypted. Although an attacker may write arbitrary ciphertext to an encrypted disk, they would generally be unable to determine where on the disk target files are located, much less be able to write ciphertext that has meaningful plaintext once decrypted by the victim. This means an attacker would face significant difficulty attempting to plant malicious software on the device, despite having physical access. BitLocker also specifically protects the preboot process from modification using the Trusted Platform Module, as detailed in the next section.

Disk encryption is not a new technology. FileVault, Apple's FDE implementation for OS X, has been available since 2003. The Linux module dm-crypt, which is usually used for disk encryption in Linux distributions, has been available since 2005. FreeBSD has two disk encryption modules, GBDE and GELI, originally released in 2002 and 2005 respectively. BitLocker, Microsoft's FDE implementation, was first included with Windows Vista in 2007. Disk encryption is also included in many smartphone operating systems; for example, Android introduced full encryption of the data partition starting with its 3.0 release in 2011.

As with any form of encryption, FDE uses a secret key to encrypt the data. Since the encrypted volume usually includes the operating system, most FDE implementations prompt the user for a password early in the boot process (before the full operating system can be read from the disk) in order to derive this secret key. We will refer to this step as pre-boot authentication. BitLocker, however, has been developed to take advantage of the Trusted Platform Module (TPM) available on many platforms. A TPM is a hardware chip capable of performing a number of cryptographic operations and storing secrets. The next section details how BitLocker uses the TPM in order to safely store its secret key for FDE, thereby enabling BitLocker to transparently decrypt the operating system volume on boot without requiring pre-boot authentication.

Even when a TPM is available, BitLocker has the option to enable pre-boot authentication, requiring the user to supply a PIN or insert a USB key containing a saved secret. However, this comes with disadvantages as described by Microsoft [4]:

> "Pre-boot authentication provides excellent startup security, but it inconveniences users and increases IT management costs. Every time the PC is unattended, the device must be set to hibernate (in other words, shut down and powered off); when the computer restarts, users must authenticate before the encrypted volumes are unlocked. This requirement increases restart times and prevents users from accessing remote PCs until they can physically access the computer to authenticate, making pre-boot authentication unacceptable in the modern IT world, where users expect their devices to turn on instantly and IT requires PCs to be constantly connected to the network.
>
> If users lose their USB key or forget their PIN, they cant access their PC without a recovery key. With a properly configured infrastructure, the organizations support will be able to provide the recovery key, but doing so increases support costs, and

users might lose hours of productive work time."

Microsoft therefore explicitly recommends not using pre-boot authentication when possible [1]. The attack described will take advantage of this configuration.

3 Trusted Platform Modules

The Trusted Platform Module (TPM) is a standard for a hardware device capable of various cryptographic operations, such as secure key generation and random number generation. It also has capabilities for performing remote attestation (that is, providing assurance to a remote third-party that particular software is being run on a device) and securely storing secrets. It is this last feature that is used by BitLocker to store a disk's decryption key, which this section will detail.

This section goes into detail on how the TPM is used to store secrets for BitLocker, but many specifics are intentionally omitted or simplified for the sake of brevity and scope. If the reader is interested in the complete technical details, the TPM Library Specification [6] has more information.

A TPM is capable of *sealing* a secret, which encrypts the secret in such a way that it is bound to the specific TPM sealing the secret, the current software running on the device, and the configuration of that software. The last two features are measured using the TPM's Platform Configuration Registers (PCRs) which are used to record the software and configuration of the device. How the PCRs is used is up to the application in question (for example, BitLocker by default only uses 4 of the 24 PCRs).

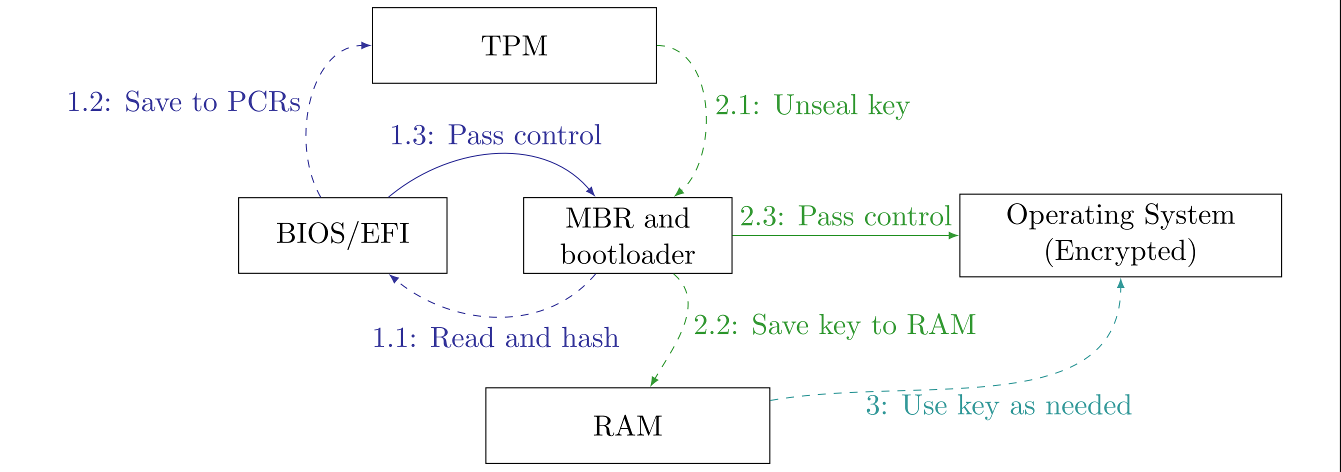
A critical aspect of the PCRs' operation is that their values cannot be set directly; instead one can only append values to PCRs. When the machine is first booted, all of the PCRs are initialized to a zero value. When the platform instructs the TPM to update a PCR with a new value, that value is appended with the PCR's previous value, hashed, and the result is saved to the PCR. This means that short of calculating a hash collision, an operating system is unable to set arbitrary values in the PCRs.

Using the PCRs, the boot process for a platform establishes a chain of trust. The first code to execute on the device (e.g. the BIOS or EFI) is assumed to be trusted. It hashes its own code and configuration and puts the result into the PCRs. It then hashes the next stage of the boot process (usually the master boot record and bootloader) as well as its configuration (such as the boot drive's partition table), puts that result in the PCRs, and then hands off control to the next boot stage. That next boot stage is then responsible for hashing the next stage of the boot process (such as the Windows boot manager), putting that value in the PCRs, and then handing off control. In this fashion, each part of the boot sequence is responsible for fingerprinting the next and putting that result into a PCR before passing on control. This way any change to the boot sequence (such as using an attacker-controlled boot drive) will result in a different set of values in the PCRs.

When the TPM *seals* a secret value, it uses any (or all) of the PCRs as a parameter. The TPM will then only *unseal* the secret if the current values of the PCRs match the values at the time the secret was sealed. In effect, this means that only the software which stored the secret in the TPM to begin with is capable of retrieving it on subsequent boots. Not only does this prevent an alternate operating system from extracting the BitLocker key from the TPM, it also provides an implicit integrity check on the boot process which can detect forms of malware such as bootkits.

A simplified version of the boot process is shown in figure 3. This figure omits many values that go into PCR registers (such as configuration values, partition tables, etc) and collapses some boot stages, but illustrates the general procedure in which each component's signature is added to the TPM's PCRs before being given control. If any part of the boot process is modified on a subsequent boot (such as booting off a different disk or removable media), the PCR values will also be different and the TPM will not unseal the decryption key.

By using the chain of trust, the TPM can be used to store BitLocker's key and therefore avoid any need for the key to be specified by a user. Thus BitLocker's FDE can operate in a completely transparent way; many users may not even realize that FDE is enabled on the device. When a machine with BitLocker configured this way is turned on, it will boot all the way into the full op-



1. The BIOS/EFI hashes the next boot stage: the MBR, bootloader, etc. It puts the result in TPM PCRs. Control is passed to the next boot stage.
2. The Windows bootloader asks the TPM to *unseal* the BitLocker encryption key, which will verify the current PCR values. The returned key is stored in RAM, the operating system is decrypted, and control is passed to the operating system.
3. The operating system continues to decrypt disk contents on the fly using the key stored in RAM.

Figure 1: Simplified BitLocker boot process

erating system without any interaction, eventually presenting the user with the usual Windows login screen.

To an attacker, there are a number of components which may be attacked with the ultimate goal of accessing the plaintext user data on the disk. If early parts of the boot chain have a defect which can be compromised, or if the root trust object (e.g. the BIOS or EFI) can be written with an unsigned image, the attacker could spoof the remaining PCRs and then pass control to an attacker-controlled operating system which can simply request the BitLocker key from the TPM. There are a number of modern defenses that make this more difficult, such as UEFI Secure Boot [5].

The TPM itself could be attacked in an attempt to retrieve the decryption key. In 2007, Evan Sparks demonstrated an attack on the TPM which allowed him to reset the PCR register values by grounding a particular circuit line while the machine was already running [11]. Using this technique, an attacker could boot into a custom operating system, reset the PCR values to zero, and then set its own PCR values directly in order to unseal the key. This issue was fixed in the 1.2 TPM specification, but it does not preclude other hardware-based attacks.

Once the TPM has released the key to the operating system, the operating system will keep that key in memory to transparently encrypt and decrypt disk sectors on the fly. This means the RAM itself can be attacked to try and retrieve the key, such as in a cold boot attack [10].

Each of the above options presents significant difficulties, and Microsoft discusses how most of these attacks are mitigated by BitLocker's design [7]. Attacking the BIOS/EFI is platform specific and is likely not be possible if its images need to be signed by the manufacturer. Attacking later parts of the boot chain such as the master boot record or the Windows bootloader would likely be difficult because these components are small and single-purpose, leaving them with a relatively small attack surface. Finally, cold boot attacks are not totally reliable, involve a greater degree of sophistication, and may be impossible if the memory components of the device are non-removable and the system is configured to not boot off of any secondary devices.

In lieu of any of the above, the final target for attack would be the operating system itself which has been given the decryption key and is tasked with protecting it and using it responsibly. Since a machine with passwordless BitLocker will transparently retrieve the decryption key and boot to the Windows login screen, Windows authentication becomes the attack surface for defeating BitLocker in this paper. The next section describes the details of Windows domain authentication in order to provide insight into how the attack is able to bypass it.

4 Kerberos and Windows Domain Authentication

As mentioned above, when BitLocker is transparently enabled using the TPM the boot process will transparently retrieve the decryption key from the TPM and boot up as usual to the Windows login screen. To an attacker, the login screen is the only barrier between the attacker and the user's data.

In the absence of full disk encryption, local Windows authentication is not a true barrier; an attacker can simply change local account passwords on the disk and use those altered credentials to login. With BitLocker enabled, this is not a viable attack.

The login screen presents an attack surface in the form of domain authentication. Corporate environments use centralized domain accounts which can be used to authenticate with multiple machines and services. When logging into a machine using a domain account, that machine contacts the domain controller (DC) in order to verify the credentials provided, and may or may not grant access based on the DC's responses.

Windows authentication is locally handled by the Local Security Authority (LSA), an extensible system which can use any number of Security Support Providers (SSPs) [3]. A number of SSPs are included with Windows including Kerberos which is the default protocol used for client-domain authentication [8].

One could write a book to completely cover the Kerberos protocol (indeed, some have [9]), so this paper will only just cover the relevant parts of the protocol and do so at a very high level. Kerberos may have different servers such as the authentication server (AS) and the key distribution center (KDC), but for this summary we will treat both of these services as residing on the domain controller (DC). We will be abstractly referring to a "target service" to which a user may want to gain access, though in our current context the target service will just be the local workstation.

Kerberos manages mutual authentication using shared secrets; for user accounts this is generally a password hash. Each machine in the domain also has an account in the Kerberos database (formally called machine accounts) which also have a shared secret key (formally called machine passwords).

Kerberos operates using session-long authentication tokens called tickets. When a user first authenticates to a Kerberos domain (using their password), they are given a ticket-granting-ticket (TGT). This ticket is then used for the rest of the session instead of the user's password whenever requesting application authorization. In particular when requesting access to an application service (such as the login to the local workstation), the TGT is used to request a ticket for that service, and this service ticket is then verified by that target service.

This workflow is visually shown in figure 2. The response from the DC in step 2 includes both the TGT and an associated secret key encrypted with the user's password. The login screen will use the typed password to decrypt the session key from the response, the login failing at this point if the typed password is different from the one in the DC's account database. Similarly, the ticket T returned by the DC in step 4 has a portion signed with the target service's machine password. When this ticket is passed to the target service in step 5, that machine verifies T using its own copy of the machine password, thus verifying that the ticket was generated by the DC.

As noted above, the target service in our scenario is the local workstation, so an attacker can't intercept or manipulate the AP_REQ/AP_REP messages. However, the domain controller is remote, and since the attacker has physical control of the machine, the attacker also has control of network communication and can direct communication to an attacker-controlled "mock" domain controller. The mock domain controller can be configured by the attacker to respond to any traffic as though it were the true domain controller.

In this exchange, there are two unknowns to

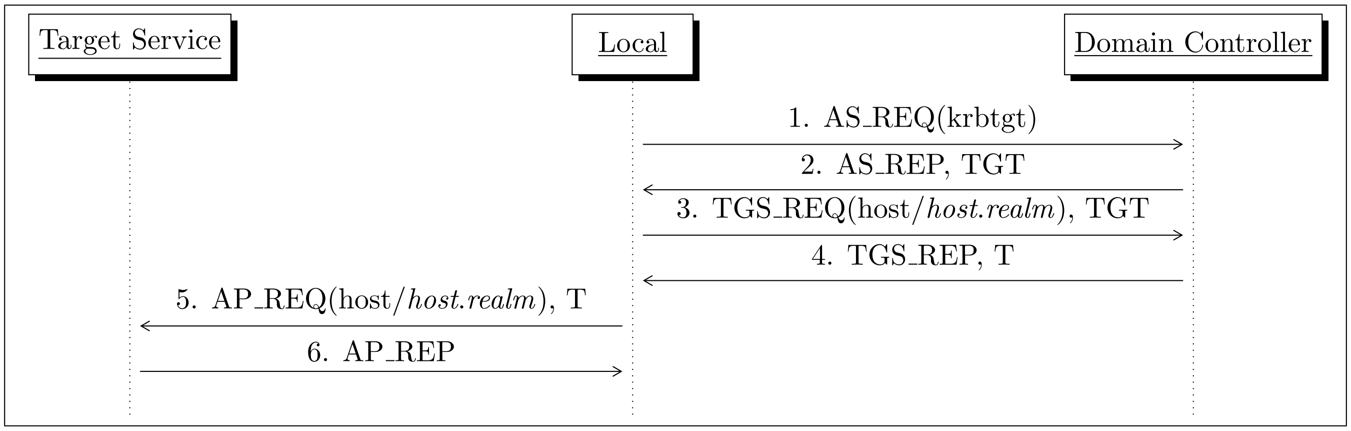


Figure 2: The Kerberos exchange for workstation login using domain credentials

an attacker: the user password (used to encrypt/decrypt the TGT key) and the machine password (used to encrypt/decrypt part of the ticket T). In our scenario however, both places where the user password is used is under control of the attacker. That is, the attacker will both be entering the password on the Windows login screen and will be setting the user password in the mock DC account database. However, the machine password is saved on the (encrypted) workstation and is therefore unknown to the attacker. If the DC uses a different machine password when creating the ticket T, the machine will be unable to verify the ticket and Windows will display the following message to the user on the login screen:

> "The trust relationship between this workstation and the primary domain failed."

So even with a mock domain controller, an attacker will be unable to generate a ticket T that the machine will accept.

Omitted from the discussion so far is what happens when authenticating with a domain account when the domain controller is unreachable. For instance, if a laptop is being used in a hotel and is not on the corporate network or is simply offline, users are still able to unlock the machine using domain credentials. This is because Windows will (by default) cache credentials and use that cache when the DC is unreachable [2].

The credential cache will be the target of the attack in this paper. By poisoning the local credential cache with an attacker-known password, the attacker is then able to log in to the system using that password. This attack is achieved using the password change Kerberos protocol to poison the credentials cache. When changing a user's password, the workstation instead of requesting a TGT will request an application ticket for the `kadmin/changepw` service. The DC will supply such a ticket (again, with a secret encrypted using the user's current password), and the workstation will use this in the request to set a new password with the DC. However, in no part of this exchange is a machine password involved. Nonetheless, when the password change protocol is complete the Kerberos SSP updates the local credential cache with this new password.

The password reset exchange for Windows domains using Kerberos is described in full detail in RFC 3244 [12]. Since there is no machine password involved, this is an exchange which the mock DC can participate. Getting the local workstation to participate in this exchange just requires the DC to announce that the user's password has expired, which will then cause the workstation to prompt the user for a new password. Using this prompt, the attacker can initiate the exchange. Fundamentally, this is the root of the issue described in this paper: the password reset exchange does not require the DC to provide authentication (i.e. an unknown machine password) and the client-side implementation of this exchange will update the local credentials cache after a successful exchange. This exact procedure for exploiting this defect is detailed in the next section.

5 The Attack

As described in the prior sections, the attack to bypass local authentication and thereby defeat BitLocker's full disk encryption assumes the following conditions:

1. BitLocker is enabled without pre-boot authentication, so the attacker is able to boot up the machine to the login screen.

2. The machine has joined a domain and an authorized domain user has previously logged into the machine.

Assuming these conditions apply, the attack simply requires setting up a mock domain controller with the target user account's password set as expired. The shell commands in figure 3 is an example of how to do this using Samba on a Debian-based Linux distribution. The example assumes the target machine is attached to the `MISKATONIC` domain and the user account name is `ihaken`. When performing this attack against an actual target, the attacker could retrieve the domain and username from network traffic (both will show up in plaintext as part of DNS/Kerberos protocols) or just by reading both names off of the login screen on the target machine.

The next step is to connect the target machine to the network where this DC advertises its presence. Once connected, the attacker logs in to the target machine using the password specified when setting up the domain user account (in the example in figure 3: "`password!23`"). Because the password is considered expired, the target machine will prompt the attacker to set a new password (which the attacker will set to an arbitrary, attacker-known value). Although the login will still fail (because the machine password on the DC is absent), the new user password value nonetheless poisons the local credentials cache. Thus, the final step is to disable the machine's network connection and login with the new password, which will be validated against the poisoned cache.

Once logged in, the attacker now has access to all of the user's data, such as emails, intellectual property, saved passwords, cached credentials, etc. If the user is a local administrator, the attacker could even dump the BitLocker key from kernel memory. Though as noted by Microsoft [7]:

> "When an attacker has sign-in access to the PC, there are few reasons for the attacker to decrypt the drive, because they would already have full access to the data within it."

In a situation where the victim does not realize their machine is under attack (e.g. the machine has been left temporarily unattended), the attacker could install malware on the device, such as a keylogger or backdoor, and return to the device to the unwitting victim (commonly called an "evil maid" attack).

This attack is 100% reliable on effected systems, is not sophisticated (no custom tool was developed, no patches to Samba were necessary, and it was executed with just a few shell commands), and can be executed in a matter of seconds. Configuring Samba manually, this can be done in under a minute. If a tool was written to respond automatically based on DNS/Kerberos requests (thus automatically determining the domain/realm and the username), this could be used to bypass the login screen in a matter of seconds.

Although this attack is specifically a bypass of Windows authentication, it is primarily of interest as a bypass of BitLocker's full disk encryption. It assumes physical access to the target machine, and the only context in which this assumption does not already imply total access is when the disk is encrypted. However, this authentication bypass applies in general and is conceivably a vector for accessing a user's machine/data in a less invasive manner than rebooting the machine into an attacker-controlled operating system (the typical attack vector when given physical access to an unencrypted device). For example, if a laptop were left unattended only briefly, an automated version of this exploit could allow an attacker to bypass the lock screen and plant malware on the computer in a matter of seconds, whereas an alternate attacks requiring a full reboot would be too time-consuming and more noticeable.

6 Impact and Mitigations

As described, any system using BitLocker without pre-boot authentication and using domain credentials is vulnerable to this attack. In testing, all versions of Windows since Vista (Vista, 7, 8, and 10)

```
$> apt-get install -y samba > /dev/null
$> samba-tool domain provision
Realm [MISKATONIC.UNI]: MISKATONIC.UNI
 Domain [MISKATONIC]: MISKATONIC
 Server Role (dc, member, standalone) [dc]:
 DNS backend (SAMBA_INTERNAL, BIND9_FLATFILE, BIND9_DLZ, NONE) [SAMBA_INTERNAL]:
 DNS forwarder IP address [192.168.56.10]: 192.168.56.1
Administrator password:
Retype password:
$> smbpasswd -a -n ihaken
Added user ihaken.
$> NOW=`date`; date -s '2001-01-01 00:00:00'
$> echo -e 'password!23\npassword!23' | smbpasswd -s ihaken
$> date -s "$NOW"
```
Figure 3: Example configuration of the mock domain controller for the attack

are vulnerable. On such machines, any data that can be read by the domain account is then readable by the attacker. If the domain account has local administrative rights, this means all data on the drive can be read. Essentially, BitLocker provides no protection on these systems.

Microsoft has investigated this issue and is planning to release an update which prevent this exploit in November 2015. As usual, the most important security procedure is to make sure you have applied all security updates to your effected systems.

Until the update is released, there is no simple solution to this problem. If data loss prevention is critical, you can enable pre-boot authentication with BitLocker, i.e. require a PIN or USB key. However, as noted in Microsoft's notes above, this increases the risk and cost of using BitLocker, and it introduces a great deal of friction for users.

Similarly, simply using a BIOS boot password would prevent this attack (since it would prevent an attacker from booting the machine at all), but deploying and managing machine BIOS passwords corporation-wide may not be practical for reasons similar to managing pre-boot authentication.

It is also possible to disable cached credentials in the Windows LSA or (equivalently) to require authentication with the DC in order to login. This also prevents this attack, but means users will not be able to use their machines when offline or otherwise disconnected from the corporate network. Since this would make a laptop unusable outside of the office, this is generally not an acceptable option.

This attack is a exploiting an artifact of the underlying protocol used for client-domain communication which has existed since this part of the protocol was first designed for Windows 2000. Indeed, in testing this attack was effective on all versions of Windows since Windows 2000 (even though this is wasn't relevant until BitLocker was introduced with Windows Vista).

I suspect this weakness in the protocol comes about because this threat model was not applicable when it was originally designed. At that point in time, an attacker having physical control of a client machine meant it was already totally compromised. However, as this paper demonstrates, the threat model needs to be revisited. What was once a perfectly reasonable protocol breaks down under this revised threat model. This is a good lesson to be taken from this research. Threat models can change over time even when the software in question doesn't. However, when threat models change, the security architecture of applications may need to be carefully revised along with them.

References

[1] "BitLocker Countermeasures." [Online]. Available: `https://technet.microsoft.com/en-us/library/dn632176.aspx`

[2] "Cached credentials security in Windows Server 2003, in Windows XP, and in Windows

2000." [Online]. Available: `https://support.microsoft.com/en-us/kb/913485`

[3] "Logon and Authentication Technologies," [Online]. Available: `https://technet.microsoft.com/en-us/library/cc780455%28v=ws.10%29.aspx#w2k3tr_sec_authn_over_tthr`

[4] "Protect BitLocker from Pre-Boot Attacks," [Online]. Available `https://technet.microsoft.com/en-us/library/dn632180.aspx`

[5] "Secure Boot Overview," [Online]. Available: `https://technet.microsoft.com/en-us/library/hh824987.aspx`

[6] "TPM Library Specification," [Online]. Available: `https://www.trustedcomputinggroup.org/resources/tpm_library_specification`

[7] "Types of Attacks for Volume Encryption Keys," [Online]. Available: `https://technet.microsoft.com/en-us/library/dn632182.aspx`

[8] "What Is Kerberos Authentication?" [Online]. Available: `https://technet.microsoft.com/en-us/library/cc780469%28v=ws.10%29.aspx`

[9] Jason Garman. *Kerberos: The Definitive Guide*. O'Reilly Media, Aug. 2003.

[10] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, et al. "*Lest We Remember: Cold Boot Attacks on Encryption Keys*," Proc. 17th USENIX Security Symposium (Sec '08), San Jose, CA, July 2008. Available: `https://citp.princeton.edu/research/memory/`

[11] Evan R. Sparks. "*A Security Assessment of Trusted Platform Modules*," June 2007. Available: `http://www.cs.dartmouth.edu/~pkilab/sparks/`

[12] M. Swift, J. Trostle, and J. Brezak, "Microsoft Windows 2000 Kerberos Change Password and Set Password Protocols," IETF, RFC 3244, Feb. 2002. [Online]. Available: `https://www.ietf.org/rfc/rfc3244.txt`