# DIGITAL 'BIAN LIAN' (FACE CHANGING): THE SKELETON KEY MALWARE

*Chun Feng*
Microsoft, Australia

*Tal Be'ery*
Microsoft, Israel

*Stewart McIntyre*
Dell SecureWorks, UK

Email {chfeng,talbe}@microsoft.com; smcintyre@secureworks.com

## ABSTRACT

Bian Lian (face changing) is an ancient Chinese dramatic art that stems from Sichuan opera – where performers can change their face masks almost instantaneously.

Interestingly, this 'face-changing' trick is not only used in Sichuan opera, it has also been adopted in the digital world by malware. A new breed of advanced persistent threat (APT), discovered by *Dell SecureWorks* and known as 'Skeleton Key', is using this face-changing trick.

When the Skeleton Key malware is installed on a domain controller (DC), the attacker can play the face-changing trick on the domain by logging in as any user it chooses and performing any number of actions on the system including, but not limited to, sending/receiving emails, accessing private files, local logging into computers in the domain, unlocking computers in the domain, etc.

This paper analyses the technical details of the Skeleton Key malware. It unveils the tricks used by the malware to tamper with NT LAN Manager (NTLM) and Kerberos/Active Directory authentication. In particular, it details the tricks used by the malware to downgrade the encryption algorithm used by Kerberos, from AES to RC4-HMAC (NTLM).

The Skeleton Key malware can be removed from the system after a successful infection, while leaving the compromised authentication in place. This can pose a challenge for anti-malware engines in detecting the compromise. This paper also discusses how on-the-wire detection and in-memory detection can be used to address some of these challenges.

## EMERGENCE OF SKELETON KEY MALWARE

*Windows* servers have been widely deployed, and are commonly used by organizations as part of network infrastructures. In particular, 'a domain controller (DC) is a server that is running a version of the *Windows Server* operating system and has Active Directory Domain Services installed' [1]. The DC is responsible for security authentications in a *Windows* domain.

In January 2015, *Dell SecureWorks* researchers discovered a new breed of malware, a threat that specifically targets DCs, in an Advanced Persistent Threat (APT) attack [2]. Once installed on the DC, this malware, which is named 'Skeleton Key malware' (hereafter it will be referred to simply as Skeleton Key), allows the attacker to play a 'Bian Lian' (face-changing) trick – the attacker can log into the affected domain with any user account and perform any number of actions on the system including, but not limited to, sending/receiving emails, accessing private files, etc.

In the wild, Skeleton Key has been distributed as a 64-bit DLL file with the following file names:

- advapi64.dll
- appmgmt.dll
- msuta64.dll
- ntfrs.dll
- ole.dll
- ole64.dll
- UI0Detect.dll

Skeleton Key has two DLL exports: one is called for installation and other is used for uninstallation, for example, ii (for installation) and uu (for uninstallation).

According to the threat analysis published by *Dell SecureWorks* [3], the attacker may use the following process to deploy Skeleton Key:

1. Upload the Skeleton Key DLL file to a staging directory on a jump host in the victim's network.
2. Attempt to access the administrative shares on the domain controllers using a list of stolen domain administrator credentials.
3. If the stolen credentials are no longer valid, use password theft tools to extract clear text domain administrator passwords from one of the following locations (which suggest a familiarity with the victim's environment):
   - Memory of another accessible server on the victim's network
   - Domain administrators' workstations
   - Targeted domain controllers.
4. If the domain administrator credentials are valid, use these to copy the Skeleton Key DLL to C:\WINDOWS\system32\ on the target domain controllers.
5. Use the PsExec [4] utility to run the Skeleton Key DLL remotely on the target domain controllers using the rundll32 command. The attacker's chosen password is formatted as an NTLM password hash rather than being provided in clear text. After Skeleton Key is deployed, the attacker can authenticate as any user using the attacker's configured NTLM password hash:

```
psexec -accepteula \\%TARGET-DC% rundll32 <DLL
filename> ii <NTLM password hash>
```

6. Delete the Skeleton Key DLL file from C:\WINDOWS\system32\ on the targeted domain controllers.
7. Delete the Skeleton Key DLL file from the staging directory on the jump host.
8. Test for successful Skeleton Key deployment using 'net use' commands with an Active Directory (AD) account and the password that corresponds to the configured NTLM hash.

A version of Skeleton Key malware observed by *Dell SecureWorks* was implicated in domain replication issues that

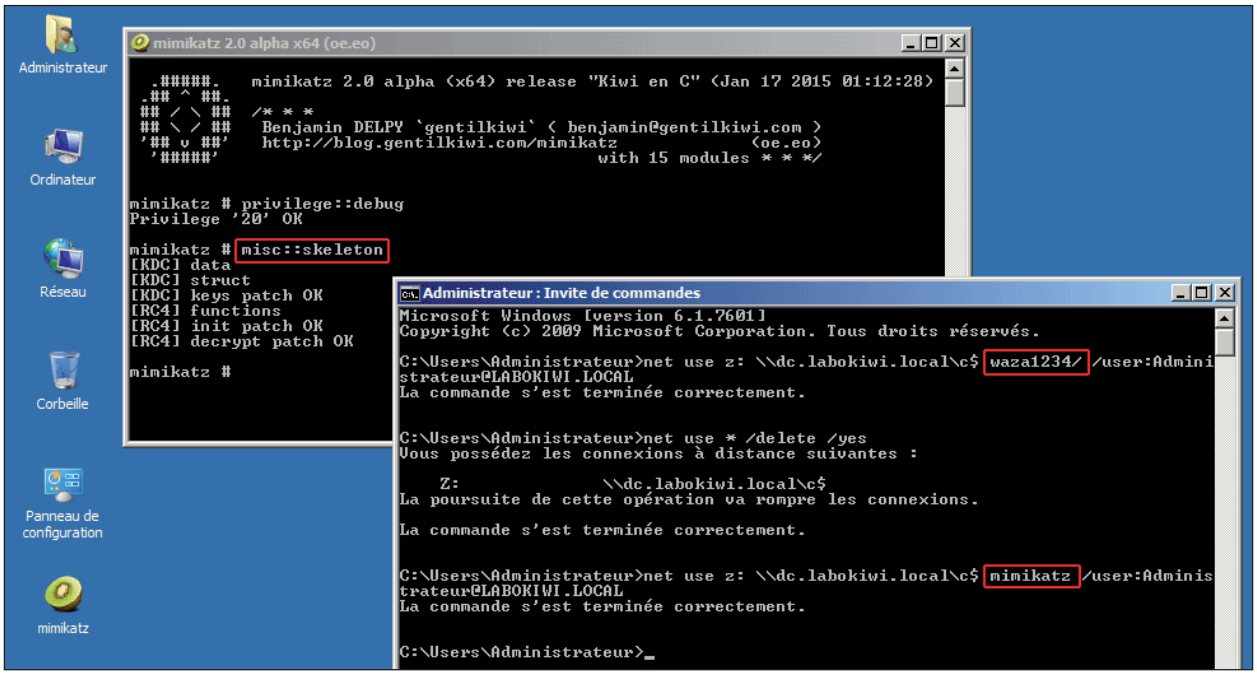


*Figure 1*: *Skeleton Key feature in Mimikatz [5].*

may indicate an infection. Shortly after each deployment of Skeleton Key malware observed by *Dell SecureWorks* CTU analysts, domain controllers experienced replication issues that could not be addressed by *Microsoft* support, and eventually required a reboot to resolve. These reboots removed Skeleton Key's authentication bypass because the malware does not have a persistence mechanism.

These replication issues may be explained by subtle bugs in this version of Skeleton Key's patches to the Active Directory's authentication functions, which allow most authentication use cases to operate as normal, but which cause replication between domain controllers eventually to cease functioning.

### Mimikatz support – a copycat version of Skeleton Key

As a result of the Skeleton Key malware disclosure, very similar functionality[1] was added to *Mimikatz*, a popular penetration testing tool (see Figure 1). Subsequently, Skeleton Key functionality is now publicly available to both penetration testers and attackers.

## WINDOWS AUTHENTICATION INTERNALS

Before we dive into the details of how Skeleton Key works, let's explore the internals of *Windows* authentications.

All current *Windows* operating systems use authentication packages to analyse logon data [6]. Authentication packages are contained in dynamic link libraries (DLLs) which are loaded into the Local Security Authority Subsystem Service (LSASS) process and client processes. According to the Kerberos Network Authentication Service, '*Windows* uses two standard authentication packages for interactive logons.' [7] In these two standard authentication packages, two authentication protocols are implemented:

- Kerberos authentication (implemented in the Kerberos authentication package)
- NTLM authentication (implemented in the MSV1_0 authentication package).

### Kerberos authentication

Kerberos is an authentication and authorization protocol, standardized and maintained by the IETF (Internet Engineering Task Force – mainly in RFC 4120 [8]) and implemented by many operating systems (OS), including but not limited to *Windows*, *Linux* and *Mac OSX*. Since *Windows 2000*, Kerberos authentication has been the default authentication method for domain accounts.

The Kerberos authentication and authorization protocol enables the transparent single sign-on (SSO) experience. The SSO enables users to actively authenticate (i.e. provide their password) only once even though they access various services – whether in the corporate network or in the cloud (where the Kerberos ticket is translated to SAML tokens).

The Kerberos authentication and authorization protocol works in the following manner:

1. The user provides the domain name, username and password to access their computer.
2. The computer authenticates to the authentication server (AS) residing on the key domain controller (KDC). Accordingly, the KDC provides the computer with a ticket granting ticket (TGT). The TGT is an identifier which enables the computer to request access to services without the user having to re-supply their credentials.
3. Each time the computer attempts to access a service, it first identifies itself to the domain controller (DC), residing on the KDC, with the TGT as provided earlier by the AS. The DC, through its ticket granting server (TGS), provides the user with a ticket for the particular requested service.

[1] For Kerberos authentication, NTLM authentication is not supported.

4. The user provides the service ticket to the service. Since the ticket was validated by the TGS, the service grants authorization. Accordingly, the connection between the user and the service is established.

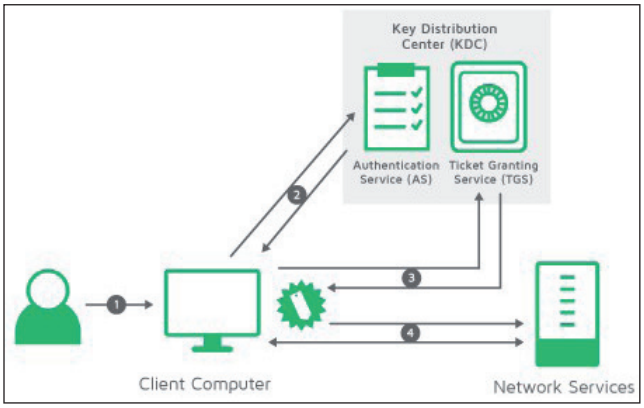


*Figure 2: Kerberos authentication flow.*

In *Windows* networks, the KDC is implemented in the Active Directory (AD) service and is installed on the DC. Therefore, we will use KDC, DC and AD interchangeably throughout this document.

In Kerberos (version 5) the server uses pre-authentication in order to validate the user identity before sending any encrypted data, encrypted with the user's long-term key (derived from the user's password).

When using passwords to authenticate, the pre-authentication method is the encrypted timestamp. The user encrypts the current timestamp with its long-term key as a proof of password knowledge.

To support this process, a handshake phase must be supported by the protocol:

- To determine a common encryption algorithm: Since the Kerberos protocol supports multiple encryption algorithms, a handshake phase is needed to determine an encryption type (EType) that can be used by both sides of the transaction.
- To determine the user's salt: Salt is a unique, non-secret string added to the password in the key derivation function, in order to ensure that two different users with the same password will not have the same key. The salt is determined by the encryption type. Some encryption algorithms do not support salt at all (RC4-HMAC).

The details of the Kerberos handshake phase are as follows:

1. The client (in most cases, the client is the *Windows* OS native client) first sends an AS-REQ message detailing the encryption types supported by it (Figure 3).

```
as-req
    pvno: 5
    msg-type: krb-as-req (10)
  padata: 1 item
    PA-DATA PA-PAC-REQUEST
  req-body
      Padding: 0
    kdc-options: 40810010 (forwardable, renewable, canonicalize, renewable-ok)
    cname
      realm: aorato.research
    sname
      till: 2037-09-13 02:48:05 (UTC)
      rtime: 2037-09-13 02:48:05 (UTC)
      nonce: 160211996
    etype: 6 items
        ENCTYPE: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
        ENCTYPE: eTYPE-AES128-CTS-HMAC-SHA1-96 (17)
        ENCTYPE: eTYPE-ARCFOUR-HMAC-MD5 (23)
        ENCTYPE: eTYPE-ARCFOUR-HMAC-MD5-56 (24)
        ENCTYPE: eTYPE-ARCFOUR-HMAC-OLD-EXP (-135)
        ENCTYPE: eTYPE-DES-CBC-MD5 (3)
```

*Figure 3: Initial AS-REQ with the client's supported ETypes.*

```
krb-error
    pvno: 5
    msg-type: krb-error (30)
    stime: 2014-03-10 20:05:07 (UTC)
    susec: 165032
    error-code: eRR-PREAUTH-REQUIRED (25)
    realm: aorato.research
  sname
  e-data: 30543031a103020113a22a04283026301da003020112a116...
    PA-DATA PA-ENCTYPE-INFO2
      padata-type: kRB5-PADATA-ETYPE-INFO2 (19)
        padata-value: 3026301da003020112a1161b14414f5241544f2e52455345...
          ETYPE-INFO2-ENTRY
            etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
            salt: AORATO.RESEARCHbugsb
          ETYPE-INFO2-ENTRY
            etype: eTYPE-ARCFOUR-HMAC-MD5 (23)
```

*Figure 4: DC's PA-ETYPE-INFO2.*

```
as-req
    pvno: 5
    msg-type: krb-as-req (10)
  padata: 2 items
    PA-DATA PA-ENC-TIMESTAMP
      padata-type: kRB5-PADATA-ENC-TIMESTAMP (2)
      padata-value: 3041a003020112a23a0438c871bc029b90195c7d2981b0cd...
          etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
          cipher: c871bc029b90195c7d2981b0cd8e4c98fa5fa747689f86e1...
```

*Figure 5: Timestamp encrypted in the mutually agreed EType.*

2. The DC responds with a Kerberos error message, detailing (within the PA-ETYPE-INFO2 field) all the encryption types in which the user keys are stored within the DC database which were also present in the client's AS-REQ message (Figure 4).
3. The client selects one of the DC-returned encrypted types, encrypts the timestamp with it and sends it as a pa-enc-timestamp (the 'KRB5-PADATA-ENC-TIMESTAMP' field in Figure 5) in AS-REQ.

### NTLM authentication

NTLM is a challenge/response-based authentication protocol. The NTLM authentication protocol works in the manner shown in Figure 6.

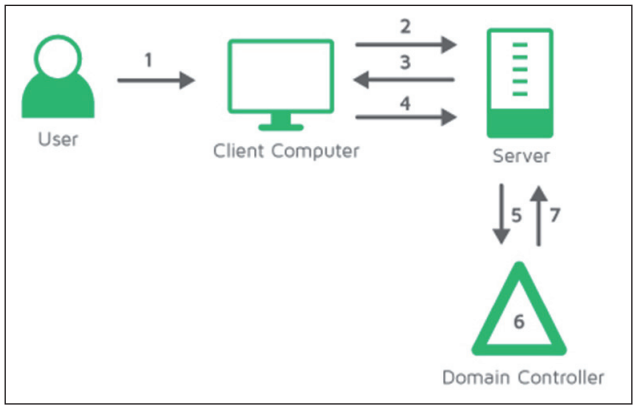


*Figure 6: NTLM authentication flow.*

1. The user provides a domain name, username and password to access their computer. The computer computes a cryptographic hash (often referred to as the NTLM/NT hash) of the password and discards the actual password.
2. When the user attempts to access the server, the computer sends the username to the server in clear text.
3. The server generates a 16-byte random number, called a 'challenge', and sends it to the user.
4. The user encrypts this challenge using the password hash and returns it to the server (a.k.a. 'response').
5. The server sends the following three items to the domain controller:
   - Username
   - Challenge sent to the client
   - Response received from the client.
6. The domain controller uses the username to retrieve the hash of the user's password from the security account manager database. It uses this password hash to encrypt the challenge.
7. The domain controller compares the encrypted challenge it computed (in step 6) to the response computed by the client (in step 4). If they are identical, authentication is successful and the domain controller notifies the server.

NTLM is an older authentication protocol than Kerberos authentication protocol [9]. While Kerberos is now the default authentication protocol for *Windows*, NTLM is still supported for backwards compatibility. NTLM may be used in the following situations:

- The service is not Kerberos-enabled.
- The service/server does not have a Service Principal Name (SPN) registered or it has duplicate SPNs registered.
- The service is accessed by IP address rather than by name.
- When the client can't access a KDC/DC, such as when it is outside the firewall or behind NAT.

On the *Windows* operating system, NTLM authentication is implemented in the MSV1_0 authentication package. For local *Windows* logon, Local Security Authority (LSA) and security accounts manager (SAM) on the local computer are used for authentication. For domain-user logon, pass-through authentication is used. The NTLM authentication request / response is passed from/to the local computer to/from the remote DC by MSV1_0 authentication packages using the Netlogon service (see Figure 7).

## FACE-CHANGING TRICKS

UNIX and UNIX-like systems have the 'su' command, which allows the root user to perform 'face changing' as any user without providing the user's password. This means on a UNIX and UNIX-like system, once the root is compromised, the attacker can easily pretend to be any user simply by executing the 'su' command. However, on the *Windows* operating system, there is no such command which allows the administrator to perform 'face changing' (though *Windows* allows a process to be executed with the 'runas' option, it

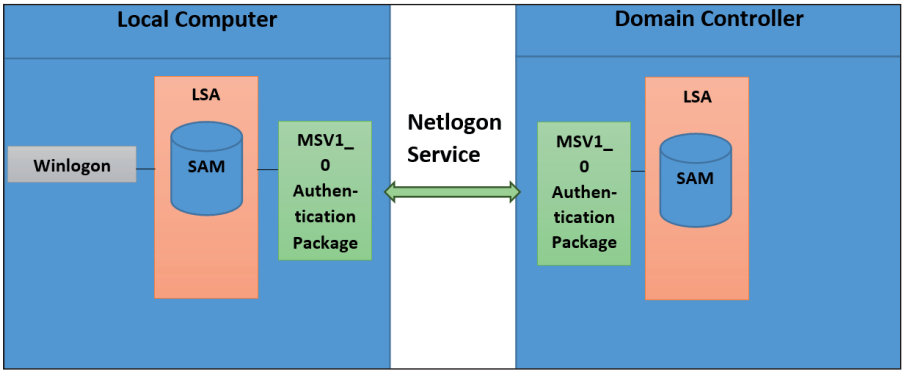


*Figure 7: MSV1_0 authentication package.*

only changes the permission but not the user identity). The attacker has devised Skeleton Key to allow the attack to perform the 'face-changing trick'. Skeleton Key is designed to tamper with NTLM and Kerberos authentication.

Skeleton Key has been designed to meet the following principles:

1. The domain user can still log in with their username and password so it won't be noticed.
2. The attacker can log in as any domain user with the Skeleton Key password.
3. Otherwise, the login would fail, i.e. if the domain user is neither using the correct password nor the Skeleton Key password, the login would fail.

## Face-changing trick for NTLM authentication

Skeleton Key malware performs the following steps to tamper with NTLM authentication and allow the attacker to log in with the Skeleton Key password:

1. It attaches to the memory space of the lsass.exe process.
2. It locates the non-exported function MsvpSamValidate() in DLL MSV1_0.dll by searching for byte patterns.
3. In the code of the MsvpSamValidate() function, there is a call to the MsvpPasswordValidate() function. It patches the call to invoke a hooked version of MsvpPasswordValidate(), which stays in a section of memory allocated by VirtualAllocEx().

The hooked version of MsvpPasswordValidate() first calls the original MsvpPasswordValidate() function. Hence if the domain users are trying to log in with their own domain username/password, it still works as expected. If this validation fails, then the hooked version of MsvpPasswordValidate() calls the original MsvpPasswordValidate() for the second time. However, rather than using the NTLM hash retrieved from SAM, it uses the NTLM hash supplied from the command line when running Skeleton Key for comparing. This means if the attacker is attempting to log in with the Skeleton Key password, when DC is trying to validate the username and password from SAM, it would always pass the validation since the NTLM hash used for comparison has been replaced.

## Face-changing trick for Active Directory/ Kerberos authentication

There is one more challenge in tampering with Kerberos authentication: newer Kerberos encryption types (such as AES) require a salt string (usually the username) to be added to the key derivation function, to make the same passwords of different users create non-identical encryption keys.

To support a salt-enabled key-derivation function, the malware would need to do one of the following:

- Compute all of the domain users' Skeleton Keys offline and store them, which requires a lot of memory.
- Compute the relevant user's Skeleton Key in real time, which is likely to cause performance issues on the DC, as the AES key derivation function (PBKDF2 [10]) is designed to be costly.

Therefore, the Skeleton Key malware chooses to support only RC4-HMAC-based Kerberos authentication, as RC4-HMAC's key-derivation function does not involve a user-based salt. As a result, the Skeleton RC4-HMAC key remains the same for all users, which greatly simplifies the malware's implementation.

In order to support Skeleton Key for Kerberos authentication:

- To make sure the users will authenticate using RC4-HMAC encryption instead of AES encryption, the SamIRetrieveMultiplePrimaryCredentials() function is hooked. The hooked SamIRetrieveMultiplePrimaryCredentials() checks for the package name 'Kerberos-Newer-Keys' [11], and returns STATUS_DS_NO_ATTRIBUTE_OR_VALUE to effectively disable AES-based authentication. The code of the hooked SamIRetrieveMultiplePrimaryCredentials() is shown in Figure 8.

```
int __fastcall hooked_SamIRetrieveMultiplePri
maryCredentials(unsigned int arg_UserHandle,
unsigned int arg_nPackages, PUNICODE_STRING arg_
PackageNames, unsigned int arg_Credentials)
{
    unsigned int Credentials; // edi@1
    PUNICODE_STRING PackageNames; // rbx@1
    unsigned int nPackages; // esi@1
    unsigned int UserHandle; // ebp@1
    unsigned __int16 *v8; // rax@2
    int result; // eax@7
    HLOCAL (__stdcall *v10)(UINT, SIZE_T); //
rax@8

    Credentials = arg_Credentials;
    PackageNames = arg_PackageNames;
    nPackages = arg_nPackages;
    UserHandle = arg_UserHandle;
    if ( arg_PackageNames
      && (v8 = arg_PackageNames->Buffer) != 0i64
      && arg_PackageNames->Length == 38
      && *v8 == 'K'
      && v8[7] == 's'
      && v8[15] == 'K' )  // Kerberos-Newer-Keys
    {
        result = STATUS_DS_NO_ATTRIBUTE_OR_VALUE;
  }
  …
```

*Figure 8: Hooked SamIRetrieveMultiplePrimaryCredentials() function.*

- CDLocateCSystem(unsigned int dwEtype, PCRYPTO_SYSTEM *ppcsSystem) in cryptdll.dll is patched. The patched CDLocateCSystem() hooks the Decrypt and Initialize function pointer fields in the CRYPTO_SYSTEM structure for dwEtype == 0x17 (RC4_HMAC). The hooked Decrypt function first calls the original Decrypt function to make sure the users can still log on with their original username and password. If this fails, it replaces the password hash with the supplied Skeleton Key NTLM hash (which is the same as the RC4-HMAC key) and calls the original Decrypt function again. So in this second call, the Decrypt function will always return success (NT_SUCCESS) since the NTLM hash (Skeleton Key NTLM hash) matches the password (Skeleton password). Hence the attacker can always pass the Kerberos authentication successfully with the Skeleton Key password.

## DETECTING 'FACE CHANGING'

The Skeleton Key malware can be removed from the system after a successful infection, while leaving the compromised authentication in place. This can pose a challenge for anti-malware engines in detecting the compromise. However, it is still possible to detect the presence of Skeleton Key malware using 'over-the-wire detection' or 'in-memory detection'.

### Over-the-wire detection

The key for the over-the-wire detection is the fact that the malware downgrades the Kerberos encryption types supported by the DC. This detection method is only relevant for domains operating in a Domain Functional Level (DFL) that enables AES (DFL is greater than or equal to 2008 [12]).

The detection of an encryption downgrade can be carried out in either a passive or an active manner.

#### *Passive downgrade detection*

A passive monitoring device situated in front of the DC can monitor ingress and egress traffic and detect the attack using the following algorithm.

1. Get the client advertised ETypes from the user's AS-REQ and check for AES support.
2. Get the DC advertised ETypes from the PA-ETYPE-INFO2 in the corresponding DC Kerberos Error.
3. If the client advertised its support for AES and the DC is known to have AES keys for this user, and yet this encryption type does not appear in the DC advertised response, then it's a good indication that some external party had fiddled with the client keys stored in the DC.

Note that no access to the client's or DC's secrets (passwords) is required in order to implement this algorithm

#### *Active downgrade detection*

An active approach to detection is to make an authentication request as a user using the AES encryption and observe a downgrade to RC4 by the DC. We have shared a publicly available standalone detection script [13] which is capable of detecting such user-encryption downgrade on DC.

The script works as follows:

1. Verifies whether the Domain Functional Level (DFL) of the current domain supports AES (>= 2008).
2. Finds an AES-supporting account (msds-supportedencryptiontypes [14] >= 8).
3. Sends a Kerberos AS-REQ to all DCs with only AES EType supported.
4. If AS-REQ fails due to AES encryption not being supported, then there's a good chance the DC is infected.

Note that no access to the client's or DC's secrets (passwords) is required in order to implement this algorithm.

### In-memory detection

Skeleton Key can be also detected in memory by checking the existence of some functions hooks. It hooks the following functions:

- cryptdll.dll!CDLocateCSystem
- samsrv.dll!SamIRetrieveMultiplePrimaryCredentials
- samsrv.dll!SamIRetrievePrimaryCredentials

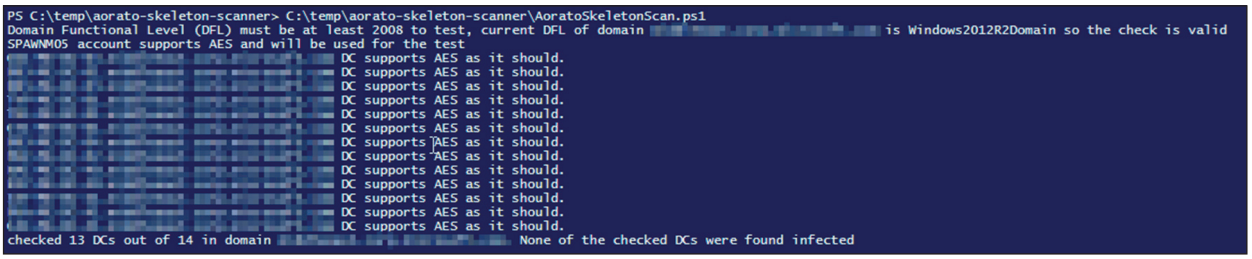


*Figure 9: Execution of the script on all DCs of a non-infected domain.*

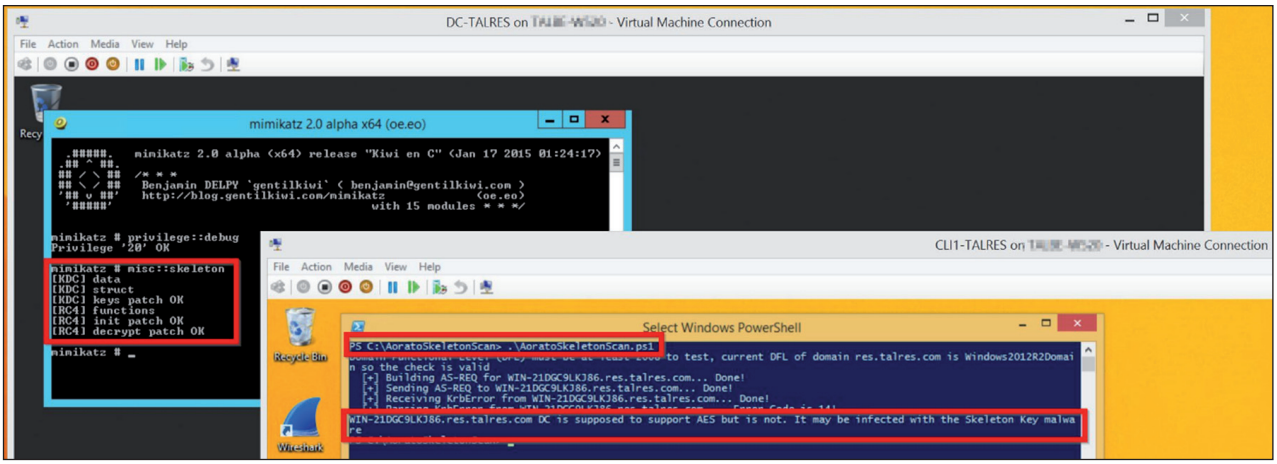


*Figure 10: Execution of the script on a Mimikatz Skeleton.*

Skeleton Key uses the Import Address Table (IAT) hooking method to hook these functions. It does the IAT hooking using the following steps (see Figure 11):

1. It enumerates the modules in the lsass.exe process and finds the kdcsvc.dll module.
2. It enumerates the Import Table of kdcsvc.dll and locates the IAT entries for the functions it needs to hook (aforementioned).
3. It overwrites the IAT entry with the hooked function address.

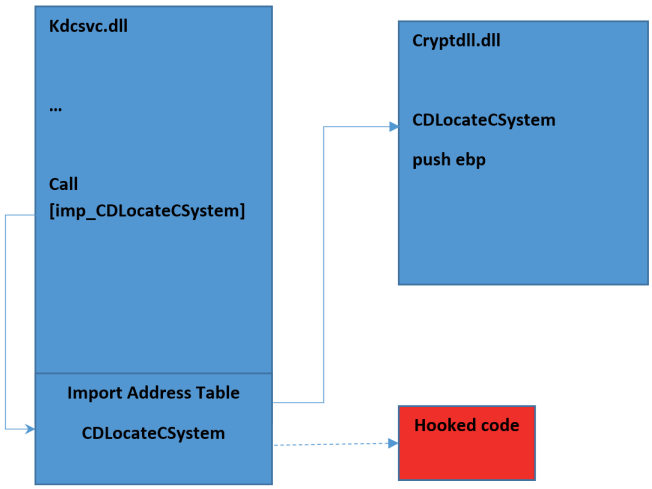


*Figure 11: Import Address Table hooking.*

The hooked function remains in an allocated memory region (returned from VirtualAllocEx()), so if any of the API's IAT entries falls in an allocated memory region, then it's an indication of Skeleton Key infection.

### Unexpected Windows Service Control Manager events

*Dell SecureWorks* observed the Skeleton Key attacker using the PsExec remote administration tool to remotely deploy Skeleton Key to target domain controllers. PsExec creates a service named PSEXESVC on the target host, which will create events in the target host's Windows Service Control Manager log. Using a SIEM solution to log events from key servers, like domain controllers, and alerting on anomalous Windows Service Control Manager events may allow the detection of attacks like Skeleton Key.

### Unexpected use of administrator credentials

The attacker requires domain administrator credentials to deploy the Skeleton Key to domain controllers. If privileged User Access (PUA) management software is deployed at an organization, investigating alerts on the unauthorized use of domain administrator credentials may identify attacker activity.

## CONCLUSION

This paper unveils Skeleton Key, a piece of malware designed to tamper with the authentication on domain controllers. Skeleton Key is able to tamper with NTLM authentication and Kerberos authentication, and allows the attacker to log in as any user on the affected domain. The Skeleton Key can be removed from the system after a successful infection while leaving the compromised authentication in place, which poses challenges for anti-malware engines in detecting it. In this paper we present two methodologies to detect it: on-the-wire detection and in-memory detection.

## REFERENCES

[1] Domain Controller Role. https://technet.microsoft.com/en-us/library/cc786438(v=ws.10).aspx.

[2] http://www.secureworks.com/cyber-threat-intelligence/threats/skeleton-key-malware-analysis/.

[3] http://www.secureworks.com/cyber-threat-intelligence/threats/skeleton-key-malware-analysis/.

[4] https://technet.microsoft.com/en-us/sysinternals/bb897553.aspx.

[5] https://twitter.com/gentilkiwi/status/556246876505509888.

[6] Authentication Packages. https://msdn.microsoft.com/en-us/library/windows/desktop/aa374733(v=vs.85).aspx.

[7] Russinovich, M.; Solomon, D.A.; Alex, I. Windows Internals (Part I, 6th edition).

[8] https://www.ietf.org/rfc/rfc4120.txt.

[9] NTLM hasn't been relevant for like 12 years... and other lies. http://markgamache.blogspot.com.au/2013/01/ntlm-hasnt-been-relevant-for-like-12.html.

[10] http://en.wikipedia.org/wiki/PBKDF2.

[11] https://msdn.microsoft.com/en-us/library/cc941808.aspx.

[12] https://technet.microsoft.com/en-us/library/understanding-active-directory-functional-levels%28v=ws.10%29.aspx.

[13] https://gallery.technet.microsoft.com/Aorato-Skeleton-Key-24e46b73/.

[14] https://msdn.microsoft.com/en-us/library/cc223853.aspx.