WATCHING THE WATCHDOG: PROTECTING KERBEROS AUTHENTICATION WITH NETWORK MONITORING

Authors:

Tal Be'ery, Sr. Security Research Manager, Microsoft

Michael Cherny, Sr. Security Researcher, Microsoft

November 2015

Contents

Introduction

Being the default authentication protocol for Windows-based networks, the Kerberos protocol is a prime target for advanced attackers. These attackers are trying to steal users' identity in order to gain access to the enterprise's most secured secrets.

Recently, it seems that advanced attackers had progressed from merely stealing the Kerberos related tokens to forgery attacks, in order to gain even greater powers within the domain.

In this document, we explore in detail three different forgery techniques, all found in the wild on attackers campaigns:

- Forged key: Skeleton Key
- Forget Ticket: Golden Ticket
- Forged PAC: MS14-068

Additionally, we suggest the “Diamond PAC” attack, a novel offspring of the Golden Ticket and Forged PAC attacks.

For each of the aforementioned attacks, we describe in detail a detection solution. All of the solutions are based only on the passive network monitoring of the Kerberos protocol traffic, and do not require any prior knowledge of the Kerberos secrets.

The Kerberos protocol

Kerberos is an Authentication and Authorization protocol, standardized and maintained by the IETF (mainly in RFC 4120[1]) and implemented by many Operating Systems (OS), including but not limited to Windows, Linux and Mac OSX.

The Kerberos protocol enables the transparent Single-Sign-On (SSO) user experience. The SSO enables users to actively authenticate (i.e. provide their password) only once even though they access various services.

Kerberos Message Flow

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

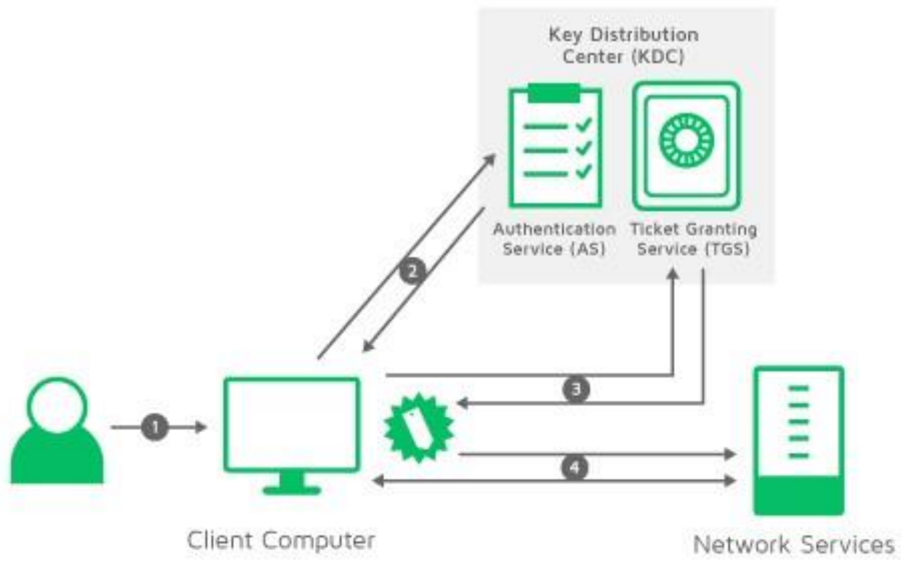


Figure 1 Kerberos Authentication Flow

1. The user provides the Domain Name, user name and password to logon to the computer.
2. The computer authenticates to the Authentication Server (AS) residing on the Kerberos Key Distribution Center (KDC). Accordingly, the KDC provides the computer with a Ticket Granting Ticket (TGT). The TGT is a token which enables the computer to request access to services without having the user to re-supply their credentials.

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

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

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

Note that for subsequent access requests only steps 3 and 4 are repeated and the Authentication Server (AS) is not involved in these transactions. The provided TGT is used as a proof that a successful authentication had taken place.

The user credentials are only used in the preliminary authentication stage. From thereafter, the Kerberos protocol only uses the TGT ticket. On the one hand, this feature improves the efficiency of the protocol. On the other hand, the TGT token now becomes a single point of failure in the authentication and authorization process.

The Kerberos TGT

The Kerberos TGT contains all of the user's relevant authentication and authorization information. This information enables the KDC to rely solely on the ticket information, without any need to make any further queries, thus improving the protocol's efficiency.

In particular, the TGT token contains the following fields:

- **Name**: The user's name the ticket is associated with
- **Start time and End time**: marks the validity period of the ticket. By default, in Windows networks, the validity period is set to ten hours.
- **Authorization-data**: Authorization data details the user's privileges and access rights. In Windows, the authorization data take the form of a Privilege Attribute Certificate (PAC) object. PAC[2] is a "Microsoft-specific authorization data present in the authorization data field of a ticket. The PAC

[2] http://msdn.microsoft.com/en-us/library/a61a2ecd-de81-4586-8db6-063873dd4910#privilege_attribute_certificate

contains several logical components, including group membership data for authorization, alternate credentials for non-Kerberos authentication protocols, and policy control information for supporting interactive logon."

To protect the TGT ticket from being fiddled with, the TGT ticket is encrypted with a key that is known to only the AS and the KDC. In Active Directory, this key is generated from the KRBTGT service account credentials.

Network Monitoring of the Kerberos Protocol

The network monitoring of the Kerberos protocol can be achieved through a passive Security Device (SD), monitoring Kerberos traffic through a network's switch port-mirroring.

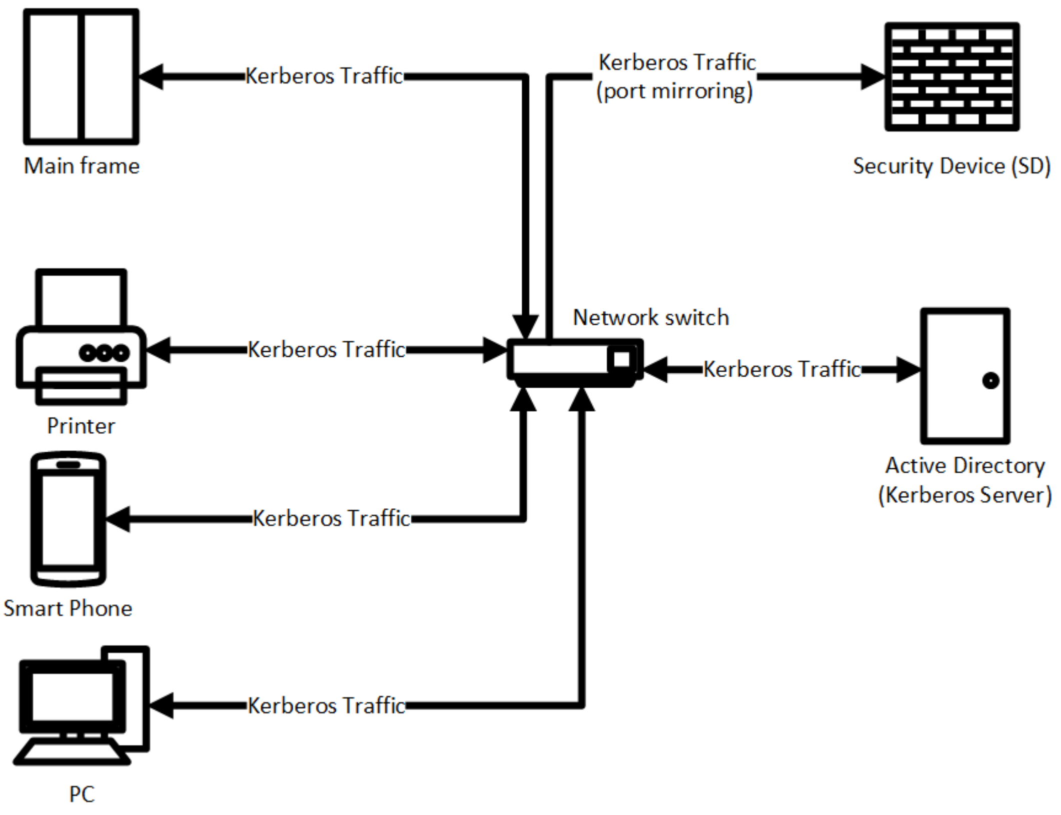


Figure 2 Network Monitoring of the Kerberos Protocol

Due to the standardization of the protocol, the same SD can monitor all of the Kerberos protocol traffic, regardless of the hardware and operating system of the originating device.

We will show below, that even though some portions of the Kerberos traffic are encrypted, the SD can detect the discussed attack, with no prior knowledge of Kerberos secrets.

Forged Key: The Skeleton Key Malware

On January 2015, Dell Secureworks had shared a report[3] on an advanced attack campaign utilizing a dedicated Domain Controller (DC) malware, named “Skeleton Key” Malware. The Skeleton Key malware modifies the DC behavior to accept authentications specifying a secret ”Skeleton key” (i.e. a “master key”) password, thus enabling the attackers to logon to any computer as any domain user without installing any additional malware. The normal users’ authentication experience remains the same, and is not affected by the malware in order to avoid detection.

The Skeleton Key Malware Technical details

The Skeleton Key malware has been designed to meet the following principles:

1. Domain users can still login with their user name and password so it won’t be noticed.
2. Attackers can login as any domain user with Skeleton Key password.
3. If the domain user is neither using the correct password nor the Skeleton Key password, the logon would fail.

To do so, the malware changes to the usual password validation process within the DC. If the usual check against the user’s key has failed, an additional check that uses the attackers’ key is performed.

However, there is one more challenge to tampering with Kerberos authentication: Kerberos newer encryption types (such as AES) require that a salt string (usually the user name) would 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 either:

- Compute all of the domain users’ Skeleton Key offline and store them, which requires a lot of memory; or
- 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[4]) 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.

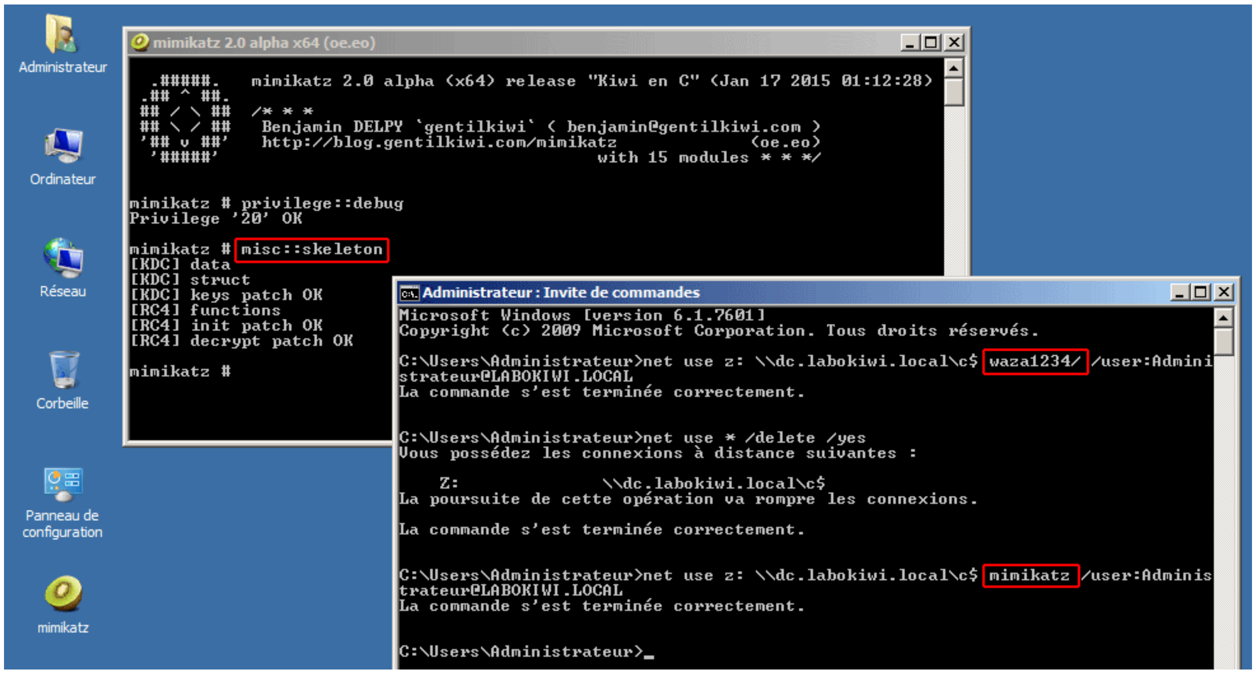
Therefore, to support Skeleton Key for Kerberos authentication, the malware:

- Hooks SamIRetrieveMultiplePrimaryCredentials() to make sure the users will authenticate using RC4-HMAC encryption instead of AES encryption. When the hooked SamIRetrieveMultiplePrimaryCredentials() function checks for the package name “Kerberos-Newer-Keys[5]”, it returns the STATUS_DS_NO_ATTRIBUTE_OR_VALUE value, to effectively disable AES based authentication.
- Patches CDLocateCSystem(unsigned int dwEtype, PCRYPTO_SYSTEM *ppcsSystem) in cryptdll.dll. The patched CDLocateCSystem() hooks the Decrypt and Initialize function pointers fields in

[3] http://www.secureworks.com/cyber-threat-intelligence/threats/skeleton-key-malware-analysis/
[4] http://en.wikipedia.org/wiki/PBKDF2
[5] https://msdn.microsoft.com/en-us/library/cc941808.aspx

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 logon with their original user name and password. If this fails, it replaces the password hash with the supplied Skeleton Key RC4-HMAC key and calls the original Decrypt function again. In this second call, decryption function will return success (NT_SUCCESS) if the skeleton password RC4-HMAC key was used to encrypt. Hence the attacker can always pass the Kerberos authentication successfully by supplying the Skeleton Key password.

As a result of the publication, a very similar Skeleton key functionality was added to Mimikatz[6], making it publicly available to both researchers and attackers.



```
  .#####.   mimikatz 2.0 alpha (x64) release "Kiwi en C" (Jan 17 2015 01:12:28)
 .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                   with 15 modules * * */

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # misc::skeleton
[KDC] data
[KDC] struct
[KDC] keys patch OK
[RC4] functions
[RC4] init patch OK
[RC4] decrypt patch OK

mimikatz #
```

```
Microsoft Windows [version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Tous droits réservés.

C:\Users\Administrateur>net use z: \\dc.labokiwi.local\c$ waza1234/ /user:Admini
strateur@LABOKIWI.LOCAL
La commande s'est terminée correctement.


C:\Users\Administrateur>net use * /delete /yes
Vous possédez les connexions à distance suivantes :

    Z:              \\dc.labokiwi.local\c$
La poursuite de cette opération va rompre les connexions.

La commande s'est terminée correctement.


C:\Users\Administrateur>net use z: \\dc.labokiwi.local\c$ mimikatz /user:Adminis
trateur@LABOKIWI.LOCAL
La commande s'est terminée correctement.


C:\Users\Administrateur>_
```

Figure 3 Mimikatz support for Skeleton Key

Kerberos Pre-Authentication

In the Kerberos protocol, the KDC requires a pre-authentication stage to take place, in order to validate the user identity before sending any encrypted data.

On passwords-based authentication, the pre-authentication stage is realized with an encrypted time-stamp. The user encrypts the current time-stamp with its long-term key as a proof of password knowledge.

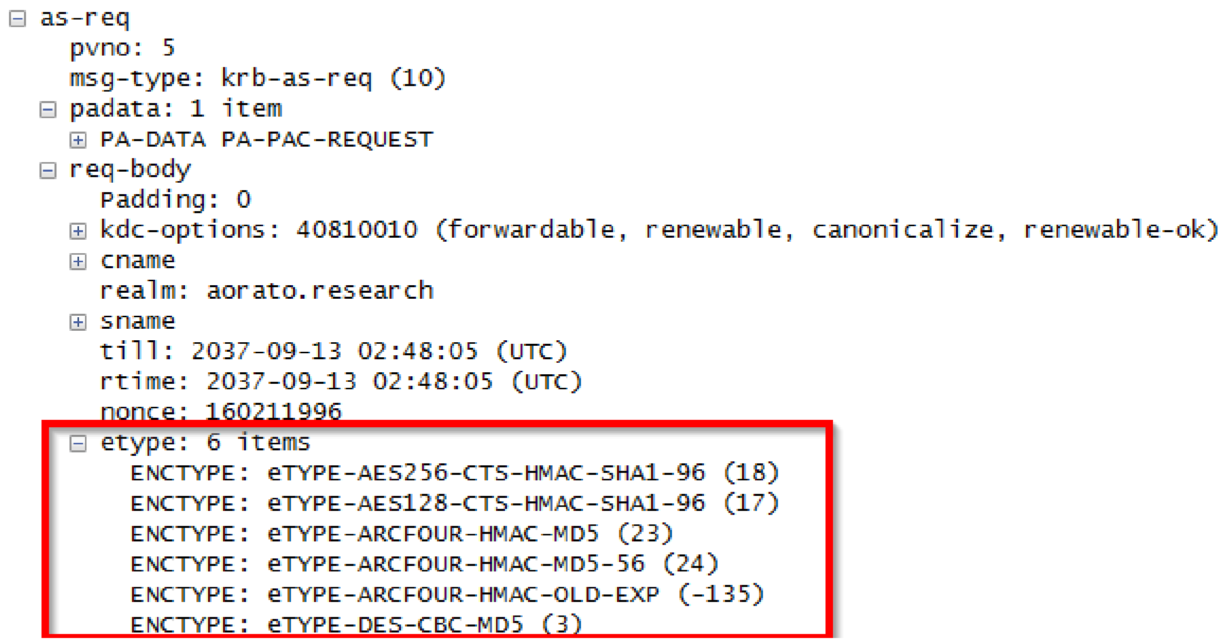
To support this process, a handshake phase must be supported by the protocol:

[6] https://github.com/gentilkiwi/mimikatz/releases/tag/2.0.0-alpha-20150117

- To determine a common encryption algorithm: Since the Kerberos protocol supports multiple encryption algorithms, a handshake phase is needed to determine the encryption types (ETypes) 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 different users having identical passwords will not have the same encryption key. The salt is determined per the encryption type. It should be noted that some encryption algorithms, such as the RC4-HMAC, does not support salt at all.

The Kerberos handshake phase details:

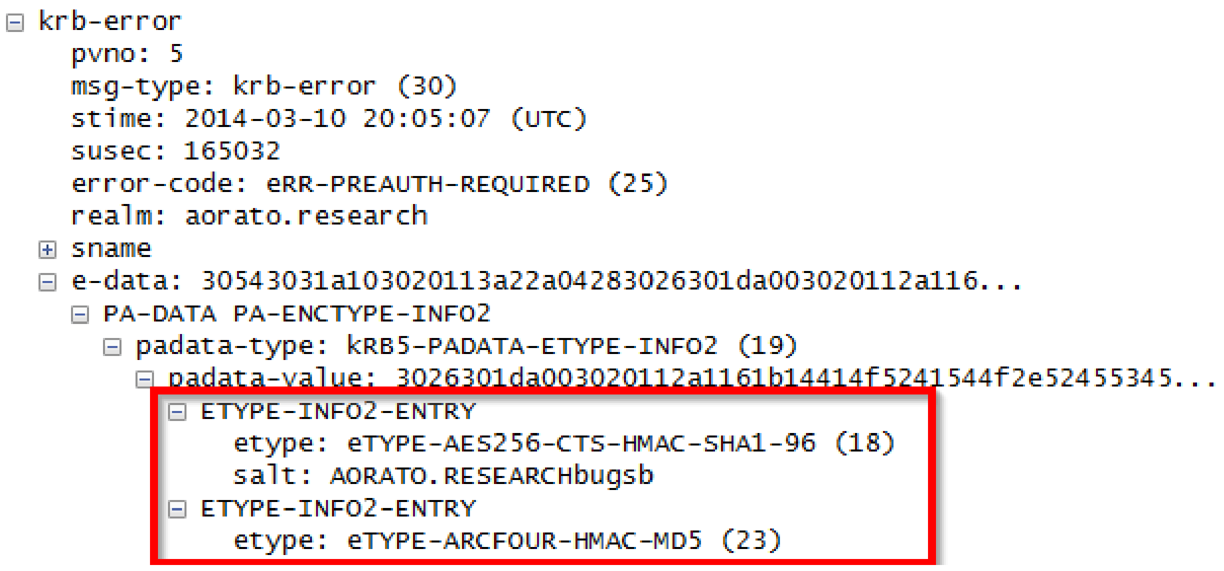
1. The client first sends an AS-REQ message detailing the encryption types it supports



```
⊟ 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 4 Initial AS-REQ with the client's supported ETypes

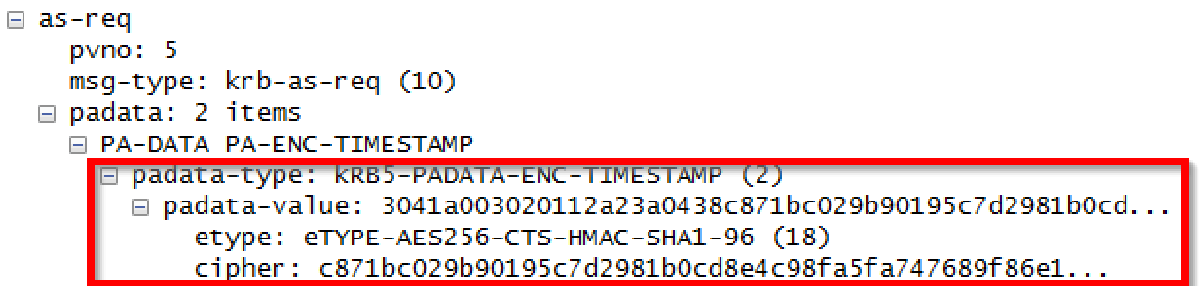
2. The DC responds with a Kerberos error message, detailing (within the PA-ETYPE-INFO2 field) all the encryption types which corresponds to the user keys stored within the DC database and were also present in client's AS-REQ message.



```
⊟ 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 5 DC's PA-ETYPE-INFO2

3. The client selects one of the DC returned encrypted types, encrypts the timestamp with it and sends it as pa-enc-timestamp in AS-REQ



```
⊟ 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 6 Timestamp encrypted in the mutually agreed EType

Network-Based Detection of the Skeleton Key Malware

Attackers can remove the Skeleton Key malware traces from the system hard drive after a successful infection, while leaving the compromised authentication process in place. This can pose a challenge for anti-malware engines to detect the compromise. However, it is still possible to detect the presence of Skeleton Key malware using "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 or equal to 2008[7]).

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

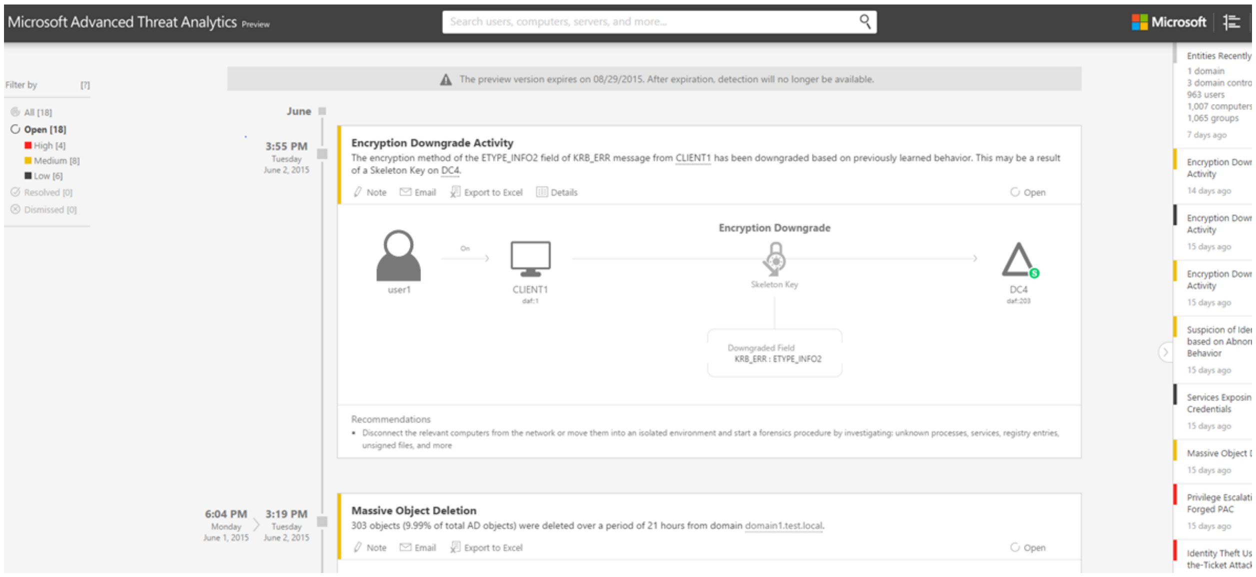
[7] https://technet.microsoft.com/en-us/library/understanding-active-directory-functional-levels%28v=ws.10%29.aspx

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. Inspect the client advertised E-Types from the user's AS-REQ and check for AES support.
2. Inspect the DC advertised E-Types 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 DC.

It should be noted that no access to either the client's or the DC's secrets (passwords or encryption keys) is required in order to implement this algorithm



The preview version expires on 08/29/2015. After expiration, detection will no longer be available.

Encryption Downgrade Activity
The encryption method of the ETYPE_INFO2 field of KRB_ERR message from CLIENT1 has been downgraded based on previously learned behavior. This may be a result of a Skeleton Key on DC4.

Recommendations
- Disconnect the relevant computers from the network or move them into an isolated environment and start a forensics procedure by investigating: unknown processes, services, registry entries, unsigned files, and more

Massive Object Deletion
303 objects (9.99% of total AD objects) were deleted over a period of 21 hours from domain domain1.test.local.

Figure 7 Network-based detection of the Skeleton key malware

Active downgrade detection

Defenders can take an active approach to detection by initiating an authentication request as a user using the AES encryption and observe a downgrade to RC4 by the DC. We had shared a publicly available standalone detection script[8] which is capable of detecting such user-encryption downgrade on DC.

The script works as follows:

1. Retrieves the Domain Functional Level (DFL) of current domain, in order to verify its support for AES (DFL >= 2008).
2. Finds an AES supporting account (msds-supportedencryptiontypes[9] >= 8).
3. Sends a Kerberos AS-REQ to all DCs with only the AES EType supported.

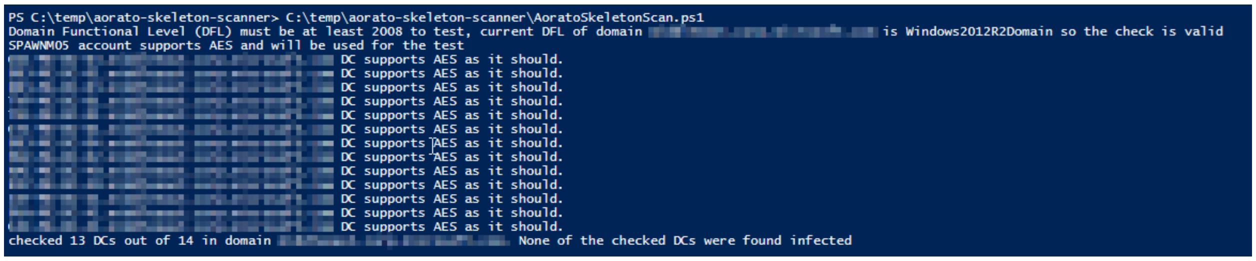
[8] https://gallery.technet.microsoft.com/Aorato-Skeleton-Key-24e46b73/

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

4. If AS-REQ fails due to “AES encryption not supported” error, then there’s a good chance that the DC is infected.

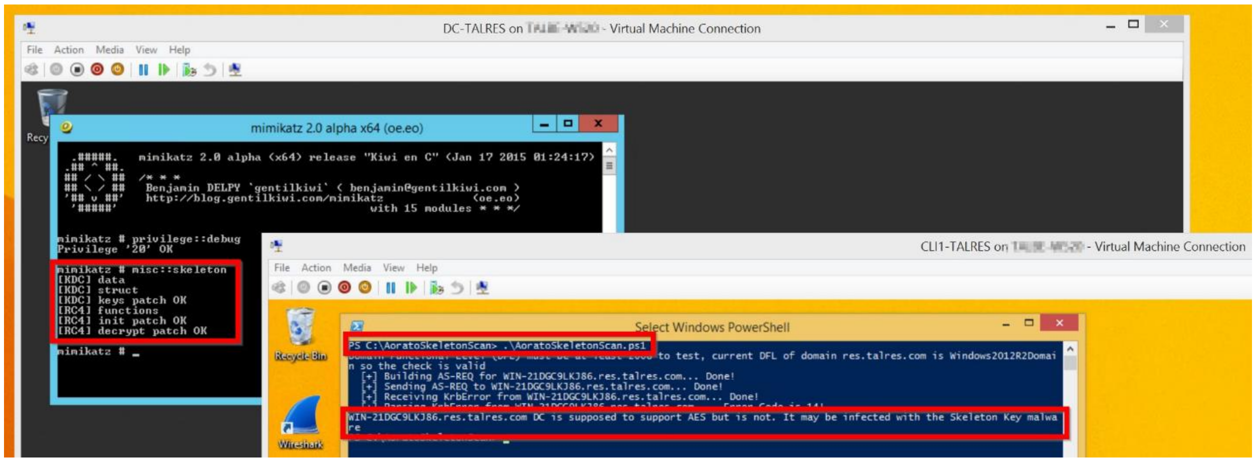
Once more, it should be noted that no access to either the client’s or the DC’s secrets (passwords or encryption keys) is required in order to implement this active detection algorithm.

.



```
PS C:\temp\aorato-skeleton-scanner> C:\temp\aorato-skeleton-scanner\AoratoSkeletonScan.ps1
Domain Functional Level (DFL) must be at least 2008 to test, current DFL of domain [redacted] is Windows2012R2Domain so the check is valid
SPAWNM05 account supports AES and will be used for the test
[redacted] DC supports AES as it should.
[redacted] DC supports AES as it should.
[redacted] DC supports AES as it should.
[redacted] DC supports AES as it should.
[redacted] DC supports AES as it should.
[redacted] DC supports AES as it should.
[redacted] DC supports AES as it should.
[redacted] DC supports AES as it should.
[redacted] DC supports AES as it should.
[redacted] DC supports AES as it should.
[redacted] DC supports AES as it should.
[redacted] DC supports AES as it should.
[redacted] DC supports AES as it should.
checked 13 DCs out of 14 in domain [redacted] None of the checked DCs were found infected
```

Figure 8 Execution of the script on all DCs of a non-infected domain



DC-TALRES on [redacted] - Virtual Machine Connection
File Action Media View Help
mimikatz 2.0 alpha x64 (oe.eo)
CLI1-TALRES on [redacted] - Virtual Machine Connection
Select Windows PowerShell
Recycle Bin
Wireshark

```
  .#####.   mimikatz 2.0 alpha (x64) release "Kiwi en C" (Jan 17 2015 01:24:17)
 .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                    with 15 modules * * */

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # misc::skeleton
[KDC] data
[KDC] struct
[KDC] keys patch OK
[RC4] functions
[RC4] init patch OK
[RC4] decrypt patch OK

mimikatz # _
```

```
PS C:\AoratoSkeletonScan> .\AoratoSkeletonScan.ps1
Domain Functional Level (DFL) must be at least 2008 to test, current DFL of domain res.talres.com is Windows2012R2Domai
n so the check is valid
  [+] Building AS-REQ for WIN-21DGC9LKJ86.res.talres.com... Done!
  [+] Sending AS-REQ to WIN-21DGC9LKJ86.res.talres.com... Done!
  [+] Receiving KrbError from WIN-21DGC9LKJ86.res.talres.com... Done!
  [+] Parsing KrbError from WIN-21DGC9LKJ86.res.talres.com... Error Code is 14!
WIN-21DGC9LKJ86.res.talres.com DC is supposed to support AES but is not. It may be infected with the Skeleton Key malwa
re
PS C:\AoratoSkeletonScan> _
```

Figure 9 Execution of the script on a Mimikatz Skeleton Key infected DC

Forged Ticket: The Golden Ticket Attack

Golden tickets are Kerberos TGTs forged by the attackers. The attacker can control every aspect of the forged ticket including the Ticket’s user identity, permissions and ticket life time.

Attackers typically set Golden Tickets to have an unusually long lifetime, which allows the possessing entity to keep using them for a long period without renewal. In addition to the lifetime, other important attributes of the ticket are typically forged to achieve other nefarious goals, such as assigning very high permissions, impersonating other users and even using non-existing user names.

Attack Details

In this Golden Ticket attack, the attacker:

1. Gains access to the Active Directory server itself
2. Steals the keys of the KRBTGT account from DC’s memory or hard drive.

3. Forges TGTs, using the KRBTGT key. Note that the forged ticket can contain arbitrary information including excessive privileges and ticket end time, hence the "golden" in its name.
4. Copies ("pass") the forged ticket to a remote machine to impersonate the victim on that machine.
5. Connects to various enterprise services on behalf of the victim from the remote machine.

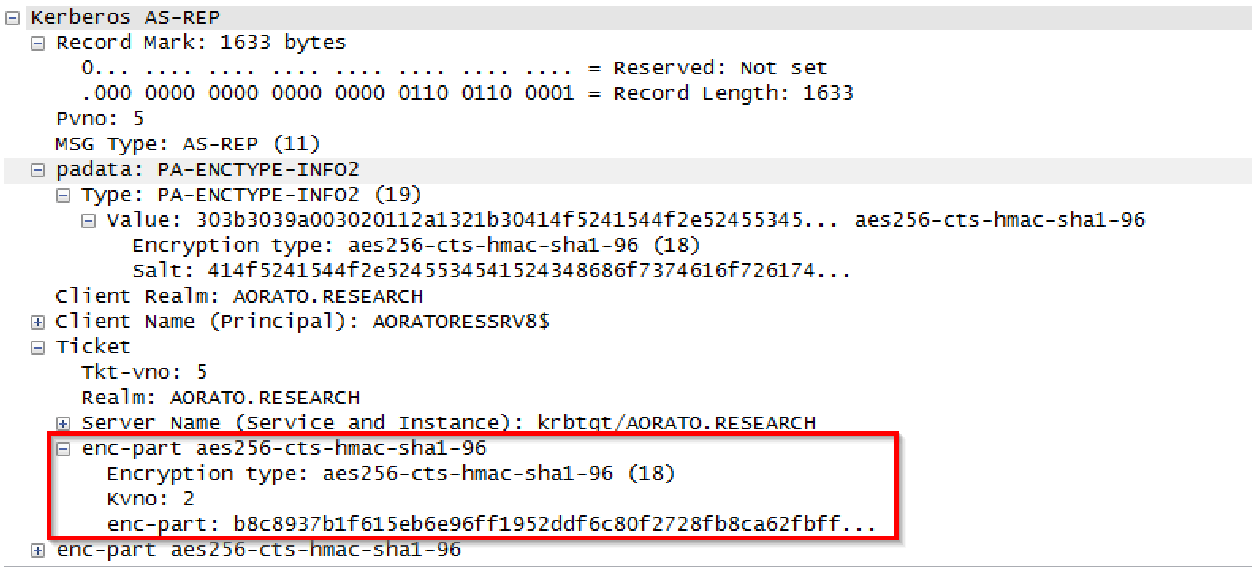
Technically, the attacker achieves this ability by providing the forged TGT on the remote machine as the authentication data for the authorization process on the TGS_REQ message.

Network Based Detection of the Golden Ticket Attack

A passive Security Device (SD), monitoring Kerberos traffic is able to recouple the specific requests together with their original authentication transaction and thus detect such decoupling attacks against the Authentication and Authorization system.

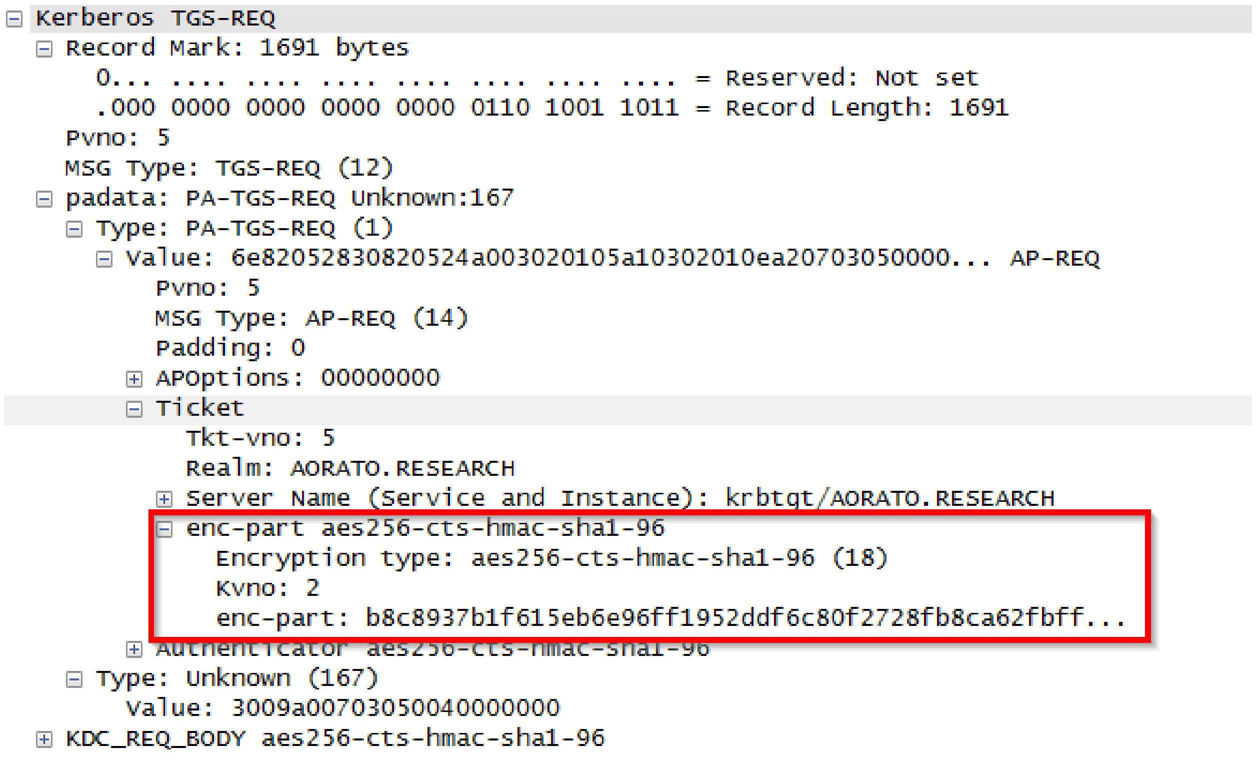
The TGT is supplied to the client from the DC with the AS-REP message and quoted by the client on the subsequent access requests in the TGS-REQ.
The key to recoupling is the fact that TGT is quoted verbatim. Therefore, the SD is able to associate the relevant authentication information as expressed in the authentication transaction (the AS-REQ, AS-REP message exchange) with the access transaction (the TGS-REQ, TGS_REP message exchange). Note that the fact that TGT itself is encrypted is irrelevant for recoupling purposes as the SD does not need to know the contents of the TGT, just to be able to identify that it’s the same across different transactions.

The figures below shows a typical flow of a single AS_REP authentication message, followed by a series of TGS-REQ access messages, all have the same TGT value, denoted by the "enc_part" column.



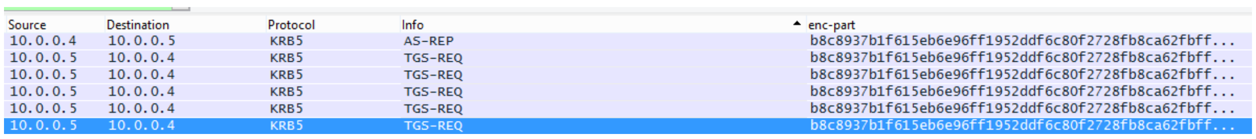
```
⊟ Kerberos AS-REP
  ⊟ Record Mark: 1633 bytes
      0... .... .... .... .... .... .... .... = Reserved: Not set
      .000 0000 0000 0000 0000 0110 0110 0001 = Record Length: 1633
    Pvno: 5
    MSG Type: AS-REP (11)
  ⊟ padata: PA-ENCTYPE-INFO2
    ⊟ Type: PA-ENCTYPE-INFO2 (19)
      ⊟ Value: 303b3039a003020112a1321b30414f5241544f2e52455345... aes256-cts-hmac-sha1-96
          Encryption type: aes256-cts-hmac-sha1-96 (18)
          Salt: 414f5241544f2e5245534541524348686f7374616f726174...
    Client Realm: AORATO.RESEARCH
  ⊞ Client Name (Principal): AORATORESSRV8$
  ⊟ Ticket
      Tkt-vno: 5
      Realm: AORATO.RESEARCH
    ⊞ Server Name (Service and Instance): krbtgt/AORATO.RESEARCH
    ⊟ enc-part aes256-cts-hmac-sha1-96
        Encryption type: aes256-cts-hmac-sha1-96 (18)
        Kvno: 2
        enc-part: b8c8937b1f615eb6e96ff1952ddf6c80f2728fb8ca62fbff...
  ⊞ enc-part aes256-cts-hmac-sha1-96
```

Figure 10 Network capture of Kerberos AS_REP, "enc-part" is TGT



```
⊟ Kerberos TGS-REQ
  ⊟ Record Mark: 1691 bytes
      0... .... .... .... .... .... .... .... = Reserved: Not set
      .000 0000 0000 0000 0000 0110 1001 1011 = Record Length: 1691
    Pvno: 5
    MSG Type: TGS-REQ (12)
  ⊟ padata: PA-TGS-REQ Unknown:167
    ⊟ Type: PA-TGS-REQ (1)
      ⊟ Value: 6e82052830820524a003020105a10302010ea20703050000... AP-REQ
          Pvno: 5
          MSG Type: AP-REQ (14)
          Padding: 0
        ⊞ APOptions: 00000000
        ⊟ Ticket
            Tkt-vno: 5
            Realm: AORATO.RESEARCH
          ⊞ Server Name (Service and Instance): krbtgt/AORATO.RESEARCH
          ⊟ enc-part aes256-cts-hmac-sha1-96
              Encryption type: aes256-cts-hmac-sha1-96 (18)
              Kvno: 2
              enc-part: b8c8937b1f615eb6e96ff1952ddf6c80f2728fb8ca62fbff...
        ⊞ Authenticator aes256-cts-hmac-sha1-96
    ⊟ Type: Unknown (167)
        Value: 3009a00703050040000000
  ⊞ KDC_REQ_BODY aes256-cts-hmac-sha1-96
```

Figure 11 Network capture of Kerberos TGE_REQ, "enc-part" contains the quoted TGT

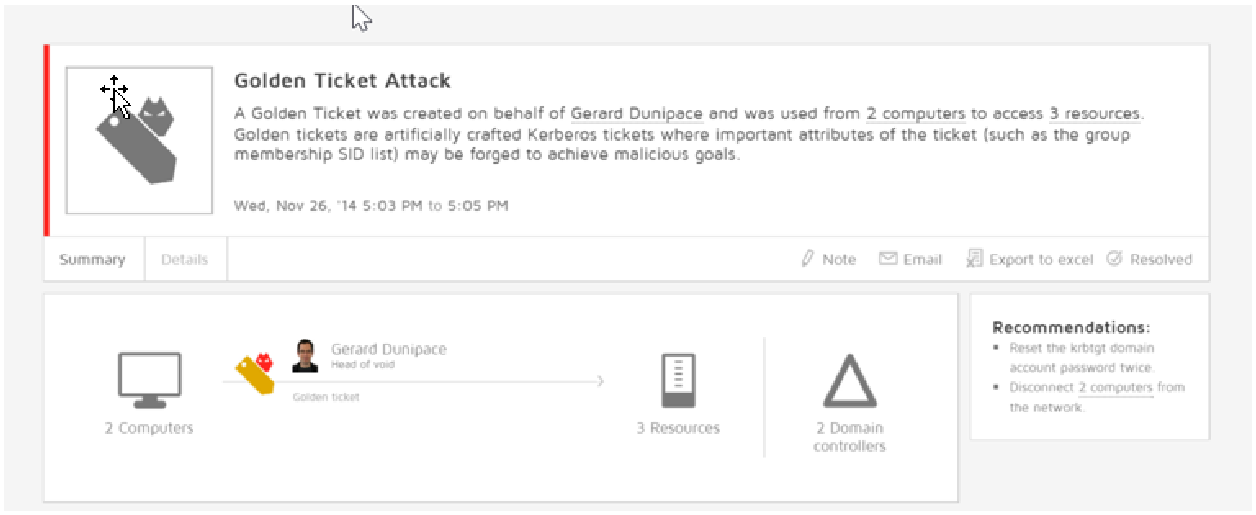


Source	Destination	Protocol	Info	enc-part
10.0.0.4	10.0.0.5	KRB5	AS-REP	b8c8937b1f615eb6e96ff1952ddf6c80f2728fb8ca62fbff...
10.0.0.5	10.0.0.4	KRB5	TGS-REQ	b8c8937b1f615eb6e96ff1952ddf6c80f2728fb8ca62fbff...
10.0.0.5	10.0.0.4	KRB5	TGS-REQ	b8c8937b1f615eb6e96ff1952ddf6c80f2728fb8ca62fbff...
10.0.0.5	10.0.0.4	KRB5	TGS-REQ	b8c8937b1f615eb6e96ff1952ddf6c80f2728fb8ca62fbff...
10.0.0.5	10.0.0.4	KRB5	TGS-REQ	b8c8937b1f615eb6e96ff1952ddf6c80f2728fb8ca62fbff...
10.0.0.5	10.0.0.4	KRB5	TGS-REQ	b8c8937b1f615eb6e96ff1952ddf6c80f2728fb8ca62fbff...

Figure 12 Network capture of Kerberos traffic, "enc-part" contains the TGT, "info" is Kerberos message name

When the attacker uses the forged TGT in a TGS-REQ message, the SD tries to retrieve the original authentication transaction that created the TGT, but fails as there is none. The SD flags this transaction as a "Golden Ticket" attack.

Once more, it should be noted that no access to either the client's or the DC's secrets (passwords or encryption keys) is required in order to implement this detection algorithm.



Golden Ticket Attack

A Golden Ticket was created on behalf of Gerard Dunipace and was used from 2 computers to access 3 resources. Golden tickets are artificially crafted Kerberos tickets where important attributes of the ticket (such as the group membership SID list) may be forged to achieve malicious goals.

Wed, Nov 26, '14 5:03 PM to 5:05 PM

Recommendations:

- Reset the krbtgt domain account password twice.
- Disconnect 2 computers from the network.

Figure 13 Network based detection of the Golden Ticket attack

Forged PAC: The MS14-068 Attack

On November 2014, a vulnerability in Windows PAC validation algorithm (CVE-2014-6324) was publicly disclosed. The vulnerability allows the attackers to forge the TGT's PAC and assign arbitrary privileges to the ticket. This vulnerability is now patched with the MS14-068 fix.

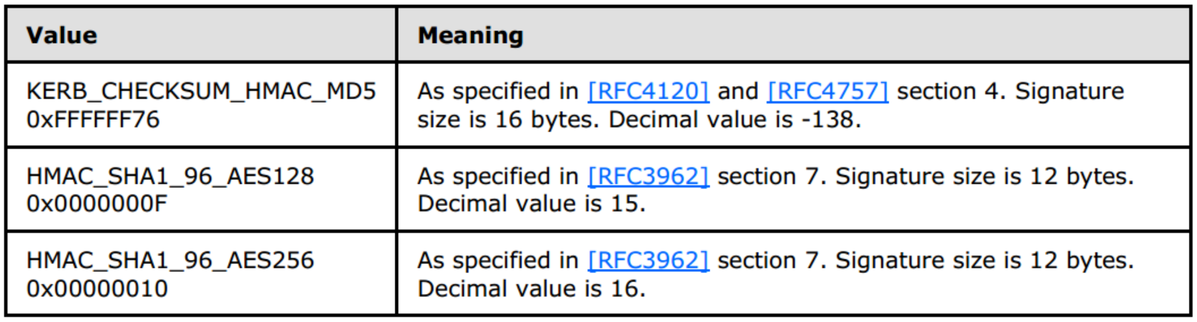
It was later published[10] that this vulnerability was used in the wild by the Duqu attackers to exploit Kaspersky labs' internal network.

The vulnerability: PAC can be signed without the key

The integrity of the PAC is guaranteed by the "Two PAC_SIGNATURE_DATA structures are appended to the PAC which stores the server and KDC signatures.[11]"

In the case of a TGT's PAC the server is the KDC, so both signatures are created by the KDC.

According to the specification, only three signing algorithms can be used to sign the PAC.



Value	Meaning
KERB_CHECKSUM_HMAC_MD5 0xFFFFFF76	As specified in [RFC4120] and [RFC4757] section 4. Signature size is 16 bytes. Decimal value is -138.
HMAC_SHA1_96_AES128 0x0000000F	As specified in [RFC3962] section 7. Signature size is 12 bytes. Decimal value is 15.
HMAC_SHA1_96_AES256 0x00000010	As specified in [RFC3962] section 7. Signature size is 12 bytes. Decimal value is 16.

Figure 14 The three available signature algorithms by [MS-PAC]

The root cause of the vulnerability is that the function which verifies the PAC signature erroneously accepts other signing algorithms besides the three mentioned above. In Windows 2008 (and before) the KDC code for signing algorithms accepts some key-less hashes such as MD5, that does not provide any security against masquerading PACs. In Windows 2012, KDC code does not allow any key-less hashing algorithms, however it allows some relatively weak signing algorithms, such as DES based signatures.

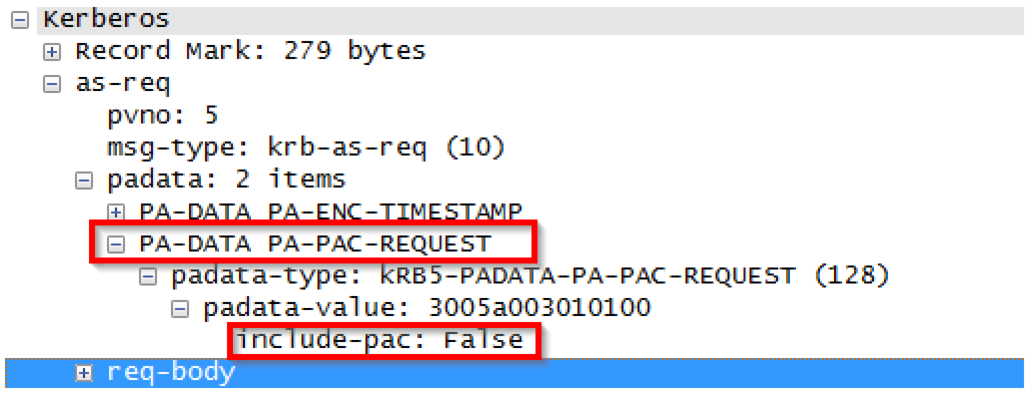
The Exploit

In order to exploit the vulnerability the attacker must include its forged PAC within a ticket. However, this mission cannot be accomplished in a trivial manner by merely substituting the existing PAC in an existing ticket with a forged one, as the tickets are encrypted.

Therefore the attacker acts as follows:

1. Sends an AS-REQ for a PAC-less ticket. The attacker does that by including the optional pre-authentication field of PA-PAC-REQUEST (128) and sets its value to false.

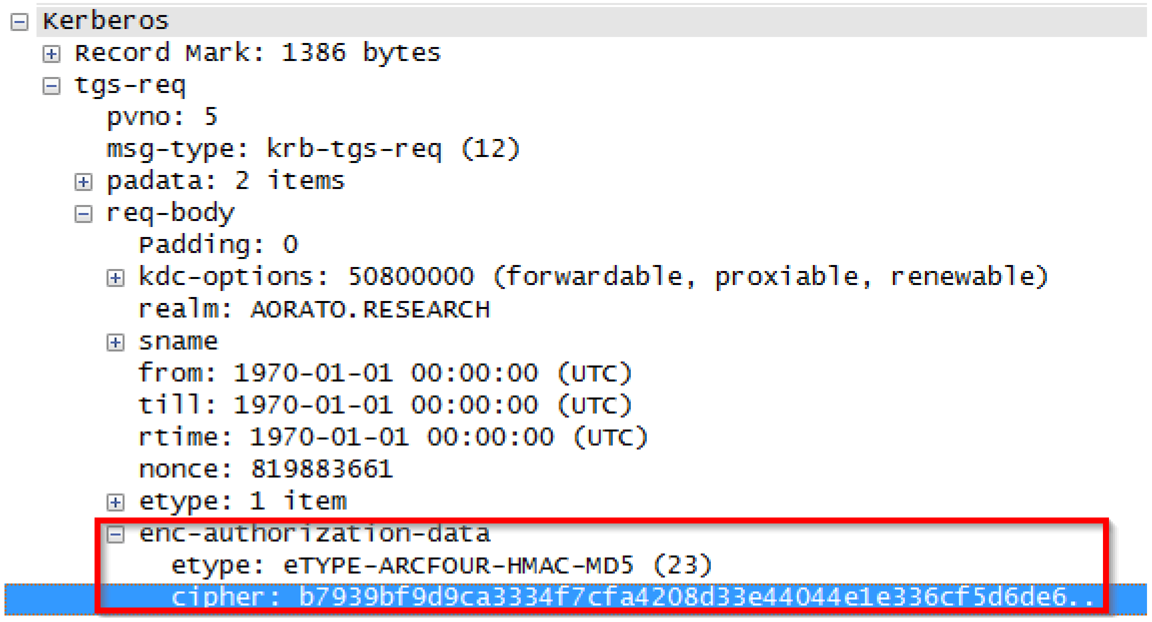
[10] https://securelist.com/files/2015/06/The_Mystery_of_Duqu_2_0_a_sophisticated_cyberespionage_actor_returns.pdf
[11] [MS-PAC]



```
⊟ Kerberos
  ⊞ Record Mark: 279 bytes
  ⊟ as-req
      pvno: 5
      msg-type: krb-as-req (10)
    ⊟ padata: 2 items
      ⊞ PA-DATA PA-ENC-TIMESTAMP
      ⊟ PA-DATA PA-PAC-REQUEST
        ⊟ padata-type: kRB5-PADATA-PA-PAC-REQUEST (128)
          ⊟ padata-value: 3005a003010100
              include-pac: False
    ⊞ req-body
```

Figure 15 PA_PAC_REQUEST set to FALSE

2. As a result a PAC-less ticket is sent to the client
3. The attacker sends a TGS-REQ with the forged PAC (encapsulated with relevant headers) and encrypted with the session key (or sub-session key, anyway it's encrypted), using the "enc-authorization-data" field in TGS-REQ.
   In order to have the PAC properly resigned, the attacker requests a ticket for the DC's KRBTGT service, normally used to renew expiring TGTs, to get a TGT in response.



```
⊟ Kerberos
  ⊞ Record Mark: 1386 bytes
  ⊟ tgs-req
      pvno: 5
      msg-type: krb-tgs-req (12)
    ⊞ padata: 2 items
    ⊟ req-body
        Padding: 0
      ⊞ kdc-options: 50800000 (forwardable, proxiable, renewable)
        realm: AORATO.RESEARCH
      ⊞ sname
        from: 1970-01-01 00:00:00 (UTC)
        till: 1970-01-01 00:00:00 (UTC)
        rtime: 1970-01-01 00:00:00 (UTC)
        nonce: 819883661
      ⊞ etype: 1 item
      ⊟ enc-authorization-data
          etype: eTYPE-ARCFOUR-HMAC-MD5 (23)
          cipher: b7939bf9d9ca3334f7cfa4208d33e44044e1e336cf5d6de6..
```

Figure 16 The enc-authorization-data Field

4. The KDC embeds the PAC in the returned ticket in the TGS-REP without ANY validation (it does not have to be a syntactically legal PAC, it work fine with a BLOB).
5. For a TGT obtained as above, the attacker requests a Service Ticket with a TGS-REQ that now carries the TGT which contains the forged PAC.
6. The KDC validates the PAC and resigns the PAC with a valid service and KDC signatures and embeds it in the returned Service Ticket in the TGS-REP.

Notes:

- Win2012 servers the attackers cannot sign the PAC with key-less hashes, however they are able to use the relatively weak DES cipher which can be bruteforced. Therefore the attacker needs to repeat stage 4 above until it succeeds in finding the right PAC.
- Requesting PAC-less ticket is mandatory for a successful exploit. If the ticket already contains a PAC, the DC disregards the PAC in the enc-authorization-data field.

Network Based Detection of MS14-068 Forged PAC

The detection is based on observing:

- An AS-REQ with a PA-PAC-REQUEST set to false.
- Adding authorization data in a subsequent TGS-REQ message
- The target service has its NA bit set to false, which means it requires PAC for authorization.

As noted above, having a PAC-less ticket is mandatory for a successful attack. When no PA-PAC-REQUEST is specified, the KDC returns a PAC. Therefore, attackers must explicitly send the PA-PAC-REQUEST and set it to FALSE.

Note, that if a client does not want a TGT or TGS to contain a PAC because the client or the target server does not support PAC based authorization, the client does not need to specify it explicitly as the DC already knows which account requires a PAC, with an AD attribute (NA flag[12]).

---

[12] [MS-KILE]:

"To support all functionality of KILE, the account database MUST be extended to support the following additional information for each principal:

AuthorizationDataNotRequired: A Boolean setting to control when to include a PAC in the service ticket. KILE implementations that use an Active Directory for the account database SHOULD use the userAccountControl attribute ([MS-ADTS] section 2.2.16) NA flag. The default is FALSE."

"3.3.5.3 PAC Generation

In either of the following two cases, a PAC [MS-PAC] MUST be generated and included in the response by the KDC when the client has requested that a PAC be included. The request to include a PAC is expressed through the use of a KERB-PA-PAC-REQUEST (section 2.2.2) PA-DATA type that is set to TRUE:

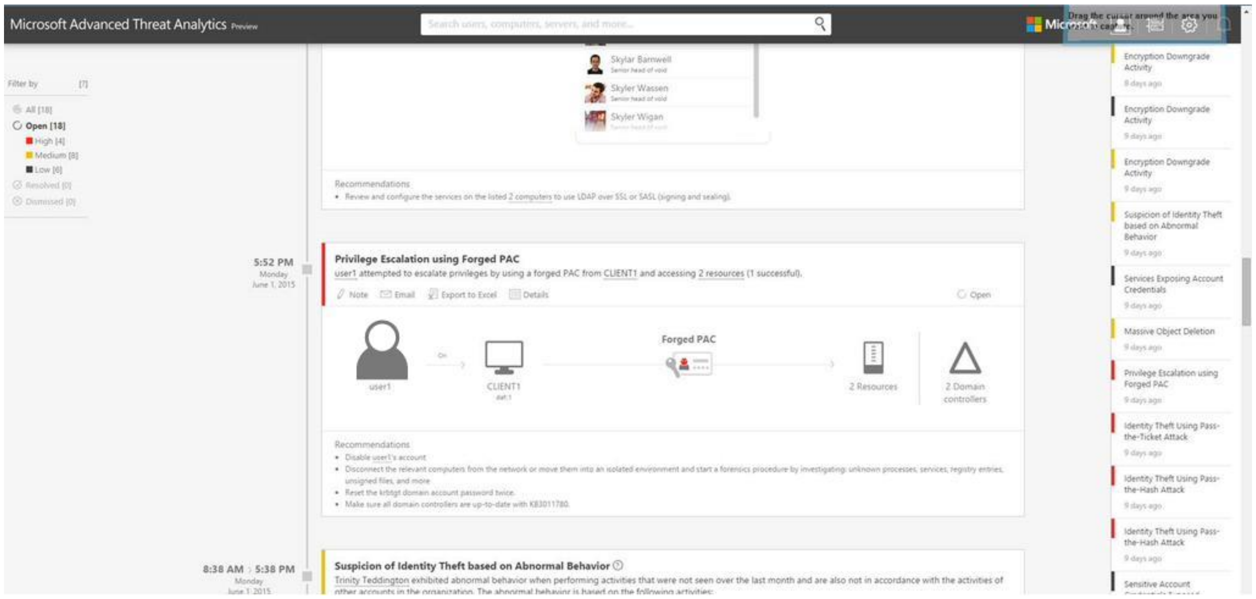
- During an Authentication Service (AS) request that has been validated with pre-authentication and for which the account has AuthorizationDataNotRequired set to FALSE.
- During a TGS request that results in a service ticket unless the NA bit is set in the UserAccountControl field in the KERB_VALIDATION_INFO structure ([MS-PAC] section 2.5).

Otherwise, the response will not contain a PAC."

[MS-ADTS]:

"NA (ADS_UF_NO_AUTH_DATA_REQUIRED, 0x02000000): Used by the Kerberos protocol.

Therefore, if the targeted service had its NA bit set to FALSE, which means they require PAC for authorization, we can safely assume the added authorization data is indeed a PAC.


Microsoft Advanced Threat Analytics Preview
Filter by [7]
All [18]
Open [18]
High [4]
Medium [8]
Low [6]
Resolved [0]
Dismissed [0]
Skylar Barnwell
Skyler Wassen
Skyler Wigan
Note
Email
Export to Excel
Details
Open
Forged PAC
user1
CLIENT1
2 Resources
2 Domain controllers
Encryption Downgrade Activity
8 days ago
Encryption Downgrade Activity
9 days ago
Encryption Downgrade Activity
9 days ago
Suspicion of Identity Theft based on Abnormal Behavior
9 days ago
Services Exposing Account Credentials
9 days ago
Massive Object Deletion
9 days ago
Privilege Escalation using Forged PAC
9 days ago
Identity Theft Using Pass-the-Ticket Attack
9 days ago
Identity Theft Using Pass-the-Hash Attack
9 days ago
Identity Theft Using Pass-the-Hash Attack
9 days ago
Sensitive Account

Recommendations
- Review and configure the services on the listed 2 computers to use LDAP over SSL or SASL (signing and sealing).

5:52 PM
Monday
June 1, 2015

Privilege Escalation using Forged PAC
user1 attempted to escalate privileges by using a forged PAC from CLIENT1 and accessing 2 resources (1 successful).

Recommendations
- Disable user1's account
- Disconnect the relevant computers from the network or move them into an isolated environment and start a forensics procedure by investigating: unknown processes, services, registry entries, unsigned files, and more
- Reset the krbtgt domain account password twice.
- Make sure all domain controllers are up-to-date with KB3011780.

8:38 AM - 5:38 PM
Monday
June 1, 2015

Suspicion of Identity Theft based on Abnormal Behavior
Trinity Teddington exhibited abnormal behavior when performing activities that were not seen over the last month and are also not in accordance with the activities of other accounts in the organization. The abnormal behavior is based on the following activities:

Figure 17 Network based detection of MS14-068 forged PAC

Once more, it should be noted that no access to either the client's or the DC's secrets (passwords or encryption keys) is required in order to implement this detection algorithm.

This bit indicates that when the Key Distribution Center (KDC) is issuing a service ticket for this account, the Privilege Attribute Certificate (PAC) MUST NOT be included. For more information, see [RFC4120]"

Forged PAC revisited: The Diamond PAC

Diamond PACs are attackers' crafted PAC, signed with a stolen KRBTGT key. The Diamond PAC can be used by attackers to assign high privileges to a user's access request, regardless of the user's original permissions.

In contrast to the Golden Ticket attack which also use a stolen KRBTGT key and can be used to provide high privileges to the attacker, the Diamond PAC attack does not craft a full Kerberos ticket, but uses the standard Kerberos flow (in the same manner of the MS14-068 exploit) to inject the crafted Diamond PAC.

By doing so, the Diamond PAC attack represents a more subtle version of the Golden Ticket attack and thus harder to detect. Detection rules designed to catch Golden Ticket attack based on the Kerberos traffic anomaly it creates of a user sending a valid TGS with no prior AS requests to obtain a TGT will not alert on the Diamond PAC attack.

Diamond PAC Generation and Injection

Similar to Golden Ticket, In order to successfully generate a Diamond PAC, several pieces of information must be provided:

- krbtgt account key: The most crucial component. The key is used to sign the PAC.
- The TGT's authentication time: Authentication time is a mandatory part of the PAC as it is used by Kerberos to identify the client and verify that the PAC corresponds to the ticket granted to that specific client. **Inconsistent authentication time between PAC and encapsulating ticket results in a failed attack – PAC validation failed!**
- The SID of the domain: must be provided, as it's a part of the PAC.
- A valid domain account credentials: A successful AS-REQ is the initial stage of the attack.

Once the Diamond PAC is created, it is injected into a valid Ticket in the same manner as described for the MS14-068 forged PAC exploit above. The difference is that the Diamond PAC is validly signed using the KRBTGT key.

Network Based Detection of the Diamond PAC Attack

Since the Diamond PAC attack is using a valid method to inject authorization data and signs the PAC with a valid key, it will be accepted even by servers or DCs patched for the MS14-068 exploit. Therefore, it seems that the only way to detect this attack is to follow the same logic suggested above to detect MS14-068 forged PAC attack:

1. An AS-REQ with a PA-PAC-REQUEST set to false.
2. Adding authorization data in a subsequent TGS message
3. The target service has its NA bit set to false, which means it requires PAC for authorization.