Taking Kerberos To The Next Level
Researcher @ Google Project Zero
Specialize in Windows
○ Local Privilege Escalation
○ RPC/COM Internals
○ Token manipulation
NtApiDotNet | D2J | OleViewDotNet

“Never met a logical vulnerability I didn't like.”
Nick Landers | @monoxgas

- Adversarial R&D @ NetSPI
- Also specialize in Windows
  - Offensive tooling suites
  - Payload architectures
  - Vulnerability research
- sRDI | Dark Side Ops

“Your Prod is our Dev.”
Assumptions

You understand the basics of Kerberos

You're (somewhat) familiar with existing remote attacks

You want to see some local privilege escalation (LPE)
Talking to Yourself can be good for you
Local Kerberos Authentication

**ABC.REALM**
- Local Security Authority
- **PAC**
- krbtgt/REALM
- TGT

**KDC.REALM**
- SPN: HOST/ABC
- KEY: abc$@REALM

**Kerberos Security Package**
- InitializeSecurityContext
- AcceptSecurityContext

**Client Code**

**Server Code**
Local Kerberos Authentication

**ABC.REALM**
- Local Security Authority
  - PAC
    - krbtgt/REALM
    - TGT
  - InitializeSecurityContext
- Client Code
- Server Code

**KDC.REALM**
- SPN: HOST/ABC
- KEY: abc$@REALM
- TGS-REQ
- Kerberos Security Package
- krbtgt/REALM
  - TGT

Diagram:
- Flow from Client Code to Kerberos Security Package
- Flow from Kerberos Security Package to TGS-REQ
- Flow from TGS-REQ to Server Code
Local Kerberos Authentication

**ABC.REALM**
- Local Security Authority
  - PAC
  - krbtgt/REALM
  - TGT
- InitializeSecurityContext
- Client Code
  - HOST/ABC
  - TGS

**KDC.REALM**
- SPN: HOST/ABC
- KEY: abc$@REALM
- TGS-REP
- Server Code
Local Kerberos Authentication

ABC.REALM

Local Security Authority

KDC.REALM

SPN: HOST/ABC

KEY: abc$@REALM

Kerberos Security Package

AcceptSecurityContext

Client Code

HOST/ABC

TGT

PAC

Server Code

HOST/ABC

TGS

PAC

AP-REQ
Local Kerberos Authentication

**ABC.REALM**
- Local Security Authority
- PAC: krbtgt/REALM
- PAC: TGT
- Client Code: HOST/ABC

**KDC.REALM**
- SPN: HOST/ABC
- KEY: abc$@REALM

**Kerberos Security Package**
- Key
- PAC

**Access Token**
- TGS
- Server Code

**Diagram Notes**
- ABC.REALM is connected to KDC.REALM through the Kerberos Security Package, which contains the key and PAC.
1. Logon with credentials to initialize key in LSA

Local Security Authority

LsaLogonUser

KEY: bob@REALM

u: REALM\bob
pw: Password!

User Session

Local Kerberos Silver Ticket
Local Kerberos Silver Ticket

1. Convert credentials to key
   - u: REALM\bob
   - pw: Password!

2. Use key to build silver ticket
   - KEY: bob@REALM
   - CIFS/Client TGS

3. Use key to build silver ticket
   - KEY: bob@REALM
Local Kerberos Silver Ticket

1. Build A-REQ and accept
2. AcceptSecurityContext
3. Parse PAC and get token
4. CIFS/Client TGS
5. Token (admin)
Demo Time
LSA Internals
and how to break them
PAC Signature Validation

**Client**

- **PAC**
  - KDC Signature
  - Server Signature
  - **Local Checksum**

- **Local Security Authority**
  - KEY: bob@REALM

- **Compute local checksum of PAC with service key**

**KDC**

- **KDC Package**
  - Kerberos Package
PAC Signature Validation

Client

- **PAC**
  - KDC Signature
  - Server Signature
  - Local Checksum

 2 Verify PAC server signature against local value

Local Security Authority

KDC

Kerberos Package
PAC Signature Validation

Client

PAC
- KDC Signature
- Server Signature
- Local Checksum

KDC

- KDC Signature
- Local Checksum

Local Security Authority

Local Checksum
KERB_VERIFY_PAC_REQUEST

3 Send the KDC the checksum and signature to validate

NETLOGON

Kerberos Package
- KDC Signature
- Local Checksum
PAC Signature Validation

Client

- **PAC**
  - KDC Signature
  - Server Signature
  - Local Checksum

KDC

- Verify signature with realm key and reply
- KEY: krbtgt@REALM
- KDC Signature
- Local Checksum

Local Security Authority

Kerberos Package
So how do Silver Tickets ever work?

(PAC validation isn't always enabled)

**Logon Session:**

```
SeTcbPrivilege || SYSTEM || LOCAL/NETWORK SERVICE
```

**Credentials Handle:**

```
cred->Flags & SECPKG_CRED_ATTR_PAC_BYPASS

(auto) AcquireCredentialsHandle w/
SECPKG_CRED_INBOUND && NT AUTHORITY\SERVICE &&
!KerbGlobalValidateKDCPACSignature

(manual) SetCredentialsAttributes w/SeTcbPrivilege
```

**ASC Context Flags:**

```
context->Flags & ASC_RET_USE_SESSION_KEY
```
So how do Silver Tickets ever work?

(PAC validation isn't always enabled)

Logon Session:

```
SetTcbPrivilege || SYSTEM || LOCAL/NETWORK SERVICE
```

Credentials Handle:

```
cred->Flags & SECPKG_CRED_ATTR_PAC_BYPASS

(auto) AcquireCredentialsHandle w/
SECPKG_CRED_INBOUND && NT AUTHORITY\SERVICE &&
!KerbGlobalValidateKDCPACSignature

(manual) SetCredentialsAttributes w/SetTcbPrivilege
```

ASC Context Flags:

```
context->Flags & ASC_RET_USE_SESSION_KEY
```
3.2.3. Receipt of KRB_AP_REQ Message

 [...] If the **USE-SESSION-KEY** flag is set in the ap-options field, it indicates to the server that user-to-user authentication is in use, and that the **ticket is encrypted in the session key from the server's TGT rather than in the server's secret key.**

See Section 3.7 for a more complete description of the effect of **user-to-user authentication** on all messages in the Kerberos protocol.

RFC 4120 - Kerberos V5
We now need the session key from our TGT to build the Silver Ticket.
Trying to get a TGT + Session Key

```
PS C:\> $ticket = Get-KerberosTicket krbtgt
PS C:\> $ticket.SessionKey

KeyEncryption : AES256_CTS_HMAC_SHA1_96
Principal      : krbtgt/DOM.LOC@DOM.LOC
NameType       : SRV_INST

PS C:\> $ticket.SessionKey.Key | Format-HexDump
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
No session key ?
```
Want to get a usable Kerberos TGT without admin rights/allowtgtsessionkey? It's easy with a delegation ticket! (enabled by default...)
No special requirement, just some love
> github.com/gentilkiwi/kek...
Thank you @elad_shamir (and @TheColonial) for evil ideas!

https://twitter.com/gentilkiwi/status/998219775485661184
Unconstrained Delegation TGT Extraction

1. Request AP-REQ for delegatable service

**User Session**

**Local Security Authority**

- **SPN**: CIFS/KDC.REALM
- **KEY**: bob@REALM

**Session Key**

- CIFS/KDC.REALM
- TGS [Delegate: OK]

**InitializeSecurityContext + ISC_REQ_DELEGATE**
Unconstrained Delegation TGT Extraction

Local Security Authority

CIFS/KDC.REALM TGS [Delegate: OK]
Session Key

krbtgt/REALM TGT
Session Key

CIFS/KDC.REALM TGS

AP-REQ

Authenticator

2 Authenticator is encrypted with session key and contains the delegation TGT
What can you add to the PAC?

PAC->LogonInfo

- Domain SID
- User ID
- Group IDs
- Extra SIDs
- Resource Domain SID
- Resource Group IDs

Any domain SID which is not local account domain.

Any domain RIDs
- 500 - REALM\Administrator
- 512 - REALM\Domain Administrator
- 1000+ - User and Computer Accounts

Most SIDs which are not the local account domain or NT AUTHORITY

Mandatory Integrity SID

Capability SIDs

Most SIDs which are not the local account domain or NT AUTHORITY
Demo Time
SeTokenCanImpersonate

- Token Level < Impersonate
- Process has Impersonate Privilege
- Process IL < Token IL
- Process User == Token User

Allowed

Restrict to Identification

Elevation Check

Session ID Check
SeTokenCanImpersonate

Token Level < Impersonate → Process has Impersonate Privilege → Process IL < Token IL → Process User == Token User

Elevation Check

Session ID Check

Can be controlled by the PAC

Allowed

Restrict to Identification

Doesn't apply to us

Ignored if user is not a UAC admin

???
if (Primary->SessionId != Impersonation->SessionId &&
    Impersonation->SessionId == 0) {
    return STATUS_PRIVILEGE_NOT_HELD;
}
Hunting for Session Update Primitives

NtSetInformationToken(..., TokenSessionId, ...)

- LsapAuApiDispatchLogonUser
- LsapCreateTokenEx
- LsaISetSupplementalTokenInfo
- LsapUpdateUserTokenSessionId
- LsapBuildAndCreateToken
- LsapSetSessionToken
- LsapApplyLoopbackSessionId
- LsapFilterElevatedTokenFull
Hunting for Session Update Primitives

NtSetInformationToken(..., TokenSessionId, ...)

- LsapAuApiDispatchLogonUser
- LsapCreateTokenEx
- LsaISetSupplementalTokenInfo
- LsapBuildAndCreateToken
- LsapSetSessionToken
- LsapApplyLoopbackSessionId
- LsapFilterElevatedTokenFull
- LsapUpdateUserTokenSessionId

Requires TCB

Not useful
Hunting for Session Update Primitives

NtSetInformationToken(..., TokenSessionId, ...)
Hunting for Session Update Primitives

NtSetInformationToken(..., TokenSessionId, ...)

- LsapBuildAndCreateToken
- LsapSetSessionToken
- LsapApplyLoopbackSessionId
- LsapUpdateUserTokenSessionId
- LsaISetSupplementalTokenInfo
- LsapFilterElevatedTokenFull
- LsapAuApiDispatchLogonUser
- LsapCreateTokenEx

Accessible via AcceptSecurityContext
LSA Loopback Library

1. Call ISC and add entry to loopback tracking table with caller's session id

User Session X

Local Security Authority

Kerberos Security Package

InitializeSecurityContext

BeginTracking

<table>
<thead>
<tr>
<th>Hash Value</th>
<th>Session ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>X</td>
</tr>
</tbody>
</table>
User Session X

Local Security Authority

Kerberos Security Package

InitializeSecurityContext

UpdateTracking

Hash Value | Session ID
FEEDACDC | X

AP-REQ

Update AES-CMAC hash with security buffer contents
Accept AP-REQ and add AES-CMAC entry

### LSA Loopback Library

#### Local Security Authority

- **Kerberos Security Package**
- **BeginTracking**
- **AcceptSecurityContext**

#### User Session

- **AP-REQ**

#### Table

<table>
<thead>
<tr>
<th>Hash Value</th>
<th>Session ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEEDACDC</td>
<td>X</td>
</tr>
<tr>
<td>FEEDACDC</td>
<td>0</td>
</tr>
</tbody>
</table>

The diagram illustrates the process flow and data tracking within the LSA Loopback Library.
Create token and lookup final hash. If a match is found, move the token to the client session.
1. Loopback Library will hash all security buffers between LSA and clients. If hashes match when a token is being built, the token will be moved to the client session.

2. We need to start using InitializeSecurityContext with our silver tickets to get the hash entry initialized.

3. We need to modify the PAC inside the AP-REQ, but if we touch the buffers the hash lookup will break. (or will it?)
PSecBufferDesc pInput = ...;
for (ULONG i = 0; i < pInput->cBuffers; ++i) {
    PSecBuffer pBuffer = &pInput->pBuffers[i];

    if (pBuffer.BufferType == SECBUFFER_TOKEN) {
        BCryptHashData(hHash, pDirectionGuid, cbDirectionGuid);
        BCryptHashData(hHash, pBuffer->pvBuffer, pBuffer->cbBuffer);
    }
}
## Security Buffer Types

<table>
<thead>
<tr>
<th>Buffer Type</th>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECBUFFEREMPTY</td>
<td>Undefined, replaced by the security package function</td>
<td>0x00000000</td>
</tr>
<tr>
<td>SECBUFFER_TOKEN</td>
<td>Security token</td>
<td>0x00000002</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECBUFFER_READONLY</td>
<td>Buffer is read-only, no checksum</td>
<td>0x80000000</td>
</tr>
<tr>
<td>SECBUFFER_READONLY_WITH_CHECKSUM</td>
<td>Buffer is read-only, and checksummed</td>
<td>0x10000000</td>
</tr>
</tbody>
</table>

The buffer types can be combined using a **bitwise-OR** operation with the READONLY buffer types.
Type Confusion in AcceptSecurityContext

- **Security Buffer Descriptor**
  - `SECBUFFER_TOKEN` | `SECBUFFER_READONLY`
  - **Modified AP-REQ with Silver Ticket**
  - `SECBUFFER_TOKEN`
  - **Original AP-REQ from InitializeSecurityContext**

- **Kerberos Security Package**
  - Ignored by Loopback library.
  - Used by Kerberos.

- **Loopback Library**
  - Hashed by LoopbackLibrary.
  - Ignored by Kerberos.
Demo Time
Fixed in Windows 11?

**Windows 10:**

```c
if (pBuffer.BufferType == SECBUFFER_TOKEN) {
    BCryptHashData(hHash, pDirectionGuid, cbDirectionGuid);
    BCryptHashData(hHash, pBuffer->pvBuffer, pBuffer->cbBuffer);
}
```

**Windows 11:**

```c
if ((pBuffer.BufferType & ~SECBUFFER_ATTRMASK) == SECBUFFER_TOKEN) {
    BCryptHashData(hHash, pDirectionGuid, cbDirectionGuid);
    BCryptHashData(hHash, pBuffer->pvBuffer, pBuffer->cbBuffer);
}
```

Masking the upper byte out
LSA Loopback Library

Danger Zone

Local Security Authority

Kerberos Security Package

(Initialize/Accept)SecurityContext

Our Code

User Session

KDC.REALM

Man in the middle is on the table?

No Fly Zone!
Modifying on the Wire

Local Security Authority
- bob@REALM
- PAC w/admin
- CIFS/ABC
- TGS U2U

KDC
- bob@REALM
- PAC
- CIFS/ABC
- TGS U2U

TGS-REP (evil)
TGS-REP
Session Key
What about Credential Guard?

**Kerberos Considerations**

When you enable Windows Defender Credential Guard, you can no longer use Kerberos unconstrained delegation or DES encryption. Unconstrained delegation could allow attackers to extract Kerberos keys from the isolated LSA process. Use constrained or resource-based Kerberos delegation instead.

LSA Loopback Library
BYOKDC

Local Security Authority
Kerberos Security Package
(Initialize/Accept)SecurityContext
Our Code
User Session

KDC.FAKE

Bring Your Own KDC?

No Fly Zone!
KDC Pinning

```c
struct SECPKG_CALL_PACKAGE_PIN_DC_REQUEST {
    ULONG MessageType;
    ULONG Flags;
    UNICODE_STRING DomainName;
    UNICODE_STRING DcName;
    ULONG DcFlags;
};
```

`MessageType` can be `SecPkgCallPackagePinDcMessage` or `KerbPinKdcMessage`
Pin our fake KDC to localhost.
TGS-REP Local KDC

Client

Local Security Authority

Kerberos Security Package

Realm | Host | PID | TID
------|------|-----|-----
FAKE  | localhost | X   | Y   

KerbMakeSocketCall

LsaLogonUser

u: FAKE\bob
pw: WooHoo!

fake credentials

User Session X

Custom KDC

CIFS/CLIENT

krbtgt@FAKE

bob@REALM
PAC w/admin

TGS U2U

Issue our own tickets with arbitrary PAC data
(despite being different domains)
Demo Time
“Security Boundaries” and where they aren't
Service Account S4U2Self

Client

Local Security Authority

LsaLogonUser(S4U, ...)

KEY: svc@REALM
CNAME: Admin

UPN: Admin
Realm: REALM

Access Token

User: REALM\Admin
Level: Identification

TGS-REP + PA-FOR-USER

We have Selimpersonate but the token is locked to Identification

KDC.REALM
S4U TCB Privilege Check

KerbCreateTokenFromLogonTicket(...) {
    if (MessageType == KerbTicketLogon || MessageType == KerbTicketUnlockLogon ||
        MessageType == KerbS4ULogon || ...
    ){
        if (!ClientInfo.HasTcbPrivilege)
            PrimaryCredentials->Flags |= PRIMARY_CRED_LOGON_NO_TCB;
    }
}

LsapAuApiDispatchLogonUser(...) {
    BOOL UseIdentify = PrimaryCredentials.Flags & PRIMARY_CRED_LOGON_NO_TCB;
    LsapCreateV3Token(...
        (UseIdentify ? TokenImpersonation : TokenPrimary),
        (UseIdentify ? SecurityIdentification : SecurityImpersonation),
        &Token
    );
}
Service Account S4U2Self

Client

Local Security Authority

krbtgt/REALM

TGT

KRB_CRED
krbtgt/REALM

TGT Extraction

Service Account Session

KDC.REALM
Service Account S4U2Self

Client

Local Security Authority

- Key: svc@REALM
- CNAME: Admin

Service Account Session

- KRBCRED
- krbtgt/REALM

TGS-REP + PA-FOR-USER

KDC.REALM
Service Account S4U2Self

Client

Local Security Authority

Access Token

User: REALM\Admin
Level: Impersonation

Service Account Session

KEY: svc@REALM
CNAME: Admin

krbtgt/REALM TGT

KEY: svc@REALM
CNAME: Admin

KRIB_CRED
krbtgt/REALM

KDC.REALM
What about UAC?

2.2.5. LSAP_TOKEN_INFO_INTEGRITY

The LSAP_TOKEN_INFO_INTEGRITY structure specifies the integrity level information for the client.<7>

```c
typedef struct _LSAP_TOKEN_INFO_INTEGRITY {
    unsigned long Flags;
    unsigned long TokenIL;
    unsigned char MachineID[32];
} [MS-KILE]: Kerberos Protocol Extensions
```
Authorization Data Entries

Ticket’s AD entry taken in preference to Authenticator’s Token Filtering Logic

AP-REQ

**Service Ticket**
- bob@REALM
- PAC

**Authenticator**
- Authorization Data
- [...]

**Authorization Data**

KERB-AD-RESTRICTION-ENTRY

{ 
  Flags = RestrictedToken;
  TokenIL = Medium;
  MachineID = {FEED-ACDC};
}
LSA Token Filtering

via LsaISetSupplementalTokenInfo()

User is a Local Account &&
Local Account Token Filtering is enabled &&
Machine is not a domain controller

Network Auth Token Filtering is enabled

info.Machineld == LsapGlobalMachineID

info.Flags & LimitedToken

Filter Token

No Filtering
Kerberos UAC Bypass

1. Request service ticket and session key from ticket cache
Kerberos UAC Bypass

1. Manually renew the service ticket without any authorization data.
Kerberos UAC Bypass

Client

Service Session

SCM RPC Service

Local Security Authority

Kerberos Security Package

Full Token

KDC.REALM

Service Account Session

Pass clean ticket to RPC server

RPC/CLIENT TGS

Auth Data

Session Key
Demo Time
Wrap Up Time
**CVE-2022-35756**

**Logon Session:**
- `SeTcbPrivilege || SYSTEM || LOCAL/NETWORK SERVICE`

**Credentials Handle:**
- **(auto) AcquireCredentialsHandle w/**
  - `SECPKG_CRED_INBOUND && NT AUTHORITY\SERVICE && !KerbGlobalValidateKDCPACSignature`

- **(manual) SetCredentialsAttributes w/SeTcbPrivilege**

**ASC Context Flags:**
- `context->Flags & ASC_RET_USE_SESSION_KEY`
Mitigation Thoughts

- **Enable KerbGlobalValidateKDCPACSignature**
  - Prevent “NT AUTHORITY\SERVICE” SID from bypassing PAC verification
  - Doesn’t prevent “LOCAL/NETWORK SERVICE” or “SYSTEM” though

- **Force Kerberos Armoring / FAST**
  - Makes it harder to tamper with network traffic

- **Enable Credential Guard**
  - Block trivial access to TGT session keys

- **Build Kerberos firewall rules**
  - Block access to KDCs outside an approved list
Special privileges assigned to new logon.

Subject:

Security ID: REALM\bob
Account Name: bob
Account Domain: REALM
Logon ID: 0x4b842

Privileges:

SeSecurityPrivilege
SeTakeOwnershipPrivilege
SeLoadDriverPrivilege
...

Event 4672
Limitations of Time

NtApiDotNet (tooling used in presentation)
https://github.com/googleprojectzero/sandbox-attacksurface-analysis-tools

UAC Bypass Trickery
https://www.tiraniddo.dev/2022/03/bypassing-uac-in-most-complex-way.html

Remote Credential Guard Code Execution
https://bugs.chromium.org/p/project-zero/issues/detail?id=2271

AppContainer Escapes
https://bugs.chromium.org/p/project-zero/issues/detail?id=2273

LSASS Impersonation Check Failures
https://bugs.chromium.org/p/project-zero/issues/detail?id=2278

Service Account S4U Elevation
https://cyberstoph.org/posts/2021/06/abusing-kerberos-s4u2self-for-local-privilege-escalation/
Acknowledgements

Elad Shamir | @elad_shamir
Benjamin Delpy | @gentilkiwi
Will Schroeder | @harmjoy
Charlie Clark | @exploitph
Christoph Falta | @cfalta
One Last Thing!
Questions ?