SDC
STORAGE DEVELOPER CONFERENCE
SNIA SANTA CLARA, 2014

Introduction to SMB 3.1

Greg Kramer
Senior Software Engineer
Microsoft

David Kruse
Principal Software Engineer
Microsoft

The SMB 3.1 preview document (MS-SMB2-Diff) is available online:

http://msdn.microsoft.com/en-us/library/ee941641.aspx

Agenda

1. Extensible Negotiation
2. Preauthentication Integrity
3. Encryption Improvements
4. Cluster Dialect Fencing
5. Cluster Client Failover (CCF) v2
6. Public Service Announcements

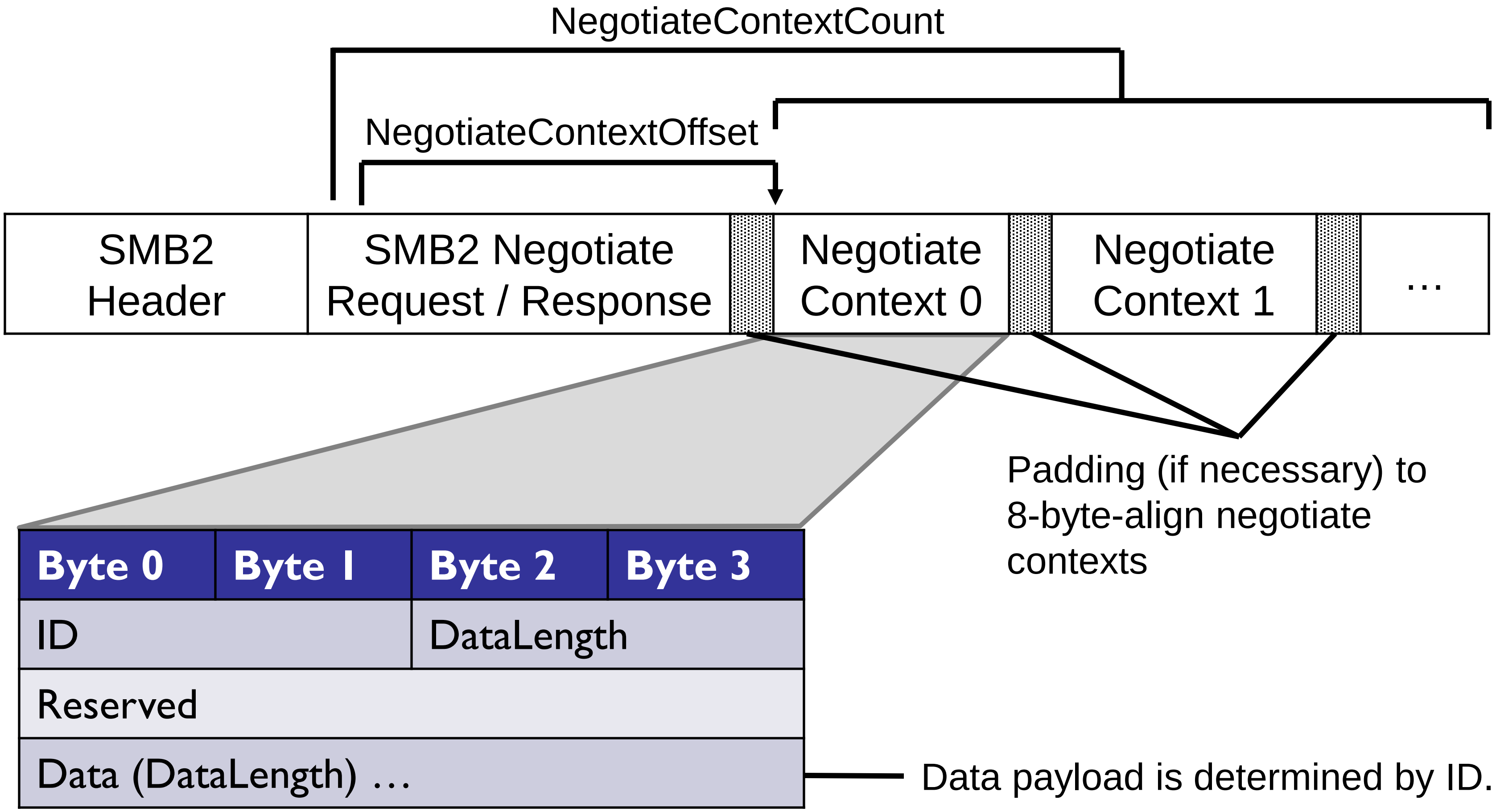
Agenda

1. Extensible Negotiation
2. Preauthentication Integrity
3. Encryption Improvements
4. Cluster Dialect Fencing
5. Cluster Client Failover (CCF) v2
6. Public Service Announcements

Overview

- How to negotiate complex connection capabilities?
  - Very few unused bits left in the negotiate request / response messages.
- SMB 3.1 Extensible Negotiation
  - Exchange additional negotiate information via negotiate contexts (same idea as the existing create contexts).
  - Repurpose unused fields in negotiate request / response as *NegotiateContextOffset* and *NegotiateContextCount* fields.
  - Add list of negotiate contexts to end of existing negotiate request / response messages.

Negotiate Contexts



Byte 0	Byte 1	Byte 2	Byte 3
ID		DataLength	
Reserved			
Data (DataLength) …			

Padding (if necessary) to 8-byte-align negotiate contexts

Data payload is determined by ID.

Key Points

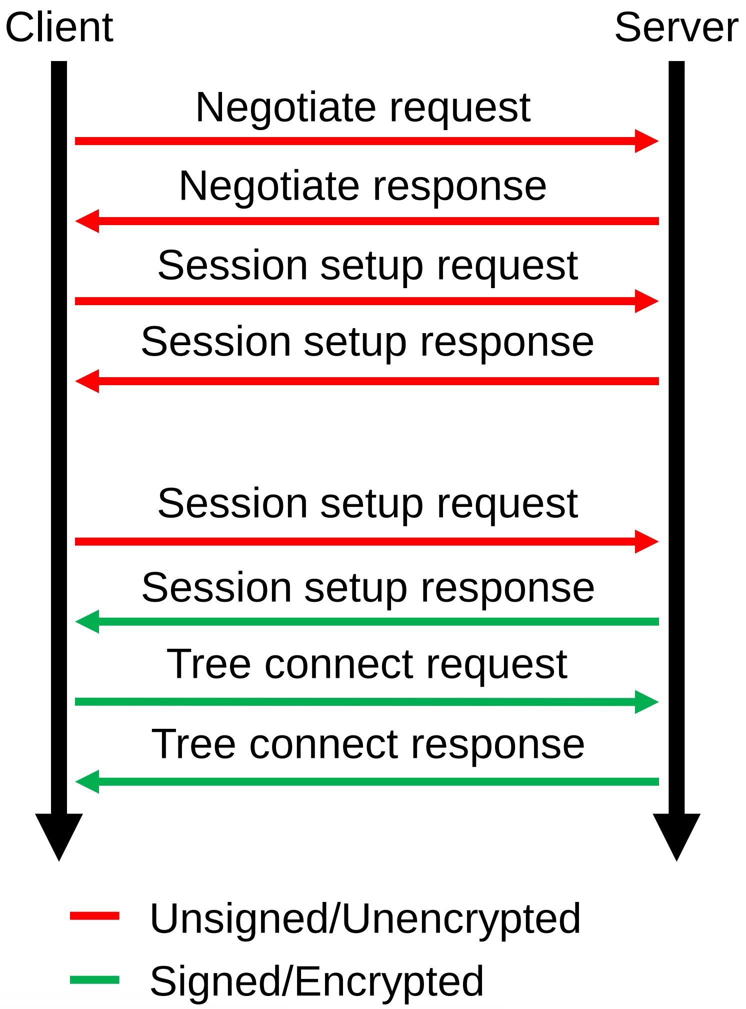
- Client sends negotiate contexts if it supports the 3.1 dialect.
- Server sends negotiate contexts if it selects 3.1 as the connection's dialect.
- Receiver must ignore unknown negotiate contexts.
- SMB 2/3 server implementations must be willing to accept negotiate requests that are larger than the SMB2_HEADER + SMB2_REQ_NEGOTIATE + Dialects array.
    - A client does not know whether a server supports SMB 3.1 before it negotiates, so must assume that it does and send negotiate contexts.
    - Windows accepts negotiate requests as large as 128 KiB

Agenda

1. Extensible Negotiation
2. Preauthentication Integrity
3. Encryption Improvements
4. Cluster Dialect Fencing
5. Cluster Client Failover (CCF) v2
6. Public Service Announcements

Overview

- How to protect preauthentication messages from tampering?
  - No protection prior to SMB 3.0
  - SMB 3.0x Negotiate Validation doesn’t protect negotiate contexts or session setup messages.
- SMB 3.1 Preauthentication Integrity
  - Provides end-to-end protection of preauthentication messages.
  - Session’s secret keys derived from hash of the preauthentication messages.
  - Signature validation/decryption of subsequent authenticated messages will fail in case of preauthentication message tampering.



Selecting a Hash Function

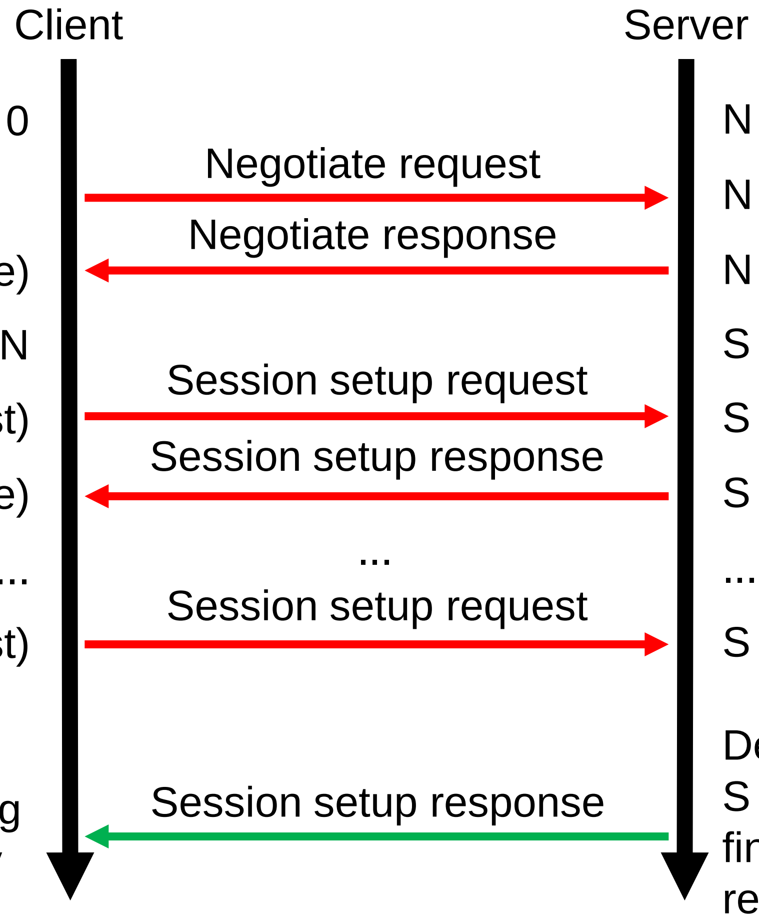
- SMB 3.1 client and server exchange mandatory negotiate contexts for each connection.
- Client's negotiate context specifies a set of supported hash functions.
- Server's negotiate context specifies the selected hash function.
- SHA-512 is currently the only supported hash function.

SMB2_PREAUTH_INTEGRITY_CAPABILITIES
(Negotiate Context ID: 0x0001)

Byte 0	Byte 1	Byte 2	Byte 3
HashAlgorithmCount		SaltLength	
HashAlgorithms			
…			
Salt			
…			

- Preimage attack resistance is provided by a salt value that the client and server generate via a secure PRNG per request/response.

Computing the Hash Value



Client

Server

N = 00…0

N = 00…0

Negotiate request

N = H(N || Request)

Negotiate response

N = H(H(N || Request) || Response)

N = H(N || Response)

S = N

S = N

Session setup request

S = H(S || Request)

S = H(S || Request)

Session setup response

S = H(S || Response)

S = H(S || Response)

...

...

...

Session setup request

S = H(S || Request)

S = H(S || Request)

Derive secret keys using S (next slide) and sign final session setup response.

Session setup response

Derive secret keys using S (next slide) and verify signature.

H(x): hash value of bit string x using the negotiated hash function
A || B: concatenation of bit strings A and B

Deriving Secret Keys from the Hash Value

DerivedKey = KDF[1](SessionKey, Label[2], Context)

Derived Key	Label
Application Key	“SMBAppKey”
Signing Key	“SMBSigningKey”
Client to server cipher key	“SMBC2SCipherKey”
Server to client cipher key	“SMBS2CCipherKey”

Session’s final preauthentication integrity hash value (S)

1. KDF is SP108-800-CTR-HMAC-SHA256 (same as SMB 3.0x)
2. Note that KDF labels have changed since SMB 3.0x

Security Analysis

Result of an attacker tampering with negotiate and/or session setup messages based on the resulting connection’s SMB dialect for a client and server that both attempt to negotiate SMB 3.1.

Connection Dialect	Result
3.1	Attack is detected when client fails to validate the signature of the final session setup response.
3.0x or 2.x	Dialect downgrade attack is detected by SMB 3.0x Negotiate Validation upon first tree connect.
1.x	Attack succeeds! SMB 1.x has no MITM attack mitigations

Note that SMB 3.1 Preauthentication Integrity cannot protect anonymous or guest sessions. The client and server can’t sign messages without an authenticated context.

Key Points

- Preauthentication Integrity is mandatory for SMB 3.1.
- Session setup hashes are only calculated for master and binding session setup exchanges, not reauthentication.
- Preauthentication Integrity supersedes SMB 3.0x Negotiate Validation for SMB 3.1 connections.
- Expect additional hardening based on security review before 3.1 is finalized

Agenda

1. Extensible Negotiation
2. Preauthentication Integrity
3. Encryption Improvements
4. Cluster Dialect Fencing
5. Cluster Client Failover (CCF) v2
6. Public Service Announcements

Overview

- SMB 3.0x mandates the AES-128-CCM cipher
  - What if a different cipher is required for performance, regulatory requirements, etc?
- SMB 3.1 Encryption Improvements
  - Ciphers are negotiated per-connection
  - Adding support for AES-128-GCM
  - Clients can mandate that sessions be encrypted even if the server does not require encryption.

Selecting a Cipher

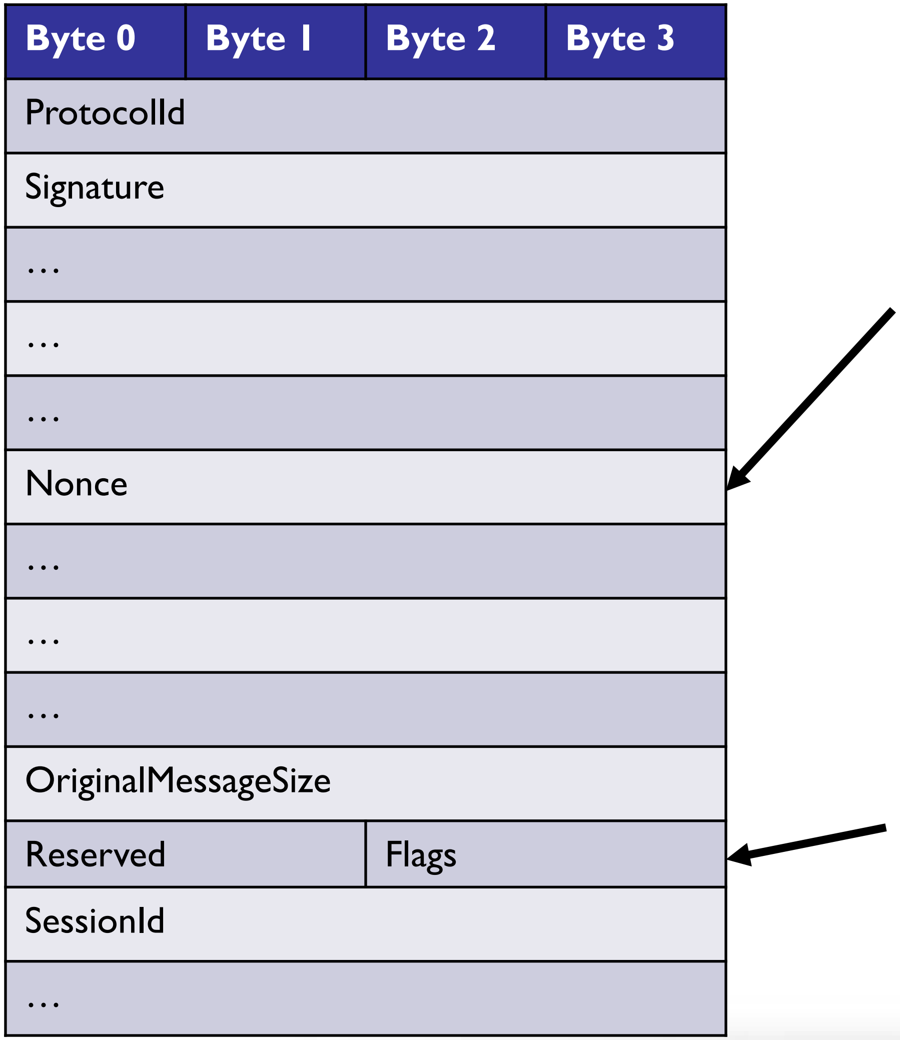
- SMB 3.1 client and server exchange negotiate contexts for each connection if they support encryption.
- Client’s negotiate context specifies a set of supported ciphers in order from most to least preferred.

SMB2_ENCRYPTION_CAPABILITIES
(Negotiate Context ID: 0x0002)

Byte 0	Byte 1	Byte 2	Byte 3
CipherCount		Ciphers	
...			

- Server’s negotiate context specifies the selected cipher.
  - Selection policy is server’s choice: client-preferred, server-preferred, etc.
  - Reserved cipher ID 0x0000 indicates that the client and server have no common cipher.
  - No SMB2_ENCRYPTION_CAPABILITIES context in server response indicates that the server does not support encryption.
- Encryption capabilities flag is never set in an SMB 3.1 Negotiate Response.

SMB2_TRANSFORM_HEADER Changes



Byte 0	Byte 1	Byte 2	Byte 3
ProtocolId			
Signature			
…			
…			
…			
Nonce			
…			
…			
…			
OriginalMessageSize			
Reserved		Flags	
SessionId			
…			

Nonce size determined by cipher:

Cipher	Nonce Size (bytes)
AES-128-CCM	11
AES-128-GCM	12

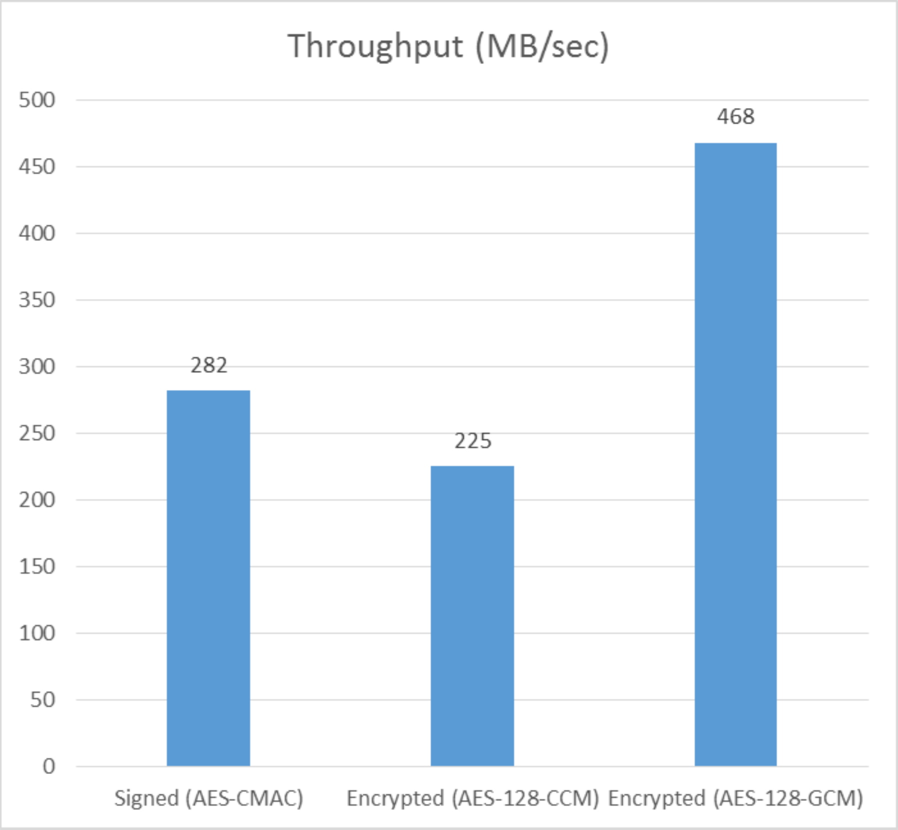
EncryptionAlgorithm field renamed to Flags:

Value	Meaning
0x0001	Payload is encrypted using cipher negotiated for the connection

AES-GCM Performance

- 2.0x faster than AES-128-CCM encryption.
- 1.6x faster than AES-CMAC signing!
  - Why use signing anymore?

Test configuration (client and server)	
CPU	2x Intel Xeon E5-2660 @ 2.2 GHz
Network	1x Intel Ethernet Server Adapter X520 @ 10 Gbps
Storage	SSD
Storage Workload	File copy (1 thread doing 8 async 1 MiB writes)



Client-mandated Encryption

- New capability for security-conscious clients.
- Client mandates session encryption by setting the SMB2_SESSION_FLAG_ENCRYPT_DATA flag in its session setup request.
- Server acknowledges the request by setting the SMB2_SESSION_FLAG_ENCRYPT_DATA flag in the session setup response.
- Server rejects all unencrypted requests for the session just as if the server had required encryption for the session.

Key Points

- AES-CCM required for SMB 3.0x compatibility.
- AES-GCM provides significant performance gains and should be supported.
- Session binding (multichannel) requires all of a session’s channels to negotiate the same cipher as the session’s original connection.
- Client-mandated encryption depends on SMB 3.1 and Preauthentication Integrity to guarantee security.
  - Not sufficient for client to simply send encrypted requests and verify encrypted responses.

Agenda

1. Extensible Negotiation
2. Preauthentication Integrity
3. Encryption Improvements
4. Cluster Dialect Fencing
5. Cluster Client Failover (CCF) v2
6. Public Service Announcements

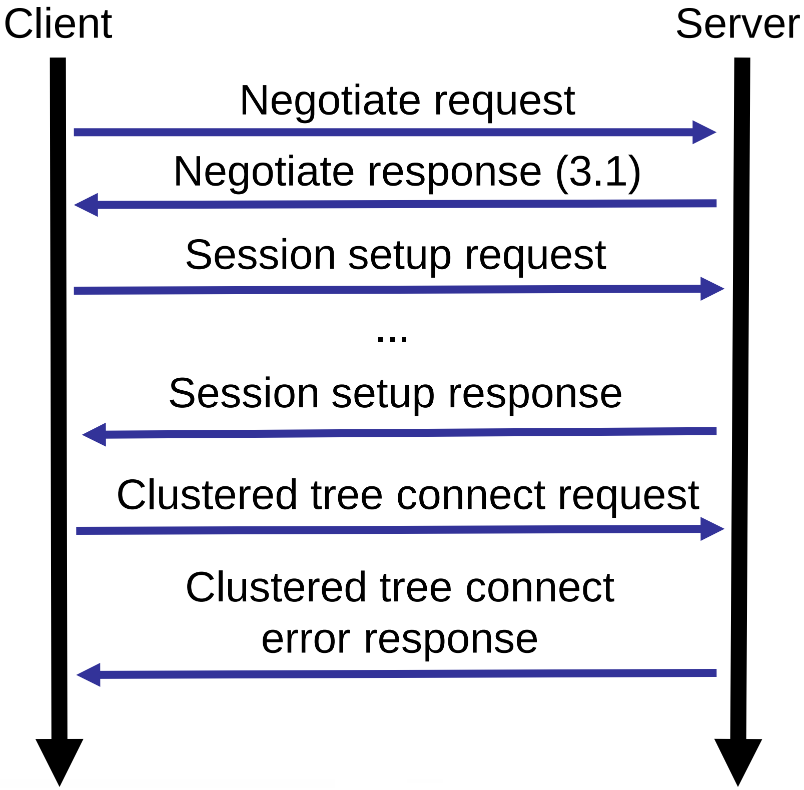
Overview

- How to support clustered file servers whose nodes have different maximum SMB dialects (for example 3.02 vs. 3.1)?
  - Currently, all cluster nodes must support the same maximum SMB dialect to allow a client to transparently failover between cluster nodes.
- SMB 3.1 Cluster Dialect Fencing
  - Define a maximum SMB cluster dialect that all nodes support.
  - Fence access to cluster shares based on the maximum SMB cluster dialect.
  - Fenced clients instructed to reconnect at a cluster-supported dialect.

Fencing Clustered Tree Connects

An SMB 3.1 client accesses a clustered file share on an SMB 3.1 server that is a member of a cluster whose maximum SMB cluster dialect is 3.02.

1. Client negotiates 3.1, authenticates then issues tree connect.
2. Server fails tree connect request with an extended error (status = 0xC05D0001) whose data payload indicates the maximum cluster-supported dialect (3.02).
3. Client disconnects, reconnects with new Client GUID, negotiates 3.02, authenticates, then reissues tree connect.



SMB2_TREE_CONNECT Request Changes

Byte 0	Byte 1	Byte 2	Byte 3
Structure Size		Flags	
PathOffset		PathLength	
Buffer			
…			

Reserved field renamed to Flags:

Value	Meaning
0x0001	Client has already successfully connected to a clustered file share on this server at the current SMB dialect.

- Once a client has successfully connected to a clustered share it must set the CLUSTER_RECONNECT (0x0001) flag on all subsequent clustered tree connect requests to the same server.
  - Addresses a race condition when the maximum SMB cluster dialect has been raised but some nodes have not yet begun allowing the new, higher dialect.

Key Points

- Dialect fencing only affects clustered share access.
  - Clients can still access non-clustered shares using dialect X even if the maximum SMB cluster dialect is < X.
  - Can’t mix clustered and non-clustered access on same connection.
- Client implementation should protect against infinite loop of tree connect failure, disconnect, reconnect, tree connect failure, …

Agenda

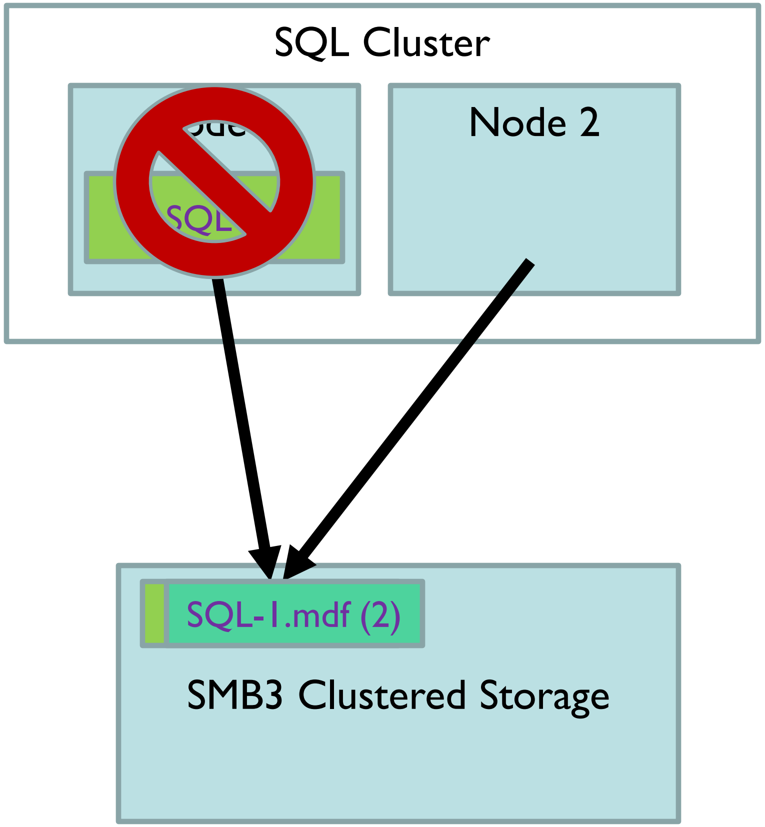
1. Extensible Negotiation
2. Preauthentication Integrity
3. Encryption Improvements
4. Cluster Dialect Fencing
5. Cluster Client Failover (CCF) v2
6. Public Service Announcements

Cluster Client Failover

- Introduced with SMB 3 for clustered applications using SMB 3 storage
- Permits clustered application to tag an open with *ApplicationInstance* identifier
- An open issued by a different client with the same *ApplicationInstance* indicates workload has transitioned to the new node, so old opens are closed.

Cluster Client Failover

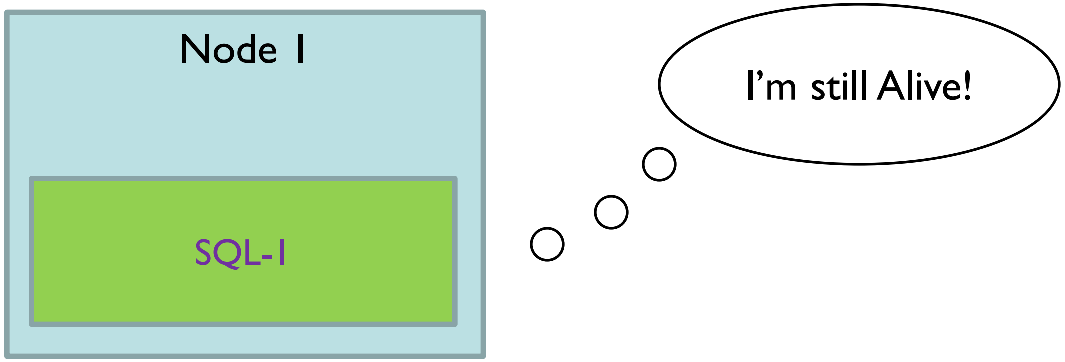
1. SQL-1.mdf is held open by Node 1 that is currently hosting the database
2. Node 1 dies.
    a) The storage system sees the disconnect and reserves the handle so Node 1 can recover if this is simply a disconnect
    b) The SQL cluster detects Node 1 is down, and starts the database on Node 2
3. Node 2 issues an open to the storage cluster.
    a) If this open has the same Application Instance but a different client GUID, it indicates the workload has failed over. Thus the existing SQL-1.mdf open is closed and Node 2 is permitted to open it.
    b) If this open does not have a matching Application Instance, the open is failed and held for Node 1 until the timeout is reached.



Cluster Client Failover

Solves the situation where resources are held on behalf of a client who has died.

But what if Node 1 is not dead? But simply has lost contact with the cluster coordinator?



Node 1 Lives…

- Node 1 is still healthy and has access to storage and network
- Node 1 can continue running the workload until a failure occurs
- In the event of a failure, it is ambiguous if this is a transient disconnect or if Node 1 has been superceded by Node 2 which is now hosting the workload.
  - If Node 1 attempts to retry after failure but the failure was caused by Node 2 taking over the workload, existing CCF logic will cause server to close Node 2’s handle, and the workload will **oscillate**.
  - If Node 1 does not attempt to retry after a failure occurs, than the workload is no longer running.

Cluster Client Failover v2

- The cluster coordinator knows which Node should be hosting the job. Along with the ApplicationInstance, it provides an ApplicationInstanceVersion to convey this knowledge to the application Node's.
- The Version is increased in some fashion every time the workload is moved.

```
struct SMB2_CREATE_APP_INSTANCE_VERSION  {
   USHORT Size;
   USHORT Reserved;
   UINT64 VersionHigh;
   UINT64 VersionLow;
};
```

Cluser Client Failover v2

- SMB3.1 Client must
  - Pass ApplicationInstanceVersion alongside ApplicationInstance on create
  - It should attempt to keep the handle alive until it receives a non-ambiguous status code from the server indicating it has been superceded by another node (or the handle has timed out)

Cluster Client Failover v2

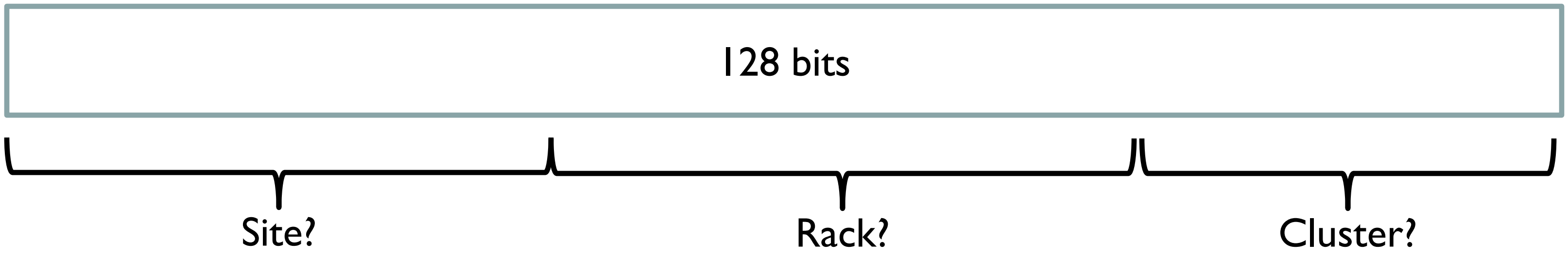
- SMB3.1 Server must
  - Compare the ApplicationInstanceVersion on an invalidating open.
    - If the version is higher, the existing open should be orphaned as normal.
    - If the version is lower, the incoming open is failed with a non-ambiguous status code indicating it has been superceded.

Cluster Client Failover v2

- To interact with older (pre-SMB 3.1) clients
  - Opens without an version are assumed to be version 0
  - A version 0 open will successfully invalidate other version 0 opens
  - Otherwise, the same rules apply
  - It would be unexpected for an application cluster to run a mix of 3.0 and 3.1 clients sharing the same workload.

Cluster Client Failover v2

- Version space is intentionally large (16 bytes) to enable hierarchical delegation of application, but its use us up to the client



Key Points

- The CCF2 extension permits a client machine to keep your clustered application running during failover situations, but release it when the workload has been formally moved.
- Extending your storage cluster to support CCF2 is simple

Agenda

1. Extensible Negotiation
2. Preauthentication Integrity
3. Encryption Improvements
4. Cluster Dialect Fencing
5. Cluster Client Failover (CCF) v2
6. Public Service Announcements

Changes coming in future Windows releases

- Removing RejectUnencryptedAccess setting
  - Always reject clients that don’t support encryption when connecting to a server/share that requires encryption.
- Removing RequireSecureNegotiate setting
  - Always perform negotiate validation if the connection’s dialect is 2.x or 3.0x.
- Restricting use of guest sessions
  - Indistinguishable from man-in-the-middle attack.
  - Let us know ASAP if you have a scenario that requires guest logons using SMB 2.x or 3.x.