14

Session Setup

> ...it is a tale
> Told by an idiot, full of sound
> and fury, signifying nothing.
>
> — *Macbeth*, Act V, Scene v,
> William Shakespeare

Originally, the `SESSION SETUP` was not required by — or even defined as part of — the SMB protocol. It was introduced in the LANMAN days in order to handle User Level authentication and could be skipped if the server was in Share Level security mode. These days, however, the `SESSION SETUP` takes care of a lot of unfinished business, like cleaning up some of the debris left by the `NEGOTIATE PROTOCOL RESPONSE`. In the NT LM 0.12 dialect there *must* be a `SESSION SETUP` exchange before a `TREE CONNECT` may be sent, even if the server is operating in Share Level security mode.

14.1 `SESSION SETUP ANDX REQUEST` Parameters

The `SESSION SETUP` SMB is actually a `SESSION SETUP ANDX`, which simply means that there's an AndX block in the parameter section. In the NT LM 0.12 dialect, the Parameter block is formatted as shown below:

```
typedef struct
  {
  uchar WordCount;  /* 12 or 13 words */
  struct
    {
    struct
      {
      uchar  Command;
      uchar  Reserved;
      ushort Offset;
      } AndX;
    ushort MaxBufferSize;
    ushort MaxMpxCount;
    ushort VcNumber;
    ulong  SessionKey;
    ushort Lengths[];  /* 1 or 2 elements */
    ulong  Reserved;
    ulong  Capabilities;
    } Words;
  } smb_SessSetupAndX_Req_Params;
```

When looking at these C-like structures, keep in mind that they are intended as descriptions rather than specifications. On the wire, the parameters are packed tightly into the SMB messages, and they are not aligned. Though the structures show the type and on-the-wire ordering of the fields, the C programming language does not guarantee that the layout will be retained in memory. That's why our example code includes all those functions and macros for packing and unpacking the packets.[1]

Many of the fields in the `SESSION_SETUP_ANDX.SMB_PARAMETERS` block should be familiar from the `NEGOTIATE PROTOCOL RESPONSE` SMB. This time, though, it's the client's turn to set the limits.

`MaxBufferSize`

`MaxBufferSize` is the size (in bytes) of the largest message that the *client* can receive. It is typically less than or equal to the server's `MaxBufferSize`, but it doesn't need to be.

---

1. To be pedantic, the correct terms are "marshaling" and "unmarshaling." "Marshaling" means collecting data in system-internal format and re-organizing it into a linear format for transport to another system (virtual, physical, or otherwise). "Unmarshaling," of course, is the reverse process. These terms are commonly associated with Remote Procedure Call (RPC) protocols, but some have argued (not unreasonably) that SMB is a simple form of RPC.

MaxMpxCount

This must always be less than or equal to the server-specified `MaxMpxCount`. This is the client's way of letting the server know how many outstanding requests it will allow. The server might use this value to pre-allocate resources.

VcNumber

This field is used to establish a Virtual Circuit (VC) with the server. Keep reading, we're almost there...

SessionKey

Just echo back whatever you got in the `NEGOTIATE PROTOCOL RESPONSE`.

Lengths

For efficiency's sake the structure above provides the `Lengths` field, defined as an array of unsigned short integers and described as having one or two elements. The SNIA doc and other references go to a lot more trouble and provide two separate and complete versions of the entire `SESSION SETUP REQUEST` structure.

Basically, though, if Extended Security has been negotiated then the `Lengths` field is a single `ushort`, known as `SecurityBlobLength` in the SNIA doc. (We touched on the concept of security blobs briefly back in Section 13.3.2.) If Extended Security is *not* in use then there will be two `ushort` fields identified by the excessively long names:

- `CaseInsensitivePasswordLength` and
- `CaseSensitivePasswordLength`.

Obviously, all of this stuff falls into the general category of authentication, and will be covered in more detail when we finally focus on that topic.

Reserved

Four bytes of must-be-zero.

Capabilities

This field contains the client capabilities flag bits.

You might notice, upon careful examination, that the client does not send back a `MaxRawSize` value. That's because it can specify raw read/write sizes in the `SMB_COM_RAW_READ` and `SMB_COM_RAW_WRITE` requests, if it sends them. These SMBs are considered obsolete, so newer clients really shouldn't be using them.

There are a couple of fields in the `SESSION SETUP REQUEST` which touch on esoteric concepts that we have been promising to explain for quite a while now — specifically virtual circuits and capabilities — so let's get it over with...

14.1.1 *Virtual Circuits*

It does seem as though there's a good deal of cruft in the SMB protocol. The `SessionKey`, for example, appears to be a vestigial organ, the purpose of which has been mostly forgotten. Originally, such fields may have been intended to compensate for a limitation in a specific transport or an older implementation, or to solve some other problem that isn't a problem any more.

Consider virtual circuits...

The LAN Manager documentation available from Microsoft's ftp site provides the best clues regarding virtual circuits (see `SMB-LM1X.PS`, for instance). According to those docs a virtual circuit (VC) represents a single transport layer connection, and the `VcNumber` is a tag used to identify a specific transport link between a specific client/server pair.

That concept probably needs to be considered in context.

The LANMAN dialects were developed in conjunction with OS/2 (an honest-to-goodness, really-truly, multitasking OS). OS/2 clients pass SMB traffic through a redirector — just like DOS and Windows — and it seems as though there was some concern that multiplexing the SMB traffic from several processes across a single connection might cause a bit of a bottleneck. So, to avoid congestion, the redirector could create additional connections to facilitate faster transfers for individual processes.[2] Under this scheme, all of the transport level connections from a client to a server were considered part of a single logical

2. If you enjoy digging into odd details, this is a great one. See the `SMB-LM1X.PS` file, also known as Microsoft Networks/SMB File Sharing Protocol Extensions, Version 2.0, Document Version 3.3. In particular, see the definition of a VC on page 2, and the description of the "Virtual Circuit Environment" in Section 4.a on page 10.

"session" (we now, officially, have way too many meanings for that term). Within that logical session there could, conversely, be multiple transport level connections — aka virtual circuits — up to the limit set in the `NEGOTIATE PROTOCOL RESPONSE`.

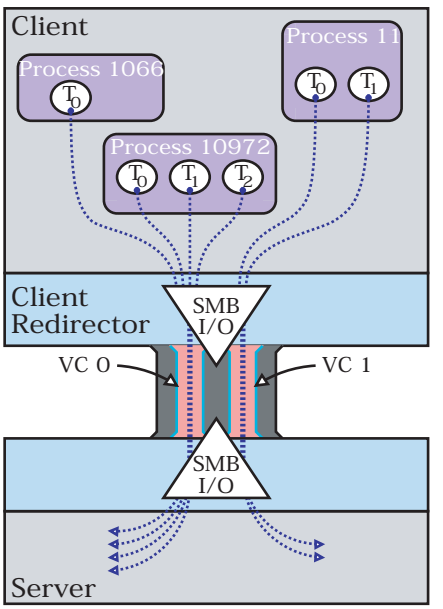

Figure 14.1: *Virtual circuits*

Process 11 has the use of virtual circuit number 1 (VC 1). VC 0 and VC 1 are separate TCP/IP connections, yet both VCs are part of the same logical client/server "session" (and the term "session" is clearly overused).

Figure 14.1 illustrates the point, and here's how it's supposed to work:

- Logical Session Creation
  - The client makes an initial connection to the SMB server, performs the `NEGOTIATE PROTOCOL` exchange, and establishes the session by sending a `SESSION SETUP ANDX REQUEST`.
  - The `VcNumber` in the initial `SESSION SETUP ANDX REQUEST` is zero (`0`).
- Additional VC Creation
  - An additional transport level connection is created.

- The client sends a new `SESSION SETUP ANDX REQUEST` with a `VcNumber` greater than zero, but less than the `MaxNumberVCs` sent by the server.
- The `SessionKey` field in the `SESSION SETUP ANDX REQUEST` must match the `SessionKey` returned in the initial `NEGOTIATE PROTOCOL RESPONSE`. That's how the new VC is bound to the existing logical session.

Ah-hahhh! The mystery of the `SessionKey` field is finally revealed. Kind of a let-down, isn't it?

Whenever a new transport-layer connection is created, the client is supposed to assign a new VC number. Note that the `VcNumber` on the initial connection is expected to be zero to indicate that the client is starting from scratch and is creating a new logical session. If an additional VC is given a `VcNumber` of zero, the server *may* assume that any existing connections with that same client are now bogus, and shut them down.

Why do such a thing?

The explanation given in the LANMAN documentation, the Leach/Naik IETF draft, and the SNIA doc is that clients may crash and reboot without first closing their connections. The zero `VcNumber` is the client's signal to the server to clean up old connections. Reasonable or not, that's the logic behind it. Unfortunately, it turns out that there are some annoying side-effects that result from this behavior. It is possible, for example, for one rogue application to completely disrupt SMB filesharing on a system simply by sending Session Setup requests with a zero `VcNumber`. Connecting to a server through a NAT (**N**etwork **A**ddress **T**ranslation) gateway is also problematic, since the NAT makes multiple clients appear to be a single client by placing them all behind the same IP address.[3]

The biggest problem with virtual circuits, however, is that they are not really needed any more (if, in fact, they ever were). As a result, they are handled inconsistently by various implementations and are not entirely to be trusted. On the client-side, the best thing to do is to ignore the concept and view each transport connection as a separate logical session, one VC per session. Oh! ...and contrary to the specs the client should always use a `VcNumber` of one, never zero.

---

3. See *Microsoft Knowledge Base Article #301673* for more information.

On the server side, it is important to keep in mind that the TID, UID, PID, and MID are all supposed to be relative to the VC. In particular, TID and UID values negotiated on one VC have no meaning (and no authority) on another VC, even if both VCs appear to be from the same client. Another important note is that the server should *not* disconnect existing VCs upon receipt of a new VC with a zero VcNumber. As described above, doing so is impractical and may break things. The server should let the transport layer detect and report session disconnects. At most, a zero VcNumber might be a good excuse to send a keep-alive packet.

The whole VC thing probably seemed like a good idea at the time.

14.1.2 Capabilities *Bits*

Remember a little while back when we said that there were subtle variations within SMB dialects? Well, some of them are not all that subtle once you get to know them. The Capabilities bits formalize several such variations by letting the client and server negotiate which special features will be supported. The server sends its Capabilities field in the NEGOTIATE PROTOCOL RESPONSE, and the client returns its own set of capabilities in the SESSION SETUP ANDX REQUEST.

The table below provides a listing of the capabilities defined for servers. The client set is smaller.

Server capabilities

Bit	Name / Bitmask	Description
31	CAP_EXTENDED_SECURITY 0x80000000	Set to indicate that Extended Security exchanges are supported.
30	CAP_COMPRESSED_DATA 0x40000000	If set, this bit indicates that the server can compress Data blocks before sending them.[4] This might be useful to improve throughput of large file transfers over low-bandwidth links. This capability requires

4. There are a few small notes scattered about the SNIA doc that suggest that the prescribed compression algorithm is something called LZNT. I haven't been able to find a definitive reference that explains what LZNT is, but it appears from the name that it is a form of Lempel-Ziv compression.

Server capabilities

Bit	Name / Bitmask	Description
		that the CAP_BULK_TRANSFER capability also be set. Currently, however, there are no known implementations that support bulk transfer.
29	CAP_BULK_TRANSFER 0x20000000	If set, the server supports the SMB_COM_READ_BULK and SMB_COM_WRITE_BULK SMBs. There are no known implementations which support CAP_BULK_TRANSFER and/or CAP_COMPRESSED_DATA. Samba does not even bother to define constants for these capabilities.
23	CAP_UNIX 0x00800000	Microsoft reserved this bit based on a proposal (by Byron Deadwiler at Hewlett-Packard) for a small set of Unix extensions. The SNIA doc describes these extensions in an appendix. Note, however, that the proposal was made and the appendix written before the extensions were widely implemented. Samba supports the SMB Unix extensions, but probably not exactly as specified in the SNIA doc.
15	CAP_LARGE_WRITEX 0x00008000	If set, the server supports a special mode of the SMB_COM_WRITE_ANDX SMB which allows the client to send more data than would normally fit into the server's receive buffers, up to a maximum of 64 Kbytes.
14	CAP_LARGE_READX 0x00004000	Similar to the CAP_LARGE_WRITEX, this bit indicates whether the server can handle SMB_COM_READ_ANDX requests for blocks of data larger than the reported maximum buffer size. The theoretical maximum is 64 Kbytes, but the client should never request more data than it can receive.
13	CAP_INFOLEVEL_PASSTHROUGH 0x00002000	Samba calls this the CAP_W2K_SMBS bit. In testing, NT 4.0 systems did not set this bit, but W2K systems did. Basically, it indicates support for some advanced requests.
12	CAP_DFS 0x00001000	If set, this bit indicates that the server supports Microsoft's Distributed File System.

Server capabilities

Bit	Name / Bitmask	Description
9	CAP_NT_FIND 0x00000200	This is a mystery bit. There is very little documentation about it and what does exist is not particularly helpful. The SNIA doc simply says that this bit is "Reserved," but the notes regarding the CAP_NT_SMBS bit state that the latter implies the former. (Counter-examples have been found in some references, but not on the wire during testing. Your mileage may vary.) Basically, though, if this bit is set it indicates that the server supports an extended set of function calls belonging to a class of calls known as "transactions."
8	CAP_LOCK_AND_READ 0x00000100	If set, the server is reporting that it supports the obsolete SMB_COM_LOCK_AND_READ SMB. ...but go back and look at the SMB_HEADER.FLAGS bits described earlier. The lowest order FLAGS bit is SMB_FLAGS_SUPPORT_LOCKREAD, and it is also supposed to indicate whether or not the server supports SMB_COM_LOCK_AND_READ (as well as the complimentary SMB_COM_WRITE_AND_UNLOCK). The thing is, traces from Windows NT and Windows 2000 systems show the CAP_LOCK_AND_READ bit set while the SMB_FLAGS_SUPPORT_LOCKREAD is clear. That doesn't make a lot of sense. Well... it *may* be that the server is indicating that it supports the SMB_COM_LOCK_AND_READ SMB but *not* the SMB_COM_WRITE_AND_UNLOCK SMB, or it may be that the server may be using the Capabilities field in preference to the FLAGS field. Avoid the use of the SMB_COM_LOCK_AND_READ and SMB_COM_WRITE_AND_UNLOCK SMBs and everything should turn out alright.

Server capabilities

Bit	Name / Bitmask	Description
7	CAP_LEVEL_II_OPLOCKS 0x00000080	If set, Level II OpLocks are supported in addition to Exclusive and Batch OpLocks.
6	CAP_STATUS32 0x00000040	If set, this bit indicates that the server supports the 32-bit NT_STATUS error codes.
5	CAP_RPC_REMOTE_APIS 0x00000020	If set, this bit indicates that the server permits remote management via Remote Procedure Call (RPC) requests. RPC is way beyond the scope of this book.
4	CAP_NT_SMBS 0x00000010	If set, this bit indicates that the server supports some advanced SMBs that were designed for use with Windows NT and above. These are, essentially, an extension to the NT LM 0.12 dialect. According to the SNIA doc, the CAP_NT_SMBS implies CAP_NT_FIND.
3	CAP_LARGE_FILES 0x00000008	If set, this bit indicates that the server can handle 64-bit file sizes. With 32-bit file sizes, files are limited to 4 GB in size.
2	CAP_UNICODE 0x00000004	Set to indicate that the server supports Unicode.
1	CAP_MPX_MODE 0x00000002	If set, the server supports the (obsolete) SMB_COM_READ_MPX and SMB_COM_WRITE_MPX SMBs.
0	CAP_RAW_MODE 0x00000001	If set, the server supports the (obsolete) SMB_COM_READ_RAW and SMB_COM_WRITE_RAW SMBs.

On the server side, the implementor's rule of thumb regarding capabilities is to start by supporting as few as possible and add new ones one at a time. Each bit is a cornucopia — or Pandora's box — of new features and requirements, and most represent a very large development effort. As usual, if there is documentation it is generally either scarce or encumbered.

Things are not quite so bad if you are implementing a client, though the client also has a list of capabilities that it can declare. The client list is as follows:

Client capabilities

Bit	Name / Bitmask	Description
31	CAP_EXTENDED_SECURITY 0x80000000	Set to indicate that Extended Security exchanges are supported. The SNIA doc and the older IETF Draft do not list this as a capability set by the client. On the wire, however, it is clearly used as such by Windows, Samba, and by Steve French's CIFS VFS for Linux. If the server indicates Extended Security support in its Capabilities field, then the client may set this bit to indicate that it also supports Extended Security.
9	CAP_NT_FIND 0x00000200	If set, this bit indicates that the client is capable of utilizing the CAP_NT_FIND capability of the server.
7	CAP_LEVEL_II_OPLOCKS 0x00000080	If set, this bit indicates that the client understands Level II OpLocks.
6	CAP_STATUS32 0x00000040	Indicates that the client understands 32-bit NT_STATUS error codes.
4	CAP_NT_SMBS 0x00000010	Likewise, I'm sure. As with the CAP_NT_FIND bit, the client will set this to let the server know that it, too, understands the extended set of SMBs and function calls that are available if the server has set the CAP_NT_SMBS bit.
3	CAP_LARGE_FILES 0x00000008	The client sets this to let the server know that it can handle 64-bit file sizes and offsets.
2	CAP_UNICODE 0x00000004	Set to indicate that the client understands Unicode.

The client should not set any bits that were not also set by the server. That is, the Capabilities bits sent *to* the server should be the intersection (bitwise AND) of the client's actual capabilities and the set sent *by* the server.

The Capabilities bits are like the razor-sharp barbs on a government fence. Attempting to hurdle any one of them can shred your implementation.

Consider adding Unicode support to a system that doesn't already have it. Ooof! That's going to be a lot of work.[5]

Some `Capabilities` bits indicate support for sets of function calls that can be made via SMB. These function calls, which are sometimes referred to as "sub-protocols," fall into two separate (but similar) categories:

- **R**emote **A**dministration **P**rotocol (RAP),
- **R**emote **P**rocedure **C**all (RPC).

Of the two, the RAP sub-protocol is older and (relatively speaking) simpler. Depending upon the SMB dialect, server support for some RAP calls is assumed rather than negotiated. Fortunately, much of RAP is documented... if you know where to look.[6]

Microsoft's RPC system — known as MS-RPC — is newer, and has a lot in common with the better-known DCE/RPC system. MS-RPC over SMB allows the client to make calls to certain Windows DLL library functions on the server side which, in turn, allows the client to do all sorts of interesting things. Of course, if you are building a server and you want to support the MS-RPC calls you have to implement all of the required functions in addition to SMB itself. Unfortunately, much of MS-RPC is undocumented.[7]

The MS-RPC function call APIs are defined using a language called **M**icrosoft **I**nterface **D**efinition **L**anguage (MIDL). There is a fair amount of information about MIDL available on the web and *some* of the function interface definitions have been published. CIFS implementors have repeatedly asked Microsoft for open access to all of the CIFS-relevant MIDL source files. Unencumbered access to the MIDL source would go a long way towards opening up the CIFS protocol suite. Since MIDL provides only the interface

---

5. It was, in fact, a lot of work for the Samba Team. Those involved did a tremendous job, and they deserve several rounds of applause. Things were much easier for jCIFS because Java natively supports Unicode.

6. Information on RAP calls is scattered among several sources, including the archives of Microsoft's CIFS mailing list. The SNIA doc has enough to get you started with the basics of RAP, but see also the file `cifsrap2.txt` which can be found on Microsoft's aforementioned FTP site.

7. Luke Kenneth Casson Leighton's book *DCE/RPC over SMB: Samba and Windows NT Domain Internals* is an essential reference for CIFS developers who need to know more about MS-RPC.

specifications and not the function internals, Microsoft could release them without exposing their proprietary DLL source code.

Both the RAP and MS-RPC sub-protocols provide access to a large set of features, and both are too big to be covered in detail here. Complete documentation of all of the nooks and crannies of CIFS would probably require a set of books large enough to cause an encyclopedia to cringe in awe, so it would seem that our attempt to clean up the mess we made with the NEGOTIATE PROTOCOL exchange has instead created an even bigger mess and left some permanent stains on the carpet. Ah, well. Such is the nature of CIFS.

14.2 SESSION SETUP ANDX REQUEST Data

The dissection of the SMB_PARAMETERS portion of the SESSION SETUP ANDX REQUEST cleared up a few issues and exposed a few others. Now we get to look at the SMB_DATA block and see what further mysteries may lie uncovered.

Fortunately, the Data block is much less daunting. It contains a few fields used for authentication and the rest is just useful bits of information about the client's operating environment. The structure looks like this:

```
typedef struct
  {
  ushort ByteCount;
  struct
    {
    union
      {
      uchar SecurityBlob[];
      struct
        {
        uchar CaseInsensitivePassword[];
        uchar CaseSensitivePassword[];
        uchar Pad[];
        uchar AccountName[];
        uchar PrimaryDomain[];
        } non_ext_sec;
      } auth_stuff;
    uchar NativeOS[];
    uchar NativeLanMan[];
    uchar Pad2[];
    } Bytes;
  } smb_SessSetupAndx_Req_Data;
```

auth_stuff

As you may by now have come to expect, the structure of the `auth_stuff` field depends upon whether or not Extended Security has been negotiated. We have shown it as a union type just to emphasize the point. Under Extended Security, the blob will contain a structure specific to the type of Extended Security being used. The `SecurityBlobLength` value in the Parameter block indicates the size (in bytes) of the `SecurityBlob`.

If Extended Security has not been negotiated, the structure will contain the following fields:

CaseInsensitivePassword and CaseSensitivePassword

If these names seem familiar it's because the associated length fields were in the Parameter block, described above. These fields are, of course, used in authentication. Chapter 15 on page 257 covers authentication in detail.

Pad

If Unicode is in use, then the `Pad` field will contain a single nul byte (`0x00`) to force two-byte alignment of the following fields (which are Unicode strings).

As you know, the Parameter block is made up of a single byte followed by an array of zero or more words. It starts on a word boundary, but the `WordCount` byte knocks it off balance, so it never ends on a word boundary. That means that the Data block always starts misaligned.[8] Typically, that's not considered a problem for data in SMB messages. It is not clear why, but it seems that when Unicode support was added to SMB it was decided that Unicode strings should be word-aligned within the SMB message (even though they are likely to be copied out of the message before they're fiddled). That's why the `Pad` byte is there.

8. I vaguely remember a conversation with Tridge in which he indicated that there was an obscure exception to the misalignment of the Data block. I'm not sure which SMB, or which dialect, but if I recall correctly there's one SMB that has an extra byte just before the `ByteCount` field. Keep your eyes open.

Note that if Unicode support is enabled the password fields will always contain an even number of bytes. Strange but true. Here's why:

- On Windows server systems, plaintext passwords and Unicode are mutually exclusive. The password hashes used for authentication are always an even number of bytes.
- Unlike Windows, Samba *can* be configured to use plaintext passwords and Unicode. In that configuration, the `CaseInsensitivePassword` field will be empty and the `CaseSensitivePassword` field will contain the password in Unicode format — two bytes per character.

Note the subtle glitch here. If Samba is configured to send Unicode plaintext passwords, the `CaseSensitivePassword` field will *not* be word-aligned because the `Pad` byte comes afterward. It seems that the designers of the NT LM 0.12 dialect did not consider the possibility of plaintext Unicode passwords.

`AccountName`

This is the username field. If Unicode has been negotiated, then the username is presented in Unicode. Otherwise, the string is converted to uppercase and sent using the 8-bit OEM character set.

`PrimaryDomain`

As with the `AccountName`, this value is converted to uppercase unless it is being sent in Unicode format.

Whenever possible, this field should contain the NetBIOS name of the NT Domain to which the user belongs. Basically, it allows the client to specify the NT Domain in which the username and password are valid — the Authentication Domain. A correct value is not always needed, however. If the server is not a member of an NT Domain, then it will have its own authentication database, and no Domain Controller need be consulted.

Some testing was done with Windows NT 4.0 and Windows 2000 systems that were not members of an NT Domain. As clients, these systems sent their own NetBIOS machine names in the `PrimaryDomain` field. The `smbclient` utility sent the workgroup name, as specified in the `smb.conf` file. jCIFS just sent

a question mark. All of these variations seem to work, as long as the server maintains its own authentication database. The `PrimaryDomain` field is really only useful when authenticating against a Domain Controller.

...and that's the end of the `auth_stuff` block. On to the rest of it.

`NativeOS`

This string identifies the host operating system. Windows systems, of course, will fill this field with their OS name and some revision information. This field will be expressed in Unicode if that format has been negotiated.

`NativeLanMan`

Similar to the `NativeOS` field, this one contains a short description of the client SMB software. `Smbclient` fills this field with the name "Samba." jCIFS used to just say "foo" here, but starting with release 0.7.0beta10 it says "jCIFS." The successful use of "foo" demonstrates, however, that the field is not used for anything critical on the server side. Just error reporting, most likely.

Email

```
   From: Gerald (Jerry) Carter
     To: Chris Hertel
Subject: NativeLanMan

Note that NT4 misaligns the NativeLanMan string by one byte
(see Ethereal for details). Also note that Samba uses this
string to distinguish between W2K/XP/2K3 for the %a smb.conf
variable. So it is used by the server in some cases.
```

`Pad2`

Some systems add one or two extra nul bytes at the end of the `SESSION SETUP`. Not all clients do this; it appears to be more common if Unicode has been negotiated. The extra bytes pad the end of the `SESSION SETUP` to the next word boundary. If these bytes are present, they are generally included in the total count given in the `ByteCount` field.

We have done a lot of work ripping apart packet structures and studying the internal organs. Don't worry, that's the last of it. You should be familiar enough with this stuff by now, so from here on out we will rely on the SNIA doc and packet traces to provide the gory details.

Don't Know When to Quit Alert

Some of the Windows systems that were tested did not place the correct number of nul bytes at the ends of some Unicode strings. Consider, for example, this snippet from an Ethereal capture:

```
0000029F                         57 00 69 00 6e 00 64 00          W.i.n.d.
000002AF 6f 00 77 00 73 00 20 00 4e 00 54 00 20 00 31 00 o.w.s. . N.T. .1.
000002BF 33 00 38 00 31 00 00 00 00 00 57 00 69 00 6e 00 3.8.1... ..W.i.n.
000002CF 64 00 6f 00 77 00 73 00 20 00 4e 00 54 00 20 00 d.o.w.s.  .N.T. .
000002DF 34 00 2e 00 30 00 00 00 00 00 4...0... ..
```

Look closely, and you will see that there are two extra nul bytes following each of the two Unicode strings in the hex dump. Under UCS-2LE encoding, the nul string terminator would be encoded as two nul bytes (`00 00`*). In the sample above, however, there are four null bytes (*`00 00 00 00`*) following the last Unicode character of each string.*

In this next excerpt, taken from a `SESSION SETUP ANDX RESPONSE` *SMB, it appears as though one of the terminating nul bytes at the end of the* `PrimaryDomain` *field has been lost:*

```
0000008F                                           57 00              W.
0000009F 69 00 6e 00 64 00 6f 00 77 00 73 00 20 00 35 00 i.n.d.o. w.s. .5.
000000AF 2e 00 30 00 00 00 57 00 69 00 6e 00 64 00 6f 00 ..0...W. i.n.d.o.
000000BF 77 00 73 00 20 00 32 00 30 00 30 00 30 00 20 00 w.s. .2. 0.0.0. .
000000CF 4c 00 41 00 4e 00 20 00 4d 00 61 00 6e 00 61 00 L.A.N. . M.a.n.a.
000000DF 67 00 65 00 72 00 00 00 55 00 42 00 49 00 51 00 g.e.r... U.B.I.Q.
000000EF 58 00 00                                        X..
```

The first two bytes of the last line (`58 00`*) are the letter 'X' in UCS-2LE encoding. They should be followed by two nul bytes... but there's only one.*

14.3 The `SESSION SETUP ANDX RESPONSE` SMB

The `SESSION SETUP ANDX RESPONSE` SMB structure is described in Section 4.1.2 of the SNIA doc.

In the NT LM 0.12 dialect, there are two versions of the `SESSION SETUP ANDX RESPONSE` message. They differ, of course, based on whether or not Extended Security is in use. In the Extended Security version the

Parameter block has a `SecurityBlobLength` field, and there is an associated `SecurityBlob` within the Data block. These two fields are missing from the non-Extended Security version. Other than that, the two are the same.

The `SESSION SETUP ANDX RESPONSE` message also has an interesting little bitfield called `SMB_PARAMETERS.Action`. Only the low-order bit (bit 0) of this field is defined. If set, it indicates that the username was not recognized by the server (that is, authentication failed — no such user) but the logon is being allowed to succeed anyway.

That's rather odd, eh?

What it means is this: If the username (in the `AccountName` field) is not recognized, the server *may* choose to grant *anonymous* or *guest* authorization instead. Anonymous access typically provides only very limited access to the server. For example, it may allow the use of a limited set of RAP function calls such as those used for querying the Browse Service.

So, the `Action` bit is used to indicate that the logon attempt failed, but anonymous access was granted instead. No error code will be returned in this case, so the `Action` bit is the only indication to the client that the rules have changed. Server-side support for this behavior is optional.