Network Analysis Techniques

Andrew Tridgell
tridge@samba.org

Many protocols are less well documented than you might wish.
Don't let that stop you implementing them.

Why decode a protocol?

- Proprietary protocol
- proprietary extensions to public protocol
- clarifying a specification
- A fun challenge!

Broad techniques

- There are many ways to approach a protocol decode. The main techniques I am going to talk about today are:
  - french cafe technique
  - structured protocol dumping
  - protocol filters
  - poke and fsck
  - trial and error analysis
  - protocol scanners
  - dual server & backtracking
  - Random attack

The French Cafe Technique

- How would you learn french if there were no books and schools?
  - watch people at a French cafe!
  - try ordering a coffee
  - spike someones drink
  - mimic another customer
  - Record everything using a video camera
- Maybe even try assaulting a waiter? You are sure to hear a lot of interesting French phrases!

Test, capture and stare

- The most commonly used approach to decoding a new protocol is to:
  - setup a fast turnaround and simple test
  - setup a method to capture all protocol bytes
  - try an operation
  - try a slightly different operation
  - look for differences
  - Take notes!

Protocol filters

- A protocol filter is a mini proxy that can make changes to the protocol data before it is passed along. Filters can be very powerful analysis tools.
  - global substitutions
  - specific substitutions
  - Negotiation changes
- The two filters that I use regularly are “sockspy” and “smbfilter”

Poke and fsck

- Programmers often print messages for unexpected conditions. The trick is learning how to trigger these messages!
- A good example is decoding a proprietary filesystem:
  - poke a byte in the filesystem
  - run fsck and look for revealing error messages
  - Record your results!

Trial and Error Analysis

- Complex protocols tend to have lots of error values. Which error you get can reveal a lot of information about the packet you sent.
- Taking full advantage of error codes requires several steps:
  - determining what each error code means
  - determine the processing order in the proprietary implementation
  - Write an error driven protocol scanner

Error Mapping

- The proprietary implementation probably has the ability to print error messages given an error code. You just need to find a way to trigger the message you want.
- An example for a file sharing protocol is:
  - modify your server to return error XXX for filename 'test_XXX.dat'
  - ask the proprietary client to access filenames from test_1.dat to test_999.dat
  - Record the messages that the client prints

Proxy Error Mapping

- Proxy error mapping is a technique invented by Andrew Bartlett. It uses a modified Samba domain controller to find the correct mapping between DOS error codes and NT status codes.
  - modify the domain controller to return DOS error code XXX for user test_XXX
  - join a Win2000 client to the domain
  - write a client that tries to authenticate to the Win2000 server as test_1 to test_999
  - Record the NT status codes that the client receives

Protocol scanners

- A protocol scanner uses error codes to guide an automated exploration of a structured protocol. Scanners can be used to find new parts of a protocol or to determine some properties of a protocol operation.
- The main trick with writing a scanner is to know exactly what an error code indicates, and how the proprietary parser will respond to particular error conditions.

Trans2 scanner

- The trans2 calls in SMB are a bit like ioctl(). Trans2 included dozens of sub-commands that implement various types of file and filesystem query.
- The trans2 scanner tries different object types, information levels and data sizes to determine what operations exist and what sizes of data are involved.

Character set methods

- With the recent emphasis on full unicode support in Samba we needed to find techniques for extracting the raw unicode data from Windows servers. The main data we needed was the case equivalence table, which is extracted like this:
  - Loop over all unicode chars
  - for each char create a filename containing that char
  - Open the file and add the char number to the file
- At the end each file contains the list of equivalent characters

Determining inputs

- When first approaching a new encryption or hash algorithm a programmer needs to know what inputs the algorithm takes. For example, does the time or machine name or IP address or username get included in the hash?
  - use a protocol filter
  - selectively destroy each candidate element
  - Observe which elements are needed
- Once the inputs are known then you can start making educated guesses at how they might be combined.

Random attack

- Random attack is one of my favourite techniques, although it is usually used to solve a very specific problem. To have any chance of success you need to:
  - find a reliable test for completion
  - reduce the scope of possible inputs as far as you can
  - develop a fast and reliable reset (protocol or machine)
  - Keep a record of what has failed
- Be aware that you may find a non-standard solution!

Dual server techniques

- The dual-server technique can be an extremely poweful way of fine tuning your knowledge of a protocol. The basic idea is to write a client that connects to a reference server in parallel to your own server. You then need:
  - a random generator for each protocol operation
  - A comparison routine to determine if the two servers gave the same result

Dual server backtracking

- In a dual server test it can happen that a difference may not be visible until thousands of operations later. To determine which operations really caused the problem you can use 'backtracking'
  - record all operations as they are sent
  - when an error occurs replay the operations with some removed
  - if the error is gone then those operations are important
  - Narrow down the important operations using a bisection search

Backtracking in locktest

- Locktest tests the CIFS byte range locking behaviour of a server.
  - two connections to each server
  - two opens per connection
  - generates reopen, lock and unlock
  - generates offset, length and read/write lock
  - Backtracks to generate minimal subset of operations

General advice

- No matter what techniques you use, there are some important general rules:
  - confirm your assumptions!
  - build up a library of captures
  - build both a client and a server
  - start with throwaway 'malleable' code
  - you don't need to be a great programmer
  - You do need a lot of persistance!
- Good luck!