gssMonger

Interoperability Testing Simplified

David L. Christiansen

Windows Core Operating System Division

Security Technology Unit

The Plan

- The Basics of Interoperability Testing
- Introduction to GssMonger
- How GssMonger Aids in Testing
- Simplified Demo of the GssMonger suite
- Future Plans for the Tool

What is Interop Testing?

Trying one implementation of something against another implementation of the same thing.

Different from Protocol Testing

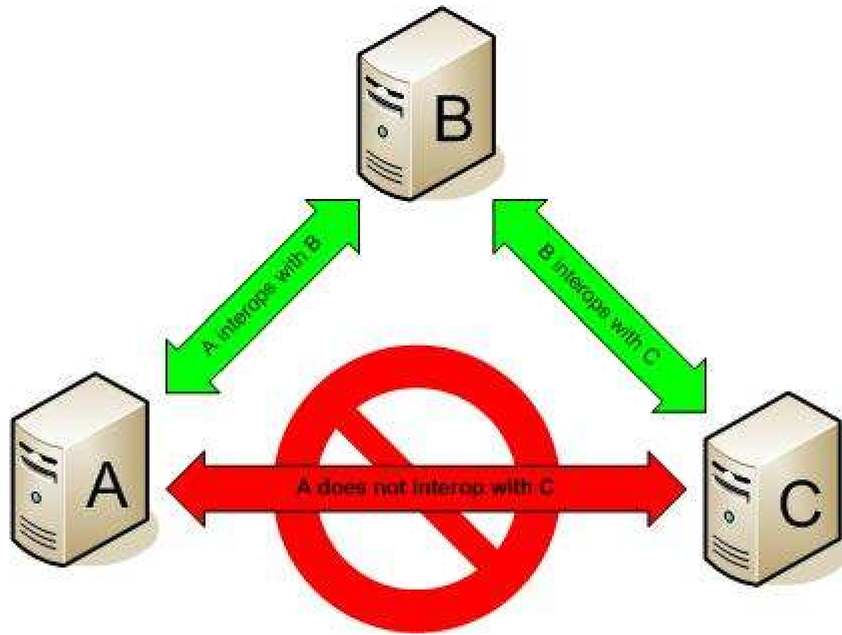
Interop Testing

- Integration Test
- *"Does my stuff work with your stuff?"*
- Requires 2+ implementations.
- Harder to debug
- Easy to measure
- Important to System Administrators
- Important to Implementers

Protocol Testing

- Targeted Test
- *"Does my stuff look like the standard?"*
- Requires only one implementation
- Viewpoint-Sensitive
- Easy(er) to debug
- Hard to measure
- Important to Implementers

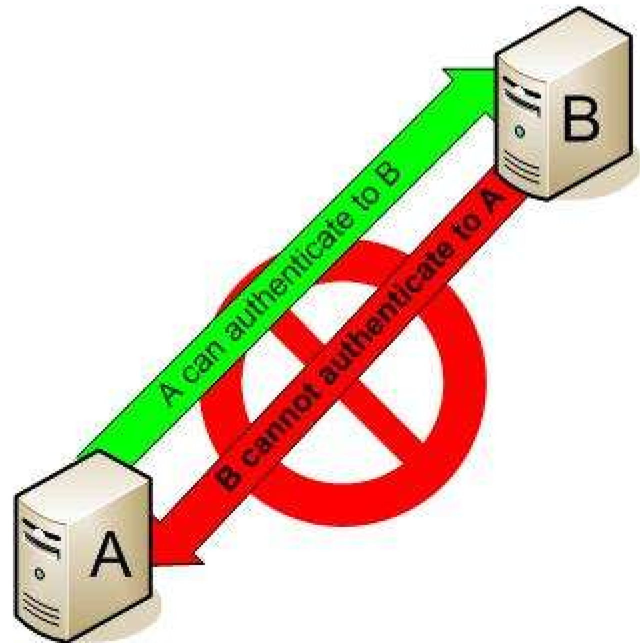
Interop is not Transitive



- A and B can interop with C, but not each other.
- Testing against the reference implementation is not enough!

Interop is not Reflexive

- Probably obvious, but it bears repeating.
- It's an easy (but bogus) assumption to make…
  - "If I can logon at the Windows machine, *obviously* I could do so on a unix machine…"



Why Test for Interoperability?

- As with any bug, it is usually much easier, cheaper, and faster to fix it *before* release than *afterward*.
- Interop bugs tend to block deployments
- Interop bugs affect customers in a meaningful, tangible way.

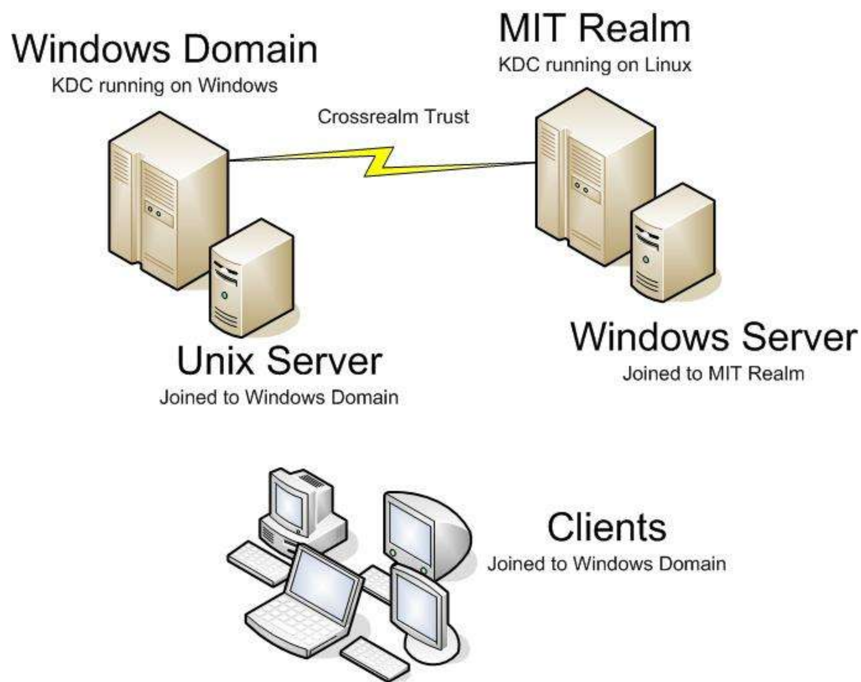
*“What, why **doesn’t** xlock work with a Microsoft KDC? Who do I report **that** bug to?”*

-Hypothetical Customer

Challenges of Interop Testing

- Expensive.
  - Requires other implementations (and understanding of them)
- Tedious
  - Tests must be run against all important platforms
  - Combinatorics are boring
  - Test matrix grows exponentially with implementations
- Philosophically and Politically Taxing
  - Requires you to define “works” for your implementation.
  - Resolving bugs sometimes requires negotiation between implementers.

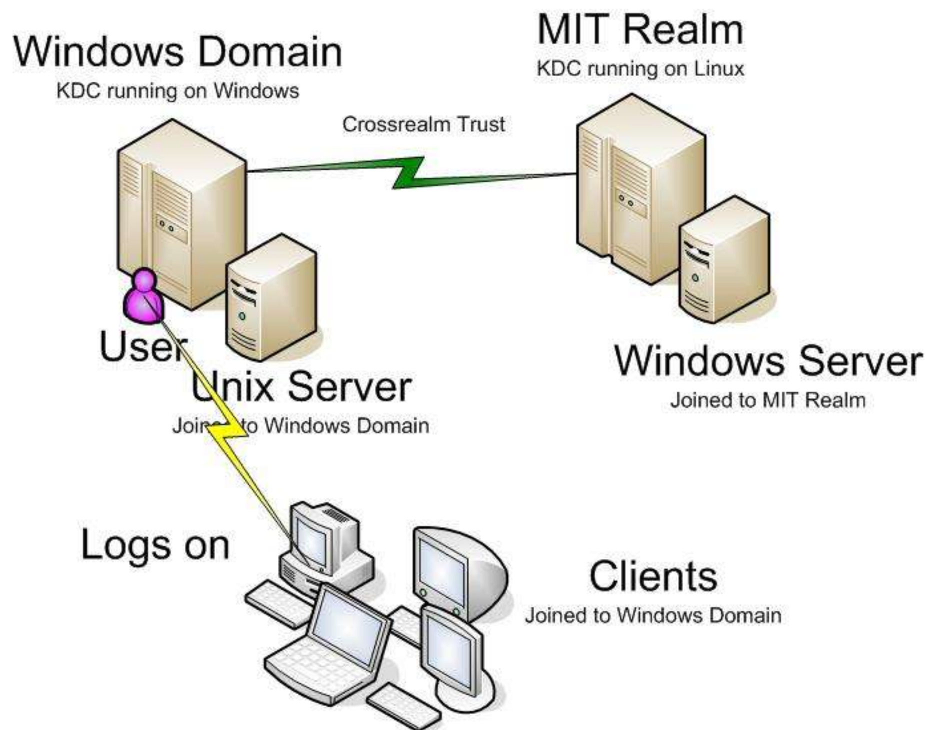
Example

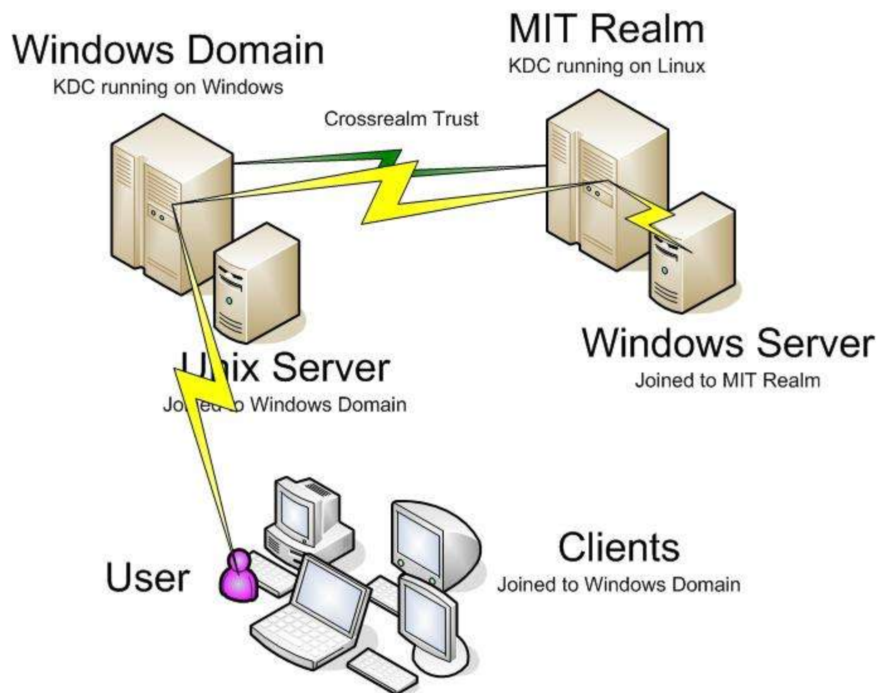


- Postulate two realms
  - An MIT Realm
  - A Windows Domain
- Each realm has one server
- The Windows domain has several clients
- All in all, a very typical heterogenous deployment.

A user in the Windows domain logs on to a client machine

(right)



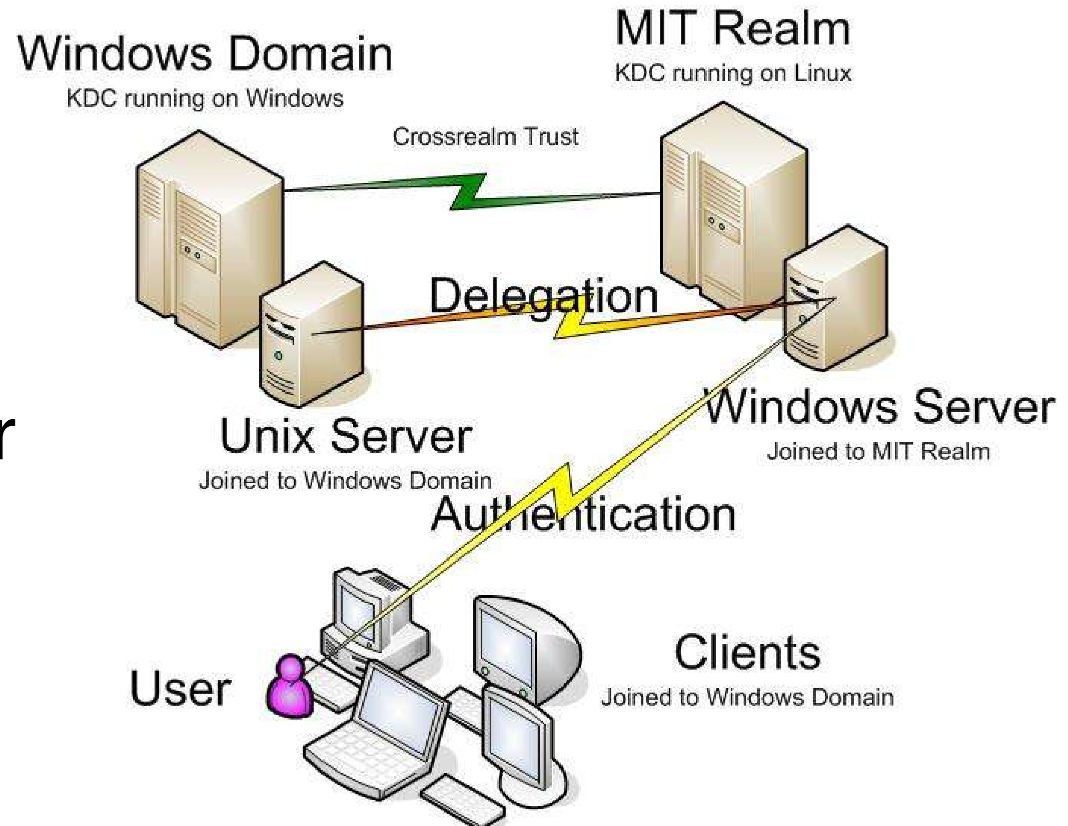


…then authenticates to a server in the MIT Realm

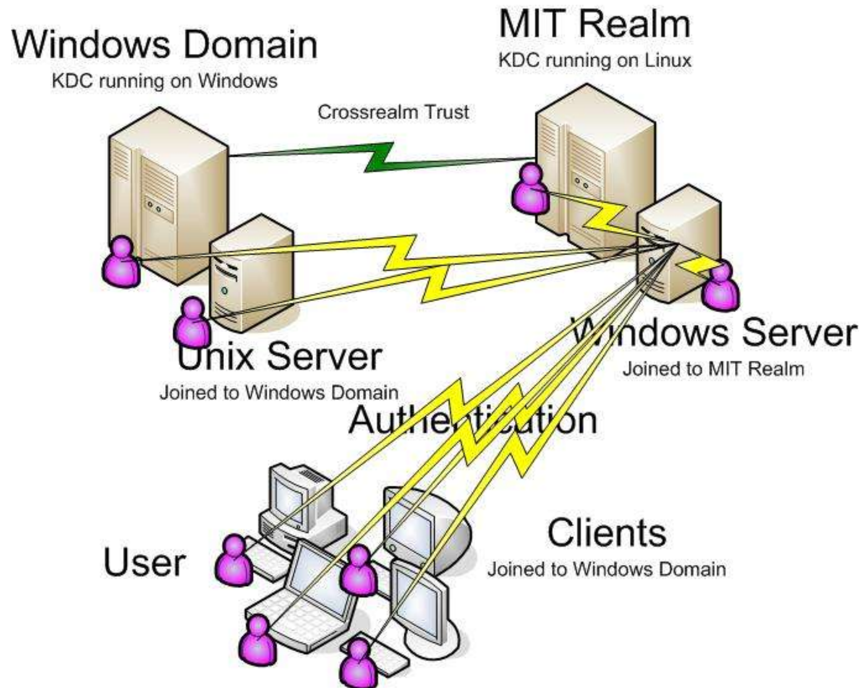
(left)

Easy, Right?

- Now, imagine that the server is a web interface to a database
- It now has to delegate to another server

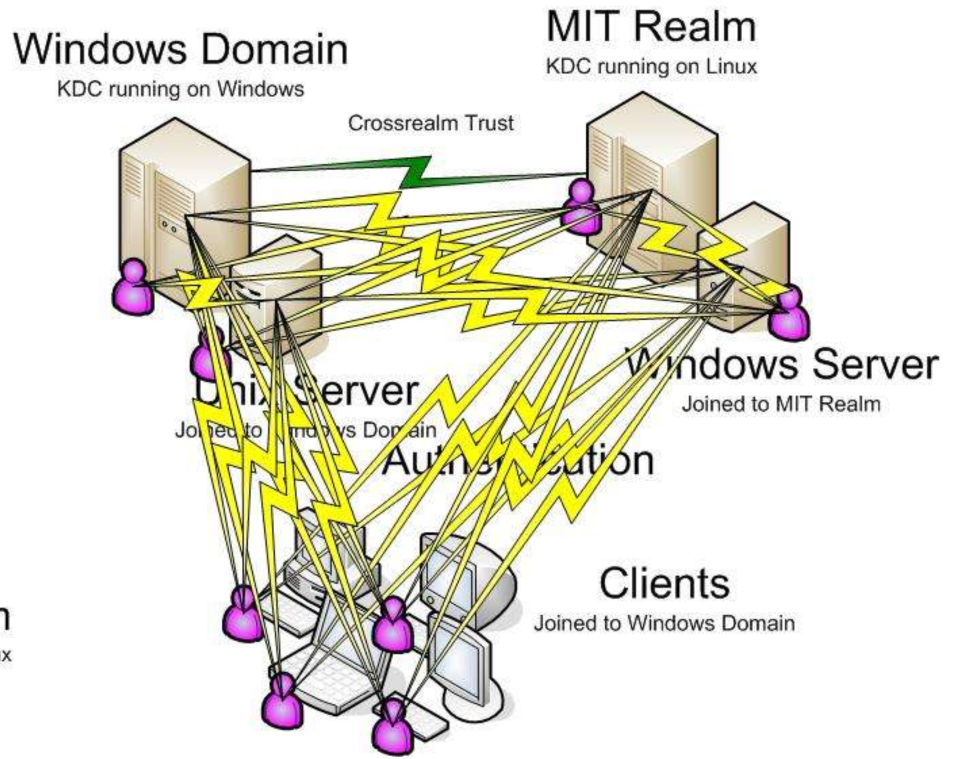


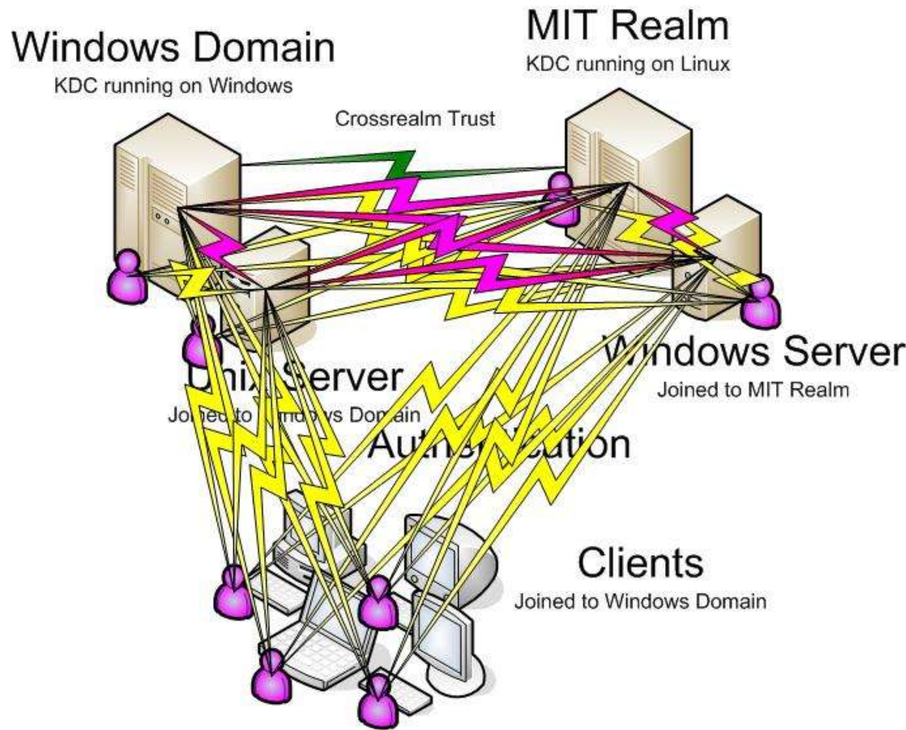
The Sysadmin's Burden



- Of course, you have to test all your client architectures too
  - since each can have its own bugs…
- And each server is also a client.
- You also want to test with principals in each realm…

- But if you're *implementing*, you want *all* the machines to interoperate.
  - Else, you have bugs that *someone* will find…





- And don't forget delegation…

Too Many Variations!

- I doubt that this level of analysis is being performed today, at least using the publicly available suites.
- All 8x4x4=128 variations above would be difficult to perform with gss-client and gss-server.
  - Add in the additional complexity of logging on to four clients in the unix realm (8x8x4=256 variations)
  - Imagine as an implementer testing all 64 combinations of gssapi flags in conjunction with the above (thousands of variations).

gssMonger to the Rescue!

How does gssMonger Help?

- Performs baseline interoperability tests
  - Against self (regression)
  - Against others (interop)
- Automation
  - Vastly reduces the tedium of running the same application in so many modes (kinit, gss-client… repeat, repeat, repeat…)
- Comprehensive
  - Tests lots of different features in various combinations.
- Disambiguating:
  - No philosophy– measurable interop statistics.
  - If gssmonger fails, it will fail for customers too
  - It “works” if the test succeeds.
- Diagnostics
  - Provides surface errors as exposed by the implementation.
  - Does not hide errors behind other layers

What does gssMonger do?

- Evaluates interoperability matrix using:
  - Context negotiation
  - Session protection (wrap, encrypt, sign)
  - Password Change
  - Password Set
  - Delegation
- Provides single interface point (the master) that can control the entire testbed.

What is gssMonger?

- Master/Slave testing framework
- Designed to test context negotiation with MIT Kerberos in the Win2000 timeframe
  - The gss-sample apps just weren't enough.
  - Abstraction ported to other platforms (such as Heimdal) over the years.
- Can also perform baseline gssapi regression (functional) testing
  - Has found non-interop errors in various implementations (MS, MIT, Heimdal).
- Extensible to new classes of tests
- Source Code Available

Two Primary Components

gssMaster

- Oversees tests
- Does not perform tests
- Collects diagnostic data from Maggots.
- Produces human-readable output
- Currently runs only on Windows.

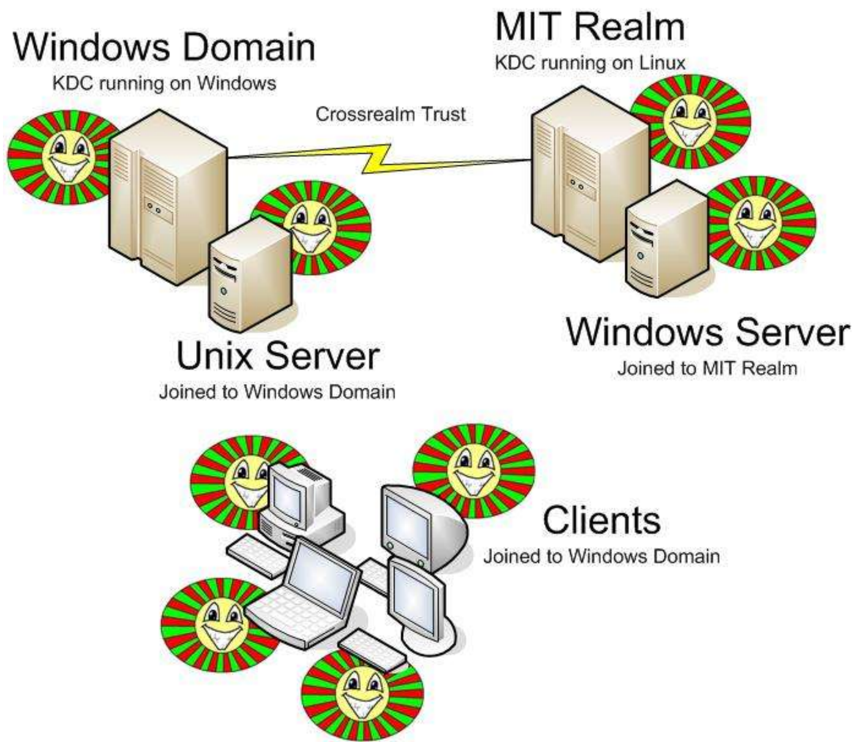
gssMaggot

- Runs tests by performing tasks as directed by Master:
  - Authenticate to so-and-so
  - Change XYZ's password
- Knows the underlying Kerberos implementation
- Portable
- Talks only to the Master.

A Specific Example

How gssMonger simplifies testing in the previously described bed

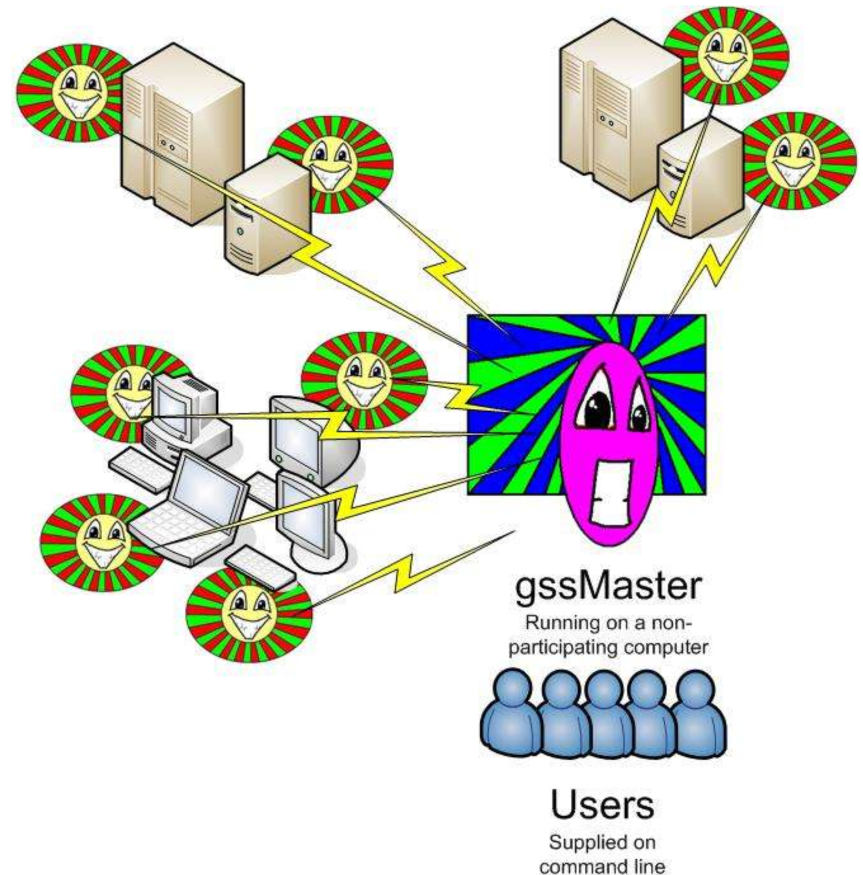
1. Install gssMaggot everywhere



- Every machine that you might authenticate to or from should run gssMaggot.
- Tell the maggot whether the machine can be a server or not.
- Maggots require very little configuration.

2. Run gssMaster somewhere

- Needs a list of principals that can be used in testing
- Needs to know where the maggots are.
- gssMaster will then coordinate testing using the maggots
- All user interaction is done by the Master.



3. Analyze Output

Summary

gssMonger.log.xml started at 16:38:38 09/01/2005, finished at 16:39:41 09/01/2005. Time elapsed in gssMonger.log.xml: one Minute, 3 Seconds.

There were 20 test blocks.

20 (100%) blocks attempted, 20 (100%) successful.

Breakdown by Levels

Level	%	count	total	comments
Pass	100%	20	20	[Attempted,Successful]

Breakdown by Blocks

FlagCombos	Pass	foo@DOMSEC6MX09	Pass	SEC6MX09:904	Pass	SEC6MX09:904	Pass	0x00	Pass	
								MA 0x02	Pass	
								CN 0x10	Pass	C:Encrypt -- Norı
										C:Wrap -- Norma
										C:Sign -- Normal
										S:Encrypt -- Norı
										S:Wrap -- Norma
										S:Sign -- Normal
								MA CN 0x12	Pass	C:Encrypt -- Norı
										C:Wrap -- Norma
										C:Sign -- Normal
										S:Encrypt -- Norı
										S:Wrap -- Norma
										S:Sign -- Normal

- Hopefully, you'll see 100% success.
- To an Admin, this means a correctly configured setup.
- To an Implementer, it means the scenario can be setup interoperably (because you did it).

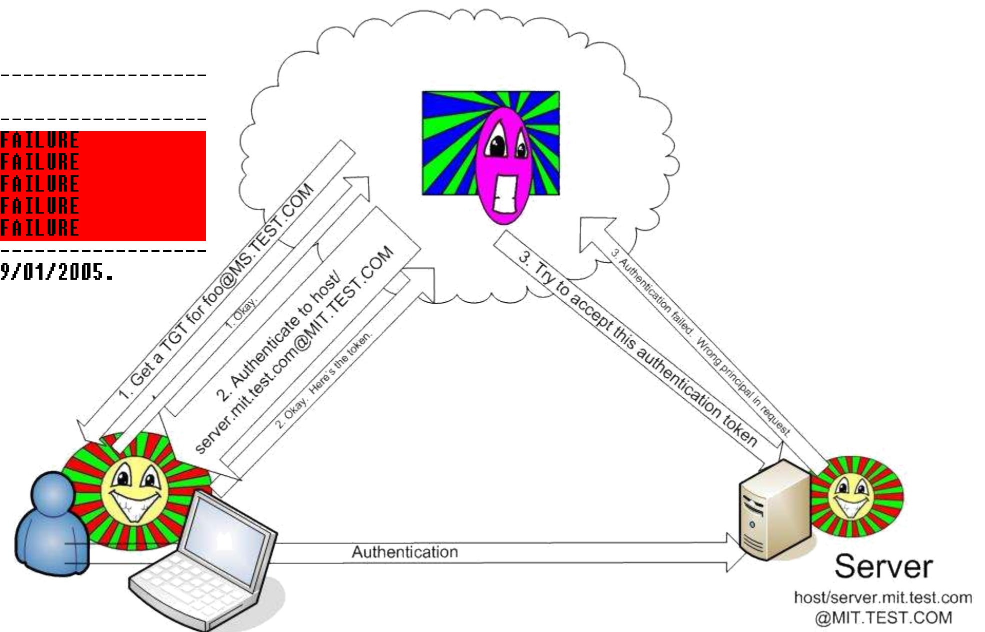
One Variation

- gssMaster tells a Maggot to authenticate to one of the server Maggots using a client principal.
- The Master reports that (in this case) authentication failed.

```
--------------------------------------------------------------------------------
                                   ezLog report
--------------------------------------------------------------------------------
FlagCombos | foo@DOMSEC6MX09 | SEC6MX09:999 | SEC6MX09:999 | MA   0x02 | FAILURE
           |                 |              | SEC6MX09:999 |           | FAILURE
           |                 | SEC6MX09:999 |              |           | FAILURE
           | foo@DOMSEC6MX09 |              |              |           | FAILURE
FlagCombos |                 |              |              |           | FAILURE
--------------------------------------------------------------------------------
gssMonger.log.xml started at 16:47:53 09/01/2005, finished at 16:47:57 09/01/2005.
Time elapsed in gssMonger.log.xml: 3 Seconds.

There were 5 test blocks.
5 (100%) blocks attempted, none (0%) successful.

            FAILURE      100% (5/5) [Attempted]
```



Full Regression Run

Summary

gssMonger.log.xml started at 16:38:38 09/01/2005, finished at 16:39:41 09/01/2005. Time elapsed in gssMonger.log.xml: one Minute, 3 Seconds.

There were 20 test blocks.

20 (100%) blocks attempted, 20 (100%) successful.

Breakdown by Levels

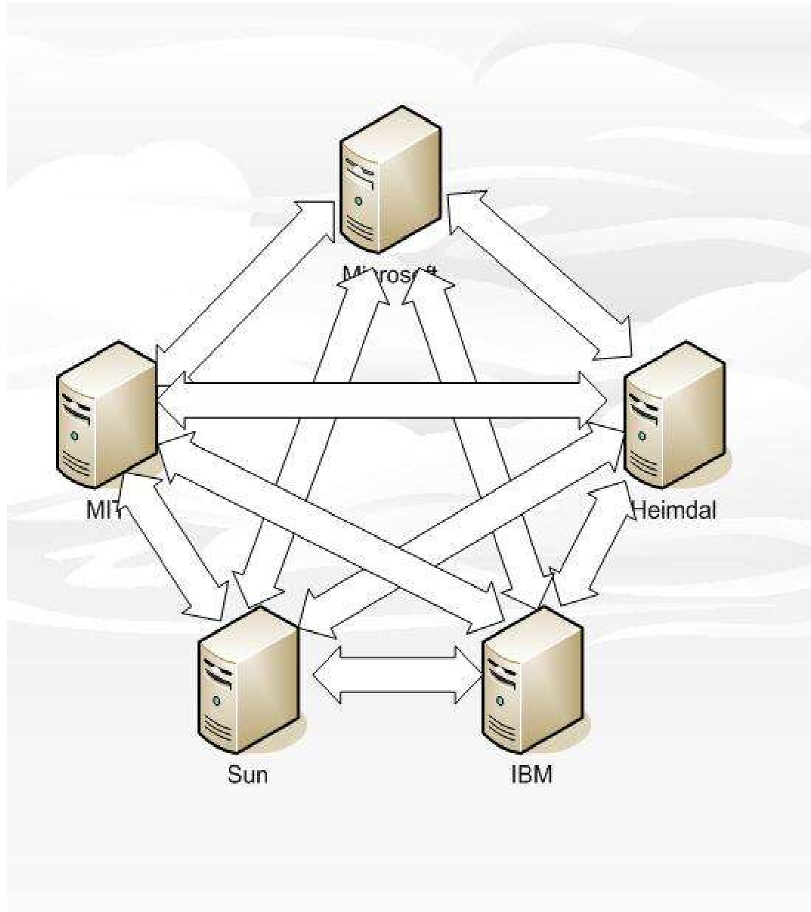
Level	%	count	total	comments
Pass	100%	20	20	[Attempted,Successful]

- Just as in the single variation case, gssMaster produces a report describing what percentage of pairings actually interoperated.
- Anything listed as a failure (previous slide) is a scenario that verifiably doesn't work.

DEMO

Going Forward

The Dream…



- I had hoped we could create a standard bed of machines that we could test against over the internet
- This proved Hard.
  - Schedules
  - Priorities
  - Infrastructures
- It’s still the dream ☺

Lessons Learned

- One team's test time is another team's crunch time
- Testing multiple prerelease platforms together is not terribly productive.

Most Importantly:

- No amount of cool test software can change the need to actually *run* it.
  - One of the reasons interop summits are productive

Future Enhancement

- There are places that gssMonger can't go right now, but could *and should* to further the goal of interoperability in the future.
  - PKINIT: in progress, needs community help
  - Other protocols (we have NTLM, some SPNEGO…)
- There are always bugs, of course…
- What would benefit the community?

Call to Action

- Please please please please please run this tool against your implementation.
  - First run it against yourself (regression).
  - If your stuff works, run it with other implementations in the mix (actual interop)
- We do run this test extensively inside Microsoft
  - But we can't keep up on new releases of other implementations.
  - If everyone tests his/her latest bits against the other major *released* implementations, the major bugs will be shaken out.

In Closing

- Interop Testing is important but not easy
- gssMonger can manage and greatly simplify this arduous task
- If everyone does a little of it, the job gets quite a bit easier
  - Please run gssmonger.

Questions?