Windows Kernel Fuzzing for Intermediate Learners

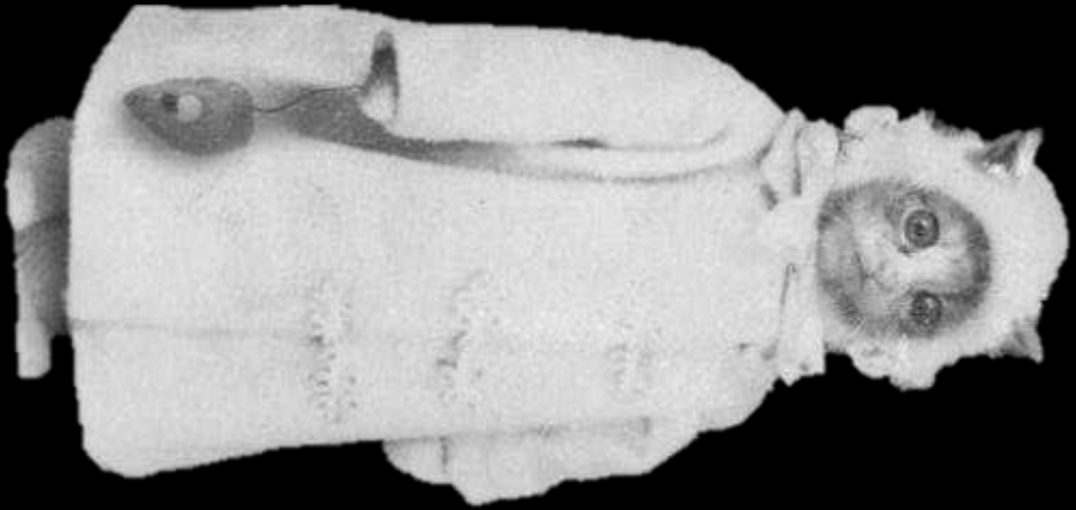
Ben Nagy

COSEINC®

Solid Security. Verified.

PSA WARNINGS

- ALLERGY: Some Recycled Material
- SPOILER: Not Really About Kernel Fuzzing
- TRIGGER: Neckbeards



About Me:

- Not oldsk00l. Just old.
- ~ 11 weeks kernel experience
- ~ 8 years fuzzing experience
- ~ 25 years nerding experience

- Hate all Technology
- Certified Windows Internals Expert!

Disclaimer:
I am aware of the prevailing opinion that fuzzing talks without bugs suck, by definition. I do not have any bugs. Even if I did have bugs, I wouldn't tell you. There are no bugs.

COSEINC®
Solid Security.Verified.

EXIT
SyScan
Certificate of Achievement
Windows Internals for Reverse Engineers

COSEINC®

Solid Security. Verified.

ALL LIES!

Not fuzzing ALPC - Fuzzing *with* ALPC

ALL LIES!

Not kernel fuzzing - new attack surface for userland

COSEINC
Solid Security.Verified.

ALL LIES!

… but we need to understand the kernel first

COSEINC

Solid Security.Verified.

Fuzzing Made Simple

- Select a Good Target
- Acquire Essential Knowledge
- Apply Fuzzing Canon
  - How do we Deliver
  - How do we Instrument
  - How do we Generate
  - How does that Scale

Phase I - Target Selection

Target: ALPC

Why ALPC?

- New
- Tricky
- Undocumented
- Everywhere

What Bug Classes?

- Privesc to SYSTEM(+) from anywhere
- Memory Helpers
  - Fill memory
  - Disclose?
- DoS
- “Jackpot” bug?

ALPC What Do?

- Interprocess Communication
- New in Vista+
- Low Level
- Sync / Async, Fast, Awesome

www.syscan.org/index.php/download/get/d596c7dc486175148fc038387dc80be2/SyScan2014_AlexIonescu_AllabouttheRPCLRPCALPCandLPCinyourPC.zip

ALPC What Do?

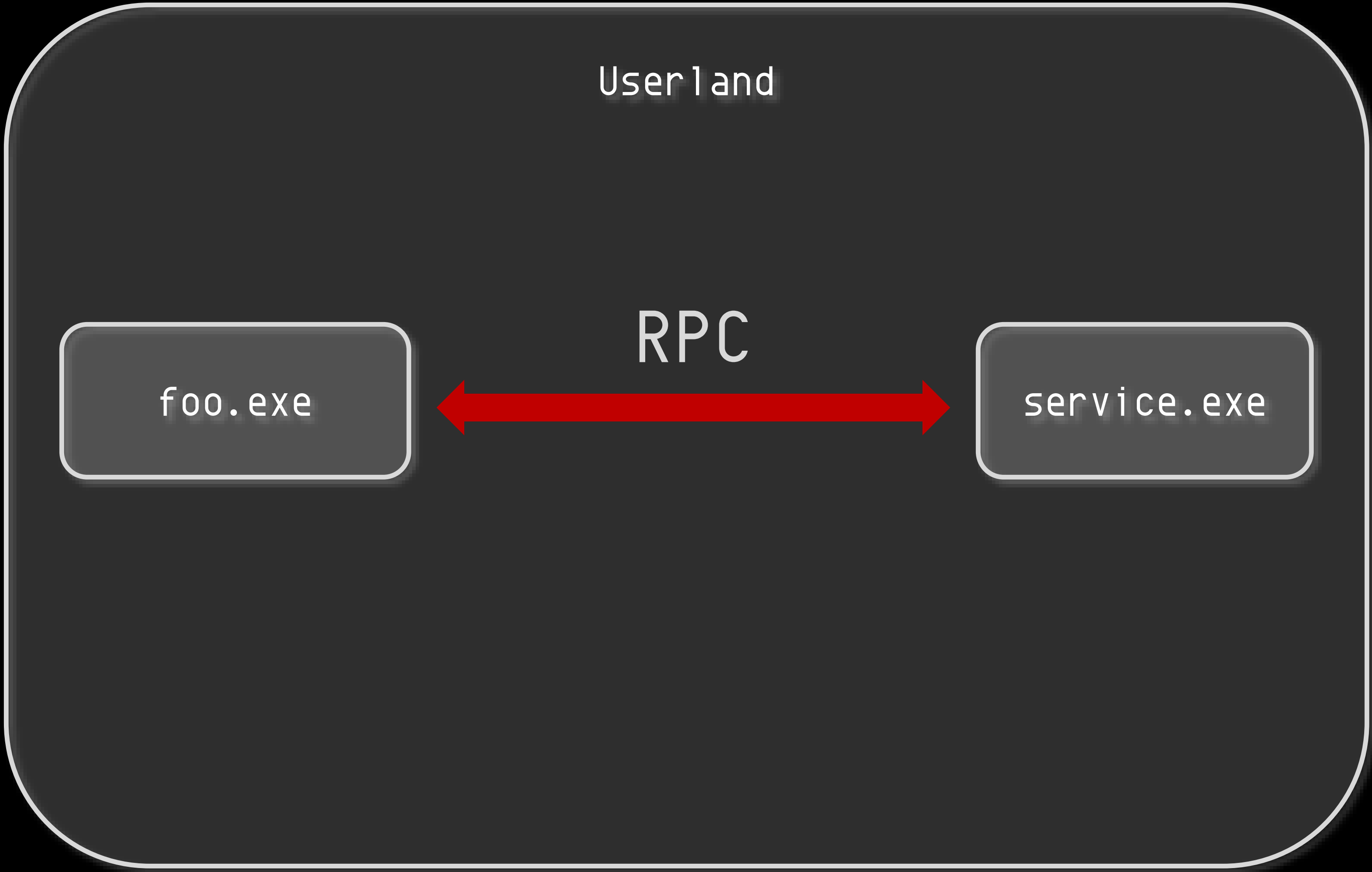
- Shared Memory Views
- IO Completion Ports
- Lots of security, enforced by the kernel
- TOCTOU Safe

www.syscan.org/index.php/download/get/d596c7dc486175148fc038387dc80be2/SyScan2014_AlexIonescu_AllabouttheRPCLRPCALPCandLPCinyourPC.zip

ALPC What Do?

- RPC / RPC-DCOM run on it
- Can also be used directly
- Imagine it like a network

www.syscan.org/index.php/download/get/d596c7dc486175148fc038387dc80be2/SyScan2014_AlexIonescu_AllabouttheRPCLRPCALPCandLPCinyourPC.zip

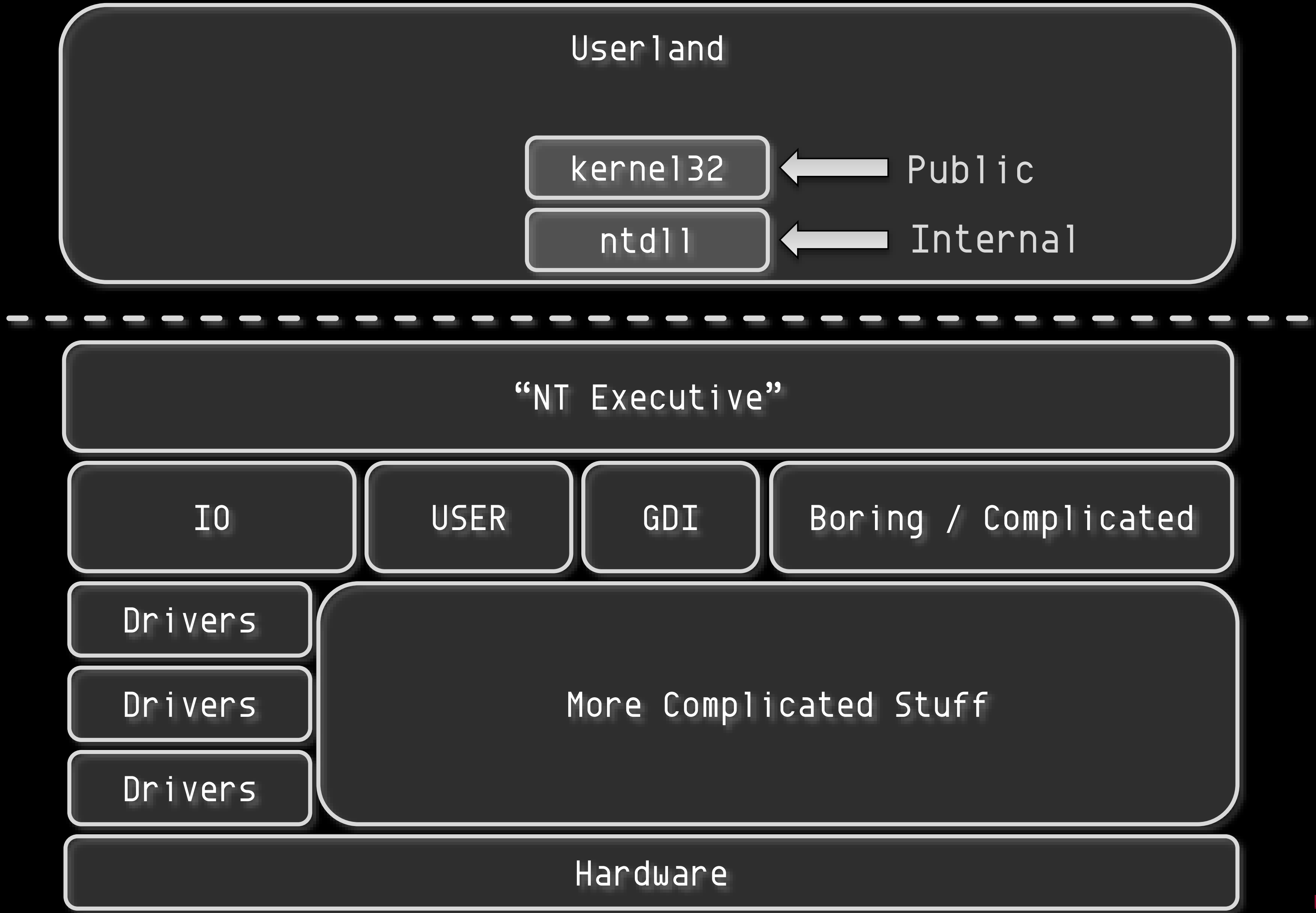

Userland
RPC
foo.exe
service.exe

© Sven Micklish

(Ohai I'm Barry)

© Sven Micklish

Kernel Recap



Userland

kernel32 ← Public

ntdll ← Internal

“NT Executive”

IO

USER

GDI

Boring / Complicated

Drivers

Drivers

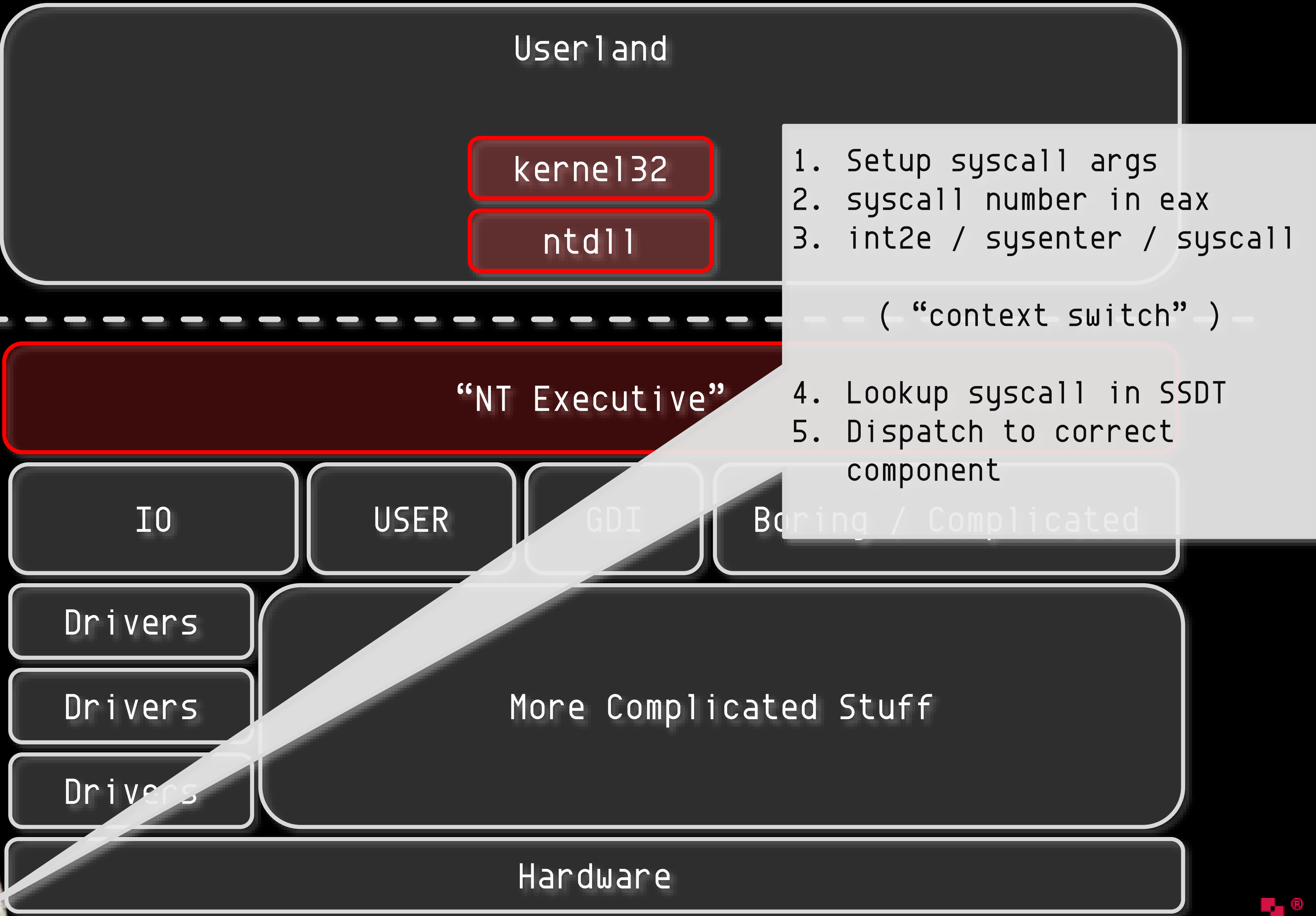
Drivers

More Complicated Stuff

Hardware

COSEINC®

Solid Security.Verified.



1. Setup syscall args
2. syscall number in eax
3. int2e / sysenter / syscall

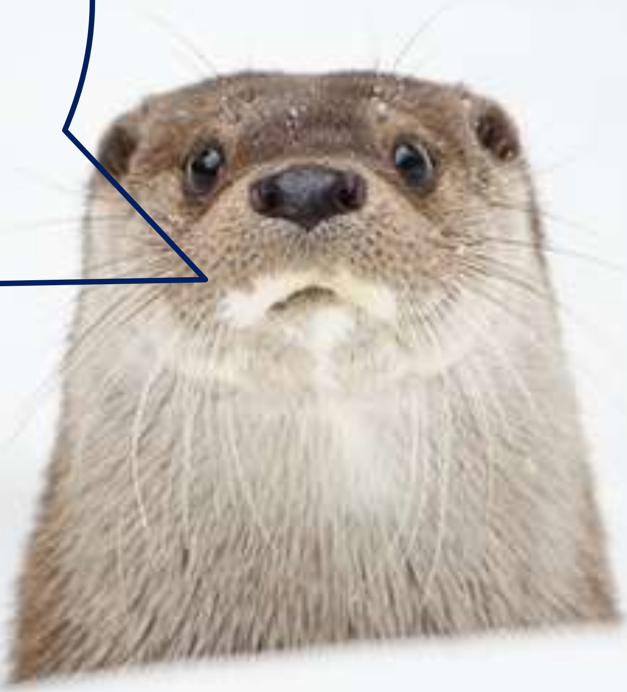
(“context switch”)

4. Lookup syscall in SSDT
5. Dispatch to correct component

COSEINC® Solid Security. Verified.

Kernel has Objects.

There are many kinds.

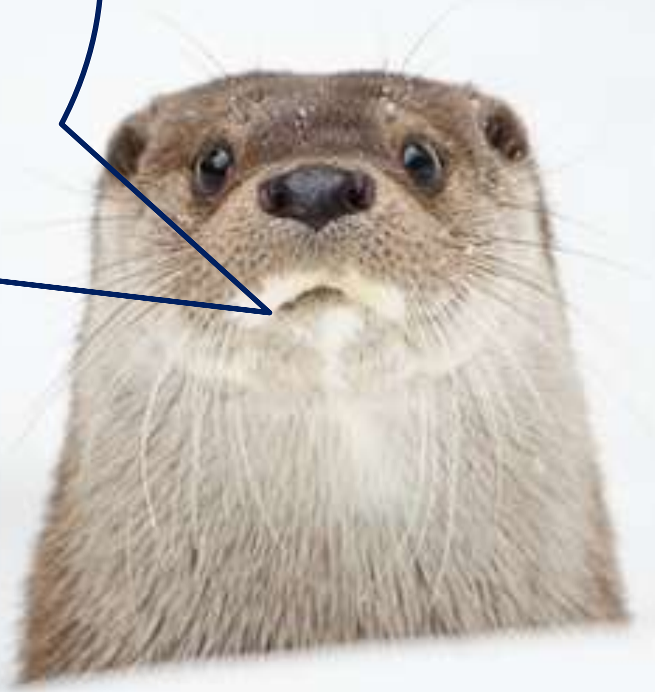


© Sven Micklish

They go in Directories

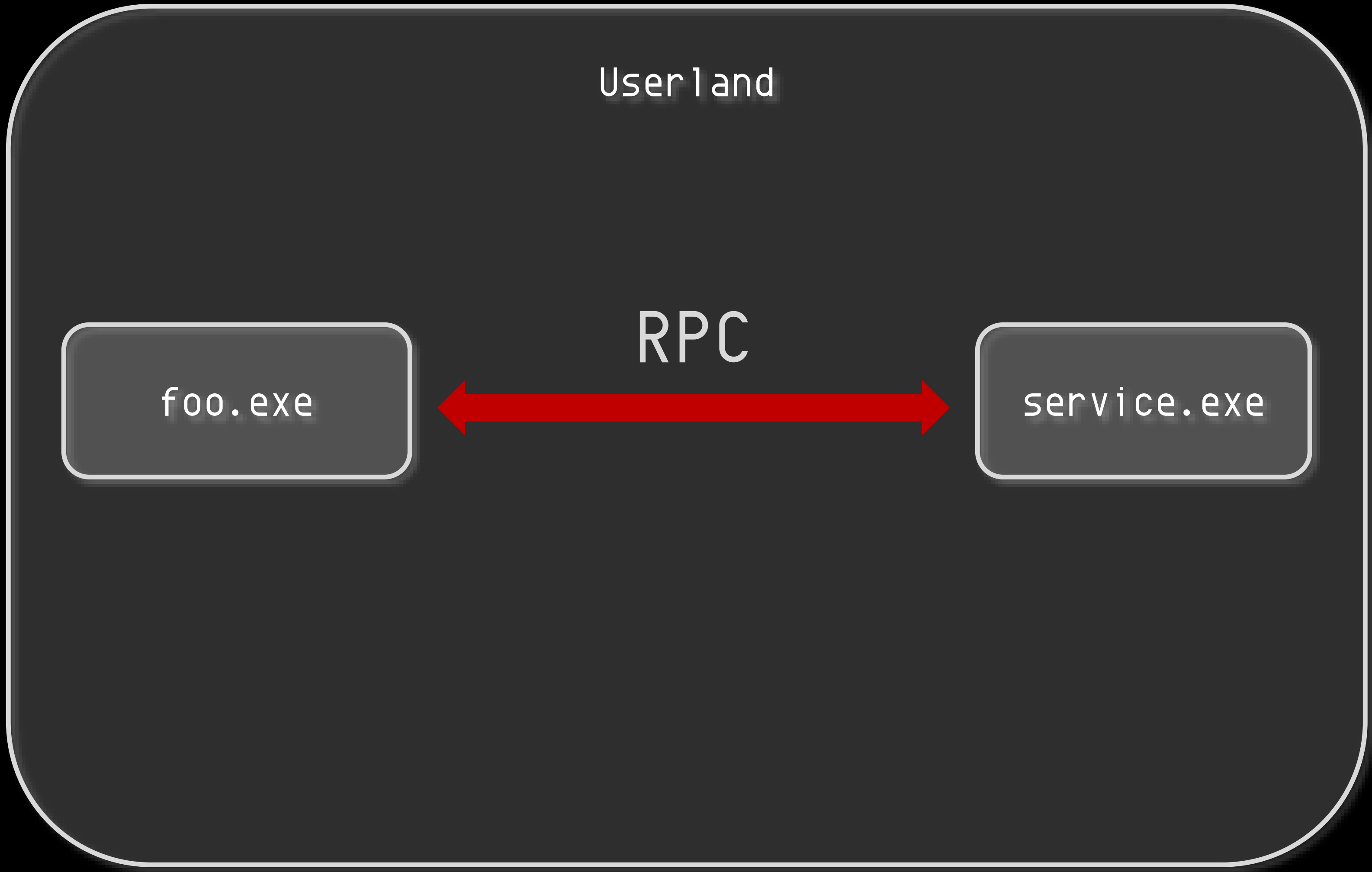
Object Manager manages them

(duh.)



© Sven Micklish

… where were we?

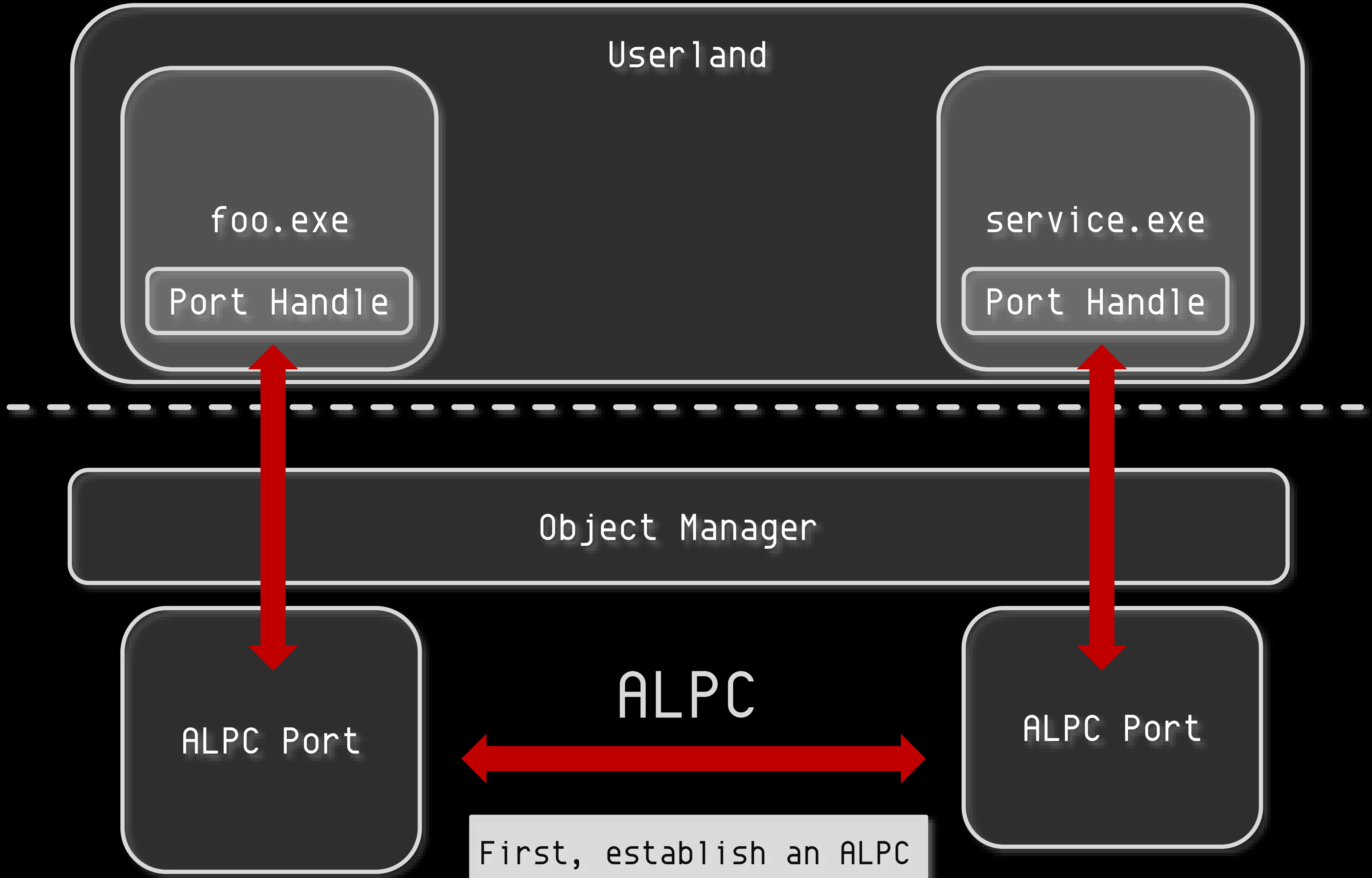


Userland

RPC

foo.exe

service.exe



Userland

foo.exe

Port Handle

service.exe

Port Handle

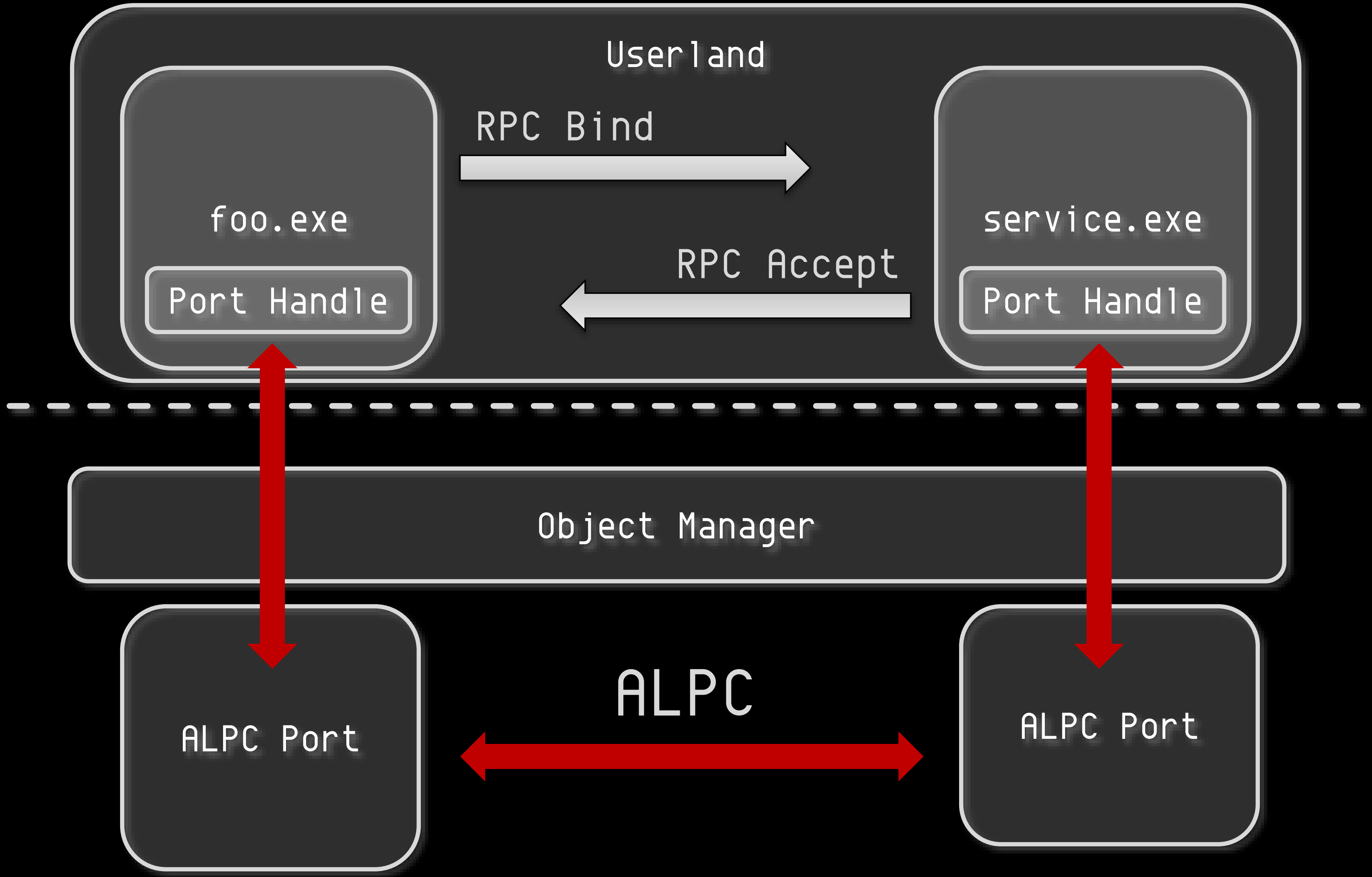
Object Manager

ALPC Port

ALPC

ALPC Port

First, establish an ALPC connection…



Userland

foo.exe

Port Handle

RPC Bind

RPC Accept

service.exe

Port Handle

Object Manager

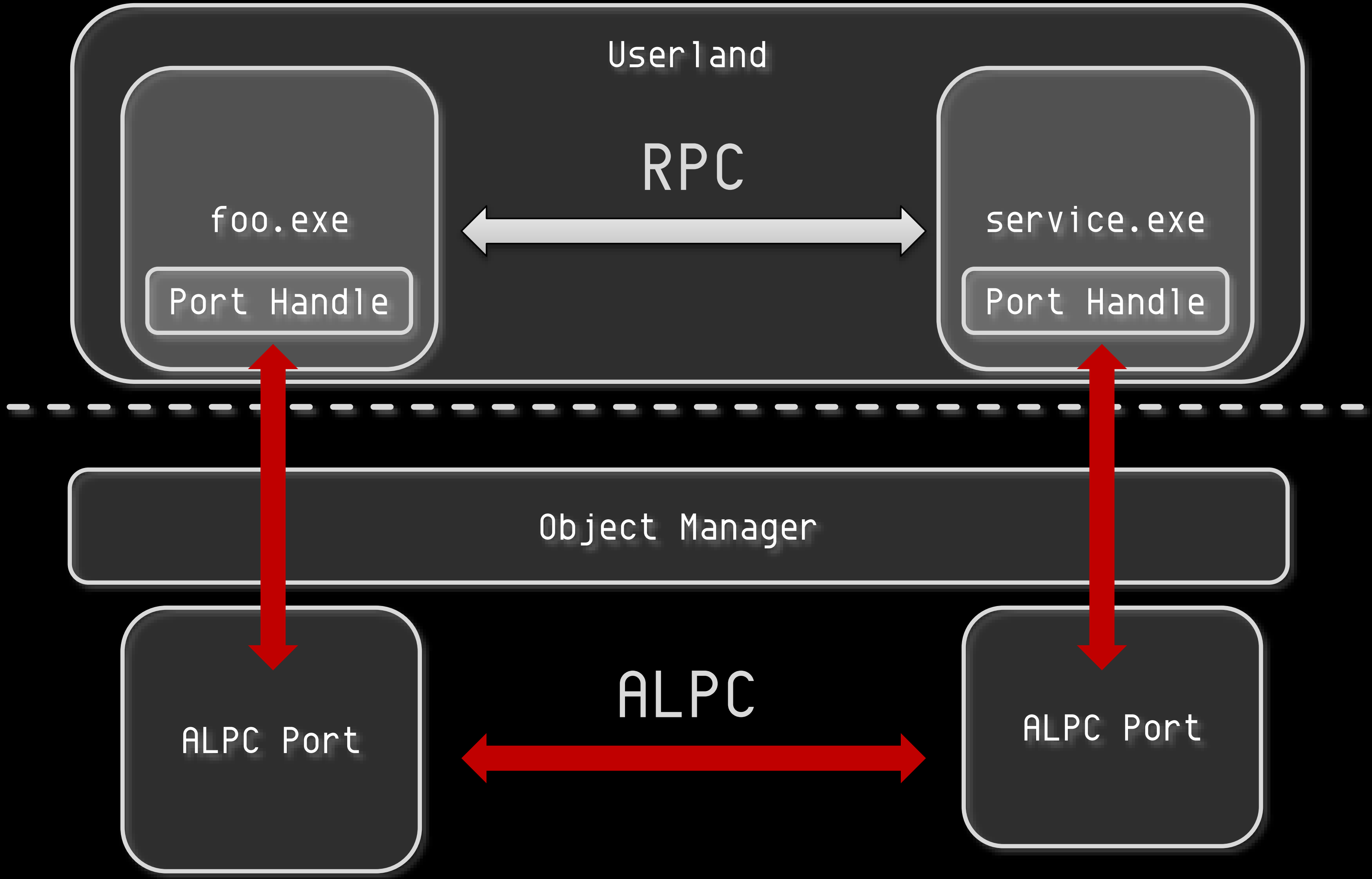
ALPC Port

ALPC

ALPC Port

COSEINC®

Solid Security.Verified.



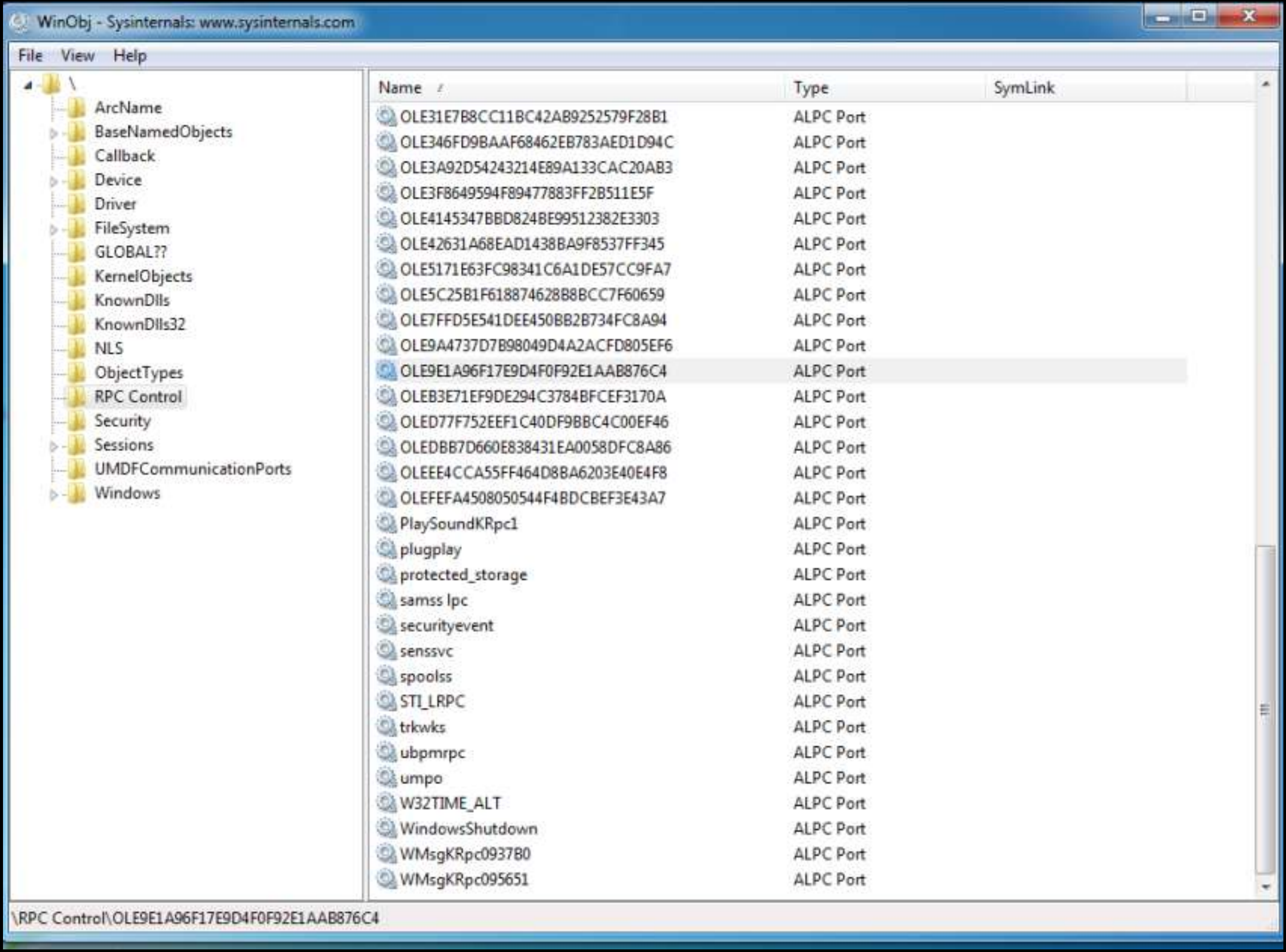
COSEINC®
Solid Security.Verified.

Phase II - Acquire Knowledge

ALPC Surface

ALPC Attack Surface

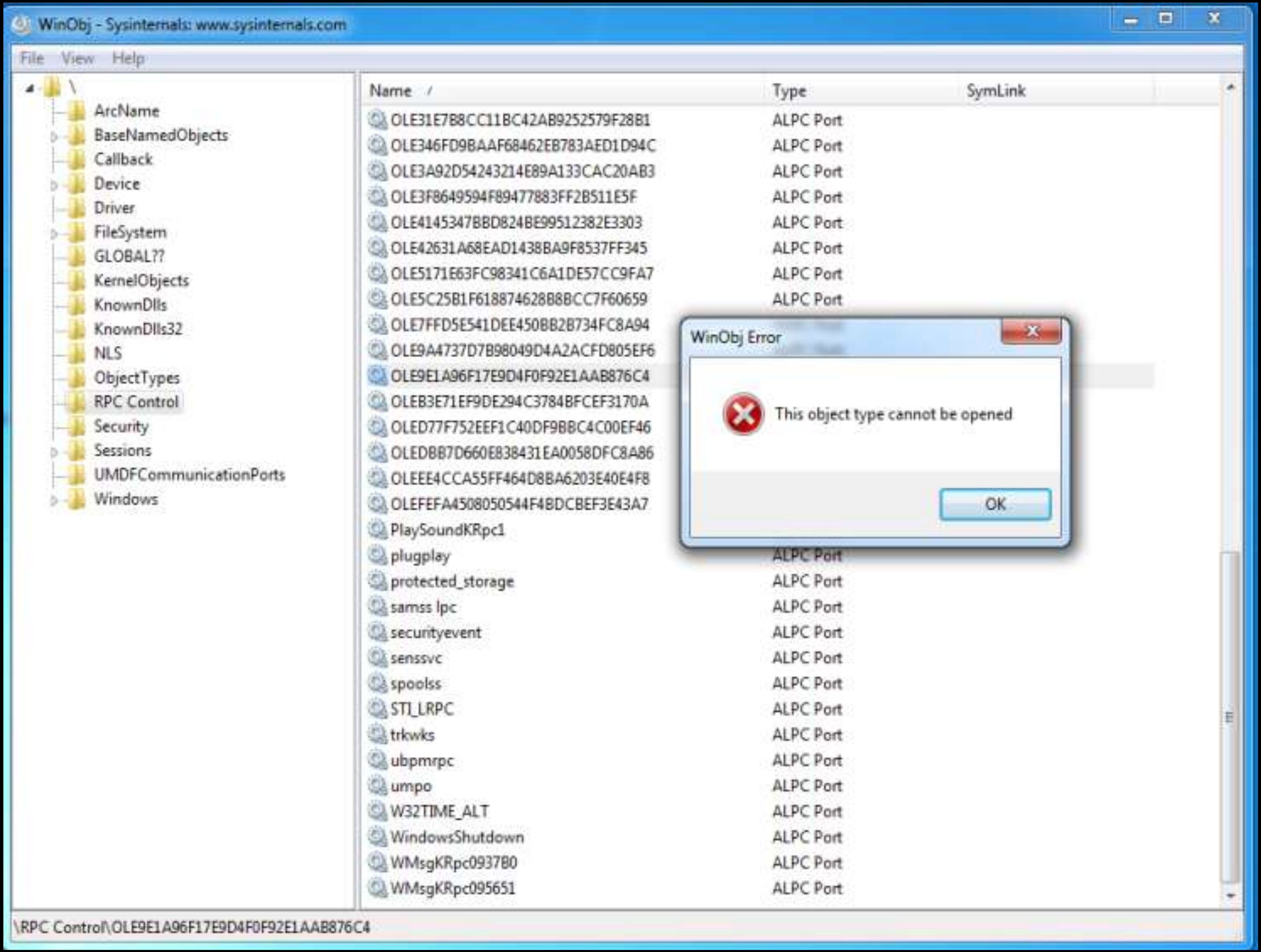
- Who talks to whom?
- Which processes have open ports?



- \
  - ArcName
  - BaseNamedObjects
  - Callback
  - Device
  - Driver
  - FileSystem
  - GLOBAL??
  - KernelObjects
  - KnownDlls
  - KnownDlls32
  - NLS
  - ObjectTypes
  - RPC Control
  - Security
  - Sessions
  - UMDFCommunicationPorts
  - Windows

Name	Type	SymLink
OLE31E7B8CC11BC42AB9252579F28B1	ALPC Port	
OLE346FD9BAAF68462EB783AED1D94C	ALPC Port	
OLE3A92D54243214E89A133CAC20AB3	ALPC Port	
OLE3F8649594F89477883FF2B511E5F	ALPC Port	
OLE4145347BBD824BE99512382E3303	ALPC Port	
OLE42631A68EAD1438BA9F8537FF345	ALPC Port	
OLE5171E63FC98341C6A1DE57CC9FA7	ALPC Port	
OLE5C25B1F618874628B8BCC7F60659	ALPC Port	
OLE7FFD5E541DEE450BB2B734FC8A94	ALPC Port	
OLE9A4737D7B98049D4A2ACFD805EF6	ALPC Port	
OLE9E1A96F17E9D4F0F92E1AAB876C4	ALPC Port	
OLEB3E71EF9DE294C3784BFCEF3170A	ALPC Port	
OLED77F752EEF1C40DF9BBC4C00EF46	ALPC Port	
OLEDBB7D660E838431EA0058DFC8A86	ALPC Port	
OLEEE4CCA55FF464D8BA6203E40E4F8	ALPC Port	
OLEFEFA4508050544F4BDCBEF3E43A7	ALPC Port	
PlaySoundKRpc1	ALPC Port	
plugplay	ALPC Port	
protected_storage	ALPC Port	
samss lpc	ALPC Port	
securityevent	ALPC Port	
senssvc	ALPC Port	
spoolss	ALPC Port	
STI_LRPC	ALPC Port	
trkwks	ALPC Port	
ubpmrpc	ALPC Port	
umpo	ALPC Port	
W32TIME_ALT	ALPC Port	
WindowsShutdown	ALPC Port	
WMsgKRpc0937B0	ALPC Port	
WMsgKRpc095651	ALPC Port	

\RPC Control\OLE9E1A96F17E9D4F0F92E1AAB876C4



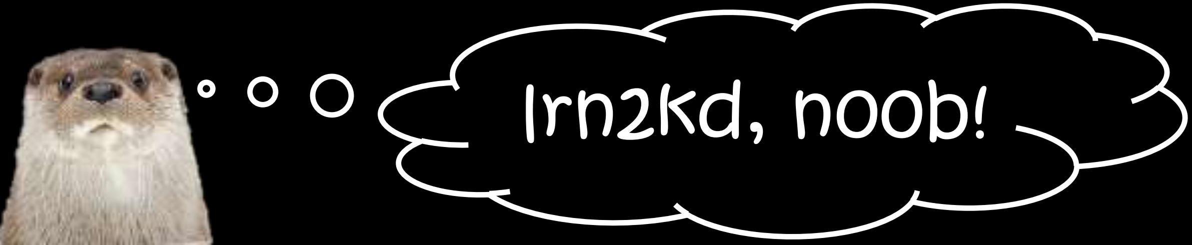
WinObj - Sysinternals: www.sysinternals.com

Name	Type	SymLink
OLE31E7B8CC11BC42AB9252579F28B1	ALPC Port	
OLE346FD9BAAF68462EB783AED1D94C	ALPC Port	
OLE3A92D54243214E89A133CAC20AB3	ALPC Port	
OLE3F8649594F89477883FF2B511E5F	ALPC Port	
OLE4145347BBD824BE99512382E3303	ALPC Port	
OLE42631A68EAD1438BA9F8537FF345	ALPC Port	
OLE5171E63FC98341C6A1DE57CC9FA7	ALPC Port	
OLE5C25B1F618874628B8BCC7F60659	ALPC Port	
OLE7FFD5E541DEE450BB2B734FC8A94		
OLE9A4737D7B98049D4A2ACFD805EF6		
OLE9E1A96F17E9D4F0F92E1AAB876C4		
OLEB3E71EF9DE294C3784BFCEF3170A		
OLED77F752EEF1C40DF9BBC4C00EF46		
OLEDBB7D660E838431EA0058DFC8A86		
OLEEE4CCA55FF464D8BA6203E40E4F8		
OLEFEFA4508050544F4BDCBEF3E43A7		
PlaySoundKRpc1		
plugplay	ALPC Port	
protected_storage	ALPC Port	
samss lpc	ALPC Port	
securityevent	ALPC Port	
senssvc	ALPC Port	
spoolss	ALPC Port	
STI_LRPC	ALPC Port	
trkwks	ALPC Port	
ubpmrpc	ALPC Port	
umpo	ALPC Port	
W32TIME_ALT	ALPC Port	
WindowsShutdown	ALPC Port	
WMsgKRpc093780	ALPC Port	
WMsgKRpc095651	ALPC Port	

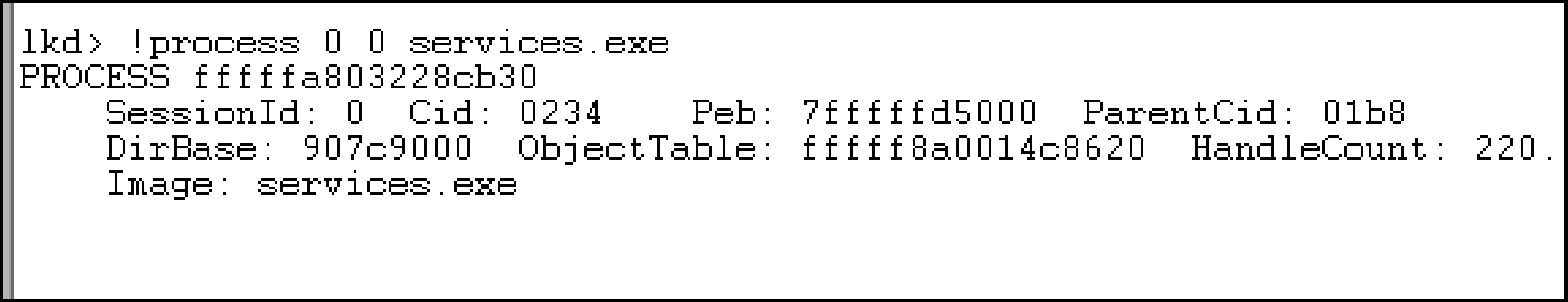
WinObj Error

This object type cannot be opened

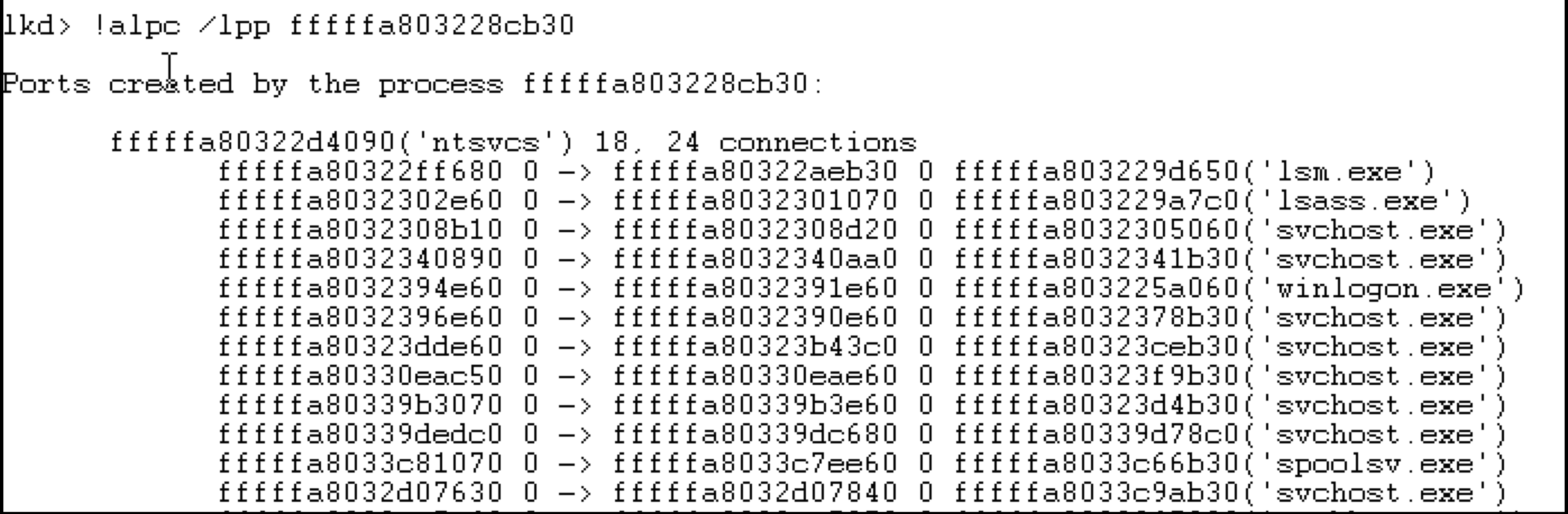
\RPC Control\OLE9E1A96F17E9D4F0F92E1AAB876C4



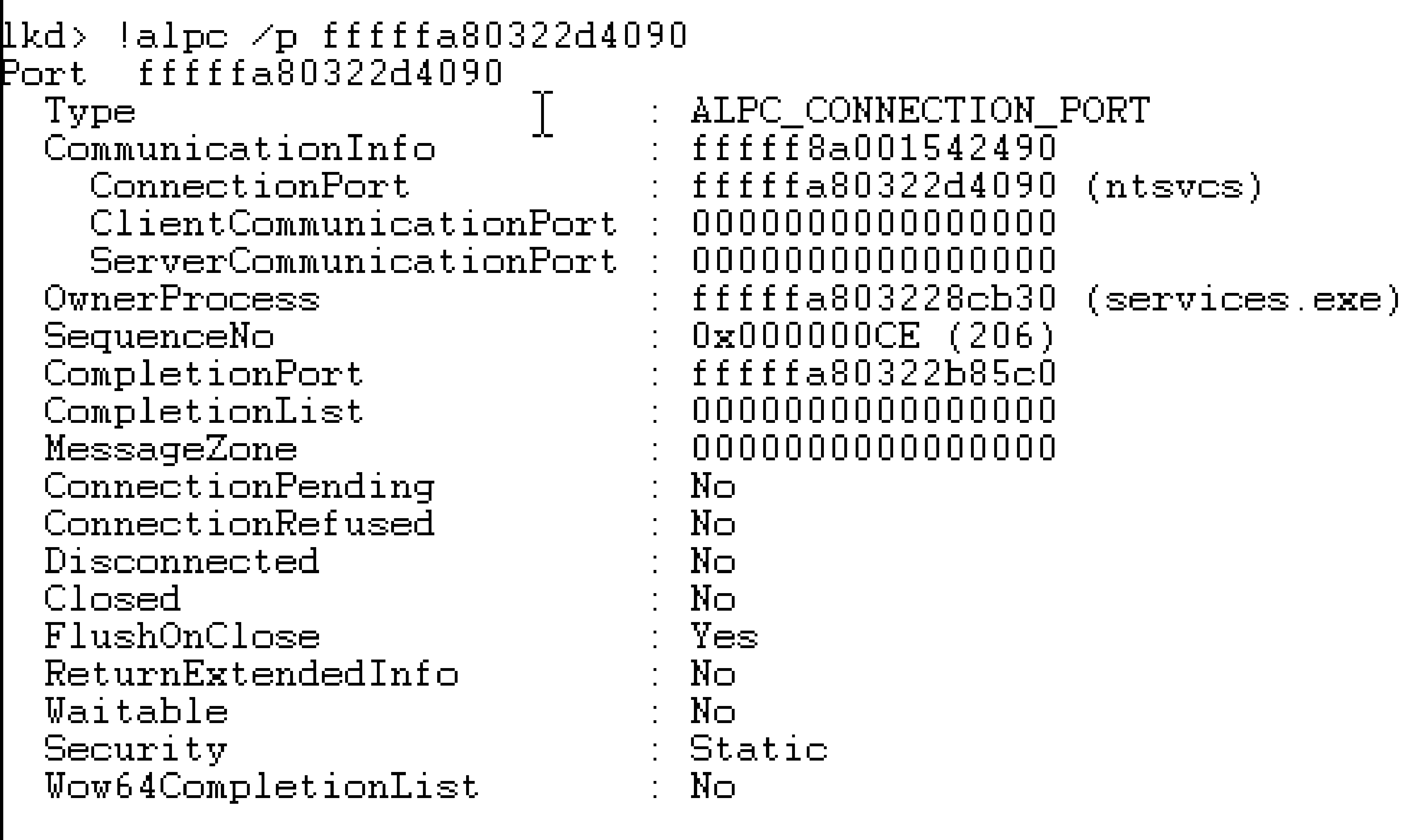
Solid Security. Verified.



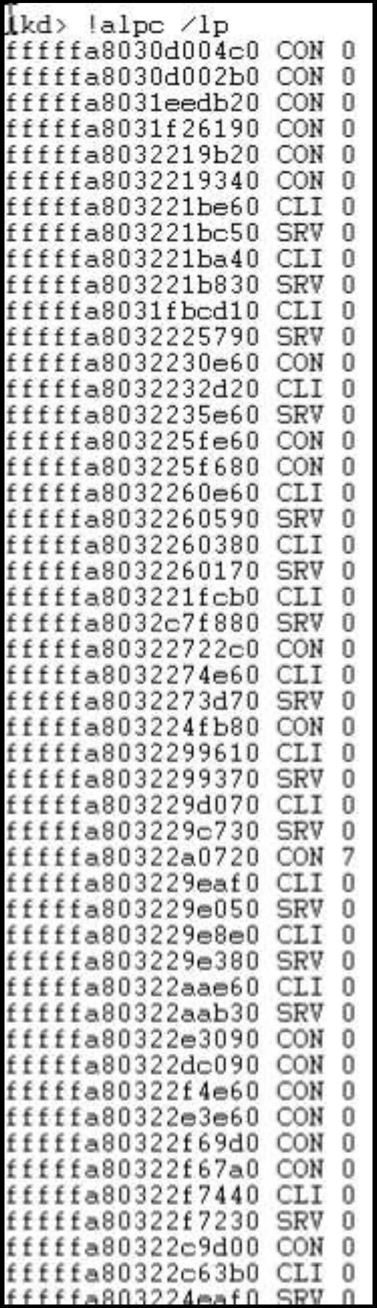
```
lkd> !process 0 0 services.exe
PROCESS fffffa803228cb30
    SessionId: 0  Cid: 0234    Peb: 7fffffd5000  ParentCid: 01b8
    DirBase: 907c9000  ObjectTable: fffff8a0014c8620  HandleCount: 220.
    Image: services.exe
```



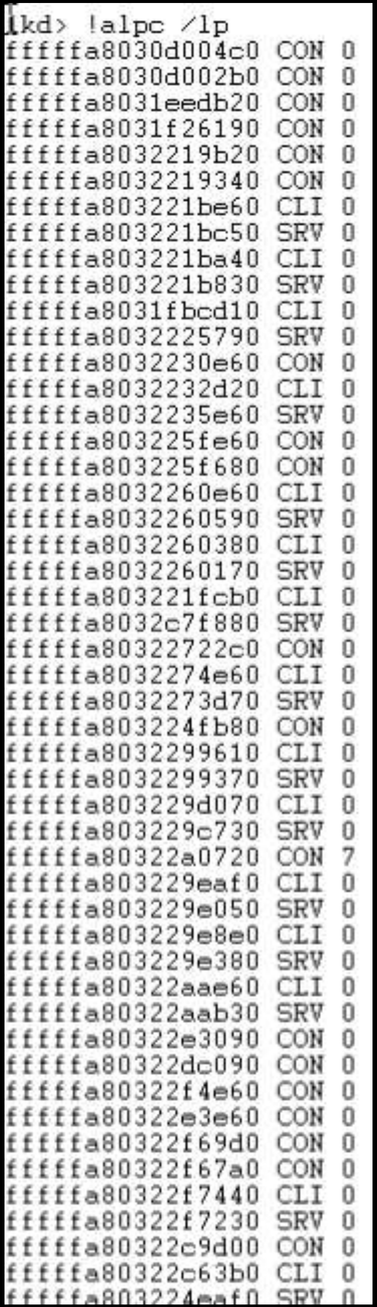
```
lkd> !alpc /lpp fffffa803228cb30

Ports created by the process fffffa803228cb30:

        fffffa80322d4090('ntsvcs') 18, 24 connections
                fffffa80322ff680 0 -> fffffa80322aeb30 0 fffffa803229d650('lsm.exe')
                fffffa8032302e60 0 -> fffffa8032301070 0 fffffa803229a7c0('lsass.exe')
                fffffa8032308b10 0 -> fffffa8032308d20 0 fffffa8032305060('svchost.exe')
                fffffa8032340890 0 -> fffffa8032340aa0 0 fffffa8032341b30('svchost.exe')
                fffffa8032394e60 0 -> fffffa8032391e60 0 fffffa803225a060('winlogon.exe')
                fffffa8032396e60 0 -> fffffa8032390e60 0 fffffa8032378b30('svchost.exe')
                fffffa80323dde60 0 -> fffffa80323b43c0 0 fffffa80323ceb30('svchost.exe')
                fffffa80330eac50 0 -> fffffa80330eae60 0 fffffa80323f9b30('svchost.exe')
                fffffa80339b3070 0 -> fffffa80339b3e60 0 fffffa80323d4b30('svchost.exe')
                fffffa80339dedc0 0 -> fffffa80339dc680 0 fffffa80339d78c0('svchost.exe')
                fffffa8033c81070 0 -> fffffa8033c7ee60 0 fffffa8033c66b30('spoolsv.exe')
                fffffa8032d07630 0 -> fffffa8032d07840 0 fffffa8033c9ab30('svchost.exe')
```



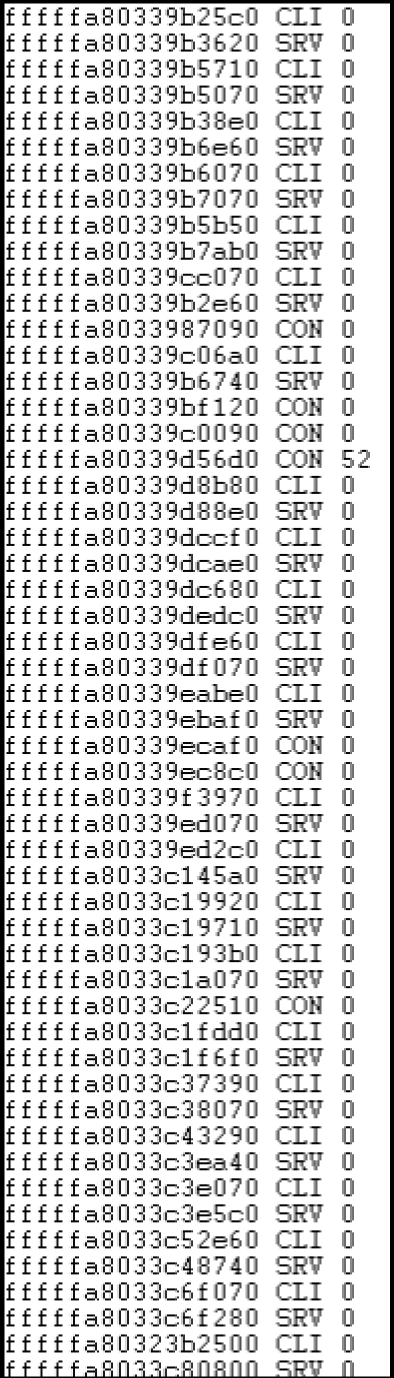
```
lkd> !alpc /p fffffa80322d4090
Port  fffffa80322d4090
  Type                      : ALPC_CONNECTION_PORT
  CommunicationInfo         : fffff8a001542490
    ConnectionPort          : fffffa80322d4090 (ntsvcs)
    ClientCommunicationPort : 0000000000000000
    ServerCommunicationPort : 0000000000000000
  OwnerProcess              : fffffa803228cb30 (services.exe)
  SequenceNo                : 0x000000CE (206)
  CompletionPort            : fffffa80322b85c0
  CompletionList            : 0000000000000000
  MessageZone               : 0000000000000000
  ConnectionPending         : No
  ConnectionRefused         : No
  Disconnected              : No
  Closed                    : No
  FlushOnClose              : Yes
  ReturnExtendedInfo        : No
  Waitable                  : No
  Security                  : Static
  Wow64CompletionList       : No
```



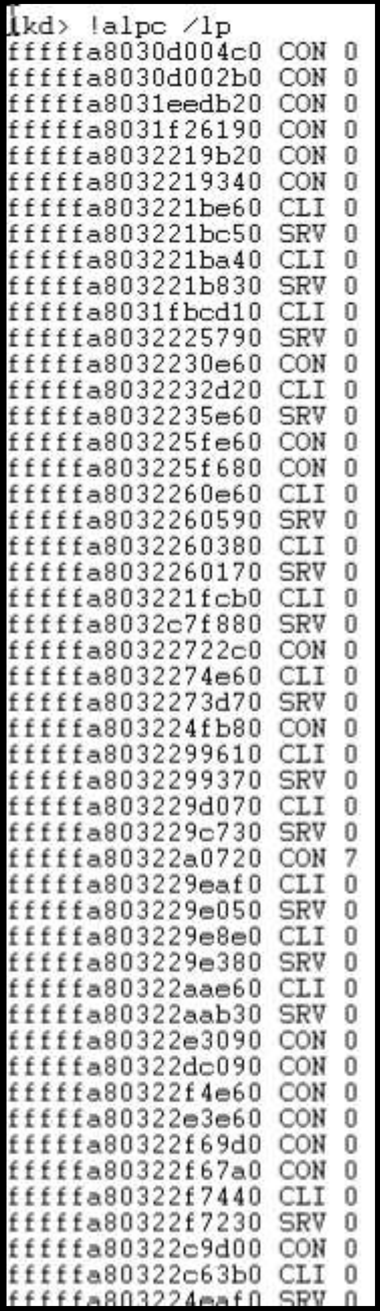
```
lkd> !alpc /lp
fffffa8030d004c0 CON 0
fffffa8030d002b0 CON 0
fffffa8031eedb20 CON 0
fffffa8031f26190 CON 0
fffffa8032219b20 CON 0
fffffa8032219340 CON 0
fffffa803221be60 CLI 0
fffffa803221bc50 SRV 0
fffffa803221ba40 CLI 0
fffffa803221b830 SRV 0
fffffa8031fbcd10 CLI 0
fffffa8032225790 SRV 0
fffffa8032230e60 CON 0
fffffa8032232d20 CLI 0
fffffa8032235e60 SRV 0
fffffa803225fe60 CON 0
fffffa803225f680 CON 0
fffffa8032260e60 CLI 0
fffffa8032260590 SRV 0
fffffa8032260380 CLI 0
fffffa8032260170 SRV 0
fffffa803221fcb0 CLI 0
fffffa8032c7f880 SRV 0
fffffa80322722c0 CON 0
fffffa8032274e60 CLI 0
fffffa8032273d70 SRV 0
fffffa803224fb80 CON 0
fffffa8032299610 CLI 0
fffffa8032299370 SRV 0
fffffa803229d070 CLI 0
fffffa803229c730 SRV 0
fffffa80322a0720 CON 7
fffffa803229eaf0 CLI 0
fffffa803229e050 SRV 0
fffffa803229e8e0 CLI 0
fffffa803229e380 SRV 0
fffffa80322aae60 CLI 0
fffffa80322aab30 SRV 0
fffffa80322e3090 CON 0
fffffa80322dc090 CON 0
fffffa80322f4e60 CON 0
fffffa80322e3e60 CON 0
fffffa80322f69d0 CON 0
fffffa80322f67a0 CON 0
fffffa80322f7440 CLI 0
fffffa80322f7230 SRV 0
fffffa80322c9d00 CON 0
fffffa80322c63b0 CLI 0
fffffa803224eaf0 SRV 0
```



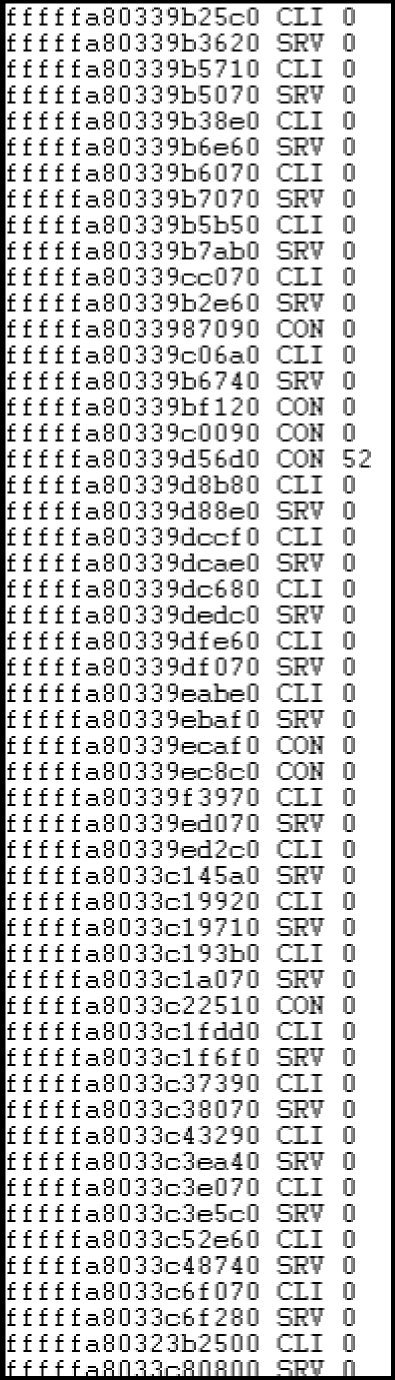
```
lkd> !alpc /lp
fffffa8030d004c0 CON 0
fffffa8030d002b0 CON 0
fffffa8031eedb20 CON 0
fffffa8031f26190 CON 0
fffffa8032219b20 CON 0
fffffa8032219340 CON 0
fffffa803221be60 CLI 0
fffffa803221bc50 SRV 0
fffffa803221ba40 CLI 0
fffffa803221b830 SRV 0
fffffa8031fbcd10 CLI 0
fffffa8032225790 SRV 0
fffffa8032230e60 CON 0
fffffa8032232d20 CLI 0
fffffa8032235e60 SRV 0
fffffa803225fe60 CON 0
fffffa803225f680 CON 0
fffffa8032260e60 CLI 0
fffffa8032260590 SRV 0
fffffa8032260380 CLI 0
fffffa8032260170 SRV 0
fffffa803221fcb0 CLI 0
fffffa8032c7f880 SRV 0
fffffa80322722c0 CON 0
fffffa8032274e60 CLI 0
fffffa8032273d70 SRV 0
fffffa803224fb80 CON 0
fffffa8032299610 CLI 0
fffffa8032299370 SRV 0
fffffa803229d070 CLI 0
fffffa803229c730 SRV 0
fffffa80322a0720 CON 7
fffffa803229eaf0 CLI 0
fffffa803229e050 SRV 0
fffffa803229e8e0 CLI 0
fffffa803229e380 SRV 0
fffffa80322aae60 CLI 0
fffffa80322aab30 SRV 0
fffffa80322e3090 CON 0
fffffa80322dc090 CON 0
fffffa80322f4e60 CON 0
fffffa80322e3e60 CON 0
fffffa80322f69d0 CON 0
fffffa80322f67a0 CON 0
fffffa80322f7440 CLI 0
fffffa80322f7230 SRV 0
fffffa80322c9d00 CON 0
fffffa80322c63b0 CLI 0
fffffa803224eaf0 SRV 0
```



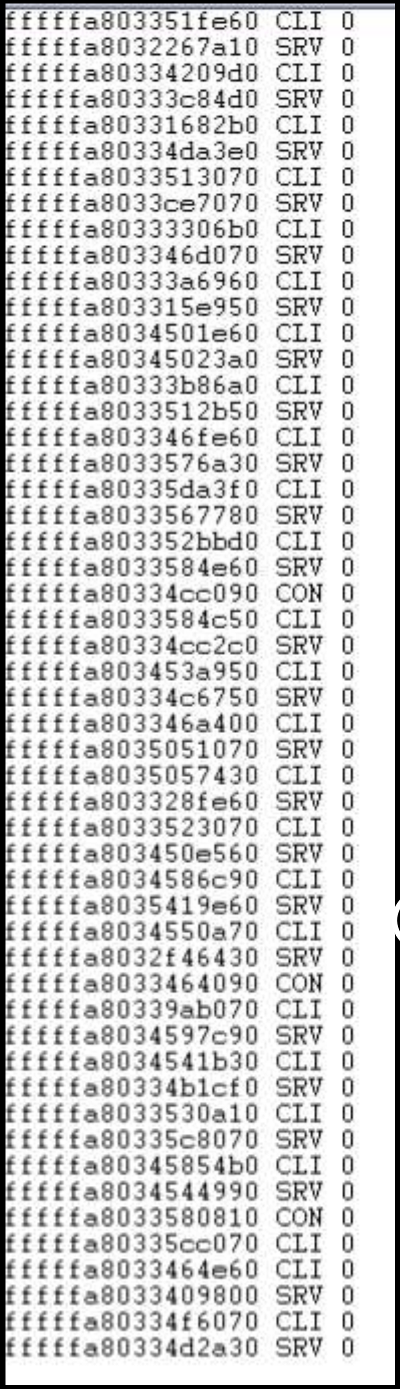
```
fffffa80339b25c0 CLI 0
fffffa80339b3620 SRV 0
fffffa80339b5710 CLI 0
fffffa80339b5070 SRV 0
fffffa80339b38e0 CLI 0
fffffa80339b6e60 SRV 0
fffffa80339b6070 CLI 0
fffffa80339b7070 SRV 0
fffffa80339b5b50 CLI 0
fffffa80339b7ab0 SRV 0
fffffa80339cc070 CLI 0
fffffa80339b2e60 SRV 0
fffffa8033987090 CON 0
fffffa80339c06a0 CLI 0
fffffa80339b6740 SRV 0
fffffa80339bf120 CON 0
fffffa80339c0090 CON 0
fffffa80339d56d0 CON 52
fffffa80339d8b80 CLI 0
fffffa80339d88e0 SRV 0
fffffa80339dccf0 CLI 0
fffffa80339dcae0 SRV 0
fffffa80339dc680 CLI 0
fffffa80339dedc0 SRV 0
fffffa80339dfe60 CLI 0
fffffa80339df070 SRV 0
fffffa80339eabe0 CLI 0
fffffa80339ebaf0 SRV 0
fffffa80339ecaf0 CON 0
fffffa80339ec8c0 CON 0
fffffa80339f3970 CLI 0
fffffa80339ed070 SRV 0
fffffa80339ed2c0 CLI 0
fffffa8033c145a0 SRV 0
fffffa8033c19920 CLI 0
fffffa8033c19710 SRV 0
fffffa8033c193b0 CLI 0
fffffa8033c1a070 SRV 0
fffffa8033c22510 CON 0
fffffa8033c1fdd0 CLI 0
fffffa8033c1f6f0 SRV 0
fffffa8033c37390 CLI 0
fffffa8033c38070 SRV 0
fffffa8033c43290 CLI 0
fffffa8033c3ea40 SRV 0
fffffa8033c3e070 CLI 0
fffffa8033c3e5c0 SRV 0
fffffa8033c52e60 CLI 0
fffffa8033c48740 SRV 0
fffffa8033c6f070 CLI 0
fffffa8033c6f280 SRV 0
fffffa80323b2500 CLI 0
fffffa8033c80800 SRV 0
```



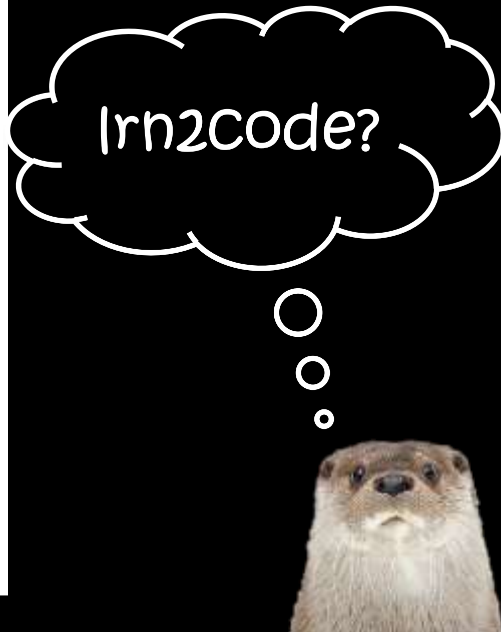
```
lkd> !alpc /lp
fffffa8030d004c0 CON 0
fffffa8030d002b0 CON 0
fffffa8031eedb20 CON 0
fffffa8031f26190 CON 0
fffffa8032219b20 CON 0
fffffa8032219340 CON 0
fffffa803221be60 CLI 0
fffffa803221bc50 SRV 0
fffffa803221ba40 CLI 0
fffffa803221b830 SRV 0
fffffa8031fbcd10 CLI 0
fffffa8032225790 SRV 0
fffffa8032230e60 CON 0
fffffa8032232d20 CLI 0
fffffa8032235e60 SRV 0
fffffa803225fe60 CON 0
fffffa803225f680 CON 0
fffffa8032260e60 CLI 0
fffffa8032260590 SRV 0
fffffa8032260380 CLI 0
fffffa8032260170 SRV 0
fffffa803221fcb0 CLI 0
fffffa8032c7f880 SRV 0
fffffa80322722c0 CON 0
fffffa8032274e60 CLI 0
fffffa8032273d70 SRV 0
fffffa803224fb80 CON 0
fffffa8032299610 CLI 0
fffffa8032299370 SRV 0
fffffa803229d070 CLI 0
fffffa803229c730 SRV 0
fffffa80322a0720 CON 7
fffffa803229eaf0 CLI 0
fffffa803229e050 SRV 0
fffffa803229e8e0 CLI 0
fffffa803229e380 SRV 0
fffffa80322aae60 CLI 0
fffffa80322aab30 SRV 0
fffffa80322e3090 CON 0
fffffa80322dc090 CON 0
fffffa80322f4e60 CON 0
fffffa80322e3e60 CON 0
fffffa80322f69d0 CON 0
fffffa80322f67a0 CON 0
fffffa80322f7440 CLI 0
fffffa80322f7230 SRV 0
fffffa80322c9d00 CON 0
fffffa80322c63b0 CLI 0
fffffa803224eaf0 SRV 0
```



```
fffffa80339b25c0 CLI 0
fffffa80339b3620 SRV 0
fffffa80339b5710 CLI 0
fffffa80339b5070 SRV 0
fffffa80339b38e0 CLI 0
fffffa80339b6e60 SRV 0
fffffa80339b6070 CLI 0
fffffa80339b7070 SRV 0
fffffa80339b5b50 CLI 0
fffffa80339b7ab0 SRV 0
fffffa80339cc070 CLI 0
fffffa80339b2e60 SRV 0
fffffa8033987090 CON 0
fffffa80339c06a0 CLI 0
fffffa80339b6740 SRV 0
fffffa80339bf120 CON 0
fffffa80339c0090 CON 0
fffffa80339d56d0 CON 52
fffffa80339d8b80 CLI 0
fffffa80339d88e0 SRV 0
fffffa80339dccf0 CLI 0
fffffa80339dcae0 SRV 0
fffffa80339dc680 CLI 0
fffffa80339dedc0 SRV 0
fffffa80339dfe60 CLI 0
fffffa80339df070 SRV 0
fffffa80339eabe0 CLI 0
fffffa80339ebaf0 SRV 0
fffffa80339ecaf0 CON 0
fffffa80339ec8c0 CON 0
fffffa80339f3970 CLI 0
fffffa80339ed070 SRV 0
fffffa80339ed2c0 CLI 0
fffffa8033c145a0 SRV 0
fffffa8033c19920 CLI 0
fffffa8033c19710 SRV 0
fffffa8033c193b0 CLI 0
fffffa8033c1a070 SRV 0
fffffa8033c22510 CON 0
fffffa8033c1fdd0 CLI 0
fffffa8033c1f6f0 SRV 0
fffffa8033c37390 CLI 0
fffffa8033c38070 SRV 0
fffffa8033c43290 CLI 0
fffffa8033c3ea40 SRV 0
fffffa8033c3e070 CLI 0
fffffa8033c3e5c0 SRV 0
fffffa8033c52e60 CLI 0
fffffa8033c48740 SRV 0
fffffa8033c6f070 CLI 0
fffffa8033c6f280 SRV 0
fffffa80323b2500 CLI 0
fffffa8033c80800 SRV 0
```



```
fffffa803351fe60 CLI 0
fffffa8032267a10 SRV 0
fffffa80334209d0 CLI 0
fffffa80333c84d0 SRV 0
fffffa80331682b0 CLI 0
fffffa80334da3e0 SRV 0
fffffa8033513070 CLI 0
fffffa8033ce7070 SRV 0
fffffa80333306b0 CLI 0
fffffa803346d070 SRV 0
fffffa80333a6960 CLI 0
fffffa803315e950 SRV 0
fffffa8034501e60 CLI 0
fffffa80345023a0 SRV 0
fffffa80333b86a0 CLI 0
fffffa8033512b50 SRV 0
fffffa803346fe60 CLI 0
fffffa8033576a30 SRV 0
fffffa80335da3f0 CLI 0
fffffa8033567780 SRV 0
fffffa803352bbd0 CLI 0
fffffa8033584e60 SRV 0
fffffa80334cc090 CON 0
fffffa8033584c50 CLI 0
fffffa80334cc2c0 SRV 0
fffffa803453a950 CLI 0
fffffa80334c6750 SRV 0
fffffa803346a400 CLI 0
fffffa8035051070 SRV 0
fffffa8035057430 CLI 0
fffffa803328fe60 SRV 0
fffffa8033523070 CLI 0
fffffa803450e560 SRV 0
fffffa8034586c90 CLI 0
fffffa8035419e60 SRV 0
fffffa8034550a70 CLI 0
fffffa8032f46430 SRV 0
fffffa8033464090 CON 0
fffffa80339ab070 CLI 0
fffffa8034597c90 SRV 0
fffffa8034541b30 CLI 0
fffffa80334b1cf0 SRV 0
fffffa8033530a10 CLI 0
fffffa80335c8070 SRV 0
fffffa80345854b0 CLI 0
fffffa8034544990 SRV 0
fffffa8033580810 CON 0
fffffa80335cc070 CLI 0
fffffa8033464e60 CLI 0
fffffa8033409800 SRV 0
fffffa80334f6070 CLI 0
fffffa80334d2a30 SRV 0
```



Cutting Edge Tech

- https://github.com/bnagy/rBuggery
- Ruby wrapper for dbgeng.dll (windbg)
- Fully scriptable debugger
  - kernel debugging
  - LOCAL kernel debugging

- Unique Features:
  - Actually works

Know what the Windows Kernel needs?

A JSON API!

- – Wrap rBuggery with Sinatra
- – Connect with Go
- – Map ALPC
- – Drink Barry’s salty ragetears

alpcmap

- Start debugger bridge on Windows
- Connect from anywhere
- Maps ports, serves webapp graph
- https://github.com/bnagy/alpcmap

alpcmap

- Automates and parses:
  - !alpc /lp, /lpc, /p
  - dt nt_OBJECT_HEADER
  - !token
  - !sd
  - !object
  - !process
  - …

Initiating demonstration…

Phase III - Generation

What to send?

Phase III - Generation

Examine existing messages!

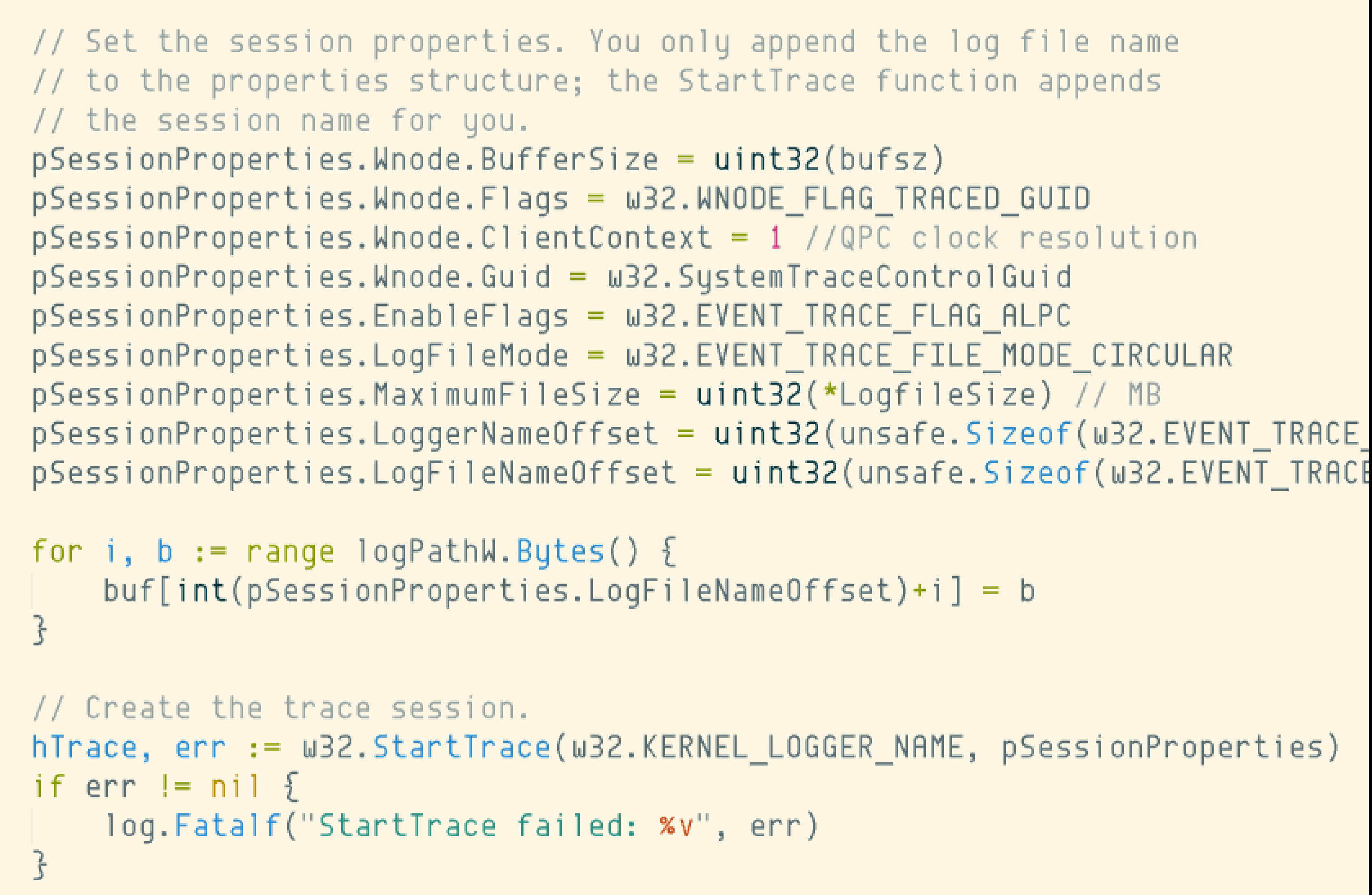
ALPC Message Logging

- Event Tracing for Windows (ETW)?
- advapi32 has StartTrace() …
- EVENT_TRACE_FLAG_ALPC …
- SystemTraceControlGuid …
- CODEZ!

ALPC Message Logging

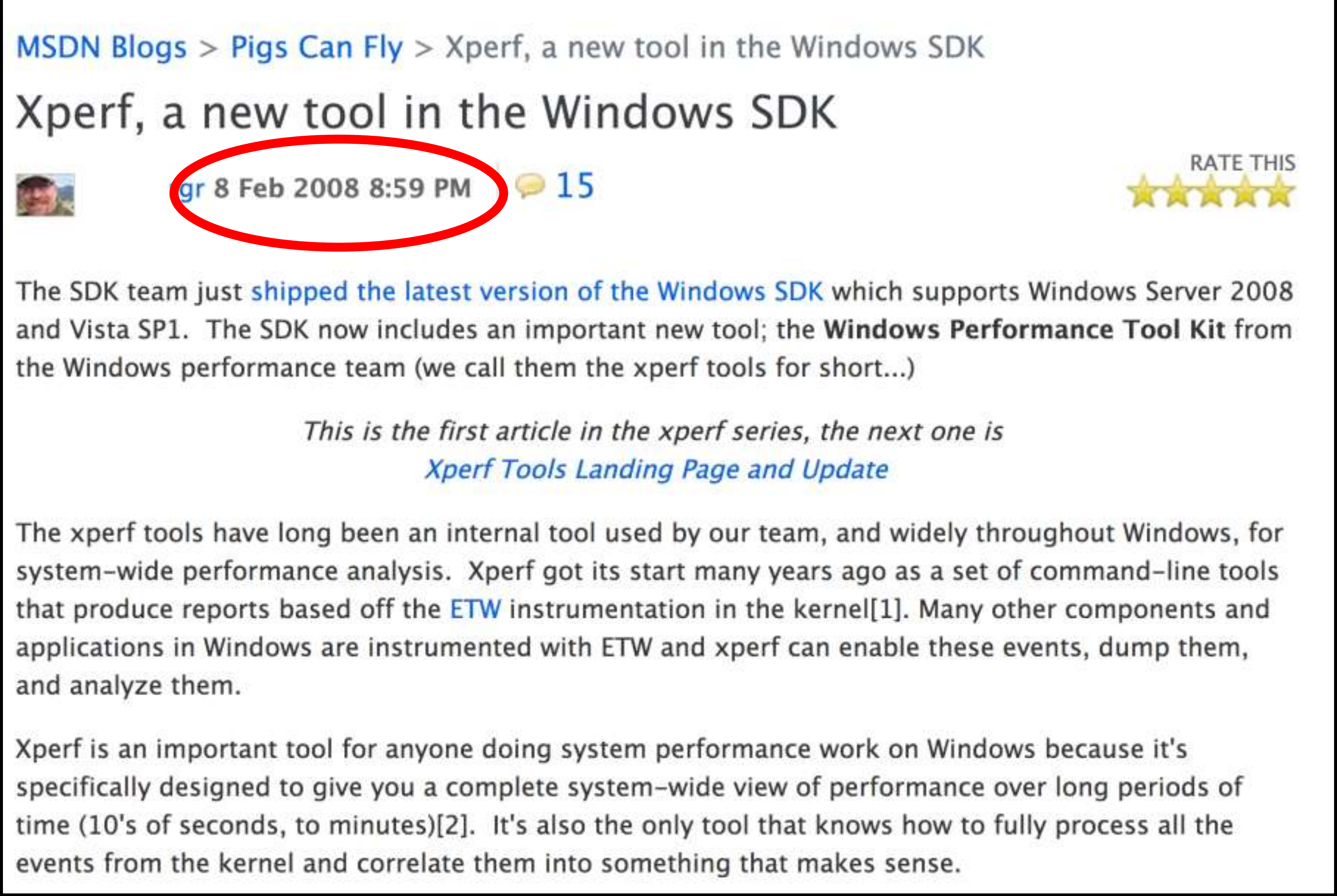
- Hacked StartTrace() support into w32
  - Needs lots of support cruft

ETW



```
// Set the session properties. You only append the log file name
// to the properties structure; the StartTrace function appends
// the session name for you.
pSessionProperties.Wnode.BufferSize = uint32(bufsz)
pSessionProperties.Wnode.Flags = w32.WNODE_FLAG_TRACED_GUID
pSessionProperties.Wnode.ClientContext = 1 //QPC clock resolution
pSessionProperties.Wnode.Guid = w32.SystemTraceControlGuid
pSessionProperties.EnableFlags = w32.EVENT_TRACE_FLAG_ALPC
pSessionProperties.LogFileMode = w32.EVENT_TRACE_FILE_MODE_CIRCULAR
pSessionProperties.MaximumFileSize = uint32(*LogfileSize) // MB
pSessionProperties.LoggerNameOffset = uint32(unsafe.Sizeof(w32.EVENT_TRACE_
pSessionProperties.LogFileNameOffset = uint32(unsafe.Sizeof(w32.EVENT_TRACE

for i, b := range logPathW.Bytes() {
    buf[int(pSessionProperties.LogFileNameOffset)+i] = b
}

// Create the trace session.
hTrace, err := w32.StartTrace(w32.KERNEL_LOGGER_NAME, pSessionProperties)
if err != nil {
    log.Fatalf("StartTrace failed: %v", err)
}
```

FAIL



MSDN Blogs > Pigs Can Fly > Xperf, a new tool in the Windows SDK

Xperf, a new tool in the Windows SDK

gr 8 Feb 2008 8:59 PM 15

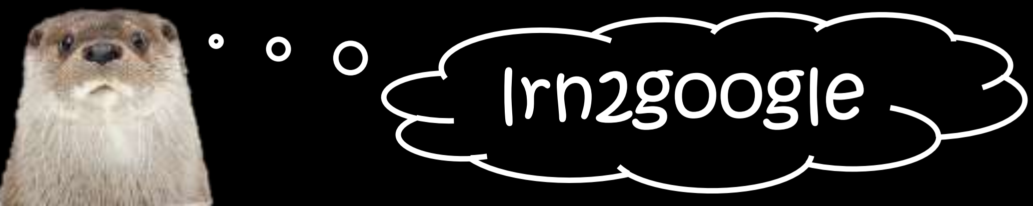
RATE THIS ★★★★★

The SDK team just shipped the latest version of the Windows SDK which supports Windows Server 2008 and Vista SP1. The SDK now includes an important new tool; the **Windows Performance Tool Kit** from the Windows performance team (we call them the xperf tools for short...)

This is the first article in the xperf series, the next one is
Xperf Tools Landing Page and Update

The xperf tools have long been an internal tool used by our team, and widely throughout Windows, for system-wide performance analysis. Xperf got its start many years ago as a set of command-line tools that produce reports based off the ETW instrumentation in the kernel[1]. Many other components and applications in Windows are instrumented with ETW and xperf can enable these events, dump them, and analyze them.

Xperf is an important tool for anyone doing system performance work on Windows because it's specifically designed to give you a complete system-wide view of performance over long periods of time (10's of seconds, to minutes)[2]. It's also the only tool that knows how to fully process all the events from the kernel and correlate them into something that makes sense.



DOUBLE FAIL

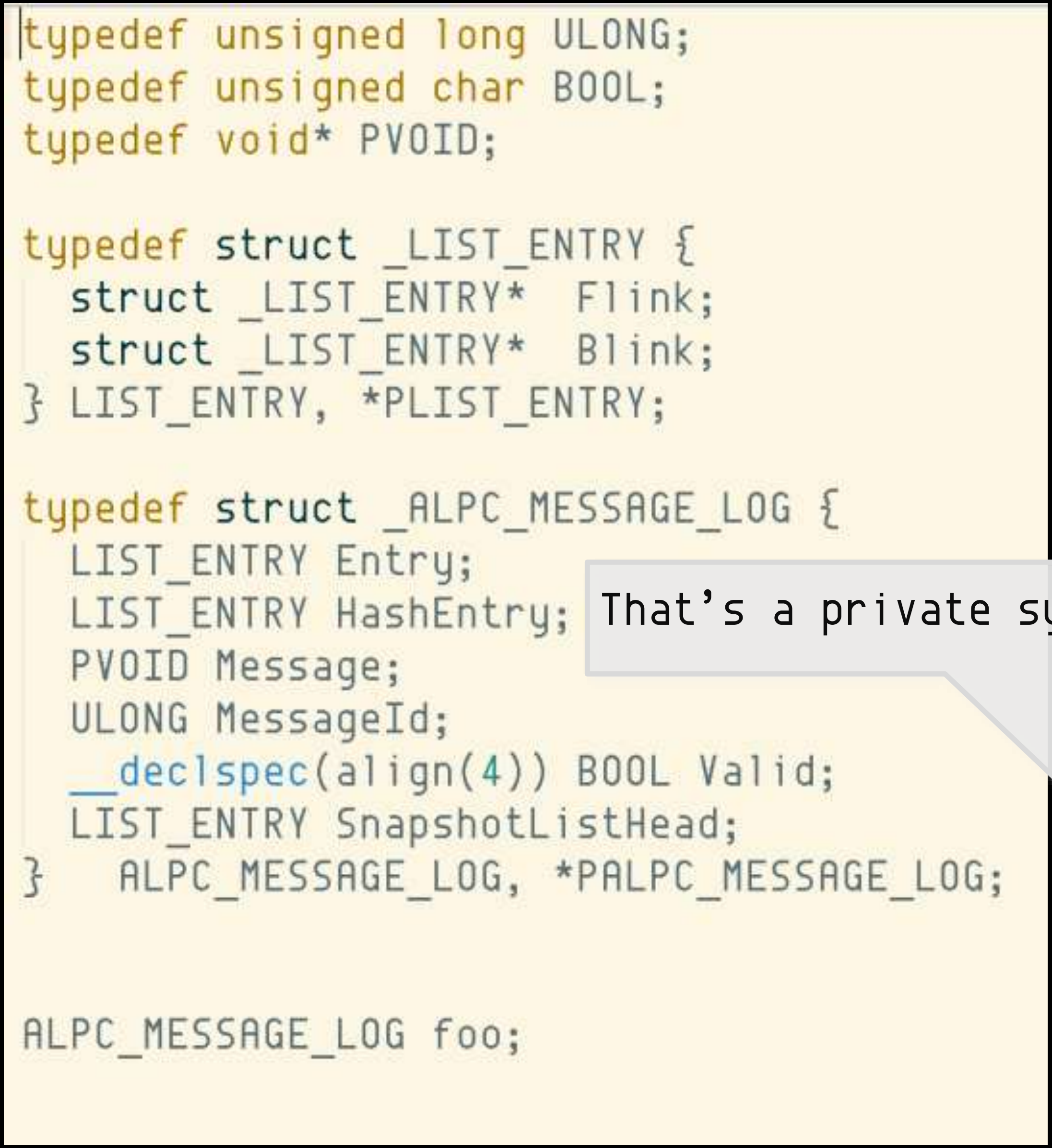
… The message contents aren’t even in the ETW output only the Message IDs ☹

Undocumented !alpc switch /lm !

Set “AlpcMessageLog” in HKLM\CCS\Control\Session !

Use this sweet trick to add private ALPC_MESSAGE_LOG symbol…

symbol.c



```
typedef unsigned long ULONG;
typedef unsigned char BOOL;
typedef void* PVOID;

typedef struct _LIST_ENTRY {
  struct _LIST_ENTRY*  Flink;
  struct _LIST_ENTRY*  Blink;
} LIST_ENTRY, *PLIST_ENTRY;

typedef struct _ALPC_MESSAGE_LOG {
  LIST_ENTRY Entry;
  LIST_ENTRY HashEntry;
  PVOID Message;
  ULONG MessageId;
  __declspec(align(4)) BOOL Valid;
  LIST_ENTRY SnapshotListHead;
}   ALPC_MESSAGE_LOG, *PALPC_MESSAGE_LOG;


ALPC_MESSAGE_LOG foo;
```

```
cl.exe /Zi /Gz /c /Fdntkrnlmp
/IC:\WinDDK\7600.16385.1\inc\ddk
/IC:\WinDDK\7600.16385.1\inc\crt
/D_X86_=1 symbols.c
// FIXME
```

Pass in the existing .pdb
It will be modified in-place
(so save a copy)

FAIL

COSEINC®

Solid Security. Verified.

Oh, BTW, /lm only works in Vista…

(except debug builds)

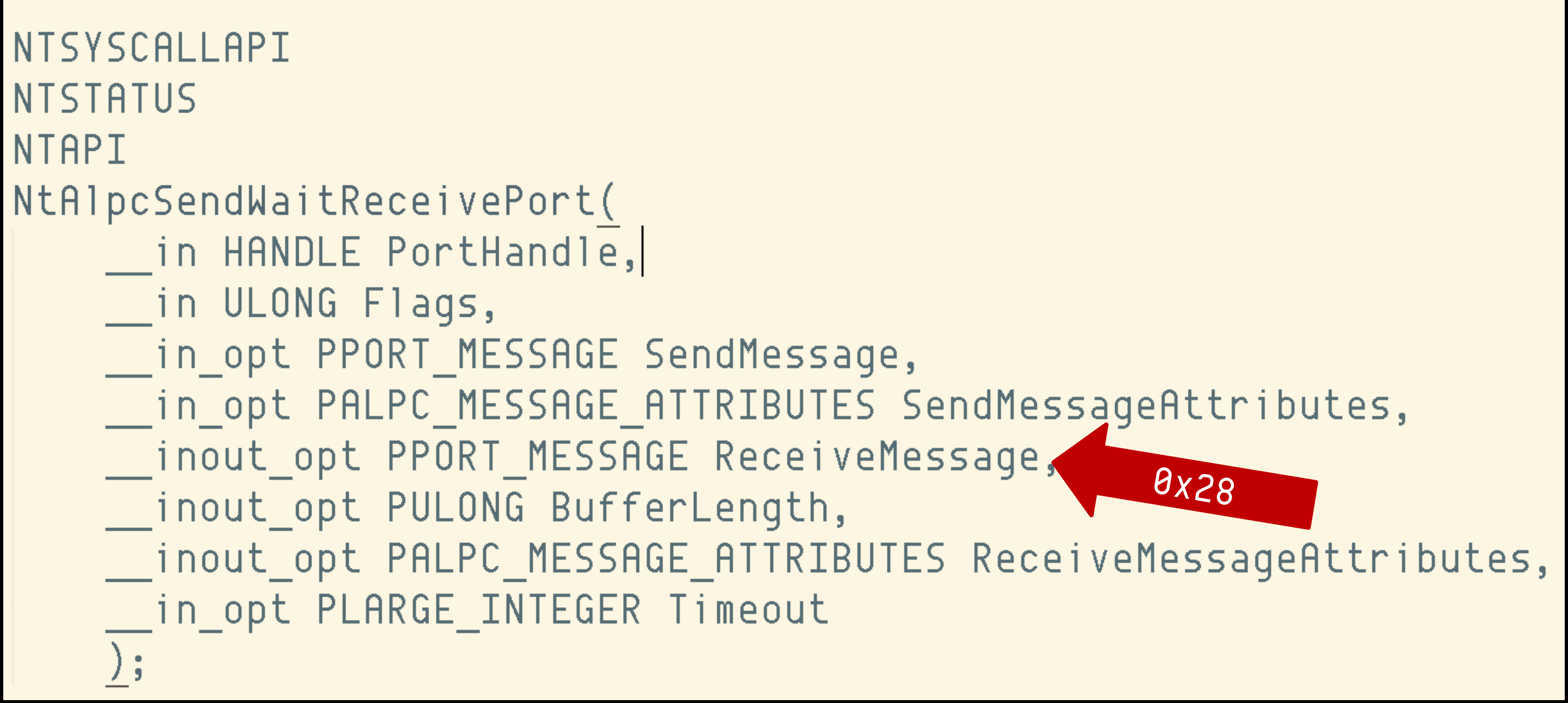
ALPC Message Logging

FINE! Let’s use rBuggery then.

```
ntdll!ZwAlpcSendWaitReceivePort:
4c8bd1          mov     r10,rcx
b882000000      mov     eax,82h
0f05            syscall
c3              ret
```

Message contents added and removed here ;-)

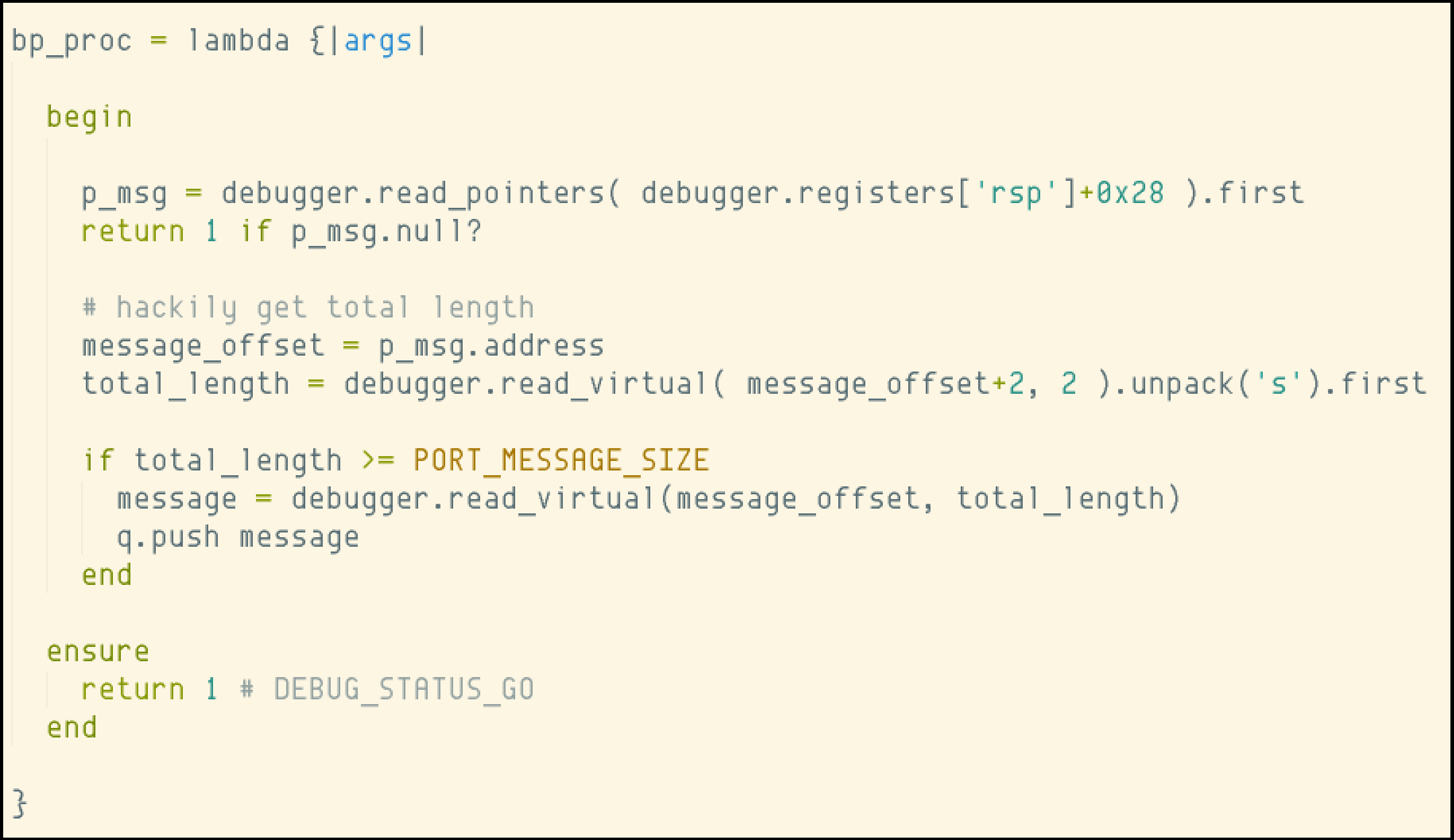
ALPC Message Logging



```
NTSYSCALLAPI
NTSTATUS
NTAPI
NtAlpcSendWaitReceivePort(
    __in HANDLE PortHandle,
    __in ULONG Flags,
    __in_opt PPORT_MESSAGE SendMessage,
    __in_opt PALPC_MESSAGE_ATTRIBUTES SendMessageAttributes,
    __inout_opt PPORT_MESSAGE ReceiveMessage,
    __inout_opt PULONG BufferLength,
    __inout_opt PALPC_MESSAGE_ATTRIBUTES ReceiveMessageAttributes,
    __in_opt PLARGE_INTEGER Timeout
    );
```

x64 fastcall uses registers for first 4 args, but space is still reserved for them on the stack…

Breakpoint Callback



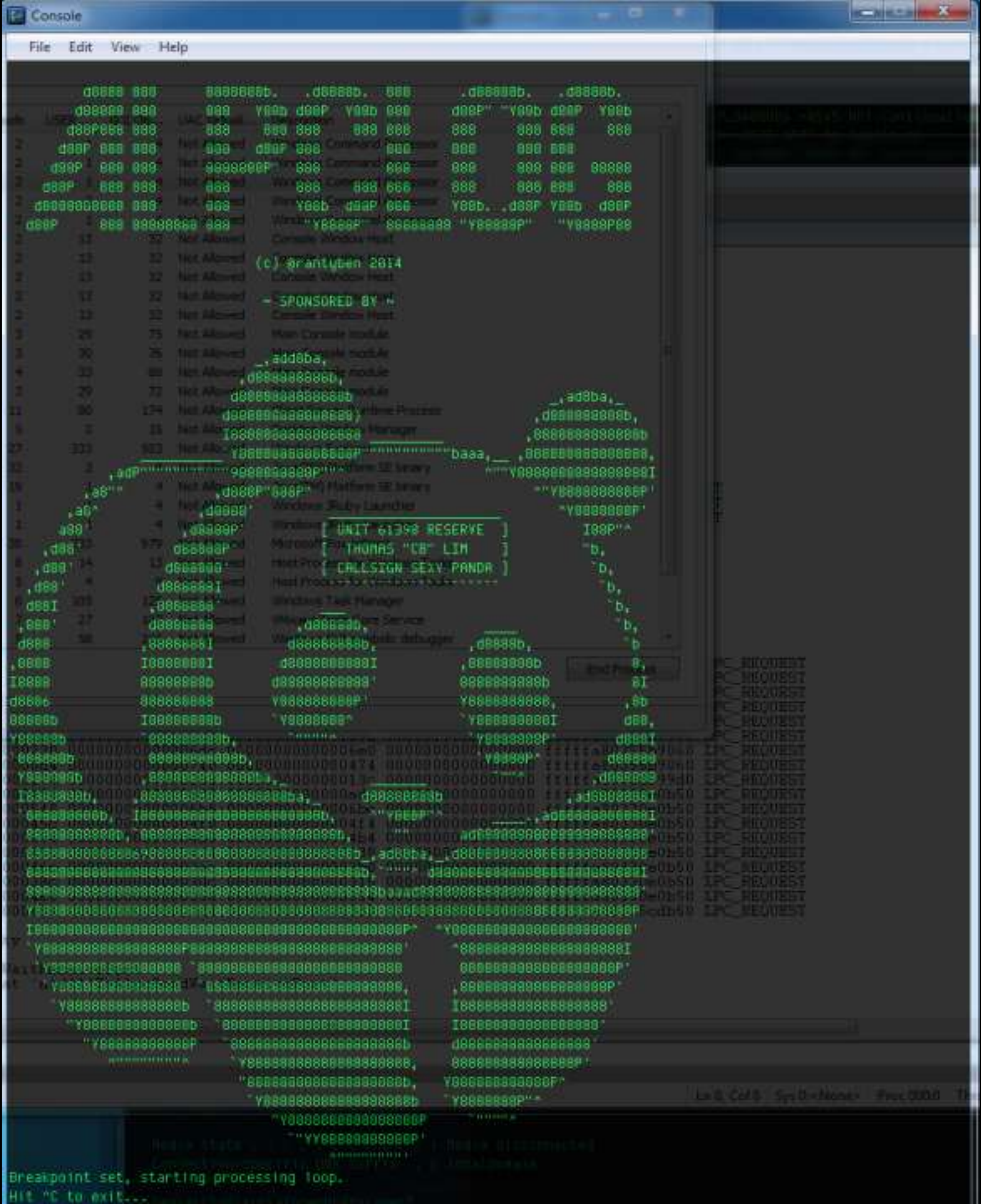
```
bp_proc = lambda {|args|

  begin

    p_msg = debugger.read_pointers( debugger.registers['rsp']+0x28 ).first
    return 1 if p_msg.null?

    # hackily get total length
    message_offset = p_msg.address
    total_length = debugger.read_virtual( message_offset+2, 2 ).unpack('s').first

    if total_length >= PORT_MESSAGE_SIZE
      message = debugger.read_virtual(message_offset, total_length)
      q.push message
    end

  ensure
    return 1 # DEBUG_STATUS_GO
  end

}
```

ᕕ(ಠ‿ಠ)ᕗ

Sappy Moralizing Interlude

- Learned cool stuff while failing
- Presenting failure helps everyone

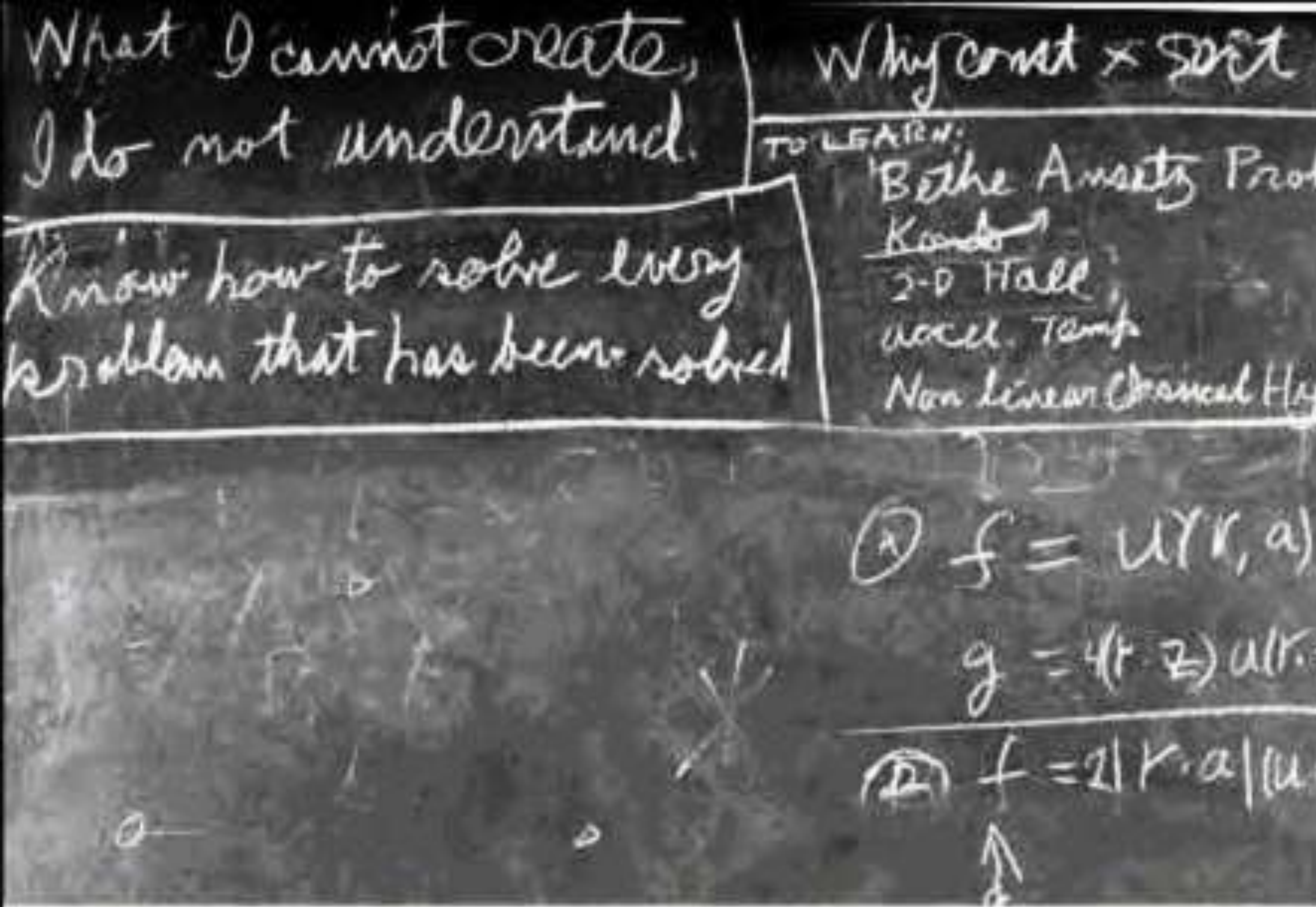
DEMO?



```
      d8888 888      8888888b.   .d8888b.  888      .d88888b.   .d8888b.
     d88888 888      888   Y88b d88P  Y88b 888     d88P" "Y88b d88P  Y88b
    d88P888 888      888    888 888    888 888     888     888 888    888
   d88P 888 888      888   d88P 888        888     888     888 888
  d88P  888 888      8888888P"  888        888     888     888 888  88888
 d88P   888 888      888        888    888 888     888     888 888    888
d8888888888 888      888        Y88b  d88P 888     Y88b. .d88P Y88b  d88P
d88P    888 88888888 888         "Y8888P"  88888888 "Y88888P"   "Y8888P88

                 (c) @grantyben 2014

                  ~ SPONSORED BY ~

                           [ UNIT 61398 RESERVE  ]
                           [ THOMAS "CB" LIM     ]
                           [ CALLSIGN SEXY PANDA ]

Breakpoint set, starting processing loop.
Hit ^C to exit...
```

Phase IV - Delivery

ALPC Programming



What I cannot create,
I do not understand.

Why const × sort

TO LEARN:
Bethe Ansatz Prob
~~Kondo~~
2-D Hall
accel. Temp
Non linear Classical Hy

Know how to solve every
problem that has been solved

(A) $f = u r(r, a)$

$g = 4(r \cdot z)\, u(r.$

(B) $f = 2|r \cdot a|(u$

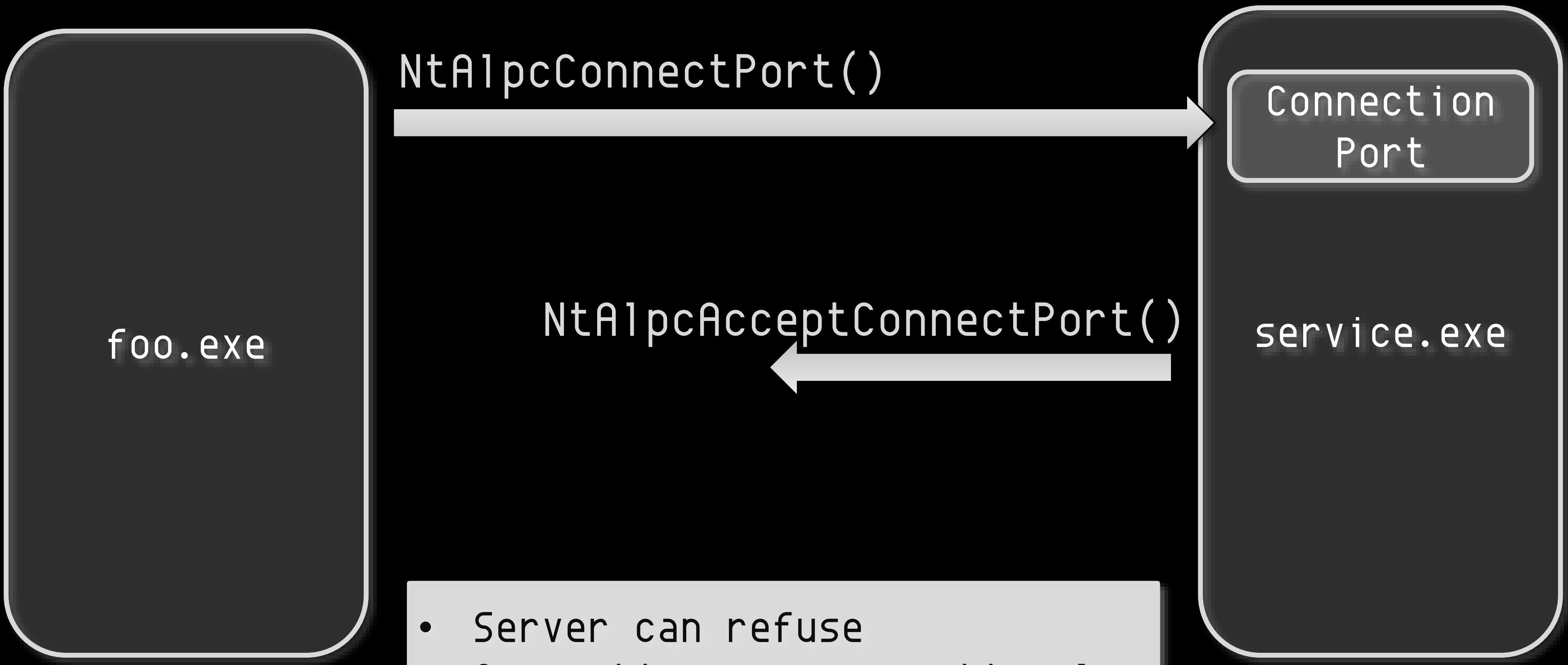
Programming with ALPC

- Very little documentation!
  - New Edition of Windows Internals
  - Some LPC stuff on j00ru’s blog
  - Alex Ionescu’s trainings
  - ntlpcapi.h

  - This project (didn’t test)
  - https://github.com/avalon1610/ALPC/tree/master/ALPC

Why use Go?

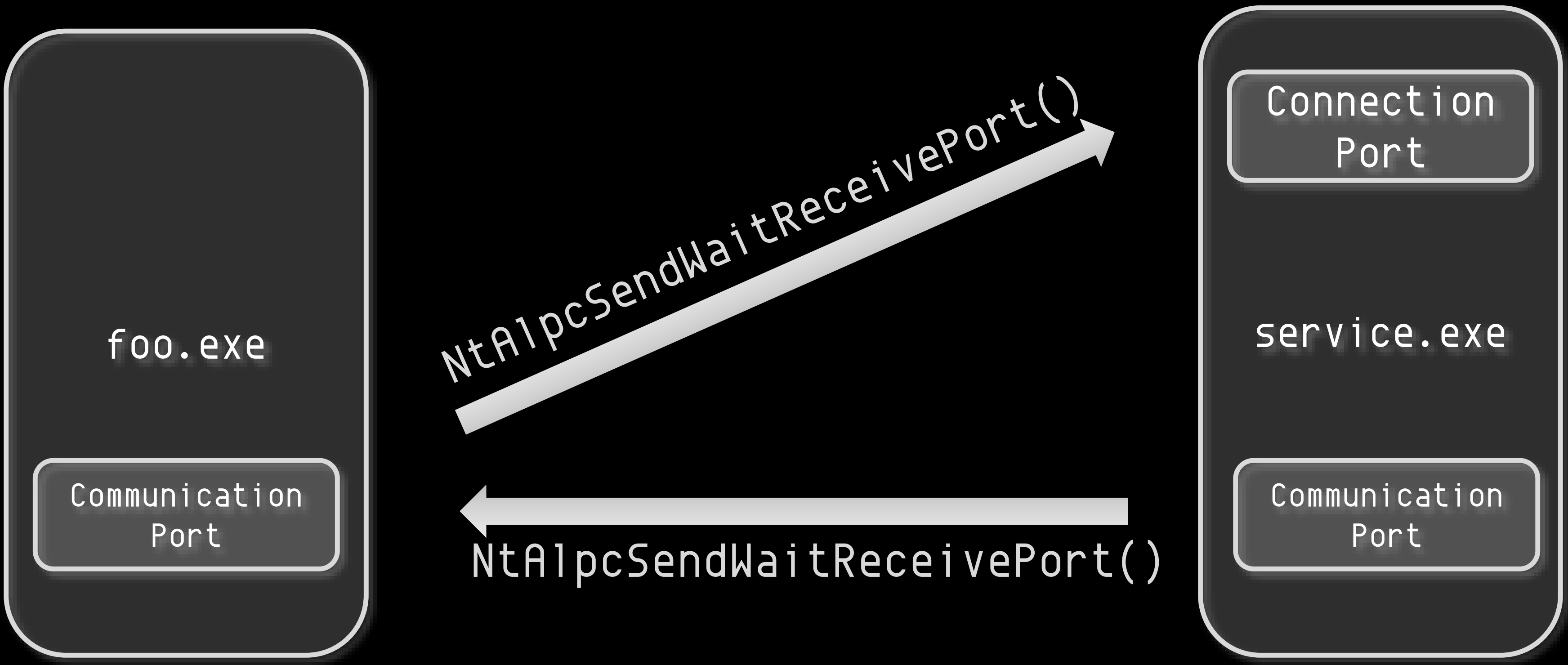
- Compiled.
  - Windows users can ship binaries
- Idiomatic Windows binding (w32)
- cgo - use headers directly in a pinch
- Raging code hipster



- Server can refuse
- Connection message optional

COSEINC®

Solid Security. Verified.



foo.exe

Communication Port

NtAlpcSendWaitReceivePort()

NtAlpcSendWaitReceivePort()

Connection Port

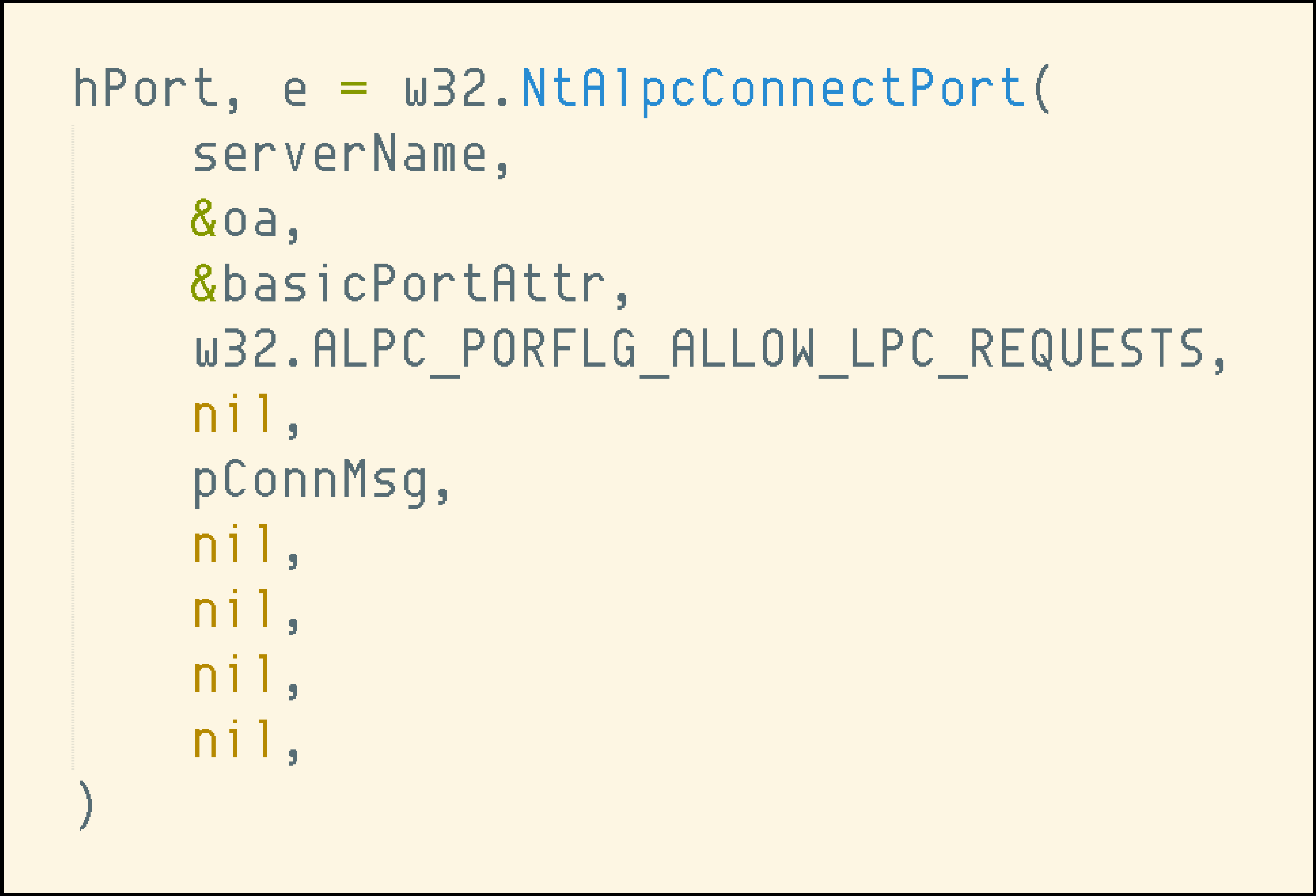
service.exe

Communication Port

Servers only wait on one port!

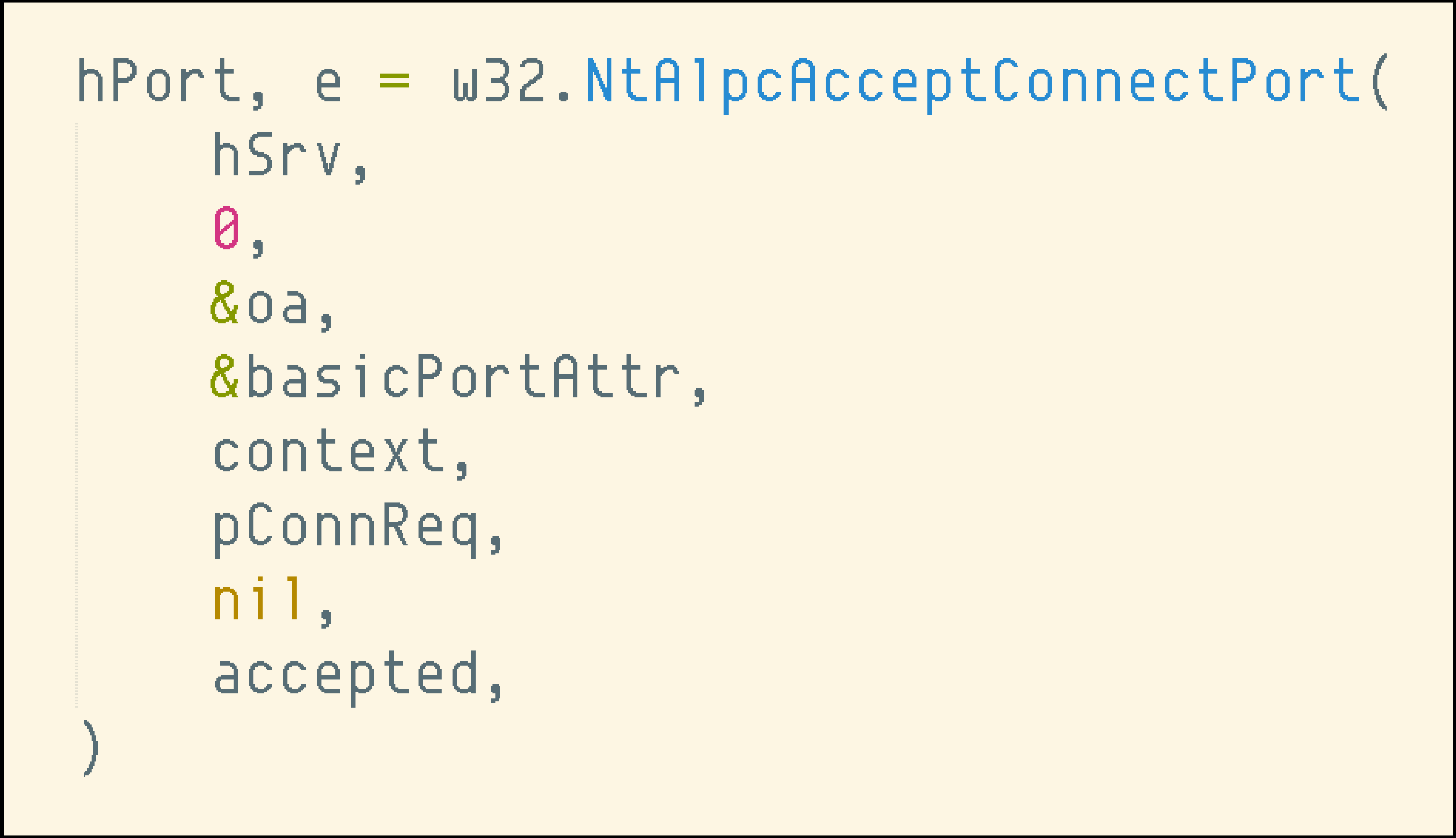
Your flippant manner wearies me. Display your pathetic code immediately.

Connection - Client



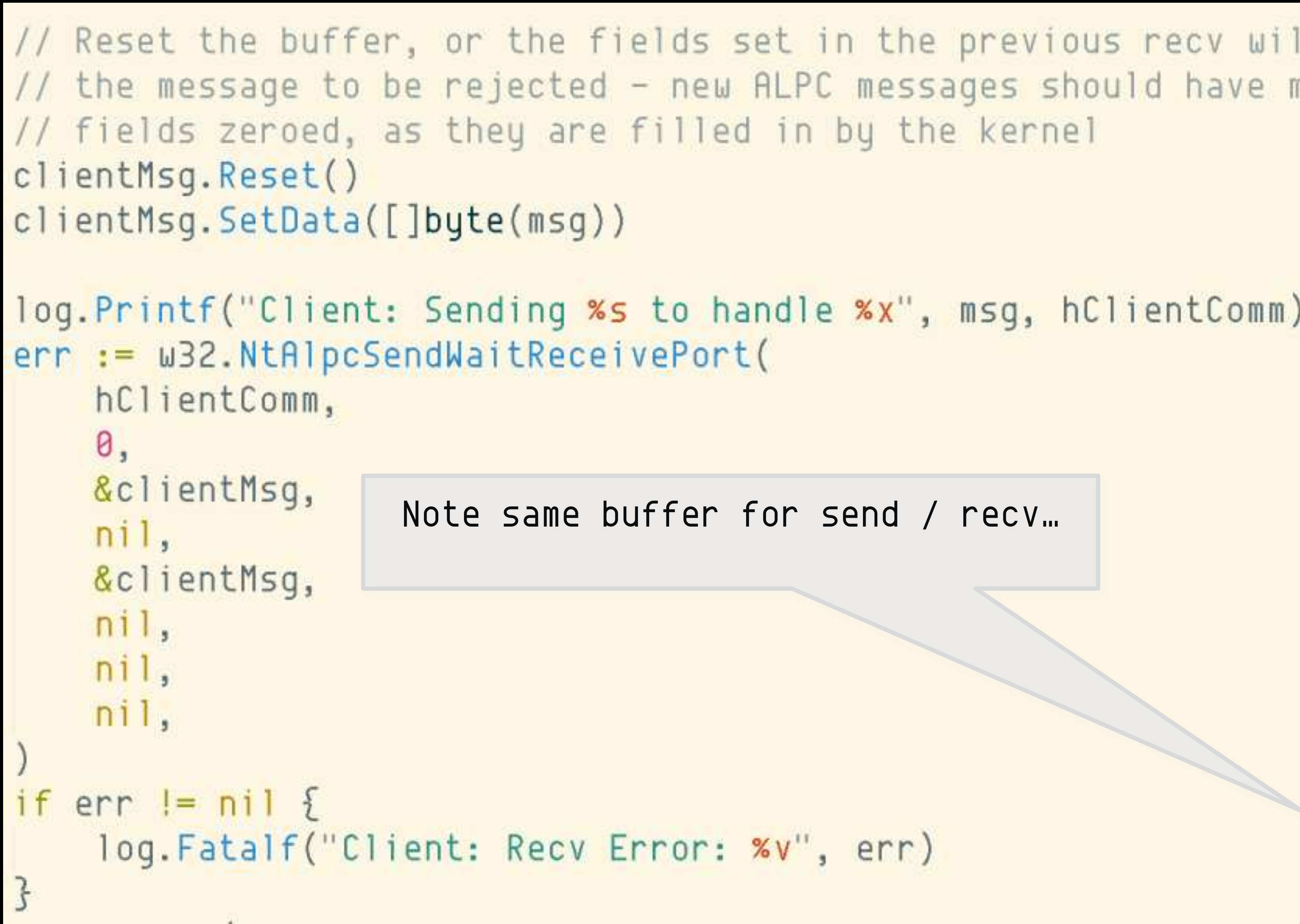
```
hPort, e = w32.NtAlpcConnectPort(
    serverName,
    &oa,
    &basicPortAttr,
    w32.ALPC_PORFLG_ALLOW_LPC_REQUESTS,
    nil,
    pConnMsg,
    nil,
    nil,
    nil,
    nil,
)
```

Acceptance - Server



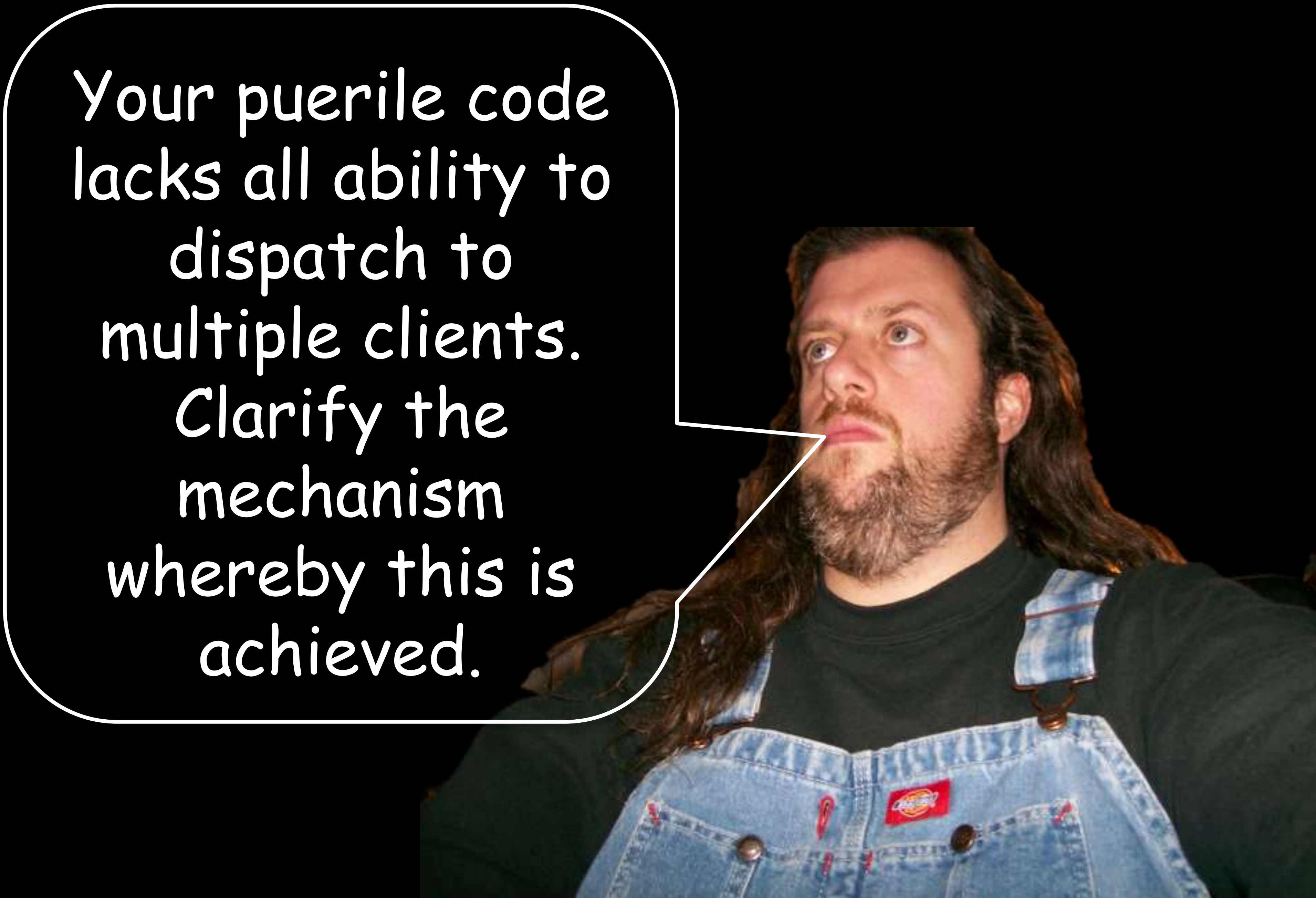
```
hPort, e = w32.NtAlpcAcceptConnectPort(
    hSrv,
    0,
    &oa,
    &basicPortAttr,
    context,
    pConnReq,
    nil,
    accepted,
)
```

Receive Loop - Client



```
// Reset the buffer, or the fields set in the previous recv will cause
// the message to be rejected - new ALPC messages should have most
// fields zeroed, as they are filled in by the kernel
clientMsg.Reset()
clientMsg.SetData([]byte(msg))

log.Printf("Client: Sending %s to handle %x", msg, hClientComm)
err := w32.NtAlpcSendWaitReceivePort(
    hClientComm,
    0,
    &clientMsg,
    nil,
    &clientMsg,
    nil,
    nil,
    nil,
)
if err != nil {
    log.Fatalf("Client: Recv Error: %v", err)
}
```

Note same buffer for send / recv…


Your puerile code lacks all ability to dispatch to multiple clients. Clarify the mechanism whereby this is achieved.

Context Attributes!

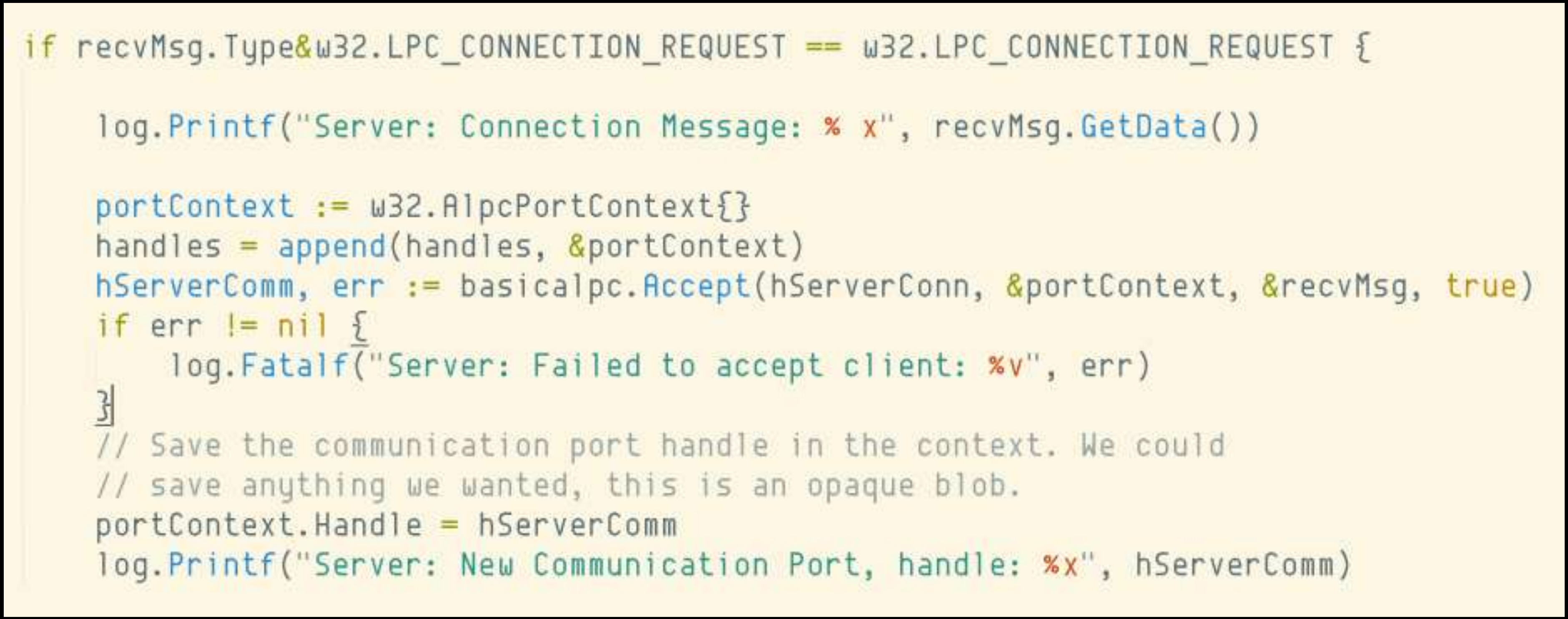
© Sven Micklish

Message Attributes

- Context - opaque struct
- Security
- Data View - share memory
- Handle - share handles

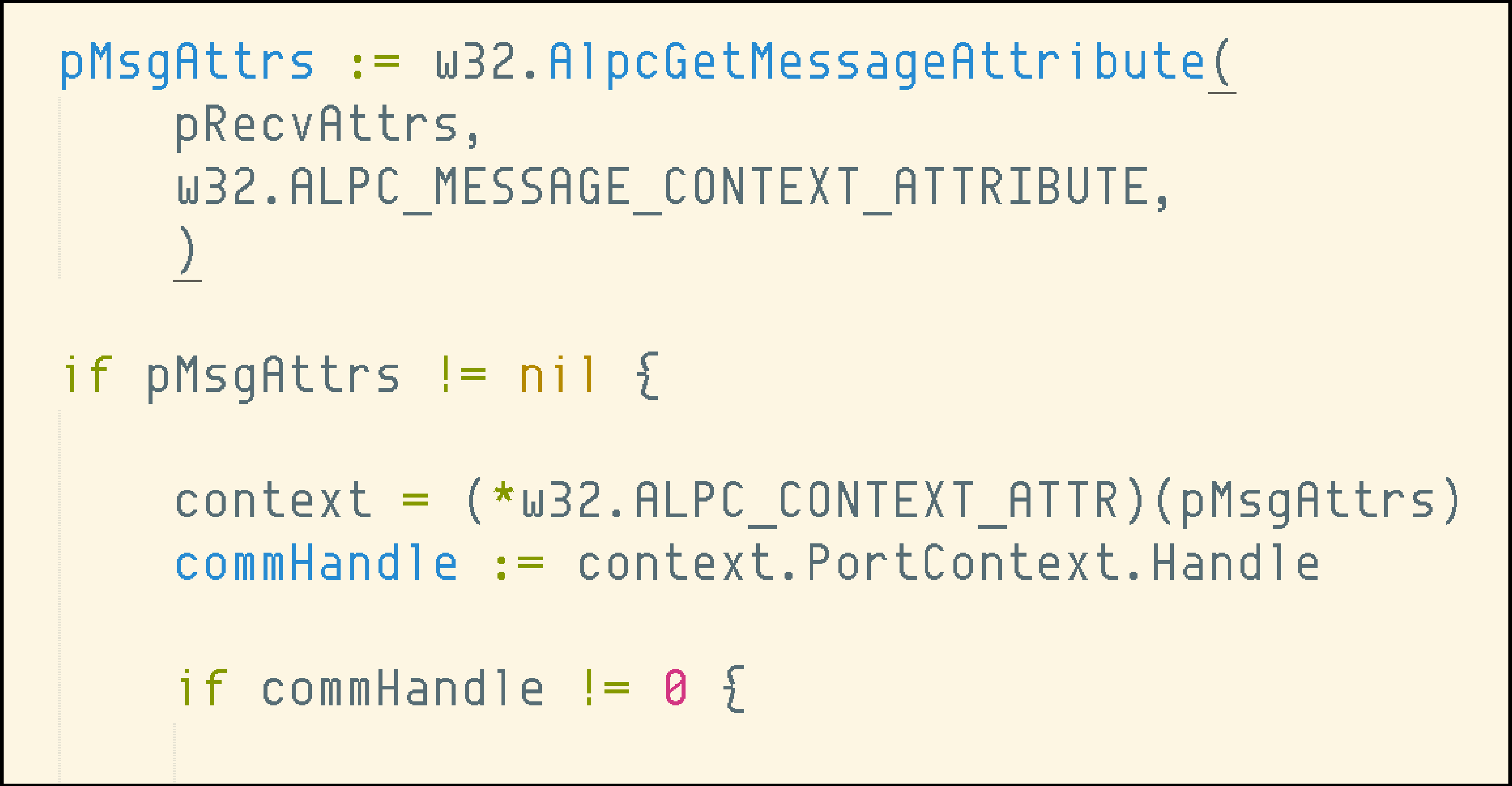
Secured “in transit” by the kernel ☹

Capture



```
if recvMsg.Type&w32.LPC_CONNECTION_REQUEST == w32.LPC_CONNECTION_REQUEST {

    log.Printf("Server: Connection Message: % x", recvMsg.GetData())

    portContext := w32.AlpcPortContext{}
    handles = append(handles, &portContext)
    hServerComm, err := basicalpc.Accept(hServerConn, &portContext, &recvMsg, true)
    if err != nil {
        log.Fatalf("Server: Failed to accept client: %v", err)
    }
    // Save the communication port handle in the context. We could
    // save anything we wanted, this is an opaque blob.
    portContext.Handle = hServerComm
    log.Printf("Server: New Communication Port, handle: %x", hServerComm)
```

Expose and Cast



```
pMsgAttrs := w32.AlpcGetMessageAttribute(
    pRecvAttrs,
    w32.ALPC_MESSAGE_CONTEXT_ATTRIBUTE,
    )

if pMsgAttrs != nil {

    context = (*w32.ALPC_CONTEXT_ATTR)(pMsgAttrs)
    commHandle := context.PortContext.Handle

    if commHandle != 0 {
```

FAILS?

COSEINC®

Solid Security. Verified.

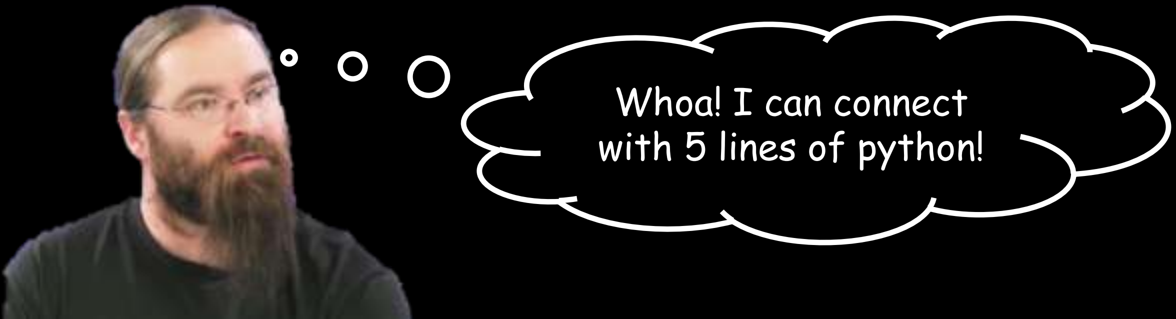
ALPC Programming Tips

- ntstatus.h - learn it, live it, love it
- Zero out reused buffers / headers
- Initialize struct Length fields
- Double check your flags
    - ALPC_PORFLG_*
    - ALPC_MSGFLG_*

Code - Go

- https://github.com/bnagy/w32
- https://github.com/bnagy/alpcgo
  - High level API
  - alpcechocli / alpcechosrv
  - alpcbridge (jsonrpc API)

Rust or Haskell would clearly have been a more felicitous choice.



WHAN RELEASE FUZZER??

COSEINC

Solid Security.Verified.

My TODOs

- Add Attribute support to Send()
  - Rating: EASY (NOW…)
- Add LRPC Parsing?
  - Rating: HARD
- Add MitM Fuzzing Proxy
  - Rating: NOT FOR RELEASE

Your TODOs

- Here’s the whole JSONRPC API:
  - Connect()
  - Send()
  - Close()

- Add radamsa and 15 lines of python
  - `Rating: TRIVIAL`

Instrumentation

- Userland Issues
  - "Normal" Exception instrumentation
  - RADAR
  - http://technet.microsoft.com/en-us/library/dd393057(WS.10).aspx
  - ProcDump
  - http://technet.microsoft.com/en-us/sysinternals/dd996900.aspx

Instrumentation

- BSOD Logging
  - Dump to disk
  - Check for dumps at startup
  - Dispatch to a triage server

My work here is done

Thanks:

- Alex Ionescu
- @miaubiz

Contact:

- ben@lrn2google.com
- @rantyben
- github.com/bnagy

COSEINC

Solid Security.Verified.

Questions?

COSEINC®
Solid Security. Verified.