Dinan Gunawardena
Microsoft Research Cambridge

MONITORING USING A WINDOWS BOX & HANDLING A DELUGE OF NETWORK DATA

Overview

- Windows Network Stack Overview
- Network Monitoring Scope
- Windows Monitoring Tools
- Additional Windows Monitoring Infrastucture

- Managing a large network Capture

Monitoring using a Windows box

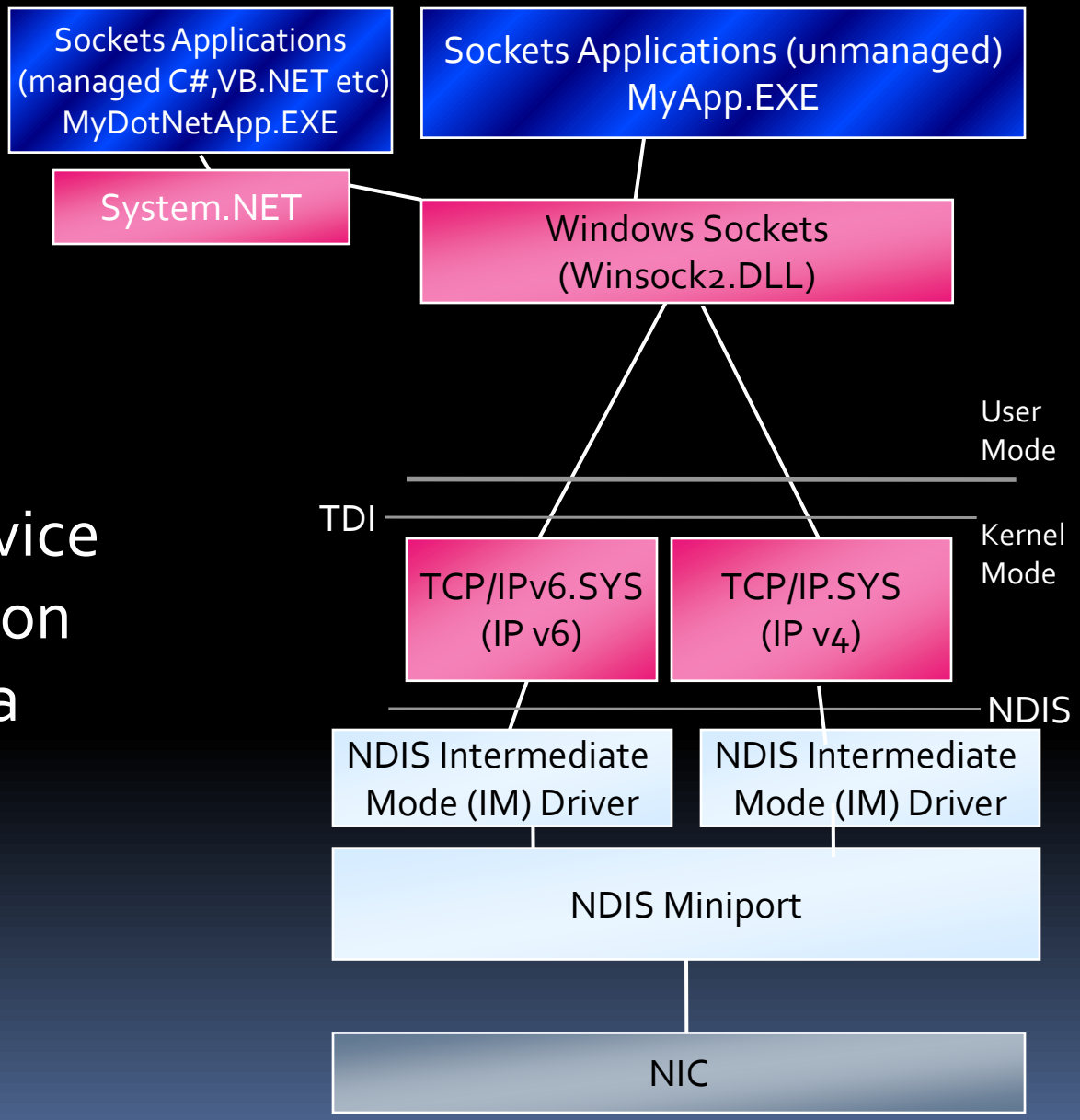
If you remember only one slide ☺

Task	Suggested Windows Approach
Figure out what is going on locally with your network interface	Run **NetMon** or **Ethereal** (both freely available on the web)
Experiment with / write a Ethernet based protocol	Start with **Windows Filterering Platform (WFP)** code samples at http://MSDN.microsoft.com or **RawEther sample** (PCUSA.com)
Do network I/O in a Windows driver	Try using **Windows Sockets Kernel (WSK)** http://MSDN.microsoft.com
Capture all the traffic on a subnet / Enterprise network	Learn **about router monitor ports** and consider writing your own **WFP / NetMon SDK / WinPCAP capture program** (start with the existing sample code)
Write network code for Windows	Download the Windows Driver Kit (WDK) from Microsoft.com

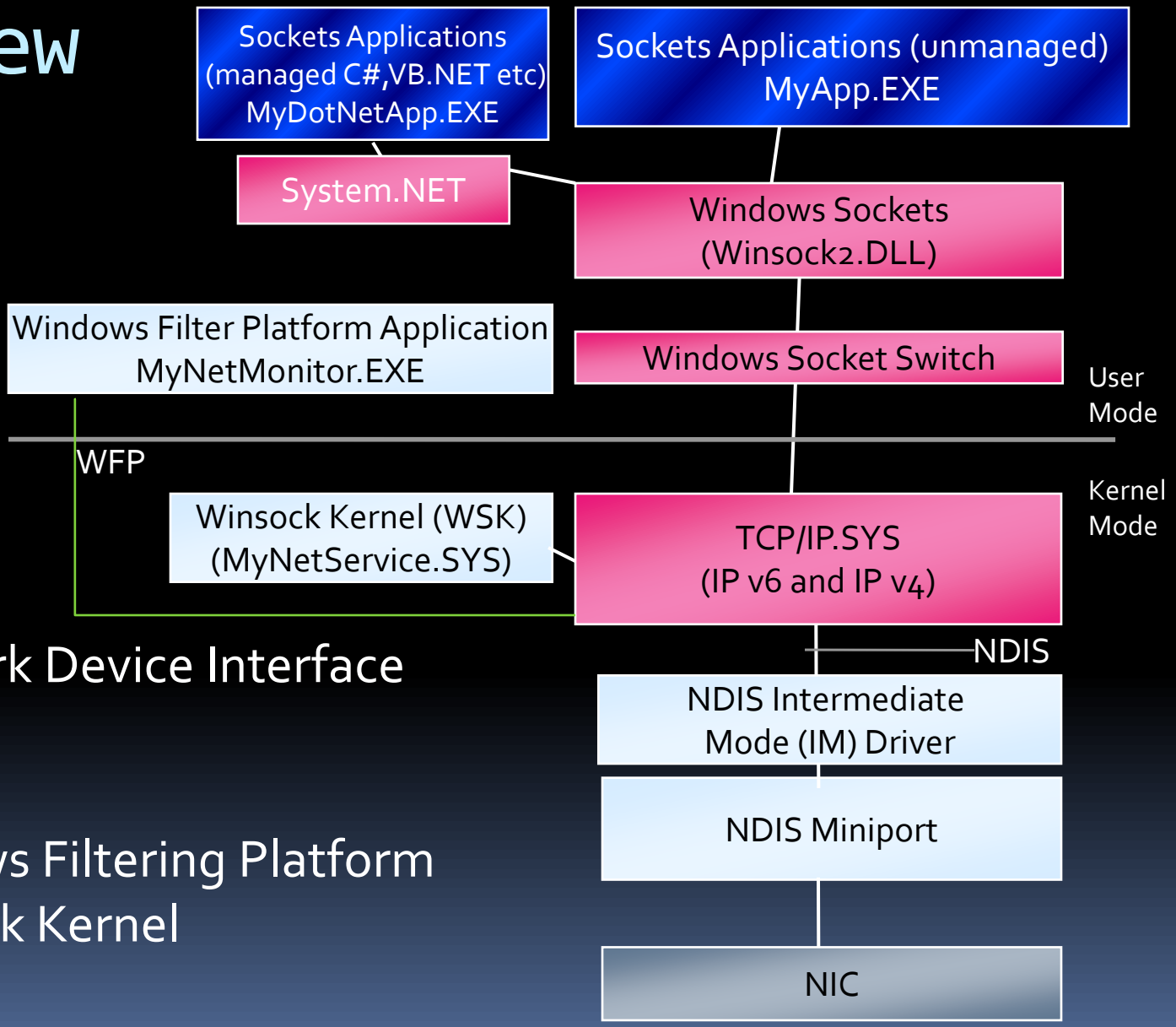
NetMon Demo

Windows XP Network Stack Overview

- Winsock
- TCP/IP stack
- NDIS – Network Device Interface Specification
- TDI – Transport Data Interface
- IPv6 and IPv4
- System.Net

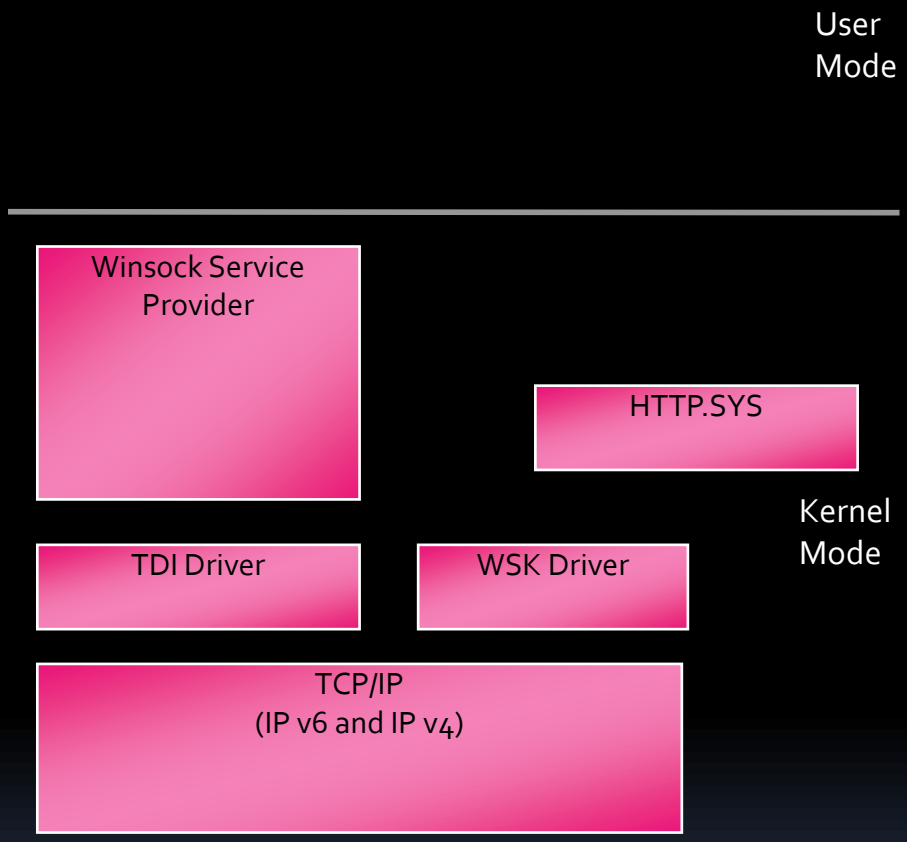


Windows Vista Network Stack Overview



- Winsock
- TCP/IP stack
- NDIS – Network Device Interface Specification
- IPv6 and IPv4
- WFP – Windows Filtering Platform
- WSK – WinSock Kernel
- System.Net

Transport Data Interface 1 (TDI)

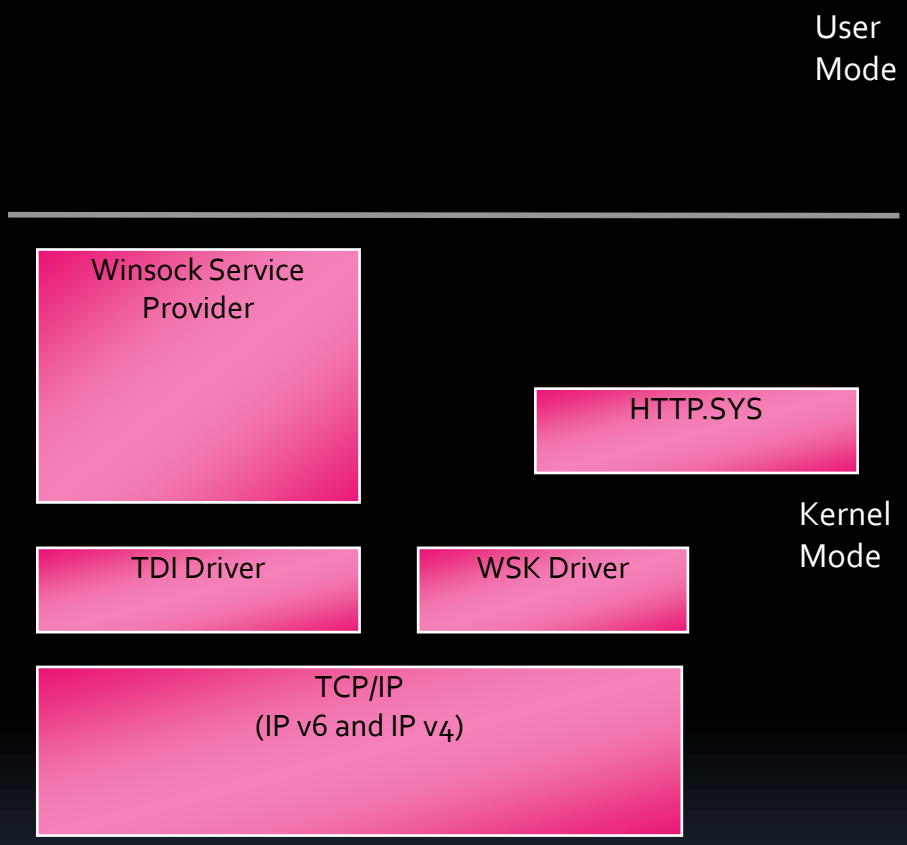


- Transport Data interface (TDI)
  - "Transport Drivers" e.g. TCP/IP and Kernel-mode users of transport drivers e.g. Windows Sock2 Kernel Mode Provider

Transport Data Interface 2 (TDI)

- **TDI Providers** : NDIS (Network Device Interface Specification) protocol drivers (aka "Transport Drivers")
  provide base implementation of network protocols e.g. TCP/IP.
  - Lower edge TDI providers interface with packet-oriented NDIS miniport drivers that communicate over the physical network
  - Upper edge TDI providers interact with their clients using the TDI interface.
- **TDI Clients** These are kernel-mode drivers that use the networking services of a TDI provider
  - A TDI client of Tcp can initiate or accept TCP connections and send or receive stream data within the kernel

WinSock Kernel (WSK)1

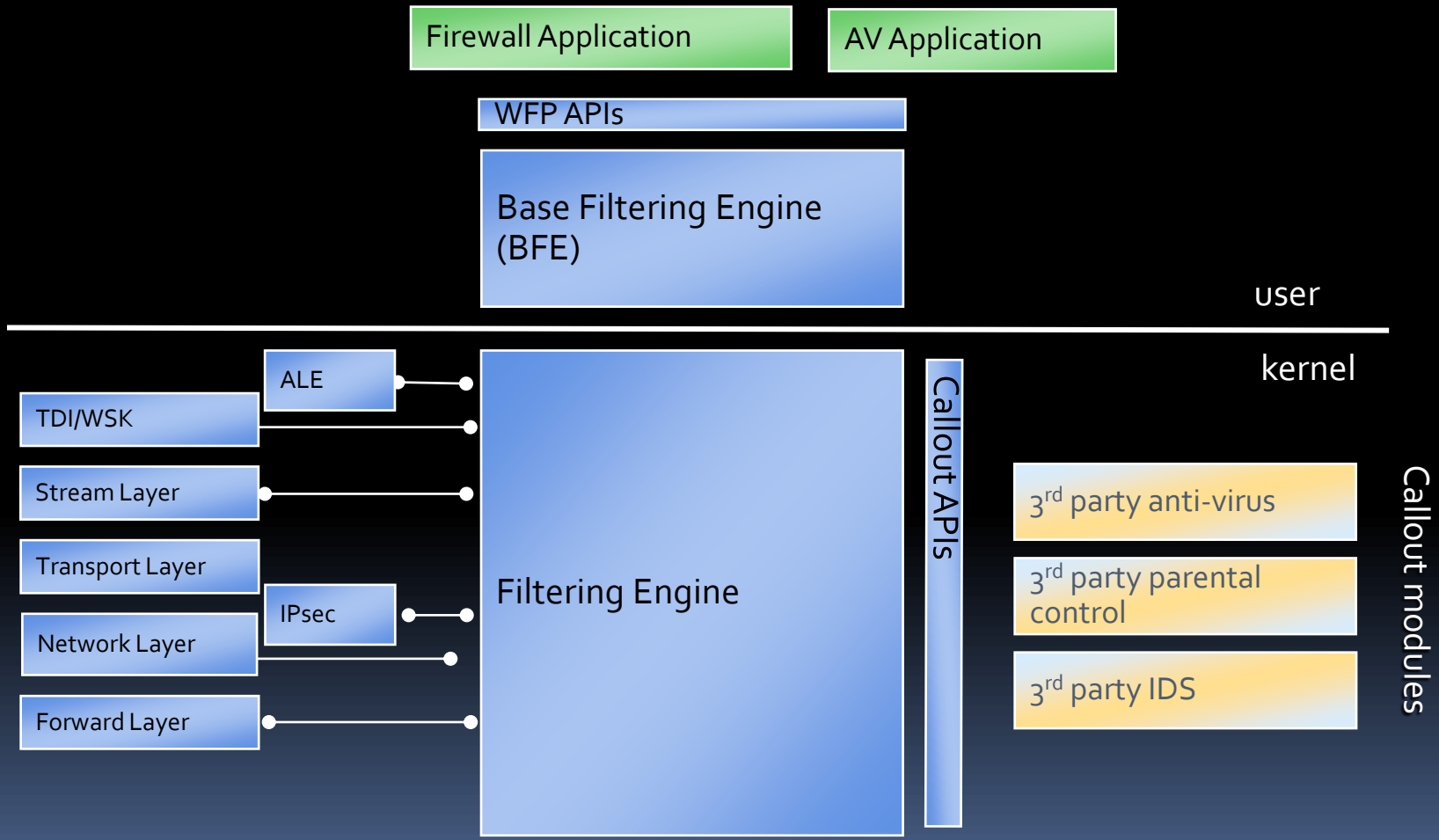


- Simple to use, Winsock2-like interface in kernel mode
  - Supercedes TDI

WinSock Kernel (WSK) 2

- Improve scalability and efficiency by improving on the performance and memory limitations of previous Network Programming Interfaces (NPI).
  - For example, WSK has improved socket creation performance and a smaller memory footprint per socket than past NPIs.
- Easy to port existing TDI clients to WSK.
  - Components such as http.sys (kernel mode HTTP handler) within Windows Vista have ported from TDI to WSK with ease
  - Supports IPv4 and IPv6
  - Handles transport discovery, load/unload and other intricacies

Windows Filtering Platform (WFP) Architecture



WFP Layers

Layers	Data Representations
Protocol specific	RPC, IKE
Stream/Data Layer	Datagram and streams
ALE (Application Layer Enforcement) Layers	Control events
Transport Layer	TCP/UDP
IP Packet Layer	Network layer traffic and local fragments
Forward Layer	Forwarded traffic
ICMP	ICMP error packets
Discard	Discarded/dropped packets

Benefits of WFP

- WFP can filter and secures (works with IPSEC) network traffic
- WFP supports both IPv4 as well as IPv6 traffic
- Integrated with hardware Offload capabilities in Windows Vista

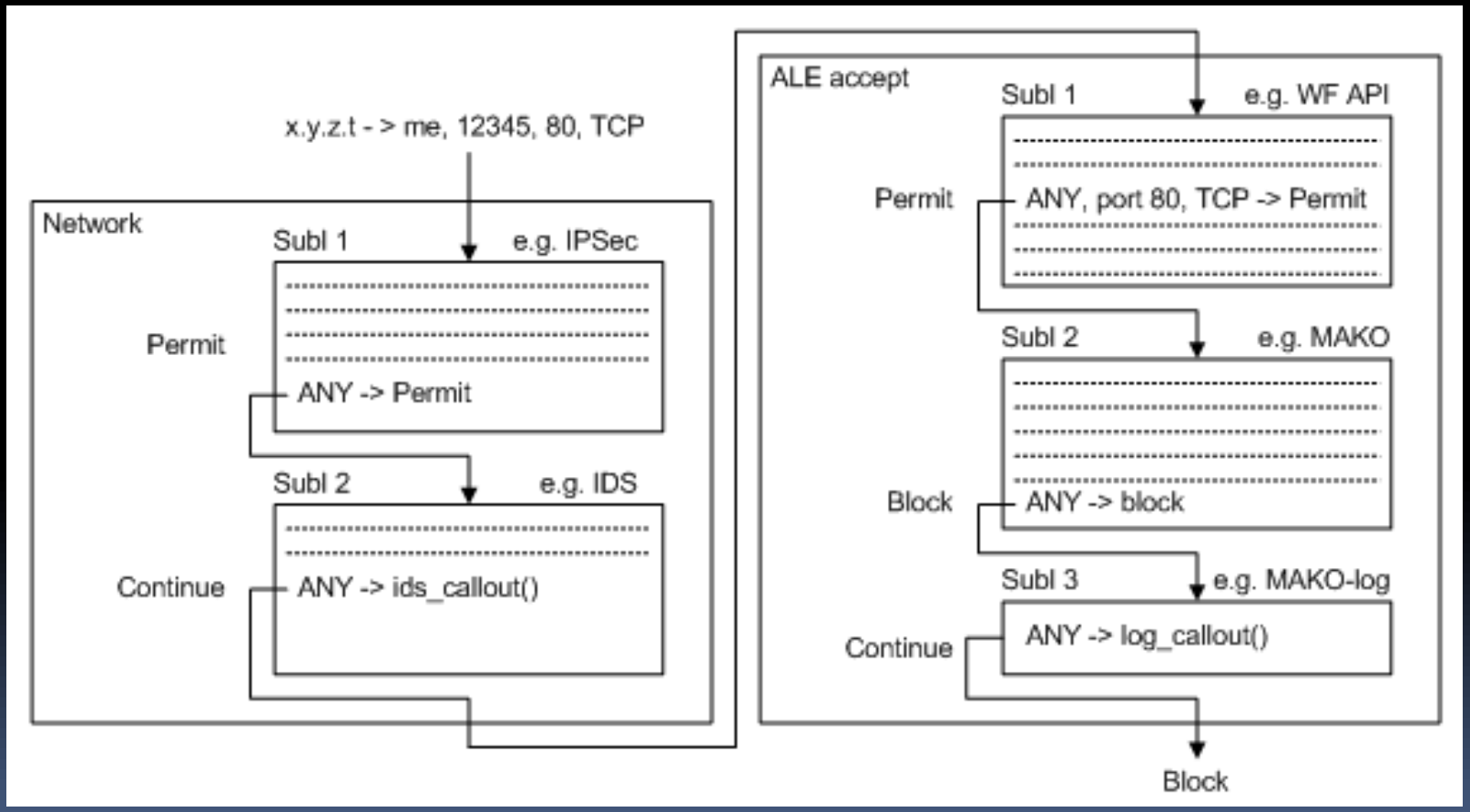
Extending WFP with Callouts

- A callout extends the capabilities of WFP
- Callouts can be registered at all layers
- Each callout has a unique GUID
- Callouts are used for
- Deep Inspection
- Packet Modification
- Stream Modification
- Data Logging
- Boot time security

For More Info:

- WFP development white paper
- http://www.microsoft.com/whdc/device/network/WFP.mspx

Filtering Model



Code Example 1

```
#include <fwpmu.h>

/// Creating a session and opening a handle to the engine
FwpmEngineOpen0(…);

FwpmTransactionBegin0(); /// Begin Transaction

FwpmSubLayerAdd0(…); /// Add a Sublayer

/// Add a Filter
FWPM_FILTER0 blockFilter; FWPM_FILTER_CONDITION0 tcpCondition;

blockFilter.layerKey = FWPM_LAYER_ALE_AUTH_RECV_ACCEPT_V4;
blockFilter.action.type = FWP_ACTION_BLOCK;
blockFilter.filterCondition = &tcpCondition;

tcpCondition.fieldKey = FWPM_CONDITION_IP_PROTOCOL;
tcpCondition.matchType = FWP_MATCH_EQUAL;
tcpCondition.conditionValue.uint8 = 0x06; /// TCP

FwpmFilterAdd0(…, &blockFilter, …);
```

Code Example 2 –Custom Callouts

```
/// Callout function: classify called whenever there is data to be
processed by callout
VOID NTAPI  classifyFn(
    IN const FWPS_INCOMING_VALUES0  *inFixedValues,
    IN const FWPS_INCOMING_METADATA_VALUES0  *inMetaValues,
    IN OUT VOID  *layerData, IN const FWPS_FILTER0  *filter,
    IN UINT64  flowContext, OUT FWPS_CLASSIFY_OUT0  *classifyOut);

/// calloutKey holds the GUID that uniquely identifies the callout
typedef struct FWPS_CALLOUT0_ {
  GUID  calloutKey;   UINT32  flags;
  FWPS_CALLOUT_CLASSIFY_FN0  classifyFn;
  FWPS_CALLOUT_NOTIFY_FN0  notifyFn;
  FWPS_CALLOUT_FLOW_DELETE_NOTIFY_FN0  flowDeleteFn;
} FWPS_CALLOUT0;

// Add a new Callout
FwpmCalloutAdd0(…, (FWPM_CALLOUT0*) callout, …);

// Register a Callout with the filtering engine
FwpsCalloutRegister0(…, (FWPS_CALLOUT0  *) callout, …);
```

Network Monitoring Scope

- Level of Capture
  - IP/Ethernet
    - Captures all the data of higher layers
  - At End System
    - IP SEC mitigation, load balancing etc.
  - Non-aggregate
    - Don't want to limit what you can do with the data
  - Unfiltered traffic
    - Some security issues
- Not covered
  - Capture at Network Infrastructure (e.g. NetFlow)
  - Non-software solutions

Windows Monitoring Tools

- NetMon2 – custom filters…
- Ethereal (/ Tethereal) WinPCap – source available, buffering / perf issues
- www.SysInternals.com tools: TDI Mon, TCPView
- Custom Tools- rolling your own ☺
  - User Mode (trade-off: simple programming environment for performance)
    - Raw Sockets: TCP limitations (an aside)
    - NDIS UIO - In Windows Dev Kit (WDK) pull up NDIS packets to User Mode used by Wireless Zero Config user mode service – source available in WDK
    - RawEther – (PCUSA.com) Send/Receive NDIS packets from User Mode – source available
  - Kernel Drivers
    - Network Device Interface Specification (NDIS) common interface to NIC drivers
    - Intermediate Mode (IM) e.g. Firewalls - Passthru driver sample
    - MiniPort e.g. NIC drivers, SCSI miniport (lowest level wrapper for a class of drivers)
    - **Vista: Better to use WinSock Kernel (WSK) / Windows Filter Platform (WFP)**

Event Tracing for Windows (ETW)

- Many, many system components wrapped
  - TCP/IP connection establishment etc.
  - OS Context Switches
  - Disk IO events
  - IIS (web server) events
  - … And many more
- Use **PerfMon** if you just want to understand local performance
  - e.g. How long is the disk write queue

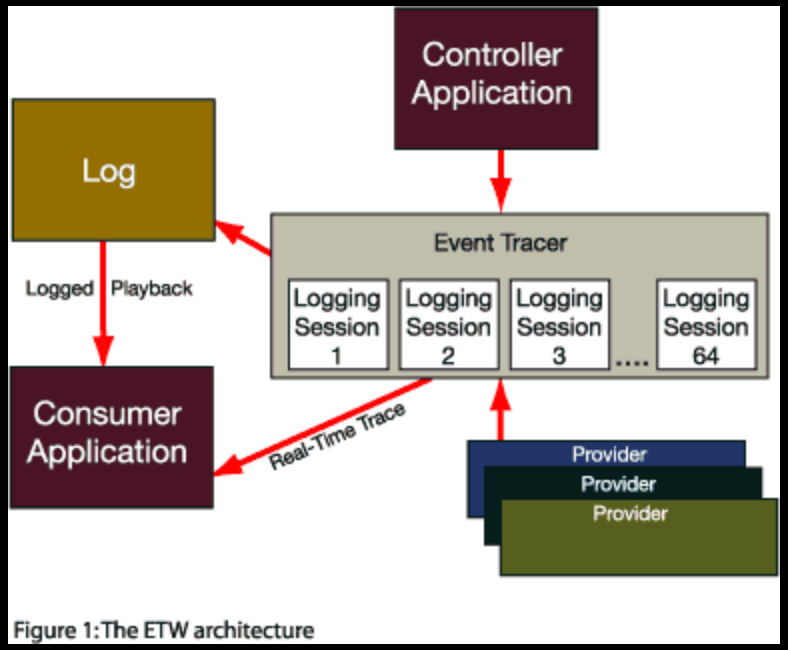


Figure 1:The ETW architecture

Event Tracer Timestamp Information
- ETW time of the event
- process ID under which the event occurs
- thread ID under which the event occurs
- user-mode CPU time
- kernel-mode CPU time

Additional Windows Monitoring Infrastucture

- NETIO debug
  - New Vista TCP/IP stack internal debugging
- Link Status Events OIDs (Object IDentiers)
  - WFP subsumes much of this
- Native WiFi
  - IEEE 802.11 upper MAC functionality, lower MAC and PHY management + Windows STA / AP service

Handling a deluge of network data

Managing a large network Capture

(6TB of data in 14 days, 300 Hosts, 3 Capture PCs, 3 Cisco SPAN ports, 50+ backup tapes)

- Hardware requirements
- Software Requirements
- Meta Data
- Privacy Issues
- Security
- Manpower Issues
- Post Processing

Hardware requirements

- CPU / Chassis
  - RAM – don't want it swapping!
  - CPU – capturing should not be too CPU intensive
  - KVMs – multiple capturing PCs, single console…
- Network Interface
  - Speed – 1000Mbps NIC even if network is 100Mbps
  - Offload support – CPU cost
  - Load balancing / redundancy – helps deal with bursts, failures
  - Interrupt Moderation… But issues with timestamps in packets
- Storage
  - Reliability – RAID 5
  - Capacity
  - Performance – multi disk arrays, eSATA, Firewire –Perf not at cost of Reliability
  - Backup – offsite / disaster proof / reliable
- Router/Network infrastructure
  - SPAN / Monitor ports
  - Fibre taps
  - Router performance impact

Software Requirements

- Reliability
  - Soak test
  - Dry runs
  - Test sample output
- Performance
  - Test under load – bursts, sustained loads
  - Turn-off Anti-Virus, search indexing service etc.
- Time Sync – NTP etc
  - Important for merging data sets

Meta Data

- DNS / WINS
  - Zone transfer records
- DHCP data
- Router config / Network config
- Maintenance scheduling
- Back-up this meta-data
  - It is as, if not more important than the captured data ☺

Privacy Issues

- Personally Identifiable Information (PII) and Legal concerns
  - Implications: may only be able to capture packet headers
- IP Packet payload discard
  - How much can you discard
  - Capture snap length may limit usefulness of data
- Anonymising IP 5-tuple
  - Depending on how paranoid you have to be

Security Issues

- Access control to captures
  - Acceptable Usage Policy (AUP)
- Physical security of storage
- Dealing with encryption
- Publishing concerns

Manpower Issues

- Managing capture is 24x7 job
  - Automation
  - Backup monitoring personnel
- Outages happen

Post Processing

- Make copies before post processing / discarding data

- Process…

1. Raw -> backup
2. Validity check
3. Correct broken files
4. De-duplicate data
5. Process for packet data + generate NetFlow-like records

- Lastly… Make meticulous notes
  - Time of events
  - Nature of logging – network info / configuration
  - Put processing scripts/tools (& results!) under revision control

Microsoft®

Questions?

Code Example 1

```
Copyright (c) Microsoft Corporation. All rights reserved.
…
#include <fwpmu.h>

   /// Creating a session and opening a handle to the engine
   HANDLE engineHandle = 0;
   FWPM_SESSION0 session;
   ZeroMemory(&session,sizeof(session));

   session.displayData.name = L"Snipit Session";
   session.displayData.description = L"Session created by
Snipit.exe";

   status = FwpmEngineOpen0(0,
                            RPC_C_AUTHN_DEFAULT,
                            0,
                            &session,
                            &engineHandle);

/// Begin Transaction
   FwpmTransactionBegin0(engineHandle);
```

Code Example 2

```
   /// Add a Sublayer
   FWPM_SUBLAYER0 sublayer;
   ZeroMemory(&sublayer,sizeof(sublayer));

   UuidCreate(&sublayer.subLayerKey);
   sublayer.displayData.name = L"Snipit Sublayer";
   sublayer.displayData.description = L"Sublayer added by
Snipit.exe";
   sublayer.weight = 1;

   status = FwpmSubLayerAdd0(engineHandle, &sublayer, 0);
…
```

Code Example 3

```
/// Add a Filter
   FWPM_FILTER0 blockFilter;
   ZeroMemory(&blockFilter,sizeof(blockFilter));
   FWPM_FILTER_CONDITION0 tcpCondition;
   ZeroMemory(&tcpCondition,sizeof(tcpCondition));

   UuidCreate(&blockFilter.filterKey);
   blockFilter.displayData.name = L"Snipit TCP block filter";
   blockFilter.displayData.description = L"Filter added by
Snipit.exe";
   blockFilter.layerKey = FWPM_LAYER_ALE_AUTH_RECV_ACCEPT_V4;
   blockFilter.action.type = FWP_ACTION_BLOCK;
   blockFilter.subLayerKey = sublayer.subLayerKey;
   blockFilter.numFilterConditions = 1;
   blockFilter.filterCondition = &tcpCondition;

   tcpCondition.fieldKey = FWPM_CONDITION_IP_PROTOCOL;
   tcpCondition.matchType = FWP_MATCH_EQUAL;
   tcpCondition.conditionValue.type = FWP_UINT8;
   tcpCondition.conditionValue.uint8 = 0x06; /// TCP

   status = FwpmFilterAdd0(engineHandle, &blockFilter, 0,
&blockFilter.filterId);
```

Code Example 4 –Custom Callouts

```
  /// Callout function: classify called whenever there is data to be
processed by callout
VOID NTAPI  classifyFn(
    IN const FWPS_INCOMING_VALUES0  *inFixedValues,
    IN const FWPS_INCOMING_METADATA_VALUES0  *inMetaValues,
    IN OUT VOID  *layerData, IN const FWPS_FILTER0  *filter,
    IN UINT64  flowContext, OUT FWPS_CLASSIFY_OUT0  *classifyOut);

/// calloutKey holds the GUID that uniquely identifies the callout
typedef struct FWPS_CALLOUT0_ {
  GUID  calloutKey;   UINT32  flags;
  FWPS_CALLOUT_CLASSIFY_FN0  classifyFn;
  FWPS_CALLOUT_NOTIFY_FN0  notifyFn;
  FWPS_CALLOUT_FLOW_DELETE_NOTIFY_FN0  flowDeleteFn;
} FWPS_CALLOUT0;

// Add a new Callout
DWORD WINAPI FwpmCalloutAdd0(HANDLE engineHandle, const
FWPM_CALLOUT0* callout, PSECURITY_DESCRIPTOR sd,  UINT32* id);

// Register a Callout with the filtering engine
NTSTATUS NTAPI  FwpsCalloutRegister0(IN OUT void  *deviceObject,
    IN const FWPS_CALLOUT0  *callout, OUT OPTIONAL UINT32
*calloutId);
```

Windows Network Stack Overview

- Winsock
- TCP/IP stack
- NDIS – Network Device Interface Specification
- IPv6 and IPv4
- WFP – Windows Filtering Platform
- WSK – WinSock Kernel
- System.Net
- Http.sys + WinHttp / WinINet
- QoS
- IPSec