Maven Windows provides two sets of subsystems used to implement protection layers for processes and network access control. Each subsystem consists of a driver part and a user application part as shown in Figure-1. Process filtering restricts the creation of processes to a list of approved applications using mavenMinifilter driver and mavenFilefilter user application. Network filtering prohibits outbound network traffic targeted to blacklisted sites using winDivertDrv and mavenNetfilter. Both protection subsystems provide logging facilities to disk that can be used by the Control Layer to identify threats. Status and command messages are directed to the appropriate subsystems from either a client or server socket through the mavenClient application.

Network Filtering

**Figure 1**

Filtered packet

response

Command

mavenFilefilter

mavenMinifiter

winDivertDrv

mavenNetfilter

mavenClient

Server  
Socketicsrms

Client  
Socketicsrms

Metricsicsrms

Network Callouts

Filesystem Callouts

Status/alarmsrms

Packet log

Packetlog.txt  
webfilter.txt

Unfiltered packet  
Status

Process log

Process Filtering

config

disk

# mavenMinifilter

This driver implements a mini-filter service for the monitoring and control of processes in the target machine. Filtering restricts the initialization of processes using a white-list. This list is read from the file-system at startup. It defines the set of applications that are permitted to run. An attempt to initialize an application that is not found in that list is prohibited and the event is logged to memory.

The filer white-list file can be found in folder C:\DriverTest. This folder is blacklisted and is therefore not accessible when in filter mode. The driver files may be found in the same folder. The driver must be installed using the driver inf file.

The driver supports a "Learn" mode that is used to record applications that normally run. This can be used to construct a process permissions white-list for later use. If this configuration file is not found during driver initialization, the driver defaults to learn mode and begins to construct a new memory-resident white-list. Filtering is not performed while in learn mode.

The driver works in cooperation with a user application that manages driver state changes, controls uploading and downloading of white-list files, and captures and reports filter events.

The driver initializes at boot time and initially enters Boot mode wherein it logs filter exceptions but doesn’t prohibit file accesses until the Windows boot sequence completes. It uses an initialization file that is read during the initialization sequence to obtain it’s white-list and a message port that is used for reporting events during the filtering process. Entries in this white list identify processes that are permitted to run when in Filter mode. Configuration white-list files are user roll-based, restricting users to applications that are consistent to their organizational needs.

An example configuration white list is below.

000000,\Device\HarddiskVolume2\Windows\System32\drivers\null.sys  
000001,\Device\HarddiskVolume2\Windows\System32\drivers\beep.sys  
000002,\Device\HarddiskVolume2\Windows\System32\drivers\BasicDisplay.sys  
…

A white-list can be tagged to indicate that it is not yet complete and should not be used for filtering. Tagging is done by putting an asterisk at the beginning of the file as shown below. When tagged, the white-list file will be read into memory at boot-time and the driver will enter Learn mode instead of Boot mode on startup. The boot sequence performed by Windows is non-deterministic. Tagging a white-list allows the administrator to successively boot the system and capture system behavior to grow a white-list.

\*000000,\Device\HarddiskVolume2\Windows\System32\drivers\null.sys  
000001,\Device\HarddiskVolume2\Windows\System32\drivers\beep.sys  
000002,\Device\HarddiskVolume2\Windows\System32\drivers\BasicDisplay.sys  
…

## Interface

The driver uses a communications port that is connected to the mavenFilefilter application. Command messages are received by the driver and responses are returned to the mavenFileFilter user application.

# mavenFilefilter

The mavenFilefilter user application initializes when the user logs in. It connects to the mavenMinifilter driver and polls it for status at a periodic interval. Status returned includes the current driver state and the number of events that are logged. If there are logged events, the application reads them from the driver and logs them to disk. The application sends status to the control layer at a periodic rate (programmable).

When events are detected and blocked by the driver, they are read from the driver and reported upstream through mavenClient.

## Interface

The application interfaces with the driver using a kernel message port. It connects to the mavenClient application using a client socket and logs events to a disk file. Events include error log messages and blocked process event logs.

# winDivertDrv

This driver intercepts packets and selectively forwards them to a user app that filters and logs them. It is based on the open-source WinDivert packet driver linked below. Changes are minimal and consist of modifications required to build with Visual Studio rather than the command line. No functional changes were made.

<https://reqrypt.org/windivert.html>

The driver is configured to initialize on-demand. This is initiated by the mavenNetfilter application. Since packet filtering is performed in the user application, initializing earlier would not be beneficial. The driver and user application must both be running before filtering and logging can commence. Once filtering begins, the driver counts packets by packet type, tcp, udp, and icmp. It inspects packets for a match with rules passed during initialization and forwards any that match to the user application where packets are either blocked or returned for processing by the network stack.

## Interface

The driver API is defined in support documentation linked above. It consists of a series of messages that are handled by windivert IOCTL. Of interest is the windivert\_install\_callout which receives an initialization string used by the driver to create driver filter rules. This is described in paragraph 7 of the linked document.

Packets that conform to driver filter rules are passed through a message port to the mavenFilter application where they are processed for blacklisting. If they are not blacklisted, they are returned to the driver and passed through the stack.

# mavenNetfilter

This application is a modified version of the winDivert Dll and winDivert webfilter example distributed with the winDivert installation. Changes include the following.

1. Packets sent are logged to a file.
2. Metrics are collected for the number of packets sent and received.
3. Alarm thresholds are checked for various packet (not yet implemented)

The driver uses two configuration files for filtering, ipV4BlackList.txt and urlBlackList.txt. These files must be placed in folder C:\maven.

During initialization, it loads and starts the driver and sends it preliminary filter rules as shown below. These rules define packets of interest that should be forwarded for blacklisting.

driverHandle = WinDivertOpen(

"icmp || " /\* ICMP \*/

"udp || " /\* UDP \*/

"!loopback && " /\* No loopback traffic \*/

"ip && " /\* IPv4 \*/

"!tcp.DstPort == 65000 && " /\* not maven port \*/

"!tcp.DstPort == 5170 && " /\* not fluentd port \*/

"!tcp.DstPort == 3389 && " /\* not RDP port \*/

"!tcp.DstPort == 4022 && " /\* not remote debugger port \*/

"tcp.PayloadLength > 0", /\* TCP data packets only \*/

WINDIVERT\_LAYER\_NETWORK, priority, 0

);

## Interface

Communication between the driver and application uses Windows overlapped IO to pass messages between them using IOCTL functions implemented in the DLL. It connects to the mavenClient application using a client socket and logs events to a disk file.

# mavenClient

The mavenClient user application acts as a message traffic-cop, collecting and distributing messages among the various applications. This application can be found in folder C:\maven. This is the root application for the filter subsystem. Starting this application initiates a sequence that starts the remaining apps. It must be started with Administrator rights. It provides three server ports for the two filter applications and one external control application. It provides one client socket that connects to a fluentd data collector application. The fluentd data collector must be running before this app starts. When initializing, it starts the two filter applications. These connect to their respective server ports. It connects to the fluentd port. It provides a server port that can optionally be connected by an external control application, Admin. The three server port connections must be identified when their socket connection is accepted. Each client sends a connect message that includes an identifier string that is used to select the appropriate socket index. This message is received and processed by thread ServerportThread. One thread is spawned for each connection. Once identified, messages can be routed appropriately.

Each port has a callback registered for processing of received messages. Functions parseNetFilterMsg and parseFileFilterMsg handle received messages from their respective filters. Function parseConsoleMsg handles messages from the Admin port.

NOTE: As currently implemented, a separate thread is used to provide the client socket to the fluentd app. The thread makes a client connection to mavenClient and a client connection to fluentd. This uses a fourth client port in mavenClient. This thread will be eliminated in the next iteration.

# Interface Messages

## Process Filtering Command/Response messages

Each command message has an enumerated command followed by an optional payload. Similarly, responses have a response status followed by an optional payload. Commands and their expected response are listed below.

1. mavenModeSetIdle  
   This message sets the driver to idle mode. The response is always mavenSuccess.
2. mavenModeSetLearn  
   This message sets the driver to Learn mode. In learn mode, the driver will filter applications and add to the current configuration white-list. The response is always mavenSuccess.  
   Note: this doesn’t update the file on disk.
3. mavenModeSetFilter   
   This message sets the driver to Filter mode. If the current configuration white-list is empty, the driver will not transition state and will return an mavenError. Otherwise, it will transition and respond with mavenSuccess.
4. mavenModeSetReset   
   This message sets the driver to idle mode and clears the current configuration white-list. The response is always mavenSuccess.
5. mavenGetVersion   
   This message doesn’t change the driver state. The response is always mavenSuccess. It returns the current driver version in the payload field.
6. mavenReadConfig   
   This message doesn’t change the driver state. If the current driver state is not Idle, the message is rejected and the response message status is set to mavenError. Otherwise, the response message status is set to mavenSuccess and it parses the command message payload to obtain a single index to the current configuration white-list. It returns the contents of that indexed record in the payload.
7. mavenWriteConfig   
   This message doesn’t change the driver state. If the current driver state is not Idle, the message is ignored, and the response message status is set to mavenError. Otherwise, it parses the command message payload to obtain a single index to the current configuration white-list. Each record index must be received in succession. If this is the first record, the current white-list must be empty. It saves the payload to the indexed record in the current configuration white-list.
8. mavenGetStatus   
   This message doesn’t change the driver state. The response is always mavenSuccess. It returns the current operating state in the payload field.
9. mavenGetLog   
   This message doesn’t change the driver state. The response is always mavenSuccess. It parses the command message payload to obtain an index to the current log. It returns the contents of that indexed record in the payload.

## mavenFilefilter/Fluentd Messages

The filefilter status message includes the current operating mode, the number of logged messages, time since startup in seconds, and the total number of events logged since startup.

FileFilter Success Mode = Idle, Log Size = 0 Time= 25.09 totalEvents = 6

When events are detected, the status message is preceeded by an event message for each event. Each is sent to the Admin and fluentd ports. This message includes the sequential event number, the user name of the user who attempted the blocked operation, and the full path to the blocked app.

FileFilter event(8) - Admin \Device\HarddiskVolume2\Windows\System32\ApplicationFrameHost.exe

## Network Filtering Command Messages

1. Filter  
   This command causes the network filter to enter filter mode.
2. Idle  
   This command causes the network filter to enter idle. Packets are not filtered.
3. Exit  
   This command is not implemented
4. Status  
   This command causes a status message to be sent.

## winDivertDrv Packet Messages

Messages from the filesystem filter include only outgoing packets. These are raw packets from the driver. If the packet is allowed, it is reinjected to the driver. If the packet is blocked, a close-connection message is formatted and returned to the driver instead of reinjecting the original message.

## mavenNetfilter/Fluentd Messages

Status is reported from the netfilter upstream at a periodic rate as shown below. This message includes event time in seconds and a count of various packet types.

NetFilter Status Time= 130.65, icmpCountInbound 0, icmpv6CountInbound 49, tcpCountInbound 10821, udpCountInbound 307

icmpCountOutbound 3, icmpv6CountOutbound 69, tcpCountOutbound 13591, udpCountOutbound 304 eventCount=0, mode=filter

Blocked packets are reported upstream when encountered as shown below.

NetFilter BLOCKED PACKET (517 bytes) -------------------------------

Packet [Timestamp=132.6346, Direction=outbound IfIdx=4 SubIfIdx=0 Loopback=0]

IPv4 [Version=4 HdrLength=5 TOS=0 Length=517 Id=0x165D Reserved=0 DF=1 MF=0

FragOff=0 TTL=128 Protocol=6 Checksum=0x0000 SrcAddr=10.0.2.15 DstAddr=192.185.23.189]

TCP [SrcPort=49788 DstPort=80 SeqNum=2831138216 AckNum=646976002 HdrLength=5

Reserved1=0 Reserved2=0 Urg=0 Ack=1 Psh=1 Rst=0

Syn=0 Fin=0 Window=64240 Checksum=0xE67C UrgPtr=0]

45000205165D4000800600000A00020FC0B917BDC27C0050A8BFBDA8269012025018FAF0E67C0000

474554202F20485454502F312E310D0A486F73743A207369656765746563686E6F6C6F676965732E

636F6D0D0A557365722D4167656E743A204D6F7A696C6C612F352E30202857696E646F7773204E54

2031302E303B20574F5736343B2072763A35392E3029204765636B6F2F3230313030313031204669

7265666F782F35392E300D0A4163636570743A20746578742F68746D6C2C6170706C69636174696F

6E2F7868746D6C2B786D6C2C6170706C69636174696F6E2F786D6C3B713D302E392C2A2F2A3B713D

302E380D0A4163636570742D4C616E67756167653A20656E2D55532C656E3B713D302E350D0A4163

636570742D456E636F64696E673A20677A69702C206465666C6174650D0A436F6F6B69653A205F67

613D4741312E322E313339353935363532322E313532323131363136323B205F5F736D5649443D66

66323539653638666364356239313864323062336163663638396366613336653031383736616431

62323366613930616662393837326663646263373061393B205F5F736D546F6B656E3D556D334939

7162547331376E4C4A614942745276784E725A0D0A436F6E6E656374696F6E3A206B6565702D616C

6976650D0A557067726164652D496E7365637572652D52657175657374733A20310D0A0D0A

# Log Files

Status and events collected at runtime is logged to disk. Three log files are created at runtime and can be found in folder C:\maven. These files are overwritten when the filter application starts. If you want to examine log files from the previous run, copy them off before starting mavenClient.

Windows caches files. Physical contents of these three files lag behind processing even though they are created as write-through.

## filefilter.txt

The filefilter file contains a log of recorded status end event messages collected during runtime.

## packetLog.txt

Packets forwarded to the netfilter application are logged to this file. Both blocked and passed packets are logged. Blocked packets and those that are forwarded to the network stack are tagged as such in the first line of each log message. This file can be quite large.

## webfilter.txt

This file contains a record of netfilter status messages collected during runtime.

# Performance

Performance testing was done with a virtualBox instance.

Boot time with filtering on – 71 seconds  
Boot time without filtering on – 68 seconds

Launch firefox with filtering on – 5 seconds  
Launch firefox without filtering on – 4 seconds