A WINDOWS FILE FILTER DRIVER FOR DATA LEAK INCIDENT HANDLING

Fotis Ailianos

Supervisors: G. Xylomenos, G. Polyzos

MSc Computer Science, AUEB 2010

Data Leak Incident Handling

- Companies lose classified files
- We have to prevent this
- At least monitor the events
- Causes
  - Worms
  - USB disks, files@work maybe safe but what about home
- We need to find where the leak started
- Why happened, virus, theft, user ignorance?
- We have to log everything & try to prevent the leak

Outline

- The Windows Driver Model
- IRP & IRP paths
- Details about our implementation
  - Monitoring
  - Pattern detection
  - File Marking
  - Policy
- Demo
- Conclusions

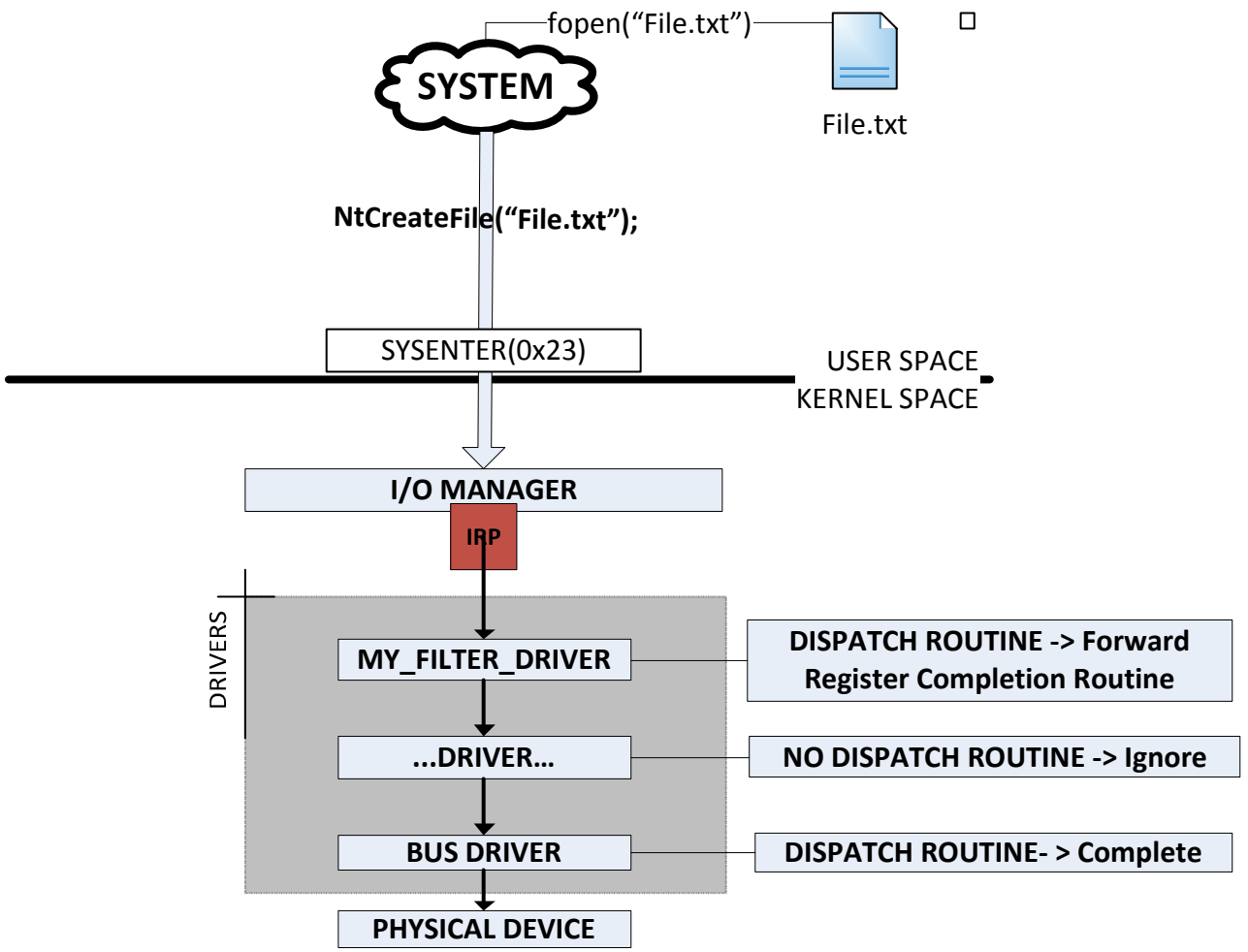
Windows Driver Model

- UserSpace vs KernelSpace
- Types of drivers
  - Function Drivers
  - Filter Drivers
  - Bus Driver
- Layered driver model
  - Each driver is stacked, the location is specific
  - Filter driver have specific position too, ALTITUDES
  - Communication by IRPs (I/O Request Packet)

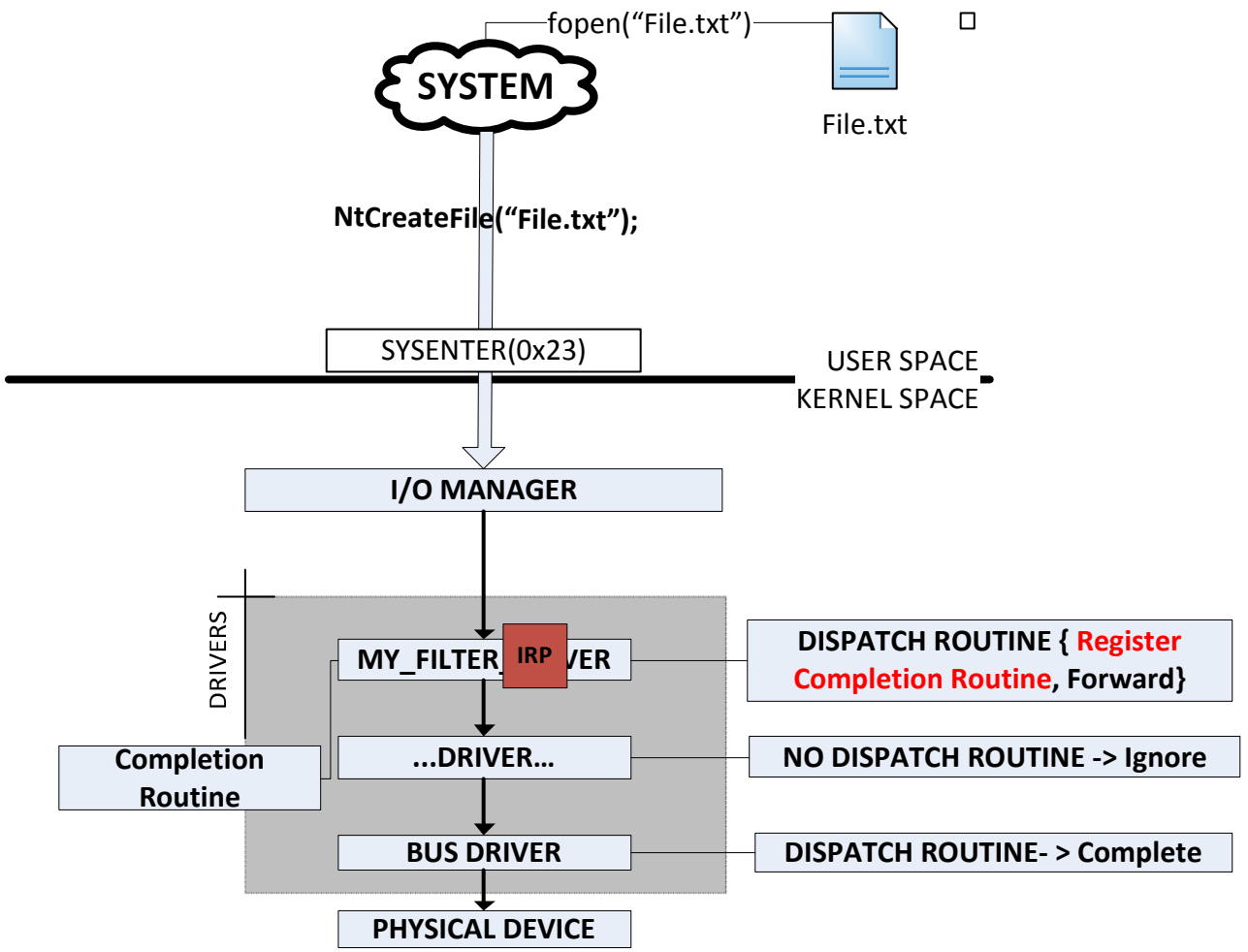
I/O Request Packet (IRP)

- Basic data block passing through drivers
- “Inter-driver” communication
- Characterized by type of Major Functions
  - **IRP_MJ_CREATE**
  - **IRP_MJ_READ, IRP_MJ_WRITE**
- Complex struct containing information about I/O operation
- Drivers register dispatch routines for each IRP type
- I/O manager allocates IRP , drivers forward IRPs accordingly, until completions
- Dispatch routines register completion callback routines, otherwise driver ignores that I/O operation
- IRP goes upwards now, and callbacks are called accordingly passing data

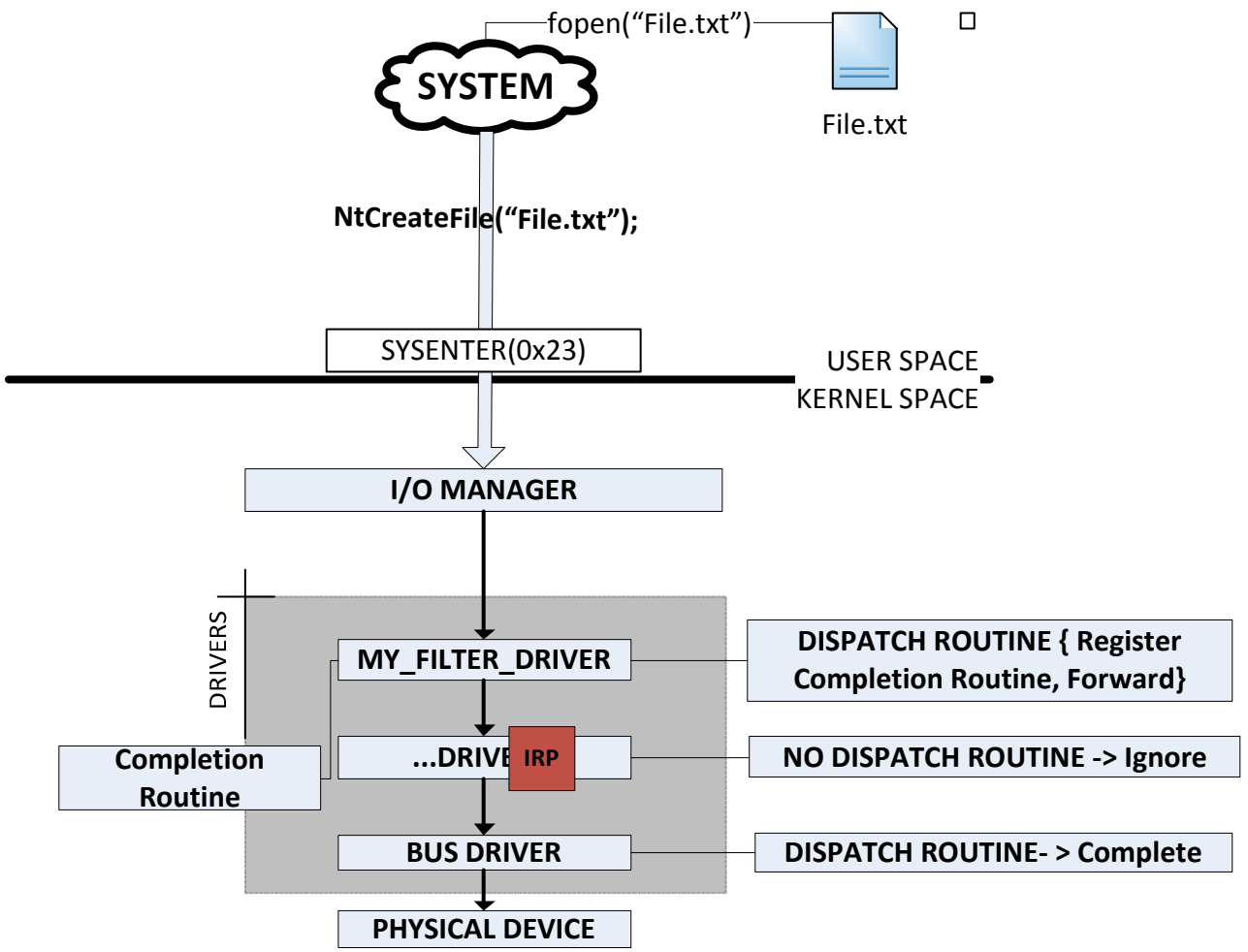
IRP path


fopen("File.txt")
SYSTEM
File.txt
NtCreateFile("File.txt");
SYSENTER(0x23)
USER SPACE
KERNEL SPACE
I/O MANAGER
IRP
DRIVERS
MY_FILTER_DRIVER
DISPATCH ROUTINE -> Forward
Register Completion Routine
...DRIVER...
NO DISPATCH ROUTINE -> Ignore
BUS DRIVER
DISPATCH ROUTINE- > Complete
PHYSICAL DEVICE

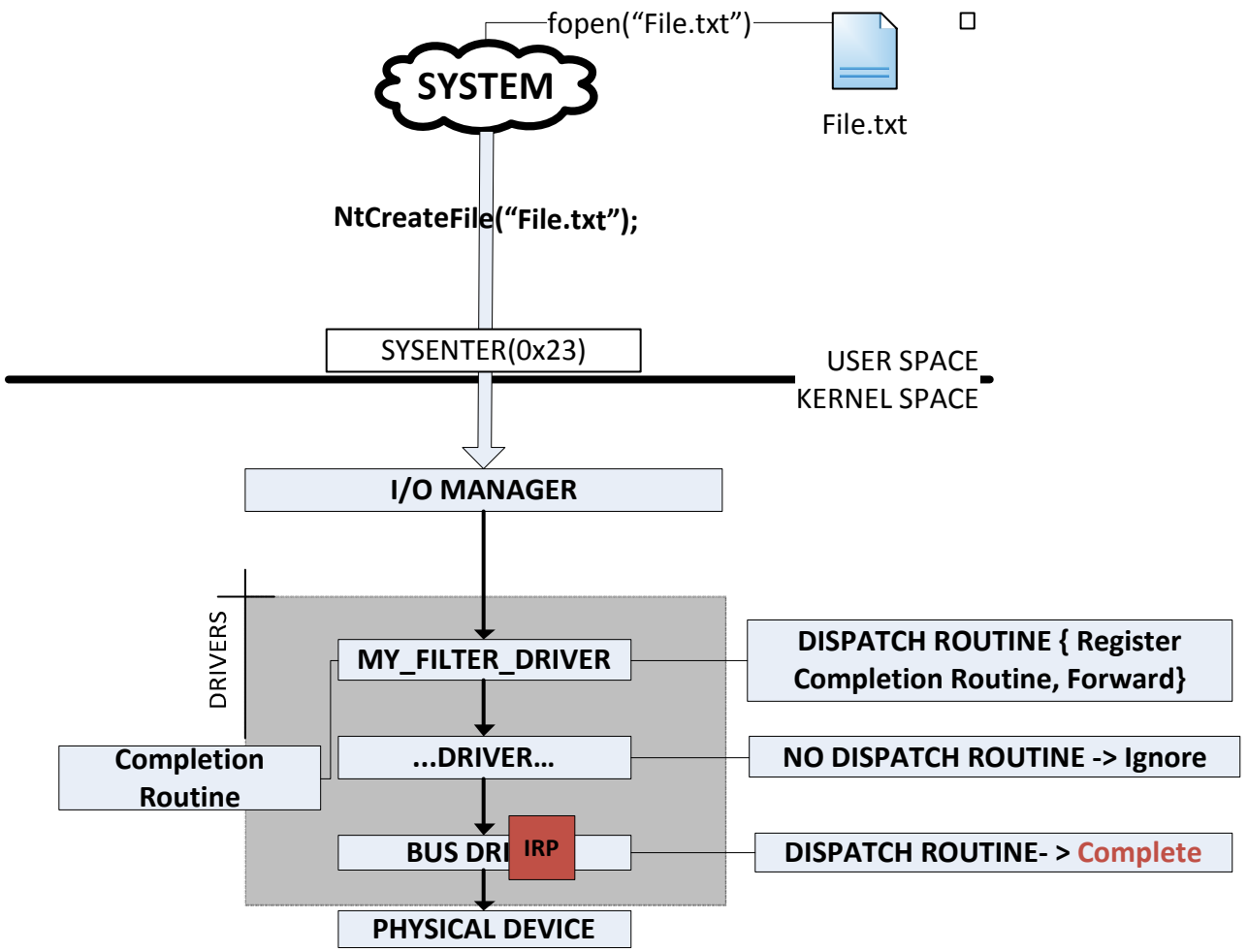
IRP path


fopen("File.txt")
SYSTEM
File.txt
NtCreateFile("File.txt");
SYSENTER(0x23)
USER SPACE
KERNEL SPACE
I/O MANAGER
DRIVERS
MY_FILTER_
IRP
VER
...DRIVER...
BUS DRIVER
PHYSICAL DEVICE
Completion Routine
DISPATCH ROUTINE { Register Completion Routine, Forward}
NO DISPATCH ROUTINE -> Ignore
DISPATCH ROUTINE- > Complete

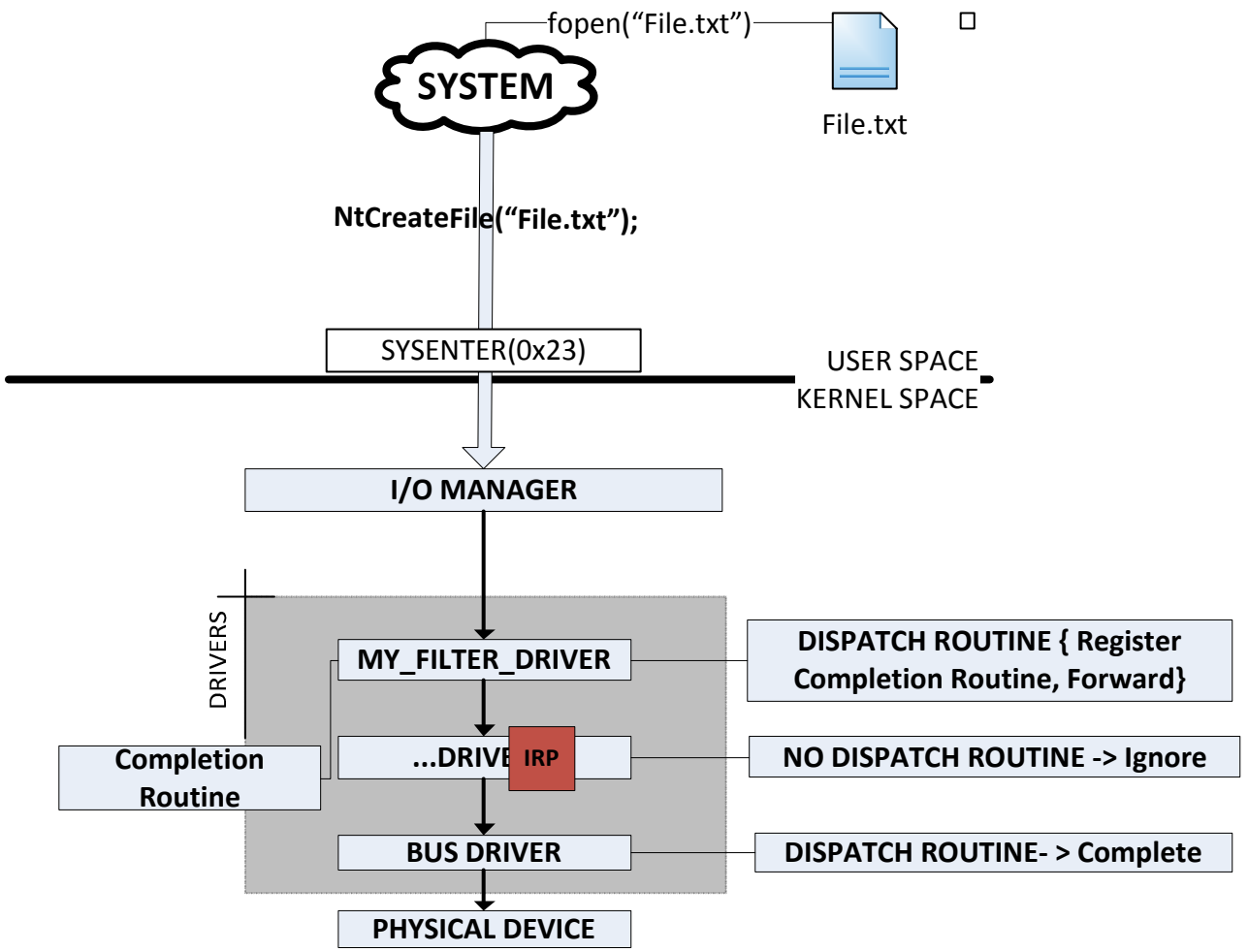
IRP path


fopen("File.txt")
SYSTEM
File.txt
NtCreateFile("File.txt");
SYSENTER(0x23)
USER SPACE
KERNEL SPACE
I/O MANAGER
DRIVERS
MY_FILTER_DRIVER
DISPATCH ROUTINE { Register Completion Routine, Forward}
Completion Routine
...DRIVE
IRP
NO DISPATCH ROUTINE -> Ignore
BUS DRIVER
DISPATCH ROUTINE- > Complete
PHYSICAL DEVICE

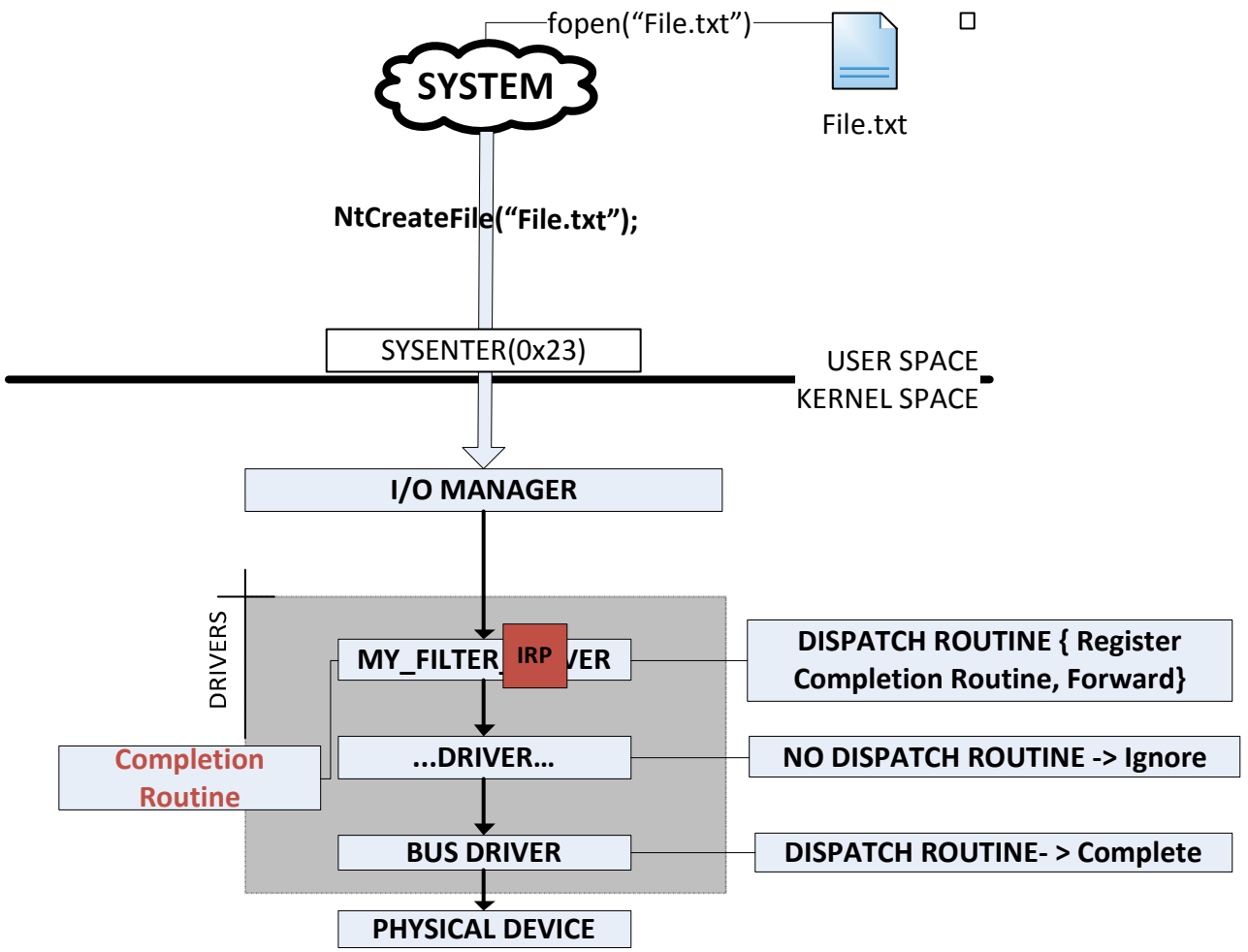
IRP path


fopen("File.txt")
SYSTEM
File.txt
NtCreateFile("File.txt");
SYSENTER(0x23)
USER SPACE
KERNEL SPACE
I/O MANAGER
DRIVERS
MY_FILTER_DRIVER
...DRIVER...
BUS DRI
IRP
PHYSICAL DEVICE
Completion Routine
DISPATCH ROUTINE { Register Completion Routine, Forward}
NO DISPATCH ROUTINE -> Ignore
DISPATCH ROUTINE- > Complete

IRP path


fopen("File.txt")
SYSTEM
File.txt
NtCreateFile("File.txt");
SYSENTER(0x23)
USER SPACE
KERNEL SPACE
I/O MANAGER
DRIVERS
MY_FILTER_DRIVER
DISPATCH ROUTINE { Register Completion Routine, Forward}
Completion Routine
...DRIVE
IRP
NO DISPATCH ROUTINE -> Ignore
BUS DRIVER
DISPATCH ROUTINE- > Complete
PHYSICAL DEVICE

IRP path


fopen("File.txt")
SYSTEM
File.txt
NtCreateFile("File.txt");
SYSENTER(0x23)
USER SPACE
KERNEL SPACE
I/O MANAGER
DRIVERS
MY_FILTER_
IRP
VER
...DRIVER...
BUS DRIVER
PHYSICAL DEVICE
Completion Routine
DISPATCH ROUTINE { Register Completion Routine, Forward}
NO DISPATCH ROUTINE -> Ignore
DISPATCH ROUTINE- > Complete

MiniFilter Drivers

- It’s a Microsoft concept based on callbacks
- We register callbacks (preOperation and postOperation)
- Full control over each I/O call and file is offered
- Dispatch: Pre-Operations
  - Organize calls, set flags
  - Preview the I/O call
- Completion: Post Operations
  - Watch the results
  - File is ready, do whatever you want

Our Driver

- FS filter driver, “active monitor” altitude
- Monitors the file system for classified files
- If files are marked classified record the I/O operation
- If I/O operations of classified files meet rules cancel the operation
- Userland app marks files & communicates with driver
- Don’t let user close file unless file is marked

Monitoring

- Driver intercepts PnP messages in order to detect insertion of removable devices
- Monitor starts on the fly for new volumes
- Copy , File, Move of classified files towards removable drivers is not allowed and blocked
- Classified files in removable drives are being recorded
- Removable drives are not allowed after all
- Declassification is allowed but monitored
- Monitors processes for I/O calls
  - iexplore.exe & firefox .exe are not allowed to read classified files
  - The list can be expanded of course

Detecting the patterns

- All files are opened with IRP_MJ_CREATE IRP type
- This means a read or write is going to follow
- If write then we expect an IRP_MJ_WRITE
- IRP_MJ_WRITE has flags
  - FO_FILE_MODIFIED is returned when we save a file
  - FO_REMOTE_FILE is returned when we copy a file
- Copy, Move, Cut to removable disk
  - IRP_MJ_WRITE, FO_REMOTE_FILE
- Saving file
  - IRP_MJ_WRITE, FO_FILE_MODIFIED

What about Worms

- Worms usually hide by using DLL injection
- Run inside a benign process context
- Explorer.exe can access classified files
- What if a worm injects inside explorer.exe
- Our driver monitors threads too
  - If another thread of explorer.exe accesses a file instead of the usual one record or block
  - Admin should give a rule based on system behavior
- Our driver is flexible ;-)

Network Usage

- Windows expose RPC, SMB over TCP on intranets
- Worms can attack these services and steal files
- Basically they use file sharing services
- Our driver can block/ban classified files when accessed via a network share
- How? Check whether an I/O operation access token has impersonation privileges (TokenImpersonation)
- Why impersonation a network service is not a user but a dummy account

File marking

- Classified files are marked by our userspace application
- A file stores the list with classified files
- This file is loaded in memory whenever the application launches, it speeds up things
- Indexing can also be implemented
- What if thousands of classified files exist in our system
- Our driver stores another ephemeral data structure that remembers the classified files of the last two minutes
- A few comparisons are only needed

Enforcing Policy

- The problem: how can we force users to classify files
- Don’t allow files to successfully be saved
- Monitor files when an IPR_MJ_WRITE appears and check for its marking
- If it exists then file has classification
- If not, cancel the I/O operation and alert the user via our userspace application

DEMO

Results

- Driver has no overhead on system performance
- All dangerous actions are logged, a forensics expert can easily discover what happened and why did the files leaked
- Prevention is not always possible in case of extreme trojans
- Easy to use and transparent

THANK YOU ALL!

ACKz

xgeorge

gp

pfrag

ppspyros