SYNACKTIV
DIGITAL SECURITY

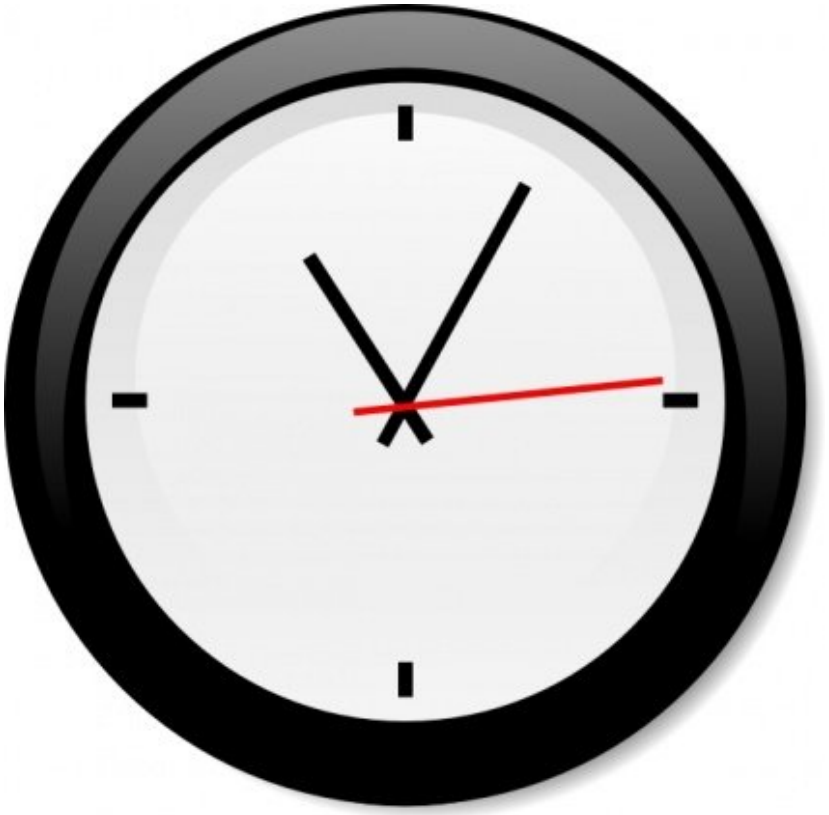
# TTD

## *Rump'in Rennes 2019*

October 4 2019

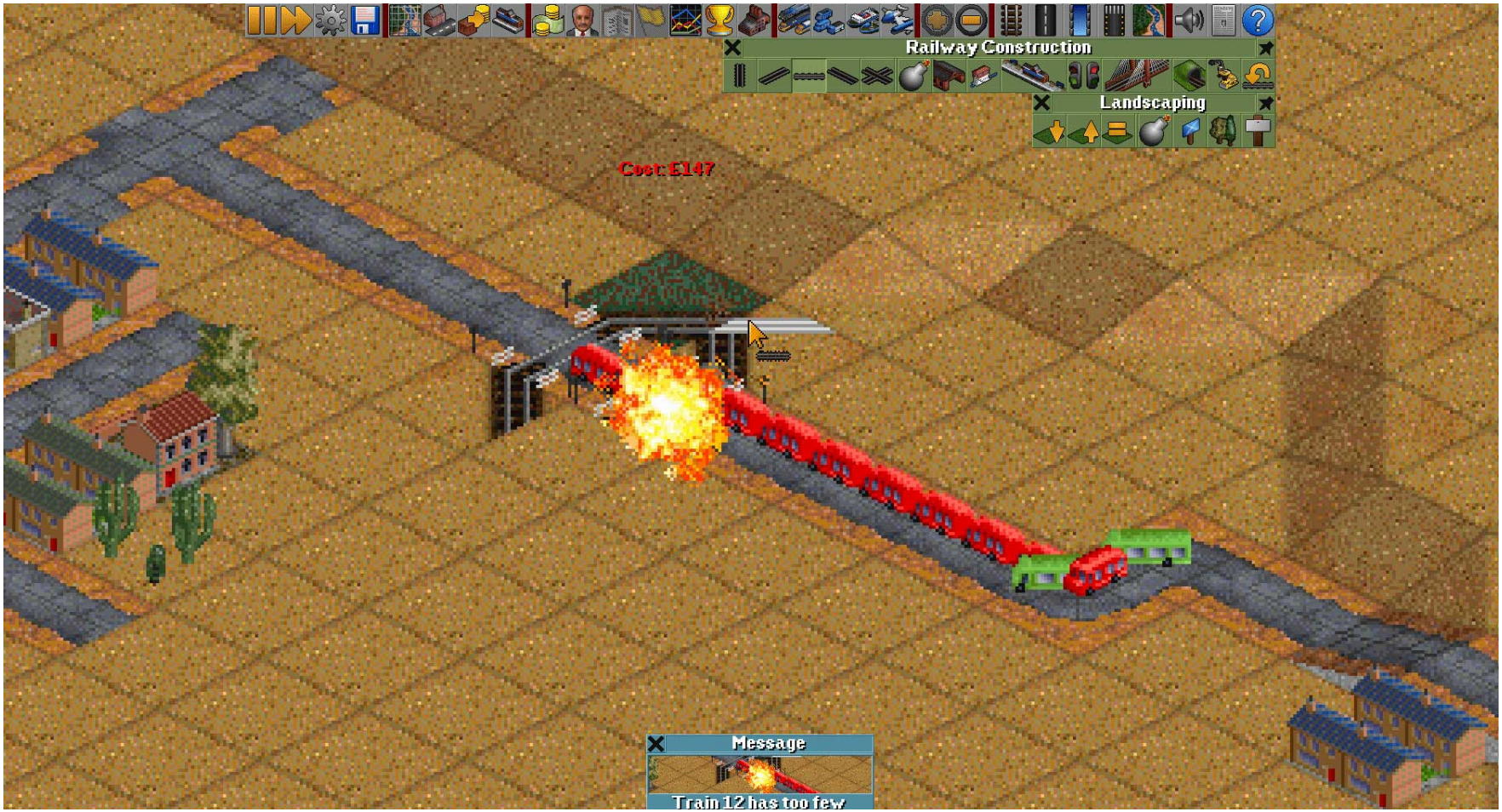
Samuel (@w4kfu) CHEVET

# Time To Drink ?



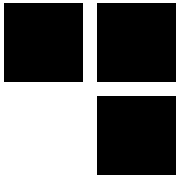
Soon !

# Transport Tycoon Deluxe ?



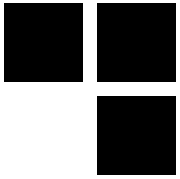

☹

# Agenda



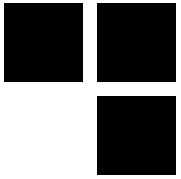
1. Time Travel Debugging
2. WinDbg Time Travel Debugging
3. Trace file
4. Conclusion

# Time Travel Debugging ?



- Reverse debuggers
- Ability to record the execution of a program
  - Every memory access
  - Every computation
  - Every system call
  - Special instructions (ex : CPUID, ...)
- Rewind and replay to inspect the program state

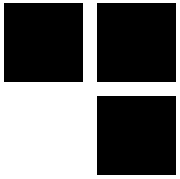
# Reverse debugging VS runtime debugging



## Runtime debugging resolved issues

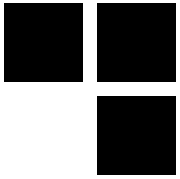
- No need to restart in case of a wrong action
- Less time-consuming (especially when programs are large !)
- Go to any previous point in the execution history
- Debugging should become less hard and less complex

# Time traveling debuggers



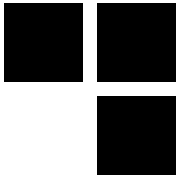
- rr : Linux
- UndoDB : Linux and Android
- ocamldebug : OCaml
- ...

# Time traveling debuggers



- rr : Linux
- UndoDB : Linux and Android
- ocamldebug : OCaml
- ...
- WinDbg Time Travel Debugging

# Agenda



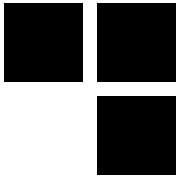
1 Time Travel Debugging

2 WinDbg Time Travel Debugging

3 Trace file

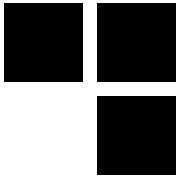
4 Conclusion

# WinDbg Time Travel Debugging



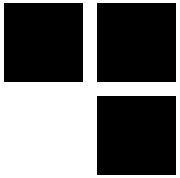
- Available in the latest preview of WinDbg (Windows store)
- x86, x64, ARM
- Only user-land
- Handle self-modifying code
- Multithreaded programs
- Not able to go back in time and change the state
- Trace through family of child processes
- Attach to a running process

# Windbg Time Travel Debugging



- TTD.exe : Trace tool
- TTDInject.exe : Application injector
- TTDAnalyze.dll : Trace analyser
- TTDLoader.dll : Runtime loader
- TTDRecord.dll : Recording manager
- TTDRecordCPU.dll : CPU recorder runtime
- TTDReplay.dll : Replay engine
- TTDReplayCPU.dll : CPU replay runtime
- TTDWriter.dll : Trace writer

# Time Travel Debugging standalone



- TTD.exe : Trace tool
- TTDInject.exe : Application injector
- ~~TTDAnalyze.dll : Trace analyser~~
- TTDLoader.dll : Runtime loader
- TTDRecord.dll : Recording manager
- TTDRecordCPU.dll : CPU recorder runtime
- ~~TTDReplay.dll : Replay engine~~
- ~~TTDReplayCPU.dll : CPU replay runtime~~
- ~~TTDWriter.dll : Trace writer~~

```
>TTD.exe -out C:\Users\lolita\trace_out\ -launch C:\Windows\notepad.exe
```

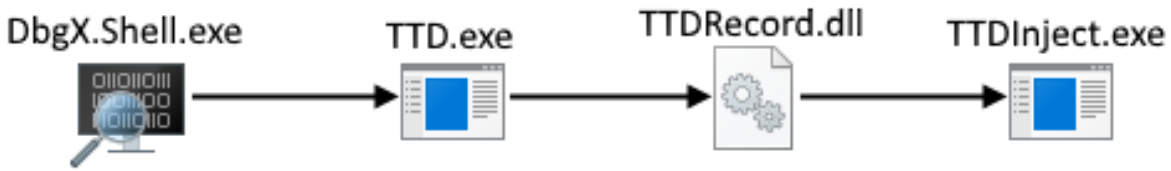
# Flow


- Windbg preview executable : DbgX.Shell.exe
  - Launch executable target in advanced mode
  - Attach to a running process
- Launch TTD.exe

# Flow


- Load TTDRecord.dll
- Prepare guest process
- Prepare communication channel
- Launch TTDInject.exe

# Flow

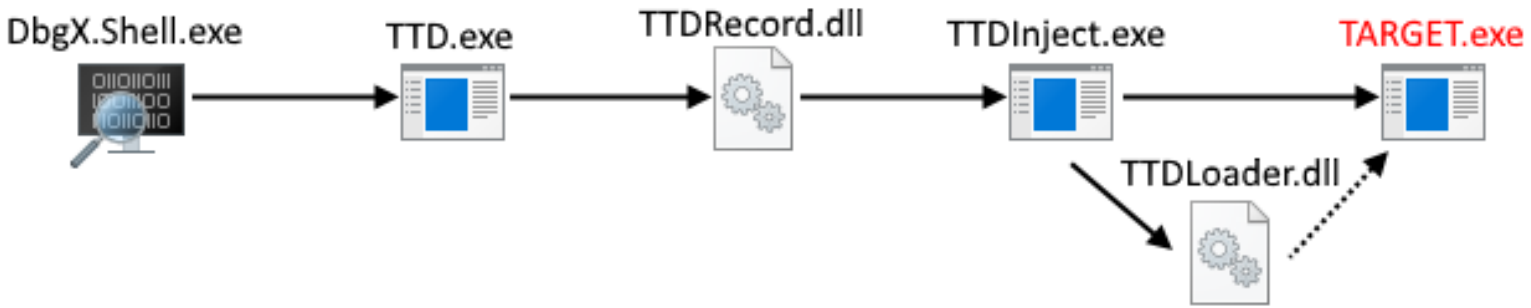



- Attach or launch : CreateProcess(..., CREATE_SUSPENDED, ...)
- Count number of threads
- Allocate Virtual CPU (Nirvana) & jit buffer remotly

# Flow

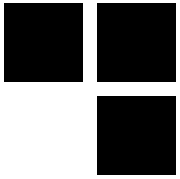

- Load `TTDLoader.dll`
- Write recording parameters (`TTDLoader!ParametersBlock`)
  - Configuration for the communication
  - Library path for the desired DLL engine
  - …
- Write the DLL to the remote process

# Flow



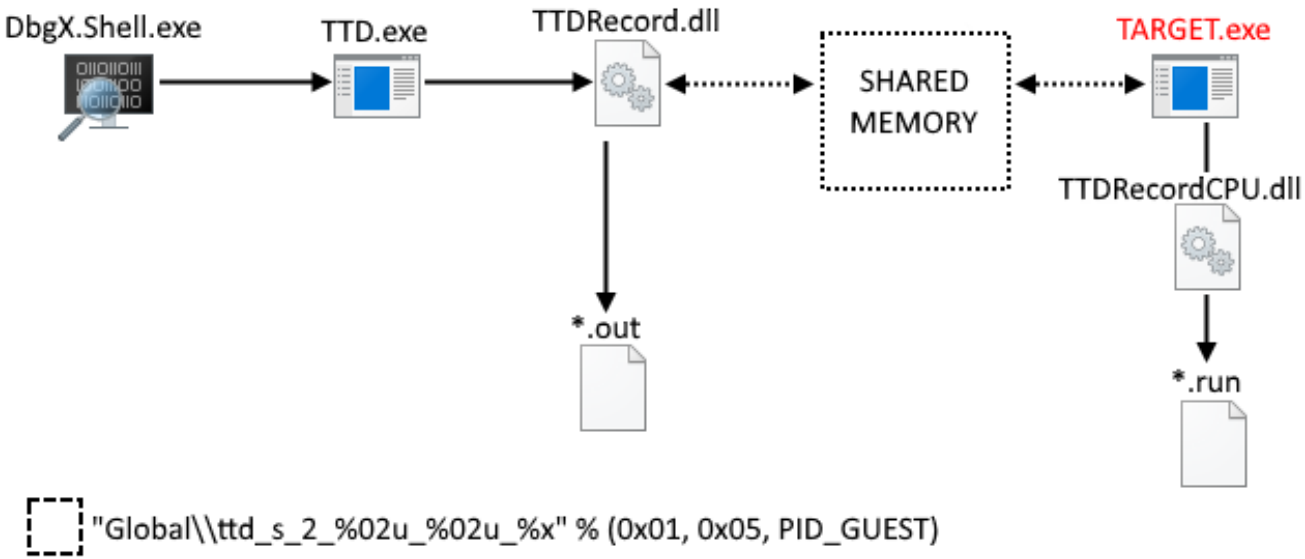

- CreateRemoteThread on TTDLoader!InjectThread
- NtSetInformationProcess(..., PROCESSINFOCLASS=ProcessInstrumentationCallback, ...)
- Callback is a fake stub that will be replaced by TTDRecordCPU!NirvOsInstrCB

# TTDLoader!InjectThread



- Load the DLL `TTDRecordCPU.dll`
  - Disassembler
  - Binary translator
  - Callbacks dispatcher
  - …
- Register DLL notification
- Open communication channels
- …

# Communication




## Commands

- Stop runtime
- Get feedback
- Terminate process
- Record process memory

# TTDRecordCPU.dll

- Translate native instructions into internal custom intermediate languages : Nirvana::SCODE

# TTDRecordCPU.dll

- Translate native instructions into internal custom intermediate languages : Nirvana::SCODE

```
48c7c042424242  mov     rax,42424242h
48c7c141000000  mov     rcx,41h
4801c8          add     rax,rcx
cc              int     3
```

# TTDRecordCPU.dll

- Translate native instructions into internal custom intermediate languages : Nirvana::SCODE

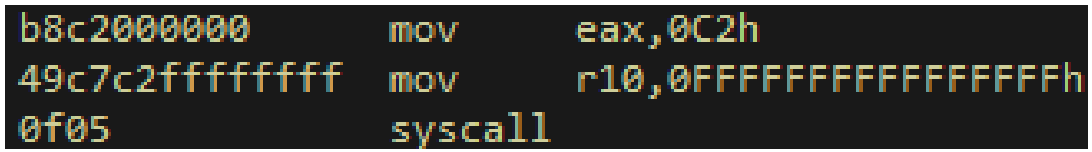
```
48c7c042424242  mov     rax,42424242h
48c7c141000000  mov     rcx,41h
4801c8          add     rax,rcx
cc              int     3
```

1. ExecuteOpLoad64_Dest_UImm32
2. ExecuteOpLoad64_Dest_UImm32
3. ExecuteOpAdd64_Dest_Src_Src_Carry_Imm
4. ExecuteOpBreakpoint

# TTDRecordCPU.dll

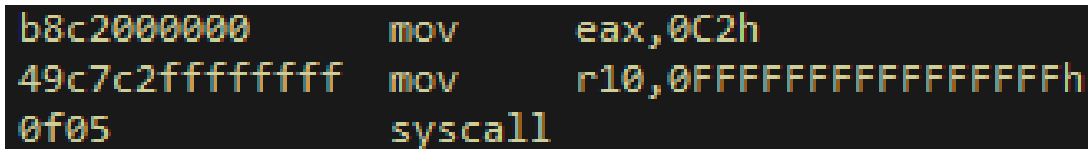
- 24 record callbacks : Write info into trace
  - Memory allocated
  - Kernel call
  - DLL loading
  - …

# Record callback example



```
b8c2000000      mov     eax,0C2h
49c7c2ffffffff  mov     r10,0FFFFFFFFFFFFFFFFh
0f05            syscall
```

# Record callback example

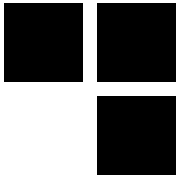


```
b8c2000000      mov     eax,0C2h
49c7c2ffffffff  mov     r10,0FFFFFFFFFFFFFFFFh
0f05            syscall
```

1. ExecuteOpLoad64_Dest_UImm32
2. ExecuteOpLoad64_Dest_UImm32
3. ExecuteOpSTOPSIM

- Exit dispatcher loop
- TTDRecordCPU!RunPostSimulationCallbacks
  → TTDRecordCPU!KernelCallCallback

# Replay

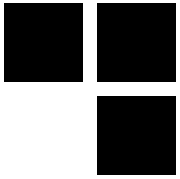


- TTDReplayCPU.dll : CPU replay runtime
- Used when opening a time travel debugging traces
- Share same code base than the recorder

## Bonus

- No need to run the program again nor having it installed
- Original program code and initial state stored in the trace file

# Agenda



1 Time Travel Debugging

2 WinDbg Time Travel Debugging

3 Trace file

4 Conclusion

# Output files

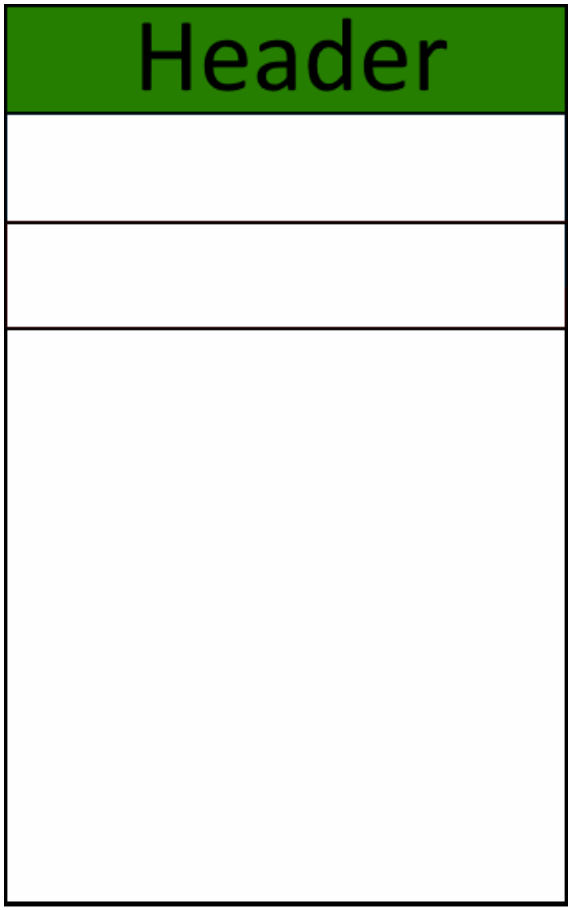
## .out

- 'out' extension
- Owner : TTDRecord.dll
- Redirected STDOUT/STDERR of TTDInject.exe / TTDLoader.dll
- Tracer output file : Log, Error, ExitCode, ...

# .run file

- ’run’ extension
- (All) data saved of the CPU recorder
- Owner : TTDRecordCPU.dll
- Kind of database with rich information
- Proprietary format ☺

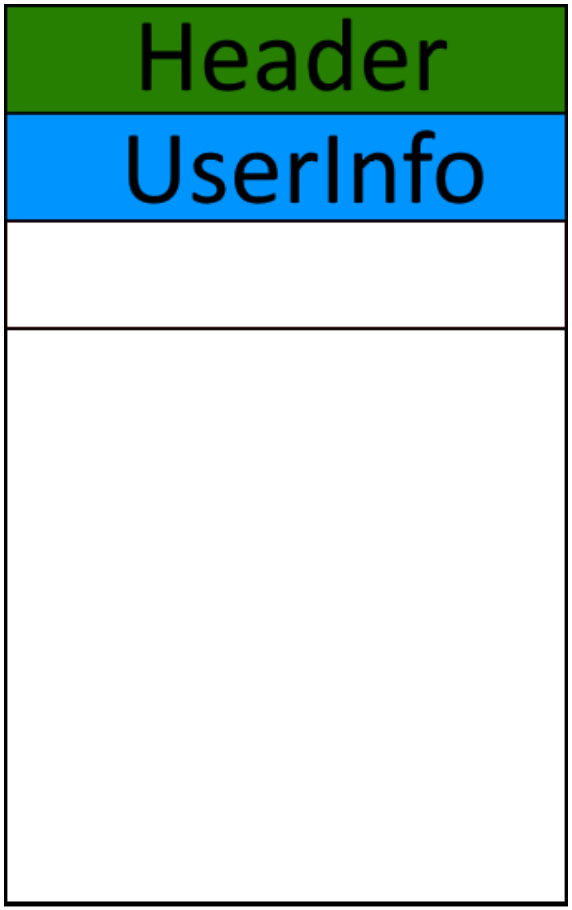
# .run file




**Header : Length 0x28 bytes**

- Magic (16 bytes)
- Recorder Major/Minor version
- Type : Disk, MemoryMapped
- Revision
- PageHeaderSize
- UserInfoLength
- …

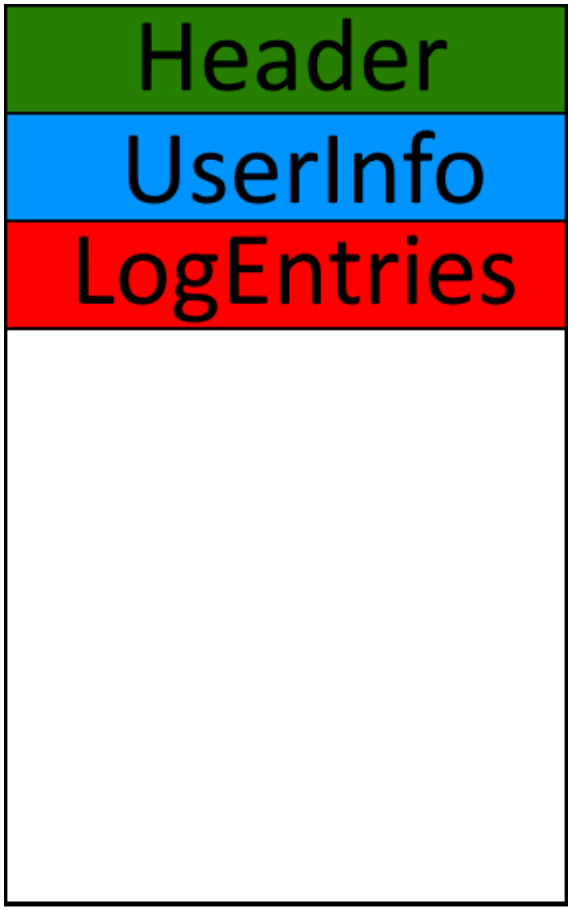
# .run file




## User information : Length in header

- Architecture
- Processor feature
- Tracer unique identifier
- PerformanceFrequency
- Min/Max application address
- …

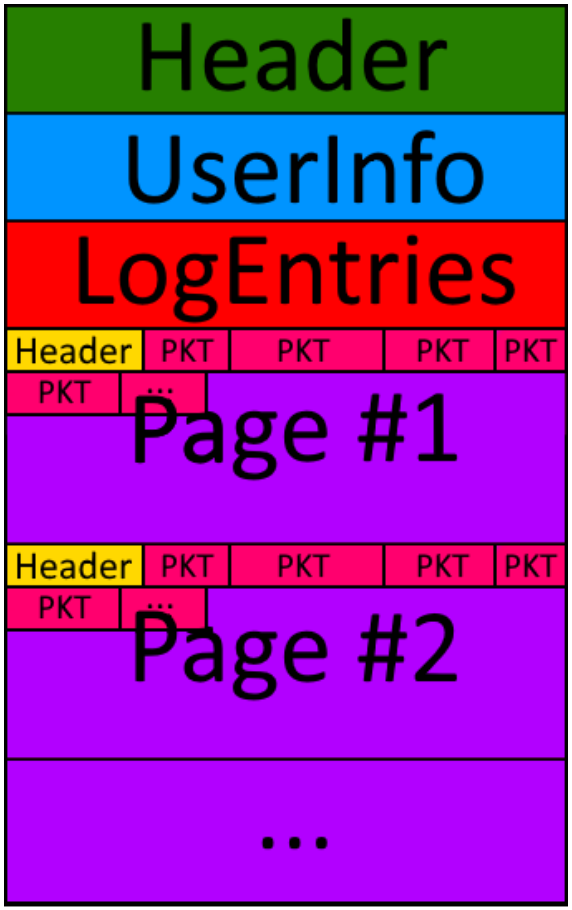
# .run file




## Log entries

- Series of different entry type
  - 0x01 : OpenStream
  - 0x02 : CloseStream
  - 0x04 : OpenPage
  - 0x05 : ClosePage
  - 0x07 : CloseFile
  - 0x08 : FastFailCloseFile

# .run file




## Page : Length 1MB

- Header
  - Signature : 'Page'
  - Index / NextIndex
  - Packet count
  - Used size
- Series of packets with different type

# Packet type

**System**
SecGeneric
ThreadExit
Sequence
SecExit
SecNative
SecInvalidInst
SecAtomicOp
SecDebugBreak
SecCodeCacheFlush
SecExceptionExit
**ModLoad**
ModUnload
Message
Reserved_CaptureTrace-
Flags
SecEtwEvent
**MemoryBlock**
**MemoryRange**
ThreadExecutedNatively
**DataCacheLine**
**CodeCacheLine**
MemoryRead1Byte
MemoryRead2Byte
MemoryRead4Byte
MemoryRead8Byte
MemoryRead16Byte
MemoryReadNByte
MemoryReadNDWord
ShortInstruction
LongInstruction
BlockOfInstructions
Rdtsc
FullFlushBegin
ValidateGuestContext
**RegisterData**
InstructionCountOverflow
FallbackExit
ReadInformation
WriteInformation
ResumeExecution
OpenClient
CloseClient
CustomEvent
StartIsland

# Packet type : System

- Only at process launch

## Content

- Current system date and time
- CreationTime / KernelTime / UserTime
- Counter/Frequency performance
- MajorVersion ; MinorVersion ; BuildNumber ; PlatformId
- PEB address
- Computername / Username

# Packet type : ModLoad

- DLL loading callback

**Content**

- GuestAddress
- ImageSize
- CheckSum
- Timestamp
- ThreaId
- ModuleName

# Packet type : MemoryBlock

- Uncommited memory region
- Loaded DLL
  - Pros : Traces are portable across machines
  - Cons : Performance overhead and trace file size
- TEB
- Stack

**Content**

- GuestAddress
- Data

# Packet type : MemoryRange

- Allocated memory callback

**Content**

- GuestAddress
- Size

# Packet type : Data/CodeCacheLine

- Reduce the number of store data values
- Handle self-modifying code

**Content**

- GuestAddress aligned
- CachedValue (16 bytes)
- ActualValue (16 bytes)

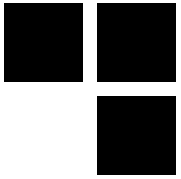
# Packet type : RegisterData

- Guest exception
- Fallback
- Thread callback
- Full cache flush
- Instruction limit

## Content

- Array of index ZERO register
- Array of index register
- Register data

# Agenda



1 Time Travel Debugging

2 WinDbg Time Travel Debugging

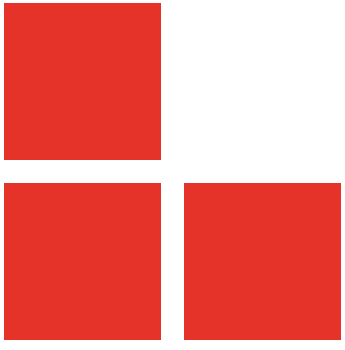
3 Trace file

4 Conclusion

# Conclusion

- TTD seems still in developpment : PacketType added/removed between versions
- Content of traces are difficult to exploit in standalone (without the replayer)
  - Register state are stored rarely
  - Heuristic on code cache ?
  - …
  - Python wrapper around the new `dbgeng.dll` (or others `TTD*.dll`) ?
- TTD brings a new approach to traditional debugging
- TTD looks really great from the inside

# QUESTIONS?



Thank you for your attention

SYNACKTIV
DIGITAL SECURITY