**SRDF: Write Your Own Security Tool**

*By Amr Thabet*

Do you see writing a security tool in windows is hard?

Do you have a great idea but you can’t implement it?

Do you have a good malware analysis tool and you don’t need it to become a plugin in OllyDbg or IDA Pro?

So, Security Research and Development Framework is for you.

**Abstract:**

This is a free open source Development Framework created to support writing security tools and malware analysis tools. And to convert the security researches and ideas from the theoretical approach to the practical implementation.

This development framework created mainly to support the malware field to create malware analysis tools and anti-virus tools easily without reinventing the wheel and inspire the innovative minds to write their researches on this field and implement them using SRDF.

**Introduction:**

In the last several years, the malware black market grows widely. The statistics shows that the number of new viruses increased from 300,000 viruses to millions and millions nowadays.

The complexity of malware attacks also increased from small amateur viruses to stuxnet, duqu and flame.

The malware field is searching for new technologies and researches, searching for united community can withstand against these attacks. And that’s why SRDF

The SRDF is not and will not be developed by one person or a team. It will be developed by a big community tries to share their knowledge and tools inside this Framework

SRDF still not finished … and it will not be finished as it’s a community based framework developed by the contributors. We just begin the idea.

The SRDF is divided into 2 parts: User-Mode and Kernel-Mode. And we will describe each one in the next sections.

**Goals:**

1. Help Researchers in Malware or Network Security fields implement their ideas.
2. Provide a full object oriented development framework with a suitable design to meet the requirements of the targeted applications
3. To unite all small and separate tools inside one development framework.

**Targeted Applications:**

* Antivirus & Virus Removal Tools
* Malware Analysis Tools (Static – Dynamic – Behavioral)
* Network Tools (Sniffers – Firewalls – IDS/IPS – Packet Analysis Tools)
* Exploitation & Security Mitigation Tools

**The Features:**

Before talking about SRDF Design and structure, I want to give you what you will gain from SRDF and what it could add to your project.

In User-Mode part, SRDF gives you many helpful tools … and they are:

* Assembler and Disassembler
* x86 Emulator
* Debugger
* PE Analyzer
* Process Analyzer (Loaded DLLs, MemoryMaps … etc)
* MD5, SSDeep and Wildlist Scanner (YARA)
* API Hooker and Process Injection
* Backend Database, XML Serializer
* And many more

In the Kernel-Mode part, it tries to make it easy to write your own filter device driver (not with WDF and callbacks) and gives an easy, object oriented (as much as we can) development framework with these features:

* Object-oriented and easy to use development framework
* Easy IRP dispatching mechanism
* SSDT Hooker
* Layered Devices Filtering
* TDI Firewall
* File and Registry Manager
* Kernel Mode easy to use internet sockets
* Filesystem Filter

Still the Kernel-Mode in progress and many features will be added in the near future.

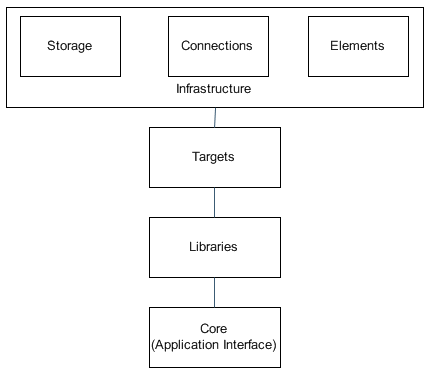
**The Design Methodology:**

I created this design form the Targeted Applications best design

Let’s now see the design:

**The User-Mode Part:**

**The Design:**



**Infrastructure:**

This includes the essential elements of any development framework and it’s not related to security like: string, hash, list, serializer, database, registry manipulation, sockets and so on.

We decided to create this part rather than depending on any development framework to make this framework independent from any other development frameworks and to be portable on any development framework

**Targets:**

This is the beginning of the SRDF. This part is simply the Target from your security tool. What do you want to secure or secure from. And it includes Files (PE Files and others), Processes and Packets.

**Libraries:**

That’s the security tools that the SRDF support. And it’s divided into two namespaces: malware and network

Malware includes the assemblers and disassemblers, emulator, debugger, API Hooker, Yara Scanner (wildcard scanner) file recursive scanner and other tools

Network includes User-Mode capturing and Firewall

**Core (The Application Interface):**

The Core includes the Logging system and the back-end Database.

And also, it’s the Application Interface. Like cConsoleApp … and you can inherit from it to create your own User-Interface.

We wish this part to be expanded to include more user interfaces and management systems

**The Infrastructure:**

**Elements:**

It’s divided into three namespaces:

1. **String**: it contains the string class, encoded string, hash and list
2. **Code:** it contains the NativeCode class and StoredProcedure … and they represents the shellcode and the code that stored in database. Like a virus detection routines inside an Antivirus
3. **XML:** and it contains the XMLEncoder and the Serializer.

**Connections:**

It’s divided into three namespaces:

1. **Internet:** and it contains the internet communication protocols like sockets, HTTP Sockets and so on.
2. **IPC:** and it contains the Inter-Process Communication protocol
3. **User-Mode to Kernel-Mode Communication:** and it contains the communication protocol to communicate to the kernel-mode part of the SRDF

**Storage:**

It’s divided into three namespaces:

1. **Databases:** and it contains the Database class and SQLiteDB and so on.
2. **Files:**and contains the File writing and logging classes
3. **Registry:** and it contains the registry read and write

**The Targets:**

**Files:**

This namespace describes the File Formats of The Files that could contain malicious code like: Executable Files (PE and ELF) and Document Files (PDF, Docx …) and so on.

Until now it contains The PE Files parser

**Process:**

And it includes one class only named cProcess. And, this class describes a running process and parses its PEB and gives you the important information about the process and its memory map. And support injecting code and create a remote thread.

**Packets:**

And it includes classes that describe an internet packets captured on the wire or generated for an attack.

**Libraries:**

It contains two namespaces:

**Malware:**

This namespace contains the scanning, Hooking and emulation libraries and contains Pokas Emulator wrapper class, Yara wrapper class (wildcard scanner), a debugger and contains a directory recursive scanner and other tools

And also, it contains the x86 assembler and disassembler (using Pokas Emulator Assembler) and allow to contain other assemblers and for other platforms.

**Network:**

This namespace should contain the User-Mode Packet capture and firewall. And should contain the Winpcap Packet capturing and firewall system.

It also should include Application Layer parsers for FTP, HTTP, IRC and all known protocols and include Pcap Reader and writer.

**The Core:**

And the core includes the cApp class that contains the back-end database and logging and the User-Interface such as cConsoleApp

**The Kernel-Mode:**

**The Kernel-Mode Goals:**

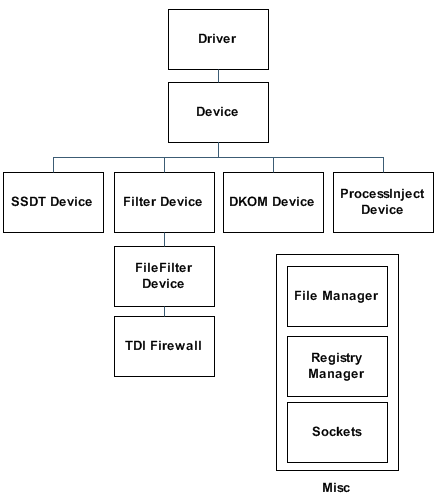
The Goals of the kernel-Mode development Framework are:

1. Easy to create a Kernel-Mode security tool
2. Support OOP using the native device driver programming APIs
3. Support detaching between devices in IRPs
4. Easy to use files, registry and so on
5. Create a User-Mode/Kernel-Mode communication protocol
6. Designed only for hooking and security tools.

The Kernel-Mode SRDF is designed on native device driver programming APIs and independent from the WDF (windows drivers foundation).

Now we will describe the design of Framework and then we will go through the IRP dispatching mechanism in the KM-SRDF

**The Design:**



**Driver:** It’s the core management system that dispatching the IRPs to the devices and manage the devices.

**Device:**it represents a device object and it contains the IRP dispatching between the control device object and the filtering device objects and includes attaching and detaching from a devices chain and all necessary functions for a device object

**SSDT Device:** this class is inherited from device class and it’s created for SSDT Hooking

**Filter Device:** this class created for attaching to a chain and filtering the inputs and the outputs of the IRPs

**File Filter Device:** this class is inherited from Filter Device and it’s created for filtering the File system I/O request packets (IRPs) or monitoring file operations

**TDI Firewall:** this class is inherited from Filter Device and it’s created for filtering the internet packets and connections and the processes that tries to connect to the internet

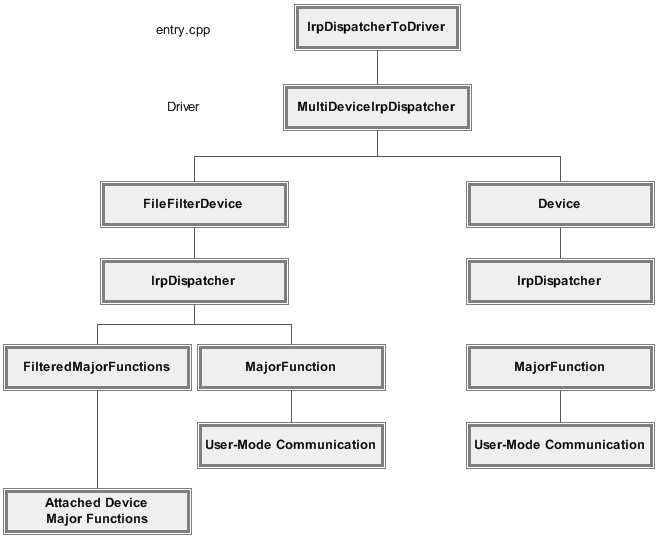
**DKOM Device:** this class created to provide a generic way to work with opaque structures in windows without worrying about windows version and subversion (under construction)

**Process Device:**this class provides a way to inject code or modify the memory of a process from the kernel-mode

**File/Registry Managers:** they are tools created to support writing files and working with registry easily without worrying about IRQL

**Sockets:** it’s an easy interface to connect to the internet using the TDI interface

**The IRP Dispatching:**



* The IRP dispatching begins from the entry.cpp and it dispatch the IRP to the Driver
* The driver checks the device object and dispatch the IRP to the related device
* The device sends the IRP to the User-Mode communication object to work with it as it’s sent to the control device object
* If it’s a FileFilter Device, the device dispatches the IRP based on the device object to the Attached Device Objects or to the control device object and the user-mode communication

**The Major Projects:**

There are major projects inside SRDF which makes it the biggest security project and a world class framework for security … I will describe some of them:

***Packetyzer*:**

This is a packet analysis project.

***X86 Disassembler Engine*:**

This disassembler engine has many features that helps you creating an instruction manipulating application like:

**Descriptive Structure:** this feature allows you to disassembler not only into string format but also in a structure describe the instruction to easily create a parser understands the instruction and act with its shape

**Detecting Invalid Opcodes:** even the project still in the development but it has the ability to detect invalid opcodes and invalid instructions and get the length and minimum information about the undefined instructions (SSE … etc)

**Assembler:** this feature allows you to assemble an instruction from string or from the descriptive structure into opcodes again

**Categorizing:** this feature allows you to understand what the instruction type … is it a ring 0 instruction or fpu, or arithmetic (add,sub, or, and …) or higher level arithmetic (mul, div, fabs, …) or flow redirection and many other categories.

***x86 Emulator*:**

This is an x86 emulator for win32 applications. It has many features like:

* **Support adding new APIs and adding the emulation function to them.**
* **Support a very powerful debugger that has a parser that parses the condition you give and create a very fast native code that performs the check on this condition.**
* **Support seh and support tib, teb, peb and peb\_ldr\_data.**
* **It monitors all the memory writes and log up to 10 previous Eips and saves the last accessed and the last modified place in memory.**
* **Emulate Shellcode and/or PE Applications**
* **Monitor LoadLibrary & GetProcAddress and could reconstruct the import table**
* **And many more.**

**The Emulator’s Debugger:**

**This Emulator’s debugger takes a text-based condition to check or takes a pointer to a function to execute on every instruction emulated to perform the check**

**Examples of Text-based Conditions:**

* *"\_\_isdirty(eip)"*
* *"\_\_disp() >=0x00401000 && ecx >10"*
* *"(eax & 0xff)> 5\*(edx & 0xff) || \_\_read(0x401000)==0x500"*
* *"\_\_isapiequal('getprocaddress') || \_\_isapiequal('loadlibraryA')"*

The supported Functions in the debugger are:

1. **isdirty**(dword ptr): this function checks if this pointer is modified before. It usually coded like this "\_\_isdirty(eip)"
2. **lastmodified**(): this function return the last modified place in memory
3. **lastaccessed**(): this function return the last accessed place in memory
4. **isapi**():checks if the eip belongs to an API
5. **isapiequal**(char\* name): this function checks if the eip belongs to an API with the name you specified (the name is not case sensitive)
6. **rm**(),**disp**(),**imm**() : these functions return some informations related to the instruction
7. **read**(dword ptr): this function reads a data from the memory given the pointer to it.

***Debugger*:**

This project is an x86 debugger inside SRDF. And this tool has many features like:

1. It could set Int3 breakpoints, memory breakpoint (read,write), Hardware breakpoints (read, write, execute) and single step breakpoint.
2. It allows monitoring the events (LoadDLL, create thread, terminate thread … etc)
3. Gives much information about the process and the PEFile using SRDF process analyzer and PE Parser

**To do list:**

1. **Antivirus:**
2. XRAY Tool
3. Heuristics Analysis
4. Behavior-based Detection Tools.
5. More File Formats (PDF, apk, …)
6. OpenSBI and other Virus Classification File Formats
7. Sandboxing Mechanism.

Using API/ SSDT Hooking

1. Update System with Flexible Mechanism
2. **Malware Analysis:**
3. SSDT Hooking for (Processes, Files, Registry and Sockets System Calls)
4. API Hooking (for the same as above)
5. Wow64 System calls Hooking
6. Improvement in Pokas Emulator, Assembler and Disassembler
7. Recursive Disassembler
8. More APIs Emulation in Pokas x86 Emulator
9. Support more Instructions (All general purpose instructions and support mmx and SSE)
10. Support idb (IDA Pro Database) to read it and use its analysis
11. **Unpackers:**I’m aiming to create a database for all static unpacking codes for the mostly common unpackers and I hope it could be updated by the community
12. **Integrations**:
13. Integration into IDA Pro Plugin Interface … and in (Debugger Menu)
14. OllyDbgPlugin Interface
15. Ollyscript Executer on cDebugger
16. Metasploit Integeration (in Meterpreter Post Exploitation
17. Python integration using SWIG
18. **Network:**
    1. Support NDIS, kernel sockets and more new libraries
    2. Process Analyzer in Kernel-Mode
    3. Packet Capturing Library
    4. More Debugging and Bug fixing
    5. Support x64 windows drivers
19. **Others:**
    1. We need to build website.
    2. We need activities for learning.
    3. We need more documentations and tutorials
    4. We need more helpful tools and applications based on SRDF

**Successful Projects based on SRDF:**

***Inspector Gadgets*:**

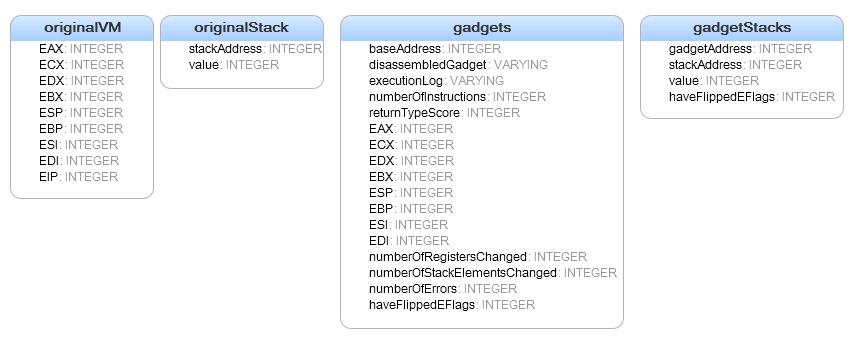
This tool is created by Jonas lykkegaard.

This tool is a ROP gadget indexing and searching tool. It is based on emulating all gadgets and saving the resulting state of the process in a searchable form.

**Features:**

* Saves all the behavior effects were done by the gadget
* Search for instructions based on the behavioral effects were done by the gadget not only by its instructions’ shape
* Support gadgets ends with ret, pop/jmp, iret and ret far
* Saves gadgets states in SQLite database for easy to retrieve again
* Allow searching by SQL Queries directly to the database
* It has more than 60 predefined SQL queries for all basic searches and needs.
* It includes scoring system to allow arranging by the best gadgets.
* Include a friendly user interface (GUI) … under development

**The Gadgets Database Design:**



1. **OriginalVM**: it includes the initial states of all registers
2. **OriginalStack**: it includes the original values of all stack variables
3. **Gadgets**: it saves the gadgets and the behavioral effects by each one of them and the score
4. **Gadgets Stack**: it saves the modified stack values for each gadget

**Join Us:**

Do you get benefit from this framework and you need to give something back?

Do you want to add something to your CV?

Do you want to meet smart developers and join a big community?

Do you want to learn new things?

Here is place … join the development community, meet new smart people and have fun.

**Source Code:**

<http://code.google.com/p/srdf/>

**Conclusion:**

This development framework will support the anti-malware technologies to grow and support implementing researches in the malware field more to withstand against the new attacks nowadays

The framework is based on community and we aim to create a big community for it. We didn’t finished the framework … we just begin

**About the Author:**

I’m Amr Thabet. I’ma Malware Researcher at Q-CERT. I began in this field from nearly 4 years.

I’m the Author of Pokas x86 Emulator. I gave a talk in Cairo Security Camp 2010 and the University of Sydney.