# Detecting malicious documentation

Syscan 2010, HoChiMinh
September 23th

NGUYEN Anh Quynh <aquynh @ gmail.com>

# My research areas

- Practical security problems of Virtual Machine (VM)
  - Protect VM
    - Live memory forensic for VM
    - Malware scanner for VM
  - Leverage VM for various security-related areas
    - Malware analysis
    - Dynamic binary analysis
    - Vulnerability research
    - etc ...

# Talk preview

- A new scanner named D-Analyzer, specially built to detect malicious documentation files
  - Reliably detect malicious attack from unknown documentation files
  - Independence of file formats
  - Zero-day attack detection is supported

# Agenda

- Modern threats from malicious documentation
- D-Analyzer solution
  - Approach
  - Background on Tainting analysis
  - Architecture, Design & Implementation
- Discussions
- Conclusions
- Q & A

# Tradional malware

- Executable file format
  - Mostly .EXE files
  - Trick users to run malware
    - Drive-by download
    - Social engineering
- Detect and clean with traditional malware scanner
  - Scan on-demand
  - Scan on-the-fly

# Upcoming malware

- From non-executable documentation files
  - Attack embeded in any type of documentation files
    - .PDF, .DOC, .XLS, .PPT, .AVI, ...
  - Silently exploit vulnerable application when opening
    - Function under the same account privilege with current user
      - Privilege escalation is possible with other local vulnerabilities
    - Setup backdoors
    - Download & install trojan, rootkits to secretly spy on the victim, and steal information

# Upcoming malware (2)

- How is it possible?
  - Most file formats are complicated, with lots of optional data fields
    - Handling file data properly is far from trivial, thus the implementation is vulnerable to security bugs
      - Buffer overflow, Format string, Integer overflow
- How to exploit?
  - Make victim to open a craft documentation file to trigger vulnerable code in vulnerable application
  - Embeded malicious payload executed to exploit the vulnerability

# Upcoming malware (3)

- Documentation-based attack become extremely popular recently
  - Not in theory, but getting more and more serious
  - Dominate other kind of bugs, and frequenly published with PoC code
- Why?
  - Client-side attack become a major trend
    - Finding client-side vulnerabilities is easier than server-based ones
    - Very few users are aware of this threat!
    - More users → more victims → generate more money for cybercrime
    - Make targeted attack feasible


search...
RedOracle
Username
Password
Login
English
Italian
Home
Services
News
Contents
Downloads
Links
Contact Us
My Account
Default Password DataBase
Security Live Distro
Free Exploits Database
Extended Donwload Collection
Menu
Passwords
Live Distro
Exploits
Vulnerabilities
Downloads
White Papers
Submit
RSS Feed
Search exploits
Submit
Latest 50 local exploits.
Exploits from 0 to 50. We have 1124 local exploits.
0 50 100 150 200 250 300 350 400 450 500 550 600 650 700 750 800 850 900 950 1000 1050 1100

EXPLOIT DATABASE

Currently Archiving 12705 Exploits

Archive Last Updated: Sun Sep 19 2010

HOME | BLOG | ABOUT | REMOTE | LOCAL | WEB | DOS | SHELLCODE | PAPERS | SEARCH | SUBMIT | ARCHIVE

# MOAUB – 30 days of 0days, Binary Analysis and PoCs

17th August 2010 - by admin

The Abysssec Security Team is about to unleash its Month Of Abysssec Undisclosed Bugs on us. Starting on the 1st of September, Abysssec will release a collection of 0days, web application vulnerabilities, and detailed binary analysis (and pocs) for recently released advisories by vendors such as Microsoft, Mozilla, Sun, Apple, Adobe, HP, Novel, etc. The 0day collection includes PoCs and Exploits for Microsoft Excel, Internet Explorer, Microsoft codecs, Cpanel and others. The MOAUB will be hosted on the Exploit Database, and will be updated on a daily basis. Get your hard-hats on, your VM's and debugging tools organized – it's gonna be a an intensive ride. Follow both the exploit-db and Abysssec twitter feed to keep updated!

# Sample vulnerabilities (1)

- CVE 2010-3000: Multiple integer overflows in the ParseKnownType function in RealNetworks RealPlayer 11.0 through 11.1 and RealPlayer SP 1.0 through 1.1.4 on Windows allow remote attackers to execute arbitrary code in an .FLV file.
- CVE-2010-1900: Microsoft Office Word 2002 SP3, 2003 SP3, and 2007 SP2; Microsoft Office 2004 and 2008 for Mac; Open XML File Format Converter for Mac; Office Word Viewer; Office Compatibility Pack for Word, Excel, and PowerPoint 2007 File Formats SP2; and Works 9 do not properly handle malformed records in a Word file (.DOC), which allows remote attackers to execute arbitrary code or cause a denial of service (memory corruption) via a crafted file, aka "Word Record Parsing Vulnerability."

# Sample vulnerabilities (2)

- CVE-2010-1246: Stack-based buffer overflow in Microsoft Office Excel 2002 SP3 allows remote attackers to execute arbitrary code via an Excel file (.XLS) with a malformed RTD (0x813) record, aka "Excel RTD Memory Corruption Vulnerability."
- CVE-2010-1681: Buffer overflow in VISIODWG.DLL before 10.0.6880.4 in Microsoft Office Visio allows user-assisted remote attackers to execute arbitrary code via a crafted .DXF file, a different vulnerability than CVE-2010-0254 and CVE-2010-0256.
- CVE-2008-2238: Multiple integer overflows in OpenOffice.org (OOo) 2.x before 2.4.2 allow remote attackers to execute arbitrary code via crafted EMR records in an .EMF file associated with a StarOffice/StarSuite document, which trigger a heap-based buffer overflow.

# Sample vulnerabilities (3)

- CVE-2010-0188: Unspecified vulnerability in Adobe Reader and Acrobat 8.x before 8.2.1 and 9.x before 9.3.1 allows attackers to cause a denial of service (application crash) or possibly execute arbitrary code via unknown vectors (.PDF file).
- Media Player Classic v6.4.9.1: Buffer Overflow Exploit for .AVI file
  http://www.exploit-db.com/exploits/11535/
- Corel VideoStudio Pro is prone to a remote buffer-overflow vulnerability because the software fails to perform adequate boundary checks on user-supplied data (.MP4 file)

  http://www.securityfocus.com/bid/40963/discuss
- CVE-2010-0718: Buffer overflow in Microsoft Windows Media Player 9 and 11.0.5721.5145 allows remote attackers to attack via a crafted .MPG file

# Sample vulnerabilities (4)

- CVE-2008-4434: Stack-based buffer overflow in (1) uTorrent 1.7.7 build 8179 and earlier and (2) BitTorrent 6.0.3 build 8642 and earlier allows remote attackers to cause a denial of service (crash) and possibly execute arbitrary code via a long Created By field in a .TORRENT file.
- Media Player Classic - v 1.3.1774.0 (.RM file) Buffer Overflow

  http://www.exploit-db.com/exploits/12704/
- CVE-2009-4195: Buffer overflow in Adobe Illustrator CS4 14.0.0, CS3 13.0.3 and earlier, and CS3 13.0.0 allows remote attackers to execute arbitrary code via a long DSC comment in an Encapsulated PostScript (.EPS) file.
- OSVDB-ID 64984: Easyzip 2000 v3.5 (.zip) 0day stack buffer overflow PoC exploit. When the application receives a malicous '.ZIP' file it fails to properly sanitize the 'filename' section on the zip resulting in a stack based buffer overflow.

# Demo

# Detect malicious documentations

- Not-yet the target of current malware scanners :-(
    - Mostly focus on EXE files
    - Or can only understand and detect malicious files in popular formats (PDF, DOC, …)
    - But any file format can be vulnerable :-(
        - .FLV, .XLS, .PPT, .DXF, .EMF, .XML, .PDF, .WAV, .OGG, .MP4, .MPG, .RM, .ZIP, .TORRENT, .EPS, ...
    - Zero-day exploitation?

# A dream scanner + analyzer

- Can detect malicious attack from documentation files
    - Support all kind of file formats
- Can report if a particular version of application is vulnerable or not
- Can analyze the attack behaviour
    - How the exploitation work?
    - What the exploitation does?
        - Modify system to do something evil?
        - Download + install malware?
        - Setup backdoors?
        - Steal information?

# Assumption

- Given a random file of a random format, how can we know:
  - Is it a malicious file?
  - Which application and versions are vulnerable?
  - How the exploitation work?
  - What the exploitation does?
    - Modify system to do something evil?
    - Download + install malware?
    - Setup backdoors?
    - Steal information?

**Idea?**

# Approach

- Open a suspected file with corresponding application
  - For ex, .PDF file can be open with Adobe Reader (with any interested version) or Foxit (any interested version)
- Detect if the suspected file exploit its application
- Monitor the exploitation behavior
  - To understand its exploitation process
  - To monitor the exploitation behavior to understand what it does at post-exploit stage

# D-Analyzer solution

- Run everything inside a VM
  - Open a given file with related application
- Perform dynamic tainting analysis on VM to detect exploitation
- Instrument the VM to monitor vulnerable application and understand exploitation behavior

# Dynamic tainting analysis

- ”In order for an attacker to change the execution of a program illegitimately, he must cause a value that is normally derived from a trusted source to instead be derived from his own input” (James Newsome et al, Usenix Security 2005)
- Tainted data: data come from untrusted input
- Tracking flow of tainted data offers multiple advantages
  - Understand in detail how data is (ilegally) used
  - Detect unknown (Zero-day) attack is possible

# Basic concepts

- Tainted source: data originated from untrusted input
- Taint propagation: monitor data flow, and mark related data as tainted
- Taint sink: tainted data is consumed in illegal way?

→ detect when data is illegally used, not written

# Tainting propagation

- Two taint targets
  - Memory
    - Area of memory is tainted or not
    - Represented by start memory addr & range size
  - Hardware register
    - Regular registers: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP
    - Other registers (segment registers, control registers, …) can be ignored

# Sample tainting propagation

- Moving data insn (MOV, XCHG, ...)
  - MOV EAX, EBX
    - EBX is tainted → EAX becomes tainted
    - EBX is untainted → EAX becomes untainted
- Algorithm insn (ADD, SUB, XOR, …)
  - ADD EAX, EBX
    - EAX becomes tainted if EBX is tainted

# Sample tainting propagation (2)

- ”Constant” insn
  - XOR EAX, EAX
  - SUB EAX, EAX
    - Always untaint EAX, no matter what
- Algorithm insn (ADD, SUB, XOR, …)
  - ADD EAX, EBX
    - EAX becomes tainted if EBX is tainted

# Sample taint sinks

- Jmp & Call insn
  - Jmp or Call to tainted memory/register?
    - EIP is tainted!
- Format string functions
  - Format string argument is tainted, with dangerous format specifier like %n ?
- Sensitive system functions (like WinExec, CreateProcessA, CreateRemoteThread, ….)
  - Function arguments are tainted?

# Tainting policy

- Taint source?
  - Network data, File, memory ranges, keyboard, … ?
- Taint propagation?
  - In a particular case, taint corresponding memory/register or not?
  - Fine-grain level of taint tracking:
    - bit/byte/word/dword, page level?
- Taint sinks?
  - Where to have taint sinks?
  - What to do when tainted data is used at taint sink?

# Implicit data flow problem

- Result data is indirectly influenced by tainted data
  - Data assignment based on branch insn

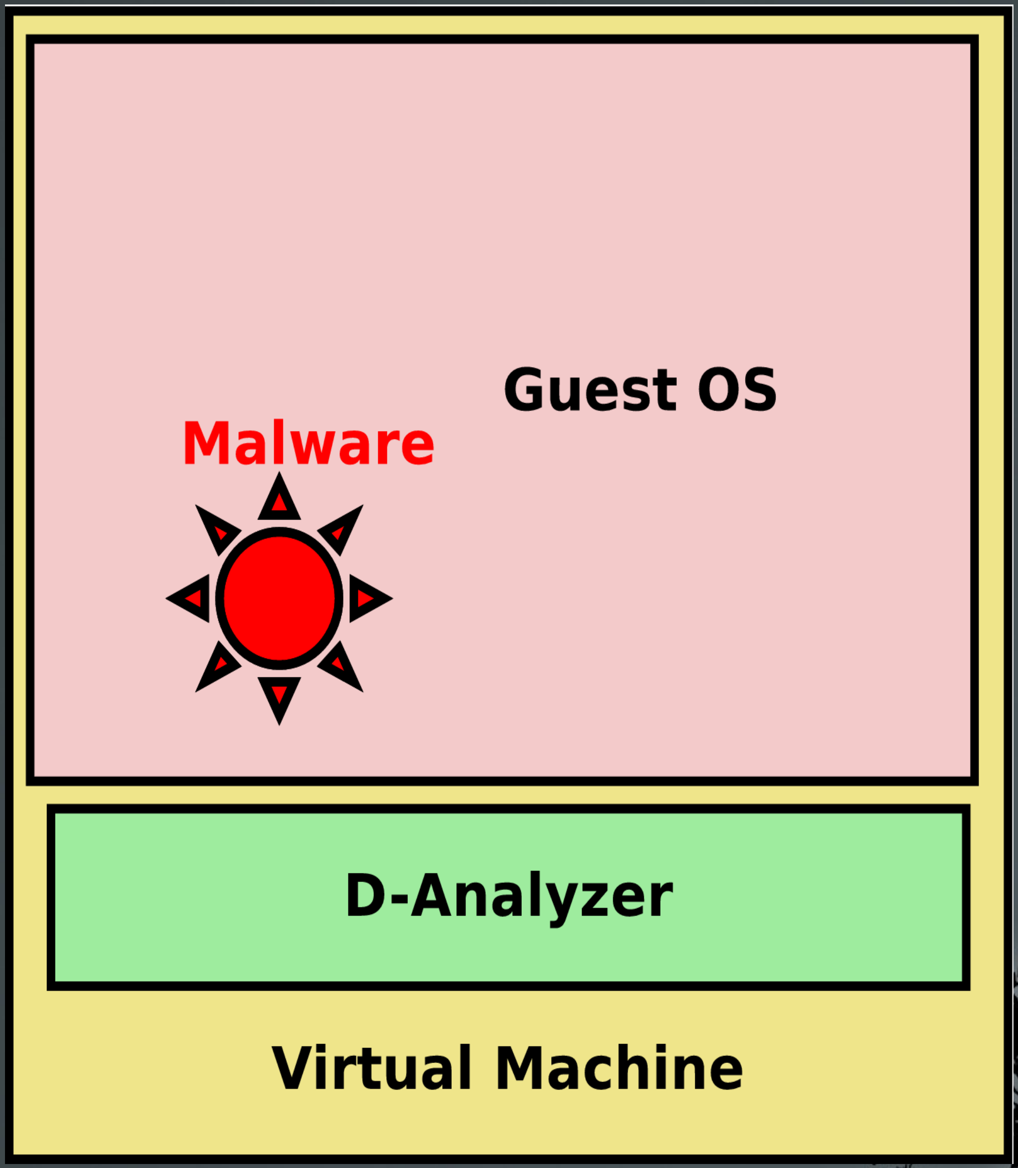
    ```
    if (a == 1) b = 1;
    else  if (a == 2) b = 2;
    ```

  - Tainted data is used as index of an array (widely used in table lookup)

    ```
    c = string[x];
    ```

- A main source of false negative issue → evade tainting analysis

# D-Analyzer architecture



# Solve the proposed questions

- D-Analyzer can detect exploitation thanks to tainting analysis mechanism
- D-Analyzer can report vulnerable applications, and also their versions
  - Open suspected file with each application
- Understanding the exploitation process and exploitation behavior is possible thanks to whole-system instrumentation monitoring

# Add-on benefits

- D-Analyzer is (mostly) invisible to exploitation, so stealthily monitoring the exploitation process is feasible
  - Never run anything inside the VM
  - Perform instrumentation from below VM allows us to monitor everything, even ring-0 code
    - Analyzing kernel rootkit is possible

# D-Analyzer requirement

- Understand VM context from outside
- Instrument guest VM execution
  - Intercept execution anywhere/anytime
- Access to VM context
  - Read/write to VM memory
  - Read/write CPU status
- Performing tainting analysis on suspected application

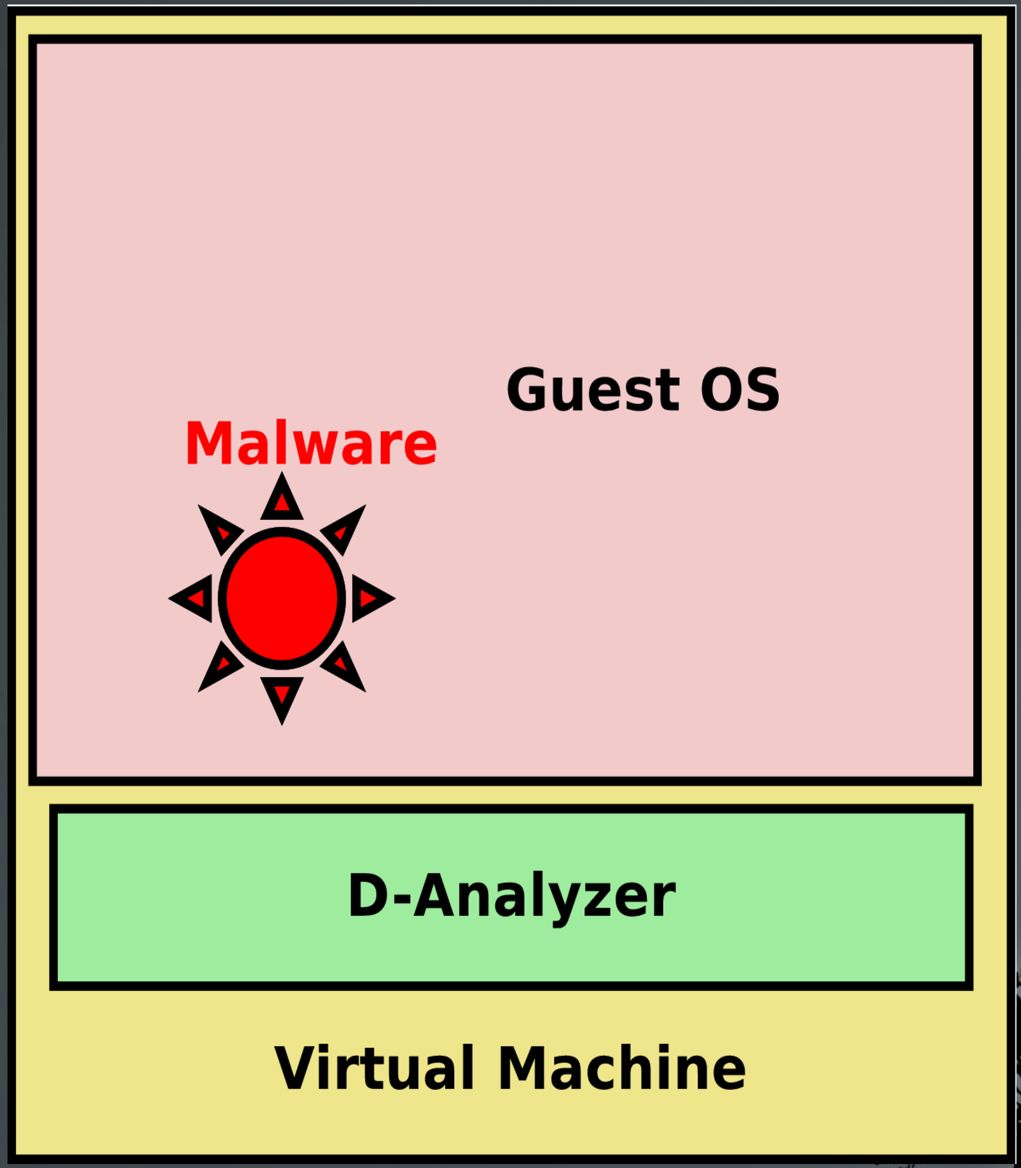
# Understand VM context

- Must be done from outside, without any support of guest VM
  - VM instrospection problem
  - Extract OS semantic objects from VM's memory
  - Support Windows OS

# VM for D-Analyzer

- Choose VM for D-Analyzer
  - Open source, so customizable (therefore VMWare is not suitable)
    - Xen? KVM?
    - VirtualBox?
    - Bochs?
    - Qemu?
      - Version 0.12.5

# D-Analyzer architecture



# Instrument guest VM

- Kobuta framework
  - Generic instrumentation framework
    - Not only for D-Analyzer, but other future projects
  - Instrument QEMU's dynamc binary translation process
    - Put hooks at right places to call out to external handlers
  - Support dynamic loaded module built on top of Kobuta
    - Module provides instrumentation handlers to be executed when called from hooks

# Instrument guest VM - Challanges

- Originally, QEMU provides no support for instrumentation
  - We are on our own, and have to build instrumentation framework from scratch
- QEMU uses Just-in-time (JIT) compiler to perform binary translation
  - Translated code is cached, and is not translated again if available in cache
    - We have to dig deeply into the translation process of QEMU to provide instrumental hooks

# QEMU JIT compiler

- Translate guest code to TCG IR, then translate TCG bytecode to native (host) code to execute on host
- The translated code is cached to be reused (to improve performance)
- Translation is done on code block basis
- CPU context (register values) is saved at the end of each block
  - So CPU context is synchronized at begining of each block
  - At middle of block, CPU context is out-of-synch

# Kobuta framework

- Hooking various events useful for generic purposes
- Fine grain instrumentation
  - Begin/end of instruction/block
  - Jump/call insn
  - Interrupt begin/end
  - Sysenter/Sysexit/Syscall/Sysret
  - Input/Output insn
  - Update control registers (CR0, CR3, CR4)
  - RDMSR, WRMSR (read/write to Model-Specific-Register)
  - Memory access (read/write)

# Kobuta module

- Need to register with Kobuta framework for interested instrumentation events
  - Then provide instrumentation handlers for those events
  - Handlers be executed when events happen in guest VM
- Leverage exported functions (from Kobuta framework) to manage guest VM from module
  - Pause and Resume VM on demand
  - Read and write to VM's memory (physical & virtual memory and CPU context
  - Single step mode
  - Dynamically enable/disable instrumentation events

# Kobuta module

- Design Kobuta module to be just a Dynamic Linked module
  - .so file in Linux, .DLL file in Windows
  - Loadable into Qemu process, and supported by OS services
  - Easy to implement your Kobuta module (just a normal DL module running in host OS)

# Manage Kobuta module

- Load module into Qemu process
  - Simply using DLL service provided by host OS
    - dlopen() in Linux
- Unload module from Qemu process
  - Also use DLL service of host OS
    - dlclose() in Linux
  - But how about code still running?

# Unloading Kobuta module

- Use reference counter for Kobuta instrumentation handlers
  - Increase counter before running a handler, and decrease it when done
  - Only run a hanlder when its module is in enable state
  - Have a manage thread to unload Kobuta module
    - Put module in disable state
    - Periodically checking for ref counter, and unload module when refcount = 0

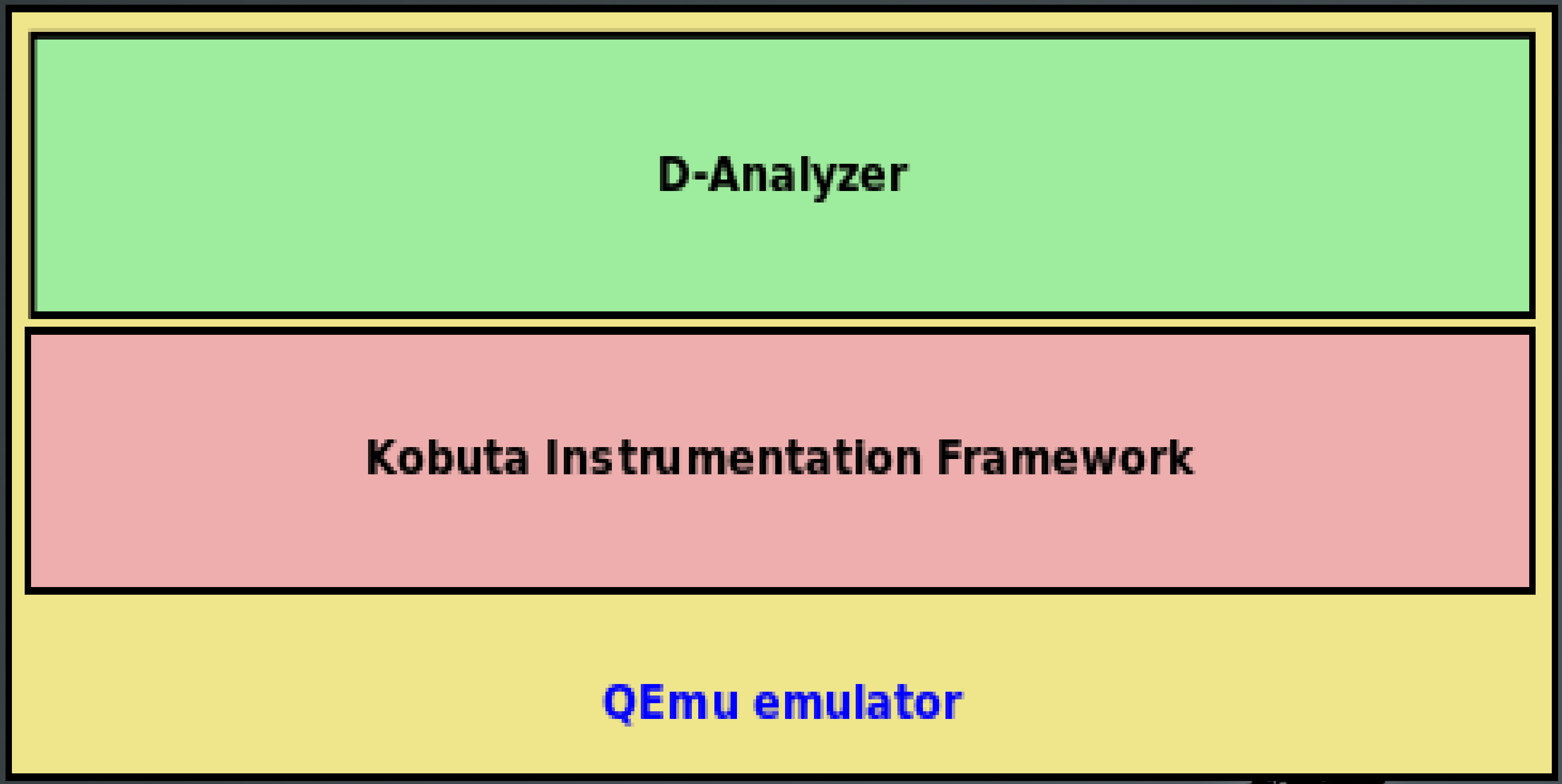
# D-Analyzer tainting analysis

- Taint source
  - A suspected file, open with related application
- Taint sinks
  - Jmp/Call target
    - Detect Buffer Overflow exploitation
  - Format string functions
    - Detect Format String exploitation
  - Sensitive functions (such as WinExec, CreateProcessA, CreateRemoteThread, …)
    - Detect false negatives

# D-Analyzer tainting analysis (2)

- Taint propagation
  - Usual policy for moving data insn, algorithm insn
  - Tainting table lookup result if index register is tainted (implicit data flow tracking problem)
  - Tainting output data by modeling function logics

# D-Analyzer module architecture



# D-Analyzer implementation

- A Kobuta module
- Register several instrumentation events
  - JmpCall
    - To capture interested function execution & their parameters
    - To handle sinks as Jmp/Call targets
  - Begin/end insn
    - To perform tainting analysis on instruction level
  - Update CR3
    - To know if we are switching to our monitored processes

# D-Analyzer implementation (2)

- Employ several tactics to speed up tainting analysis process
  - On-demand instrumentation
    - Turn off insn level instrumentation when switching to other processes
    - Turn off insn level instrumentation on ”uninteresting” functions, which produce no output nor modify parameters
  - On-demand tainting analysis by modeling function logic (such as string functions or File related functions)
    - Stop tracking functions at function entrance, and resume when function returns. Tainting its output and parameters according to function logic

# D-Analyzer implementation (3)

- Tracking data flow from given input file
  - Open (tainted) input file with the related application
- Report if tainted data is illegally used at sinks
  - Conclude this file is malicious → report
  - Also this particular application (of this particular version, too) is vulnerable with this malicious file
- Repeat the checking process with all the requested applications, on the same input file

# Discussion

- Implicit data flow is not a major problem for us
  - Can be mostly handled by tainting public functions
  - Unlike malware, legitimate applications never try to evade tainting analysis :-)
    - False negative is greatly reduced
- Control flow corruption problem is solved, but exploitation targeting data & data pointers remain a problem
  - Partly solved by monitoring sensitive functions

# Future plan - Development

- Improve performance
  - Other tricks to speed up taiting analysis
  - Using KVM to speed up emulation even further
    - Even currently, KQEMU is not too bad, either
- Track down the culprit area in malicious input file
- Focus on exploitation corrupting data & data pointers
- Extract shellcode and analyze exploitation behavior

# Related works

- BitBlaze project
  - Based on old QEMU version (0.9.1)
  - Too slow due to whole system taint tracking
  - File input as tainted source does not seem to be supported
- Joedoc.org
  - Limited documentation files and applications
  - Signature-based detection
- Vicheck.ca
  - Unknown technique

# Conclusions

- D-Analyzer is an effective scanner for malicious documentations
  - Support all kind of file formats
  - Can detect zero-day malicious attacks
  - Analyze malicious attacks is possible