Reliable Library Identification Using VMI Techniques

Nick de Bruijn
Leandro Velasco

University of Amsterdam
Faculty of Physics, Mathematics and Informatics
MSc System and Network Engineering
Research Project: 1

February 7, 2017

Introduction

- Enhance cloud security
- Vulnerabilities in libraries can have major consequences
- Efficient way of detecting vulnerabilities in libraries is needed

Research Question

To which extent can one reliably identify the version of a selected running library using the VMI techniques provided by LibVMI?

How can one identify a running library in a VM where the library name can not be trusted?

Related Work

Related work virtual machine introspection:

- 2003, A virtual machine introspection based architecture for intrusion detection. In NDSS, volume 3, pages 191 - 206. Tal Garfinkel, Mendel Rosenblum, et al.
- 2012, Simplifying virtual machine introspection using libVMI. Sandia report, pages 43 - 44. Bryan D Payne.
- 2016, Vmicvs: Cloud vulnerability scanner. Anil Kumar Konasale Krishna and Robert Ricci.

Related work library identification:

- 2017, Automatic Library Version Identification, an Exploration of Techniques Thomas Rinsma.

Virtual machines

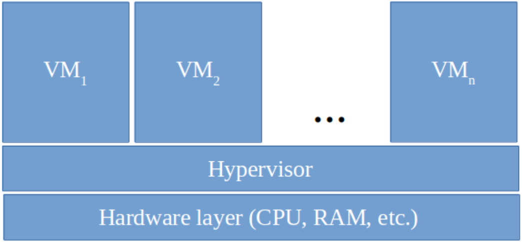


Figure: Vitual Machine Architecture[1]

- Hypervisor has access to the binary representation of the virtual memory used by the OS running inside the virtual machine

[1] http://www.cse.wustl.edu/ jain/cse571-09/ftp/vmsec/index.html

Semantic gap (1/2)

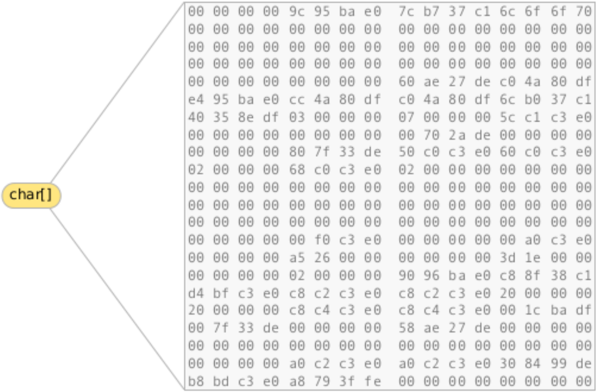


Figure: Memory from the hypervisor's perspective[2]

[2] C. A. Schneider. Full Virtual Machine State Reconstruction for Security Applications, 2013

Semantic gap (2/2)

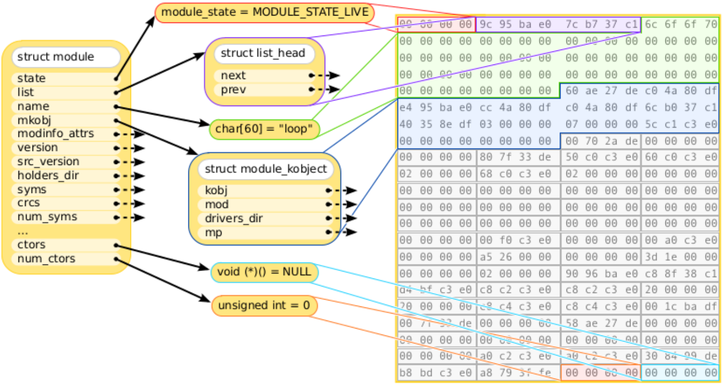


Figure: Memory from the guest's OS perspective[3]

[3] C. A. Schneider. Full Virtual Machine State Reconstruction for Security Applications, 2013

Virtual machine introspection

- Method to interpret/translate the hypervisor's perspective
- Knowledge of the guest's OS is needed

LibVMI (1/3)

1. A virtual machine introspection library based on XenAccess
   (64-bit VM guest support, KVM support, fixes on bugs and memory leaks)
2. Provides a useful application programming interface (API) for reading and writing to a virtual machines memory
3. Access memory using physical addresses, virtual addresses, or kernel symbols
4. Overcomes the semantic gap by providing the lacking information
   (OS type, location of symbolic information, offsets used to access data)

LibVMI (2/3)

1. Request to view kernel symbol
2. LibVMI finds the virtual address for kernel symbol
3. Kernel page directory mapped to find correct page table
4. Page table mapped to find correct data page
5. Data page returned to LibVMI Library
6. LibVMI returns the data requested

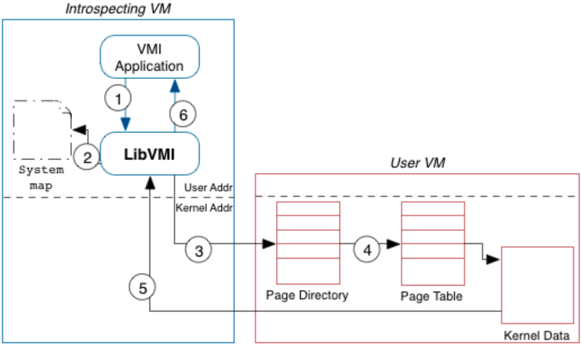


Figure: LibVMI memory mapping[4]

[4] http://libvmi.com/docs/gcode-intro.html

LibVMI (3/3)

Result of LibVMI:

1. Mapped virtual memory view
2. Access to the virtual memory

Library identification (1/3)

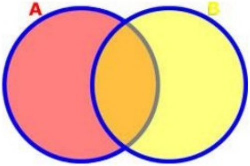
1. Version number extracting
   - Extract library version from its name or binary
2. Behaviour based identification
   - Look at behaviour of the library (system calls, wrapper functions)
3. Fingerprint identification
   - Extract information from a binary to create a fingerprint
   - Strict vs Fuzzy fingerprints

Library identification (2/3)

Printable strings:

- Uses a set of printable strings extracted from the library executable
  (Error messages, copyright or usage information)
- Tian et al. show that such a list of strings can be an accurate signature of an executable object when used for malware classification
- Thomas Rinsma concludes this to be the most efficient method to identify libraries
- Printable strings can be extracted by using Unix **strings** command
- Measure similarity of sample sets using the **Jaccard index**:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$



Library identification (3/3)

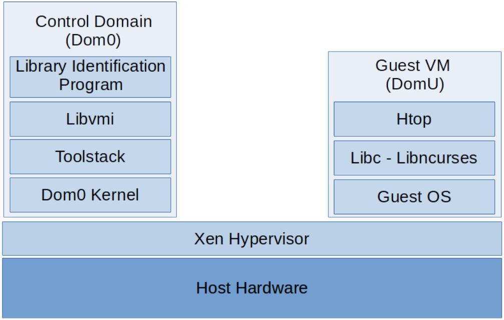
```
setrpcent
__progname
mbrtoc32
_IO_free_backup_area
creat
setnetent
wcschr
__strxfrm_l
posix_spawn_file_actions_addclose
argp_err_exit_status
getgrgid_r
__vfwprintf_chk
unshare
_seterr_reply
__recv_chk
_IO_getline_info
__fwriting
__finitel
_itoa_lower_digits
inet6_opt_finish
pthread_cond_init
_IO_default_xsputn
```

Figure: Example of strings obtained with the Unix command strings

Experimental Environment

The environment consist of:

- Privileged Host Dom0, in charge of performing the introspection
- Guest VM, system that will be introspected



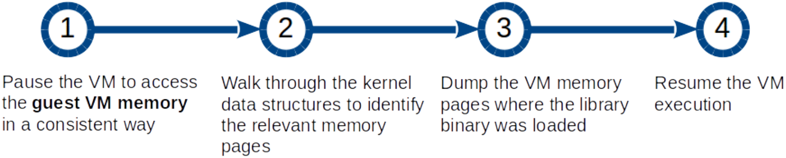
Library Identification Program Design

The program consist of the following components:

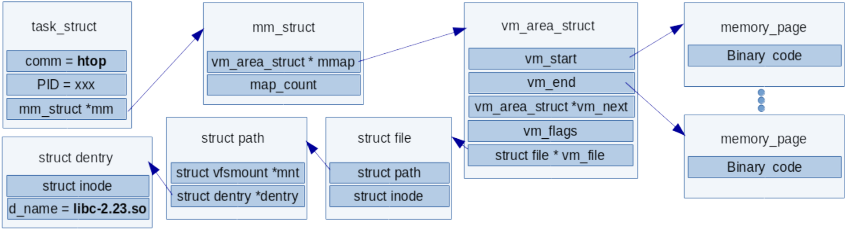
- **Library extractor:** This module handles the **introspection** aspects required to extract the library binary from the guest VM memory. It does so by making use of LibVMI
- **Library Identifier:** This module generate the **fingerprint** of the selected library and then compares it against the reference data base
- **Reference Data Base:** It contains 151 fingerprints from different versions of different libraries

Library Extractor Implementation

This module is in charge of:



Kernel Data Structures:



Library Identifier Implementation

This module is in charge of:

- Generate a fingerprint from the extracted library. This is done by executing the Unix command **Strings**.
- Calculate the Match Score for each fingerprint in the reference DB
    - $MatchScore = \frac{|Sample \cap Reference|}{|Sample \cup Reference|}$
- Sort the results and return the top five Match Scores

Reference Data Base Creation

The following step were followed to create the DB:

1. Download the **source code** from different versions of different libraries. Including the ones that will be tested (**libc** and **libncurses**)
2. Build the different libraries by only passing the argument
   - - *prefix*=<*directory*>
3. Generate a fingerprint for each share object created during the building procedure. This is done by executing the Unix command **Strings**

Library Identification Program Output

Match	Fingerprint in the DB
20.59%	libc-2.23.so.strings
19.73%	libc-2.22.so.strings
19.71%	libc-2.24.so.strings
19.34%	libc-2.21.so.strings
18.78%	libc-2.20.so.strings
18.25%	libc-2.19.so.strings
3.56%	libjpeg.so.9.2.0.strings
2.91%	libncurses.so.5.9.strings

Table: Output for libc-2.23

Match	Fingerprint in the DB
15.50%	libncurses.so.5.9.strings
15.47%	libncurses.so.5.8.strings
15.20%	libncurses.so.5.7.strings
14.00%	libncurses.so.6.0.strings
4.89%	libjpeg.so.9.2.0.strings
4.65%	libmenu.so.6.0.strings
4.48%	libresolv-2.23.so.strings
4.41%	libresolv-2.24.so.strings

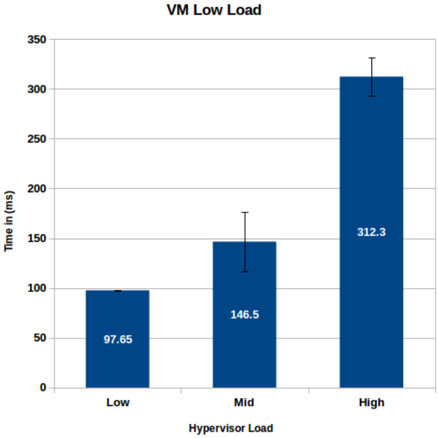
Table: Output for libncurses-5.9

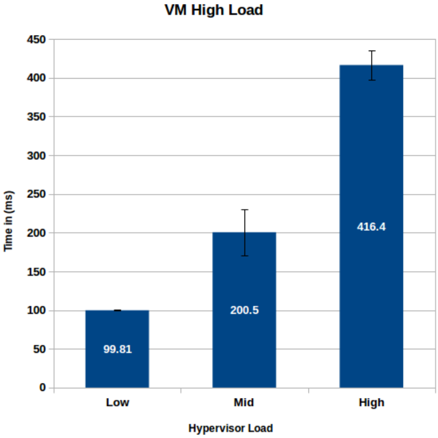
- The low match scores are due to the way the DB was built and the fact that some pages may be swapped out
- The match score obtained with the original .so that was loaded in memory is : 97.06%
- Less than 9% is considered a mismatch

Efficiency and Effectiveness Experiments Design

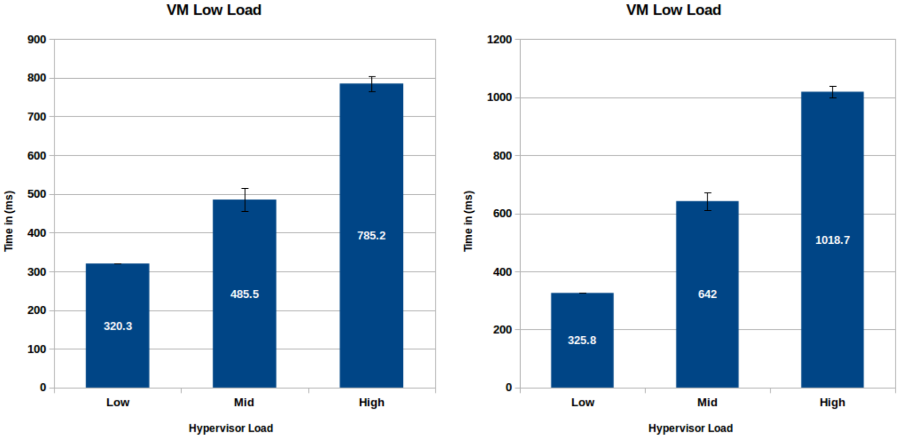
- The program was executed 100 times per load configuration and per library (libc-2.23 or libncurses-5.9)
- Each load configuration represent either the hypervisor's CPUs or the guest VM's CPUs stressed at 0% (low), 50% (mid) or 100% (high)
- Data gathered during the experiments:
  - Pause Time
  - Identification Time
  - Memory Usage
  - CPU Usage
  - Match Score
- For each of the above values the mean and the standard deviation was calculated
- Two extra experiments were executed in which either the hypervisor's memory or the guest VM's memory was stressed at 100%

Pause Time Results - libc





Identification Time Results - libc



Effectiveness Results

- Match scores are not affected by the CPU load
- However they are affected by the memory load as shown in the following table:

Library	Not Stressed Memory	VM Memory at 100%	Xen Memory at 100%
libc	20.606% $\pm$0.007	20.585% $\pm$0.007	20.248% $\pm$0.005
libncurses	15.500% $\pm$0.000	15.500% $\pm$0.000	15.489% $\pm$0.020

Table: Effectiveness Under Heavy Memory Load

Library Tampering Experiments (1/2)

- Are the strings containing version information relevant to the library identification?

```
GLIBC_2.22
GLIBC_2.23
GLIBC_2.24
glibc 2.24
NPTL 2.24
GNU C Library (Ubuntu GLIBC 2.24-0ubuntu5)
    stable release version 2.24, by Roland
    McGrath et al.
```

Figure: Example of strings containing version information of libc-2.24

- Manually tamper the sample fingerprint to include strings containing version information of libc-2.24

Sample Fingerprint	Libc-2.23 Ref. Fingerprint	Libc-2.24 Ref. Fingerprint
libc-2.23 original	20.60%	19.82%
libc-2.23 tampered	20.59%	19.83%

Library Tampering Experiments (2/2)

- Remove every string containing version information from the sample and reference fingerprint

Match	Fingerprint in the DB
20.59%	libc-2.23.so.strings
19.73%	libc-2.22.so.strings
19.71%	libc-2.24.so.strings
19.34%	libc-2.21.so.strings
18.78%	libc-2.20.so.strings
18.25%	libc-2.19.so.strings

Table: Normal Scenario

Match	Fingerprint in the DB
20.54%	libc-2.23.so.stripped
19.70%	libc-2.22.so.stripped
19.68%	libc-2.24.so.stripped
19.31%	libc-2.21.so.stripped
18.74%	libc-2.20.so.stripped
18.22%	libc-2.19.so.stripped

Table: Stripped Scenario

Implementation Limitations

1. Unix only
2. One to many comparison
3. Dynamically linked libraries only
4. Identification time directly depend on the amounts of records in the reference data base
5. LibVMI offsets requires guest kernel access
6. Swapping of memory pages affect the results
7. When a library that is not included in the reference data base goes through the identification process, false positives can be observed

Conclusion

1. LibVMI can be used to efficiently extract libraries from the VM's memory
2. Printable strings can be used as fingerprints to accurate identify a library when the library is in the database
3. Performance measurements shows that our implementation perform in a reasonable manner, even under high system load
4. Accuracy of identification was not effected by the load of the systems

Future work

- Explore ways to;
  - improve the database creation to obtain better matching results
  - improve the scalability of the program
  - identify library behaviour using VMI techniques
- Extend the functionality of our program to support vulnerable library scanning