Atomizer: Fast, Scalable and Lightweight Heap Analyzer for Virtual Machines in a Cloud Environment

Salman Javaid  Aleksandar Zoranic  Irfan Ahmed
Golden G. Richard III

University of New Orleans
Greater New Orleans Center for Information Assurance (GNOCIA)
Layered Assurance Workshop

December 4, 2012
Outline

- Introduction
- Atomizer Architecture
- Implementation
- Evaluation
- Performance Analysis
- Conclusion
Introduction

Heap-based Attacks

- Heap Sprays [3]
- JIT (just-in-time) Sprays [7]

Possible Solutions

- NOZZLE [11]
- ZOZZLE [9]
- Cova et al. JSAND [8]
- LeMasters. Heap Inspector Tool [10]
Atomizer

- Heap Analysis Framework particularly applicable for Cloud Environment
- Uses Virtual Machine Introspection (VMI) to seamlessly access process heap
- Modular Design Approach which allows to implement various heap analysis tools
- Atomizer has minimum effect on the cloud server
- Atomizer cannot be detected and disabled by in-guest VM malwares
Atomizer Architecture

Key Architecture Component

- Process Information Extractor
- Heap Extractor
- Swapped Heap Page Extractor
Atomizer Architecture

Figure: Atomizer Architecture
Implementation

Platform

- XEN [6]
- Fedora 16 (64 bit)

Libraries Used

- LibVMI [5]
- LibGuestFS [4]
Heap Extraction

**Extract Important Addresses**

1. Search for PEB (Process Environment Block)
2. Extract Heap Address and Number of Heaps

**Extract Heap via VAD (Virtual Address Descriptor) Tree**

1. Simple Binary Tree Browse
2. Both Page by Page and Byte by Byte Extraction
Swapped Heap Page Extractor

- Uses the same procedure as OS
- Two Step Process

**Swapped Page Extraction**

- **Step 1**
  - Retrieve the page file number and the page offset
  - Obtain the value of the virtual address from the page file
- **Step 2**
  - Use the LibguestFS to access the page file
Simple NOP Sled Detection

1. Page by Page Heap Analysis used
2. Byte by Byte Analysis of each Page for sequences of NOPs/NOP replacements
3. Uses a hash table of NOPs/NOP replacements
Evaluation

Experiment Settings

- Simple Cloud Environment
  - Quad Core i7 (2.67 GHz) server with 18 GB of RAM
  - Fedora 16 (kernel 3.3.2-6) along with Xen 4.1.2 (Dom0)
  - Five VM clones (DomU: Dom1–Dom5) 32 bit Window XP (SP2)

Malware Detection

- Random Polymorphic NOP Sled Detection
- Skypher heap spray generator [2]
- Aurora [1]
- Heap Feng Shui [12]
(a) 1–5 Idle VMs running

(b) 1–5 VMs with light load & Atomizer

**Figure:** CPU Performance (CPU usage in Dom0)
Atomizer can be easily extended by implementing new detection methods for any type of heap-based attacks.

Experimental results show that Atomizer successfully detects various heap spray attacks and randomly generated polymorphic NOP sled samples.

Further work is required to improve the performance of our method, via a multi-threaded implementation of Atomizer.
References

Heap spray exploit tutorial: Internet explorer use after free aurora vulnerability.

Heap spray generator.

Internet explorer iframe src&name parameter bof remote compromise.

Libguestfs.
http://libguestfs.org/.

Libvmi.
http://code.google.com/p/vmitools/.

XEN.
http://www.xen.org/.

D. Blazakis.
Interpreter exploitation: Pointer inference and JIT spraying.

M. Cova, C. Kruegel, and G. Vigna.
Detection and analysis of drive-by-download attacks and malicious javascript code.
C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert.
ZOZZLE: Low-overhead mostly static javascript malware detection.

A. LeMaster.
Heap spray detection with heap inspector.
In Blackhat USA, Las Vegas, Nevada, US, 2011.

P. Ratanaworabhan, B. Livshits, and B. Zorn.
NOZZLE: A defense against heap-spraying code injection attacks.

A. Sotirov.
Heap fang shui in javascript.
In BLACK HAT Europe, Amsterdam, Netherlands, March 2007.
Algorithm 1 Heap Memory Browsing using VAD tree

```plaintext
for i = 0x7FFD0000 to 0x7FFDF000 do
    if ((5 == i + 0xa4) && (1 == i + 0xa8)) then
        PEB = i
        break;
    end if
end for

HEAPNUM := PEB + 0x88
HEAPADDRESS := PEB + 0x090
heapCounter := 0

while heapCounter < HEAPNUM do
    HEAPNODE := HEAPADDRESS + (4 * heapCounter)
    segmentCounter := 0
    while segmentCounter < 64 do
        HEAPSEGMENT := HEAPNODE + 0x58 + (4 * segmentCounter)
        HEAPENTRY := HEAPSEGMENT + 0x20
        while (HEAPENTRY + 0x05) ≠ 0 do
            HEAPSIZE := HEAPENTRY
            READ_MEMORY(HEAPENTRY, HEAPSIZE)
            HEAPENTRY := HEAPENTRY + (HEAPSIZE * 8)
        end while
        segmentCounter++
    end while
    heapCounter++
end while
```
**Algorithm 2** Simple NOP Sled Detection

\[
\begin{align*}
\text{NOPZ} & \leftarrow \text{HASH-TABLE of NOPs/NOP-replacements} \\
\text{LIMIT} & \leftarrow 150 \\
\text{BUFFER} & \leftarrow \text{Memory buffer from Heap size} = \text{SIZE} \\
\text{SKIP} & \leftarrow 1 \\
\text{index} & := 0 \\
nops & := 0 \\
\text{skipped} & := 0 \\
\text{while} \index < \text{SIZE} \text{ do} & \\
\quad \text{if} \ \text{NOPZ[ BUFFER[index++]] then} & \\
\quad \quad \text{nops}++ & \\
\quad \text{else if} \ \text{skipped} < \text{SKIP then} & \\
\quad \quad \text{skipped}++ & \\
\quad \text{else} & \\
\quad \quad \text{nops} := 0 & \\
\quad \text{end if} & \\
\quad \text{if} \ \text{nops} == \text{LIMIT then} & \\
\quad \quad \text{NOP sled detected} & \\
\quad \text{end if} & \\
\quad \text{end while} &
\end{align*}
\]