Effectiveness of PaX ASLR & ASLR in Vista

Amanda Crowell
February 15, 2011
ASLR – General Idea

• Randomize addresses of
  – Portions of executables and memory
  – Dynamic libraries

• To make the attacker have to
  – *guess* addresses (libc functions)

• Single attack vector not applicable on all instantiations
Effectiveness of (PaX) ASLR
PaX ASLR

- Patch for Linux kernel
- Goal: least privilege for memory

- Provisions:
  - Data memory non-executable
  - Program memory non-writable
  - Program memory randomly arranged

Source: wikipedia
Process User Address Space

- Each randomized separately
  - Base address + Random Offset
  - At process creation
- For Intel x86
  - Executable: 16 bits
  - Mapped: 16 bits
  - Stack: 24 bits
Weaknesses

• Only base address randomized
  – Once you know the delta value, you win
  – Only as strong as the entropy of the randomness
    • 16 bits = $2^{16}$ guesses (65,536)

• Randomness only applied at start-up
  – Once you know the delta value, you win
The Attack (return-to-libc)

• Target: Apache web server
  – Forks child processes on requests

• Method: Typical buffer overflow
  – Guess address of the `usleep()` function
  – To find the delta for the mapped area

• Measure:
  – Success, connection hang
  – Failure, child process crashes

• Experiment:
  – 10 trials
  – Average 216 seconds, Max 810, Min 29
Suggested Improvements

• 64-bit architecture
  – Lots of entropy (40 bits)
  – Brute force attack will get attention

• Randomize more frequently
  – After each probe (non-forking server)
  – Only doubles number of guesses

• Different randomizations
  – Compile time
  – Add entropy
  – Reorder functions
Effectiveness of ASLR in Vista
ASLR in Vista

• Images (.exes & .dlls) must “elect”

Randomization Process:
1) Random image offset – @ reboot
   - 256 possibilities
2) Stack & stack pointer offset -- @ program start
   - 32 possibilities for stack, 16K for stack pointer
3) Heaps offsets -- @ program start
   - 32 possibilities for each, must avoid previous heaps
Paper’s View : Effectiveness

Three Factors:
1) Predictability of random memory layout
2) Exploit’s ability to handle randomization
3) Number of practical exploit attempts
What did they do?

- Program that prints addresses to measure randomization
  - Image base: address of function
  - Stack: address of automatic variable
  - Heaps
    - CRT: first value returned by malloc
    - Initial Heap: first value returned by HeapAlloc
    - Other Heap: address selected with CreateHeap
- Ran lots of tests, analyzed the results
How good is...image base selection?

• Unique values
  – Expected 256, observed 255

• Number of duplicates
  – Expected 45, observed 39
  – Suggests small bias (in selection of address)

• Summary
  – More predictable (easy to guess)
How good is...stack location selection?

• Unique Values  
  – Expected 16K, observed 8.6K

• Number of Duplicates  
  – Expected <1, observed 1

• In general, not enough samples to draw conclusions

• However, distribution near uniform  
  – *unpredictable*
How good is...heap location selection?

<table>
<thead>
<tr>
<th></th>
<th>Unique Values</th>
<th>Duplicate Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Expected</td>
<td>Observed</td>
</tr>
<tr>
<td>Malloc</td>
<td>&gt;= 32</td>
<td>192</td>
</tr>
<tr>
<td>HeapAlloc</td>
<td>&gt;= 32</td>
<td>95</td>
</tr>
<tr>
<td>CreateHeap</td>
<td>&gt;= 32</td>
<td>209</td>
</tr>
</tbody>
</table>

Unique Values Observations:
- More variability than expected
- Microsoft’s HeapAlloc less varied than malloc

Duplicate Values Observations
- More observed malloc duplicates than expected, so biased
- HeapAlloc observed duplicates not uniformly distributed, so biased
Final Observations for Vista ASLR

• Not as *robust* as expected
  – Weakness in implementation by Microsoft

• Microsoft HeapAlloc() function not as “random” as malloc()}