Not-So-Random Numbers in Virtualized Linux and the Whirlwind RNG

Adam Everspaugh, Yan Zhai, Robert Jellinek, Thomas Ristenpart, Michael Swift
University of Wisconsin — Madison
Random Number Generators

Inputs
system events

RNG

Outputs
uniformly distributed
Random Number Generators

Inputs
system events

RNG

Outputs
uniformly distributed

Linux /dev/(u)random:

Interrupt Pool
interrupt events

Input Pool
disk events
keyboard events
mouse events
hardware RNGs

Random Pool
/get_random_bytes()

URandom Pool

/dev/random

/dev/urandom

Cryptographic hash
Random Number Generators

Inputs
system events

RNG

Outputs
uniformly distributed

Previous Analyses (mostly showing failures)

Cryptanalysis of Windows RNG [DGP07]
Factorable RSA Keys [HDWH12]
/dev/random not Robust [DPRVW13]

Linux RNG [GPR08]
Linux RNG Revisited [LRSV12]

Taiwan National IDs [BCCCHLS13]
RNGs in Virtualized Environments

1. Desktop virtualization, data center virtualization, and cloud computing are increasingly popular.

2. RNGs designed without virtualization in mind (1990s)

Are system RNGs secure in virtualized environments?
RNGs in Virtualized Environments

- Interrupt Pool
  - Interrupt events
  - disk events
  - keyboard events
  - mouse events
  - hardware RNGs
- Input Pool
- Random Pool
  - /dev/random
- URandom Pool
  - /dev/urandom
  - get_random_bytes()
RNGs in Virtualized Environments

Virtual Machine

Interrupt Pool -> Input Pool
- Interrupt events
- disk events
- keyboard events
- mouse events
- hardware RNGs

Random Pool -> /dev/random

URandom Pool -> /dev/urandom

get_random_bytes()
RNGs in Virtualized Environments

Folklore concerns regarding security:

1. Do full-memory snapshots cause problems for system RNGS? [Garfinkel, Rosenblum 05] [Ristenpart, Yilek 10]

2. Are input sources entropy-poor inside a virtual machine? [Stamos, Becherer, Wilcox 09]
Our Contributions

• First study of system RNGs in modern virtualized settings

• Snapshots cause problems? ➔ YES
  Bad RSA keys from OpenSSL

• Entropy-poor inputs? ➔ NO

• New clean-slate RNG design ➔ Whirlwind
VM Use Cases

Boot-from-image

disk → Amazon EC2 → Rackspace → Microsoft Azure

Snapshot-Reset

disk → Snapshot → Resumption

vmware → Xen → Parallels
Security Problems with VM Resets

VM Reset Vulnerabilities [GR05] [RY10]

App starts
/dev/urandom

Initialization

Derives key

Read

[RY10] Suggested countermeasure:
Narrow gap between deriving and using random numbers

Snapshot

Use key

Use key

Are system RNGs reset secure?
Linux RNG *Not* Reset Secure

One of our experiments:
- Boot VM in Xen, idle for 5 minutes
- Start measurement process, capture snapshot
- Resume from snapshot, read 512-bits from /dev/urandom every 500 us

Repeat for 8 distinct snapshots
Do 20 resumptions/snapshot
/dev/urandom outputs after resumption

<table>
<thead>
<tr>
<th>21B8BEE4</th>
<th>21B8BEE4</th>
<th>21B8BEE4</th>
</tr>
</thead>
<tbody>
<tr>
<td>9D27FB83</td>
<td>9D27FB83</td>
<td>9D27FB83</td>
</tr>
<tr>
<td>6CD124A6</td>
<td>6CD124A6</td>
<td>6CD124A6</td>
</tr>
<tr>
<td>E8734F71</td>
<td>E8734F71</td>
<td>E8734F71</td>
</tr>
<tr>
<td>111D337C</td>
<td>111D337C</td>
<td>111D337C</td>
</tr>
<tr>
<td>1E6DD331</td>
<td>1E6DD331</td>
<td>1E6DD331</td>
</tr>
<tr>
<td>8CC97112</td>
<td>8CC97112</td>
<td>8CC97112</td>
</tr>
<tr>
<td>2A2FA7DB</td>
<td>2A2FA7DB</td>
<td>2A2FA7DB</td>
</tr>
<tr>
<td>DBBF058C</td>
<td>DBBF058C</td>
<td>DBBF058C</td>
</tr>
<tr>
<td>26C334E7</td>
<td>26C334E7</td>
<td>26C334E7</td>
</tr>
<tr>
<td>F17D2D20</td>
<td>F17D2D20</td>
<td>45C78AE0</td>
</tr>
<tr>
<td>CC10232E</td>
<td>CC10232E</td>
<td>E678DBB2</td>
</tr>
</tbody>
</table>

Reset 1  Reset 2  Reset 3
/dev/urandom outputs after resumption

Linux RNG is not reset secure:
7/8 snapshots produce mostly identical outputs

<table>
<thead>
<tr>
<th>Reset 1</th>
<th>Reset 2</th>
<th>Reset 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1E6DD331</td>
<td>1E6DD331</td>
<td>1E6DD331</td>
</tr>
<tr>
<td>8CC97112</td>
<td>8CC97112</td>
<td>8CC97112</td>
</tr>
<tr>
<td>2A2FA7DB</td>
<td>2A2FA7DB</td>
<td>2A2FA7DB</td>
</tr>
<tr>
<td>DBBF058C</td>
<td>DBBF058C</td>
<td>DBBF058C</td>
</tr>
<tr>
<td>26C334E7</td>
<td>26C334E7</td>
<td>26C334E7</td>
</tr>
<tr>
<td>F17D2D20</td>
<td>F17D2D20</td>
<td>45C78AE0</td>
</tr>
<tr>
<td>CC10232E</td>
<td>CC10232E</td>
<td>E678DBB2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Reset insecurity and applications

Generate RSA key on resumption:

```bash
openssl genrsa
```

30 snapshots; 2 resets/snapshot   (ASLR Off)
- 27 trials produced **identical** private keys
- 3 trials produced unique private keys
Why does this happen?

Buffering and thresholds prevent new inputs from impacting outputs

Linux /dev/(u)random
An even more extreme case: Entropy Starvation Attack

Prevents new inputs from ever reaching /dev/urandom

No entropy in /dev/urandom outputs

Read from /dev/random

```
if (entropy estimate >= 64)
if (count > 64 or elapsed time > 1s )
```

```
get_random_bytes()
```
Reset Vulnerabilities Effect Other Platforms

 FreeBSD
/dev/random produces **identical** output stream
Up to 100 seconds after resumption

Microsoft Windows 7
Produces **repeated** outputs indefinitely
rand_s (stdlib)
CryptGenRandom (Win32)
RngCryptoServices (.NET)
Reset vulnerabilities summary

Using snapshots can compromise security of applications relying on system RNGs

Many different VM platforms / operating systems

Infrastructure-as-a-service providers don’t yet support full-memory snapshots, but could in the future.
Our Contributions

- First study of system RNGs in modern virtualized settings
- Snapshots cause problems? YES
- Entropy-poor inputs?
- New clean-slate RNG design — Whirlwind
Our Contributions

• First study of system RNGs in modern virtualized settings

• Snapshots cause problems? ➔ YES

• Entropy-poor inputs?

• New clean-slate RNG design — Whirlwind
Estimating Input Entropy

- Instrumented Linux RNG
- Collected all inputs, outputs on boot
- Gathered data from: native, Xen, VMware, and EC2
- Statistical hypothesis testing to estimate entropy of each input
- Use input entropy to estimate security of each output
RNG Operation

Initial state

Input 1

Input 2

Input 3

Cryptographic hash function

Output
Attacker’s View

Initial state

Input 1
Input 2
Input n

Output

h
Attacker’s View

- Branching factor:
  - estimated entropy per input
  - focus on LSBs of cycle counters

- Depth = # inputs

# of nodes in tree = Complexity of an attack
= Security of an output

Initial state

Input 1
Input 2
Input n
## Results: Boot Security

<table>
<thead>
<tr>
<th>Output #</th>
<th>Native</th>
<th>Xen</th>
<th>VMware</th>
<th>EC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>129</td>
<td>129</td>
<td>784</td>
<td>134</td>
</tr>
<tr>
<td>15</td>
<td>129</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
</tr>
</tbody>
</table>

No inputs before first output: constant value

Search space ($\log_2$) required of adversary for outputs of Linux /dev/(u)random during boot
Our Contributions

• First study of system RNGs in modern virtualized settings

• Snapshots cause problems? ➞ YES

• Entropy-poor inputs? ➞ NO

• New clean-slate RNG design — Whirlwind
Our Contributions

• First study of system RNGs in modern virtualized settings

• Snapshots cause problems? → YES

• Entropy-poor inputs? → NO

• New clean-slate RNG design — Whirlwind
Same interfaces - new design

if (entropy estimate >= 64)

if (count > 64 or elapsed time > 1s)

Interrupt Pool

Input Pool

Random Pool

URandom Pool

/dev/random

/dev/urandom

interrupt events

Interrupt events

disk events

keyboard events

mouse events

hardware rngs

if (entropy estimate >= 192)

interrupt events

interrupt events

disk events

keyboard events

mouse events

hardware rngs

/dev/random

/dev/urandom
Whirlwind RNG Design

Seed\textsubscript{fast}, Seed\textsubscript{slow}, CTR
PID, cycle counter, hw rng

interrupts, disk, keyboard, mouse

Seed\textsubscript{fast}
Seed\textsubscript{slow}
Seed\textsubscript{slow'}

/dev/random
/dev/urandom

Online hashing input process [DPRVW13]
Two-state structure (fast-slow) borrowed from Yarrow (FreeBSD) [SF99]

Seed\textsubscript{fast} ensures secure state change \textit{rapidly}
Seed\textsubscript{slow} prevents checkpoint (aka tracking) attacks [SF99]
Whirlwind RNG Design

interrupts, disk, keyboard, mouse

$Seed_{fast}$

$Seed'_{slow}$

$Seed_{slow}$

$Seed_{fast}$, $Seed_{slow}$, CTR
PID, cycle counter, hw rng

/dev/random
/dev/urandom

On boot: securely initialize RNG with large number of inputs

On snapshot resumption: inject randomness from Xen hypervisor
Whirlwind RNG

1. Drop-in replacement for legacy Linux RNG
2. Simple
3. Cryptographically sound
4. First output is not predictable
5. Reset security *by design*
Conclusions

• Linux, FreeBSD, and Windows are **vulnerable** on snapshot resumption

• First output of Linux /dev/(u)random on boot is a constant value

• Virtual settings have **sufficient** entropy

• Whirlwind RNG gives **reset security** by design

More information:  
http://pages.cs.wisc.edu/~ace/