



Netherlands Forensic Institute  
*Ministry of Justice*

# Pre-boot RAM acquisition and compression

## Martijn Bogaard

Student of Master in  
System and Network Engineering

University of Amsterdam

02 July 2015



# Why memory forensics?

- What was the user doing?
- What applications were running?
- Is the system infected with malware?



# Why memory forensics?

- What was the user doing?
- What applications were running?
- Is the system infected with malware?
  
- **Cryptokeys!**



## (Cold) boot attack

- Demonstrated in 2008 by Halderman *et al.*
  - “Lest We Remember: Cold Boot Attacks on Encryption Keys”



## (Cold) boot attack

- msramdmp
- {bios, efi}\_memimage
- Boot minimal OS?



## Open challenges

- What if we want to acquire evidence from:
  - Many systems?
  - Both memory **and** disk?
  - Over the network?
  - Systems with 4G+ RAM?



## Related work

- Bootable Linux CD / PXE for remote acquisition of multiple computers. (Cortjens 2014)
- Remote data acquisition on block devices in large environments. (van den Haak 2014)



# Research question

“Is pre-boot compression a useful technique to reduce the destruction of data when an operating system is loaded?”





# Goals

- Overwrite as little as possible
- Support >4G
- PXE & USB
- In a reasonable timeframe



## Proposed solution

- Compress RAM content before starting OS
- Start Linux based OS
- Extract compressed data from RAM



# Steps

- Analysis of RAM content (Shannon Entropy)
- Selection of data compression algorithm
- Development of acquisition algorithm
- Development of Proof of Concept



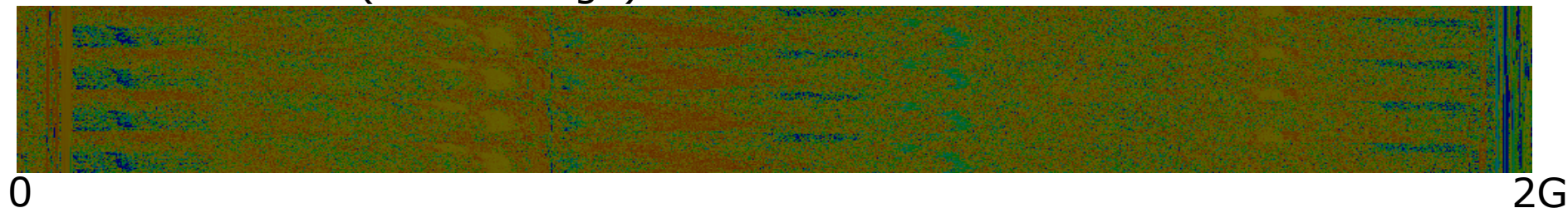
# RAM entropy

- 12 Dump from VMs
  - 256 MiB – 8 GiB
  - Windows & Linux
  - Several roles (desktop, server, live CD)
- Shannon Entropy ( $H$ )
  - In bits / byte of data
- Measured over whole RAM and in blocks
  - 4 & 16 kilobyte
- Average  $H$  5.36 ( $\sigma$  1.46)

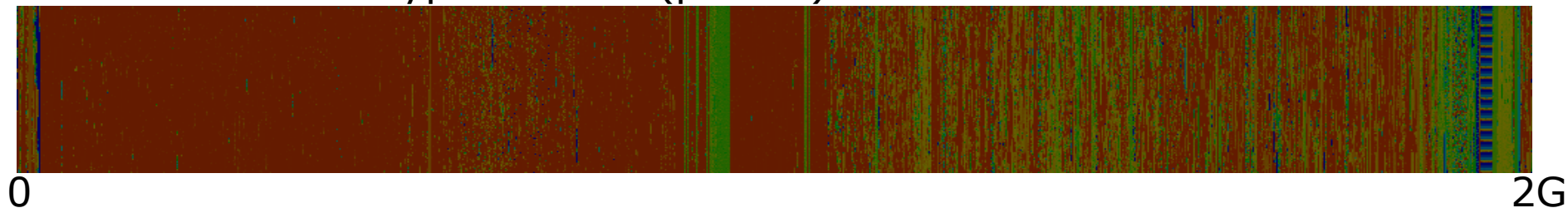


# RAM entropy

Windows 7 x86 (Office usage)



Tails 1.4 with encrypted folder (photos)





# Data compression algorithms

- Tested 13 algorithms
  - Some with multiple presets
  - 19 tests in total
- Focused on memory usage
  - Every byte used is written over original data
  - Measured using Valgrind with Massif
- But also duration, compression factor, theoretical worst-case scenario...
  - Tested against the RAM dumps of prev. step



# Data compression algorithms

- Selected LZW for Proof of Concept
  - 3,6 seconds / GiB (compression)
  - 60% avg. space saved
  - 7.7k mem usage (4.5k code, 3.2k stack, 0 heap)
  - Worst case output up to 104% of input length



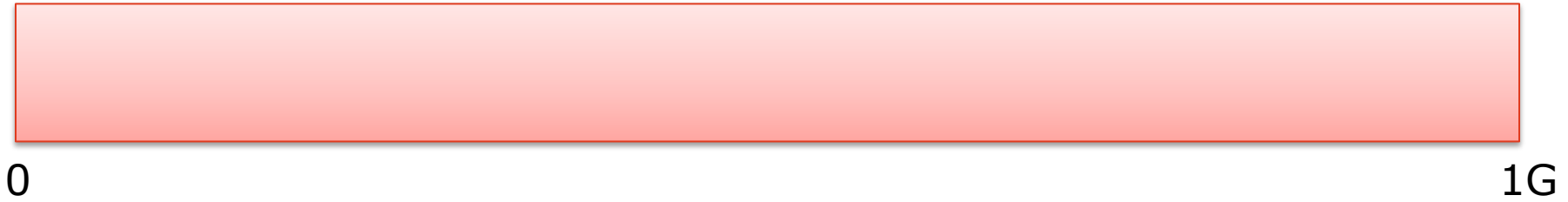
# Acquisition algorithm

- Work in non-contiguous address space
- Don't destroy more than absolutely necessary
- Make enough space to boot OS
- Protect compressed data from OS
- Provable forensic integrity



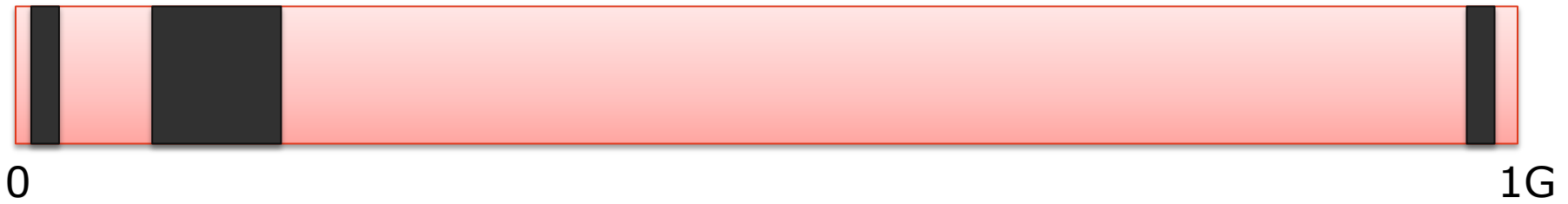


# Acquisition algorithm



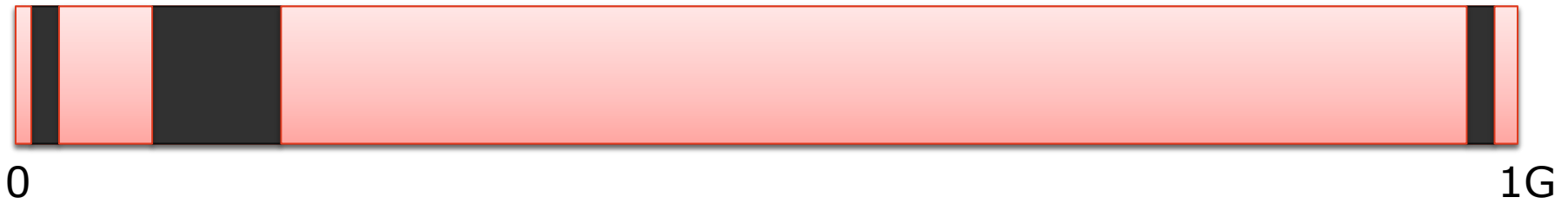


# Acquisition algorithm





# Acquisition algorithm



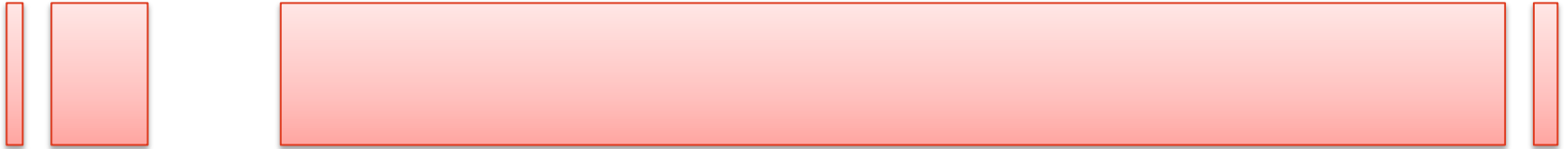


# Acquisition algorithm



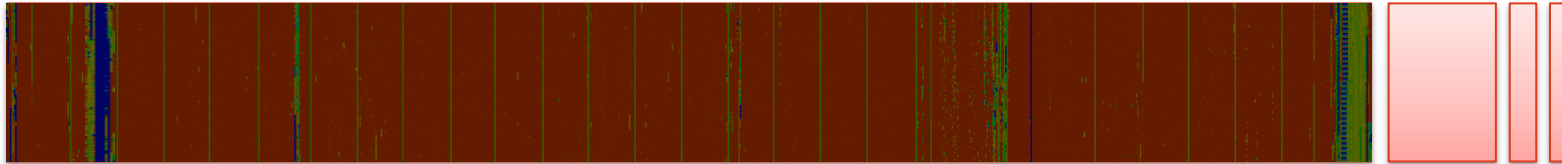


# Acquisition algorithm



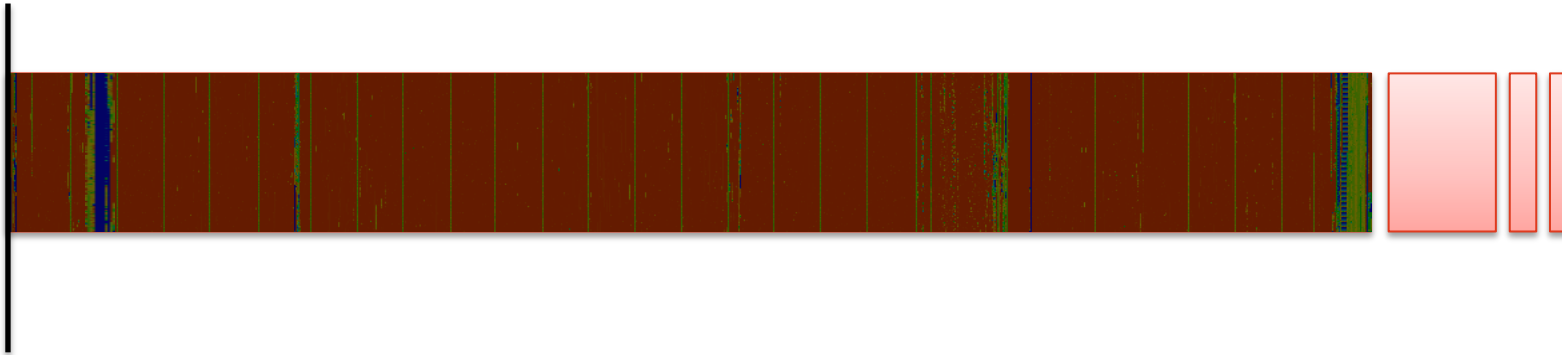


# Acquisition algorithm



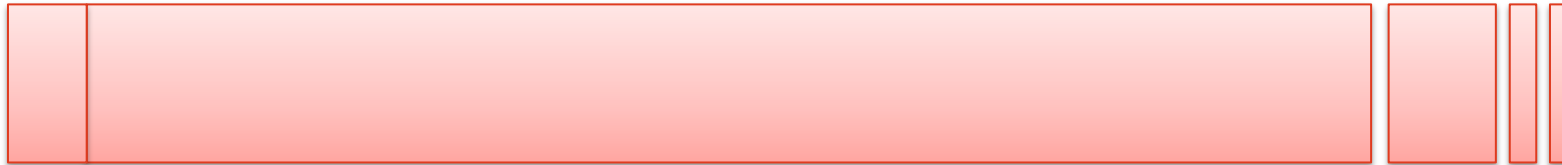


# Acquisition algorithm





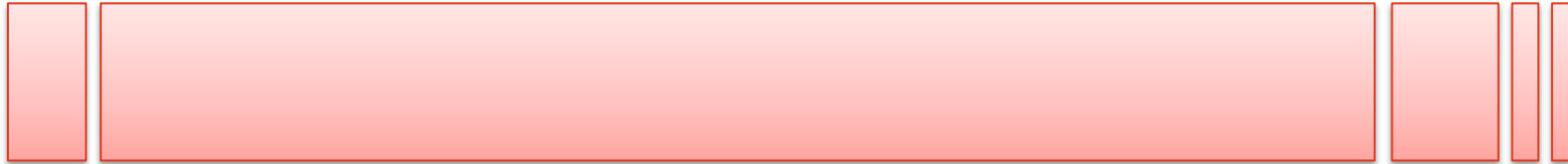
# Acquisition algorithm







# Acquisition algorithm





# Acquisition algorithm



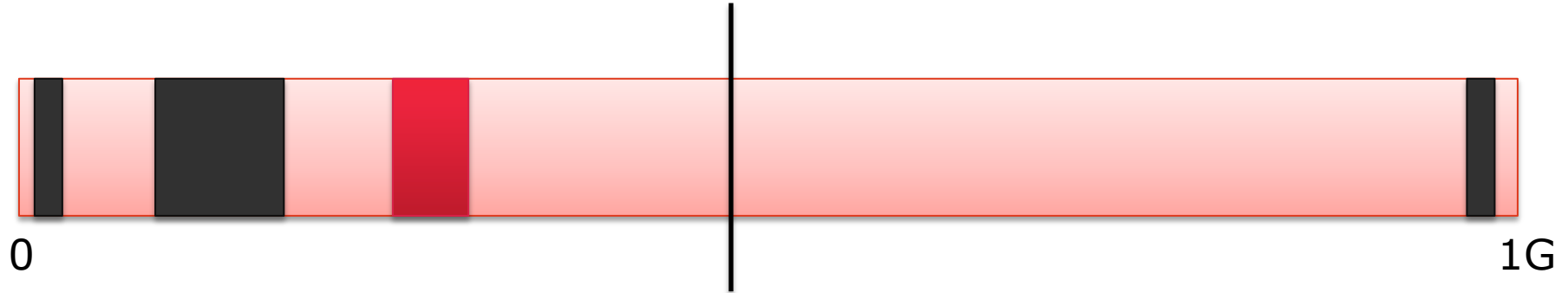


# Acquisition algorithm





# Acquisition algorithm



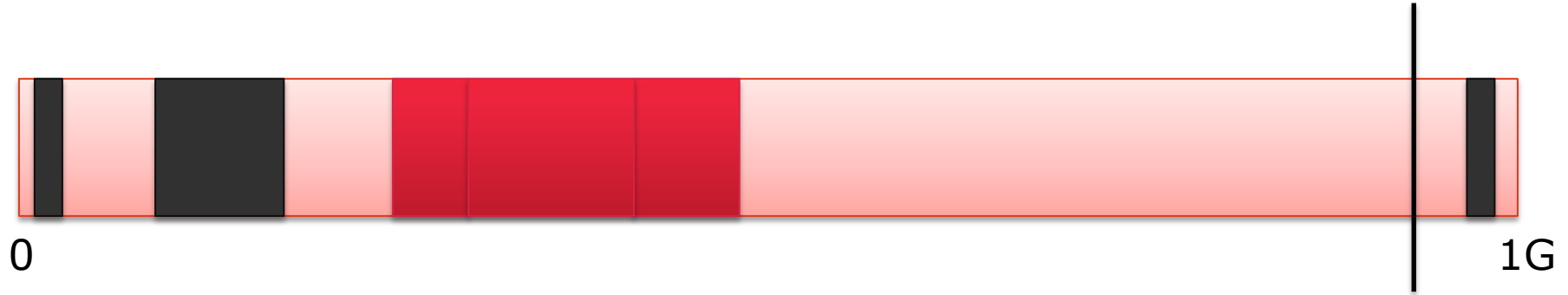


# Acquisition algorithm



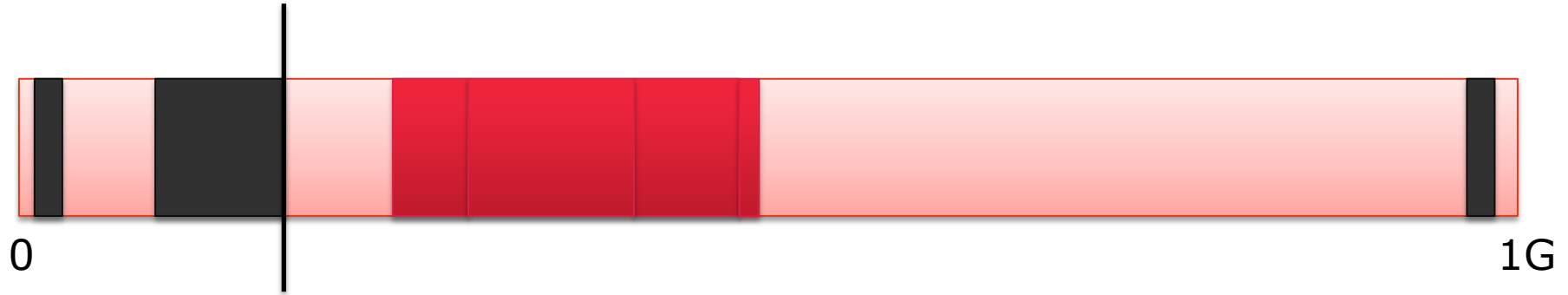


# Acquisition algorithm



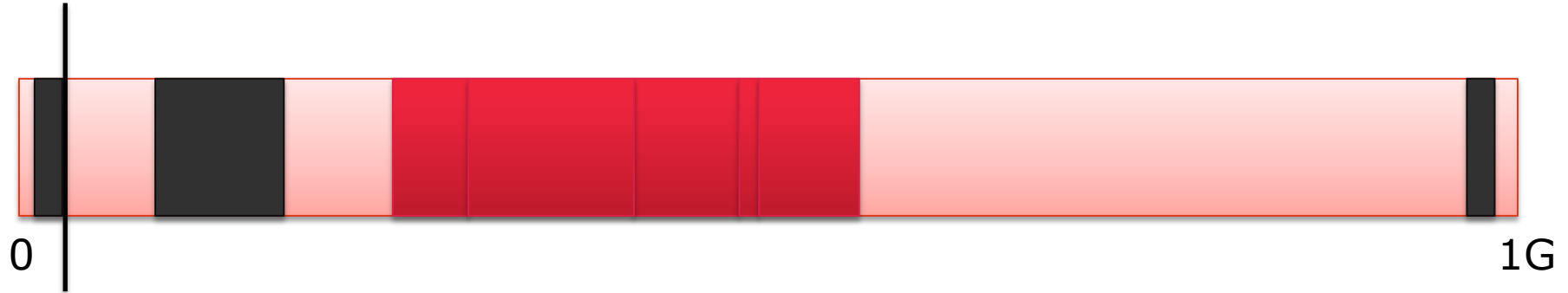


# Acquisition algorithm





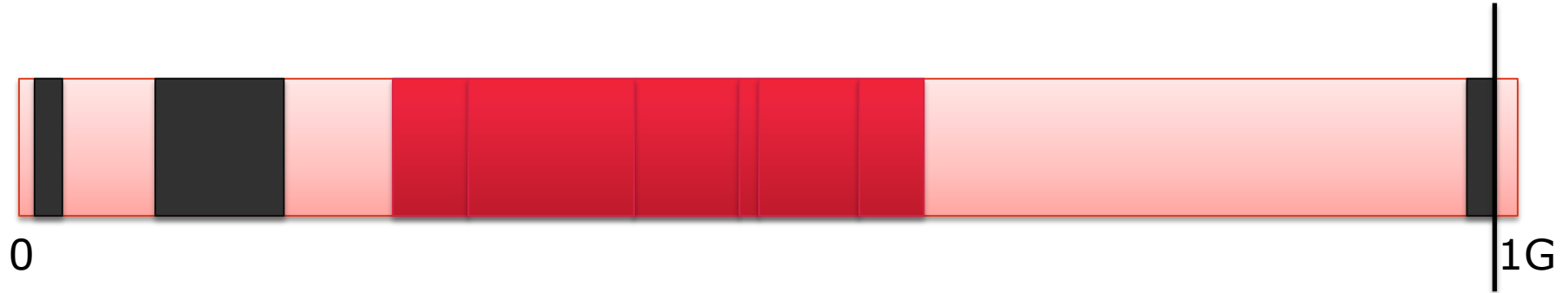
# Acquisition algorithm





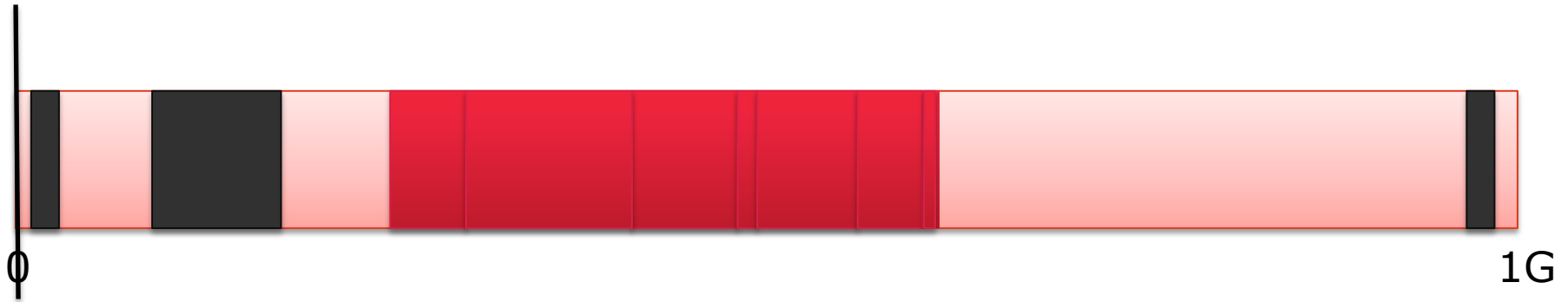


# Acquisition algorithm





# Acquisition algorithm





# Acquisition algorithm





# Proof of Concept

- Syslinux module to compress RAM
  - Supports CD, USB stick, PXE on BIOS & UEFI
  - Compress 100 MiB / block
  - SHA256 checksum over input
  - Modifies firmware memory map to hide compressed data
- OpenWRT based OS
  - Very small & low memory footprint
  - PXE boot needs 82 MiB free memory incl. ram disk
- Python script to extract compressed data
  - Patched /dev/mem interface



# Proof of Concept

- Tested with USB & PXE
- Store compressed data to NFS volume
- Decompress on different machine
  - In worst-case  $\sim 20$  MiB free memory available
- Modified QEMU to fill memory with pattern



# Proof of Concept

```
$ ./decompress.py dumps/03a78c78-dd57-436f-b81e-5e66d8e3dc49
```

...

Memory map:

```
[          0] - [          9F7FF] OK
[         9F800] - [          FFFFF] MISSING
[        100000] - [        FFFFFFF] OK
[       1000000] - [       73FFFFFF] Checksum INVALID!
[       7400000] - [       D7FFFFFF] OK
[       D800000] - [      13BFFFFFF] OK
[      13C00000] - [      19FFFFFFF] OK
[      1A000000] - [      1FEEFFFFFF] OK
[      1FEF0000] - [      1FEFEFFFF] OK
[      1FEFF000] - [      1FEFFFFFFF] OK
```



# Comparison with existing solutions

<b>Method</b>	<b>Recovered</b>		<b>Not recoverable</b>	
msramdmp	1022.8 M	99.883%	1.2 M	0.117%
bios_memimage	1022.7 M	99.872%	1.3 M	0.128%
Proof of Concept	1019.5 M	99.556%	4.5 M	0.444%
OpenWRT (ref)	878.0 M	85.700%	146.0 M	14.3%



# Conclusion

- Concept works
- Slightly increased memory usage
  - But can also be used for other evidence gathering
  - Mostly accountable to Syslinux





## Future work

- Test with UEFI based systems
- Modify Syslinux
  - 64-bit or PAE support
  - Lower memory usage?
- Test more scenario's with low amount of RAM
- More samples to predict likelihood of success



# QUESTIONS?

Martijn Bogaard  
martijn.bogaard@os3.nl