PANDA SECURITY | One step ahead.

Graph, Entropy and Grid Computing:

Automatic Comparison of Malware

Ismael Briones

Aitor Gomez

PandaLabs Malware Researcher

Virus Bulletin 2008, Ottawa

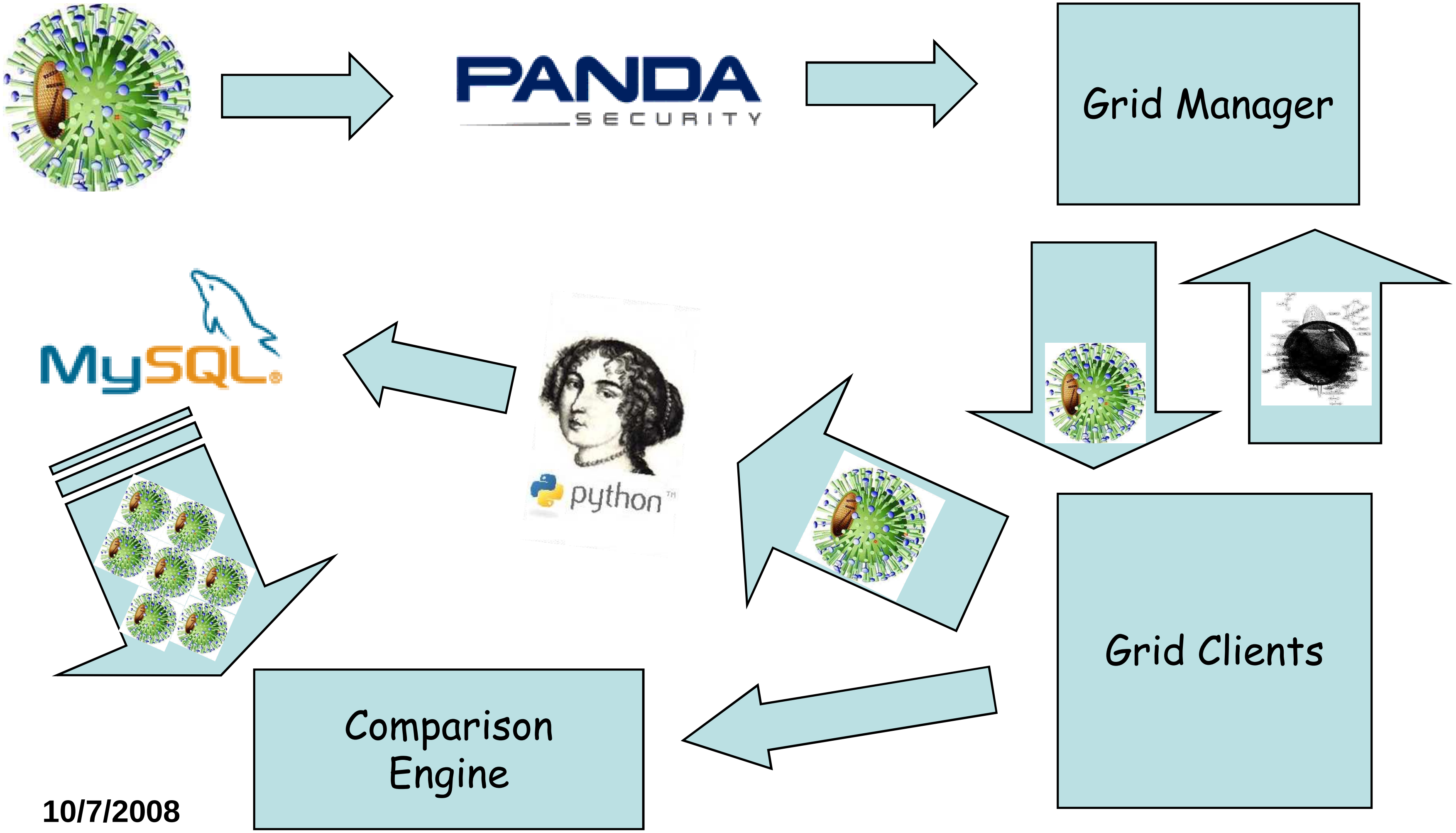
Challenges

- Identify new samples
- Automatically
- ASAP
- Improve detection rate
- Malware nomenclature

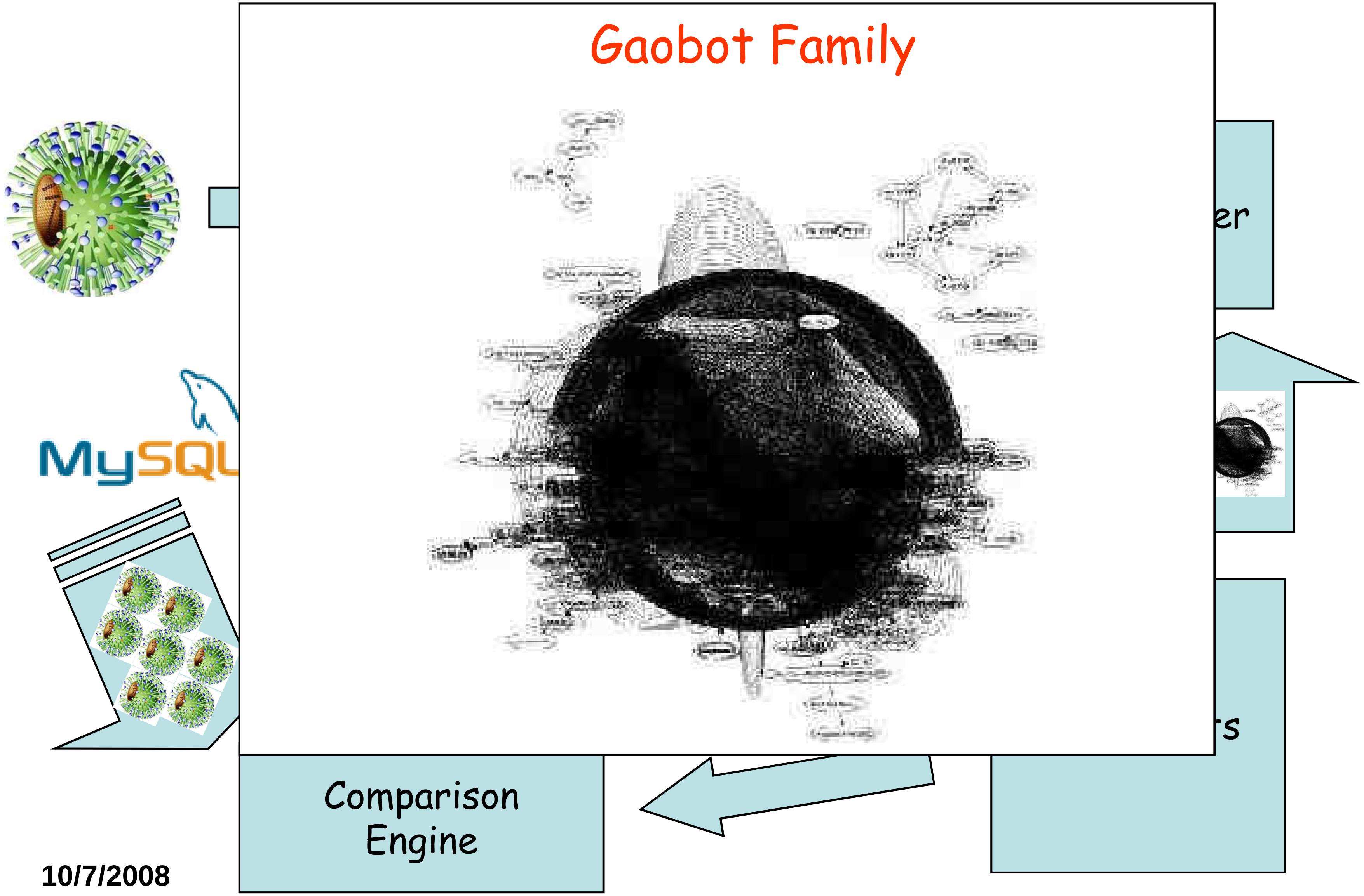
State of the Art

- Ero Carrera & Gergely Erdélyi
  - Digital Genome Mapping
- Halvar Flake
  - Lot of researches in graphs analysis
  - VxClass
- Marius Gheorghescu
  - An Automated Virus Classification system

The System



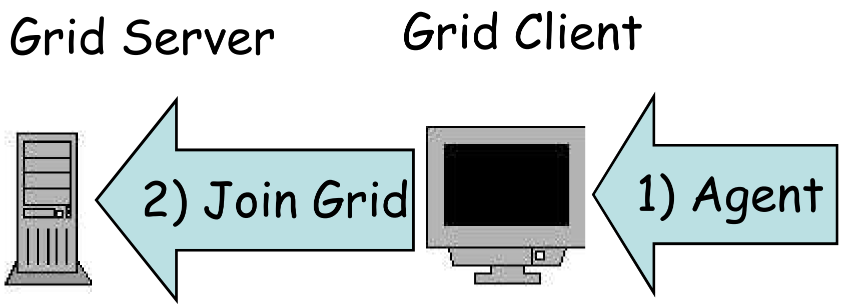
Automatic Comparison of Malware

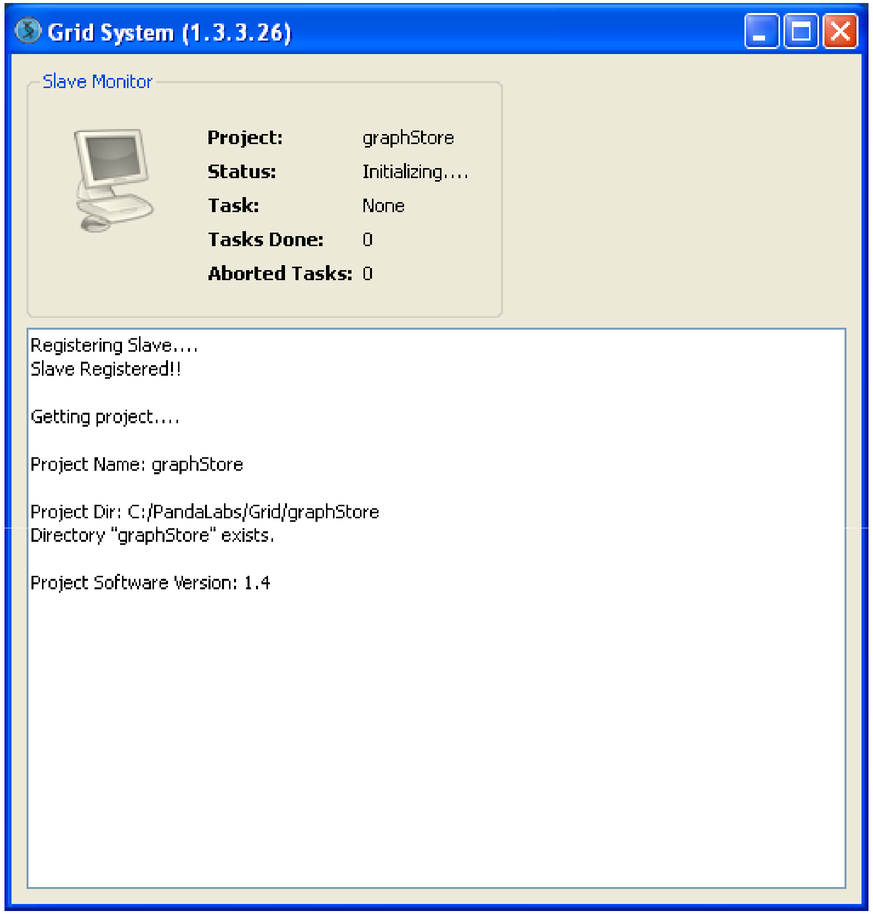


Grid System

- Analyse as many samples as possible
- IDA Analysis take too much time
- BOINC, GLOBUS, ARC (Advanced Resource Connector)
- XMLRPC

Automatic Comparison of Malware



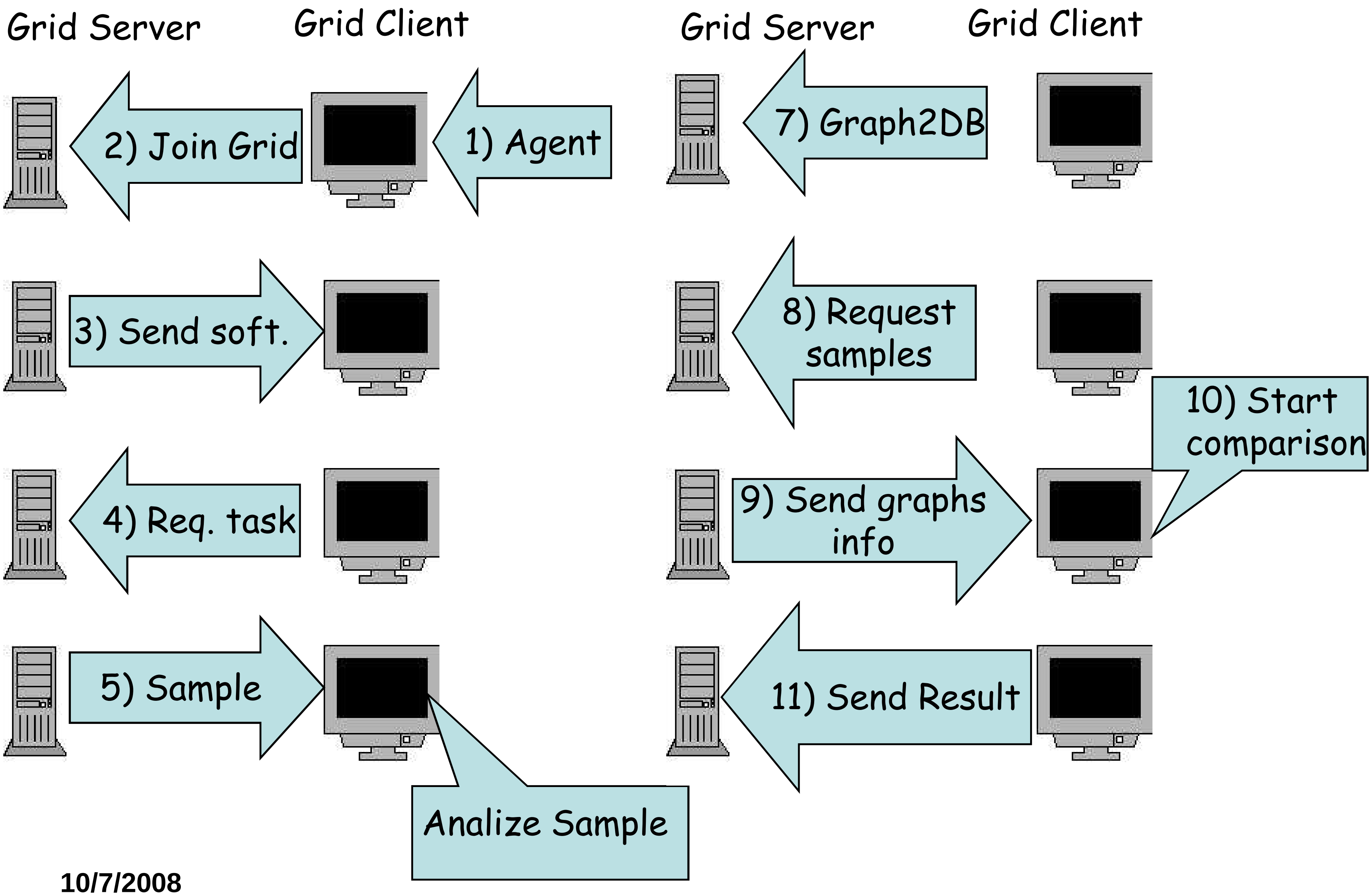


Project: graphStore
Status: Initializing....
Task: None
Tasks Done: 0
Aborted Tasks: 0

```
Registering Slave....
Slave Registered!!

Getting project....

Project Name: graphStore

Project Dir: C:/PandaLabs/Grid/graphStore
Directory "graphStore" exists.

Project Software Version: 1.4
```

10/7/2008

Automatic Comparison of Malware


Grid Server

Grid Client

2) Join Grid

1) Agent

3) Send soft.

4) Req. task

5) Sample

Analize Sample

Grid Server

Grid Client

7) Graph2DB

8) Request samples

10) Start comparison

9) Send graphs info

11) Send Result

10/7/2008

Sample Analysis

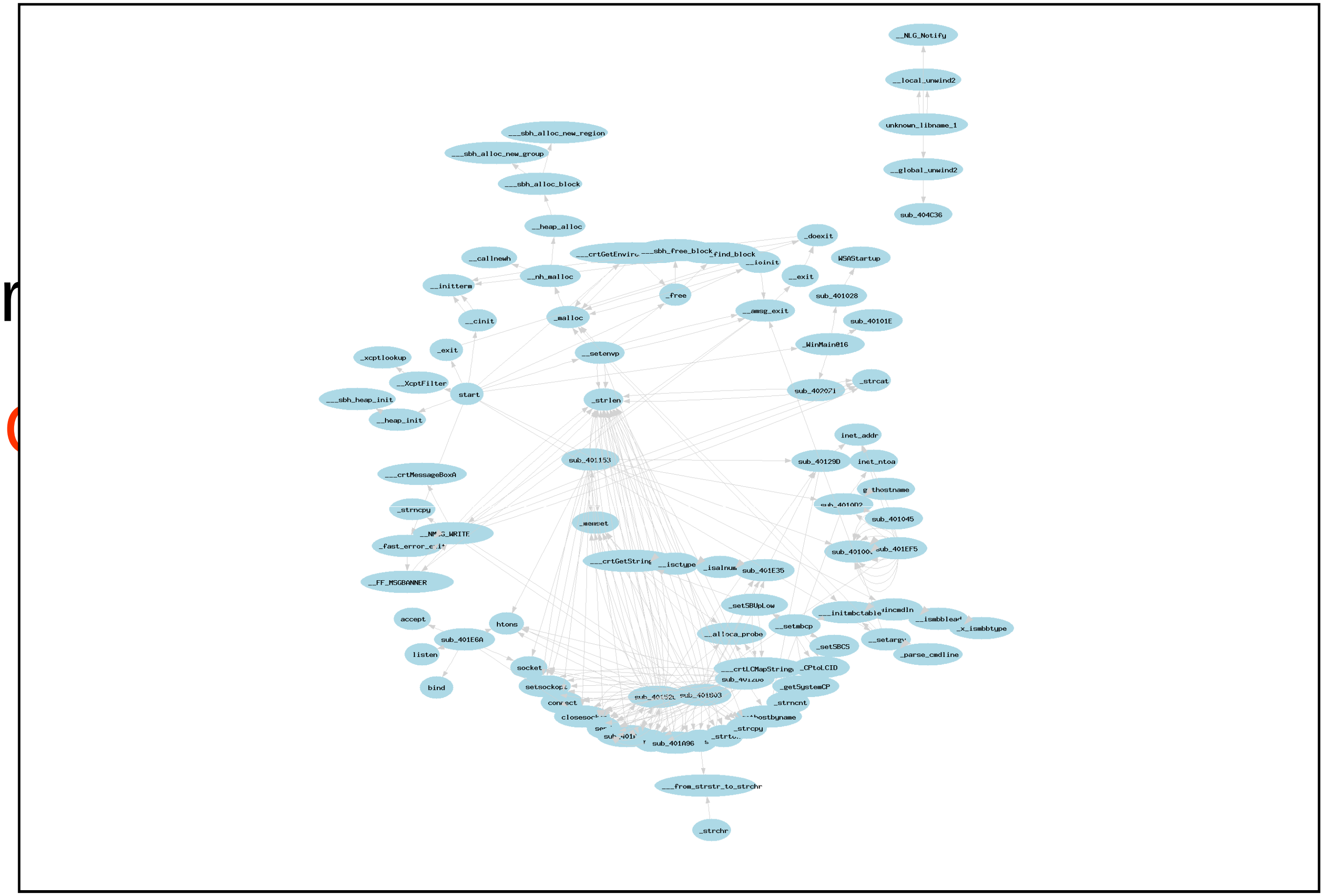
- Ida Pro + IdaPython

Sample Analysis

- Ida Pro + IdaPython
- Flow Graph

Automatic Comparison of Malware

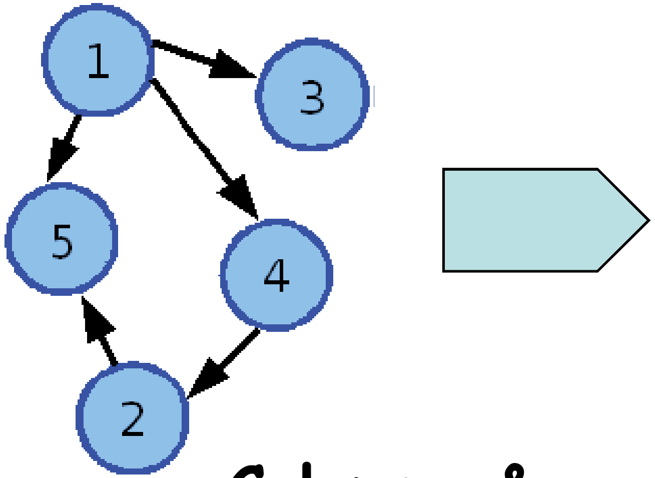
- Ida Pr
- Flow (



10/7/2008

- Ida Pr
- Flow (

Adjacency Matrix



$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 0 & 1 & 1 & 1 \\ 2 & 0 & 0 & 0 & 0 & 1 \\ 3 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 1 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 \end{array}$$

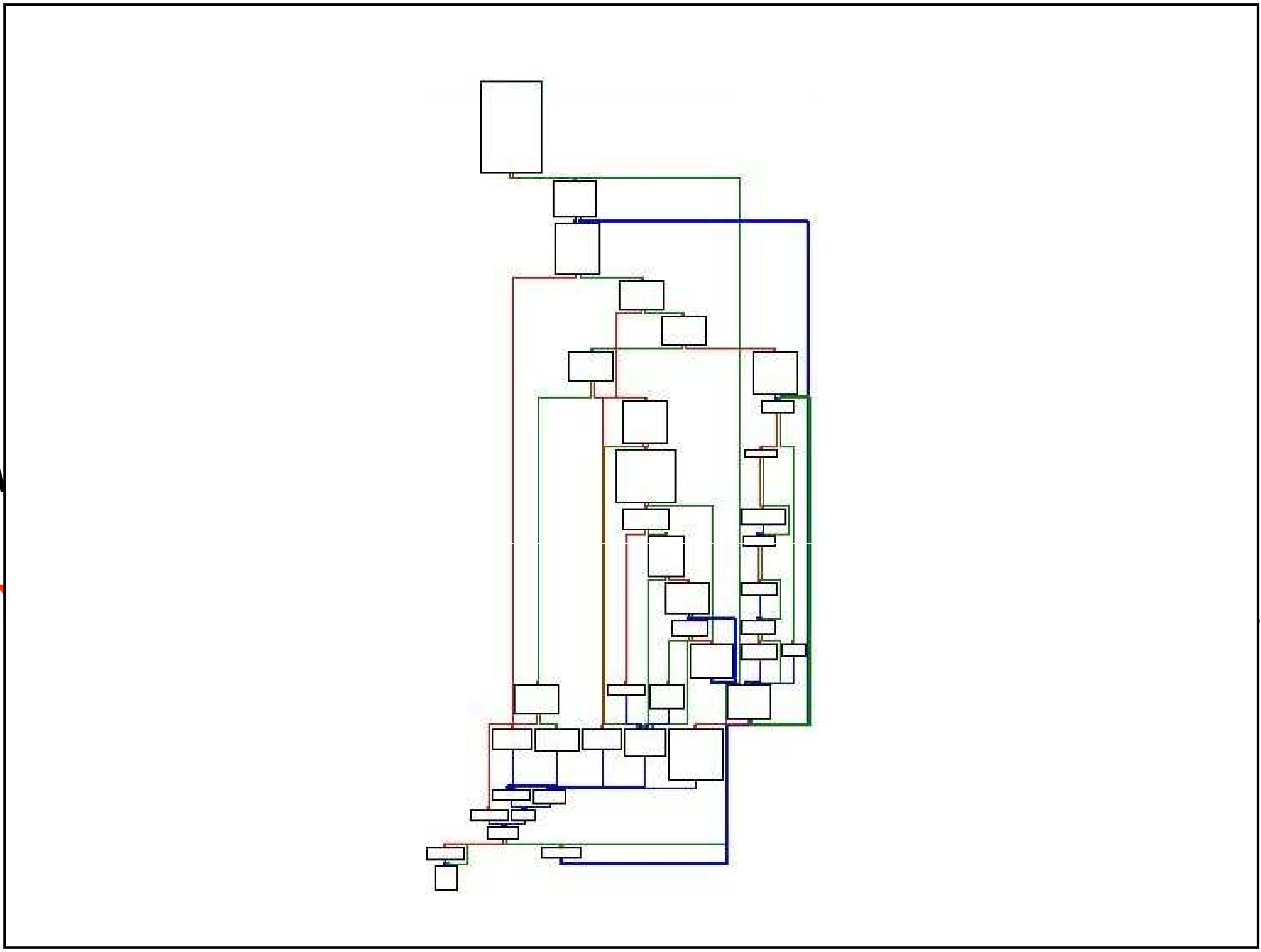
Columns & rows = graph nodes

(i,j) = 1, if i calls j

Sample Analysis

- Ida Pro + IdaPython
- Flow Graph
- Functions Control Flow Graph (CFG) signature

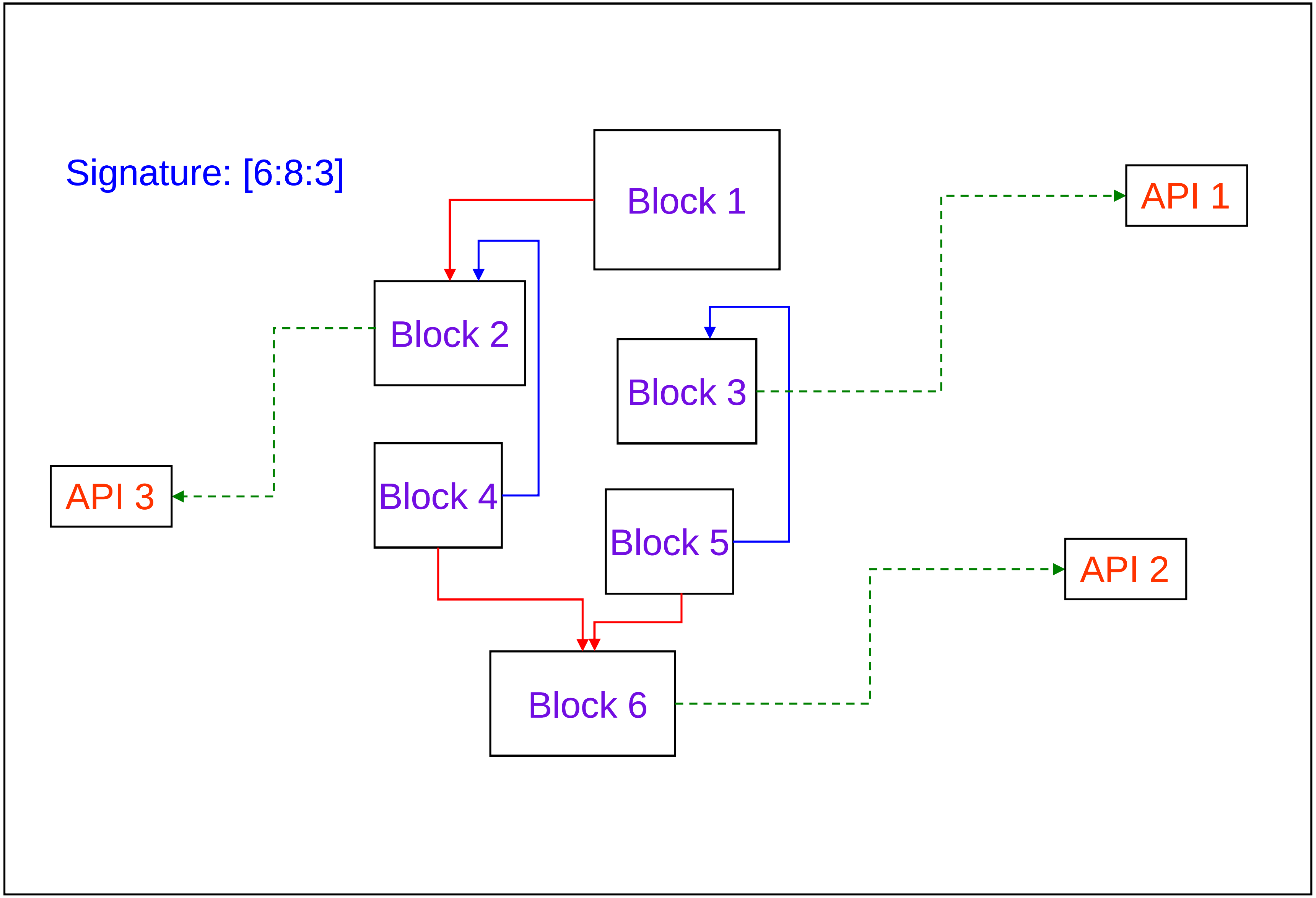
Automatic Comparison of Malware



- Ida
- Flov
- Fur ture

10/7/2008

Automatic Comparison of Malware



Signature: [6:8:3]

Block 1

Block 2

Block 3

Block 4

Block 5

Block 6

API 1

API 2

API 3

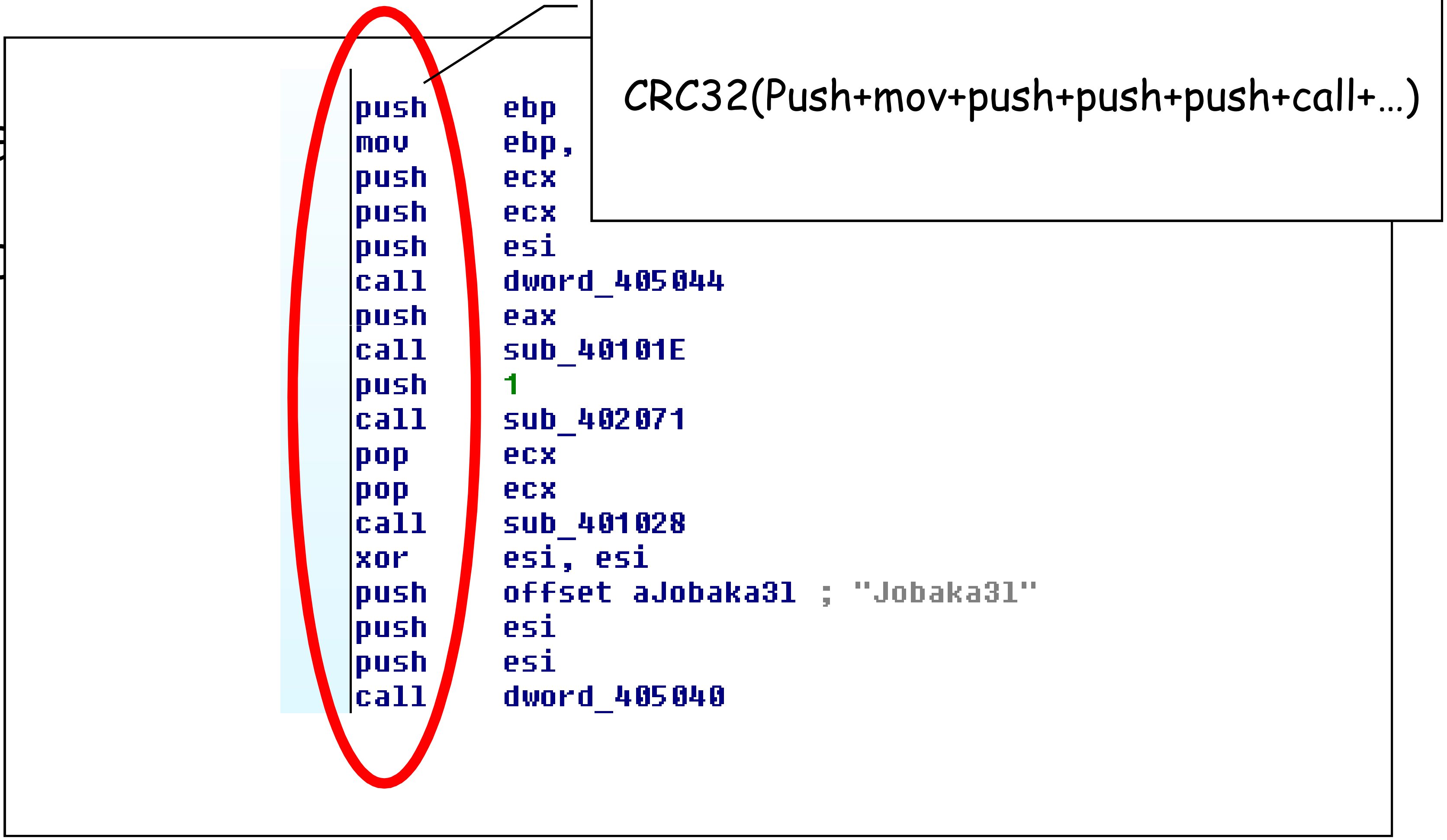
10/7/2008

Sample Analysis

- Ida Pro + IdaPython
- Flow Graph
- Functions Control Flow Graph (CFG)
- Function’s crc32

Sample Analysis

- ➢Ida
- ➢Flc
- ➢Fu
- ➢Fu



```
push    ebp
mov     ebp,
push    ecx
push    ecx
push    esi
call    dword_405044
push    eax
call    sub_40101E
push    1
call    sub_402071
pop     ecx
pop     ecx
call    sub_401028
xor     esi, esi
push    offset aJobaka31  ; "Jobaka31"
push    esi
push    esi
call    dword_405040
```

CRC32(Push+mov+push+push+push+call+…)

Sample Analysis

- Ida Pro + IdaPython
- Flow Graph
- Functions Control Flow Graph (CFG)
- Function's crc32
- Functions names (sub_, nullsub_,….)
- Operating Systems and Library Calls (API's)

Comparison Algorithm

- **Select best samples**

- Compiler type (Peid Signature)
- File Size
- Number Api Functions
- Number custom functions
- Checksum & Entropy

Checksum

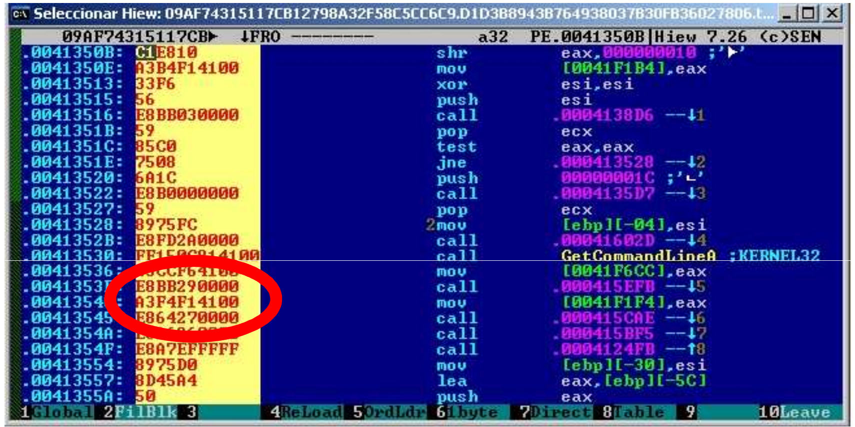
‘A checksum is a form of redundancy check, […] It works by adding up the basic components of a message, typically the assorted bits [in our case each byte]*, and storing the resulting value.’*

[from wikipedia]

$$(AA)+(BB)+(CC)+(DD)\rightarrow Checksum=\mathrm{0x30E}$$

Checksum Properties

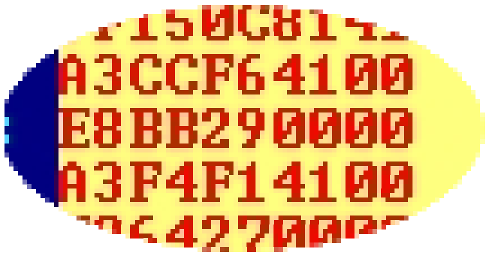
- Similar blocks has very close checksum values



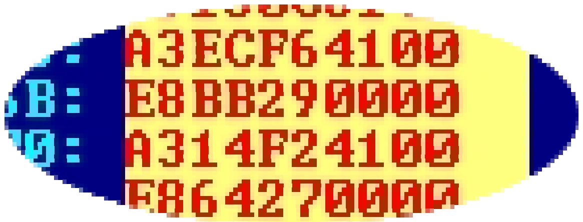
Seleccionar Hiew: 09AF74315117CB12798A32F58C5CC6C9.D1D3B8943B764938037B30FB36027806.t...

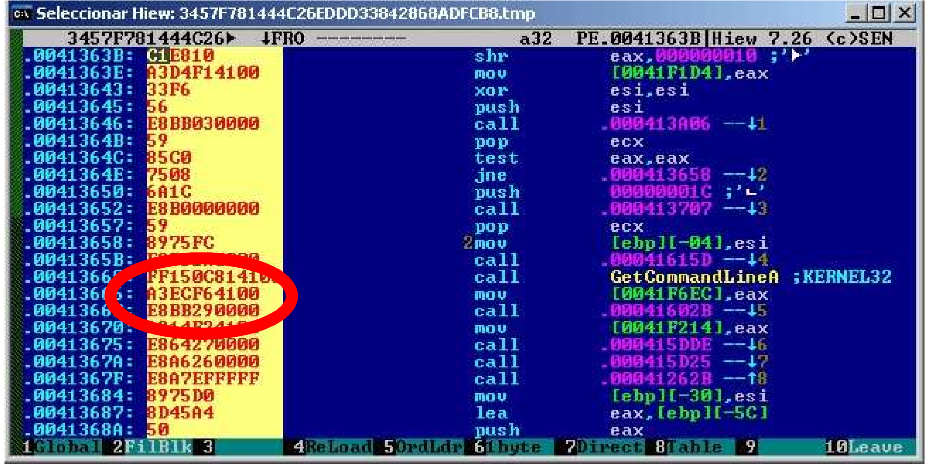
```
   09AF74315117CB▸  ↓FRO ---------          a32  PE.0041350B|Hiew 7.26 (c)SEN
.0041350B: C1E810                 shr     eax,000000010 ;'▸'
.0041350E: A3B4F14100             mov     [0041F1B4],eax
.00413513: 33F6                   xor     esi,esi
.00413515: 56                     push    esi
.00413516: E8BB030000             call   .0004138D6 --↓1
.0041351B: 59                     pop     ecx
.0041351C: 85C0                   test    eax,eax
.0041351E: 7508                   jne    .000413528 --↓2
.00413520: 6A1C                   push    00000001C ;'∟'
.00413522: E8B0000000             call   .0004135D7 --↓3
.00413527: 59                     pop     ecx
.00413528: 8975FC                2mov     [ebp][-04],esi
.0041352B: E8FD2A0000             call   .00041602D --↓4
.00413530: FF150C814100           call    GetCommandLineA ;KERNEL32
.00413536: A3CCF64100             mov     [0041F6CC],eax
.0041353B: E8BB290000             call   .000415EFB --↓5
.00413540: A3F4F14100             mov     [0041F1F4],eax
.00413545: E864270000             call   .000415CAE --↓6
.0041354A: [illegible]            call   .000415BF5 --↓7
.0041354F: E8A7EFFFFF             call   .0004124FB --↑8
.00413554: 8975D0                 mov     [ebp][-30],esi
.00413557: 8D45A4                 lea     eax,[ebp][-5C]
.0041355A: 50                     push    eax
1Global 2FilBlk 3       4ReLoad 5OrdLdr 61byte  7Direct 8Table  9        10Leave
```

Checksum: 0x260A



Checksum: 0x276D



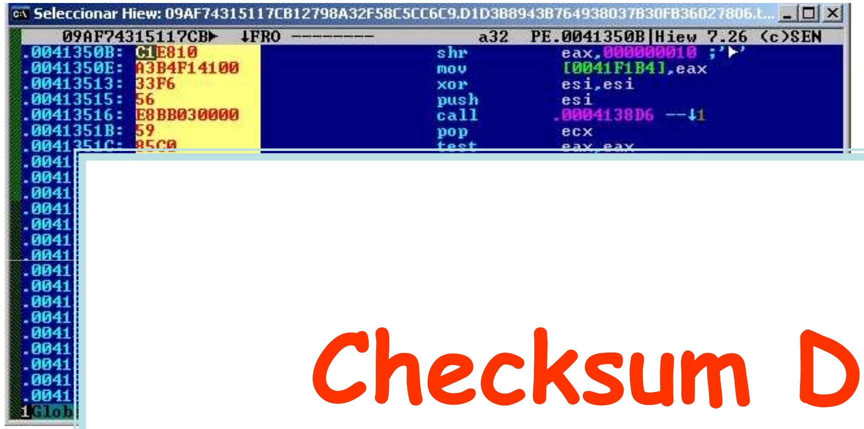


Seleccionar Hiew: 3457F781444C26EDDD33842868ADFCB8.tmp

```
   3457F781444C26▸  ↓FRO ---------          a32  PE.0041363B|Hiew 7.26 (c)SEN
.0041363B: C1E810                 shr     eax,000000010 ;'▸'
.0041363E: A3D4F14100             mov     [0041F1D4],eax
.00413643: 33F6                   xor     esi,esi
.00413645: 56                     push    esi
.00413646: E8BB030000             call   .000413A06 --↓1
.0041364B: 59                     pop     ecx
.0041364C: 85C0                   test    eax,eax
.0041364E: 7508                   jne    .000413658 --↓2
.00413650: 6A1C                   push    00000001C ;'∟'
.00413652: E8B0000000             call   .000413707 --↓3
.00413657: 59                     pop     ecx
.00413658: 8975FC                2mov     [ebp][-04],esi
.0041365B: [illegible]            call   .00041615D --↓4
.00413660: FF150C814100           call    GetCommandLineA ;KERNEL32
.00413666: A3ECF64100             mov     [0041F6EC],eax
.0041366B: E8BB290000             call   .00041602B --↓5
.00413670: A314F24100             mov     [0041F214],eax
.00413675: E864270000             call   .000415DDE --↓6
.0041367A: E8A6260000             call   .000415D25 --↓7
.0041367F: E8A7EFFFFF             call   .00041262B --↑8
.00413684: 8975D0                 mov     [ebp][-30],esi
.00413687: 8D45A4                 lea     eax,[ebp][-5C]
.0041368A: 50                     push    eax
1Global 2FilBlk 3       4ReLoad 5OrdLdr 61byte  7Direct 8Table  9        10Leave
```

Checksum Properties

- Similar blocks has very close checksum values

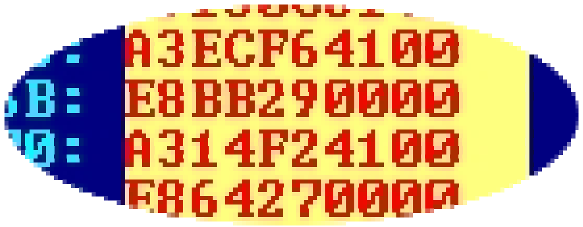


```
Seleccionar Hiew: 09AF74315117CB12798A32F58C5CC6C9.D1D3B8943B764938037B30FB36027806.t...
   09AF74315117CB►  ↓FRO ---------         a32  PE.0041350B|Hiew 7.26 (c)SEN
.0041350B: C1E810                      shr     eax,000000010 ;'►'
.0041350E: A3B4F14100                  mov     [0041F1B4],eax
.00413513: 33F6                        xor     esi,esi
.00413515: 56                          push    esi
.00413516: E8BB030000                  call    .0004138D6 --↓1
.0041351B: 59                          pop     ecx
.0041351C: 85C0                        test    eax,eax
.0041
.0041
.0041
.0041
.0041
.0041
.0041
.0041
.0041
.0041
.0041
.0041
.0041
.0041
.0041
.0041
1Glob
```

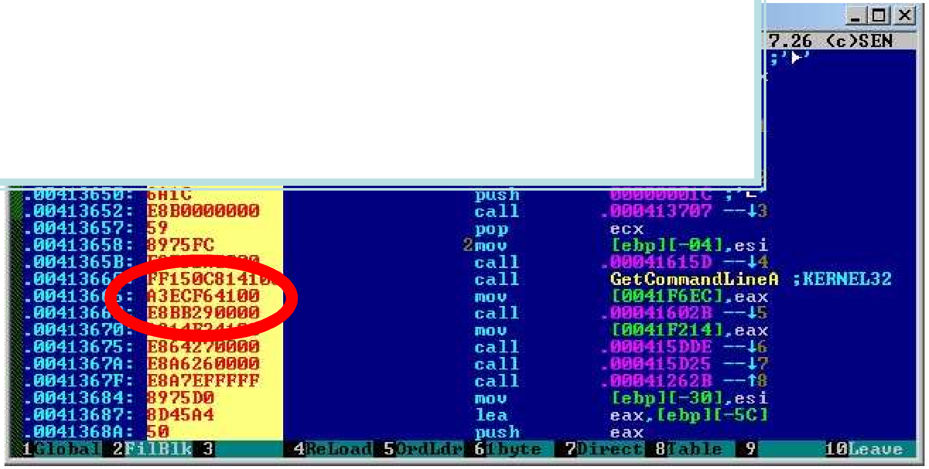
Checksum: 0x260A

Checksum Diff: 0x163

Che



```
B: A3ECF64100
0: E8BB290000
   A314F24100
   E864270000
```



```
7.26 (c)SEN
;'►'
.00413650: 6A1C                        push    00000001C ;'∟'
.00413652: E8B0000000                  call    .000413707 --↓3
.00413657: 59                          pop     ecx
.00413658: 8975FC                     2mov     [ebp][-04],esi
.0041365B:                             call    .00041615D --↓4
.0041366 : FF150C8141                  call    GetCommandLineA ;KERNEL32
.0041366 : A3ECF64100                  mov     [0041F6EC],eax
.0041366 : E8BB290000                  call    .00041602B --↓5
.00413670:                             mov     [0041F214],eax
.00413675: E864270000                  call    .000415DDE --↓6
.0041367A: E8A6260000                  call    .000415D25 --↓7
.0041367F: E8A7EFFFFF                  call    .00041262B --↑8
.00413684: 8975D0                      mov     [ebp][-30],esi
.00413687: 8D45A4                      lea     eax,[ebp][-5C]
.0041368A: 50                          push    eax
1Global 2FilBlk 3      4ReLoad 5OrdLdr 61byte 7Direct 8Table 9     10Leave
```

10/7/2008

Checksum disadvantages

- A checksum is not changed by:
  - Inserting or deleting zero-valued bytes

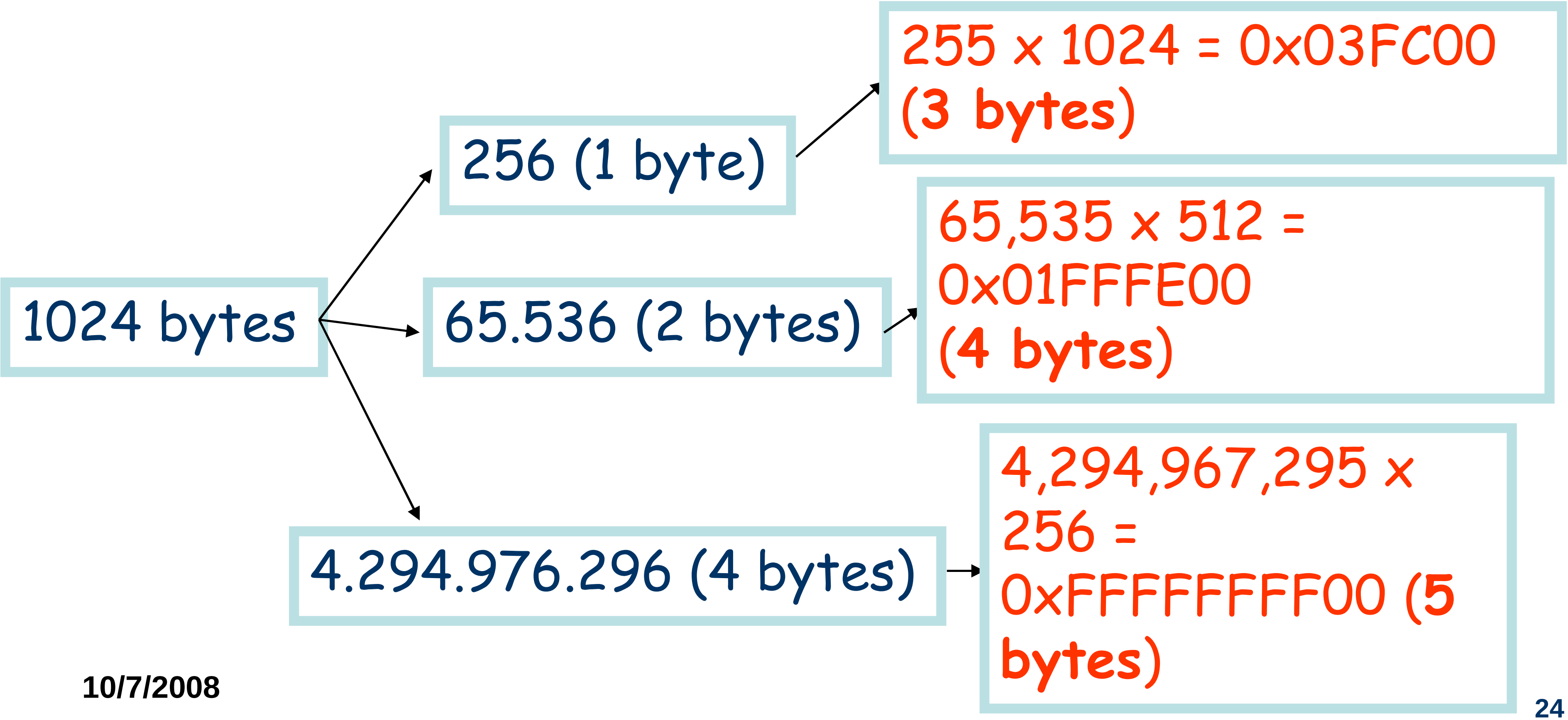
```
0xAA+0xBB = 0x165
0xAA+0x00+0xBB+0x00=0x165
```

  - Reordering of the bytes in the data.

```
0xAA+0xBB = 0x165
0xBB+0xAA=0x165
```

Automatic Comparison of Malware

- Checksum value depends on 2 factors:
  - Size of data block
  - Base (basic component: bits, bytes,…)



1024 bytes

256 (1 byte)

255 x 1024 = 0x03FC00 **(3 bytes)**

65.536 (2 bytes)

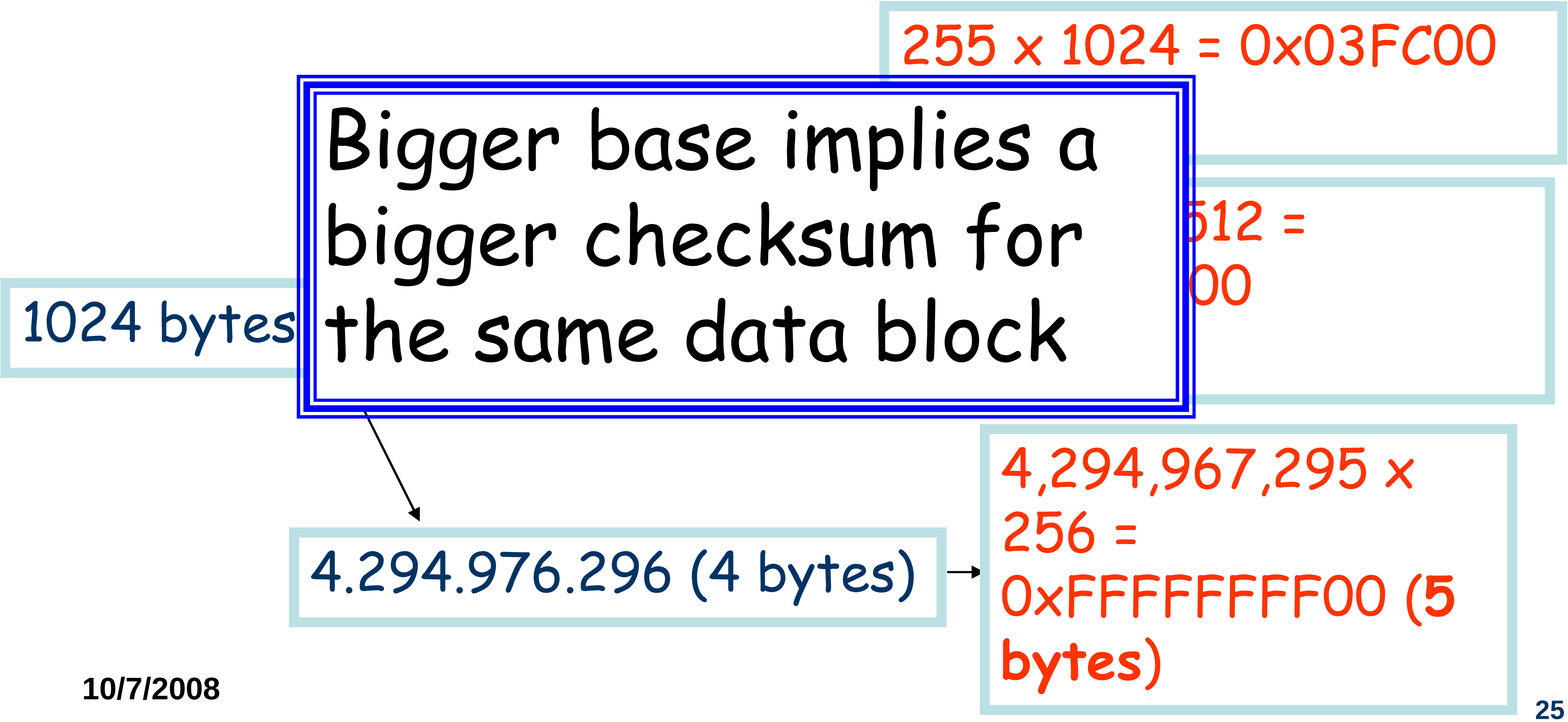
65,535 x 512 = 0x01FFFE00 **(4 bytes)**

4.294.976.296 (4 bytes)

4,294,967,295 x 256 = 0xFFFFFFFF00 (**5 bytes**)

Automatic Comparison of Malware

- Checksum value depends on 2 factors:
  - Size of data block
  - Base (basic component: bits, bytes,…)



255 x 1024 = 0x03FC00

512 =

00

1024 bytes

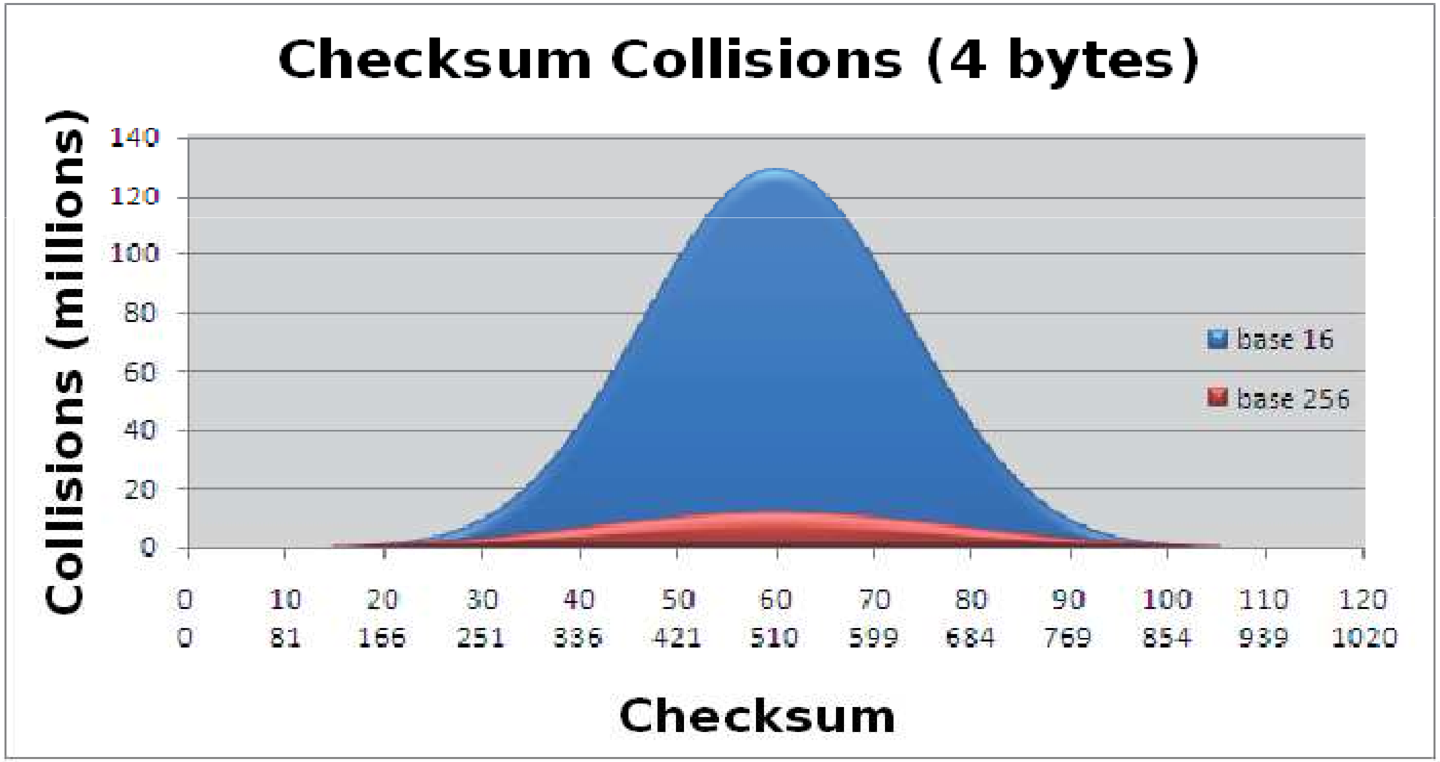
Bigger base implies a bigger checksum for the same data block

4.294.976.296 (4 bytes)

4,294,967,295 x 256 = 0xFFFFFFFF00 (**5 bytes**)

Bigger base involve a Bigger checksum

- Decreases the probability of collision (same checksum)



10/7/2008

Bigger base involve bigger checksum difference

- Similar data blocks could get significantly different checksum values

$$Base\,256(BYTE)$$

$$(AA)+(BB)+(CC)+(DD)\rightarrow Checksum=\text{0x30E}$$
$$(BA)+(BB)+(CC)+(DD)\rightarrow Checksum=\underbrace{\text{0x31E}}_{(diff=\text{0x10})}$$

$$Base\,65536(WORD)$$

$$(AABB)+(CCDD)\rightarrow Checksum=\text{0x017798}$$
$$(BABB)+(CCDD)\rightarrow Checksum=\underbrace{\text{0x018798}}_{(diff=\text{0x1000})}$$

Bigger base involve bigger checksum difference

- Similar data blocks could get significantly different checksum values

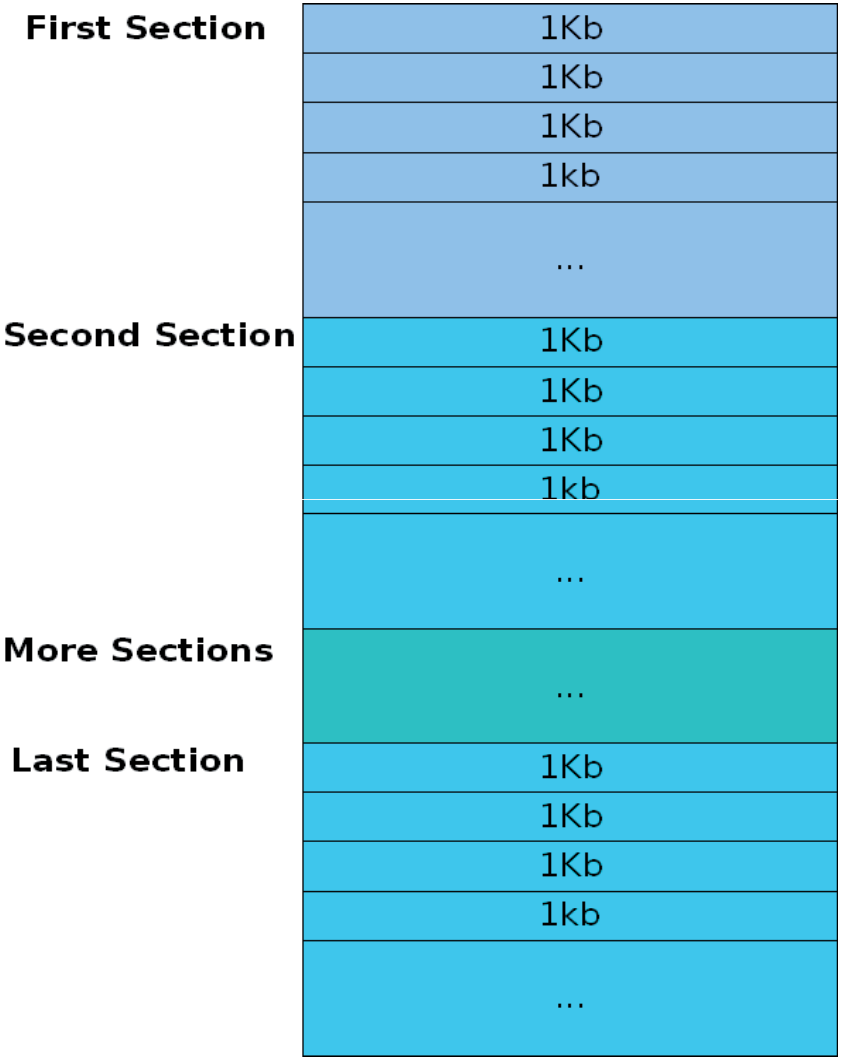
As code is represented as bytes, our approach will use a base of 256 (1byte)

$$(AABB)+(CCDD)\rightarrow Checksum=0x017798$$

$$(BABB)+(CCDD)\rightarrow Checksum=\underbrace{0x018798}_{(diff=0x1000)}$$

Our Checksum

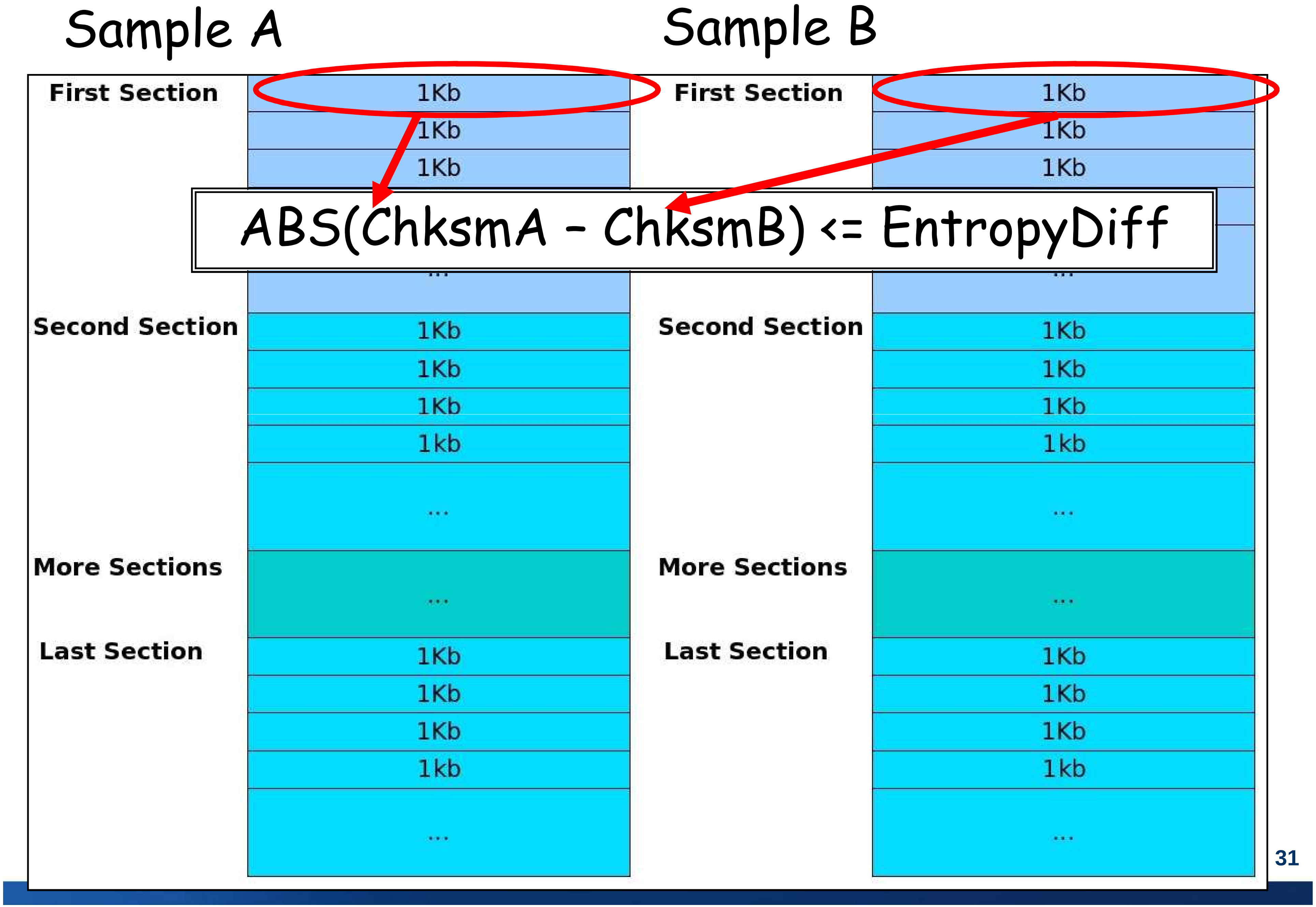
- 4KB, in blocks of 1KB, from the beginning of the 1st , 2nd and last sections



Checksum Algorithm

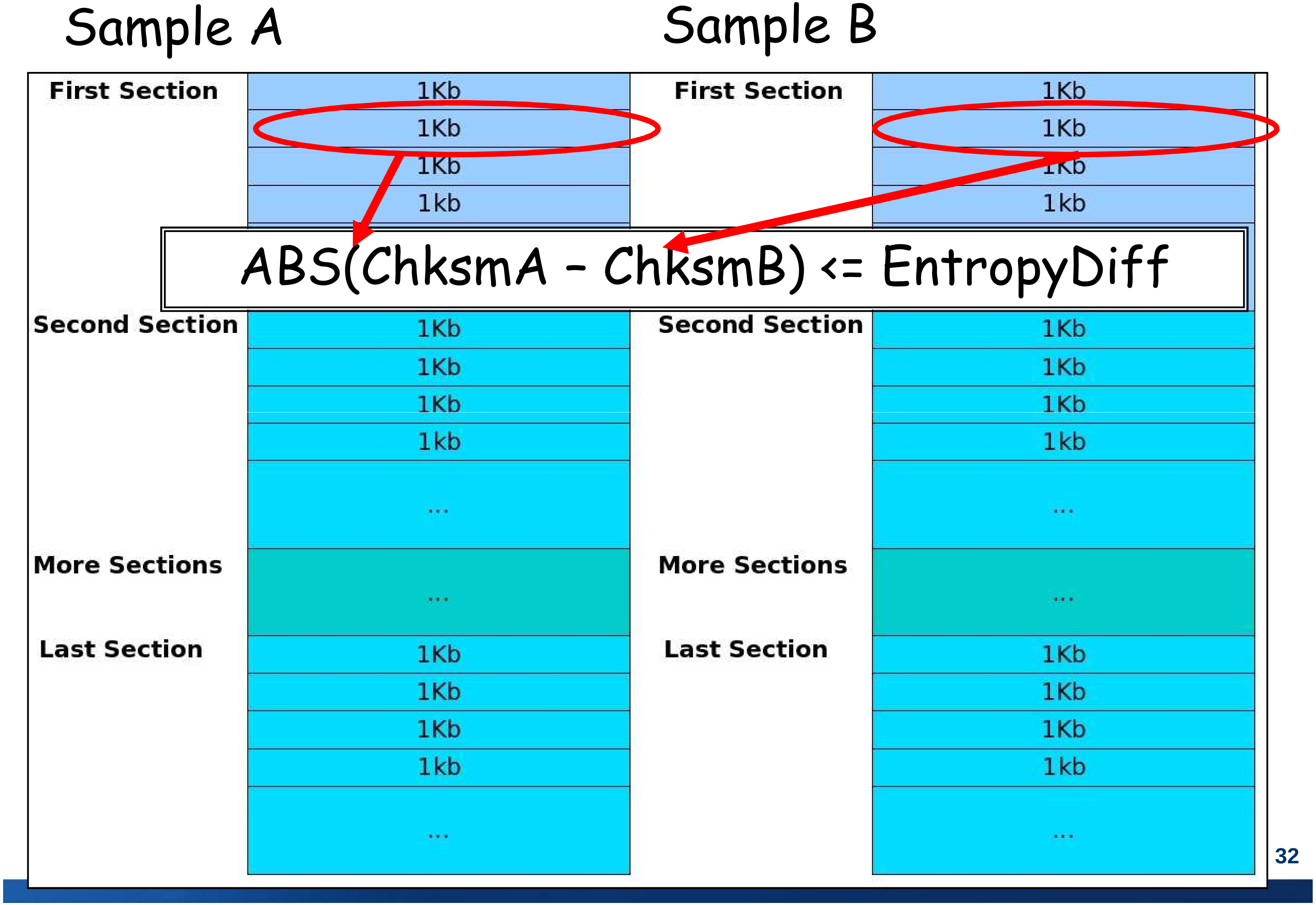
- Substract checksum from both files

Automatic Comparison of Malware



ABS(ChksmA – ChksmB) <= EntropyDiff

Automatic Comparison of Malware



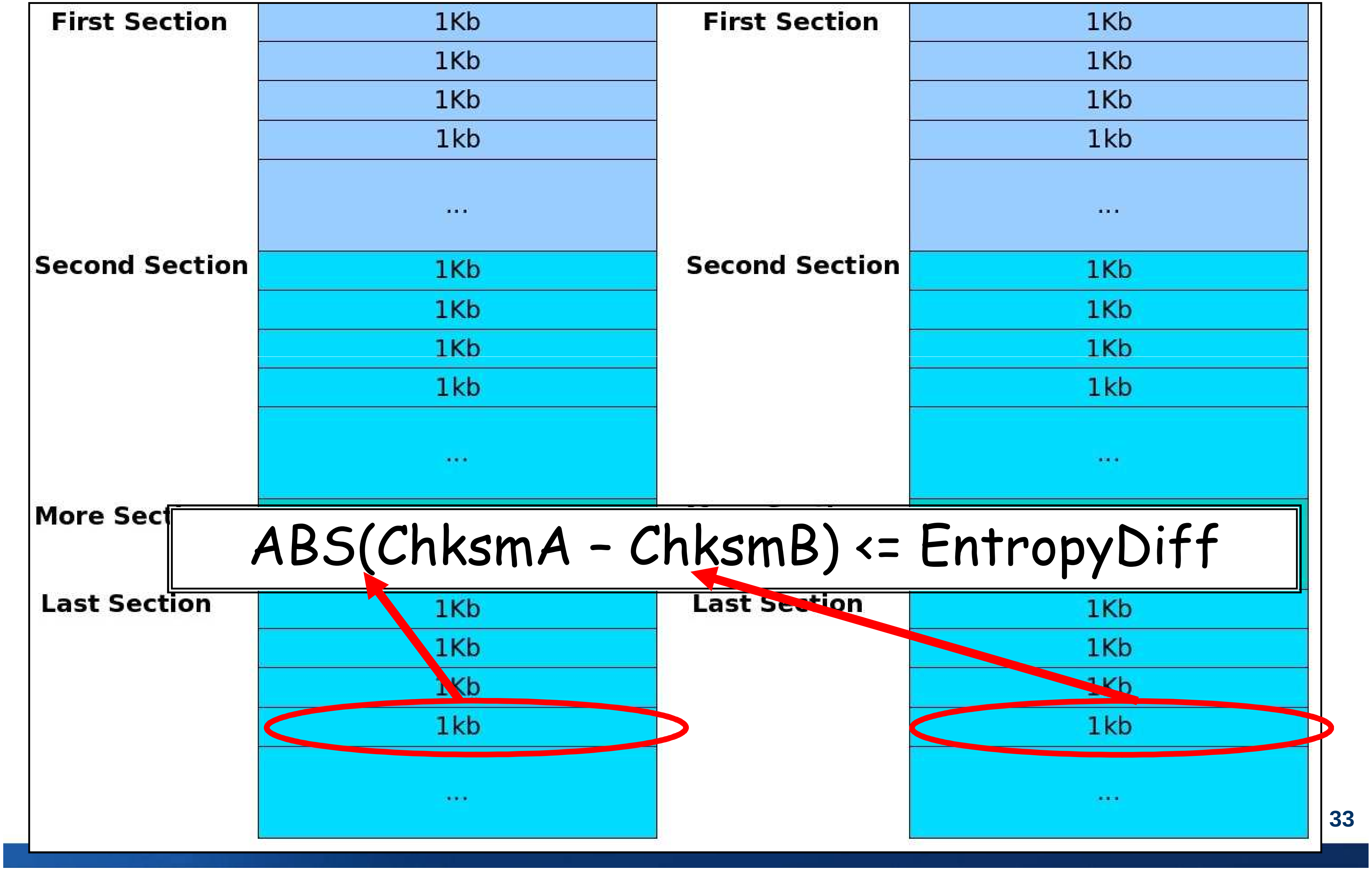
Sample A		Sample B	
First Section	1Kb	First Section	1Kb
	1Kb		1Kb
	1Kb		1Kb
	1kb		1kb
Second Section	1Kb	Second Section	1Kb
	1Kb		1Kb
	1Kb		1Kb
	1kb		1kb
	...		...
More Sections	...	More Sections	...
Last Section	1Kb	Last Section	1Kb
	1Kb		1Kb
	1Kb		1Kb
	1kb		1kb
	...		...

ABS(ChksmA - ChksmB) <= EntropyDiff

Automatic Comparison of Malware

Sample A

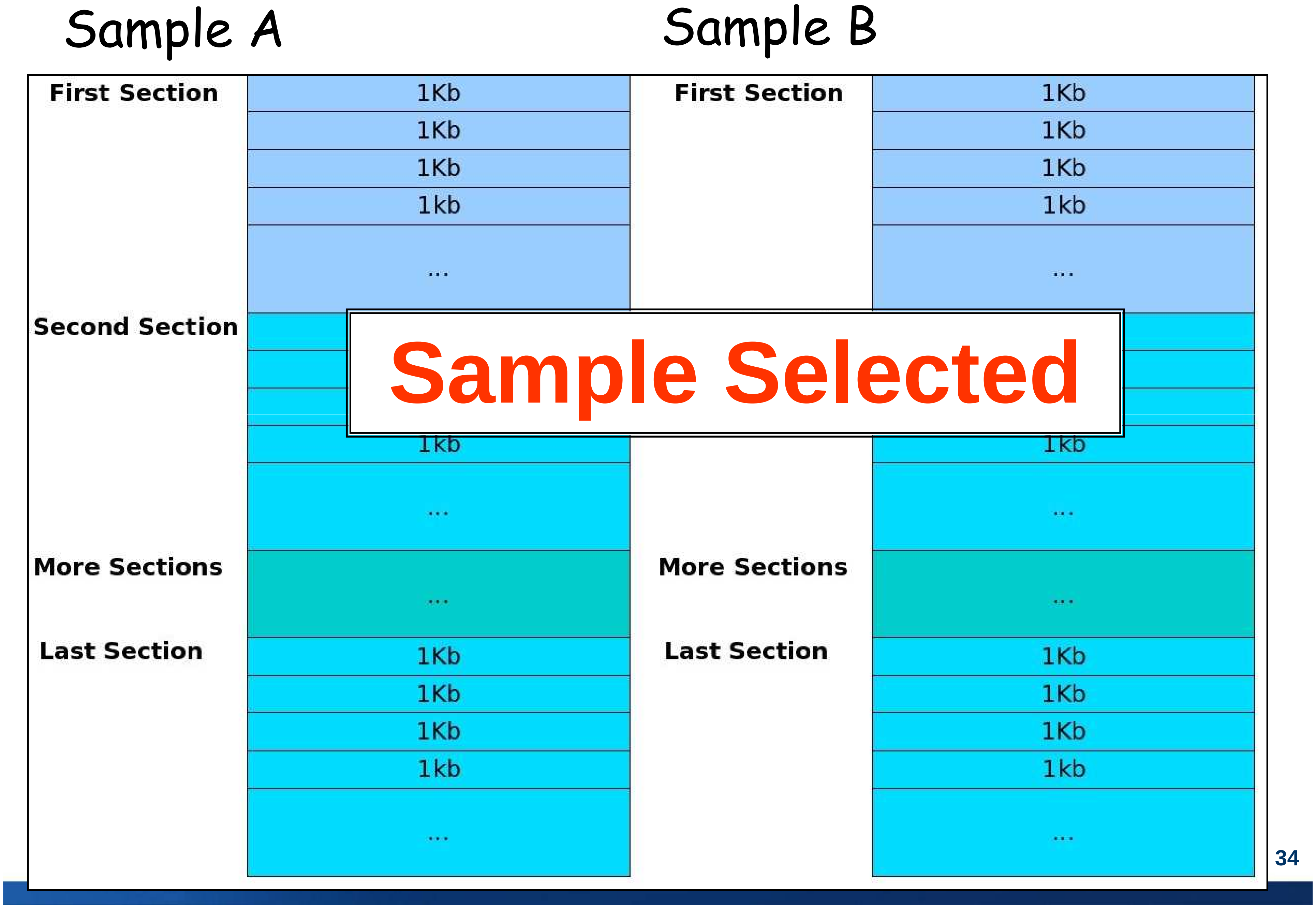
Sample B



First Section	1Kb	First Section	1Kb
	1Kb		1Kb
	1Kb		1Kb
	1kb		1kb
	...		...
Second Section	1Kb	Second Section	1Kb
	1Kb		1Kb
	1Kb		1Kb
	1kb		1kb
	...		...
More Sect			
Last Section	1Kb	Last Section	1Kb
	1Kb		1Kb
	1Kb		1Kb
	1kb		1kb
	...		...

ABS(ChksmA – ChksmB) <= EntropyDiff

Automatic Comparison of Malware



Sample A

Sample B

First Section

1Kb

1Kb

1Kb

1kb

...

Second Section

1kb

...

More Sections

...

Last Section

1Kb

1Kb

1Kb

1kb

...

First Section

1Kb

1Kb

1Kb

1kb

...

1kb

...

More Sections

...

Last Section

1Kb

1Kb

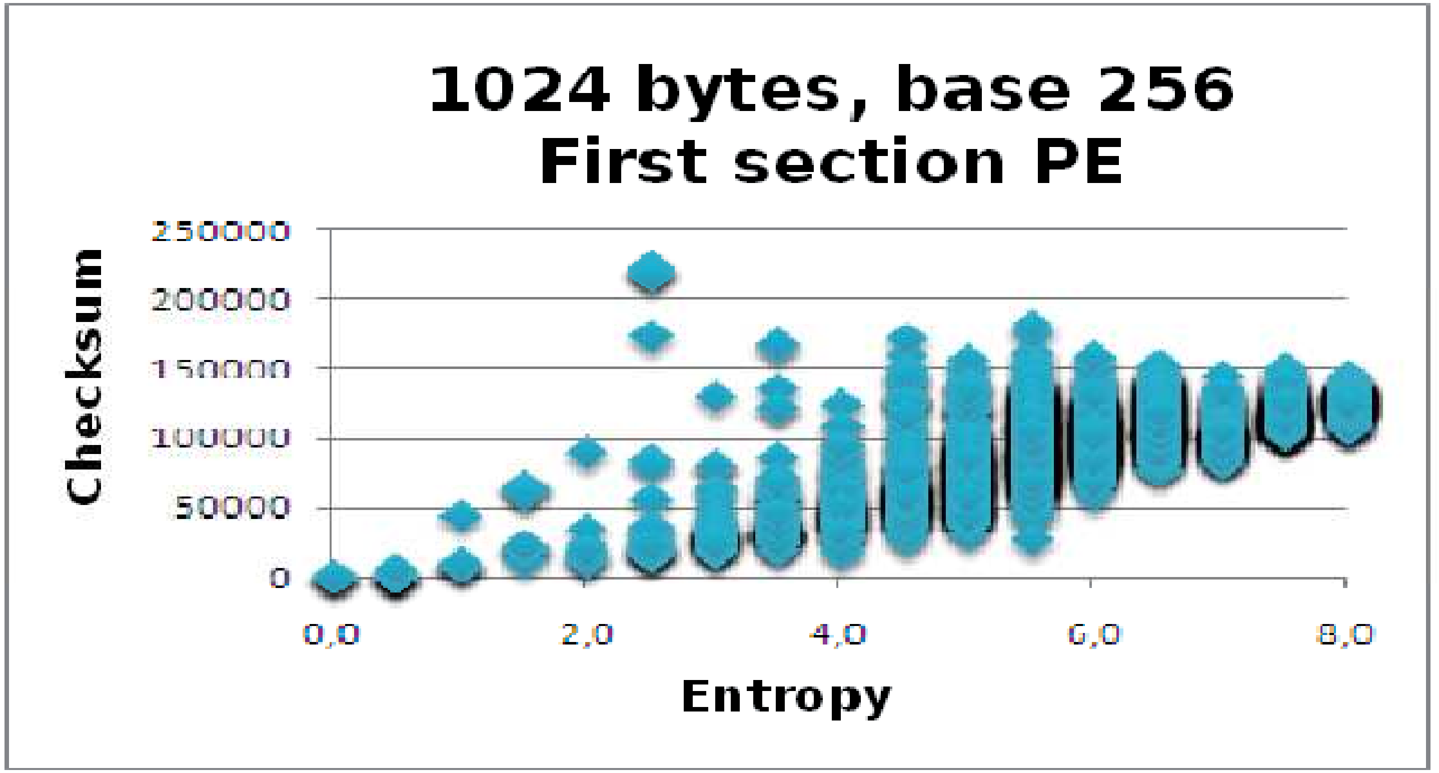
1Kb

1kb

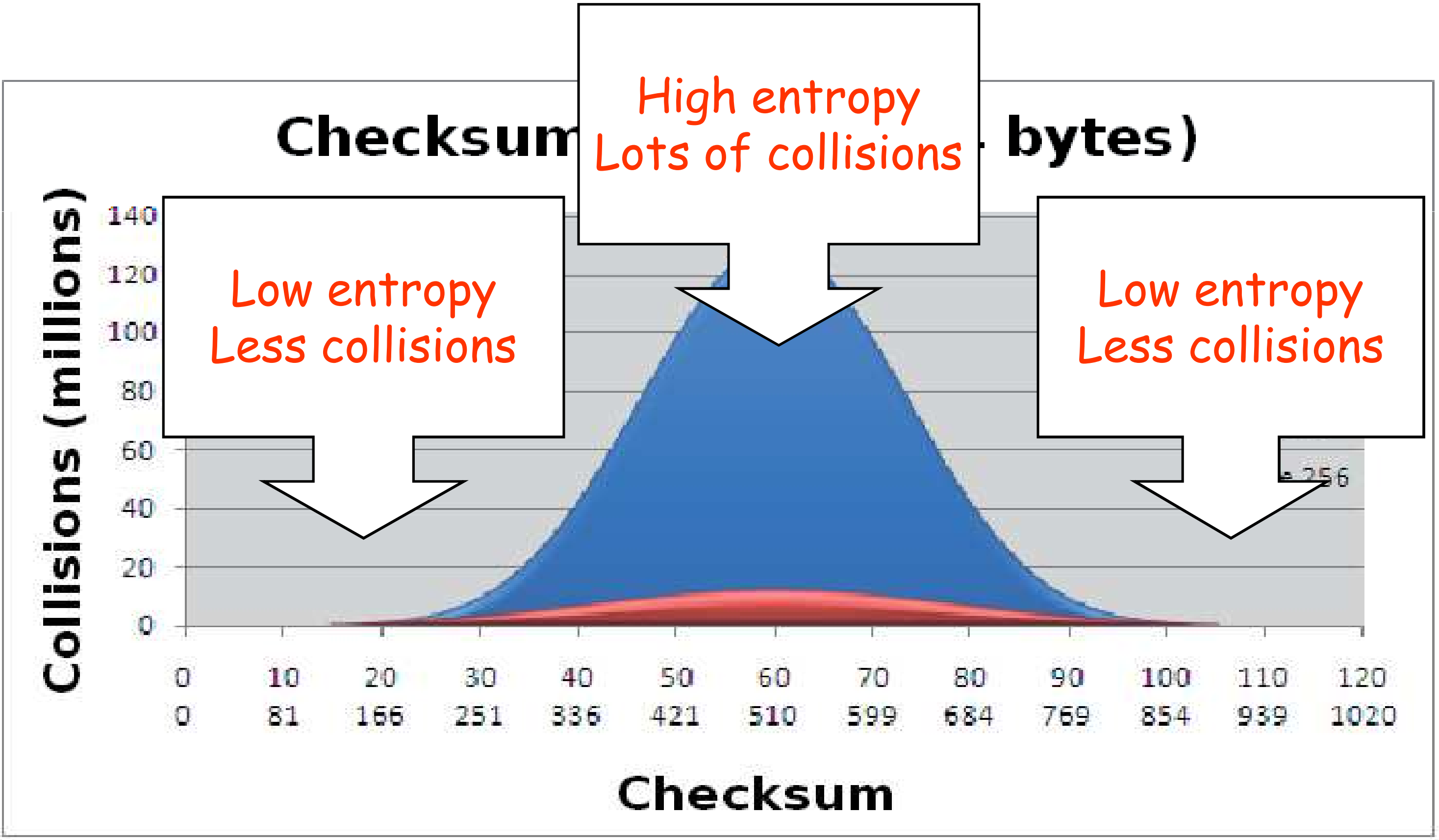
...

Sample Selected

Checksum vs Entropy



Checksum Collisions & Entropy

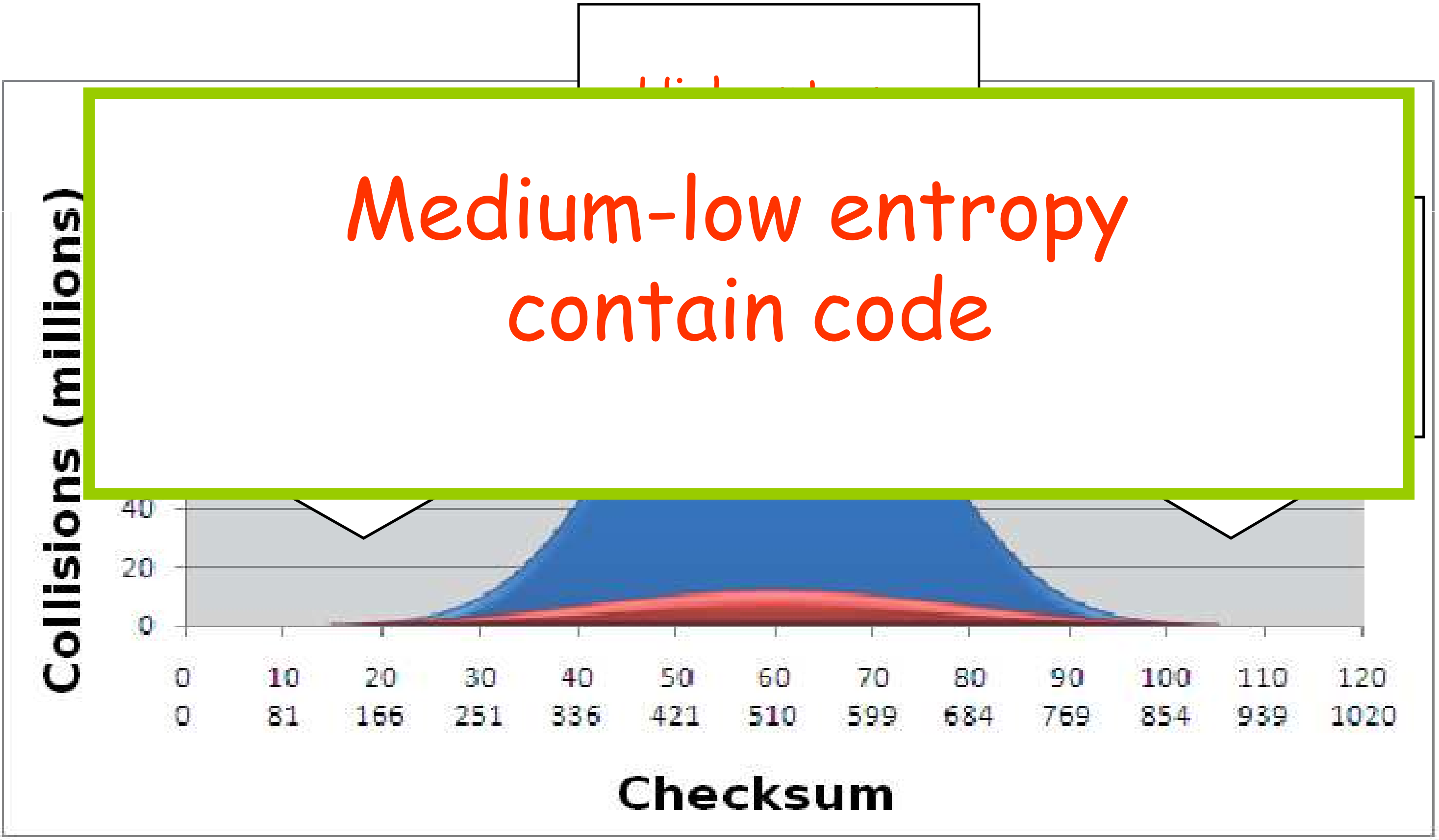


High entropy
Lots of collisions

Low entropy
Less collisions

Low entropy
Less collisions

Checksum Collisions & Entropy



Medium-low entropy contain code

Penalized checksum

- Don’t use blocks with high entropy
- Penalized checksum:

$$if\ (Entropy \geq 7)$$

$$\Downarrow$$

$$PenalizedChecksum = \log2((8 - Entropy)) \text{ x } (-60) \text{ x } Checksum$$

- Minimal checksum changes in high-entropy blocks will generate distant checksum values.

Comparison Algorithm

- Select best samples
- Start comparison:
  - Graph Comparison

Automatic Comparison of Malware

- Matrix A

	a1	a2	a3	a4
f1	0	1	0	1
f2	0	0	0	0
f3	1	0	0	0
f4	0	0	1	0

- Matrix B

	a1	a2	a3	a4
f1’	0	1	0	1
f2’	0	0	0	0
f3’	1	0	0	0
f4’	1	0	0	0

Automatic Comparison of Malware

➢Matrix A

	a1	a2	a3	a4
f1	0	1	0	1
f2	0	0		
f3	1	0		
f4	0	0		

➢Matrix B

	a1	a2	a3	a4
f1'	0	1	0	1
				0
				0
				0

Common functions:

- **API Functions**
- **Same and unique CRC32**
- **Same and unique CFG**

- Matrix A

	a1	a2	a3	a4
f1	0	1	0	1
f2	0	0	0	0
f3	1	0	0	0
f4	0	0	1	0

- Matrix B

	a1	a2	a3	a4
f1'	0	1	0	1
f2'	0	0	0	0
f3'	1	0	0	0
f4'	1	0	0	0

- Matrix A

	a1	a2	a3	a4
f1	0	1	0	1
f2	0	0	0	0
f3	1	0	0	0
f4	0	0	1	0

- Matrix B

	a1	a2	a3	a4
f1’	0	1	0	1
f2’	0	0	0	0
f3’	1	0	0	0
f4’	1	0	0	0

f1=f1’ and f1 != {f2’,f3’,f4’}

- Matrix A

	a1	a2	a3	a4	f1
f2	0	0	0	0	1
f3	1	0	0	0	1
f4	0	0	1	0	0

- Matrix B

	a1	a2	a3	a4	f1’
f2’	0	0	0	0	1
f3’	1	0	0	0	1
f4’	1	0	0	0	0

Automatic Comparison of Malware

PANDA SECURITY

- Matrix A

	a1	a2	a3	a4	f1
f2	0	0	0	0	1
f3	1	0	0	0	1
f4	0	0	1	0	0

- Matrix B

	a1	a2	a3	a4	f1’
f2’	0	0	0	0	1
f3’	1	0	0	0	1
f4’	1	0	0	0	0

f2=f2’ and f2 != {f3’,f4’}

- Matrix A

	a1	a2	a3	a4	f1	f2
f3	1	0	0	0	1	1
f4	0	0	1	0	0	1

- Matrix B

	a1	a2	a3	a4	f1’	f2’
f3’	1	0	0	0	1	0
f4’	1	0	0	0	0	1

f3 != {f3’,f4’} and f4 != {f3’,f4’}

Two functions have been identified: f1 and f2

Comparison Algorithm

- ➢Select best samples
- ➢Start comparison:
  - ➢Graph Comparison
- ➢Match more functions with CFG

Control Flow Graph

- CFG signature = 3-tuple vector in Euclidean space
- Find minimal and unique ED among functions

$$P=(p_x,p_y,p_z)$$
$$Q=(q_x,q_y,q_z)$$

$$\sqrt{(p_x-q_x)^2+(p_y-q_y)^2+(p_z-q_z)^2}$$

Three-dimensional Euclidean distance

Comparison Algorithm

- Select best samples
- Start comparison:
  - Graph Comparison
- Match more functions with CFG
- Index of Similarity

Index of Similarity

- Measure how close two binaries are.

Index of Similarity

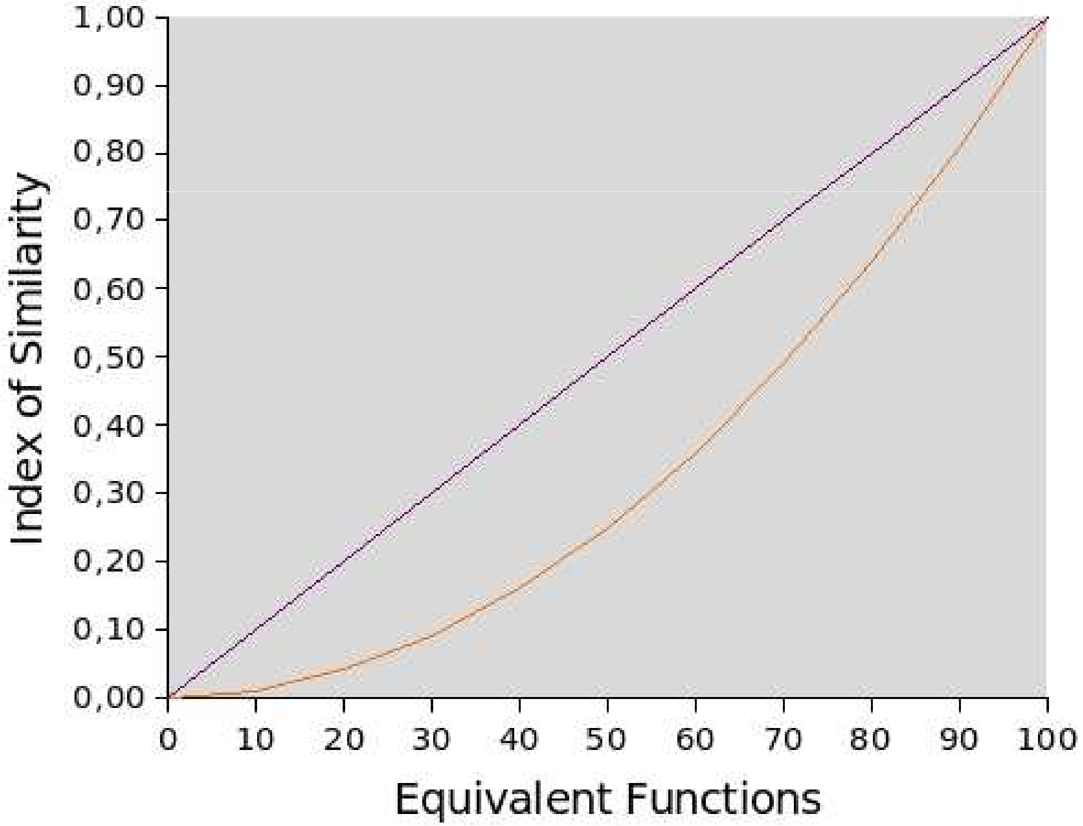
Index of similarity vs number of matched functions

First Exp.

$$\frac{|A_e||B_e|}{|A_f||B_f|} = \frac{|matchedfunction|^2}{|A_f||B_f|}$$

Second Exp.

$$\frac{|A_e|}{2|A_f|} + \frac{|B_e|}{2|B_f|}$$



Improvements

- More Initial Fixed Points

Automatic Comparison of Malware

More Fixed Points
SPP (Small Prime Product)
Identical String references
In/Out degree (similar number of calls to/calls from)
Match same name (sub_XXXXXX) if same CRC32
Stack Frame Size (similar stack frame size)

Improvements

- More Initial Fixed Points
- Python version improvements

Python version improvements

- Psyco: a Just-In-Time compiler
- Extend Python code with C

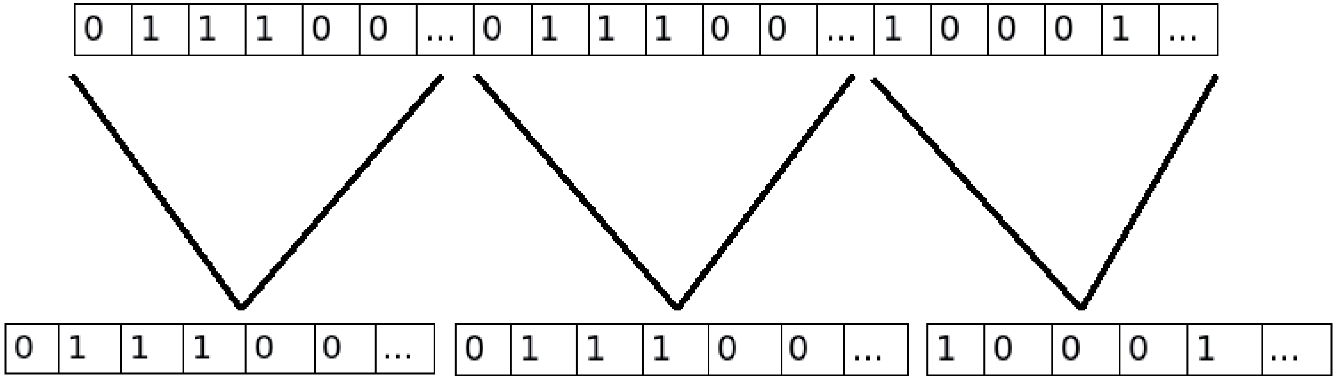
- From Python to C/C++
  - Develop the comparison algorithm in C/C++

Improvements

- ➢ More Initial Fixed Points
- ➢Python version improvements
- ➢Matrix rows as bits

Matrix rows as bits

- ➢Avoid string comparison (string comparison is a time-consuming task)
- ➢Treat rows as groups of bits (100110011101…)
- ➢Split rows as groups of 32 bits
- ➢Compare as integers (integer comparison is faster)



| |

0 | 1 | 1 | 1 | 0 | 0 | ... | 0 | 1 | 1 | 1 | 0 | 0 | ... | 1 | 0 | 0 | 0 | 1 | ...

0 | 1 | 1 | 1 | 0 | 0 | ...

0 | 1 | 1 | 1 | 0 | 0 | ...

1 | 0 | 0 | 0 | 1 | ...

Improvements

- More Initial Fixed Points
- Python version improvements
- Matrix rows as bits
- Stream SIMD extensions

Streaming SIMD extensions

- SSE added eight 128-bits registers: **XMM0-XMM7**
- Each register packs together four 32-bit integers
- Compare 4x32 (four integers) in one instruction:
  - __m128 _mm_cmpeq_ps(__m128 a, __m128 b)

Improvements

- More Initial Fixed Points
- Python version improvements
- Matrix rows as bits
- Stream SIMD extensions
- NVIDIA Cuda

NVIDIA Cuda

- Cuda: compiler and SDK for NVIDIA GPUs
- GPUs:
  - Parallel "many-core" architecture
  - Each core: thousands of threads simultaneously
- Not tried yet. Future development:
  - Code algorithm for graphics processing unit (GPU)
  - Launch one thread for each compared sample
  - Launch a thread for each compared row

Possibilities

- Port information from malware database
- Shared code => generic signature
  - Functions with same CRC32
- Clusterization of malware automatically
  - Classify unknown samples (index of similarity)

Barriers

- Only unpacked samples
- Hard bound to IDA
- Big database storage
- Delphi and VB samples don’t work well

Demo

Thank you very much

Questions?

ismael.briones@pandasecurity.com
aitor.gomez@pandasecurity.com