Experiments with Malware Visualization

Yongzheng Wu and *Roland H.C. Yap*

Singapore U. of Tech. & Design

Natl. U. of Singapore

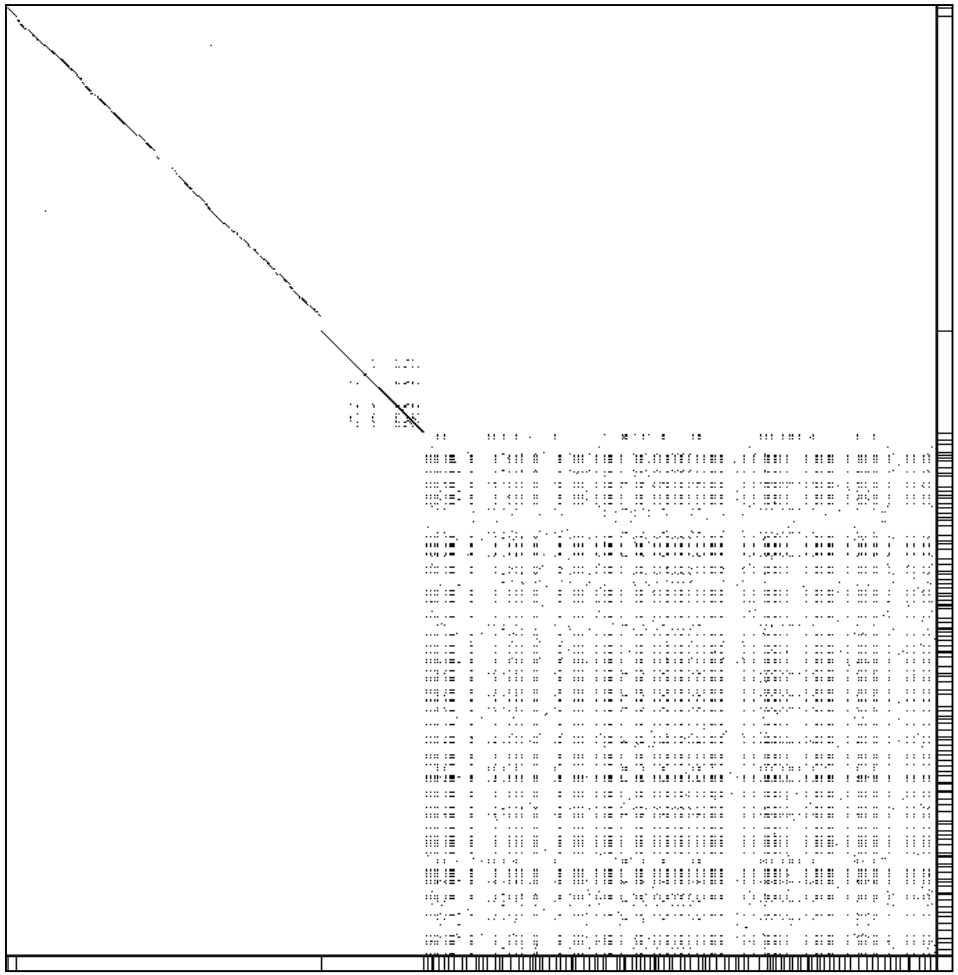
Why Malware Visualization

- Malware comparison, classification and clustering is not well defined
  - Sharing & Evolution: Reusable components and complex co-evolution history
  - No definite answers: Different anti-virus software give different classifications
- Can visualization show relationships between malware?
  - Not automatic analysis
  - Complementary to analysis

Motivating Applications

- Identify common components of two malware sample
- Identify new code in a new malware variant
- Identify changes made on benign software from virus
- Study relation between malware families
- Identify the family of an unknown malware sample

Visualization Preview



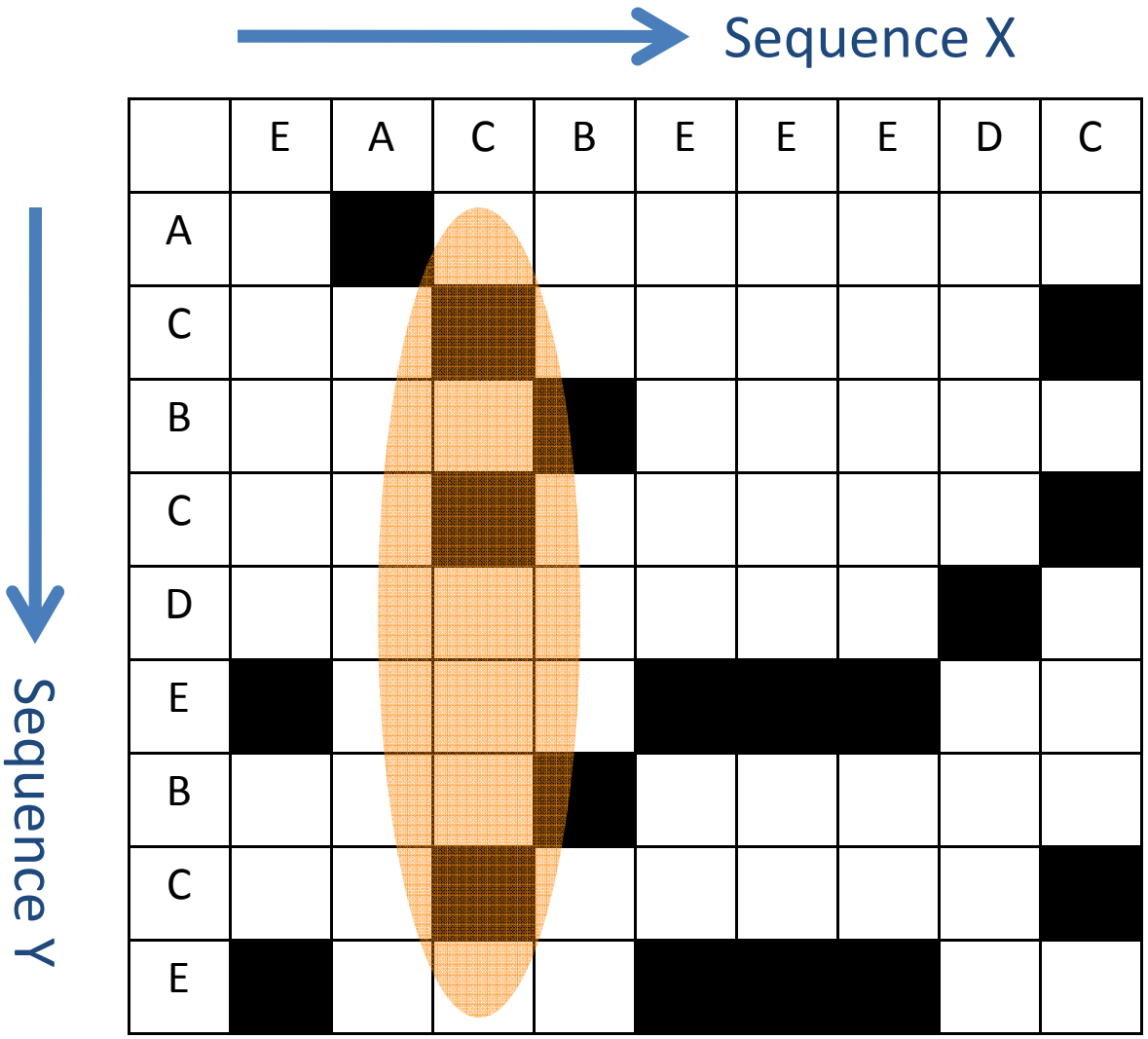
Background on DotPlot

Sequence X

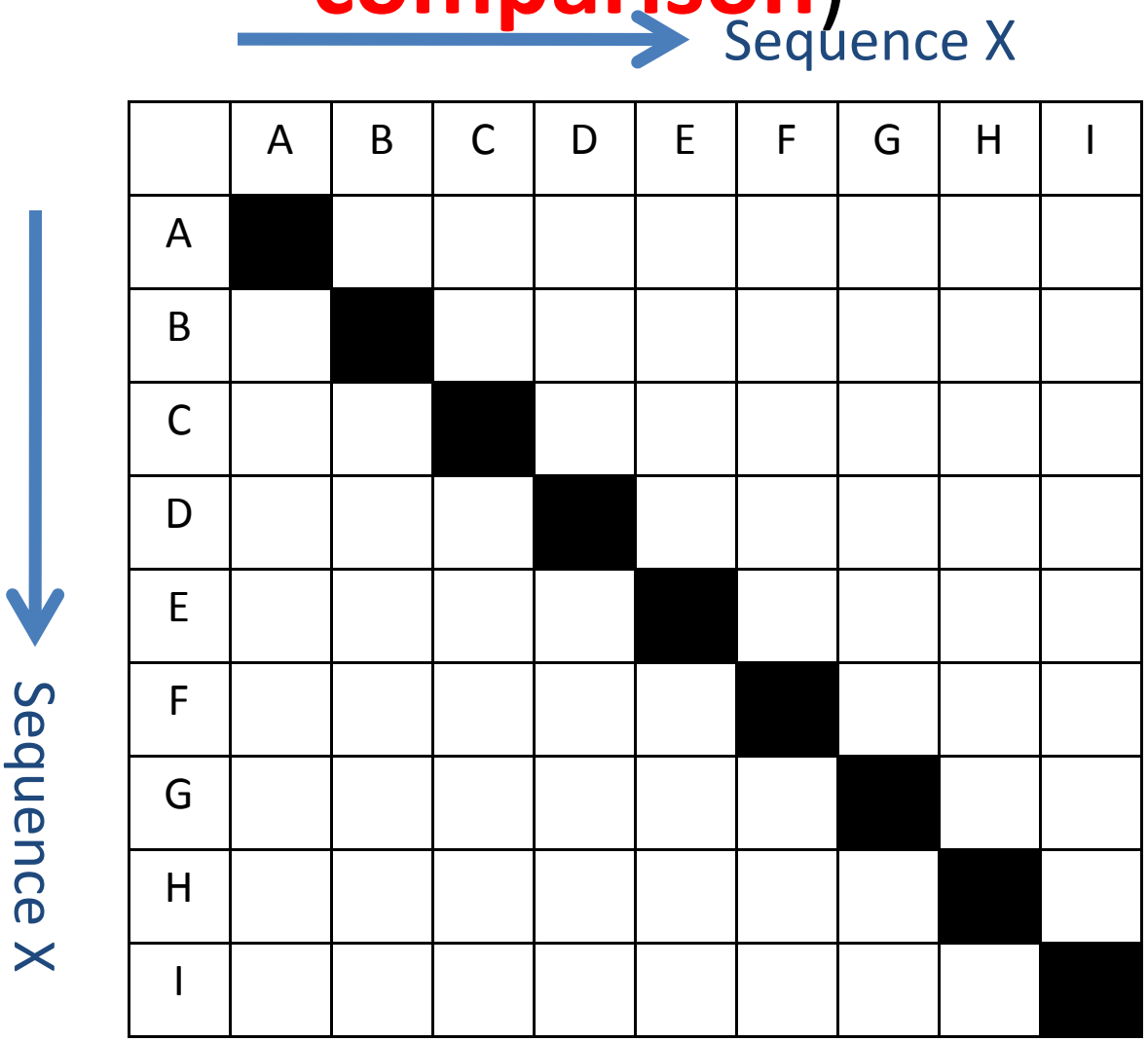
Sequence Y

	E	A	C	B	E	E	E	D	C
A									
C									
B									
C									
D									
E									
B									
C									
E									

Background on DotPlot

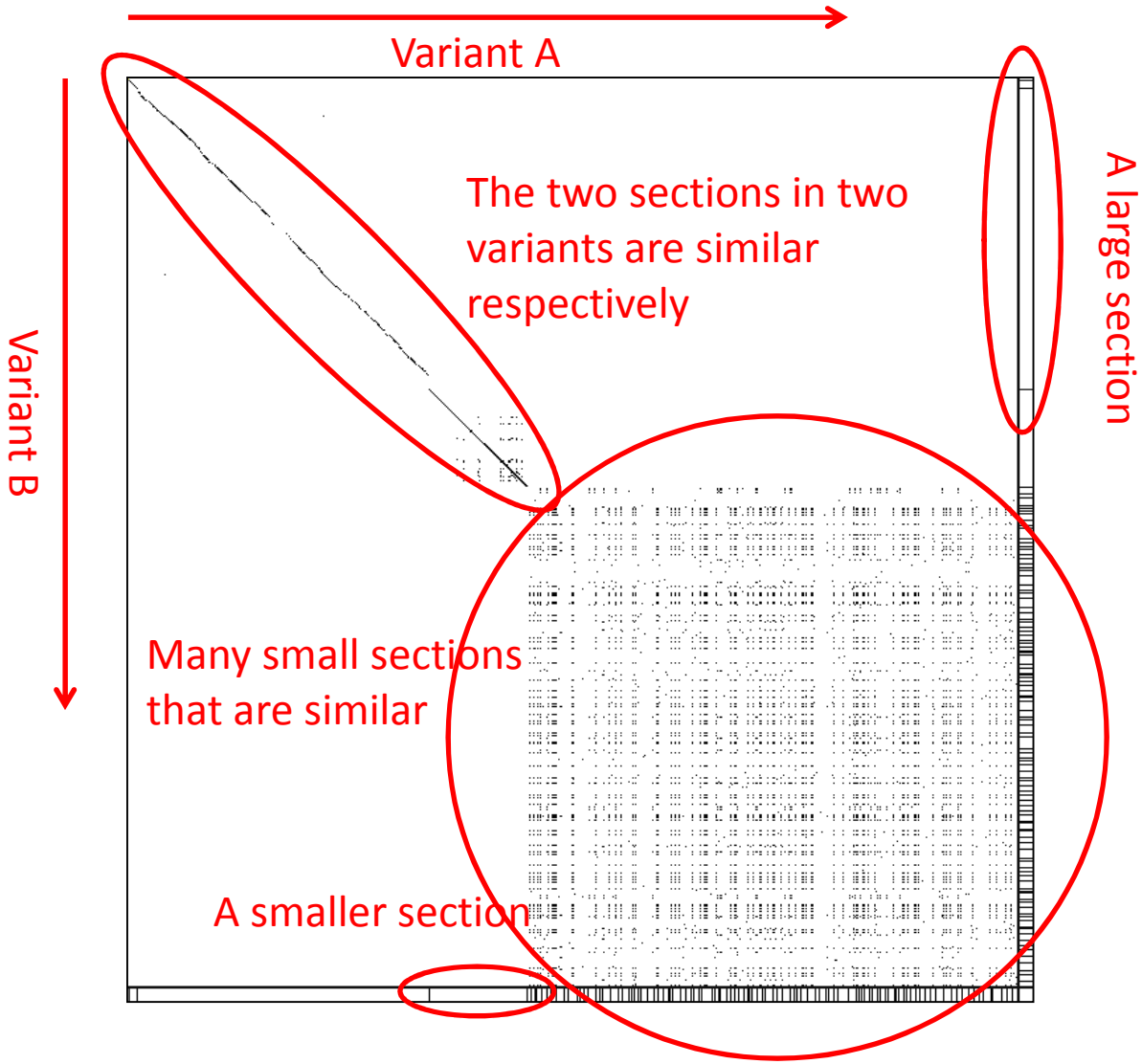


Background on DotPlot (**self-comparison**)



An example

Comparing two variants of Bagle

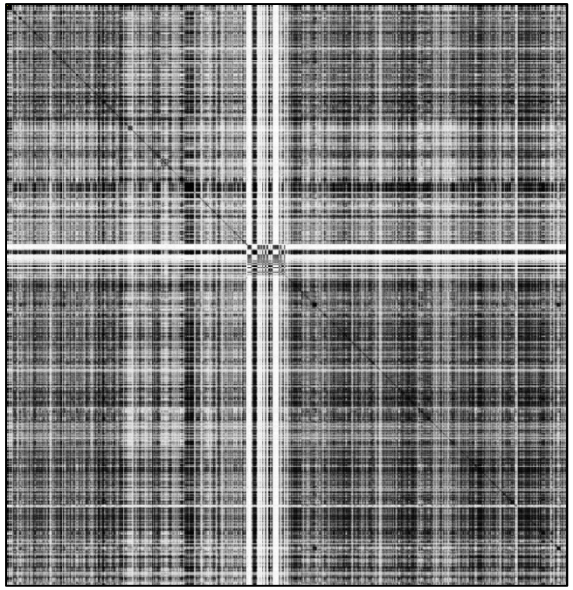


Sequence: Content & Sections

- Sequence coresponds to content of memory
  - Subset of "**memory dump**"
  - Executable pages (focus on the code)
  - Obtained after unpacking
- Sequence is divided into many sections, e.g. exe, DLL, anonymous

Processing The Instruction Sequence

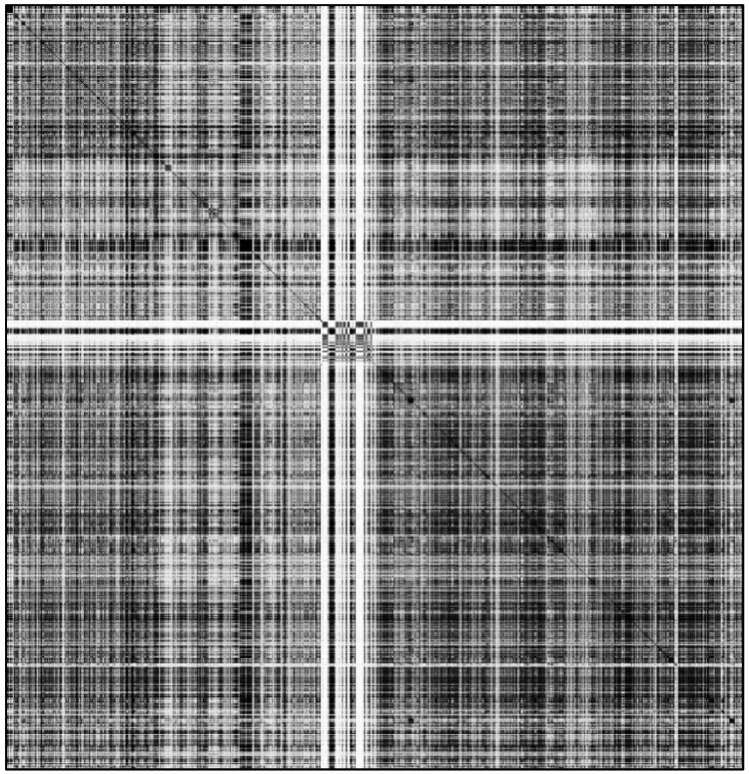
- Problem 1: Direct dotplot of the raw instruction sequence yelds too much similarity
  - Because of common instructions such as
    - `ret`
    - `nop`
    - `xor eax, eax`



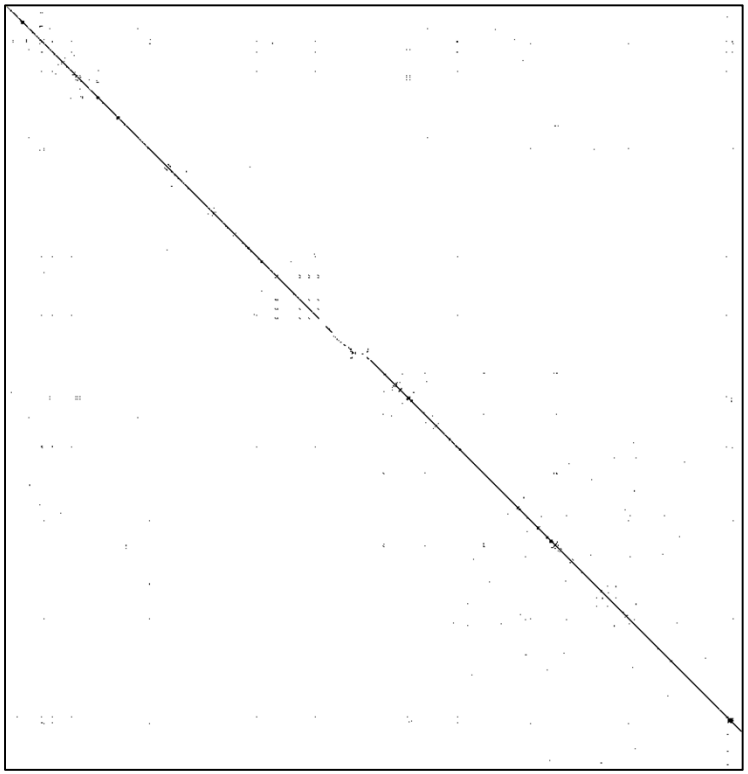
Processing The Instruction Sequence

- Solution: use n-gram
  - Compares n consecutive bytes rather than individual bytes
- What is n-gram?
  - Original: A B C A C D B
  - 2-gram: AB BC CA AC CD DB
  - 3-gram: ABC BCA CAC ACD CDB

Processing The Instruction Sequence



Raw Instructions

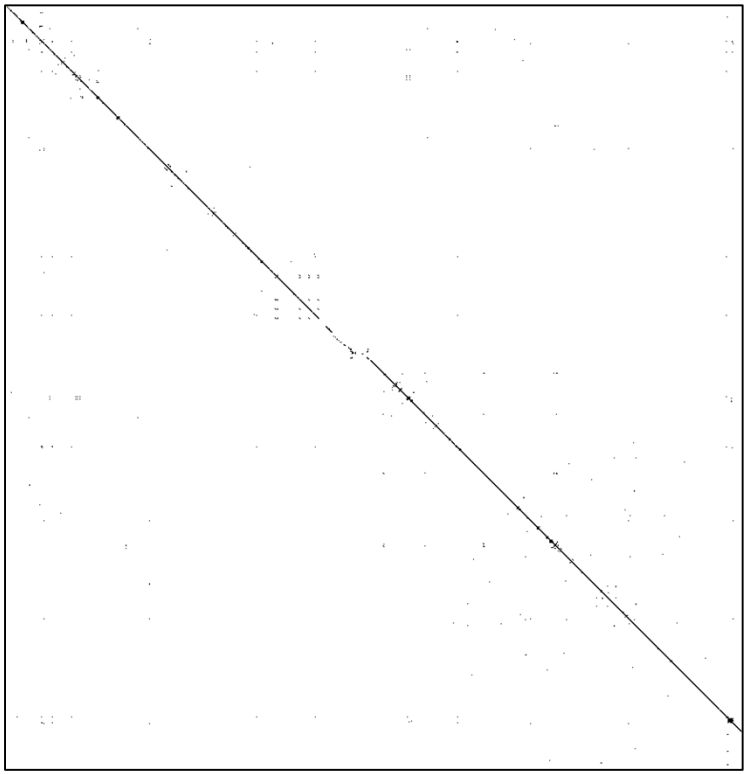


16-gram

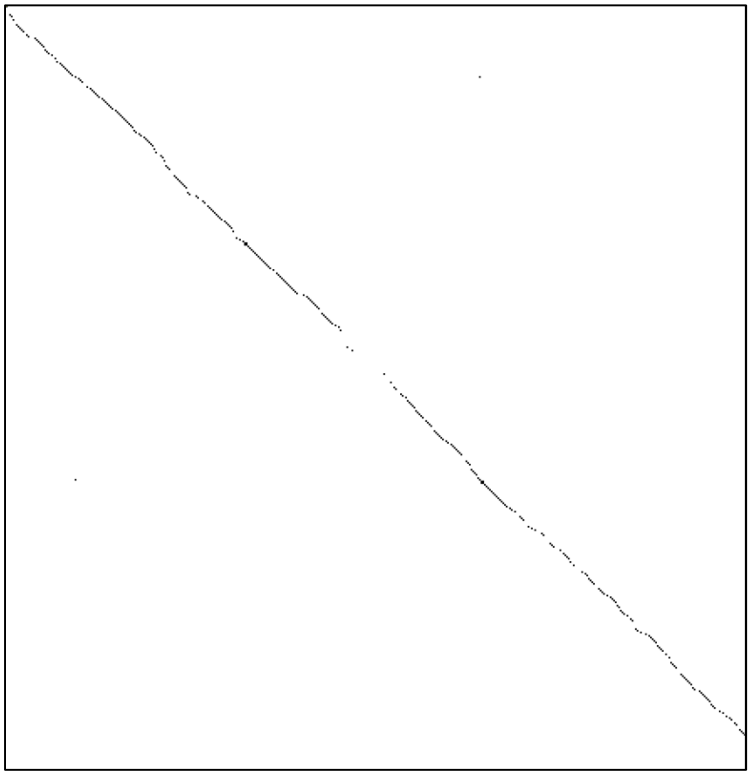
Processing The Instruction Sequence (cont.)

- Problem 2: Sequence is too large for visualization
  - The size of memory dump is typically 10 to 100MB
  - Comparing two 10M sequences yelds 10Mx10M image, i.e. 100T pixels!
  - Our interactive visualizer handles sequences up to ~500K (Gigapixel images)
  - Note: n-gram only reduce the size by n-1
- Solution: hash-based sampling
  - Reduce a sequence of size N to N/k
  - Sample an n-gram if its hash modulo k is 0

Processing The Instruction Sequence (cont.)



No sampling

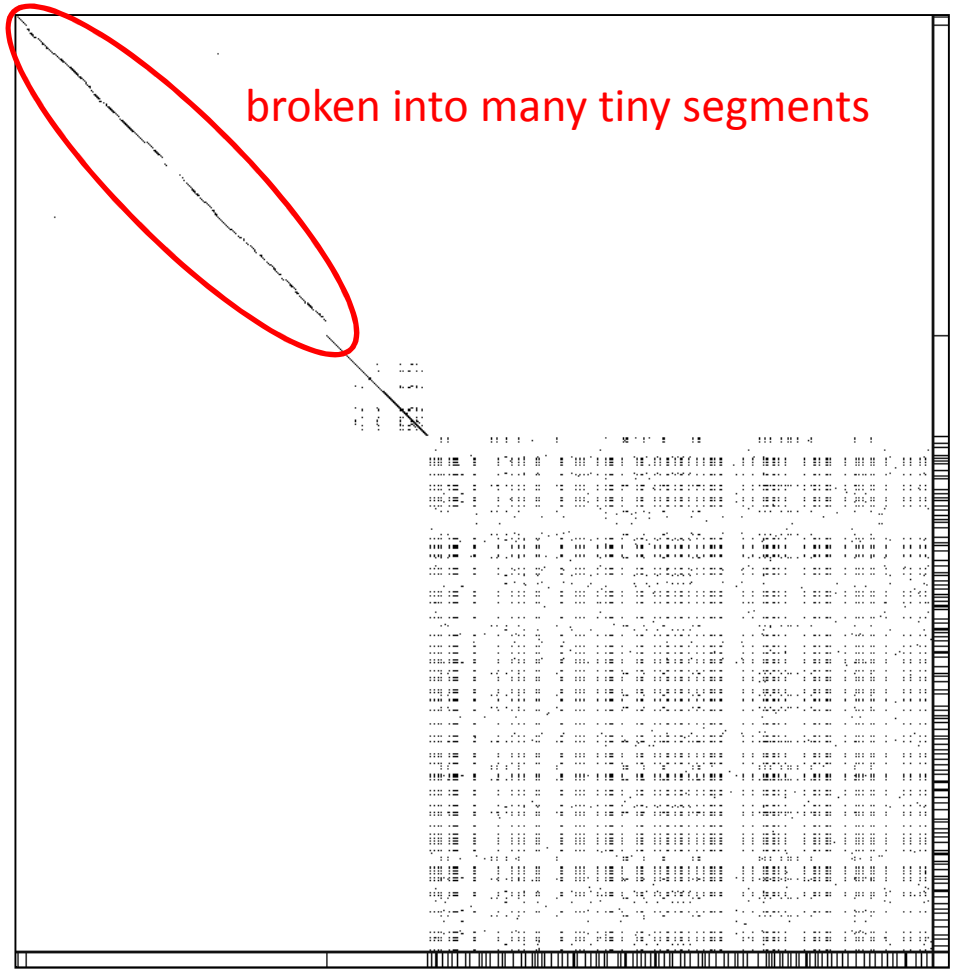


After 1:500 sampling

Application 1: Two Variants from Same Family

- Objective
  - Visualize similarity and difference of two variants from same family
- Data set used
  - Two Bagle variants

Application 1: Visualization



Application 1: Information Learned

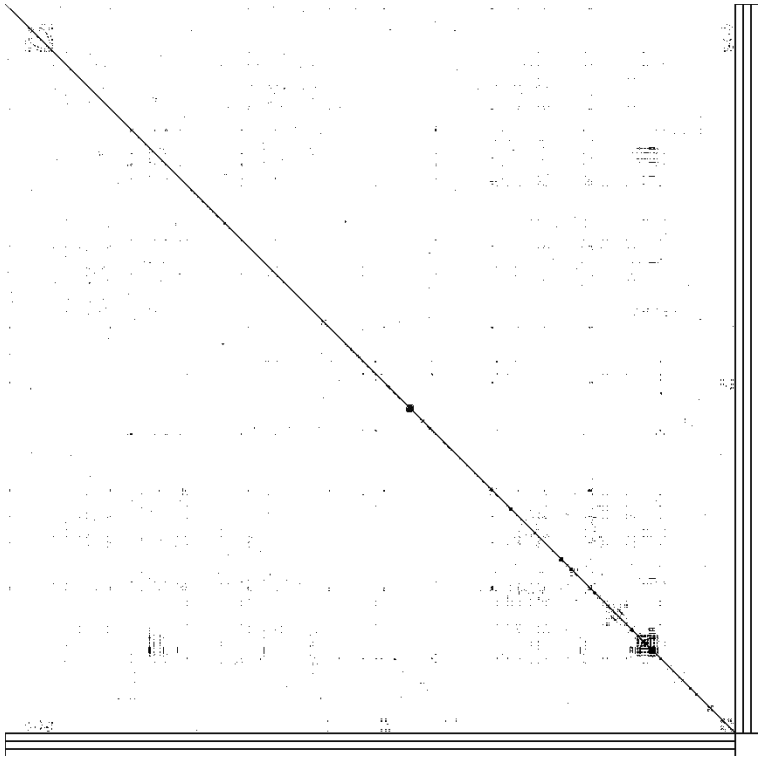
address	Bagle 1		Bagle 2	
	opcode	instruction	opcode	instruction
004013c1	68c0204500	`push 0x4520c0`	68c0204500	`push 0x4520c0`
004013c6	90	*nop*	e8c6055402	*call 0x2941991*
004013c7	e8f1045402	*call 0x29418bd*	90	*nop*
004013cc	ff15c0204500	`call [0x4520c0]`	ff15c0204500	`call [0x4520c0]`
004013e9	7505	`jnz 0x4013f0`	7505	`jnz 0x4013f0`
004013eb	e8af9a0100	*call 0x41ae9f*	e821a60100	*call 0x41ba11*
004013f0	50	`push eax`	50	`push eax`
004013f1	e8337a0300	*call 0x438e29*	e8a5850300	*call 0x43999b*
004013f6	cc	`int3`	cc	`int3`

- Trivial polymorphic code
- About 5000 different fragments (6%) like this
- 94% code is same in both variants

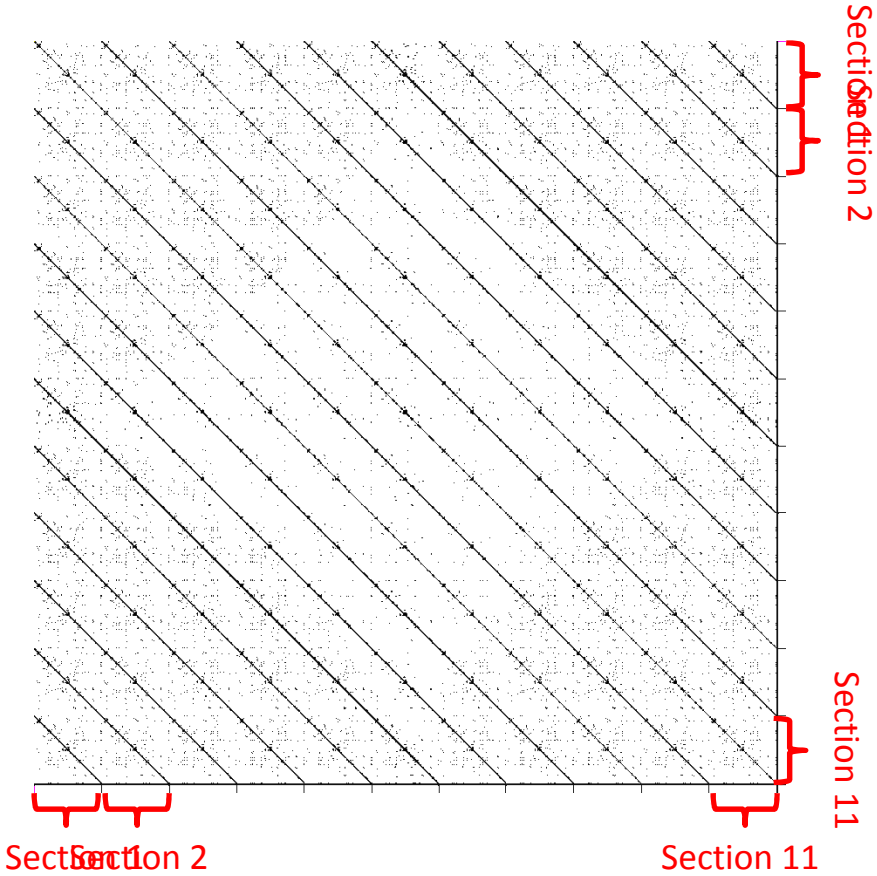
Application 2: Discover API Hooking by Comparing System DLL

- API hooking is usually done by patching the API function entry
- Without hooking, sections of a system DLL are same in different dumps
- We can compare sections, which are **different**, of a system DLL.

Application 2: Visualization



Two different sections of **kernel32.dll** from Hupigon



Self-comparison of 11 different sections of **ntdll.dll** (10 Conficker variants and 1 benign software)

Application 2: Information Learned

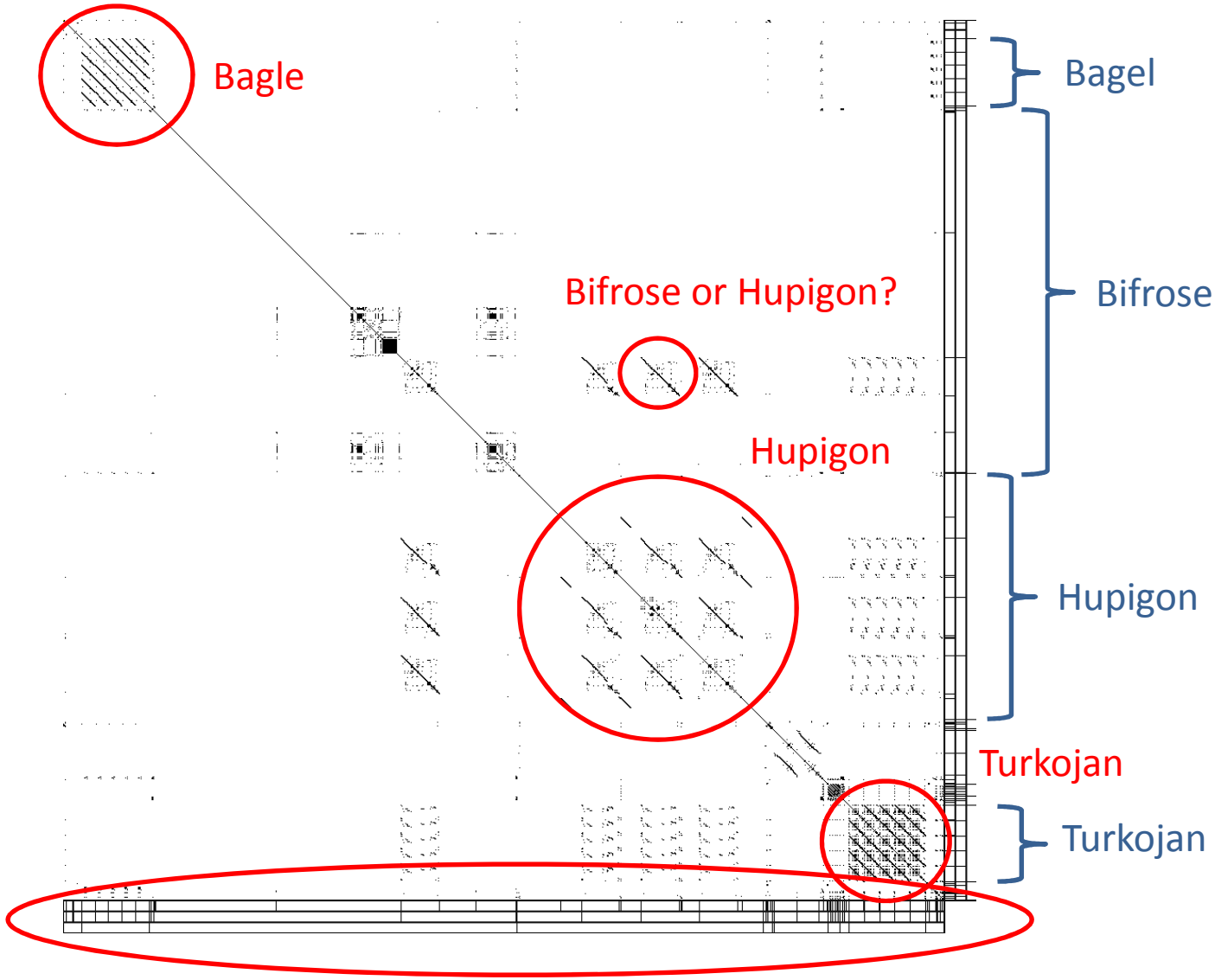
address	benign opcode	benign instruction	Hupigon opcode	Hupigon instruction
7c801d7a	90	nop	90	nop
7c801d7b	8bff	*mov edi,edi*	e9dd22c483	*jmp 0x44405d*
7c801d7d	55	*push ebp*		
7c801d7e	8bec	*mov ebp,esp*		
7c801d80	837d0800	cmp [ebp+0x8],0	837d0800	cmp [ebp+0x8],0
7c8197af	90	nop	90	nop
7c8197b0	68080a0000	*push 0xa08*	e9079dc283	*jmp 0x4434bc*
7c8197b5	68889a817c	push 0x7c819a88	68889a817c	push 0x7c819a88

- API hooking in Hupigon. 0x7c801d7b is the entry of `LoadLibraryA()`
- 0x7c8197b0 is the entry of `CreateProcessInternalW()`

Application 3: Visualizing Malware Families

- Comparing 60 malware instances: 5 instances × 12 malware families
  - Total size 142M
- Try to visualize malware clustering

Application 3: Visualizing Malware Families (self comparison, only exe sections)

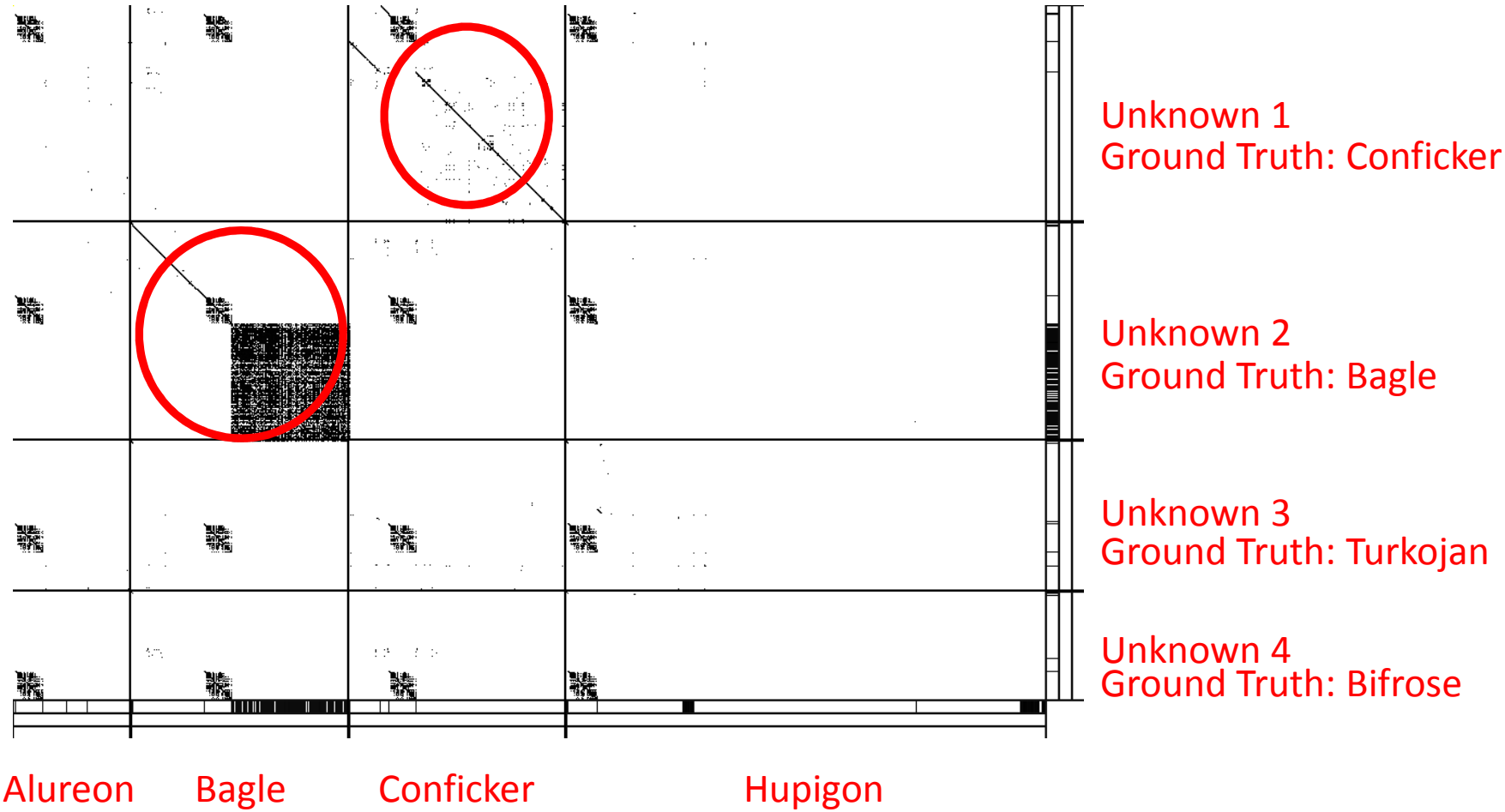


Three Barcodes.
Inner: Sections
Middle: Variants
Outer: Family

Application 4: Identify Unknown Malware

- Given a few known samples.
- We want to compare against existing known families
  - Can we identify family of a new sample?

Application 4: Identify Unknown Malware



Limitations & Conclusion

- Limitations
  - Sophisticated obsfucated code
  - Scalability: meant to work with selected samples
- Conclusion
  - Effective in showing the similarity in the internal structure of malware.
  - Show similarities between families.
  - Identify unknown malware sample
  - Can visualize other properties of sequence
    - Instruction/basic block/function sequence
    - System call sequence
    - Memory access