Hybrid defense: how to protect yourself from polymorphic 0-days

Svetlana Gaivoronski
PhD student

Dennis Gamayunov
Senior researcher

Lomonosov Moscow State University

Summary

- Motivation
- The state-of-the-art
- Proposed approach
- Demorpheus
- Evaluation

Why should one care about 0days at all

Isn't it 2013 out there?

Memory corruptions, 0 days, shellcodes

Nowdays... CONS

- Old exploitation technique, too old for Web-2.0-and-Clouds- Everywhere-World (some would say...)
- According to Microsoft's 2011 stats*, user unawareness is #1 reason for malware propagation, and 0-days are less than 1%
- Endpoint security products deal with known malware quite well, why should we care about unknown?..

* http://download.microsoft.com/download/0/3/3/0331766E-3FC4-44E5-B1CA-2BDEB58211B8/Microsoft_Security_Intelligence_Report_volume_11_Zeroing_in_on_Malware_ Propagation_Methods_English.pdf1

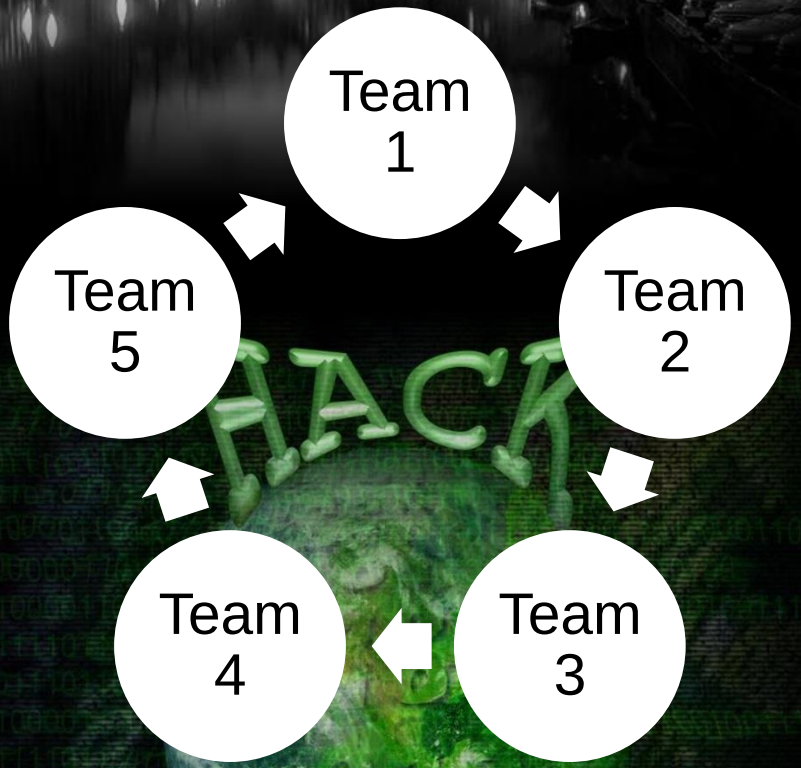
Nowdays... PROS

Memory corruption vulns are still there ;-)

- Hey, Microsoft, we're all excited with MS12-020
- Heyyy, Sun!.. Oracle, sorry. We're even more excited with CVE-2013-0422, thaanks

- Tools like Metasploit are widely used by pentesters and blackhat community
- Targeted attacks of critical infrastructure - what about early detection?
- Endpoint security is mostly signature-based, and does not help with 0-days

CTF Madness



- Teams write 0-days from scratch
- Game traffic is full of exploits all the time
- Detection of shellcode allows to get hints about your vulns and ways of exploitation…

Privacy and Trust in Digital Era

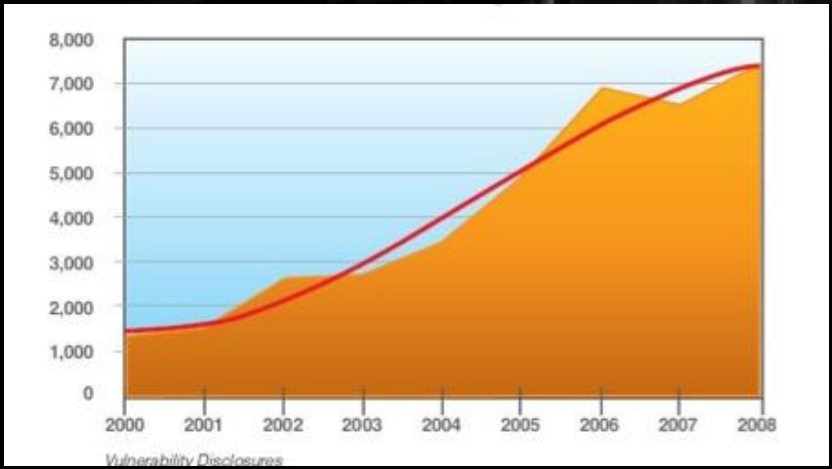
We share almost all aspects of our lives with digital devices (laptops, cellphones and so on) and Internet:

- Bank accounts
- Health records
- Personal information

Recent privacy issues with social networks and cloud providers:

- LinkedIn passwords hashes leak
- Foursquare vulns
- What's next?..

May be risk of 0-days will fade away?



- Modern software market for mobile and social applications is too competitive for developers to invest in security
- Programmers work under pressure of time limitation; managers who prefer quantity and no quality, etc.

Despite the fact of significant efforts to improve code quality, the number of vulnerabily disclosures continues to grow every year…

The state-of-the-art

Types of shellcode detection

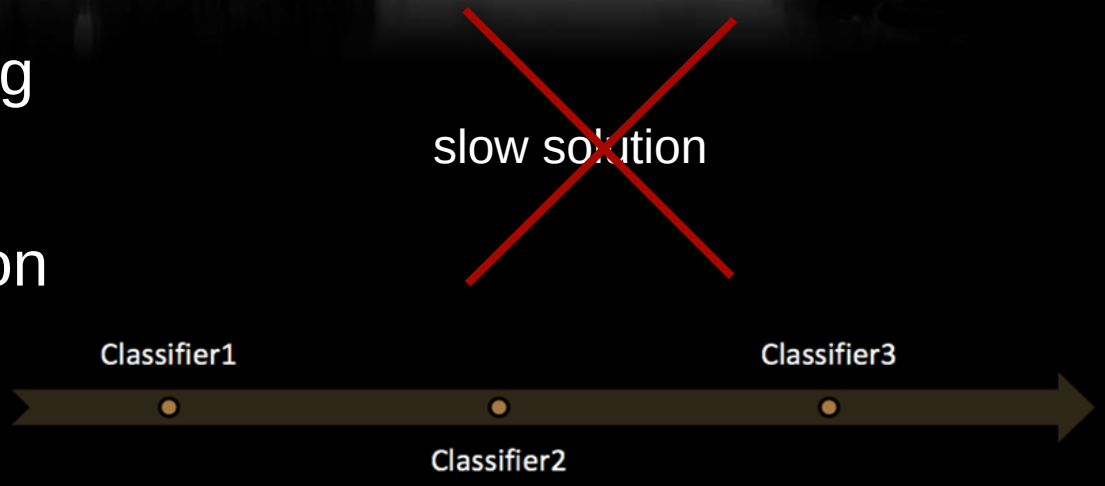
Static analysis

Dynamic analysis

Hybrid analysis

Techniques

- Static
  - signature matching
  - CFG\IFG analysis
  - NOP-sled detection
  - APE
- Dynamic
  - emulation
  - automata analysis



Virtues and shortcomings

Static methods	Dynamic methods
+ Complete code coverage + In most cases work faster	+ More resistant to obfuscation
- The problem of metamorphic shellcode detection is undecidable - The problem of polymorphic shellcode detection is NP-complete	- Require some overheads - Consider a few control flow paths - There are still anti-dynamic analysis techniques

Conclusion?

- Methods with low computation complexity have high FP rate
- Methods with low FP have high computation complexity
- They are also have problems with detection of new types of 0-day exploits
- **None of them is applicable for high throughput data channels**

Proposed approach

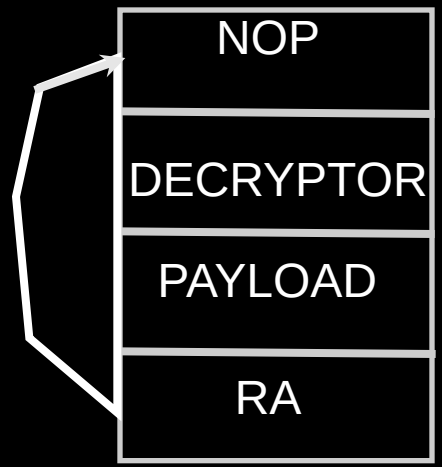
Shellcode schema

```
1   42                    -  inc    %edx
2   96                    -  xchg   %eax,%esi
3   f8                    -  clc
4   3c 48                 -  cmp    $0x48,%al
5   88 d5                 -  mov    %dl,%ch
6   90                    -  nop
7   97                    -  xchg   %eax,%edi
8   41                    -  inc    %ecx
9   35 93 98 24 92        -  xor    $0x92249893,%eax
10  4b                    -  dec    %ebx
11
12  d9 d0                 -  fnop
13  be 93 f2 c1 1e        -  mov    $0x1ec1f293,%esi
14  d9 74 24 f4           -  fnstenv -0xc(%esp)
15  5a                    -  pop    %edx
16  2b c9                 -  sub    %ecx,%ecx
17  b1 0a                 -  mov    $0xa,%cl
18  83 ea fc              -  sub    $0xfffffffc,%edx
19  31 72 13              -  xor    %esi,0x13(%edx)
20  03 e1                 -  add    %ecx,%esp
21  e1 23                 -  loope  0x3d
22
23  eb 9c                 -  jmp    0xffffffb8
24  6c                    -  insb   (%dx),%es:(%edi)
25  ab                    -  stos   %eax,%es:(%edi)
26  4c                    -  dec    %esp
27  cc                    -  int3
28  98                    -  cwtl
29  bf 6c f0 58 ef        -  mov    $0xef58f06c,%edi
30  09 84 3b c0 a2 0c dd  -  or     %eax,-0x22f35d40(%ebx,%edi,1)
31  7a 2a                 -  jp     0x59
32  bb 1d d8 dc f5        -  mov    $0xf5dcd81d,%ebx
33  1f                    -  pop    %ds
34  de 1c a0              -  ficomp (%eax,%eiz,4)
35  d2 5e 76              -  rcrb   %cl,0x76(%esi)
36  53                    -  push   %ebx
37  b5 93                 -  mov    $0x93,%ch
38  07                    -  pop    %es
```

NOP-sled

DECRYPTOR

ENCRYPTED PAYLOAD



Why not?

- We are given the set of shellcode detection algorithms characterized by:
  - execution time
  - FP and FN rate
  - classes coverage
- Let’s try to construct optimal data flow graph:
  - execution time and FP are optimized
  - classes coverage is complete

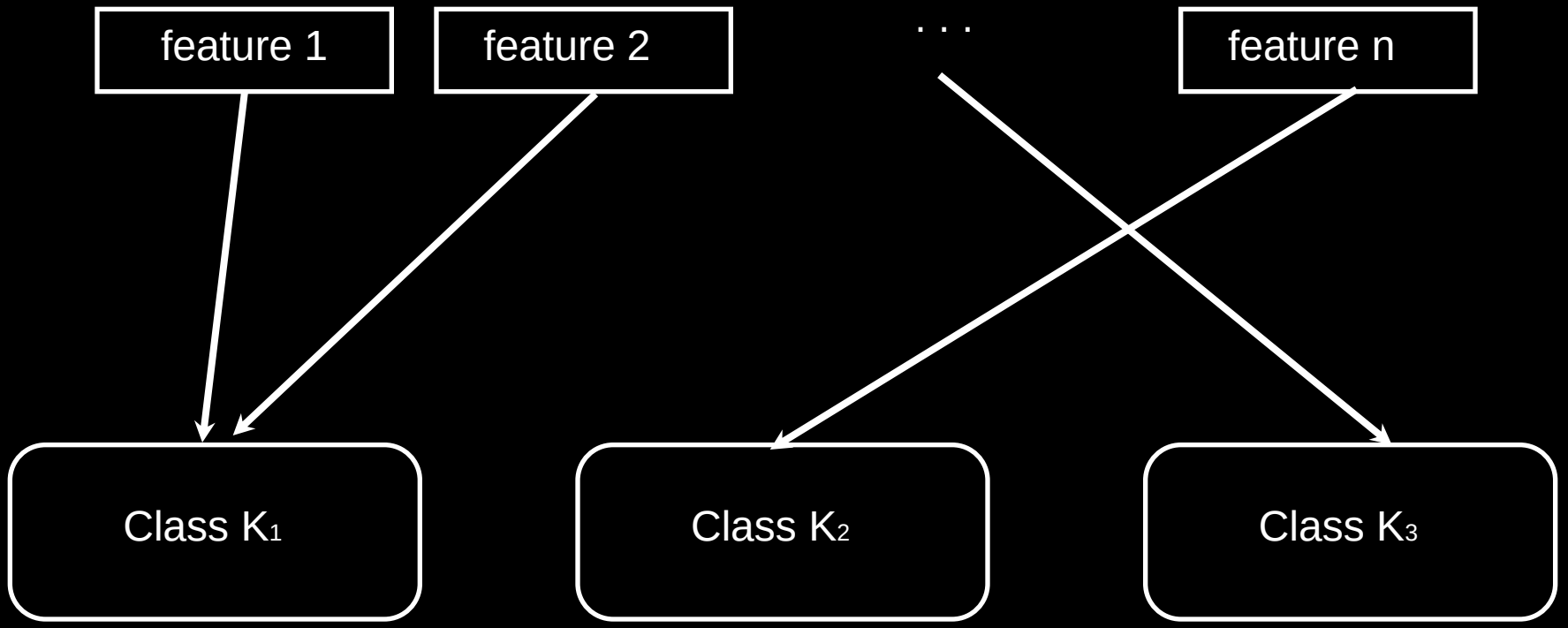
Shellcode static features

Generic features	Specific features
- Correct dissasembly int chain at least of K instructions; - Number of push-call patterns exceeds threshold; - Overall size does not exceed threshold; - Operands of self-modifying and indirect jmp are initialized; - Cleared IFG contains chain with more than N instructions;	- Correct disassembly from each and every offset; - Conditional jumps to the lower address offset; - Ret address lies within certain range of values; - MEL exceeds threshold; - Presence of GetPC; - Specific type of last chain instruction; Last instruction in the chain ends with branch instruction with immediate or absolute addressing targeting lib call or valid interruption

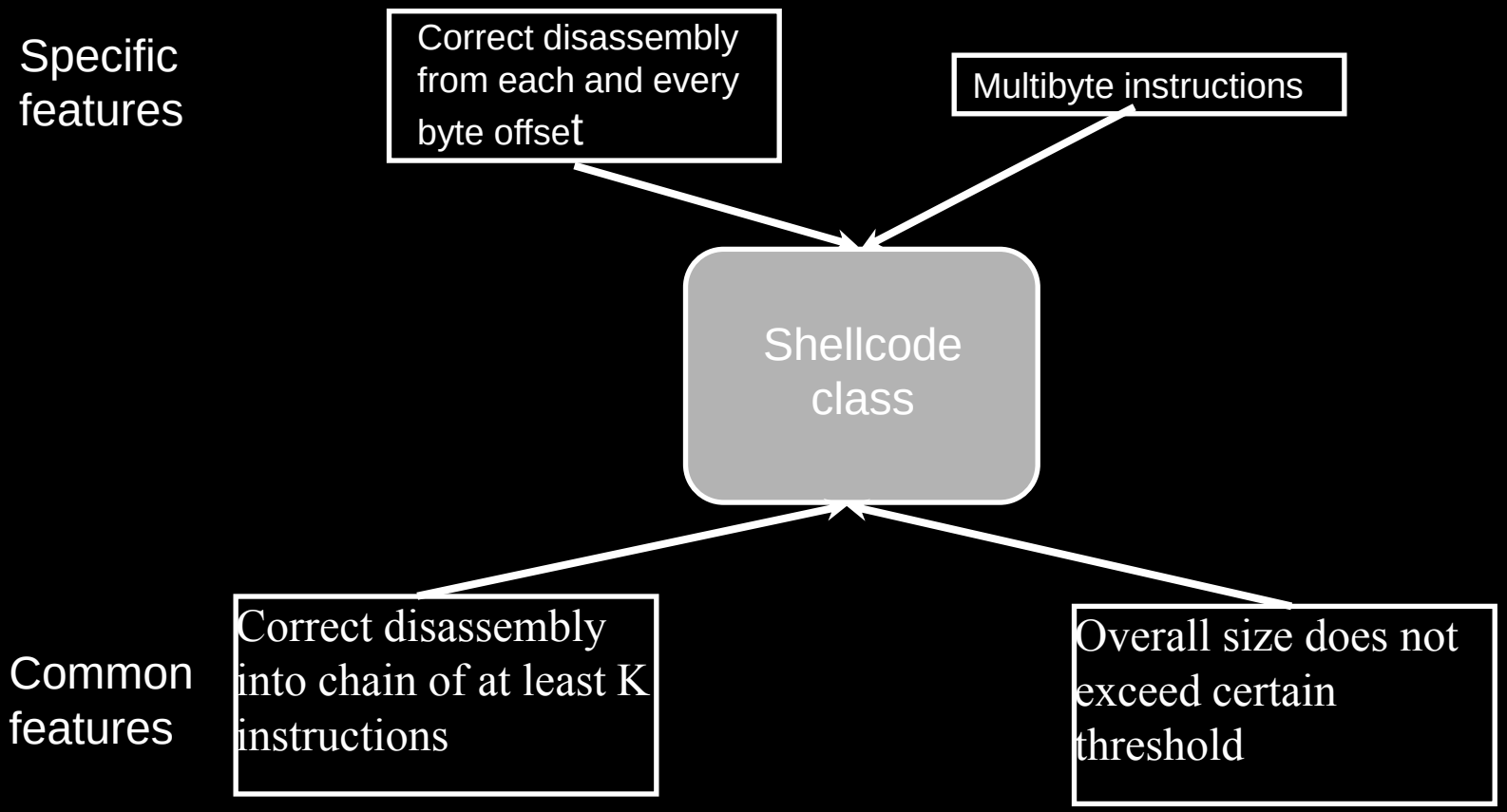
Shellcode dynamic features

Generic features	Specific features
- Number of near reads within payload exceed threshold R - Number of unique writes to different memory location exceeds threshold W	- Control at least once transferred from executed payload to previously written address - Execution of wx-instruction exceeds threshold X

Shellcode classes. Main idea



Example. Multibyte NOP-equivalent sled



List of activator-based classes

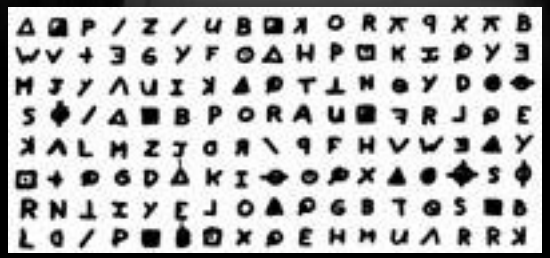
- Contain simple NOP-sled of 0x90 instruction which does not affect control flow, and only increases program counter
- Contain one-byte NOP-equivalent sled
- Contain multi-byte NOP-equivalent sled
- Contain four-byte aligned sled
- Contain trampoline sled
- Contain trampoline sled, obfuscated by injection NOP-equivalent instruction
- Contain static analysis resistant sled
- Contain GetPC code
- …

List of payload-based classes

- Contains plain, unobfuscated shellcodes
- Shellcodes with data obfuscation
- Shellcodes obfuscated with instruction reordering
- Shellcodes obfuscated by replacing instructions with other instructions with same operational semantics
- Shellcodes obfuscated with code injection
- Metamorphic shellcodes, using two levels of metamorphism: algorithm level and opcode level
- …

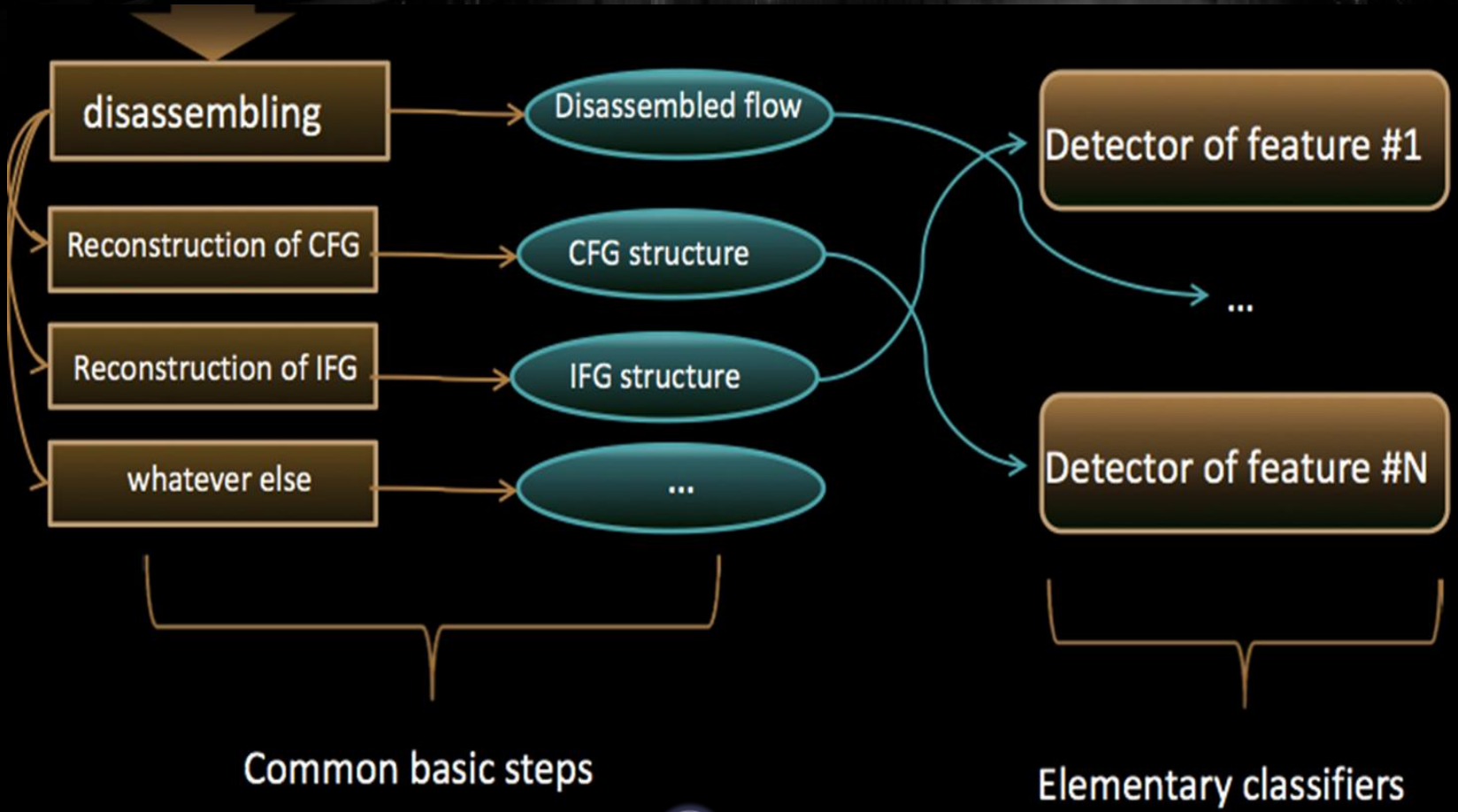
List of decryptor/RET-based classes

- Self-unpacking shellcodes
- Self-ciphered shellcodes
- Non-self-contained shellcode
- …
- Shellcodes with invariant ranges of return address zone
- Shellcodes with obfuscated return address

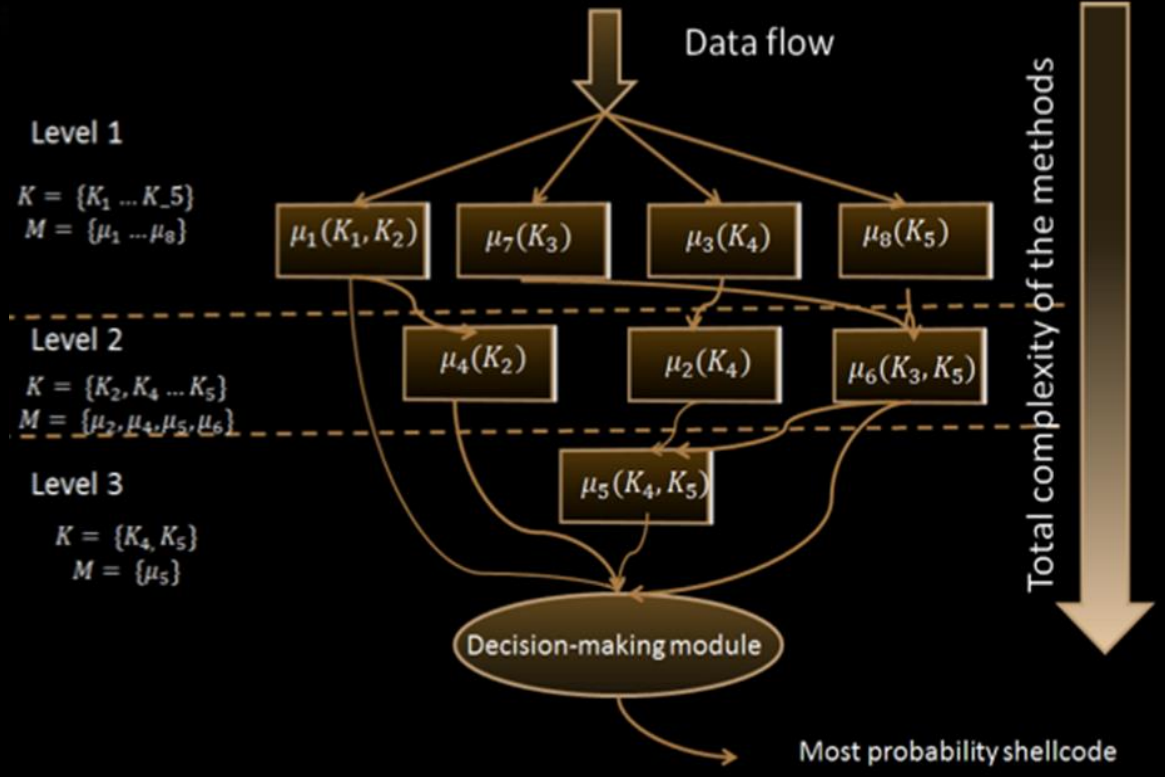


Demorpheus

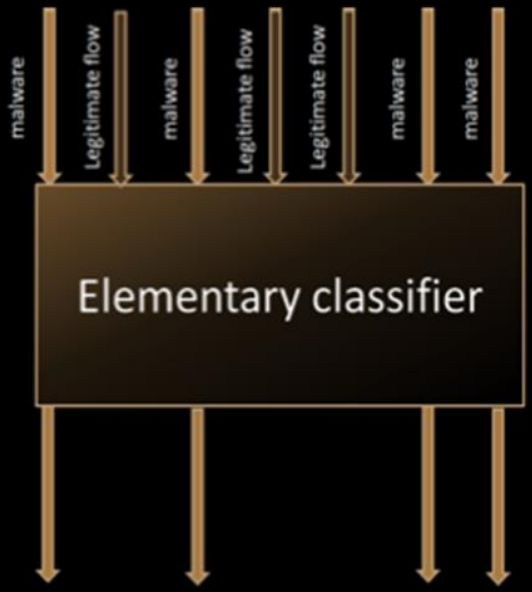
Shellcode detection library



Hybrid shellcode detector

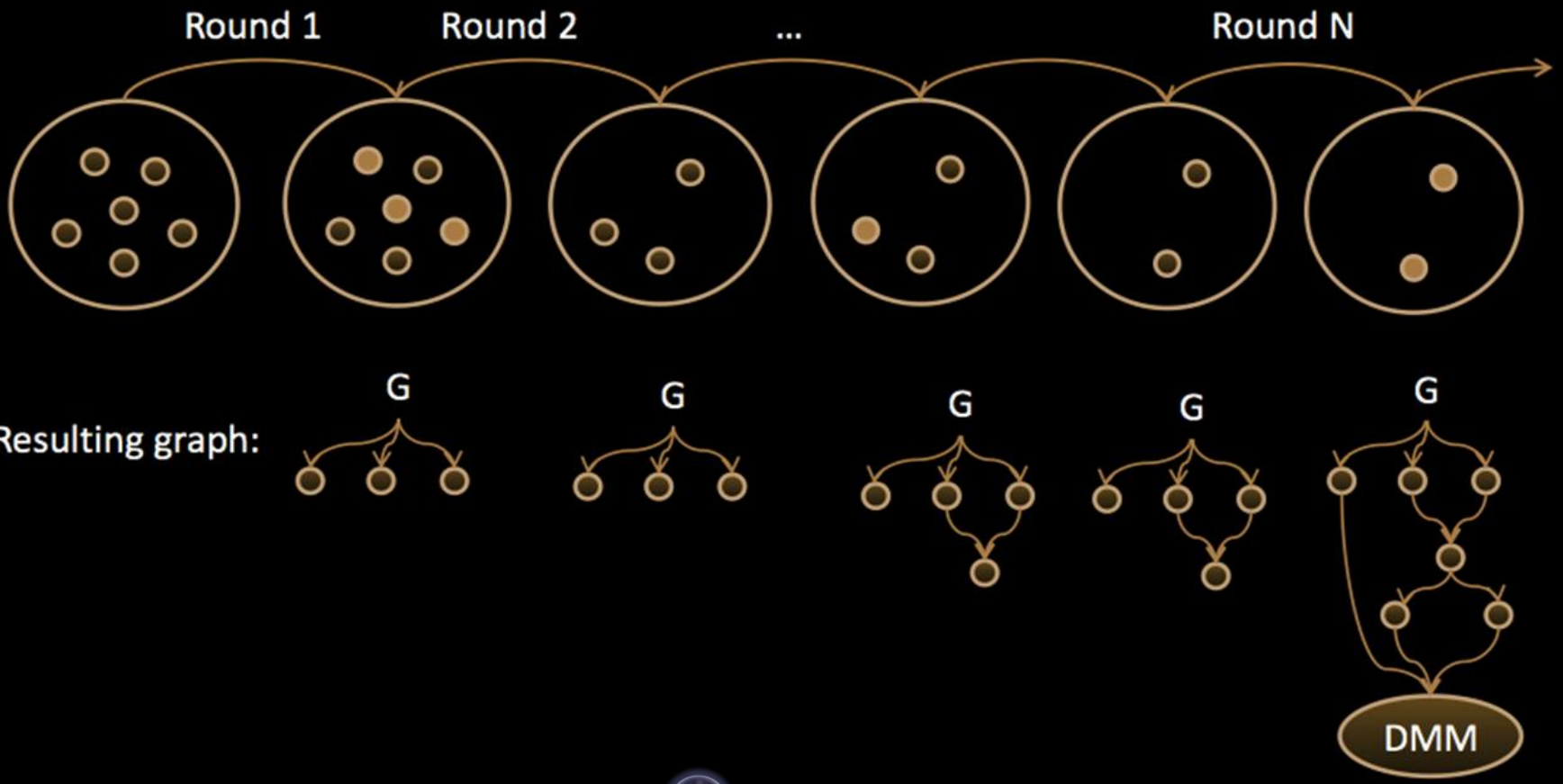


Topology example



Example of flow reducing

Building classifier



Selecting classifiers for the next layer

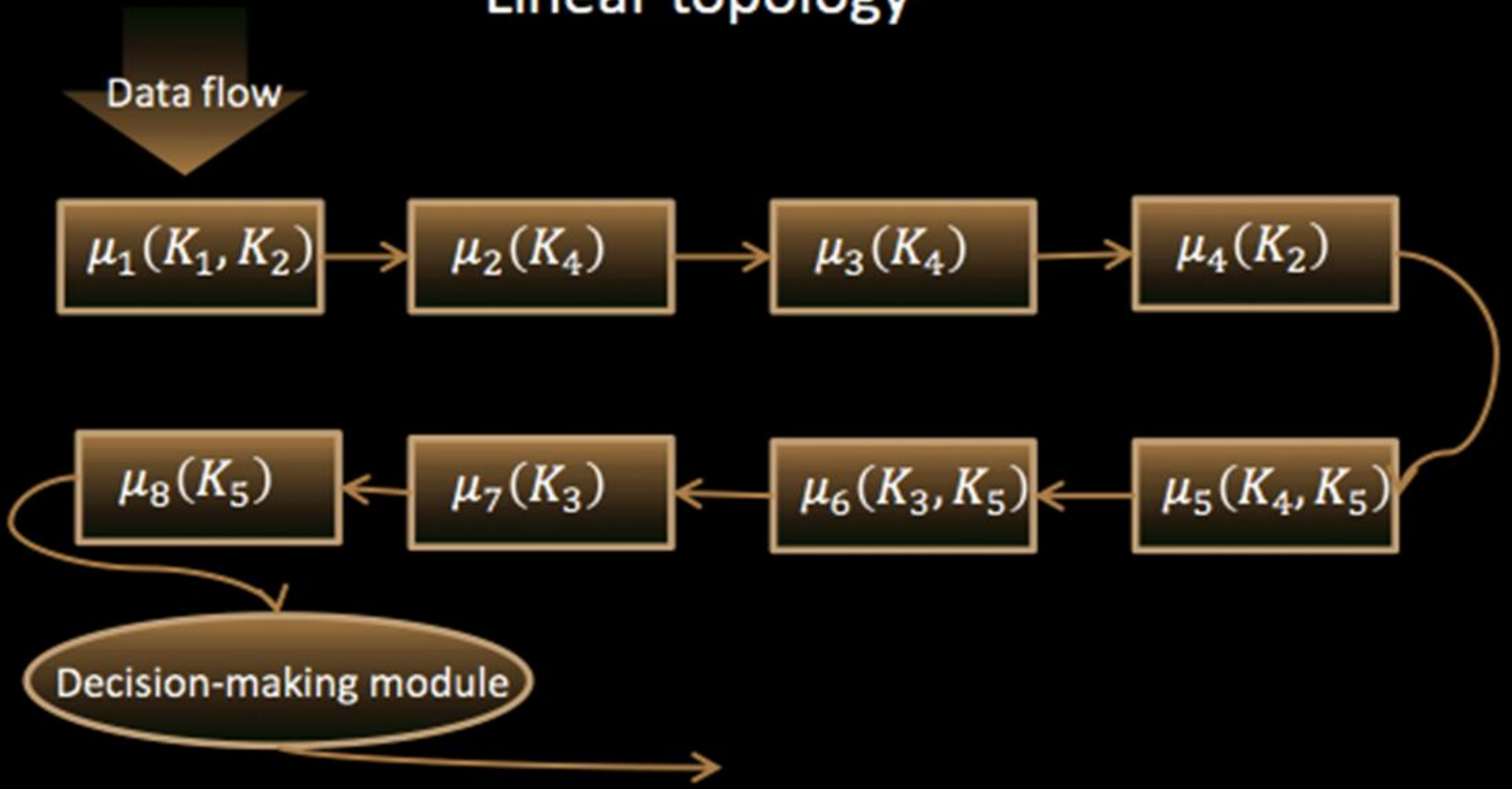
- Select different combination of classifier which provides complete coverage of shellcode classes

- Select combination, optimal in terms of FP and time complexity

Evaluation

Evaluation

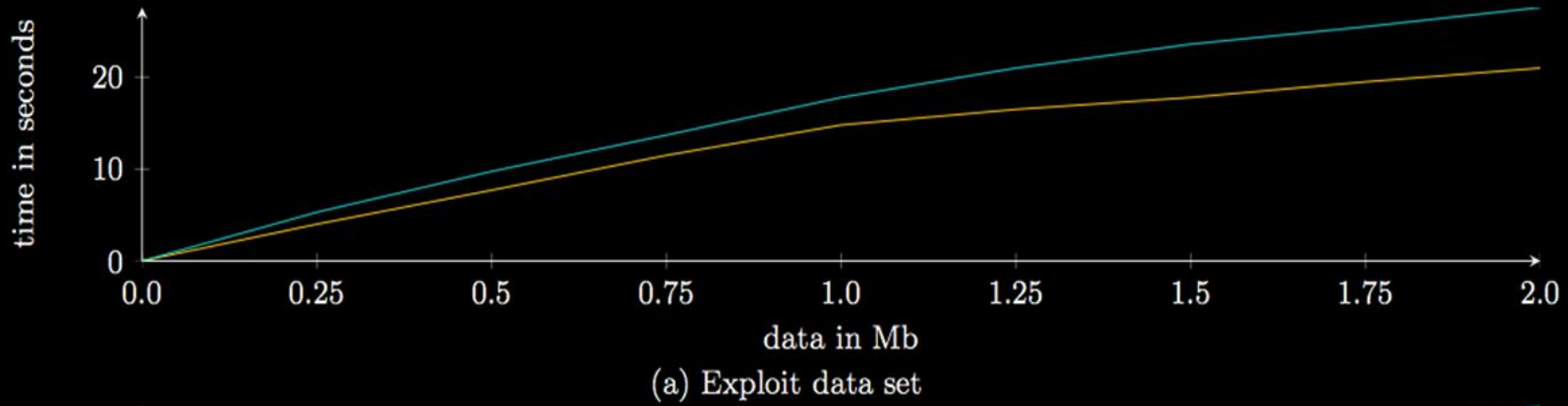
Linear topology



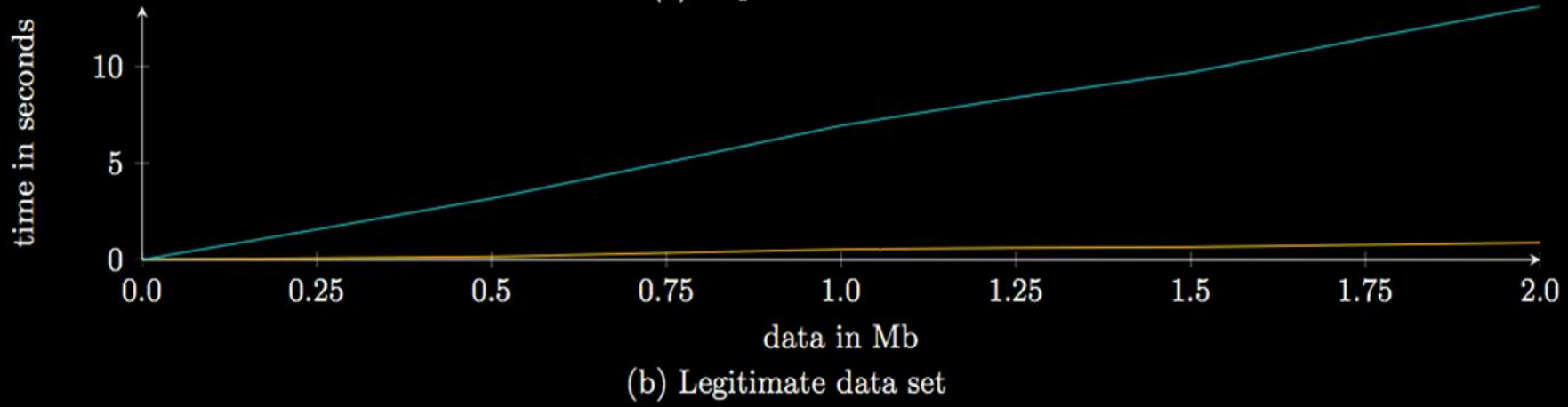
Evaluation: numbers

Data set	Linear			Hybrid		
	FN, *100%	FP, *100%	Throughput, Mb\sec	FN, *100%	FP, *100%	Throughput, Mb\sec
Exploits	0.2	n/a	0.069	0.2	n/a	0.11
Benign binaries	n/a	0.0064	0.15	n/a	0.019	2.36
Random data	n/a	0	0.11	n/a	0	3.7
Multimedia	n/a	0.005	0.08	n/a	0.04	3.62

Visualization of evaluation

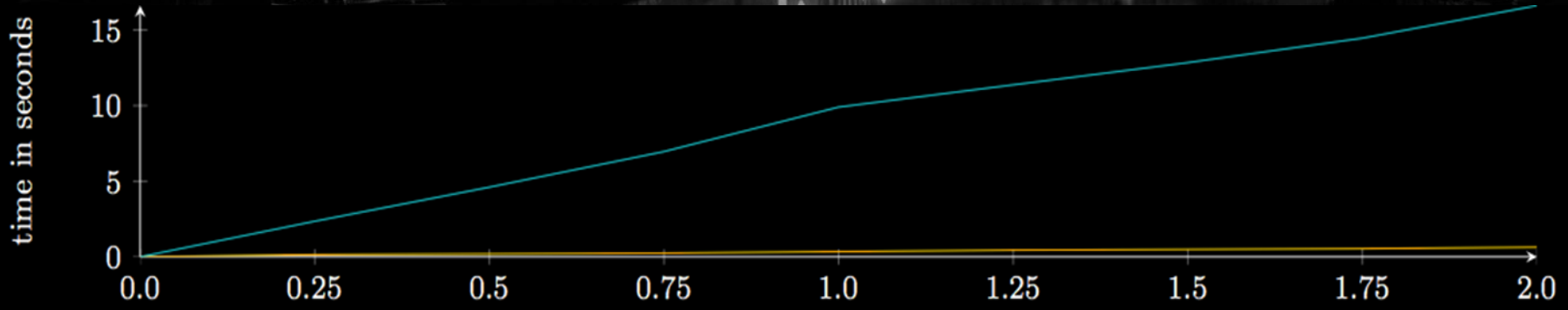


(a) Exploit data set

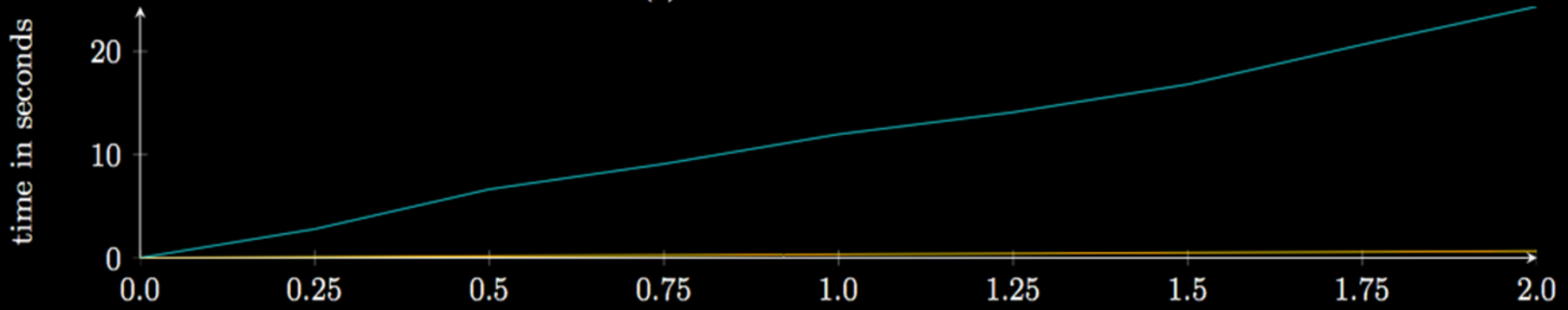


(b) Legitimate data set

Visualization of evaluation



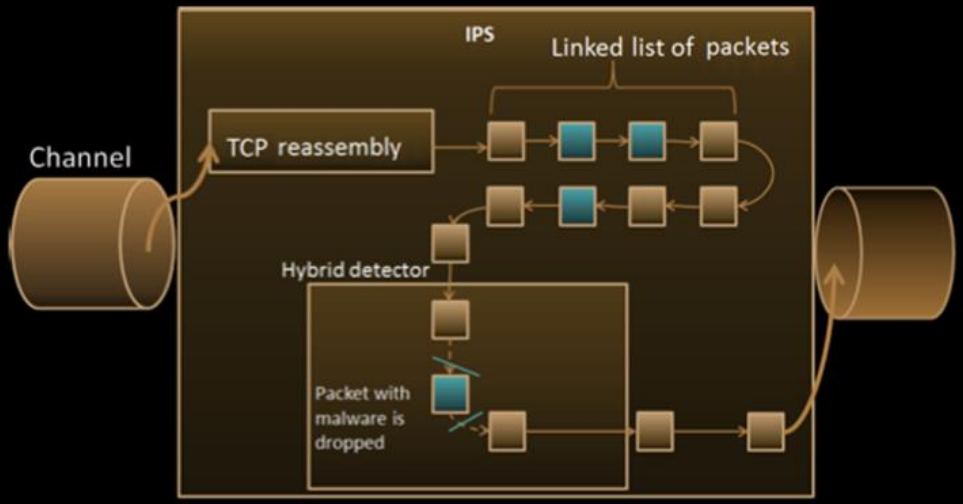
(c) Random data set



(d) Multimedia data set

Use-cases

- 0-days exploits detection and filtering at network level
- CTF participation experience



How does it work?

Where to find?

- Demorpheus

  https://gitorious.org/demorpheus

- Svetlana Gaivoronski
  - s.gaivoronski@gmail.com
  - GPG: 0xBF847B1F37E6E634
- Dennis Gamayunov
  - gamajun@cs.msu.su
  - GPG: 0xA642FA98


THERE ARE SOME
QUESTIONS THAT
GOOGLE
CANNOT ANSWER