Demorpheus: Getting Rid Of Polymorphic Shellcodes In Your Network

Svetlana Gaivoronski,
PhD Student, Moscow State University
Dennis Gamayunov,
Senior Researcher, Moscow State University

Memory corruption, 0-days, shellcodes…

Why should anyone care about shellcodes in 2012?

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 propagaion, and 0-days are less than 1%
- Endpoint security products deal with known malware quite well, why should we care about unkown?..

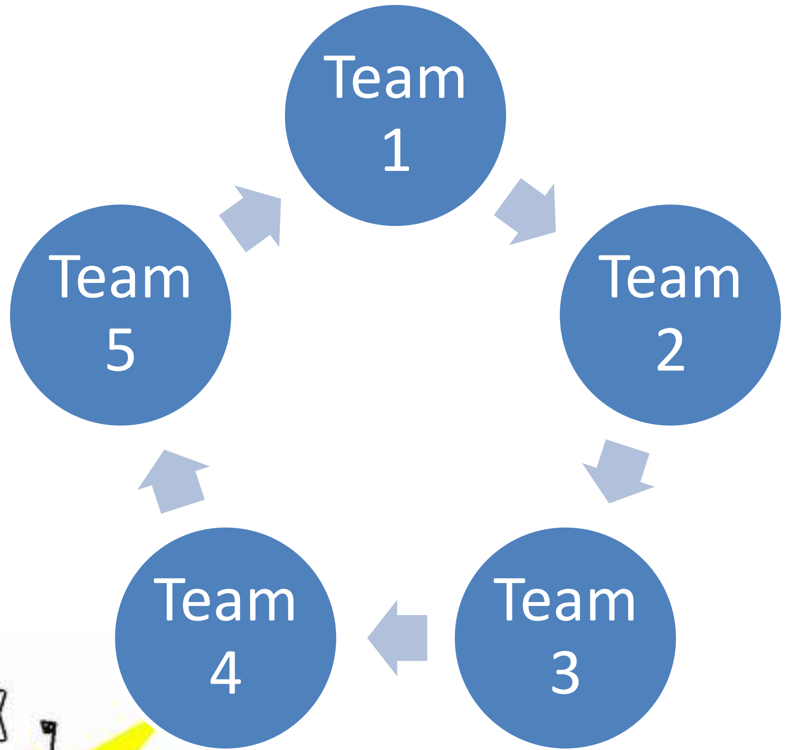
PROS:

- Memory corruption is still there ;-)
- Hey, Microsoft, we're all excited with MS12-020
- 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
- It's fun! ;)

* 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

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…



Another POV: Privacy and Trust in Digital Era

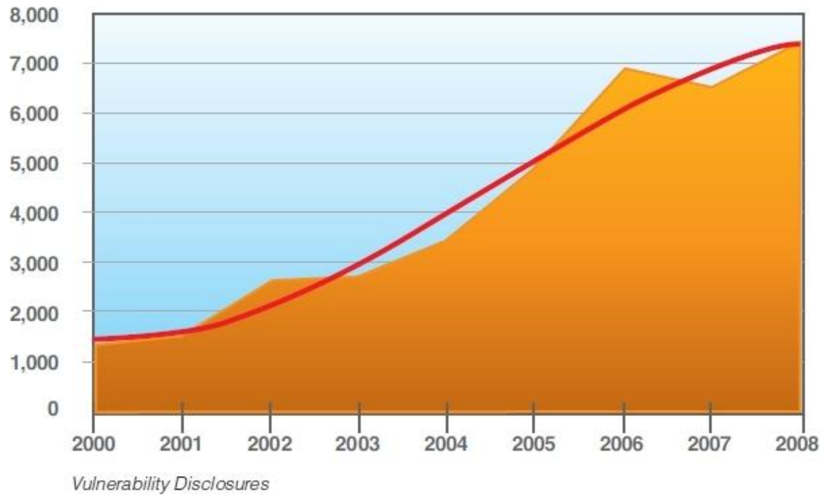
We share almost all aspects of our lives with digital devices (laptops, cellphones and so on) and Internet:

- Bank accounts
- Health records
- Personal information

Recent privacy issues with social networks and cloud providers:

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

Maybe the 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…

Shellcode detection: what do we have

Static analysis

- Signature matching
  - Content-dependent
  - Behavioral

  (*Hamsa[1], Polygraph[2]*)
- CFG\IFG analysis (*SigFree[3]*)
- NOP-sled detection *(RaceWalk[4])*
- APE *[5]*

Hybrid analysis

([9], PonyUnpack[10])

Dynamic analysis

- Emulation *([6], [7])*
- Automata analysis *(IGPSA[8])*

The problem: if we simply try to run all methods for each portion of data, it would be extremely slow

Virtues and shortcomings

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

And more:

- 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 real network channels in real-time

STAND BACK

I'M GOING TO TRY

SCIENCE

Why shellcode detection is feasible at all

NOP - sled

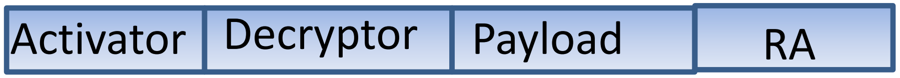
```
42                      inc     %edx
96                      xchg    %eax,%esi
f8                      clc
3c 48                   cmp     $0x48,%al
88 d5                   mov     %dl,%ch
90                      nop
97                      xchg    %eax,%edi
41                      inc     %ecx
35 93 98 24 92          xor     $0x92249893,%eax
4b                      dec     %ebx

d9 d0                   fnop
be 93 f2 c1 1e          mov     $0x1ec1f293,%esi
d9 74 24 f4             fnstenv -0xc(%esp)
5a                      pop     %edx
2b c9                   sub     %ecx,%ecx
b1 0a                   mov     $0xa,%cl
83 ea fc                sub     $0xfffffffc,%edx
31 72 13                xor     %esi,0x13(%edx)
03 e1                   add     %ecx,%esp
e1 23                   loope   0x3d

eb 9c                   jmp     0xffffffb8
6c                      insb    (%dx),%es:(%edi)
ab                      stos    %eax,%es:(%edi)
4c                      dec     %esp
cc                      int3
98                      cwtl
bf 6c f0 58 ef          mov     $0xef58f06c,%edi
09 84 3b c0 a2 0c dd    or      %eax,-0x22f35d40(%ebx,%edi,1)
7a 2a                   jp      0x59
bb 1d d8 dc f5          mov     $0xf5dcd81d,%ebx
1f                      pop     %ds
de 1c a0                ficomp  (%eax,%eiz,4)
d2 5e 76                rcrb    %cl,0x76(%esi)
53                      push    %ebx
b5 93                   mov     $0x93,%ch
07                      pop     %es
```

DECRYPTOR

ENCRYPTED PAYLOAD



As opposed to feature-rich viruses, shellcode has certain limitations in terms of size and structure

Proposed approach

- Given the set of shellcode detection algorithms, why don't we try to construct **optimal** data flow graph, so that:
  - the execution time and FP rate are optimized,
  - and the FN rate is not more than some given threshold

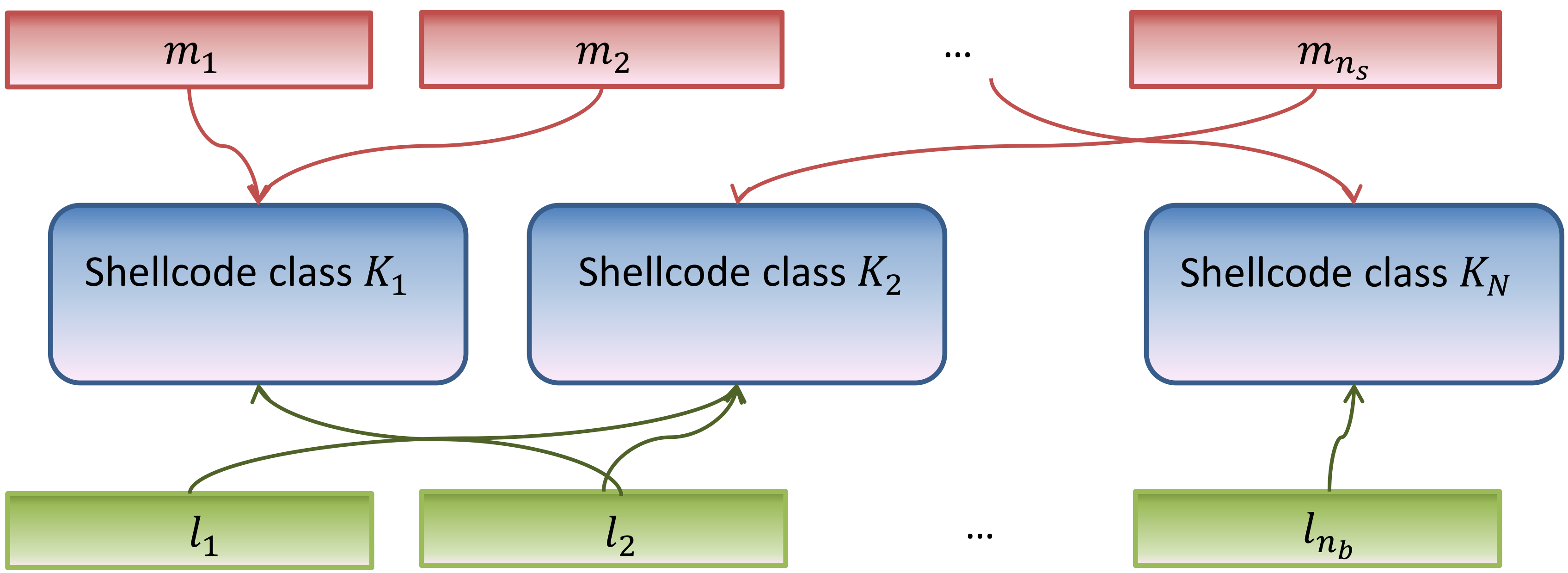
Shellcode features

	Generic features	Specific features
Static	Correct disassembly into chain at least K instruction	Correct disassembly from each and every offset. (NOP)
	Number of push-call patterns exceeds threshold	Conditional jumps to the lower address offset. (Encrypted shellcode)
	Overall shellcode size does not exceed threshold	Ret address lies within certain range of values. (non-ASLR systems)
	Operands of self-modifying and indirect jmp are initialized	MEL exceeds threshold. (NOP)
	Cleared IFG contains chain with more than N instructions	Presence of GetPC. (Encrypted shellcode)
		Last instruction in the chain ends with branch instruction with immediate or absolute addressing targeting lib call or valid interruption. (non-ASLR systems)
Dynamic	Number of near reads within payload exceed threshold R	Control at least once transferred from executed payload to previously written address. (non-self-contained shellcode)
	Number of unique writes to different memory location exceeds threshold W	Execution of wx-instruction exceeds threshold X (non-self-contained shellcode)

Shellcode classes

$SH = \{m_1, \ldots, m_{n_s}\}$- set of shellcode features, $BEN = \{l_1, \ldots, l_{n_b}\}$ - set of benign code features.

Shellcode space S is splitted for K classes with respect to identified shellcode features.



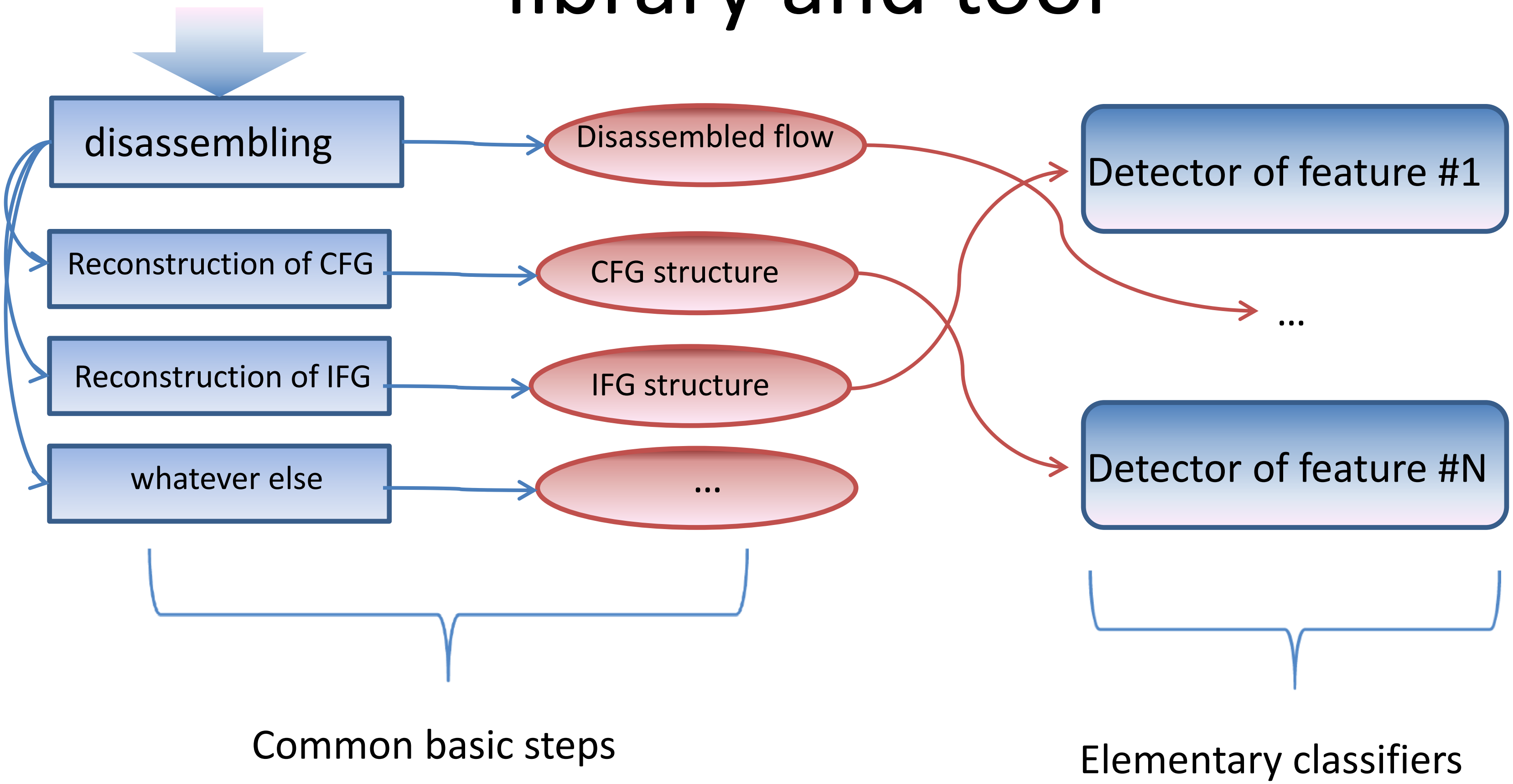
Shellcode classes. Example

K_{NOP_3} (contains multi-byte NOP-equivalent sled)	• Correct disassembly from each and every byte offset • Multi-byte instructions	Specific features
	• Correct disassembly into chain of at least K instructions • Overall size does not exceed certain threshold • …	Common features
K_{SC} (self-ciphered)	• Conditional jmps to the lower address offsets • …	Specific features

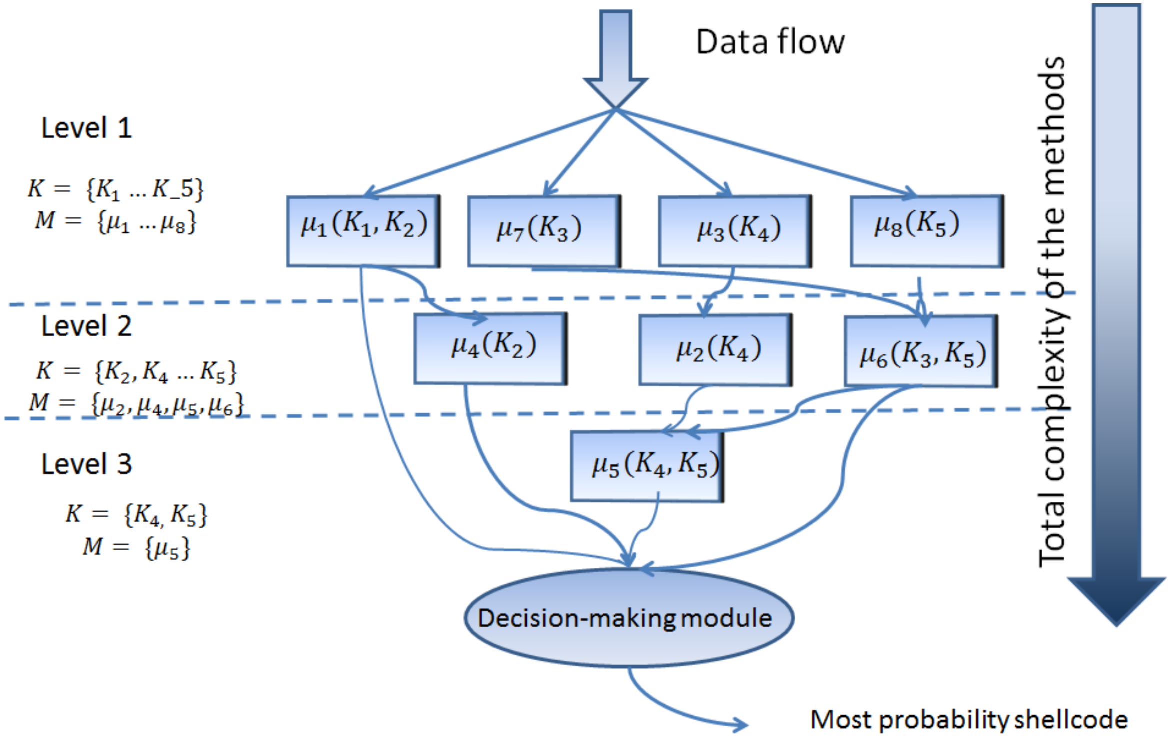
- Totally we identified 19 classes

NOTE: none of existing shellcode detection metods provides complete coverage of identified classes

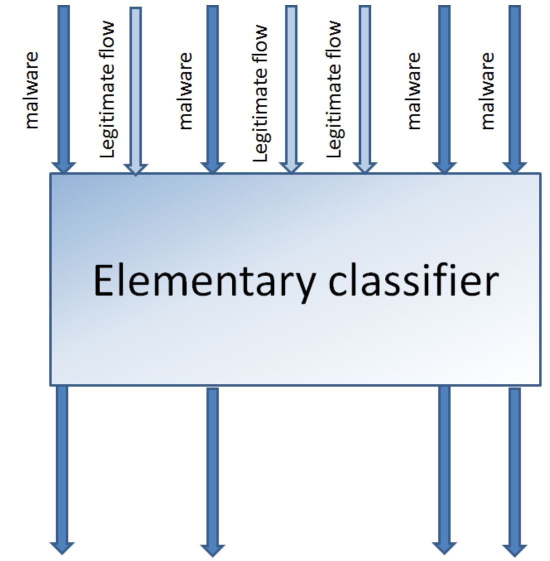
Demorpheus: shellcode detection library and tool



Hybrid shellcode detector

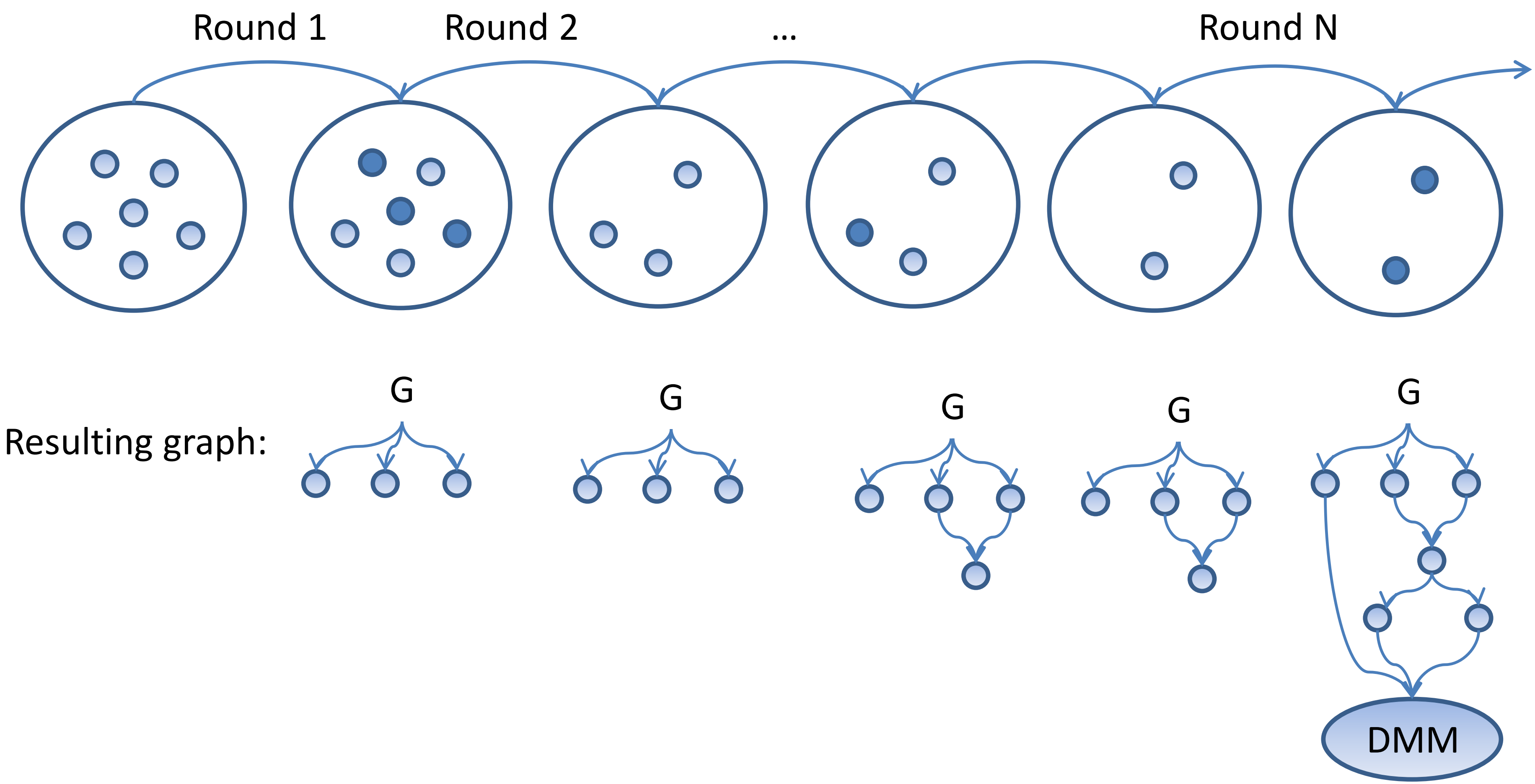


Topology example



Example of flow reducing

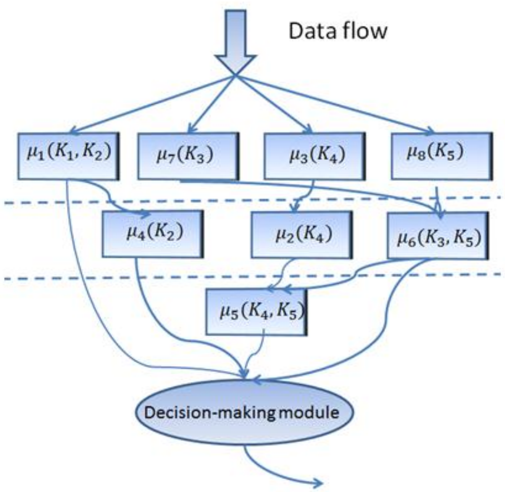
Building hybrid classifier



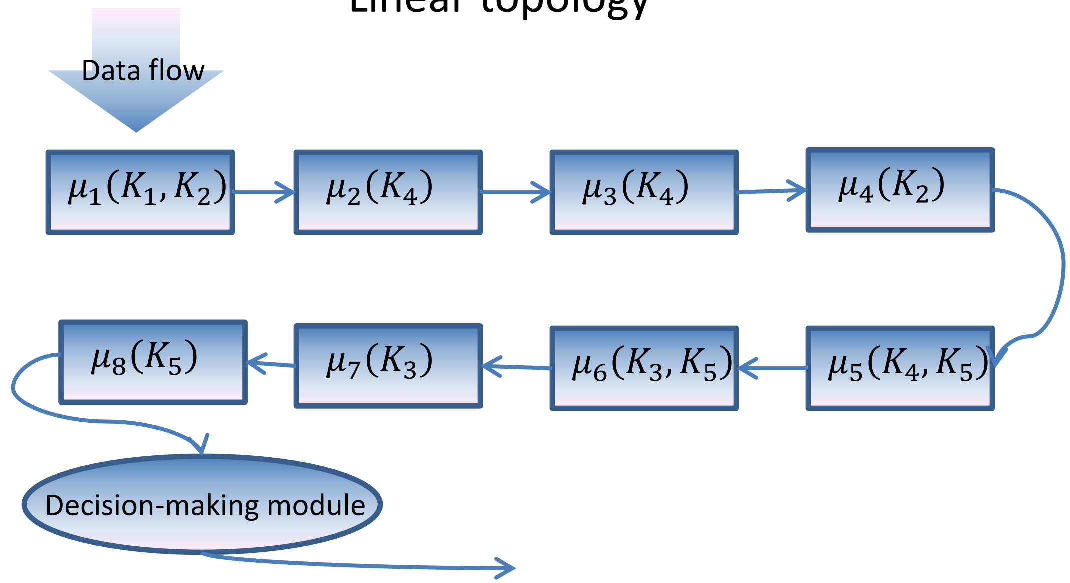
Evaluation: base line

One of the important goals - minimization of false positives rate

Hybrid topology



Linear topology

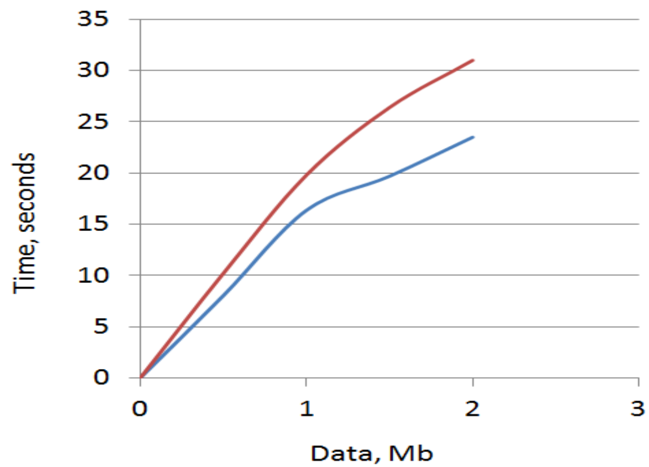


- Minimum false positives rate
- No flow reducing

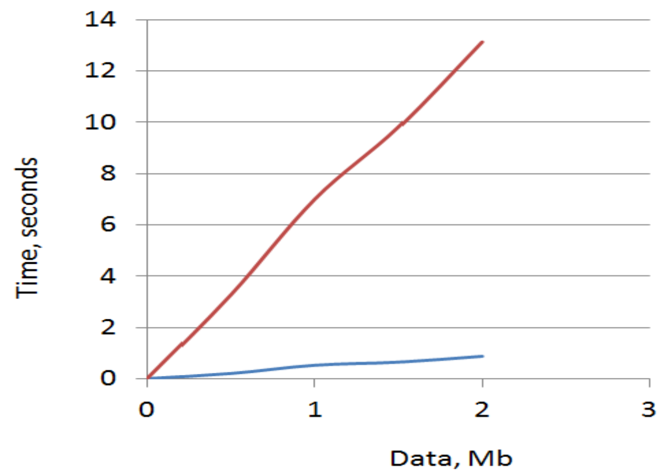
Experimental results: numbers

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

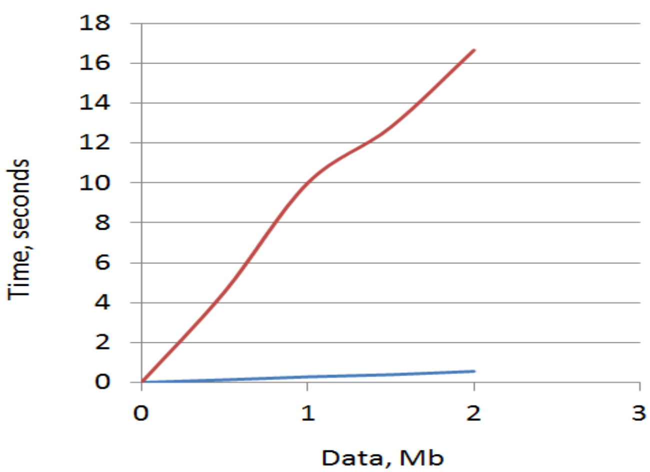
Experimental results: plots



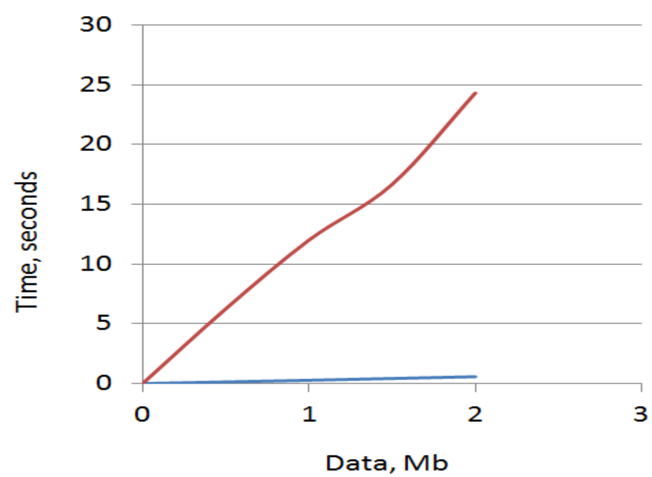
(a)



(b)



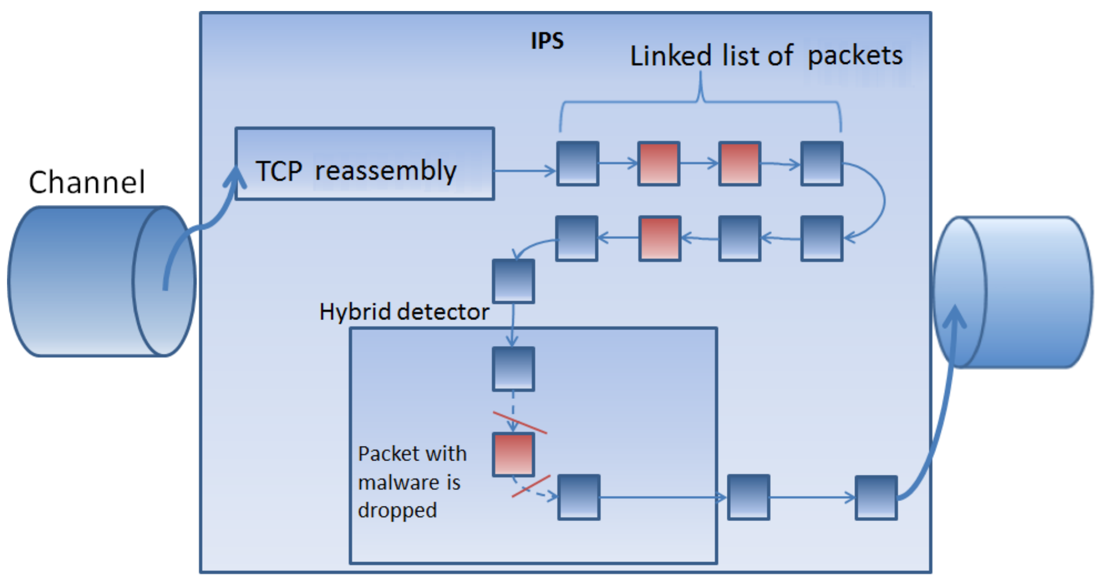
(c)



(d)

A couple of use-cases for hybrid classifier

- 0-days exploits detection and filtering at network level
- CTF participation experience:
  - could help to increase defense level of team
  - could help to gather ideas from other teams



Demonstration here

Conclusion

- Good news everyone!
- Shellcodes may be now detected up to 45 times faster than before
- You could download the Demorpheus source code from git@gitorious.org:demorpheus/demorpheus.git and integrate it with your own tool

Authors

- Svetlana Gaivoronski
  - E-mail: s.gaivoronski@gmail.com
  - Skype: svetik.sh
  - GPG key: 0x38428E3B
    16A2 2F9D 5930 7885 F9E5 9DA5 09ED D515 3842 8E3B
- Dennis Gamayunov
  - E-mail: gamajun@cs.msu.su
  - Skype: jama.dharma
  - GPG key: 0xA642FA98
    14E1 C637 4AEC BF0D 0572 D041 0622 7663 A642 FA98

References

1. Zhichun Li, Manan Sanghi, Yan Chen, Ming yang Kao, and Brian Chavez. *Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience.* In S&P06: Proceedings of the 2006 IEEE Symposium on Security and Privacy, pages 32-47. IEEE Computer Society, 2006.
2. James Newsome. *Polygraph: Automatically generating signatures for polymorphic worms.* In Proceedings of the IEEE Symposium on Security and Privacy, pages 226-241, 2005.
3. Xinran Wang, Chi-Chun Pan, Peng Liu, and Sencun Zhu. *Sigfree: A signature-free buffer overflow attack blocker.* IEEE Transactions On Dependable And Secure Computing, 7(1):65-79, 2010.
4. Dennis Gamayunov, Nguyen Thoi Minh Quan, Fedor Sakharov, and Edward Toroshchin. *Racewalk: Fast instruction frequency analysis and classiffication for shellcode detection in network flow*. In Proceedings of the 2009 European Conference on Computer Network Defense, EC2ND '09, pages 4-12, Washington, DC, USA, 2009. IEEE Computer Society.
5. Thomas Toth and Christopher Kruegel. *Accurate buffer overflow detection via abstract payload execution.* In In RAID, pages 274-291, 2002.
6. Michalis Polychronakis, Kostas G. Anagnostakis, and Evangelos P. Markatos. *Network-level polymorphic shellcode detection using emulation.* In In Proceedings of the GI/IEEE SIG SIDAR Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), pages 54-73, 2006.
7. Michalis Polychronakis, Kostas G. Anagnostakis, and Evangelos P. Markatos. *Emulation-based detection of non-self-contained polymorphic shellcode.*
8. Lanjia Wang, Hai-Xin Duan, and Xing Li. *Dynamic emulation based modeling and detection of polymorphic shellcode at the network level*. Science in China Series F: Information Sciences, 51(11):1883-1897, 2008.
9. Qinghua Zhang, Douglas S. Reeves, Peng Ning, and S. Purushothaman. *Analyzing network traffic to detect self-decrypting exploit code.* In In Proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIACCS), 2007.
10. Paul Royal, Mitch Halpin, David Dagon, Robert Edmonds, and Wenke Lee. Polyunpack: *Automating the hidden-code extraction of unpack-executing malware*. In Proceedings of the 22nd Annual Computer Security Applications Conference, pages 289-300, Washington, DC, USA, 2006. IEEE Computer Society.