On Emulation-Based NIDS

Ali Abbasi

Jos Wetzels, Wouter Bokslag

Emmanuele Zambon, Sandro Etalle

UNIVERSITY OF TWENTE.

History of Exploitation

- 1972: first paper introducing buffer overflows.
- 1988: the Morris worm uses a buffer overflow on the “finger” service.
- 1990: Washburn designs the first polymorphic virus.
- 2004: the Sasser worm propagates by means of shellcode exploiting a buffer overflow on MS LSASS.
- 2008: the Conficker worm propagates by means of a fixed byte XORed (***polymorphic***) shellcode.
- **Exploits become more complex over time.**

Shellcodes in Network Packets

- Everything evolves, shellcode makes no exception.
- In 1988 first shellcode used in Morris worm on a VAX system.
- In 2001 K2 introduces ADMmutate, a polymorphic engine to generate shellcodes.
- The ADMmutate primary goal was to evade an exploit signature.

Morris *finger* payload

```
pushl $68732f ’/sh\0’
pushl $6e69622f ’/
bin’
movl sp, r10
pushl $0
pushl $0
pushl r10
pushl $3
movl sp,ap
chmk $3b
```

Signature Based IDS

- Simple Exploit Code:

- Detection based on:
  - Return Addresses
  - NOP Instructions[1]
  - Shellcode signatures
  - Detecting polymorphic encoder signatures

1. Prabhu, Pratap, Yingbo Song, and Salvatore Stolfo. "Smashing the Stack with Hydra: The Many Heads of Advanced Polymorphic Shellcode." (2009).

Problems with Signature based intrusion detection

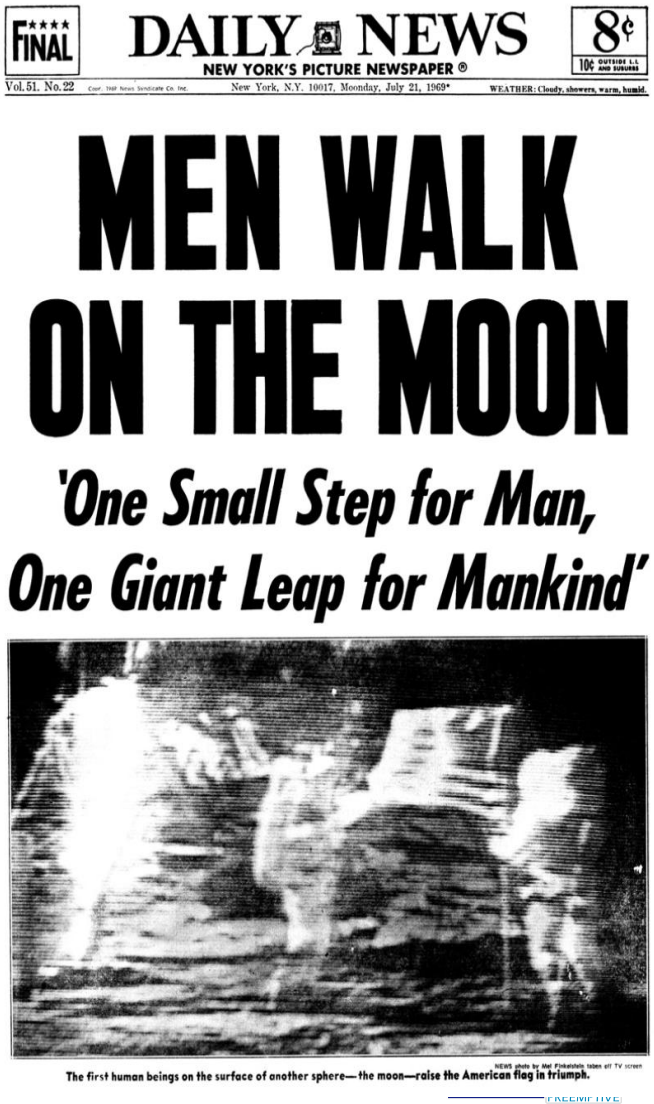
- Change a single byte of the payload to evade detection.
- Polymorphic shellcodes with custom encoders/decoders will evade detection.

- You must always update and maintain your signatures.

Emulation-Based NIDS, a Giant Leap

- Emulation-Based NIDSes emulate suspicious payloads.

- Meant to address the problem of **detecting polymorphic shellcodes.**
  - Detect polymorphic shellcodes regardless of which type of encoding technique is used.
  - Can detect 0-day shellcodes.
  - Do not rely on any specific vulnerability (signatures).
  - Use heuristics, a behavior blacklisting technique.


FINAL

DAILY NEWS

NEW YORK'S PICTURE NEWSPAPER ®

8¢

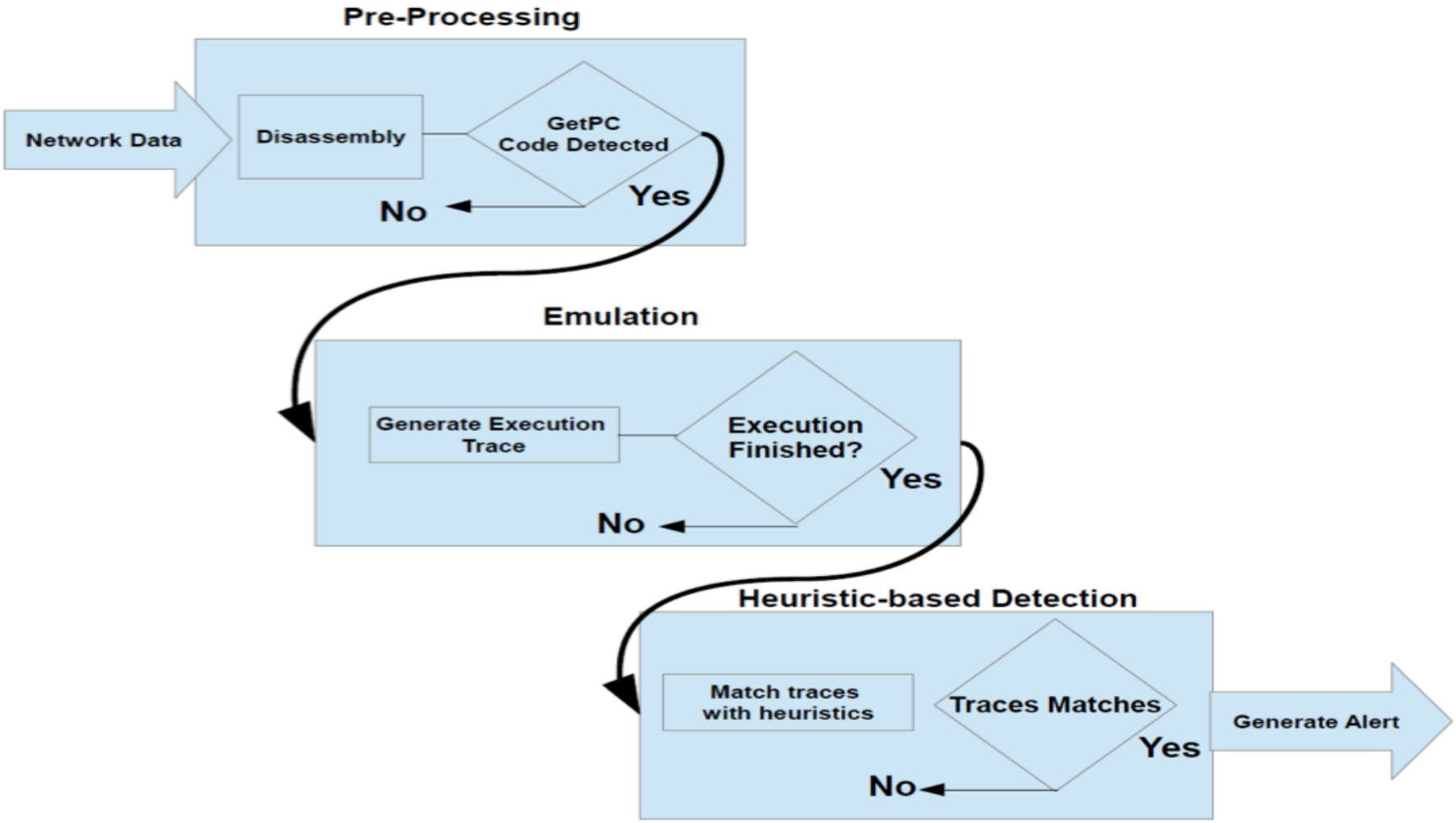
Vol. 51. No. 22 New York, N.Y. 10017, Moonday, July 21, 1969* WEATHER: Cloudy, showers, warm, humid.

MEN WALK ON THE MOON

'One Small Step for Man, One Giant Leap for Mankind'

The first human beings on the surface of another sphere— the moon—raise the American flag in triumph.

How Emulation-Based NIDSes Work?



The emulation-based approach is widely adopted

SGNET

libemu
x86 emu

i-code

Amun

Honeynet Project

NEMU

dionaea
catches bugs

UNIVERSITY OF TWENTE.

PREEMPTIVE

Is the problem solved?

- We believe not and we prove it.
- Emulation-Based NIDSes introduce an advanced and more complicated detection technology.
- In this work we show how attackers leverage this complexity to evade detection.

Emulation-Based NIDS Limitations

	Implementation	Intrinsic
Pre-Processing	✗	
Emulation	✗	✗
Heuristics	✗	

- Tested two Emulation-Based NIDSes: **Nemu** and **Libemu**.
  - Nemu is the state of the art in emulation based network intrusion detection because of its broad range of heuristics.
  - Libemu is a simple shellcode detection engine (used in several Honeynet projects).

Intrinsic limitations

- Unavailable context data
  - Emulation-based NIDSes cannot have a complete memory image of all possible targets.
  - Context keying.
  - Non-self contained shellcodes.
- Execution threshold
  - The emulator needs to stop at some point, the attacker can wait.
- Cannot deal with fragmented shellcode
  - Send the shellcode payload in multiple (non-consecutive) fragments.

Intrinsic limitations results

- Context keying
  - Modified version of the Context CPUID Metasploit key generator stub.
  - **Not detected.**
- Non-self contained shellcodes:
  - Dynamically built the entire GetPC code and the shellcode decoder out of ROP gadgets.
  - **Not detected.**
- Execution Threshold
  - Built shellcodes with four types of time-intensive loops.
  - Nemu could **detect half** of the shellcodes (loops were not taking enough time).
  - Libemu could **not detect any**.

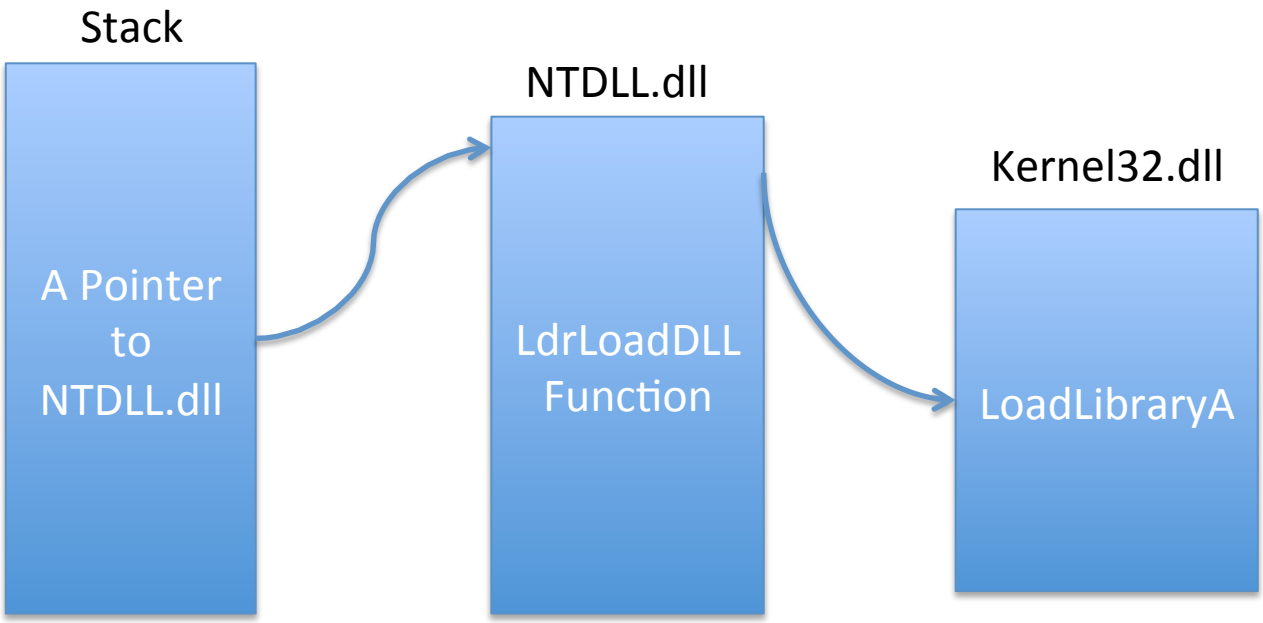
	Opaque loop	Intensive loop	Integrated loop	RDA
Nemu	9/9	9/9	0/9	0/9
Libemu	0/1	0/1	0/1	0/1

Implementation limitations

- Heuristics are kind of black listing
  - You have to list all possible shellcode behavior patterns, attackers can always find a missing one.
- Runtime difference (Emulator detection)
  - Shellcode can detect if it is being emulated.
- Unsupported instructions
- Detection relies on successful shellcode disassembly
  - Malware already applies anti-disassembly techniques to avoid analysis

Results of our implementation limitation tests

- Heuristics Detection
  - Five heuristics described. Two designed by us.
  - We could **bypass Libemu and Nemu** heuristics.
  - Example: bypassing the Nemu LoadLibraryA Kernel32.dll heuristic.



Results of our implementation limitation tests

- Emulator detection
  - Built shellcode that stop execution if instructions are executed "too slowly".
  - **Not detected.**

Run time in milliseconds	Opaque Instructions	Intensive Loops	Integrated Loops	RDA
Nemu	6.8	9.08	37.81	52.90
Libemu	44.07	75.20	173.49	177.56
Native	0.148	2.10	0.30	0.68

- Unsupported Instructions
  - Built shellcode using SSE, MMX, FPU and undocumented instructions in the decoder.
  - Both Nemu and libemu **failed to detect** the shellcodes when we used FNSAVE ,MMX, SSE and undocumented instructions.

	FPU (FNSTENV)	FPU (FNSAVE)	MMX	SSE	OBSOL
Nemu	9/9	0/9	0/9	0/9	0/9
Libemu	1/1	0/1	0/1	0/1	0/1

Results of our implementation limitation tests

- Anti-disassembly
  - Built shellcode using 4 different types of instruction patterns to fool the disassembler.
  - Libemu could detect 2 types out of 4.
  - Nemu could detect most anti-disassembly techniques.

	Garbage Byte	Flow Redirect	Push/Pop Math	Code Transposition
Nemu	9/9	9/9	8/9	8/9
Libemu	0/1	1/1	0/1	1/1

Conclusion

- Emulation based Network Intrusion Detection System is a great step forward but still there are problems:
  - Unfortunately, there seems to be **no (easy) solution** for some of those problems.
  - Most of the **evasion techniques are easy to implement**.
  - Some of them are **already in use** by some malware.
- At least implementation limitations should get fixed.
  - We fully disclosed the evasions we used with people who created the emulation based NIDSes.

Questions?

This work has been partially supported by the European Commission through project FP7-SEC-607093-PREEMPTIVE funded by the 7th Framework Program.

UNIVERSITY OF TWENTE.