symantec™

Confidence in a connected world.

A Study of the Packer Problem and Its Solutions

Fanglu Guo, Peter Ferrie, and Tzi-cker Chiueh

09/16/2008

The Problem

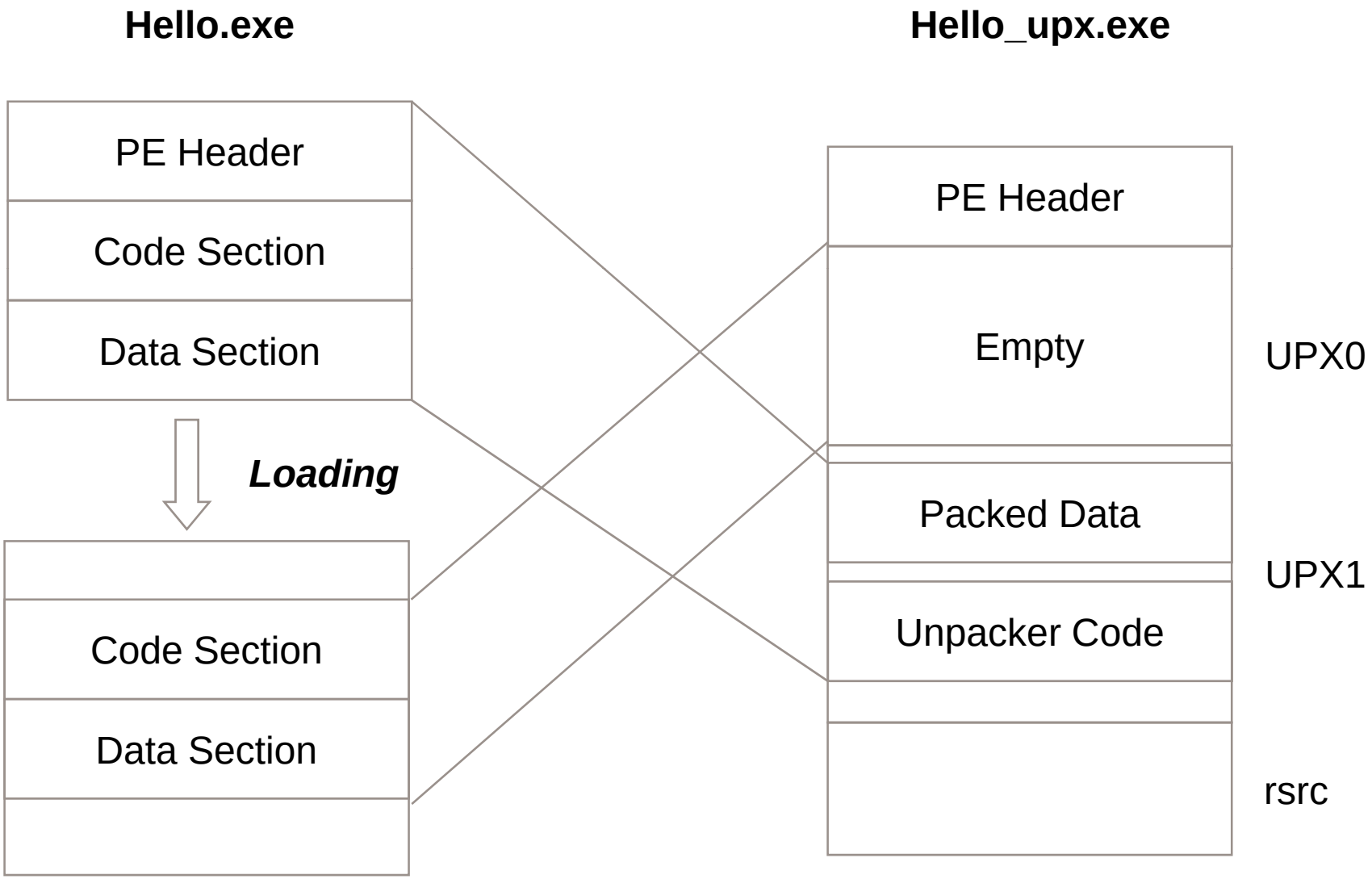
- Most malicious programs are packed by a class of programs called packers, in order to evade signature-based AV (Anti-Virus) detection
- Packers
  - Programs that transform an executable's appearance without affecting its execution semantics
- Results
  - Low detection rate (30-70%) across the entire AV industry (BlackHat 06)
  - The number of AV signatures increases

The Traditional Solution

- Reverse-engineer a packer manually
- Write recognizer code to identify if a program is packed by a specific version of a specific packer
- Write unpacker code to restore the packed program
- The problem – It is hard to keep up with the development of packers

	Known Packers	SYMC Identifiable	SYMC Unpackable
Number of Families	200	150	110
Number of Packers	2000	1200	800

Packer Background



Desirable Features of Unpacking Solutions

- Effectiveness – Restore the programs to their original form
- Generic – Restore as many different programs packed by different packers as possible
- Safe – Restore the programs without harming the system
- Portability – Run on different operating systems

Existing Solutions

- The traditional solution – Recognizer and unpacker
  - Largely effective, safe, and portable. Not generic
- Emulator – Run a packed program inside an emulator and use the memory image
  - Generic, safe, and portable. Not that effective
  - High fidelity is difficult to achieve
  - Memory image can change during the course of emulation
- Memory image of a running process
  - Generic. Not safe, portable, or that effective
- Execution of dirty memory – PolyUnpack, Renovo, OllyBone, Saffron, OmniUnpack, Justin
  - Effective, generic. Not safe or portable

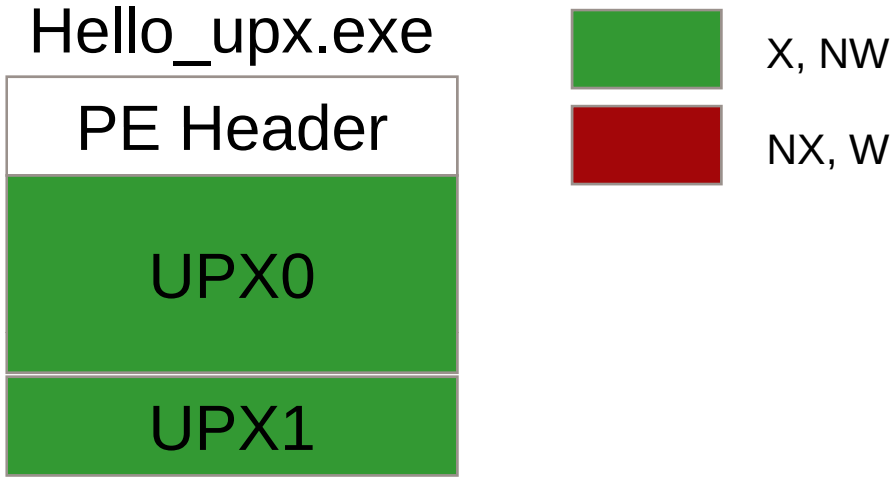
Existing Solutions (2)

- PolyUnpack
  - Disassemble and single step to check if the current instruction sequence is in the original program
- Renovo
  - Instrument instructions to track memory write and execution of dirty memory
- OllyBone and Saffron
  - Exploit x86 separate data TLB and instruction TLB to track execution of dirty pages
- OmniUnpack
  - Based on OllyBone. Defer AV detection at the “dangerious” system call

Justin

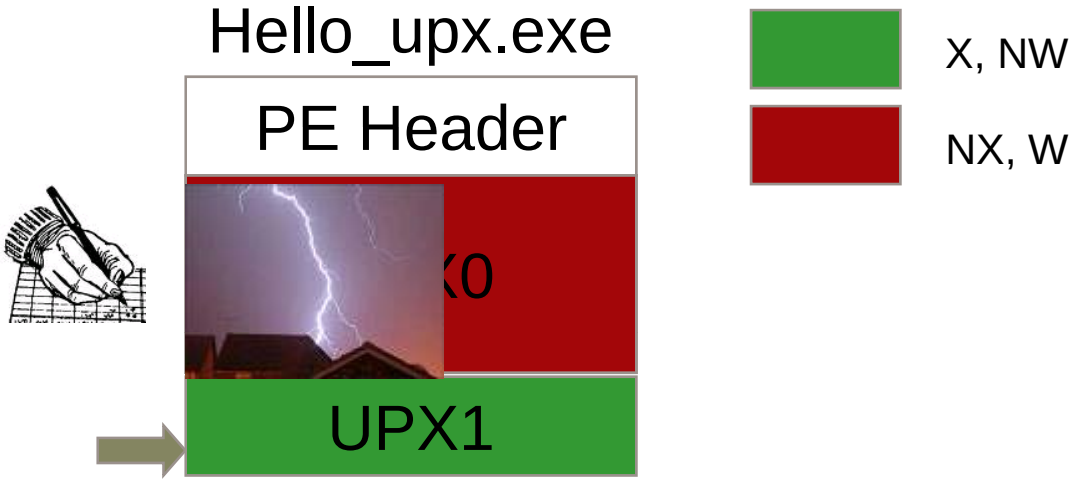
- Justin – Just-In-Time AV scanning
  - Use CPU NX (No eXecute) and Non-writable page protection to track execution of dirty page
  - Countermeasures against anti-unpacking
  - Heuristics to detect the end of unpacking

How Justin Works



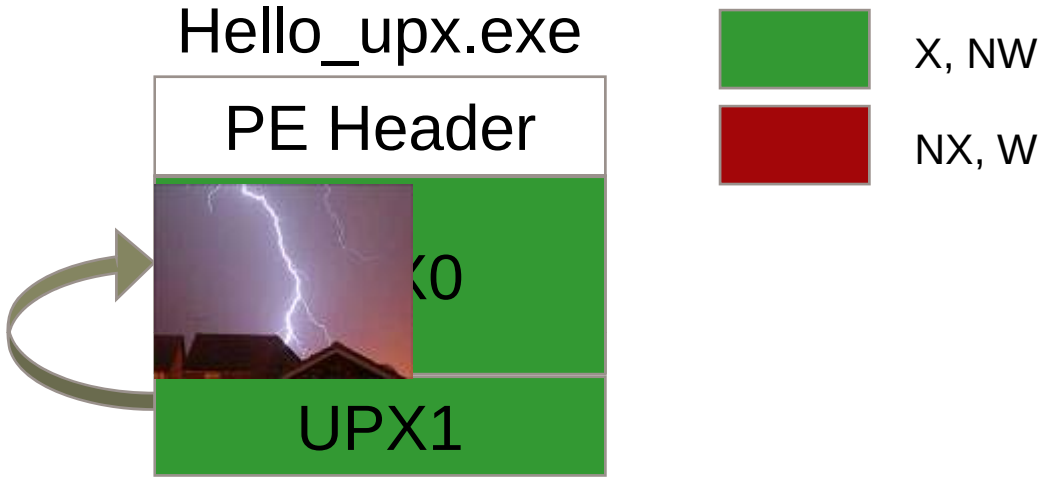
- Initialization
  - If any page is both executable (X) and writable (W), set it to be executable but not writable (X, NW)
- At run time
  - When write exception happens, change the page to be writeable but not executable (NX, W)
  - When execution exception happens, change the page to be executable but not writable (X, NW)

How Justin Works



- Unpacker code derives the original program from packed data
- Writes it to pages in UPX0
- VM hardware automatically detects a write exception and the OS delivers this exception to the user process
- Justin catches this exception, changes the page to be writable but non-executable, and resumes the execution of the unpacker code

How Justin Works



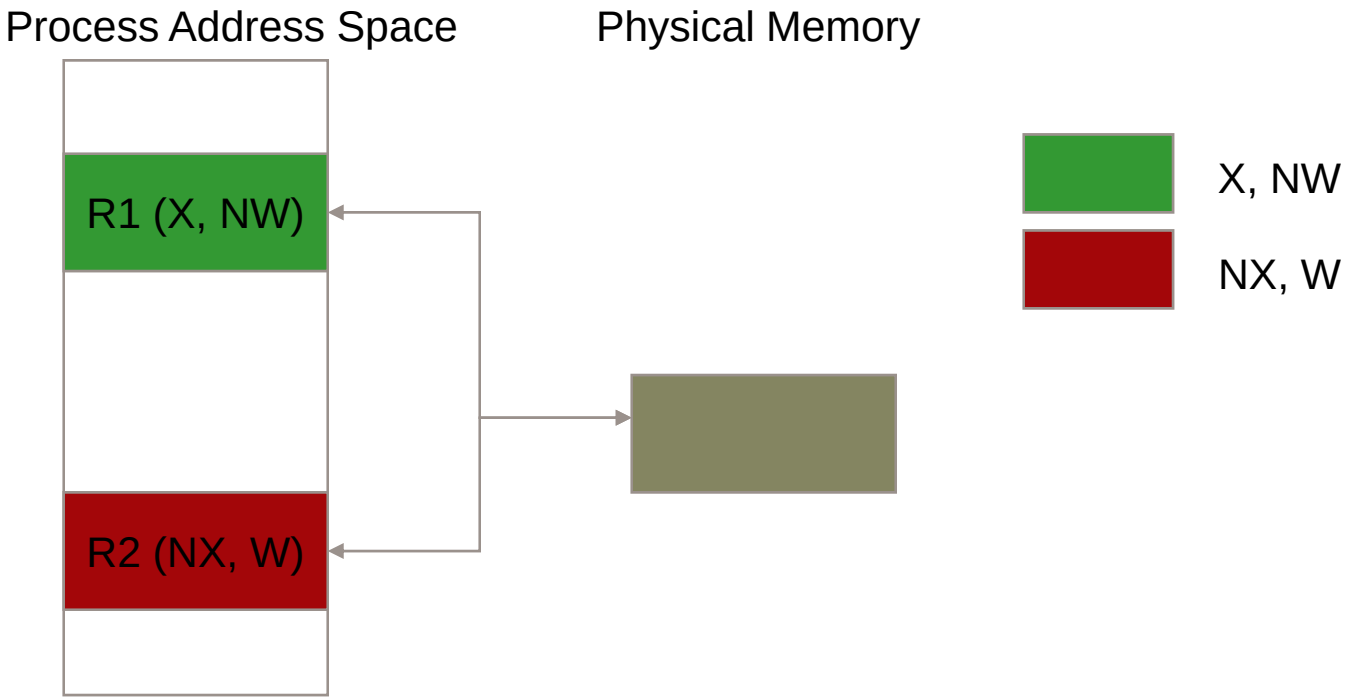
- After restoring the original program, unpacker code jumps to the entry point (the first instruction) of the original program
- VM hardware detects an execution exception and the OS delivers this exception to the user process
- Justin catches this exception and calls AV scanner to scan the address space snapshot at this point
- If scanner detects malware, Justin terminates the process; otherwise, Justin changes pages to executable and not-writable and resumes

Summary

- Just-In-Time AV scanning
  - Just before the original program is going to run, it will be scanned to decide if it is malicious
- High fidelity
  - Programs run in native environment
- High performance
  - No code instrumentation/disassembly

Countermeasures Against Anti-unpacking

- Memory-Mapped Files (shared memory)
  - Write at R2 then execute at R1
- Solution
  - Track **MapViewOfFile(Ex)**



Countermeasures Against Anti-unpacking

- Same page execution and write
  - A deadlock!
  - Write violation → Execution violation
- Solution
  - Change the page to be writable and executable
  - Single-step the instruction and change the page's protection bits back to their original state

Countermeasures Against Anti-unpacking

- What if the program wants to change page protection?
  - Track **VirtualProtect(Ex)** and filter/record program requests

Program Request	Justin Action	Reason
X, W	NX, W. Record program intention	Violate mutual exclusion of X and W
X, NW	NX, NW. Record program intention	Otherwise Justin fails to catch original program execution
NX, W	No change	Not a threat
NX, NW	No change	Not a threat

Countermeasures Against Anti-unpacking

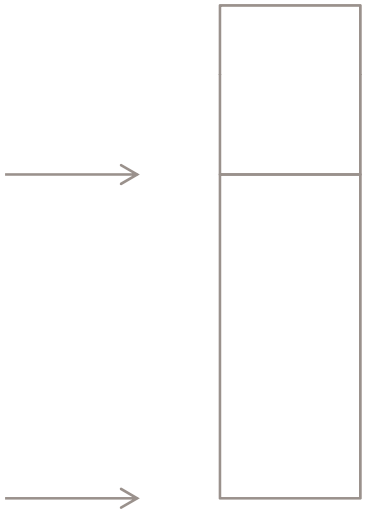
- Exceptions are mixed together
  - Exceptions are extensively used in packers to make reverse engineering difficult
  - Justin also uses exceptions to scan memory and change page protections
  - These two kinds of exceptions are mixed together
- Solution
  - Dispatch exceptions based on program's original intention
  - If the original program's intention should not cause memory access violation, Justin handles the exception
  - Otherwise, Justin passes the exception to the original program
  - E.g., If a page is set to be non-writable by the program, write violation exception will be passed to the program. If the page is set to be writable but write violation happens, Justin handles it.

Countermeasures Against Anti-unpacking

- Execute the original program in a separate process
  - Generate the original program in a file and execute that file
    - Solution
      - Real-time AV scanning will detect it
  - Write the original program to another process and resume the process
    - Solution
      - When write to the address space of the other process, the process's address space is not executable
      - Enforce Justin logic in all processes

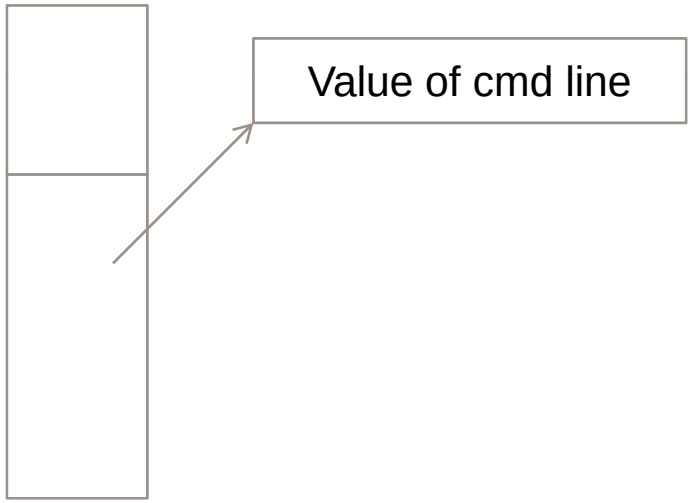
Detect the End of Unpacking

- Stack Pointer-based detection



Detect the End of Unpacking

- Command line argument-based detection



Evaluations – Effectiveness

- Results
  - Can unpack as long as the program can run and do unpacking
  - Better output for AV scanning

Packers	Junstin Unpack Failure	Justin Detection Failure	Justin Improvement Over SymPack
ASPack	4	0	-4
BeroPacker	0	4	13
Exe32Pack	32	0	-32
Mew	1	8	0

Evaluations – Efficiency

- Results
  - Unpacker memory avoidance greatly decreases the number of exceptions
  - Stack pointer
  - Command line argument

Packers	Dirty Page Execution	Unpacker Memory Avoidance	Stack Pointer	Command Line Argument
ASPack	96	12	2	3
ASProtect	1633	12	12	3
UPack	442084	12	2	3

Qestions?

Thank You!

Fanglu Guo

Fanglu_Guo@symantec.com

424-750-7486