# Semi-Automatic Unpacking on IA-32 Using OllyBonE

Joe Stewart

# Introduction

- Packers – so many to reverse-engineer, so little time!
- We've got a lot of malware to analyze, we need to spend our time looking at the malicious code, not the packer
- Let's take a look at how it's being done now...

# The Hard Way

- Painstakingly reverse-engineer the packer code
- Write an unpacking engine to handle the specific algorithms/tricks of the code
- Who has time for this? Even AV companies have a hard time keeping up with every version of every packer
- Even if they could, the scanner  engine gets increasingly bloated

# The Expensive Way

- Buy (or write) an emulation engine which can pretend to execute the code and unpack it along the way
- Then you only have to deal with minor variations/tricks in the code
- A lot of time to write, and even more time to maintain (therefore usually not free)

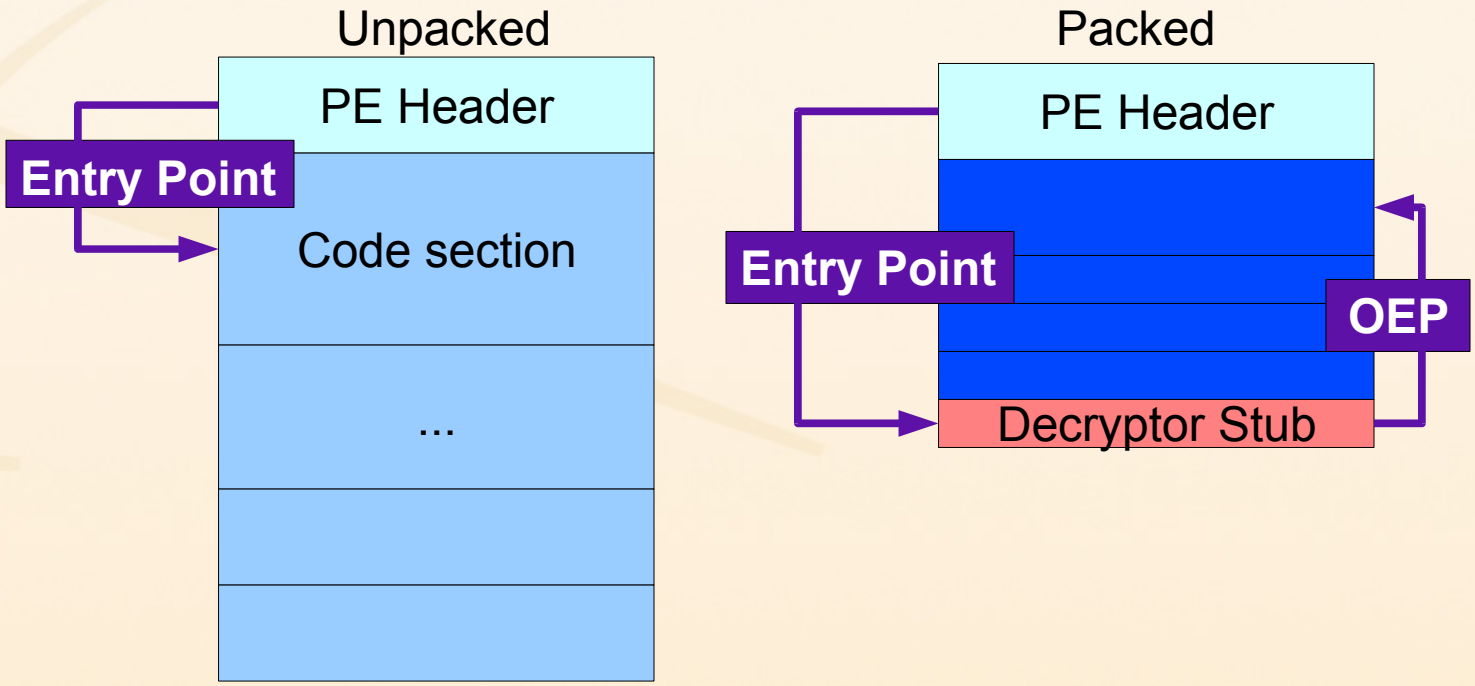
# Cheating

- Just run the code on a goat system and dump it from memory after it's unpacked
- Doesn't give us a clean starting image – variables in memory which have changed since the start of execution are now dumped at their current value
- Where is the OEP?

# Simplifying it all

- Most (not all) unpacking code works the same way from a high-level view
- Code is packed/encrypted, and a stub section is added to the end
- The EntryPoint in the PE header now points to the stub
- Unpacking code runs, unpacks the other sections, then jumps to the code section

# Most common packing method



# Your wishlist

- How many times have you wished for a way to set a breakpoint in OllyDbg on an entire section of code?
- Well, of course you can already do this – but this is break-on-access
  - The stub code has to write to the section you're going to execute
  - So you'd be breaking thousands of times before it executed
  - Might as well trace it if you're going to do that

# Tracing

- What's wrong with tracing?
  - Oldest trick in the book
  - Trivially detected
  - Can be slooooowwwww...
- So, you say, “I wish I had a simple way to just break on execution of a memory section, without all that tracing...”

# OllyBonE

- OllyBonE = Break on Execute for Olly!
- How is this possible? X86 architecture at least doesn't allow for NX without a special CPU
- But wait, this problem was already solved for stack/heap overflows, by the PaX project
- So, we adapt the idea of protecting stack/heap to protecting arbitrary pages of memory

# PaX Review

- We all know how PaX works, right?
- VA translation lookaside buffers
- x86 architecture uses separate TLBs
  - DTLB for read/write
  - ITLB for execution
  - We can cache one and not the other – let the OS read the stack, but kill process if it tries to execute it
  - Marks pages by overloading the meaning of the user/supervisor PTE bit

# Making it work as an unpacker

- Instead of protecting the stack/heap from execution, we protect all pages of a target PE section in memory
- Instead of killing the process on execution attempt, we jump from the page fault handler to the INT1 handler
- This raises a single-step exception inside OllyDbg – returning control to us

# OllyBonE Architecture

- OllyBonE.dll - OllyDbg plugin
- ollybone.sys – kernel driver, implements arbitrary PaX-like page protection
- OllyBonE interfaces with ollybone.sys via IOCTL, tells it what page of virtual memory to protect or un-protect

# Internals

- Data access
  - Unpacking program attempts to write to target section
  - VA translation is not already cached; page table walk generates page fault
  - Our page fault handler checks to see if page fault is our fault
  - If so, check to see if this is a data access (faulting address != EIP)
  - If so, toggle the PTE bit, then read from the page in order to cache the DTLB entry
  - Toggle the PTE entry back to original state

# Internals 2

- Instruction (execute) access
  - Unpacking program attempts to execute code in target section
  - VA translation is not already cached; page table walk generates page fault
  - Our page fault handler checks to see if page fault is our fault
  - If so, check to see if execute access (faulting address == EIP)
  - If so, pop extra argument off the stack and jump to INT1 handler

# Virtual Machines

- Virtual machines don't always correctly implement the IA-32 TLBs
- VMs that can run OllyBone:
- Known to work:
  - VMWare
- Doesn't work:
  - Bochs
  - Qemu
- Unknown:
  - Microsoft Virtual PC

# Using OllyBonE

- Usage is straightforward
  - Load target EXE in OllyDbg
  - Locate potential final code segment
  - Toggle break-on-execute flag
  - Run
  - Program encounters INT1 (single-step break) when trying to execute protected page
  - Control is passed back to OllyDbg
  - We are at the OEP, unpacked (hopefully)

# Demonstration

- Video demo
  - Brought to you by xvidcap & Cinelerra-CV
    - Cinelerra-CV needs more developers!
      - Help out at http://cvs.cinelerra.org/
  - Special thanks go to Piotr Bania for providing a copy of his packed sample library to demonstrate on
    - http://www.piotrbania.com/

# Pitfalls

- Problems with running code under debugger are not solved
- So packers with anti-debugging code will have to be thwarted in other ways
- Packers which dynamically unpack code as the program is run will not fall prey to this type of attack
- But there aren't many of those

# Evasion

- Once the packer author knows what we are doing, they can change the code to work differently
- For example, appending the stub code as part of the code section, instead of a whole new section
  - But, we could still get finer-grained with our page protection

# Evasion 2

- Packer could detect ollybone.sys in the loaded drivers list or even send its own IOCTL to un-protect the code section
- Use the source, Luke!
  - Change the naming convention and IOCTL numbers and recompile
- Affecting memory permissions via VirtualProtect?
  - May need to maintain marker bit during execution, or hook VirtualProtect

# Code

- You can download OllyBonE right now:
  - http://www.joestewart.org/ollybone/
- Code is released under the GNU GPL
- TODO:
  - Implement break-on-execute for heap/stack locations
  - Implement break-on-execute for shared DLL memory sections
    - Force Copy-on-Write, then set BoE?

# Q & A

Joe Stewart
<jstewart@lurhq.com>
Senior Security Researcher
LURHQ