



# Malware Software Armoring Circumvention

**Shmocon 2008**

**Offensive Computing, LLC**

Danny Quist

Valsmith

[dquist@offensivecomputing.net](mailto:dquist@offensivecomputing.net)

[valsmith@offensivecomputing.net](mailto:valsmith@offensivecomputing.net)



# Danny Quist

- Offensive Computing, Cofounder
- Ph.D. Student at New Mexico Tech
- Reverse Engineer
- Exploit Development
- cDc/NSF



# Valsmith

- Offensive Computing, Cofounder
- Malware Analyst/Reverse Engineer
- Metasploit Contributor
- Penetration Tester/Exploit developer
- cDc/NSF



# Offensive Computing, LLC

- Community Contributions
  - Free access to malware samples
  - Largest open malware site on the Internet
  - ~350k hits per month
- *"It's like an anti-virus company, but without that fake "We're better than you" attitude."*  
- Dave Aitel
- Business Services



# Overview of Talk

- Problem Discussion
- Software Armoring Techniques
- Covert Debugging Requirements
- Dynamic Instrumentation for Debugging
- OS Pagefault Assisted Covert Debugging
- Application – Generic Autounpacking
- Results



# What are the problems?

Malware analysis necessary for defense

- Creating signatures
- Understanding attacks (targeted/untargeted)
- Data mining trends and unknown threats
- Determining phylogeny of variants



# What are the problems?

Malware wants to stop us from analyzing and understanding it

- Packing hinders our analysis
- Anti-analysis techniques
- Obfuscation hinders automation
- Automation is key to rapid analysis



# What is the problem?

- Huge number of malware samples
  - Example: We have almost 300,000
  - More hitting victims every day
- Analyst time is expensive
  - Individual samples can take hours to analyze
- We must automate the process to keep up
- Packers degrade automation
- **We need to automatically decrypt malware!**





# Previous Work

- Shadow Walker
  - Rootkit Memory Hiding
  - Jamie Butler, Sherrie Sparks
- PaX
  - Linux buffer-overflow prevention
- OllyBonE
  - Break on Execute for OllyDbg
  - Joe Stewart
- Memalyze
  - Tracing memory access
  - Skape
- PolyUnpack – Paul Royal et. al @ Georgia Tech
- Halvar's VxClass auto-unpacker



# Gaps

- The available solutions are detectable
- Not all are fully automatable
- Smaller percentage of success
- Some rely on signature based techniques
- In some cases slow
- No one solution addresses all these problems



# What we will show you

- Techniques that are a crucial step in the process of automating Malware decryption
- Example code that may help you in implementing your own automated decryption tools
- Ideas on what further steps are needed to solve the malware analysis automation problem



# What we will not show

- This is research code, not production
- Proof-of-concept
  - a short and/or incomplete realization (or synopsis) of a certain method or idea(s) to demonstrate its feasibility, or a demonstration in principle, whose purpose is to verify that some concept or theory is probably capable of exploitation in a useful manner (wikipedia ftw)



# Implications

- Analysis automation now within our reach
- Obfuscation no longer a major obstacle
- Ability to process 1000's of files rapidly
- Malware authors will have to step it up
  - Raising the bar
- Advanced tools/products can be developed



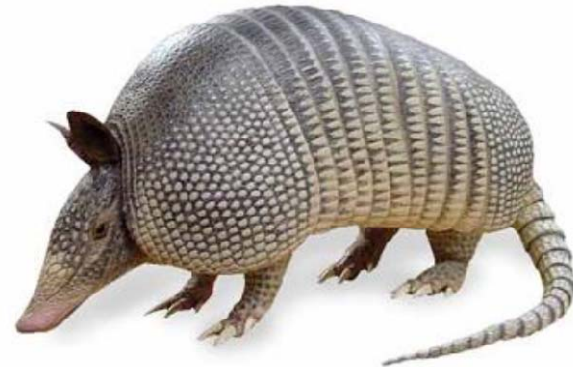
# Software Armoring Overview





# Software Armoring

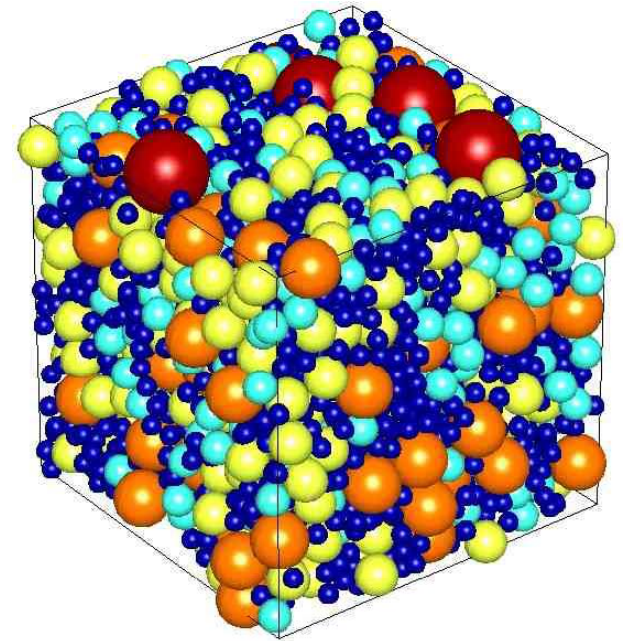
- Packing/Encryption
- SEH Tricks
- VM Detection
- Debugger Detection
- Shifting Decode Frame





# Packing/Encryption

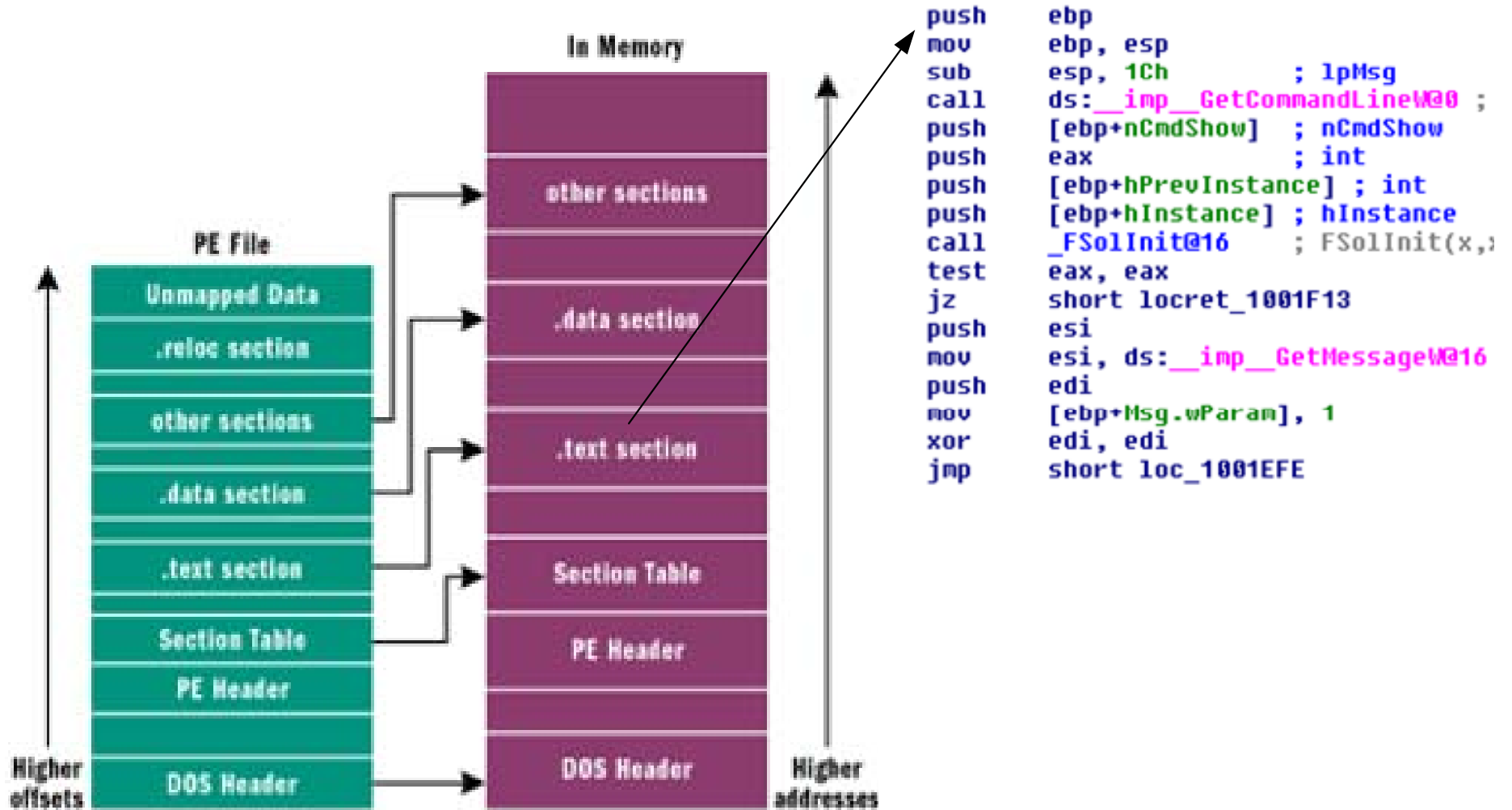
- Self-modifying Runtime Code
  - Small Decoder Stub
  - Decompresses the main executable
  - Restores imports
- Play Tricks with Portable Executables
  - Hide the Imports
  - Obscure relocations
  - Encrypt/compress the executable





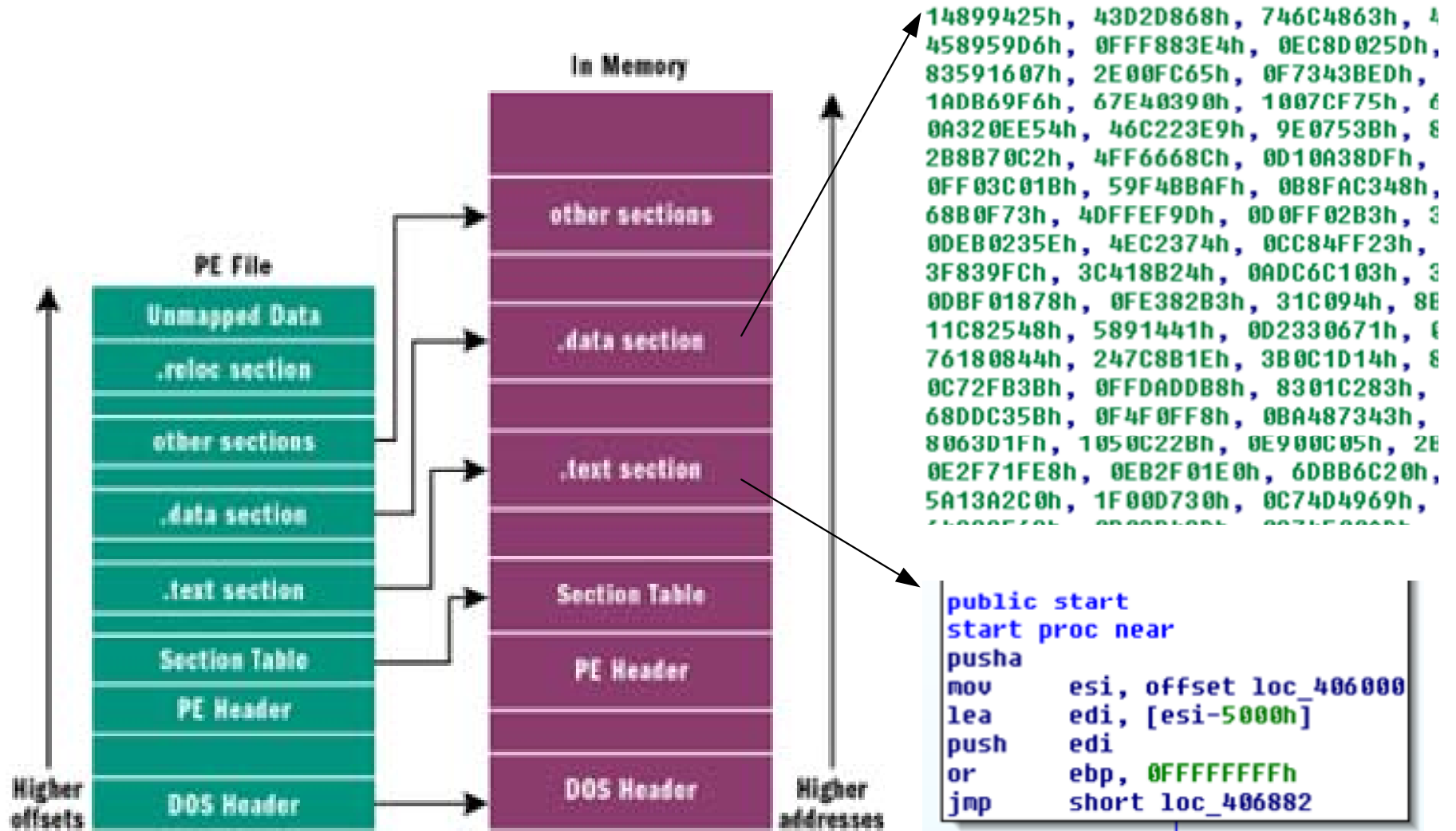


# Normal PE File





# Packed PE File





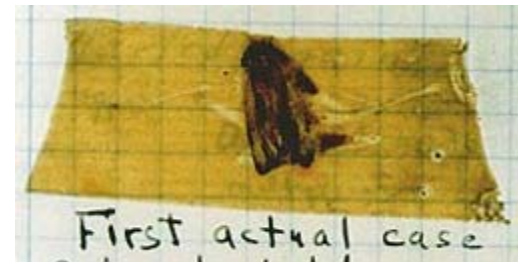
# Virtual Machine Detection

- Single instruction detection
  - SLDT, SGDT, SIDT
  - See: Redpill, Scoopy-Doo, OCVmdetect
- Instructions for Privileged/Unprivileged CPU mode
  - VMs try to be efficient, some instructions insecure
  - Do not fully emulate x86 bug for bug



# Debugger Detection

- Windows API
  - IsDebuggerPresent() API call
  - Checks PEB for magic bit
  - Bit toggling works
- Timing Attacks
  - Issue RDTSC instruction, compare to known values
  - Amazingly effective





# Debugger Detection (cont.)

- Breakpoint Detection
  - Int3 (0xCC) Instruction Scanning
  - Checksumming of executable
- Hardware Debugging Detection
  - Check CPU Flags for debug signatures
- SoftICE Detection
  - Modification of Int3 Scanning
  - Checksumming
  - BoundsChecker and other signatures



# SEH Tricks

- Structured Exception Handler
- Used to handle errors in running code
- Malware will overload this function to unpack code
- Debugger thinks SEH exceptions are for it
- Debugger dies
  - Divide by 0



# Shifting Decode Frames

- Execution is split at the basic block level
- Block is decoded, executed, and then encoded again
- Hard to defeat!
- Implemented in Patchguard for Vista 64 and Windows Server 2003 64-bit



# Use Hardware for Analysis

- Nearly as capable as VM solutions
- Just as cheap\*
- Almost impossible to detect
- Safe solutions available
- Real hardware control possible
  - As will be demonstrated

\* Assuming software licensing costs





# Cost Comparison

## Hardware

- Cheapest Dell \$349
  - Brand new
  - Cheaper elsewhere
  - XP License included\*
  - Deepfreeze \$45

**Total cost: \$394**

## Software

- VMWare - \$189
  - XP \$278.99 \*
  - Other solutions cheaper

**Total cost: \$467.99**

\* Assuming relevant US piracy laws followed



# Replacing Vmware Snapshots

- Faronics Deepfreeze
  - Implements copy on write protection
  - Analogous to VMWare snapshot
  - Kernel driver
  - Not perfect, and hackable (like anything)
  - [www.faronics.com](http://www.faronics.com)
- Disk Image Safe Installation
  - dd your drive in case Deepfreeze fails
  - Last resort restoration



# Other Good Tools

- Firewire kernel debugger
  - WinDBG (thanks MSFT)
- Syser Debugger
  - [www.sysersoft.com](http://www.sysersoft.com)
  - SoftICE replacement
- Debuggers detectable (telock) so be careful



# Software Armoring Achilles Heel

**If it executes,  
it can be unpacked.**

[[http://www.security-assessment.com/files/presentations/Ruxcon\\_2006\\_-\\_Unpacking\\_Virus,\\_Trojans\\_and\\_Worms.pdf](http://www.security-assessment.com/files/presentations/Ruxcon_2006_-_Unpacking_Virus,_Trojans_and_Worms.pdf)]



# Manual Unpacking





# Unpacking

- How an Unpacker Works:
  - Writes to an area of memory (decode)
  - Memory is read from (execute)
  - More writes to memory (optional re-encoding)
- CPU Only Executes Machine Code
- This process can be monitored
- Unpacking is directly related to timing
  - At some point, it ***must*** be unpacked



# Manual Unpacking Process

- Consists of several stages
  - Identify Packer Type
  - Find OEP or get process to unpacked state in memory
  - Dump process memory to file
  - Fixup file / rebuild Import Address Table (IAT)
  - Ensure file can now be analyzed



# Manual Unpacking Process

- Several methods to identify packer type
  - PEiD
  - Msfpescan
  - PEFile from Ero Carrera
    - OC patched to harden against ~275k Malware
  - Manually look at section names
  - Other packer scanners like
    - Protection-id
    - Pe-scan





# Manual Unpacking Process

**Offensive Computing**

**Malware Search**

Search for sum or name

Search

Malware: 42550

Malware:

**Support OC**

by Google

Select Metasploit Framework

Modes:

- j <reg> Search for jump equivalent instructions
- s Search for pop+pop+ret combinations
- x <regex> Search for regex match
- a <address> Show code at specified virtual address
- D Display detailed PE information
- S Attempt to identify the packer/compiler

Options:

- A <count> Number of bytes to show after match
- B <count> Number of bytes to show before match
- I address Specify an alternate ImageBase
- n Print disassembly of matched data

msf > msfpescan -f upx\_scrambler\_calc.exe -S

upx\_scrambler\_calc.exe: UPX-Scrambler RC v1.x [667] (1 matches)

msf >

**PEiD v0.94**

File: C:\packers\upx1.20\_calc.exe

Entrypoint: 00020310 EP Section: UPX1

File Offset: 00007710 First Bytes: 60,BE,00,90

Linker Info: 7.0 Subsystem: Win32 GUI

UPX 0.89.6 - 1.02 / 1.05 - 1.24 -> Markus & Laszlo

Multi Scan Task Viewer Options About Exit

Stay on top

Filetype: PE executable for MS Windows (GUI) Intel 80386 32-bit, UPX compressed

Packer: UPX v0.89.6 - v1.02 / v1.05 - v1.22 [599] (1 matches)



# Manual Unpacking Process

- Methods to find OEP / unpacked memory
  - OllyScripts
    - <http://www.tuts4you.com>
    - <http://www.openrce.org>
  - OEP finder tools
    - OEP finders for specific packers
    - OEP Finder (very limited)
    - PE Tools / LordPe
    - PEiD generic OEP finder



# Manual Unpacking Process

The screenshot displays the OllyDbg interface for debugging the process `upx1.20_calc.exe`. The CPU window shows the main thread at address `01012475` with the instruction `PUSH 70`, which is identified as the OEP (Original Entry Point). A comment indicates this was found by 'fly'. Below this, the instruction `PUSH upx1.20_.010015E0` is shown. A message box from OllyScript says: "Just : OEP ! Plz Dump and Fix IAT . Good Luck".

The PEID v0.94 window is open, showing the file `C:\packers\upx1.20_calc.exe` with the following details:

- File: `C:\packers\upx1.20_calc.exe`
- Entrypoint: `00020310`
- EP Section: `UPX1`
- File Offset: `00007710`
- First Bytes: `60,BE,00,90`
- Linker Info: `7.0`
- Subsystem: `Win32 GUI`

The Generic OEP Finder FX [v0.8 Beta] window shows the analysis progress at 100% and indicates that the OEP has been reached. It lists possible OEPs, with the first one being `01012475`.

The GenOEP window confirms the findings: "Found OEP: 01012475".

The taskbar at the bottom shows the running processes `c:\bin\reversing\ida\idag.exe` and `c:\downloads\framework-2.7.exe`.



# Manual Unpacking Process

- Dump process memory to file
  - OllyDump
  - LordPE
  - Custom tools
- Example:
  - `OpenProcess()`
  - `ReadProcessMemory()`
  - `CreateFile()`
  - `WriteFile()`



# Manual Unpacking Process

The screenshot displays the manual unpacking process using OllyDbg and LordPE Deluxe. The background shows OllyDbg with the disassembly window open to address 01012475, where a comment indicates the OEP (Original Entry Point) has been found. The registers window on the right shows EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI, and EIP.

In the foreground, the 'OllyDump - upx1.20\_calc.exe' dialog box is open, showing the following settings:

- Start Address: 1000000
- Size: 28000
- Entry Point: 20310 (Modified to 12475)
- Base of Code: 19000
- Base of Data: 21000
- Checked: Fix Raw Size & Offset of Dump Image
- Checked: Rebuild Import
- Method 1: Search JMP[API] | CALL[API]
- Method 2: Search DLL & API name string

The 'Section' table within the dialog shows:

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
UPX0	00018000	00001000	00018000	00001000	E0000080
UPX1	00008000	00019000	00008000	00019000	E0000040
.rsrc	00007000	00021000	00007000	00021000	C0000040

Below the dialog, the disassembly window shows the following instructions:

```
010124E1 6A 02      PUSH 2
010124E3 FF15 0C120001 CALL [10012001]
```

The 'LordPE Deluxe' window is open, showing a list of loaded modules:

Path	PID	ImageBase	ImageSize
c:\bin\reversing\lordpe\lordpe.exe	00000168	00400000	00036000
c:\program files\mozilla firefox\firefox.exe	000007B4	00400000	006F4000
c:\bin\reversing\ida\idag.exe	00000310	00400000	00295000
c:\downloads\framework-2.7.exe	000003E4	00400000	00033000
c:\bin\reversing\peid\peid.exe	00000188	00400000	0007B000
c:\windows\system32\cmd.exe	00000184	4AD00000	00061000
c:\program files\metasploit\framework2\bin\b...	000006F8	00400000	00088000
c:\program files\metasploit\framework2\bin\p...	000001A4	00400000	0000E000
c:\windows\system32\cmd.exe	000002F0	4AD00000	00061000
c:\packers\upx1.20_calc.exe	000003AC	01000000	00028000

The 'Rebuild Status' dialog box shows the following progress:

- Dumpfix... done
- Wipe Relocation...no Relocation present
- Realigning...done
- Current filesize: 24F75h
- File minimized to: 92%
- Rebuild ImportTable...done
- Validate PE image...done
- Binding Imports...failed
- New filesize: 24F75h
- File minimized to: 92%
- Rebuilding finished.



# Manual Unpacking Process

- Fixup file / rebuild Import Address Table (IAT)
  - ImportRec probably best tool
  - Revirgin by +Tsehp
  - Manually with a hex editor (tedious)
- IAT contains list of functions imported
  - Very useful for understanding capabilities

Address	Ordinal	Name	Library
01001214		??1type_info@@UAE@xZ	msvcrt
01001210		??3@YAXPAX@Z	msvcrt
01001220		?terminate@@YAXZ	msvcrt
010010B8		CallWindowProcW	USER32
010010F0		CharNextA	USER32
0100111C		CharNextW	USER32
010010B0		CheckDlgButton	USER32
01001144		CheckMenuItem	USER32
01001148		CheckMenuRadioItem	USER32
0100110C		CheckRadioButton	USER32
010010...		ChildWindowFromPoint	USER32
010010F4		CloseClipboard	USER32
0100106C		CloseHandle	KERNEL32
0100116C		CreateDialogParamW	USER32



# Manual Unpacking Process

The screenshot shows the manual unpacking process using LordPE Deluxe and Import REConstructor. The main window is LordPE Deluxe, which is attached to the process `c:\packers\upx1.20_calc.exe (000003AC)`. The 'Imported Functions Found' list shows several DLLs and their functions, including `advapi32.dll`, `gdi32.dll`, `kernel32.dll`, `shell32.dll`, `user32.dll`, and `msvcrt.dll`. The 'Log' window shows the progress of the unpacking process, including the addition of a new section and the saving of the unpacked file.

The 'Import REConstructor v1.6 FINAL (C) 2001-2003 MackT/uCF' window is also visible, showing the 'Attach to an Active Process' dropdown set to `c:\packers\upx1.20_calc.exe (000003AC)`. The 'Imported Functions Found' list in this window is empty, and the 'Log' window shows the progress of the unpacking process.

The 'Revirgin by +Tsehp 1.5 public version' window is also visible, showing the 'Select Module to Attach' dropdown set to `upx1.20_calc.exe 000003AC 00028000 01000000`. The 'IAT Critical Values' section shows the OEP, RVA, and Length values for the process.

Module	Ordinal	Name	Address	IATRva	Refs
upx1.20_calc.exe	000003AC	00028000	01000000		

IAT Critical Values

Field	Value
OEP	01020310
RVA	00001000
Length	00000228

IT Values + generator

Field	Value
RVA	
Length	

Log

```
Fixing a dumped file...
6 (decimal:6) module(s)
84 (decimal:132) imported function(s).
*** New section added successfully. RVA:00028000 SIZE:00001000
Image Import Descriptor size: 78; Total length: B30
C:\packers\unpacked\upx1.20_calc_lordPE_dumped_exe saved successfully.
```



# Manual Unpacking Process

- Ensure file can now be analyzed
- Clean disassembly should be available
- IAT should be visible
- Functions should be found
- Strings clear and useful
- Manual unpacking process can be tedious
- Hardest part is generally finding the OEP





# Manual Unpacking Process

The screenshot displays the IDA Pro interface for a file named 'C:\packers\unpacked\upx1.20\_calc\_lordPE\_dumped\_exe'. The main window shows assembly code for a function starting at address 01010B13. The code includes instructions like 'push ebp', 'mov esp, ebp', and 'mov ecx, [ebp+...]', ending with 'jns short loc\_1...'. The 'Imports' window lists various system API calls such as 'RegOpenKeyExA', 'RegQueryValueExA', and 'SetBkColor' from 'advapi32', and 'GlobalCompact', 'GlobalFree', and 'GlobalAlloc' from 'kernel32'. The 'Functions window' lists numerous subroutines like 'sub\_10013D1' through 'sub\_1004491'. The 'Names window' shows a list of symbols including 'a0123456789abcd' and 'aAll'. The 'Strings window' lists strings like 'gdi32.dll', 'SetBkColor', and 'tTextColor'.



# Manual Unpacking Process

- Show Manual Unpacking Movie





# So What?

- These are all variations on a theme
- There should be a generic way to debug
- Need to modify at a fundamental level
- Solution should be:
  - Generic – Work across set of executables
  - Efficient – Good performance for non-debug
  - Undetectable (as much as possible)
  - Extensible – Automation is the key



# Unpacking: The Algorithm

- Track written memory
- If that memory is executed, it's unpacked
- Must monitor:
  - Memory writes
  - Memory Executions
- Automate the process



# Dynamic Instrumentation





# Dynamic Instrumentation

- Allows a running process to be monitored
- Intel PIN
  - Uses Just-In-Time compiler to insert analysis code
  - Retains consistency of executable
  - Pintools – Use API to analyze code
  - Good control of execution
    - Instruction
    - Memory access
    - Basic block
  - Process Attach / Detach



# Dynamic Instrumentation

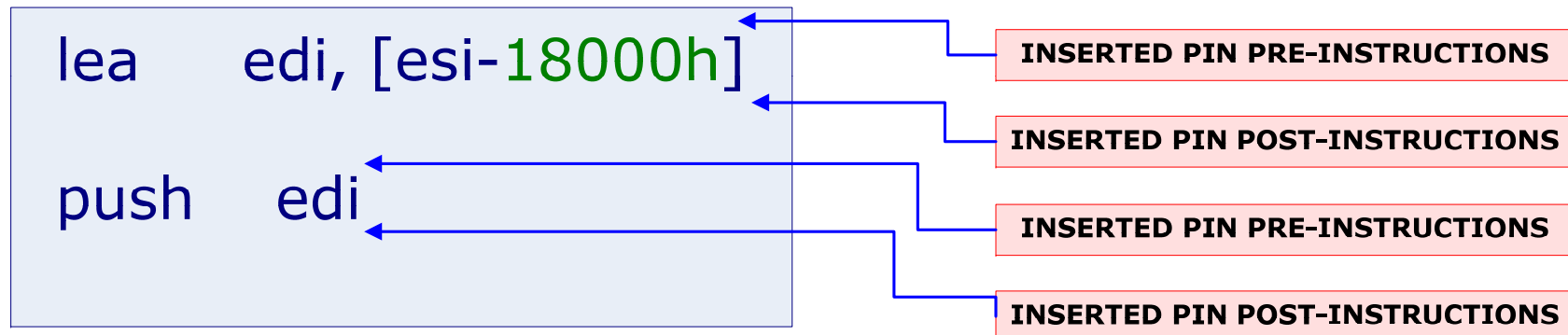
## NORMAL INSTRUCTIONS

```
pusha  
mov     esi, offset dword_1019000  
lea     edi, [esi-18000h]  
push   edi  
or      ebp, 0FFFFFFFFh  
jmp     short loc_1020332
```



# Dynamic Instrumentation

## PIN MODIFIED INSTRUCTIONS







# Dynamic Instrumentation

## PIN MODIFIED INSTRUCTIONS

```
pusha
    PIN INSTRUCTIONS
mov    esi, offset dword_1019000
    PIN INSTRUCTIONS
lea    edi, [esi-18000h]
    PIN INSTRUCTIONS
push   edi
    PIN INSTRUCTIONS
or     ebp, 0FFFFFFFh
    PIN INSTRUCTIONS
jmp    short loc_1020332
```



# Implementation

- Use PIN hooks for
  - Memory Writes
  - Executes
- Track writes in hash table
- If execution occurs on written data, dump



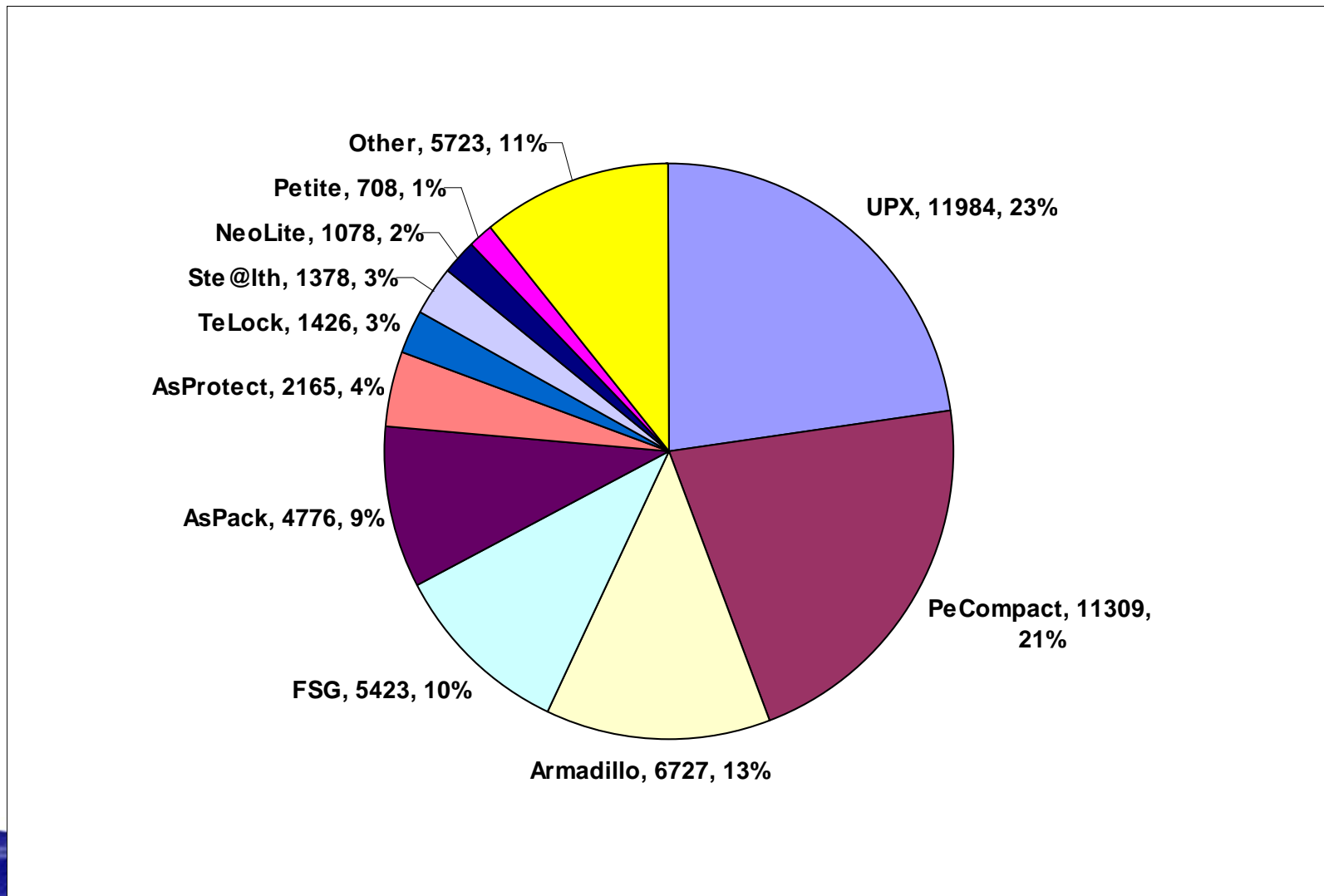
# Results

- Successful against:
  - Most commonly used packers
  - Packers that don't self verify
  - ~70% of packed malware in OC collection



# Dynamic Instrumentation - Packers

- 153701 Samples Scanned / 54123 Detected Packers





# Dynamic Instrumentation

- Instruction tracing for the following packers
  - Aspack
  - FSG
  - PECompact
  - UPX
- Created Simple Hello World Application
- Graphed results with Oreas GDE

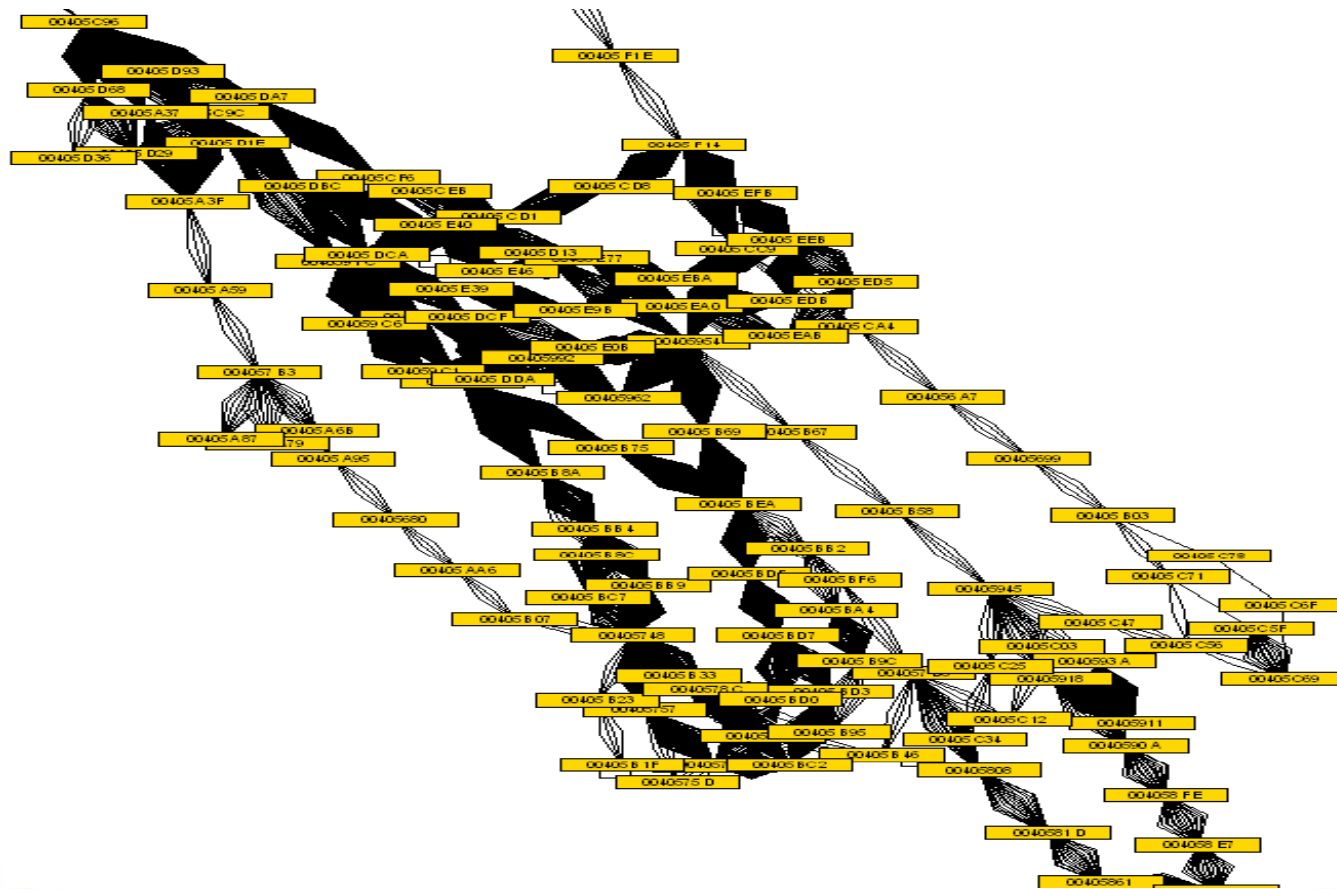


Aspack 2.12



# Results

- Unpacking loop is easy to find





# Dynamic Instrumentation Results

- Generic Algorithm Described Previously works well
- All addresses verified by manual unpacking
- Addresses display clustering, which must be taken into account
- Attach / Detach is effective for taking memory snapshots of an executable





# Dynamic Instrumentation Caveats

- Detectable
  - Memory checksums
  - Signature scanning
- Difficult to use (sorry)
- Extend this to work generically, non-detectably
- Slow – ~1,000 times slower than native
  - Other methods/tools can be even slower
- Need faster implementation



# Towards a Solution

- Core operating system component that:
  - Monitors all memory
  - Intercepts memory accesses
  - Fast Interception and Logging
  - Fundamental part of OS



# **Overloading the Memory Management Unit**

or

## **OS 101 How Virtual memory Works**

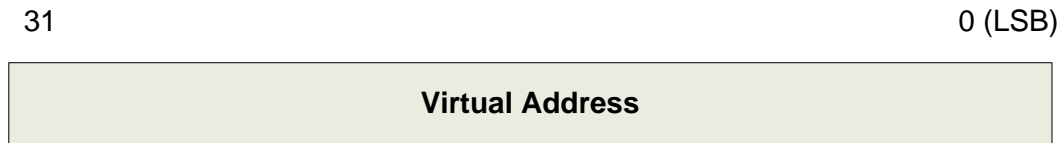


# Intel Memory Management

- Each process has its own memory
- Memory must be translate from Virtual to Physical Address
- Non-PAE Mode 32bit Processors use 2 page indexes and a byte index
- Each process has its own Page Directory



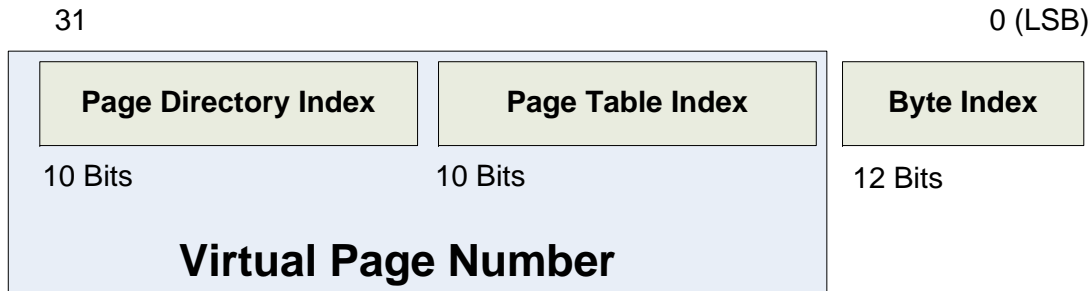
# Example Memory Translation



CPU References Virtual Memory Address

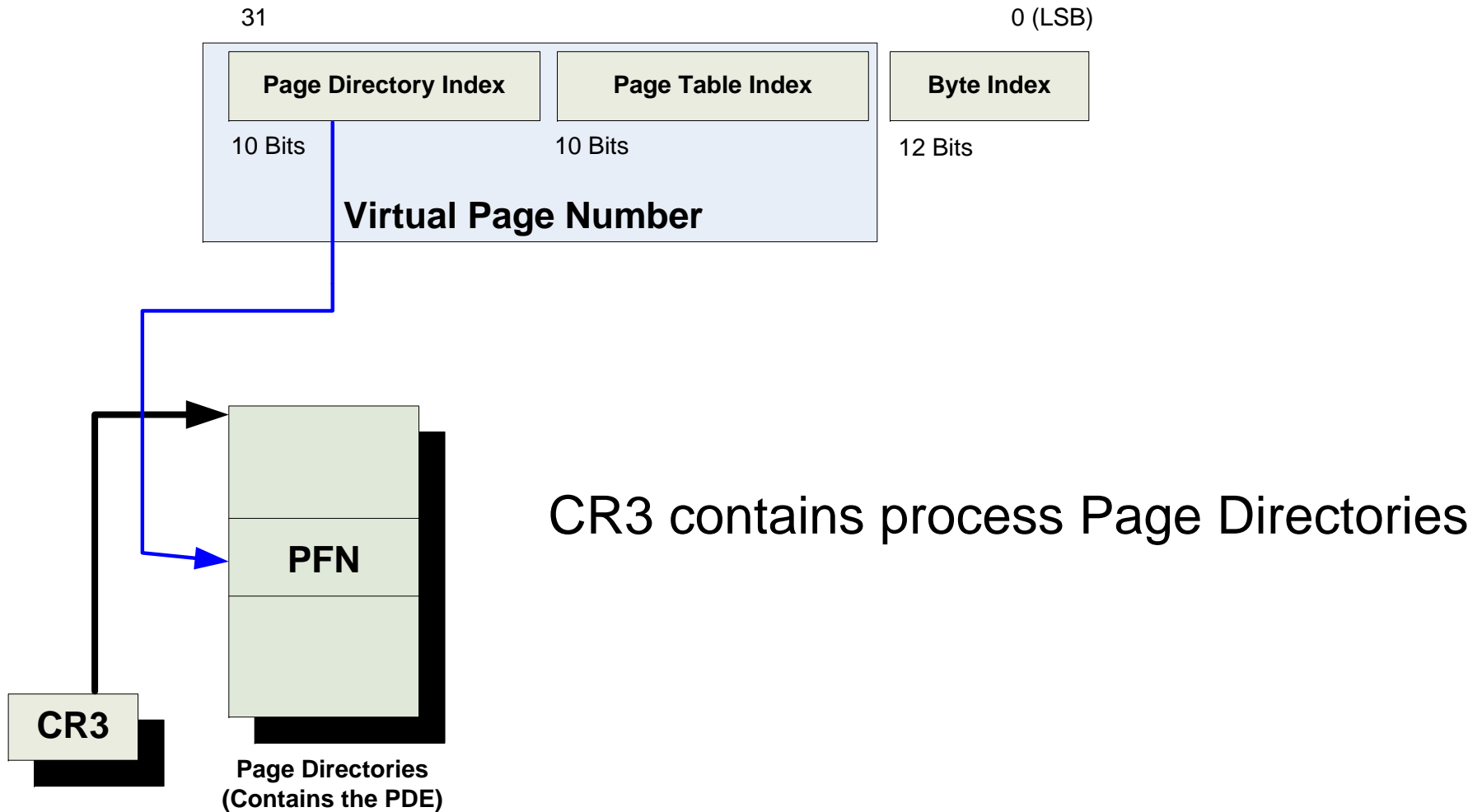


# Example Memory Translation



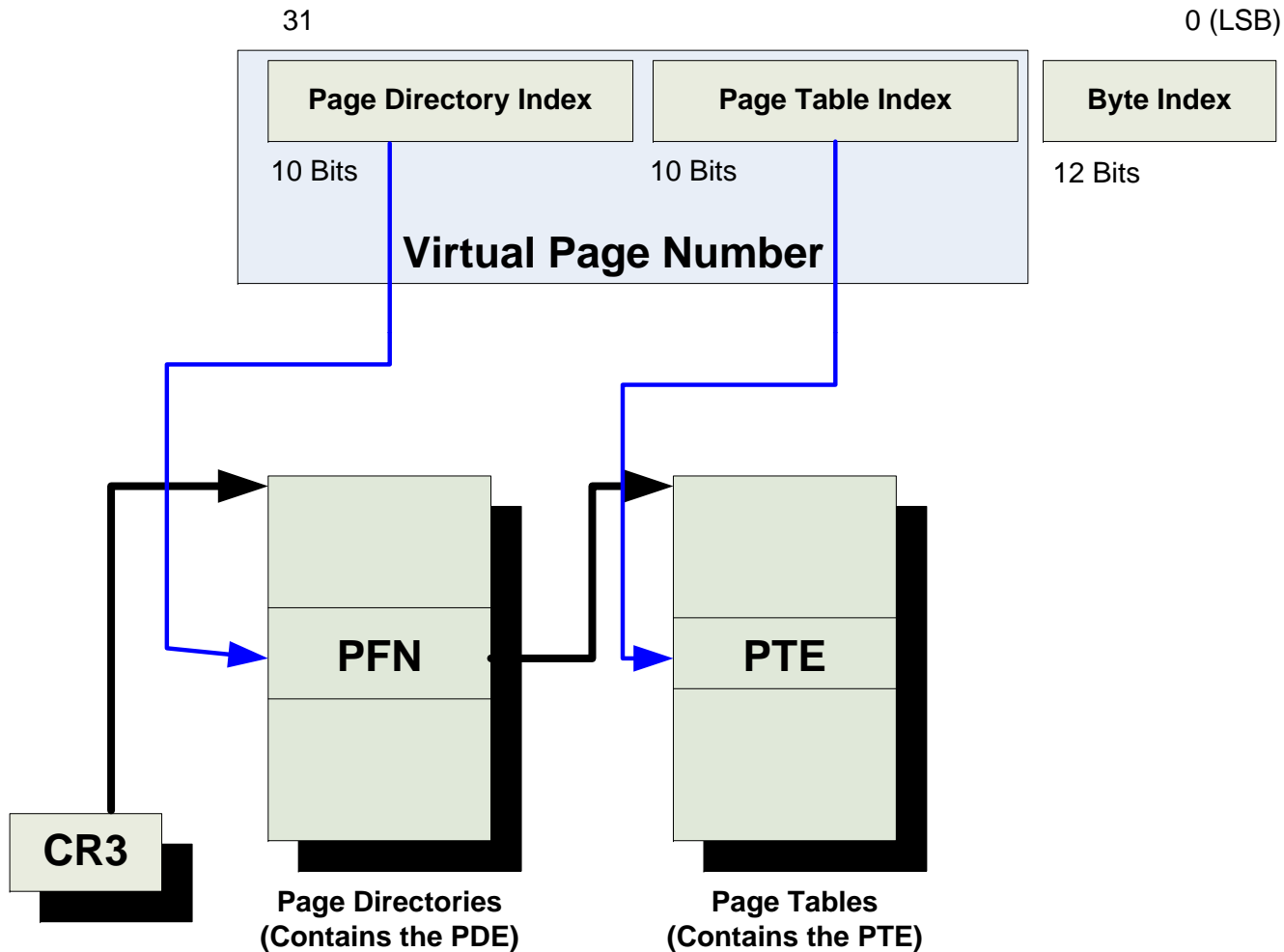


# Example Memory Translation





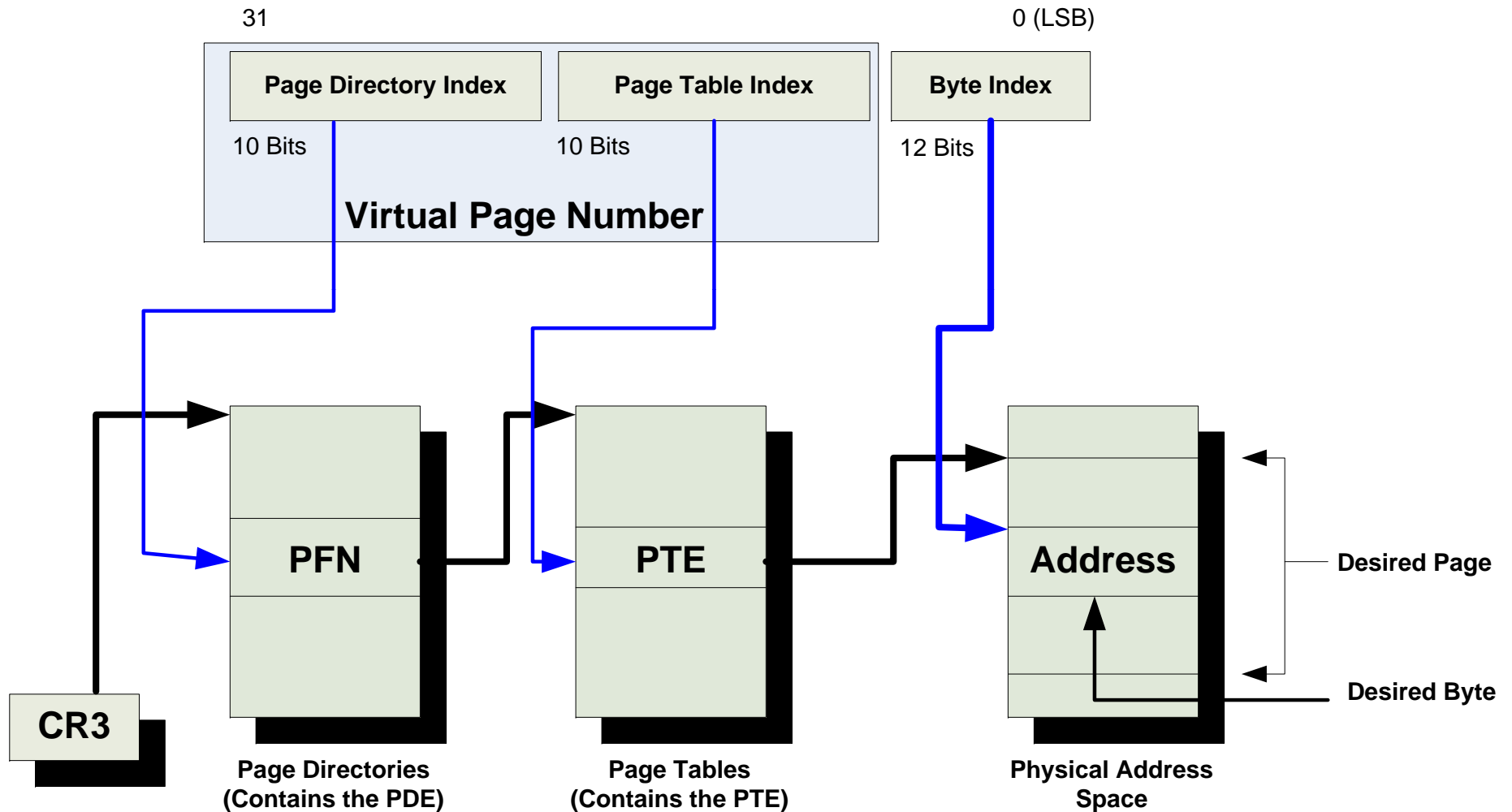
# Example Memory Translation







# Example Memory Translation





# MMU Data Structures

- Page Directory Entry is hardware defined
  - Contains permissions, present bit, etc.
- Page Table Entry also hardware defined
  - Permissions (Ring0 vs. all others)
  - Present bit (paged to disk or not)
  - “User” defined bits (for OS)



# Virtual Address Translation

- Translation Lookaside Buffer (TLB) is major source of optimization
- Hardware resolves as much as possible
- Invokes page fault handler when
  - Page is not loaded in RAM
  - Incorrect privileges
  - Loaded, but mapped with demand paging
  - Address is not legal (out-of-range)
- All indicated by special fields



# Intel TLB Implementation

- Two TLBs maintained
  - Data **DTLB**
  - Instructions **ITLB**
- ITLB more optimized than DTLB
  - Less lookups for instructions == faster code
  - DTLB accessed less



# Intel TLB Population

- Data TLB

- Address is cached upon lookup

```
mov eax, dword ptr [eax]
```

- Instruction TLB

- Address is cached upon execution

```
mov ecx, dword ptr [eax]  
mov [eax], 0xC3 // 0xC3 is a near ret  
call eax  
mov [eax], ecx
```



# INTRODUCING SAFFRON



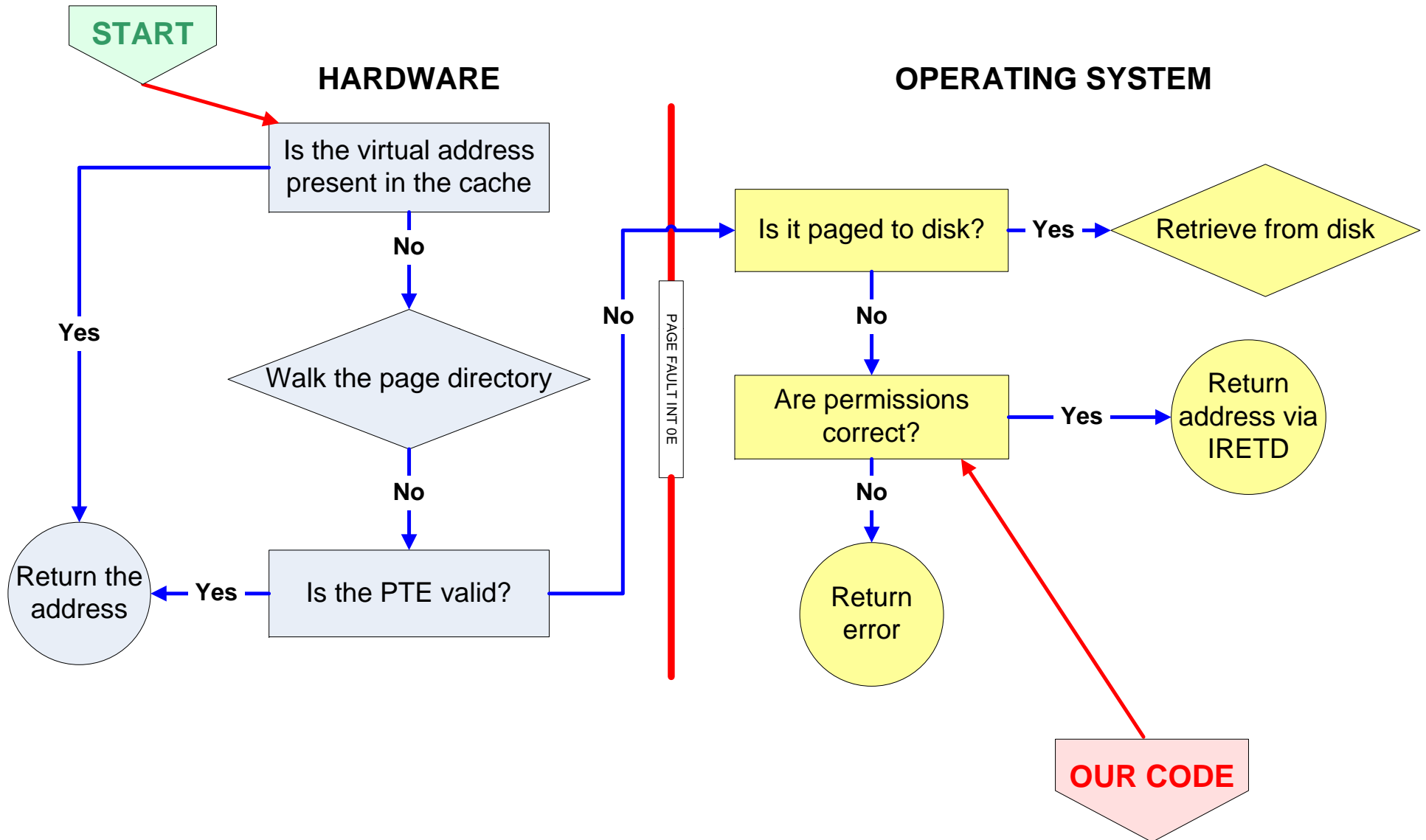


# Introducing Saffron

- Intel PIN and Hybrid Page Fault Handler
- Inspired by OllyBonE (Joe Stewart, DC14)
- Designed for 32-bit Intel x86 CPUs
- Replaces Windows 0x0E Trap Handler
- Logs memory accesses



# Offensive Computing - Malware Intelligence

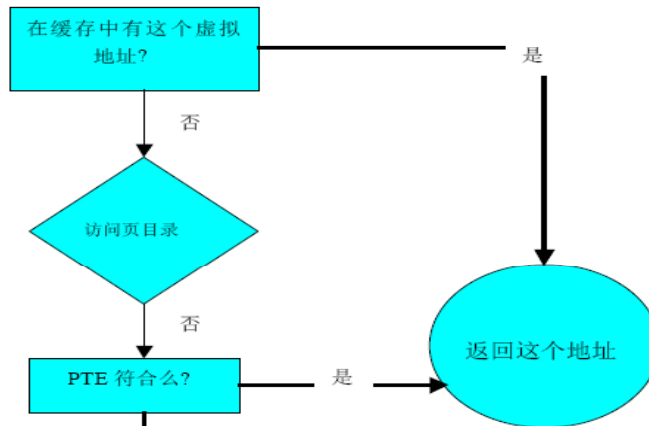




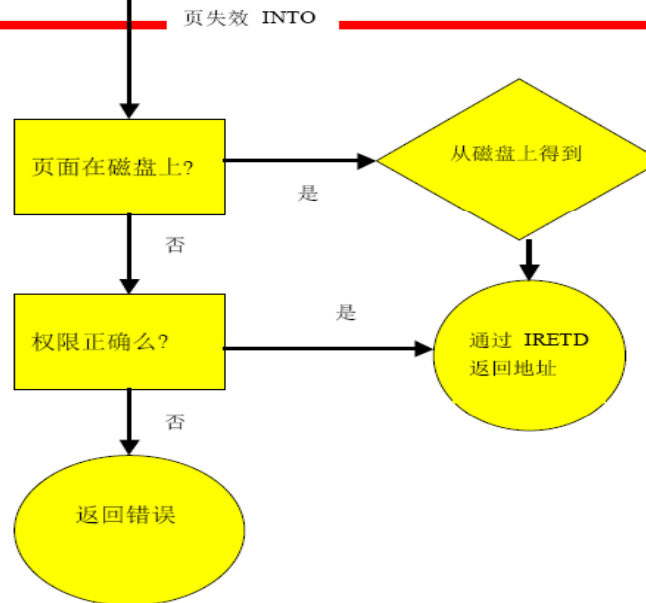


# Translated (stolen) Version

硬件

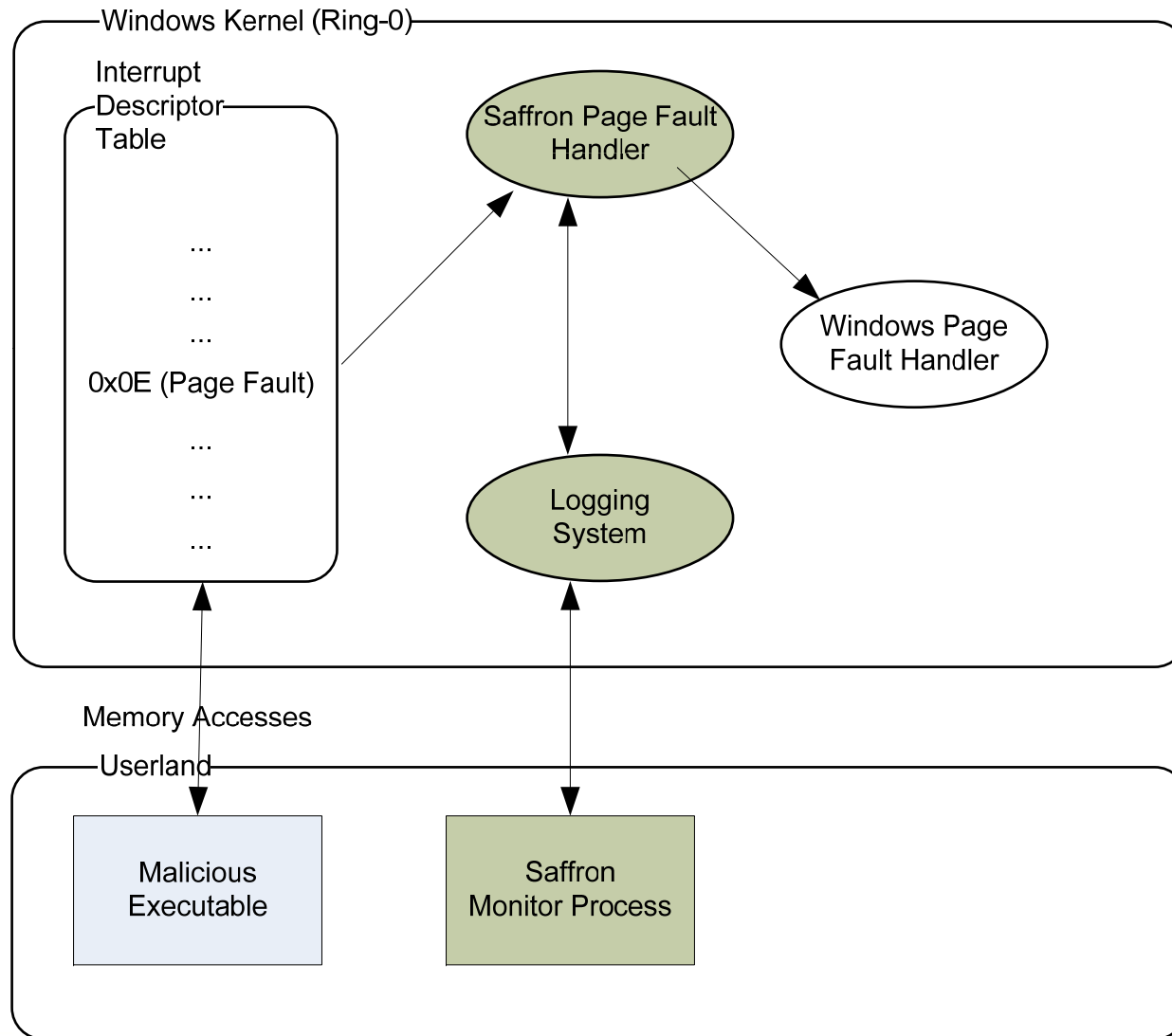


操作系统





# Saffron System Implementation





# Process Monitoring

- Mechanism
  - Overloading of supervisor bit in page fault handler
  - Mark supervisor bit on each valid PTE
  - Invalidate the page in the TLB with INVLPG
- Finding Memory
  - All process memory must be found
  - Iterate through all pages for a process
  - Read PE Header and find sections



# Trap to Page Fault Handler

- Determine if a watched process
- Unset the supervisor bit
- Loads the memory into the DTLB
- Resets supervisor bit



# Modifying the Autounpacker

- Watch for written pages via ITLB
- Monitor for executions into that page
- Mark Address as Original Entry Point
- Dump memory of the process



# Results

- Reads, writes, and executes are exposed
- Program execution can be tracked, controlled
- Memory reads, writes are extremely apparent
- Executions only show for each individual page
- Very Fast!



# Autounpacker Results

- Effective method for bypassing detection
  - SEH decode problem is easily solved
  - Memory checksum
    - No process memory is modified
    - Good dumps obtained
  - Effective across wide range of packers



# Autounpacking Caveats

- System Requirements
  - Windows XP, SP2
  - No Data Execution Prevention (DEP)
  - Single CPU
    - Disable multiple CPUs in BIOS
    - /ONECPU flag in boot.ini
  - 32-bit Only (could be ported to 64bit)





# Big Announcement

Technique now works on Vmware 6



# Autounpacking Caveats

- Real Hardware / VMWare 6.0 or higher
  - Virtual Machines (Older versions of Vmware)
    - Play their own tricks with the ITLB
    - Extremely detectable
  - Real Hardware Take proper precautions
    - Restoration procedure
    - Isolated network
- Must not have a kernel debugger attached
  - Hilarity will ensue (silly TeLock)



# Demo of Unpacking

- Demonstrate Saffron PFH





# Future Work

- Initial release of Saffron-DI  
Blackhat USA 2007
- Packaged Version of Saffron-Kernel
  - Drag and drop unpacking
- Offensive Computing Integration
  - Any day now <sup>TM</sup>



# Questions?

- Paper, presentation, code available at [www.offensivecomputing.net](http://www.offensivecomputing.net)
- Thanks to:
  - Lorie Liebrock, Houdini, Skape, Bugcheck, Skywing, Ty Bodell, Uninformed, #vax

