

# Strike/Counter-Strike: Reverse Engineering Shiva

Chris Eagle

Naval Postgraduate School

[cseagle@nps.navy.mil](mailto:cseagle@nps.navy.mil)



**Black Hat Briefings**

# Outline

- Introduction
- Runtime encryption tools
- Shiva review
- Reversing Shiva
- Summary



# Introduction

- Executable encryption/obfuscation
  - Post compilation manipulation of an executable to prevent/slow reverse engineering efforts



# Introduction (II)

- Typical approach
  - Encrypt/compress executable
  - Bind it with an unwrapper front end
  - Unwrapper provides minimal compliance with executable format standards



# Introduction (III)

- Execution
  - Unwrapper extracts (in some way) the original binary
  - Unwrapper transfers control to the entry point of the original binary
  - Unwrapper is effectively jettisoned



# Outline

- Introduction
- **Runtime encryption tools**
- Shiva review
- Reversing Shiva
- Summary



# Tools Overview

- Windows PE manipulators
  - UPX, ASPack, tElock
  - Scramble UPX
- Linux ELF manipulators
  - UPX, Burneye
  - Shiva



# Outline

- Introduction
- Runtime encryption tools
- **Shiva review**
- Reversing Shiva
- Summary



# Shiva

- Developed by Neel Mehta and Shaun Clowes
- Introduced at CanSecWest 2003
- Discussed again at Black Hat USA 2003
- Released as a Shiva protected binary only



# Shiva Goals\*

- Introduce some novel new techniques
- Advance the state of the art for runtime encryption of Unix executables
- Promote interest in reverse engineering on Unix platforms

\* Mehta - Black Hat USA 2003



# Shiva Protective Measures

- Outer encryption layer
  - Defeats “strings” cripples
  - Slows access to the protected code
- TRAP flag detection
  - Defeat single-stepping
- “checkme” data check



# Shiva Protections (II)

- ptrace defense
  - Exits if ptrace is active
  - Clones itself and the two processes ptrace each other
    - Prevents PTRACE\_ATTACH
    - A process can only be ptraced by one other process
    - Dubbed “inter-ptrace” by Mehta



# Shiva Protections (III)

- Timing checks
- Optional AES, password protected middle encryption layer
  - Protected binaries won't run unless correct password is supplied
- Inner encryption layer
  - Provides runtime protection



# Shiva Protections (IV)

- /proc defenses
  - Only portions of the binary are decrypted at any given time
    - Demand mapped blocks
  - Can't dump fully decrypted image via /proc file system



# Shiva Protections (V)

- INT 3 instruction replacement
  - Some instructions are replaced with INT 3
    - Software breakpoint
  - The instruction's operands are stored
  - When encountered, Shiva emulates the instruction
  - Even if you capture a decrypted code block, some instructions may be missing!



# Outline

- Introduction
- Runtime encryption tools
- Shiva review
- **Reversing Shiva**
- Summary



# Reversing Shiva

- This talk focuses on static analysis techniques
- You just can't hide from static analysis
- But we need to make it faster/easier
- Won't discuss password protected binaries
  - Cryptographic attacks rather than R.E.



# Static Analysis

- Given the defenses present in Shiva, this seems like a good (only?) approach
- IDA Pro Rocks!
- But, Shiva tries to make disassembly tough
  - Jumping into the middle of instructions
  - Polymorphic code generation



```

LOAD: 0A04B0D0 ;
LOAD: 0A04B0D0
LOAD: 0A04B0D0 loc_A04B0D0: ; CODE XREF: start+B1j
LOAD: 0A04B0D0 sub esp, 4
LOAD: 0A04B0D6 mov [esp], esi
LOAD: 0A04B0D9 push ecx
LOAD: 0A04B0DA push edi
LOAD: 0A04B0DB push eax
LOAD: 0A04B0DC jz short near ptr loc_A04B0E1+2
LOAD: 0A04B0DE push eax
LOAD: 0A04B0DF jnz short near ptr loc_A04B0E1+1
LOAD: 0A04B0E1
LOAD: 0A04B0E1 loc_A04B0E1: ; CODE XREF: LOAD:0A04B0DF↑j
LOAD: 0A04B0E1 ; LOAD:0A04B0DC↑j
LOAD: 0A04B0E1 mov eax, 0BE535258h
LOAD: 0A04B0E6 xlat
LOAD: 0A04B0E7 sub dl, [edx]
LOAD: 0A04B0E9 aad 81h
LOAD: 0A04B0EB out dx, al
LOAD: 0A04B0EC jz short loc_A04B14D
LOAD: 0A04B0EE jmp near ptr 70CDE25Dh
LOAD: 0A04B0EE ;
LOAD: 0A04B0F3 dd 814B63B9h, 2C0000C1h, 0EBCE3160h, 92B9BE01h, 3107CF97h
LOAD: 0A04B0F3 dd 80BF66FFh, 0C78126h, 3110A4C0h, 52E981F9h, 89176B40h
LOAD: 0A04B0F3 dd 0F7D0B9CFh, 0F181596Fh, 28D44DEAh, 0B866C031h, 0C0811DDh
LOAD: 0A04B0F3 dd 46E50000h, 0C889C129h, 90C701EBh, 5740775h, 8041404h
LOAD: 0A04B143 db 68h
LOAD: 0A04B144 ;
LOAD: 0A04B144
LOAD: 0A04B144 loc_A04B144: ; CODE XREF: LOAD:0A04B174↓j
LOAD: 0A04B144 cmp edi, 4
LOAD: 0A04B14A jz short loc_A04B14F
LOAD: 0A04B14A ;
LOAD: 0A04B14C db 75h ; u
LOAD: 0A04B14D ;
LOAD: 0A04B14D
LOAD: 0A04B14D loc_A04B14D: ; CODE XREF: LOAD:0A04B0EC↑j
LOAD: 0A04B14D add bh, bh
LOAD: 0A04B14E

```

# Minor Annoyance

- In IDA, just undefine the false target and redefine code at the proper places
  - We can make it almost painless as we shall see
- Much more tedious with gdb



# What Can We Achieve

- Static analysis will only give us a glimpse into the unwrapping algorithm
- It won't execute it for us
  - Do it in our head for fun!
- IDA scripting offers some capability
- IDA plugins offer MUCH more



# Getting Past Layer 1

- Unlike UPX, Shiva offers no option to undo itself
- Ideally, let Shiva run itself through the outer decryption routine
  - gdb, b \*0x0A048068, r, generate-core-file
  - A048068 is currently the address of the first function called following decryption



# But I Want to Live in IDA!

- We can load the core dump into IDA and analyze
  - Without some help, which function is the entry point?
- Analyzing the layer 1 decryption provides better understanding



# Scripted Decryption

- If the algorithm is well-defined we can write an IDA script to mimic it
  - Decrypt and patch the binary within IDA
  - Done for UPX
  - Succeeds where UPX fails when Scramble has been applied
- Shiva isn't so nice



# What I Wanted

- As close to automated script generation as possible
- IDA has great annotation and navigation features
- BUT it won't run code
- Tired of running it in my head

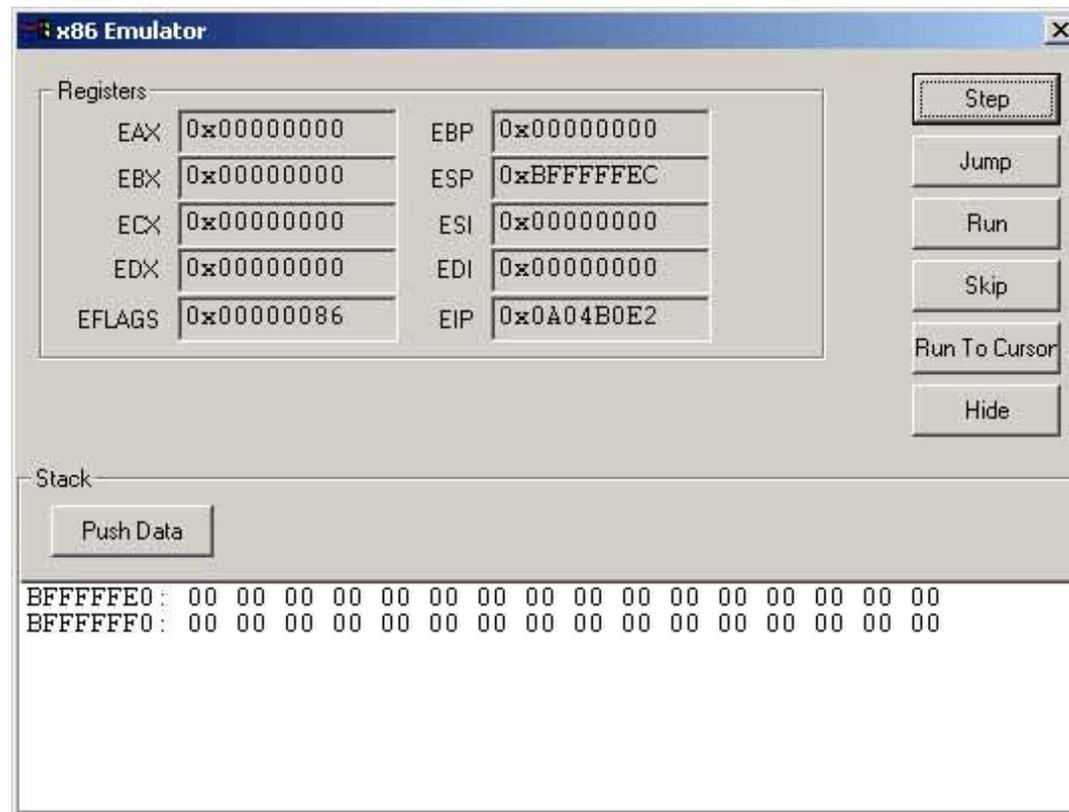


# What I Built

- Virtual x86 plugin for IDA
- Utilizes IDA database for virtual address space
- Provides it's own stack
- Allows you to step through x86 code within IDA
- No need for scripts, just run it!



# Demo



The screenshot shows an x86 Emulator window with the following state:

Registers	
EAX	0x00000000
EBX	0x00000000
ECX	0x00000000
EDX	0x00000000
EFLAGS	0x00000086
EBP	0x00000000
ESP	0xBFFFFFFEC
ESI	0x00000000
EDI	0x00000000
EIP	0x0A04B0E2

Stack section:

Push Data

```
BFFFFFFE0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
BFFFFFFF0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Control buttons: Step, Jump, Run, Skip, Run To Cursor, Hide



# Some Benefits

- No need to generate scripts for unpackers/decryptors
  - Just run the code
- Almost a debugger
  - No library descent
- Step through any x86 code
  - Not tied to a specific OS

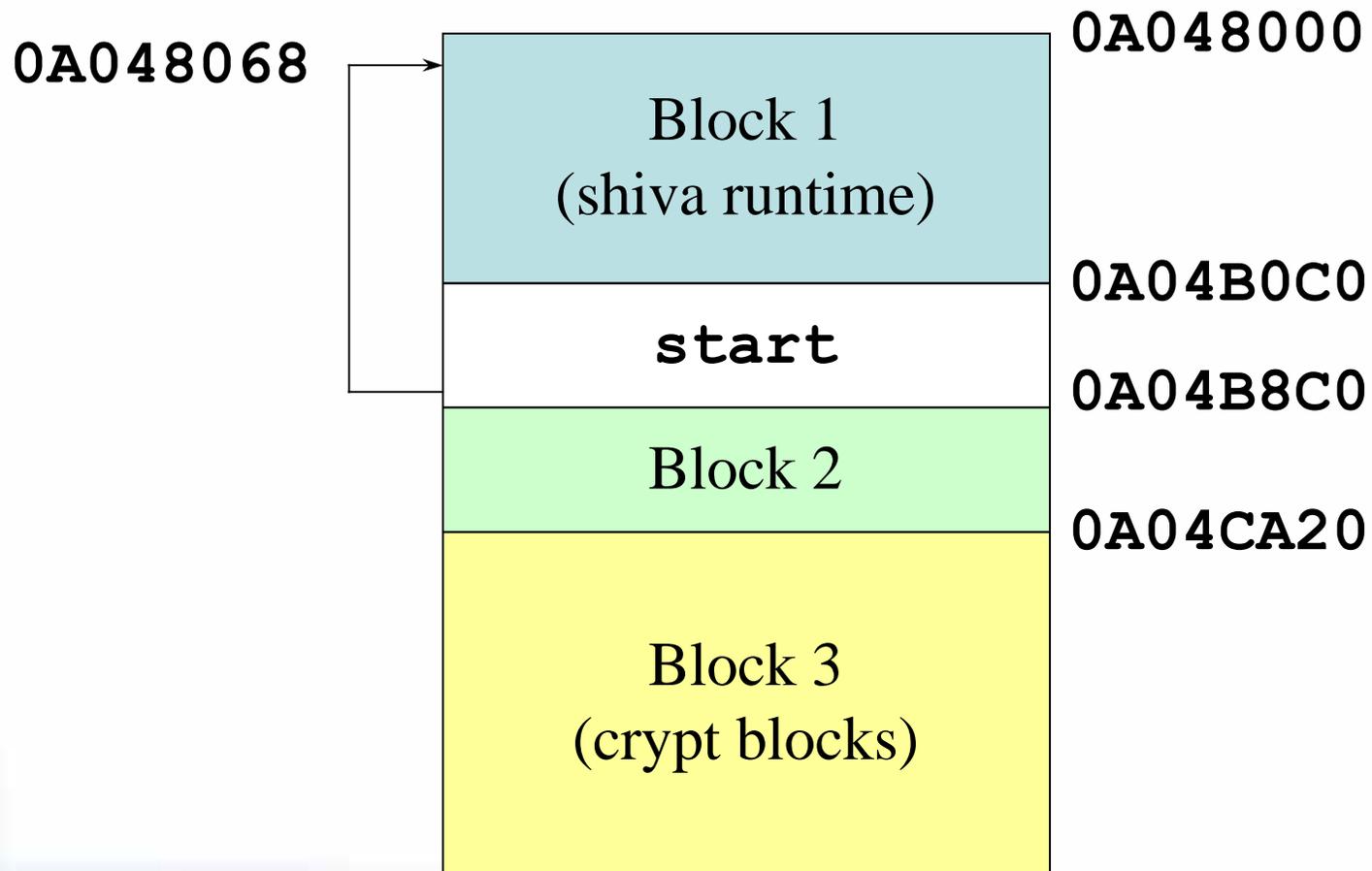


# Back To Shiva

- Layer 1 details
  - Simple XOR and ADD loops over three data blocks
    - Block 1 - Shiva runtime support
    - Block 2 - .rodata for Shiva runtime
    - Block 3 - .data for Shiva runtime
  - Block 3 contains the encrypted user binary



# Shiva Protected File Layout



# Shiva Runtime

- Following layer 1 decryption control transfers to the Shiva runtime controller
- Performs anti-R.E. checks
- Allocates a heap
- Clones monitor process
- Decrypts static crypt blocks
  - User application .data among others



# Layer 3 Encryption

- Remember:
  - layer 2 was optional password protected AES
- Utilizes Tiny Encryption Algorithm (TEA)
  - 128 bit keys
  - Keys obfuscated within binary



# Crypt Blocks

- Shiva breaks a binary up into blocks
- Primarily along the lines of code vs data
  - Data blocks align roughly on natural data boundaries
    - I'll call these Type II blocks
    - Decrypted into place immediately, remain for life of program



# Crypt Blocks (II)

- Code blocks partitioned to about 1k in size
  - I'll call these Type III blocks
  - May split in the middle of functions
  - This is why they need to do instruction length decoding (see Mehta's presentation)
  - Demand paged



# Demand Paging

- Shiva keeps unused memory filled with 0xCC
  - 0xCC = INT 3
  - Jump to empty location or run off end of block generates trap
- In response Shiva decrypts and maps the required page



# Memory Image

- Shiva maintains a page table for Type III crypt blocks
  - Table size is  $\frac{1}{3}$  the number of Type III blocks (min size is 10)
  - For sufficiently large programs no more than  $\frac{1}{3}$  of the program will be decrypted at any given time
  - Random page replacement once table fills



# Other Crypt Blocks

- Type 0 and Type I blocks
  - describe the program's memory layout
    - Abstracted ELF header information
  - A program has 1 of each of these
- Type IV crypt block
  - Master index of on-demand crypt blocks
  - Only one Type IV block as well
  - Decrypted to the heap at startup



# Crypt Block Key Recovery

- Each type of crypt block gets its own key
  - Blocks of same type share the same key
- In this case we need to recover 5 keys in order to decrypt all of the types of blocks



# Key Obfuscation

- Shiva contains a key reconstruction function for each type of crypt block
- Block decryption
  - Identify block type (0-IV)
  - Call appropriate key reconstruction function
  - Decrypt block
  - Clear the key



# Key Construction

- Functions are obfuscated
  - Similar to layer 1 decrypt
  - Differ from one binary to the next
  - Resistant to script based recovery
- But
  - They are easy to locate



# Key Extraction

- ~~Hand trace the functions~~
- Use the plugin to run the functions and collect the keys!
- Demo



# Using the Keys

- With 5 keys in hand it is possible to decrypt all of the crypt blocks
- Each block is identified by a magic number that provides it's type (0-IV)
- All blocks are contiguous
- Drop the keys in an IDA script and run it



# IDA Decrypt Script

- Implements TEA
- Patches original bytes in IDA database
- Unfortunately the IDC language has lousy array support
  - Script is ugly



# Last Line of Defense

- Some instructions replaced with INT 3 traps (software breakpoint)
- When encountered, Shiva emulates them using the ptrace interface
- An emulation record entry is maintained for each such instruction

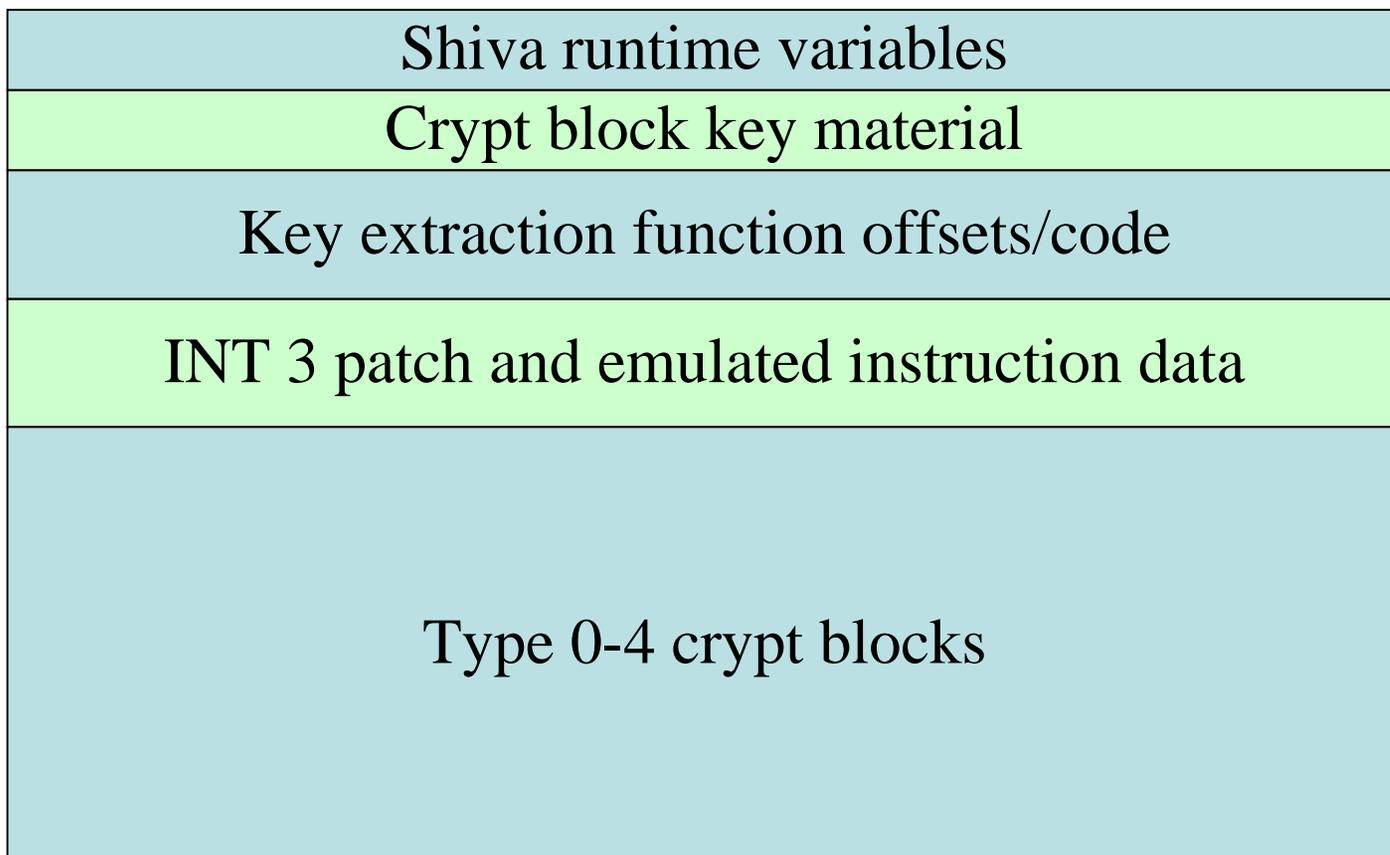


## Last Line of Defense (II)

- We must repair decrypted blocks by restoring these instructions
- Walk the emulation record list to patch over Shiva inserted INT 3 instructions
- Currently emulates
  - PUSH (3 flavors)
  - JMP (2 flavors)
  - CALL



# Block 3 Structure



# Binary Recovery

- Ultimate goal is to recover the original binary
- Decrypted blocks contain
  - Memory layout information (Elf32\_Phdr)
  - Code
  - Data



# Binary Recovery (II)

- Emulation record list contains enough information to repair all code blocks
- Once repaired, ELF headers and segments can be generated to construct an unwrapped binary



# Binary Recovery (III)

- Automated process once the data is pulled out of IDA
  - Automatically patch the INT 3s
  - Automatically generate ELF headers
  - Automatically paste (de)crypt blocks into segments
- Then you get to reverse the recovered binary!

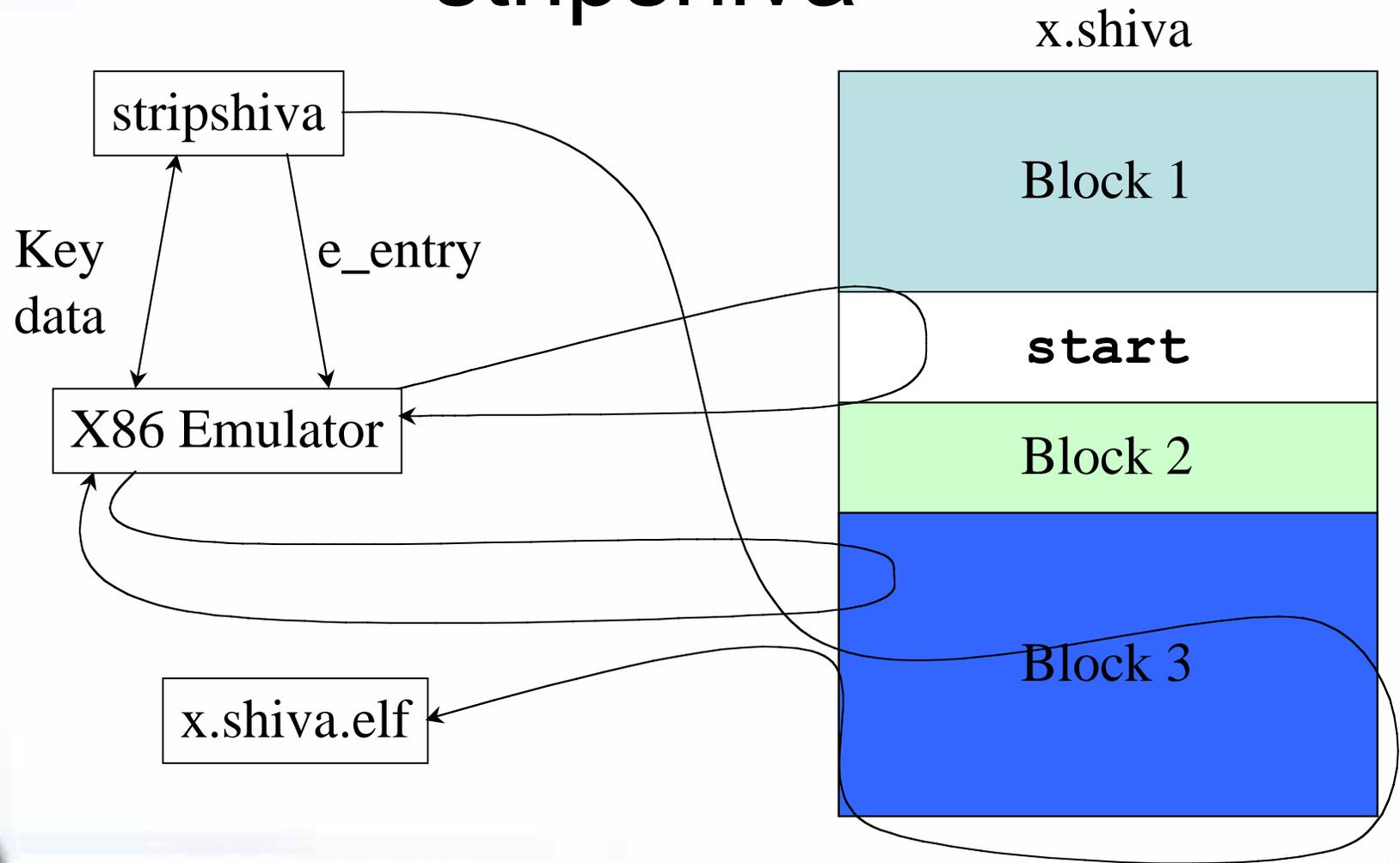


# Full Auto Mode

- Not everyone owns a copy of IDA
- stripshiva
  - Command line tool to remove Shiva protection
  - Contains an x86 emulator
  - Performs all of the steps previously outlined to yield an unprotected binary
  - On your CD



# stripshiva



# Active Analysis?

- /proc fs snapshots over time
  - At best a third of the binary at a time
  - How to stimulate all control paths?
    - Some blocks never paged in
  - Still need to capture emulated instruction data
  - Can't read /proc/<pid>/mem unless you PTRACE\_ATTACH!



# Kernel Module Approach

- Load module
- Walk process list
  - Look for Shiva characteristics
    - 0x0A048000, checkme
- Dump data segment to file
- Use stripshiva to recover binary from dump file



# Kernel Module Approach (II)

- Advantages
  - Bypasses /proc defenses
  - Only way (without brute forcing) to recover password protected binaries
- Limitations
  - Must keep process alive long enough to insert lkm



# Outline

- Introduction
- Runtime encryption tools
- Shiva review
- Reversing Shiva
- Summary



# Other

- Performance Impact of Shiva
  - Paging/decryption overhead
  - ptrace/emulated instruction overhead



# Summary

- Recovery of Shiva protected binary is possible
- Can be done with static analysis tools only
- You may hate Windows, but you've got to love IDA Pro!



# Questions?

- Thanks for coming
- Contact info:
  - Chris Eagle
  - [cseagle@nps.navy.mil](mailto:cseagle@nps.navy.mil)



# References

- Armouring the ELF: Binary encryption on the UNIX platform, grugq & scut,  
<http://www.phrack.org/phrack/58/p58-0x05>
- Shiva: Advances in ELF Runtime Binary Encryption, Clowes & Mehta, Black Hat USA 2003,  
<http://www.blackhat.com/presentations/bh-usa-03/bh-us-03-mehta/bh-us-03-mehta.pdf>



# References

- Shiva-0.96, Clowes & Mehta,  
<http://www.blackhat.com/presentations/bh-usa-03/bh-us-03-mehta/bh-us-03-shiva-0.96.tar>
- Burneye-1.0.1, scut, <http://teso.scene.at/releases/burneye-1.0.1-src.tar.bz2>
- IDA Pro, Data Rescue,  
<http://www.datarescue.com/idabase/>

