Thwarting Themida:
Unpacking Malware with SMT Solvers

8 May 2013

Authors and thanks

- **Ian Blumenfeld**
- **Roberta Faux**
- **Paul Li**

Work overseen by **Mark Raugas** – Director CyberPoint Labs

Special thanks to **Levent Erkok** for technical help with the SBV library

CyberPoint is a cyber security company.

We're in the business of protecting what's invaluable to you.

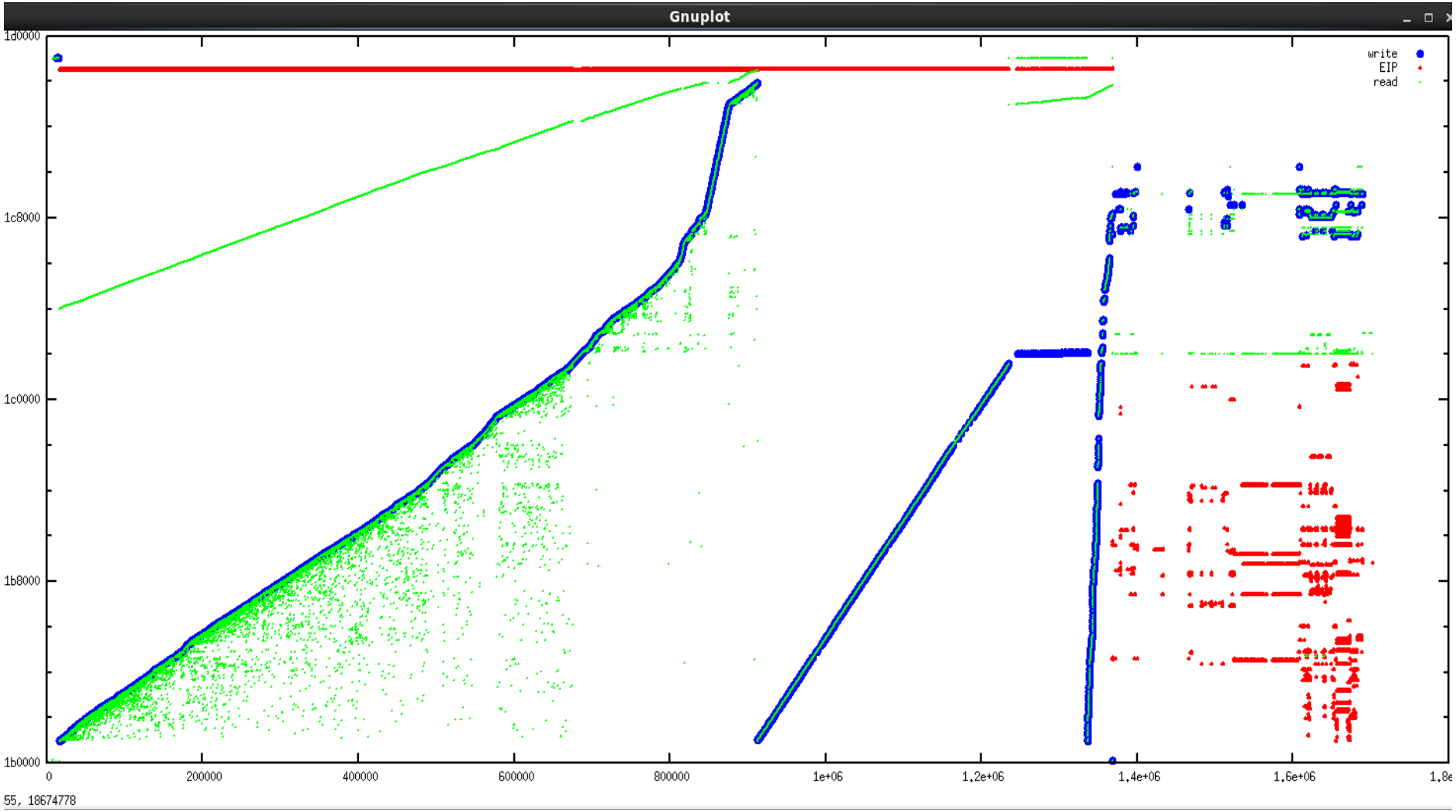
Malware analysis

- We must understand the behavior of a piece of malware.
- Obfuscation techniques make manual analysis time-consuming.
- Skilled malware analyst time is expensive.

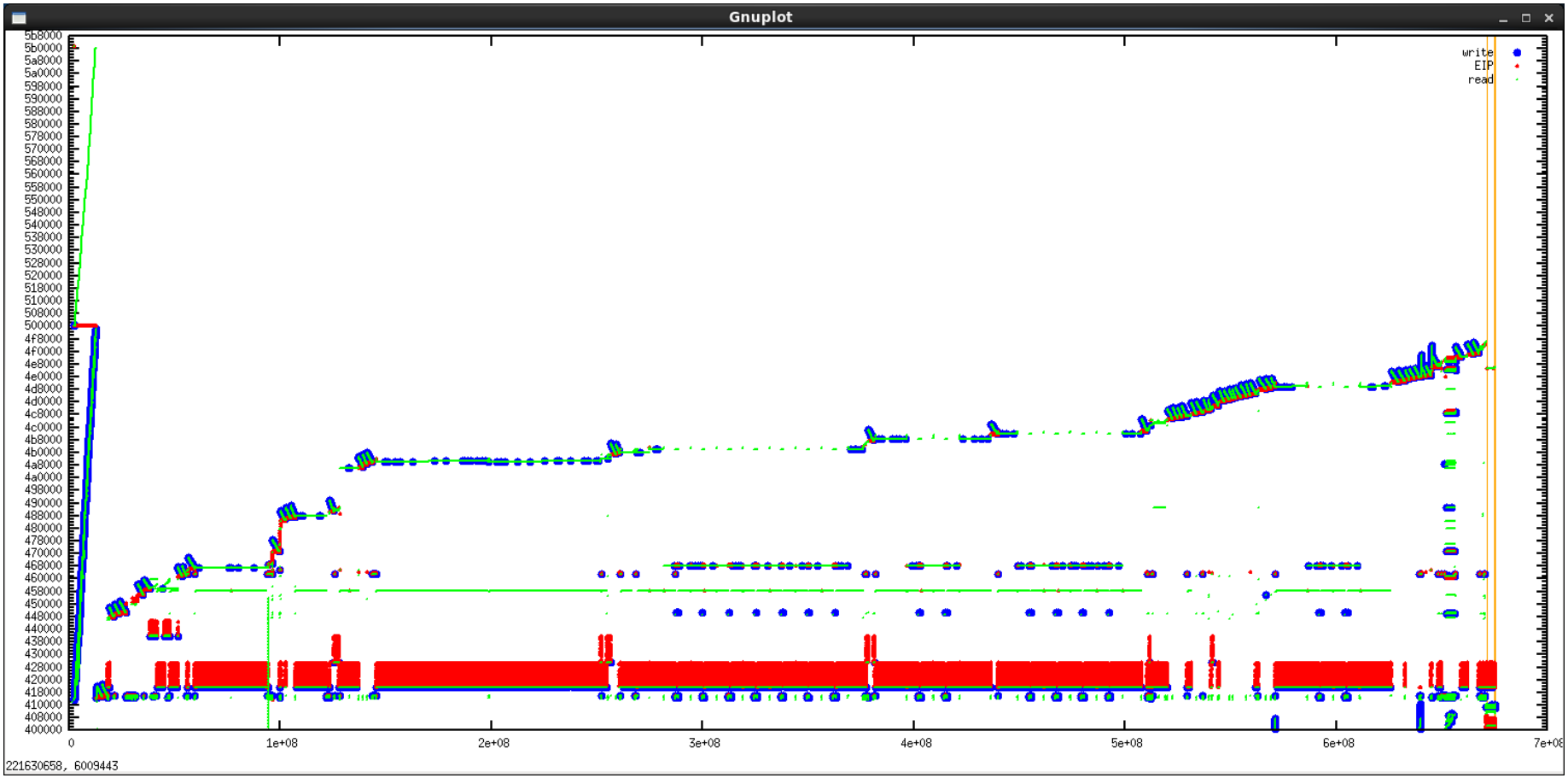
Binary packers

- Transform a binary program into a new program that has the same functionality
- Insert unpacking code that restores the original code in memory and then jumps to it
- Use additional techniques to deter analysis
- Range from freeware to expensive commercial products to custom packers written by malware authors

A simple packer: UPX



A harder packer: Themida



Virtualization

- Themida embeds a virtual processor within the packed binary.
- Certain instructions are converted to byte-code for that processor.
- Each byte-code instruction is interpreted by a handler.
- Even the handlers are obfuscated.
- Different runs of the packer result in different handlers.

A (simple) handler

```
push dword 0x4a99
mov [esp],ebx
push esp
pop dword [esp]
push edx
mov edx,0x4
add [esp+0x4],edx
pop edx
push dword [esp]
pop edx
push edx
mov [esp],ebx
mov [esp],esi
mov [esp],ecx
mov ecx,esp
add ecx,0x4
push dword 0x504a
mov [esp],ebx
mov [esp],eax
push ecx
mov ecx,0x4
mov eax,ecx
mov ecx,[esp]
add esp,0x4
add ecx,eax
mov eax,[esp]
push esi
mov esi,esp
add esi,0x4
add esi,byte +0x4
xor esi,[esp]
xor [esp],esi
xor esi,[esp]
mov esp,[esp]
xchg ecx,[esp]
pop esp
```

Complications

- Handlers may be much more complex than the example, and not end in the real instruction.
- Some handlers run on byte-code encrypted with a special constant.
- Obfuscation techniques for handlers may be random and handlers are repeated.

Symbolic simulation

- CyberPoint has written a Haskell library to symbolically simulate handlers.
- We use the open-source SBV library to provide a generic API to hook our code into SMT solvers.
  - CVC4
  - Z3
  - Yices
  - Boolector
- We model our machine using bit-vector and uninterpreted function theories
- We can use SMT to prove properties about the handler code.

Deobfuscating simple handlers

- Compile a list of reference handlers:
  - Homegrown rewriter
  - Manual analysis
  - Examples from the literature
- Use trace tools to locate handlers in the virtual processor
- Use symbolic simulation and SMT solver to try to prove equivalence of an isolated handler to each reference handler

Results on simple handlers

- On purely symbolic machine state identification takes 5 minutes.
- By intelligently concretizing certain values to reduce to candidates, simpler handlers can be identified in about 5 seconds.
- All results make use of the embarrassingly parallel nature of the problem.

Obfuscation constants

- Constants are used to “encrypt” using a variety of methods
- Once method is obtained, can use SMT for function inversion to discover the constant
- Preliminary success in simple handlers that use constants, though all constants recovered so far can also be found through other means

Stolen code

- Understanding which API calls are made is critical information for a malware analyst.
- Sometimes the packer will "steal" the first several instructions from an API function. It:
  - Obfuscates them
  - Inserts them into the caller
  - Eventually jumps to an address to get it back to the API function
- Wastes analyst time on a normally simple part of their task

Stolen code example: original API function

```
;; kernel32!InterlockedIncrement
776fc3b0: 8bff            mov edi, edi
776fc3b2: 55              push ebp
776fc3b3: 8bec            mov ebp, esp
776fc3b5: 5d              pop ebp
776fc3b6: eb88            jmp 0x776fc340
     .                         .
     .                         .
     .                         .
```

Stolen code example: obfuscation in the caller

```
10a00000: 8bff          mov edi, edi
10a00011: 95            xchg ebp, eax
10a00012: 50            push eax
10a00013: 52            push edx
10a00029: 0f31          rdtsc
10a0002b: 60            pushad
10a0002c: 8bca          mov ecx, edx
10a0002e: 50            push eax
10a0002f: 52            push edx
10a00030: 0f31          rdtsc
10a00032: 5a            pop edx
10a00033: 58            pop eax
10a00034: 61            popad
10a00035: 5a            pop edx
10a00036: 58            pop eax
10a00037: 50            push eax
10a00038: 50            push eax
10a00039: 52            push edx
10a0004f: 0f31          rdtsc
10a00062: 5a            pop edx
10a00063: 58            pop eax
10a00064: 95            xchg ebp, eax
10a00073: 8bec          mov ebp, esp
10a00075: 50            push eax
10a00076: 52            push edx
10a00077: 60            pushad
10a00078: 66bb707c      mov bx, 0x7c70
10a0007c: e814000000    call 0x10a00095
10a00095: 58            pop eax
10a00096: 61            popad
10a00097: 0f31          rdtsc
10a00099: 60            pushad
10a0009a: 50            push eax
10a0009b: 5f            pop edi
10a0009c: 0fb7cb        movzx ecx, bx
10a0009f: 61            popad
10a000a0: 5a            pop edx
10a000a1: 58            pop eax
10a000a2: 5d            pop ebp
10a000a3: 60            pushad
10a000a4: 8bc7          mov eax, edi
10a000a6: 9c            pushfd
10a000b4: 80dc56        sbb ah, 0x56
10a000c8: 9d            popfd
10a000c9: 61            popad
10a000ca: e9e7c2cf66    jmp 0x776fc3b6
```

Results on stolen code

- If we know a priori which code is stolen, we can loop over possible API functions to prove equivalence
- For full generality, we would need to figure out how many instructions from the API calls are stolen
- Proofs do work, but limited advantages over other techniques
- Symbolic termination can be a problem

Moving forward

- Given the initial successes, we should do real comparisons of SMT based approaches with others?
- Can we use our work to actually rewrite the binary to a deobfuscated form?
- How can we expand the work on Themida to other packers:
  - VMProtect
  - Enigma
  - Custom packers
- What other role can SMT solvers play in the world of cybersecurity?

Questions?

Thank you,
have a great day!


cyberpoint