Why Don’t You Just Tell Me Where The ROP Isn’t Supposed To Go

David Dorsey
@trogdorsey

Who’s this… guy

- 10 years on the defensive side
- File analysis & RE
- Recently doing research using Machine Learning

Level Setting

- ROP
  - Technique to bypass non-executable memory
  - Bounce around in memory executing small gadgets that typically end with a return instruction
- PIN
  - Pin is a dynamic binary instrumentation framework for the x86 and x86-64
  - Does not require recompiling of source code and can support instrumenting programs that dynamically generate code

Basic Idea

- A whitelist for offsets that can be a target of indirect branch or ret
- We know valid targets for calls and rets
  - Functions
  - Instructions after call instruction
- If an indirect call or a ret goes to a different location, then ROP
- Store the offset to these locations

How Do We Get Those?

- BranchTargetDetector pintool
- When DLL is loaded, exported functions are analyzed
- All calls and returns are instrumented as well
- Great because we get actual values
- Not so great because you only get values from functions pin can detect and what it actually executes

BranchTargetDetector

- Pros
  - We get real, actual used values
- Cons
  - Not the fastest thing
  - Only get values from functions pin can detect and what it actually executes
  - If DLL isn't loaded, you don't get data for it

How Else Can We Get Those?

- pyew
- Much better at detecting functions
- Can bulk run all DLLs

Have Data, Now What?

- Store offsets in file by md5 hash of dll
- Allows for handling of different versions of the same dll

ROPDetector

- When a DLL is loaded, load the white list for that DLL
- Instrument all indirect calls and RETs and alert when target is not on the white list

Example 1

- Adobe Reader 9.3 on Windows XP
- 32dbd816b0b08878bd332eee299bbec4
- CVE-2010-2883
    - Stack-based buffer overflow in CoolType.dll

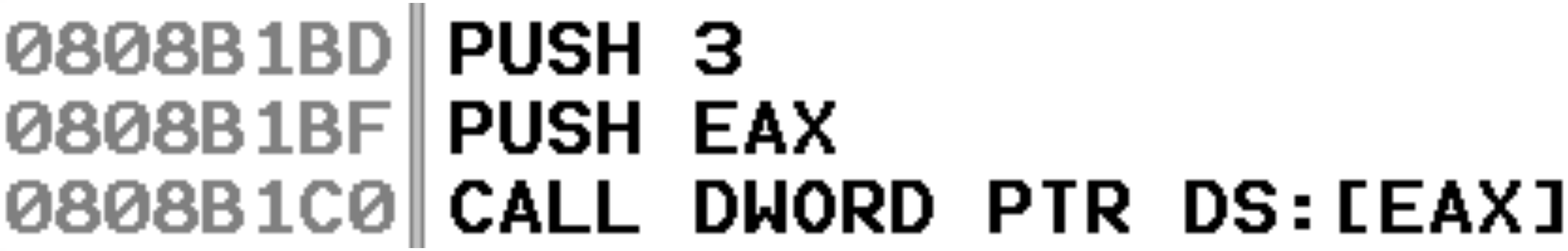
Detection!

```
C:\Program Files\Adobe\Reader
9.0\Reader\icucnv36.dll
0x4a80cb3f: ret
Target: 0x4a82a714 (0x2a714)
```

Yay?

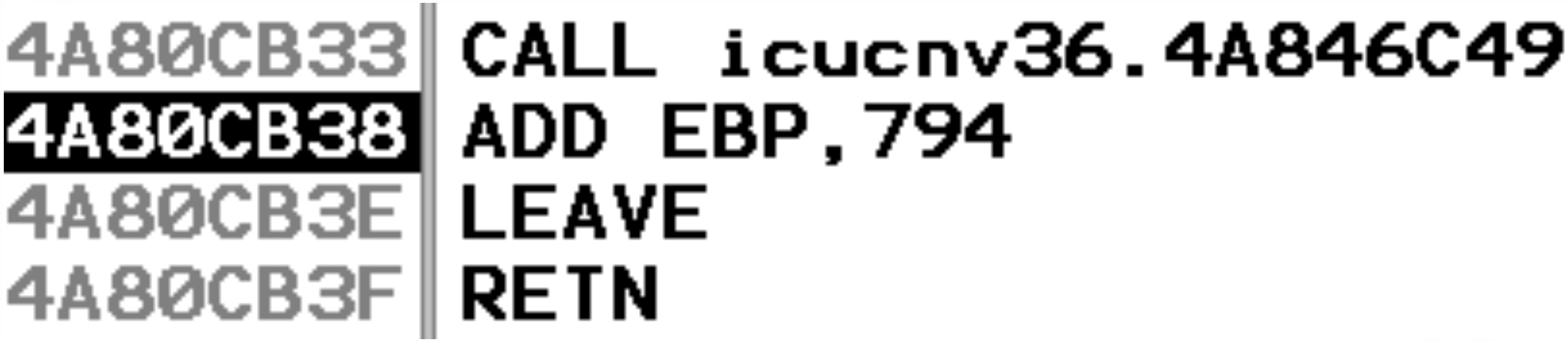
- We detected one of the ROP chains
- Only 1

Let’s Take A Look



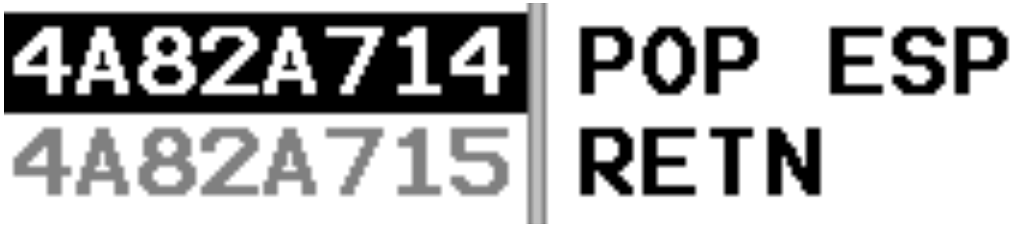
```
0808B1BD | PUSH 3
0808B1BF | PUSH EAX
0808B1C0 | CALL DWORD PTR DS:[EAX]
```

Let’s Take A Look



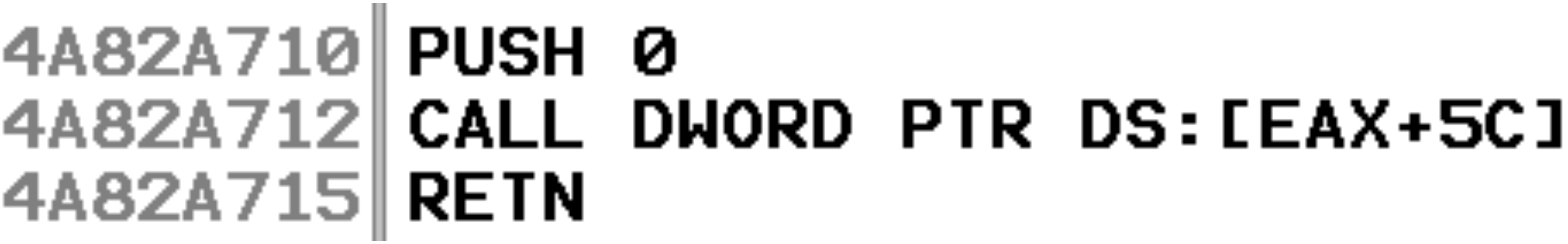
```
4A80CB33  CALL icucnv36.4A846C49
4A80CB38  ADD EBP,794
4A80CB3E  LEAVE
4A80CB3F  RETN
```

Let’s Take A Look



```
4A82A714 POP ESP
4A82A715 RETN
```

Let’s Take A Look



```
4A82A710 PUSH 0
4A82A712 CALL DWORD PTR DS:[EAX+5C]
4A82A715 RETN
```

Why Only One?

- Dies on stack pivot
- Pin affects memory layout
  - (run everything in pin?)

How Would We Have Done?

- 45 chains in ROP sequence
- Only 14 unique addresses
- 2 indirect calls, 43 returns
- 3 of the 14 addresses on whitelist
  - Each address only called once
- 42 of 45 chains would be detected

Example 2

- Adobe Reader 9.5 on Windows XP
- 6776bda19a3a8ed4c2870c34279dbaa9
- CVE-2013-3346
  - ToolButton Use After Free

Example 2 Results

- Nothing, just Adobe crashing
- Pin messed up memory layout again

The Neighborhood Of Make Believe

- 208 chains in ROP sequence
  - Dominated by 191 chain sled
- Only 15 unique addresses
- All returns
- 3 of the 15 addresses on whitelist
- 204 of 208 chains would be detected

A Little Math

- Probability of detecting at least one address (assuming 11/14 detections is average)

Unique Addresses	Probability of Detection
1	78.6%
2	95.4%
3	99.0%
4	99.8%
5	99.96%
10	99.999980%

A Little More Math

- Probability of detecting at least one address (assuming 50% detection rate)

Unique Addresses	Probability of Detection
1	50.0%
2	75.0%
3	87.5%
4	93.8%
5	96.9%
10	99.9%

Limitations

- Pin
  - Breaks on stack pivot
  - Slow
- Doesn't handle Jump Oriented Programming (JOP)

Possible Improvements

- Smarter instrumentation
- Push analysis into a different thread
- Figure out heap problem
- Check for JOP

Smarter Ways

- Debugger?
- Detours?
- Monitor Last Branch MSRs?
  - kbouncer

Thanks!

- https://github.com/trogdorsey/rop
- https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool
- https://code.google.com/p/pyew/
- http://www.cs.columbia.edu/~vpappas/papers/kbouncer.pdf