BlockFighting with a HOOKER

Shane.Macaulay@IOACTIVE.com

BLOCKFIGHTER II (@DEFCON24 #w00w00 #Blah!)

(K2) ktwo@ktwo.ca
What are we doing here?

- Methodology & tools for high speed comprehension binary trace/analysis/steering
  - Super simple to use and FUN!
  - Monitor and Alter execution
- BlockFighters
  - ROP DEFENDER
- RAN$OM E$CROW
  - I HAVE A CERTAIN SET OF SKILLS
- EhWinAFL
  - Almost defiantly not the best backend ever ;(
- Hyepervisor DoS / detection!
- Execution Graph tools
  - Block views  Also + disassembly  FLAME GRAPH!
Intro/Outline

• Hooking/Tracing
  • Trace: What is executing?
  • Hooker: Can we also modify/detour flow?

• Frustrations/Hurdles
  • What worked, what didn’t work, what also works ;)

• Friendly inputs
  • Symbol support

• Evil inputs
Intro: Current & OTHER / TOOLS / CODE

- `https://Github.com/K2`
  - `Github.com/ShaneK2/inVtero.net`
    - Evolution from CSW14 (process detection)
    - Cross platform (Windows, *BSD, Linux) memory analysis
    - Cross microarchitecture (sandy bridge, sky lake, ...)
    - Cross hypervisor (based on auto-magic VMCS / EPTP extraction)
      - Includes nested support (D33P introspection)

- `EhTrace` (pronounced “A Trace”)
  - What were going to cover this time: `DEFCON24!!! <<<⬅️=----------------------`

- Let me know if I missed any code in the check-in!!
Goals

• Trace or Modify execution
  • Bare metal
    • Can run under a hypervisor

• Binary steering
  • This method (EhTrace) does not require multiple executions
    • Using profiling hardware supported logging is faster at run time yet slower in some cases due to requiring us to re-run our target

• EhTrace in a DBI – for fuzzing?
  • Maybe in the cloud or malicious binaries
  • Benign binaries check out Richard Johnson’s work “Go Speed Tracer”
  • Let’s see what AFL does in any event (hohooood)
Dreams

• Fun sandbox that is flexible and easy to play with
• Lots of blockfighters
  • Swiss Army stuff
  • Defend against mal-code
  • Attack to understand anything
• Uber perf
  • Slicing is your friend
• Kernel look
Dependencies

- I use Windows 10 x64
- Cloud / older versions YMMV
- Thanks Feryno
  - http://x86asm.net/articles/backdoor-support-for-control-transfer-breakpoint-features/
- https://github.com/K2/EhTrace/wiki
  - Thanks InGap Jeong (laughfool)
  - http://fdbg.x86asm.net/add_debugctl_support_ws2008R2_w7.UEFI.BIOS.ver048.zip
  - For detail information's check below link. http://fdbg.x86asm.net/debugctl.article.txt
More References

- Danny Quist, Valsmith – DEFCON15
  - Covert debugging / vm / debugger detection & countermeasures
  - Page fault assisted “Saffron”

- Super modern stuff => [http://triton.quarkslab.com/](http://triton.quarkslab.com/) (seems like ideal design! 😊?..
  hosted on github, (also since lots of darpa challengers do binary trace... Trail Of Bits GRR & tools)
  - Speed taint,
  - dynamic symbolic execution,
  - replay trace’s with snapshot,
  - SMT/constraint solvers, AST logic,
  - tracer independent & python
Traditional Trace Techniques

• If hooking -- disassembly required
  • Insertion of a “detour” we need to know how to insert a trampoline
    • Stack stuff “arguments(A,B);”
    • Replicate instructions

• Slow
  • Debuggers – EhTrace is sort of an in proc debugger
  • Less context switching and avoids having to use based pointers everywhere 😊

• Disassembly needed for some circumstances, not explicit requirement
  • Thanks to capstone this is not a hard problem
    • Perf is an ongoing thing
Hooking execution

- Detours
  - Requires an instruction length decoder
  - Rewrites function prolog into a specialized function which performs logging, analysis etc...

- Usually static, can be dynamic/jitter,
  - may jmp to a leaf like detour which can work without knowing the function prototype/stack requirements

- Most of the time you will need symbols or really good logic in the hooker to not break execution

- Perf not perfect since were blowing shared cache, adding code and may require more
What’s the problem again?

- Debuggers are slow, really slow
  - Second process context switching is fairly expensive
  - Logic for conditional breakpoints is exponentially more expensive

- Being detected by EVIL c0d3

- Checksums
  - Malicious binaries often checksum their code to validate they are not being analyzed
  - Highly secure environments may checksum their binaries to make sure they are not tampered with
Perf

• Cost is mostly setting up the exception pump
  • logging(trace), RoP defender & Key Escrow very cheap on top of A pump ;)

• Microbenchmarks show between 20-150% performance hit
  • TODO: Cache / Checkpoint implementation
    • Dynamically turn on/off depending on needs
    • Detect self-modifying code & otherwise adversarial stuff
  • TBH un-sliced hookers looking at 1000% TOTAL worst case, hey one order of magnitude is better than 2+!!

• Theoretically can execute faster than native execution
  • Sort of the purpose of some of these trace interfaces to accelerate slow code or to bypass calls which can be simulated in a mem-cache
    • Eternal Space/Time trade off exercise left to an exercise by the reader
• Aboot time for a trace eh?
  • Whatever I’m dual citizen

• Uses VEH under the covers
  • Need to be a little careful
  • Don’t want to alter or change
    behavior of what we’re looking at
Ret2 code

• Original libc work, Solar designer
  • http://seclists.org/bugtraq/1997/Aug/63

• Handy since most overflows contain a pointer to useful addresses
  • Your input
  • System libraries

• Still used to this day (RoP)
Stack Hooking

- Attempted to use as alternative to what we wound up using
- From a second “manager” thread
  - Load from a RoP chain pool (memory area with RoP gadgets)
  - Borrow memory from the executing stack from above the stack top
    - Usually some spare memory there
- Not very great
  - Only post condition hooking
  - Have to find a way to get notification on new calls
    - Do some sort of shadow stack/memory protection trickery
    - Tends to be fairly fragile
EhTrace – how it works

LONG WINAPI vEhTracer(PEXCEPTION_POINTERS ExceptionInfo) {

    ExceptionInfo->ContextRecord->EFlags |= 0x100;
    ExceptionInfo->ContextRecord->Dr7  |= 0x300;

    // Remarkably easy to trigger branch stepping of a binary
    // In the VEH handler set 3 bits and return.
    // THAT'S IT
    // TRAP FLAG
    // OTHER FLAGS :D
EhTrace – RoP Hooks

- Register a VEH handler `CreateRemoteThread(... &VeH_RoP,..);`
  - `VeH_RoP` – use a RoP gadget finder (there are many)
  - Handler only needs to set the 3 bits then exit with continue status

- Using the exception dispatcher were able to now get the preconditions we missed with the stack/shadow model

- Pretty straight forward, just need to maintain control in flags since it’s cleared out of the context.
What else is it good for?

- Branch stepping is pretty sweet!
  - A lot more than detours on functions
  - Basic block analysis
  - Code coverages
  - Can we put this into a DBI (Dynamic Binary Instrumentation) framework?

- Do we need to emulate? Isn’t that slow?
  - If were dealing with a malicious binary we have several things to consider.
  - Of course we need to also watch out for an otherwise non-mal binary doing something that might disrupt our trace
Maintaining control

• Maybe use page protection to force an exception on execution (don’t want to place an int3 obviously)
  • When page is attempted to be executed we check to see what emulation is needed
    • If somebody tries to take over VEH

• What about intra-block stuff?
  • Can’t they just write over our VEH handler in memory?
  • Sure, maybe register 2! Also setup the VEH continue handler

• Do some hybrid stack rewriting (inject LOP’s) + EhTrace to steer and manage target binary
Blockfighting with a hooker

- BlockFighter has to be smart, fast and in total control!
- Much like a StreetFighterII champ!
K2: Blockfighting with a hooker... or so it goes.

Leverage block stepping to simplify program introspection such that we may analyze code coverage and execution in detail.

Sort of a self-debugging that is fast and able to operate without any code patches, traditional hooks/detours so that check summing will match as needed.

In a way feels like code-wars/core-wars in a way.
Initial release is targeted towards friendly binaries.

blockfighting defenses? async code async completion, defend setthreadcontext, Use page protection to detect/defend execution of code? redirect attempts to start new threads to ensure that they start suspended and are able to be tracked before execution xsave scanning pop/ret exception handling user callbacks *callbacks threads & fiber create
BlockFighting

• Simplified analysis
  • Using capstone we & the branch step
  • At the point of any jmp/ret/call control transfer we can stop our fight until the next round
    • Round 2 FIGHT!
    • Actually were so good we always “give second round”!
    • That means really that if there’s a conditional we need to follow through a conditional
      • Jne – we follow the non-jump to ensure we complete the context until a ret/jmp/call

• Eventually add [RJL]oP engine and things get a lot more easy with binary steering... (perf good, nearly native speed, drop most exception overhead! 😊)
BlockFighting

- Watch the eflags & DR any manipulation will cause problems for us
- DEBUG_MSR ?
- Lots of things probably
- Overall however we have a platform to build primitives on that can eventually do battle in a structured way
  - Maybe combine blockfighter with stack injection to ensure we have additional post-condition checks on our flag/branch-step/veh state
Ransom Warrior

- Enforce cryptographic key escrow
  - Trace the binary
    - Escape random read’s => network, protected enclave / hypervisor assisted
  - Prototype block fighter can expand into a more refined set of interfaces
    - ROP / JOP / LOP Building got to love the lop op – LOP!
Coverage

• Can you hear me now?

• Flame graph
  • Current minimal state includes RIP, LAST_RIP, TID, FLAGS and ESP
  • This is sufficient to build any code graph! **Intra-procedural, call graph or full trace**

• **FLAMING BlockFighter!**
  • [http://www.brendangregg.com/FlameGraphs](http://www.brendangregg.com/FlameGraphs)
Execution history - Blocks
Execution history – With ASM
CPU FLAME GRAPH

- CLICK HERE
- Orig from here -&gt; http://www.brendangregg.com/FlameGraphs/cpu-bash-flamegraph.svg

(PowerPoint doesn’t do SVG’z i.e. SVG is navigable)
CPU FLAME GRAPH

- https://github.com/K2/EhTrace/blob/master/support/x1_100k.png
Upcoming stuff...

• MSAGL graphmaps – fun/interactive mesh graph, sort of looks like an expandable spiderweb!

• SVG builder (without the .pl scripts from Brendan)

• Tighter Symbols (graphs and images not as fun without English eh?)
Upcoming stuff: Blockfighters

• A Flagfighter
  • Rflags checks

• A PageFighter
  • Page protection monitor
  • E.g. protect the entrypoint CreateRemoteThread call’s before it calls the specified &func argument to detect remote threads before the DLL thread notification run’s
    • Use tricks like this to ensure your not being tricked yourself
  • Page fighter should be slicing the input based on what you want to trace (i.e. manage trace A B or C .DLL and leverage page protection as

• Emu Fighter
  • Emulate an operation that would otherwise detect us
Private implementations differ!

- Your fighters will be various
  - i.e. if your not using any system/runtime API you don’t need to worry about locking as much (obviously)
Questions -- DEMOz?

- Feedback, bugs & Feature requests please

- [https://github.com/K2](https://github.com/K2)
Thank you