APATE Interpreter - A Kernel Hook Rule Engine

Christoph Pohl [1,2,3]   Michael Meier [3]   Hans-Joachim Hof [1,2]

[1]MuSe - Munich IT-Security Research Group

[2]University of Applied Sciences Munich

[3]Fraunhofer FKIE / University of Bonn

July 2, 2015

ToC

1 Introduction
2 Related Work
3 Proposed Solution
4 Evaluation?
5 What is the research impact?
6 Conclusion
7 Questions
8 References

Introduction to APATE

APATE Interpreter is just a small part of APATE Kernel Module
The APATE Kernel Module is a sub part of:

Field of Research

Decoy and Monitoringstrategies for Linux Systems with VMI, Kernelmodules, adapted uLibC and BusyBox

Features of APATE

[1] Pohl, C., Hof, H. J., and Meier, M.
Apate - A Linux Kernel Module for High Interaction Honeypots.
In *SECURWARE 2015, The Ninth International Conference on Emerging Security Information, Systems and Technologies* (Aug. 2015).

Features

- Change the behavior of Linux Kernel 3.x with a fast rule engine.
- Rules can have any parameter from kernel as input
- APATE is able to build logs upon the rules
- APATE has some decoy strategies to prevent timing attacks, debugging and forensics
- Rules can be built with a high level turing complete language
- APATE Kernel is part of Kernel or LKM

Function Manipulation

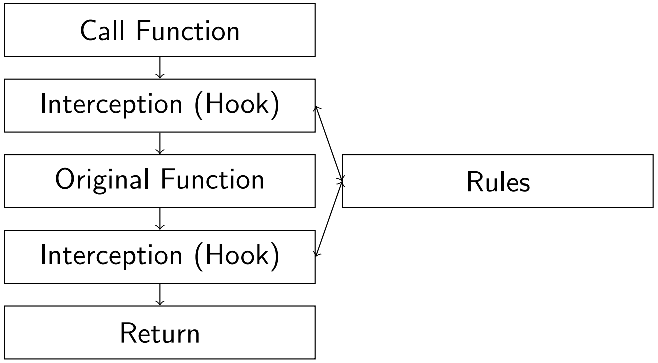

Figure. Conceptual overview for function manipulation (just a hook)

Rule Generation

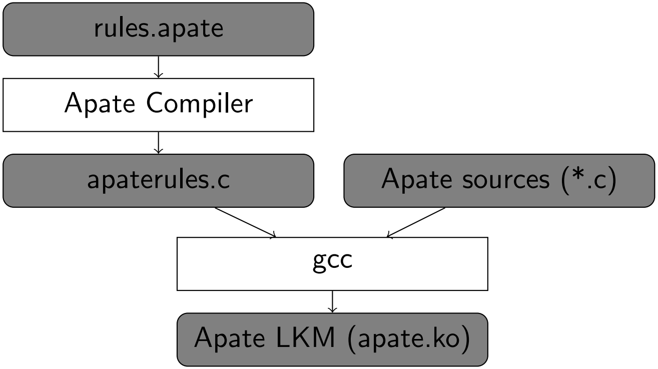

Figure. Rule generation in APATE Kernel for LKM

Adressed Problem

Adressed Problem within APATE Kernel

- APATE Kernel compiles RULES with Flex and Bison to C
- Kernel needs recompilation (or recompilation from LKM)
- Rules can not be injected on the fly
- Kernel Hotpatching over VMI is not suitable
- Anti Forensics (or dirty tricks) are hard to integrate in Compiler
- Regex Rules or iftable rule engine and other stuff are not suitable

Some example I

```
define c1,c2,c3 as condition
define r1,r2 as rule
define a1,a2 as action
define cb1 as conditionblock
define rc1 as rulechain
define sy1 as syscall

let c1 be testforpname
let c2 be testforparam
let c3 be testforuid
let a1 be manipulateparam
let a2 be log
let sy1 be sys_open

let cb1 be {(c1("mysql") && c2(0;"/var/lib \
```

Some example II

```
    /mysql/*"))}

let r1 be {cb1->a1(0;"/var/lib/mysql/*" \
   ;"/honey/mysql/")}
let r2 be {{c3(">",0)}->a2()}
let rc1 be {r2,:r1} // :defines exit

bind rc1 to sy1
```

In words: If process is mysql and path of syscall wants to have */var/lib/mysql/** then redirect to */honey/mysql/**. Whenever the user is not root, log the syscall. Process the first rule first and then the second one, despite the first rules matches or not. Be aware that this is transparent to the user.

Research Question

Research Question

- How to generate a rule engine in kernel space to manipulate kernel behavior and sensors
- How to implement it in a performant way
- How to enable the rule engine to generate rules on the fly
- How to inject rules from host
- How to hide the engine from disassembler and debugger

Related Work

- [6, 2] Sebek, just logging and no manipulation
- [19, 9, 8] VMI-based approaches, just logging and no manipulation
- [18] SE-Linux, Hooking, Manipulation and Logging (even some rule engine), but not designed for honeypot purposes, not able to hook everything, focused on hardening (opposite target), no decoy functionalities. Language is not turing complete.
- [13] GRSecurity with PAX [15], somewhat the same than SE-Linux

Proposed Solution

Generate an interpreter for hook and sensor behavior

Solution

- The interpreter is a runtime-like engine
- The language is assembly like (or something like bytecode)
- The high level language gets compiled into APATE assembly
- There are special machine intructions for hooking, decoyness, sensors and further more
- The assembly can be fragmented over real memory
- The runtime can use real and virtual (own) adress space

Conceptual Overview

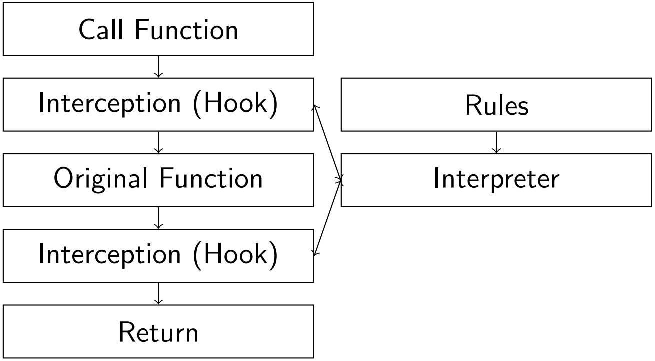

Figure. Conceptual overview

Rule Generation

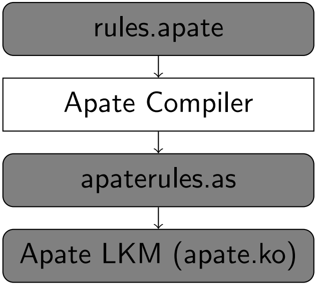

Figure. Rule generation in APATE Kernel for LKM with Interpreter

The Interpreter

Description Interpreter

- Register Based Interpreter
- Adressable Stack (in fact a register)
- Fixed width intruction set
- Shortcut instructions
- Hotpatchable over VMI
- Interpreter can be fragmented

The Interpreter

Description Interpreter Register

- Interpreter has 12 fixed size Register
- 0 – 6 just like eax,ebx...(64 Bit)
- 7 last adress (like eip) (64 Bit (can be shared for kernel memory space))
- 8 – 11 bitmaps for locking, real/virtual memory . . . (8 Bit)

Whenever the register are "out of space", the interpreter uses the adressable stack

The Interpreter

Description Interpreter Stack

- Interpreter has fixed adressable stack (12 Elements)
- Each stack element has 64 Bit
- Stack can grow until kernel memory limit
- Stack can not shrink (despite one uses the dealloc instruction from interpreter)

The Interpreter

Instruction Set

- Any instruction has two params $< opcode >< dest >< src >$
- There are about 20 opcodes (+shortcuts)
- p.ex j(u)mp, p(u)sh, p(u)ll, c(o)mp(are)

The Interpreter

Shortcuts

- A shortcut is a special opcode p.ex. for decision making in rules
- Other shortcuts are for logging purposes
- Other shortcuts are for convenience or speed

Hot Patching

Hot Patching

- Assembly Code can be “patched” over VMI
- Code can be hidden in “lost” data segments of real assembly
- APATE - Assembly can be generated by the interpreter on the fly

What about evaluation?

What about evaluation?

- APATE Kernel works quite well
- Interpreter is under development
- I can only measure some low level functions of the interpreter (makes no sense to measure them)
- At moment each part of interpreter works more or less well, but it is not combined yet

What can you do with it?

What can you do with it?

- You can change the behavior of kernel functions during runtime
- You can log everything (Apate has different logging functionalities)
- You can build rules which can build rules or code themself (for example to react on user input or to enable/disable sensors)

What is the research impact?

What is the research impact?

- It is the first honeypot interpreter for a Linux Kernel (OK thats just engineering)
- Some new technologies for anti forensics (Playing with IDA Pro), usable to decoy honeypots, lure debuggers and even disassembling tools
- Usable as base framework or to manipulate the kernel during runtime, based on user behavior to enable/disable sensors or function behavior

Conclusion

Conclusion

- With a low level runtime it is possible to manipulate the Kernel behavior and sensors on the fly
- It is possible to do this in a performant way (first results)
- It is possible to generate rules with another rules during runtime
- It is possible to harden this against disassembling and debugging

Any Questions?

Any Questions

Say: “OK Google”...

Email: c00clupea@gmail.com

References I

[1] Balas, E.
Sebek: Covert Glass-Box Host Analysis.
;login: THE USENIX MAGAZINE, December 2003, Volume 28, Number 6 (2003).

[2] Dornseif, M., Holz, T., and Klein, C.
NoSEBrEaK - Attacking Honeynets.
In *Proceedings of the 2004 IEEE Workshop on Information Assurance and Security* (June 2004).

[3] Fox, M., Giordano, J., Stotler, L., and Thomas, A.
Selinux and grsecurity: A case study comparing linux security kernel enhancements.

[4] Garfinkel, T., and Rosenblum, M.
A virtual machine introspection based architecture for intrusion detection.
In *In Proc. Network and Distributed Systems Security Symposium* (2003), pp. 191–206.

[5] Herley, C.
Security, cybercrime, and scale.
Communications of the ACM 57, 9 (Sept. 2014), 64–71.

[6] Holz, T., and Raynal, F.
Detecting honeypots and other suspicious environments.
In *Proceedings from the Sixth Annual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop, 2005.* (2005), IEEE, pp. 29–36.

References II

[7] Honeynet Project.
Know Your Enemy: Sebek.

[8] Jiang, X., and Wang, X.
"Out-of-the-Box" Monitoring of VM-Based High-Interaction Honeypots.
In *Recent Advances in Intrusion Detection* (2007), Springer Berlin Heidelberg, pp. 198–218.

[9] Lengyel, T. K., Neumann, J., Maresca, S., Payne, B. D., and Kiayias, A.
Virtual machine introspection in a hybrid honeypot architecture.
In *CSET'12: Proceedings of the 5th USENIX conference on Cyber Security Experimentation and Test* (Aug. 2012), USENIX Association.

[10] Ligh, M. H., Case, A., Levy, J., and Walters, A.
The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory.
John Wiley & Sons, 2014.

[11] Marlow, S.
Haskell 2010 language report.
https://www.haskell.org/onlinereport/haskell2010/, 2010.
Visited 25.02.2015.

References III

[12] NSA (Initial developer).
Selinux.
https://www.nsa.gov/research/selinux/index.shtml, 2009.
Visited 25.02.2015.

[13] Open Source Security.
grsecurity.
https://grsecurity.net, 2015.
Visited 25.02.2015.

[14] OpenBSD.
Pf: The openbsd packet filter.
http://www.openbsd.org/faq/pf/, 2015.
Visited 25.02.2015.

[15] PAX Team.
Pax.
https://pax.grsecurity.net, 2015.
Visited 25.02.2015.

[16] Pohl, C.
Github apate sourcecode gpl2.
https://github.com/c00clupea/apate, 2015.
Visited 25.02.2015.

References IV

[17] Pohl, C., Hof, H. J., and Meier, M.
Apate - A Linux Kernel Module for High Interaction Honeypots.
In *SECURWARE 2015, The Ninth International Conference on Emerging Security Information, Systems and Technologies* (Apr. 2015).

[18] Smalley, S., Vance, C., and Salamon, W.
Implementing selinux as a linux security module.
NAI Labs Report 1, 43 (2001), 139.

[19] Song, C., Ha, B., and Zhuge, J.
Know Your Tools: Qebek – Conceal the Monitoring — The Honeynet Project.
http://www.honeynet.org/papers/KYT_qebek.
Visited 25.02.2015.

[20] Torvalds, L.
Linux kernel release 3.x source linux/sched.h.
https://github.com/torvalds/linux/blob/master/include/linux/sched.h, 2015.
Visited 25.02.2015.

[21] (TurboBorland), T. B.
Modern linux rootkits 101.
http://turbochaos.blogspot.de/2013/09/linux-rootkits-101-1-of-3.html, 2013.
Visited 25.02.2015.

References V

[22] Wagener, G., State, R., Dulaunoy, A., and Engel, T.
Self Adaptive High Interaction Honeypots Driven by Game Theory.
In *Stabilization, Safety, and Security of Distributed Systems Lecture Notes in Computer Science* (Berlin, Heidelberg, 2009), Springer Berlin Heidelberg, pp. 741–755.