Building a state tracing kernel

Dr.Vinay G. Vaidya
Ananth Chakravarthy
Symbiosis Deemed University, Pune, India

Agenda

- The trigger
- The architecture
- Description of the tools
- Description of the interpreter
- Description of the new Kernel
- Conclusion

The trigger

- Anti-virus
  - Based on signatures
  - What if the signature is yet to be generated
- Buffer-overflow attacks
  - Generally exposed by an internet posting
  - Fix procedure involves updating the software

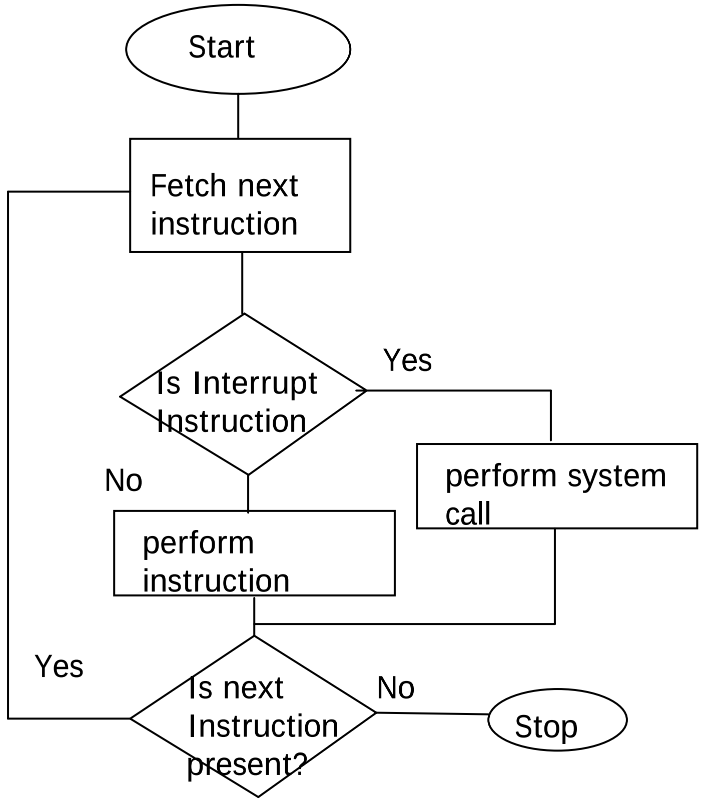
The trigger - Continued

- Some flaws in current security solutions
  - Not reactive
    - Wait for the attack to happen (anti-virus)
    - Wait for the vulnerability to be exposed (internet posting)
    - IDS – what if the signature is yet to be generated?
    - How safe are we in believing the ‘complacency’ of the end users?

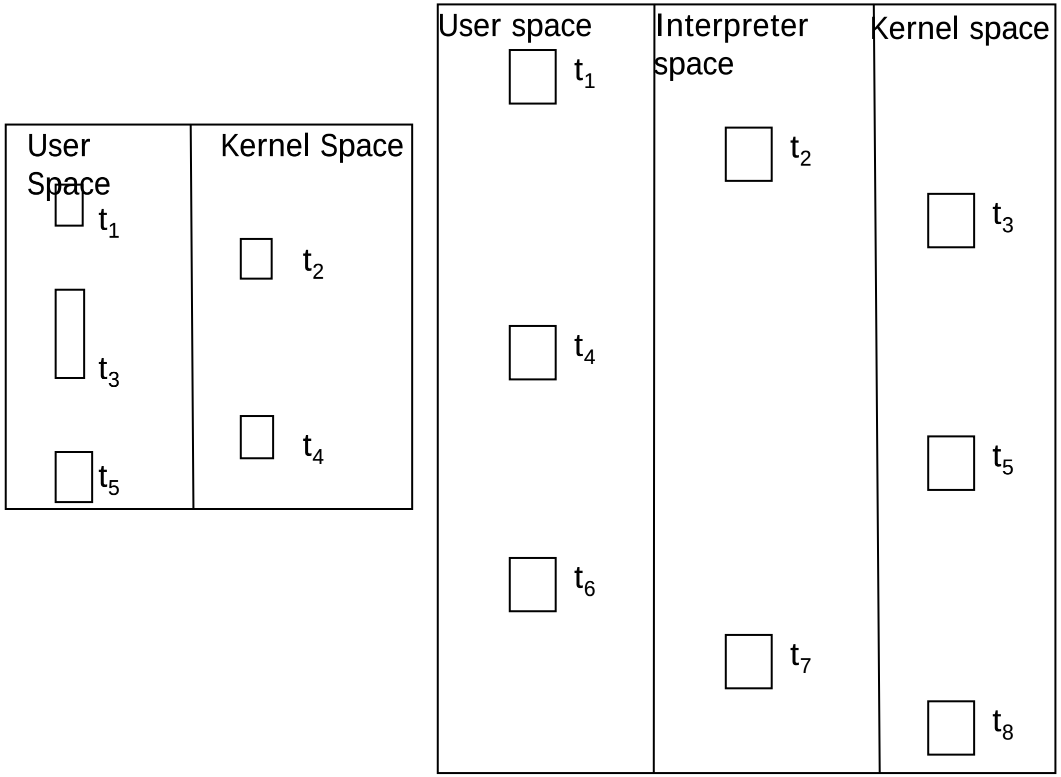
The trigger

- Hence a need for a system that
  - Attempts to protect before an attack actually happens.
  - The entire context of execution happens to be with the operating system rather than individual tools
  - Based on the semantics of execution of the binary

Current flow of execution



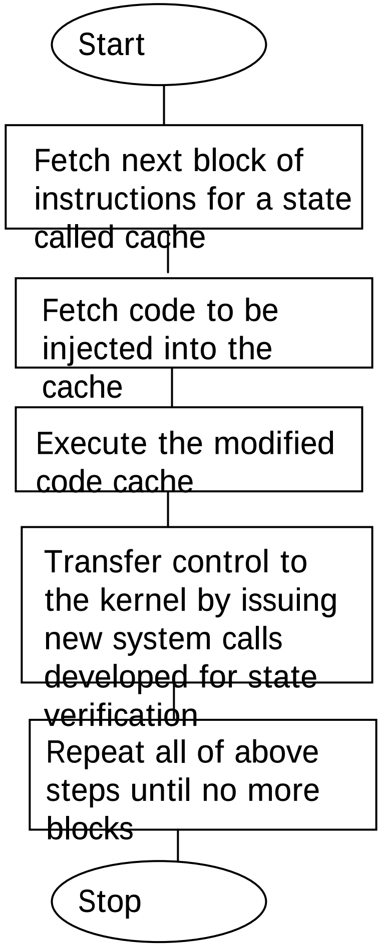
Architecture of the new system



Overall approach

- Tool to reverse engineer a binary to identify the complete set of states
- Tool to identify what are the characteristics for each of the states identified in the above step.
- An interpreter which keeps triggering the kernel verification code whenever there is a state transition.
- A modified kernel that accepts calls from the interpreter and verify the state transitions
- A mechanism inside the kernel to verify various aspects of the running process

Sequence as per new flow


Start
Fetch next block of instructions for a state called cache
Fetch code to be injected into the cache
Execute the modified code cache
Transfer control to the kernel by issuing new system calls developed for state verification
Repeat all of above steps until no more blocks
Stop

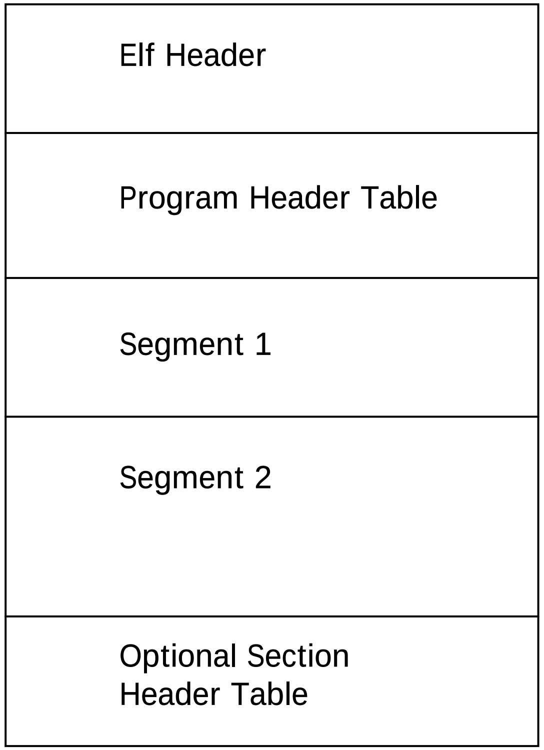
Deductions from the new architecure

- The amount of total time taken to execute the binary is definitely going to increase.
- The Interpreter acts as a sandbox under which the binary to be executed is to be run.
- There is some code as part of the interpreter which is executed intermixed with the code of the binary
- The number of system calls may increase proportionally to the number of states.

State defined

- A state may be defined as the collection of sequential instructions that do not branch off due to a jump (conditional/non-conditional), int or call instructions

Elf format defined



Elf Header

Program Header Table

Segment 1

Segment 2

Optional Section
Header Table

Sample disassembled code

- <FunctionCodeChunk funcName=_ZN11PLTModifier12copy_partialEiij
  > <InstructionList>
-
```
        08056DEE            55          push        ebp
        08056DEF            89 E5       mov         ebp         esp
        08056DF1            81 EC 18 10 00 00           sub         esp
        0x00001018
        08056DF7            C7 85 F4 EF FF FF 00 00 00 00           mov
        [ebp-4108]          0x00000000
        08056E01            8B 85 F4 EF FF FF           mov         eax
        [ebp-4108]
        08056E07            05 00 10 00 00    add       eax
        0x00001000
        08056E0C            3B 45 14          cmp       eax         [ebp+20]
        08056E0F            0F 83 8E 00 00 00           jnc
        0x08056EA3
```

Identifying state characteristics

- Memory state of the registers
- Memory state of some of the global variables
- Memory state of the function variables.
- Allowed state transitions
- Allowed set of system calls also termed as Actions
- Sequence of system calls

Additional requirements

- Commands
  - Used to capture state info at the kernel level
- Use cases
  - Capture a semantic set of actions
- Global Declarations
  - Common files to be loaded (libs)

Memory state of registers

- Not ‘collectible’ for all states
- Some of the mechanisms that can be used to capture are
  - Absolute value of registers
  - Relative value of registers
    - Value increases/decreases from a given state by a definite value
  - Stack based register signature

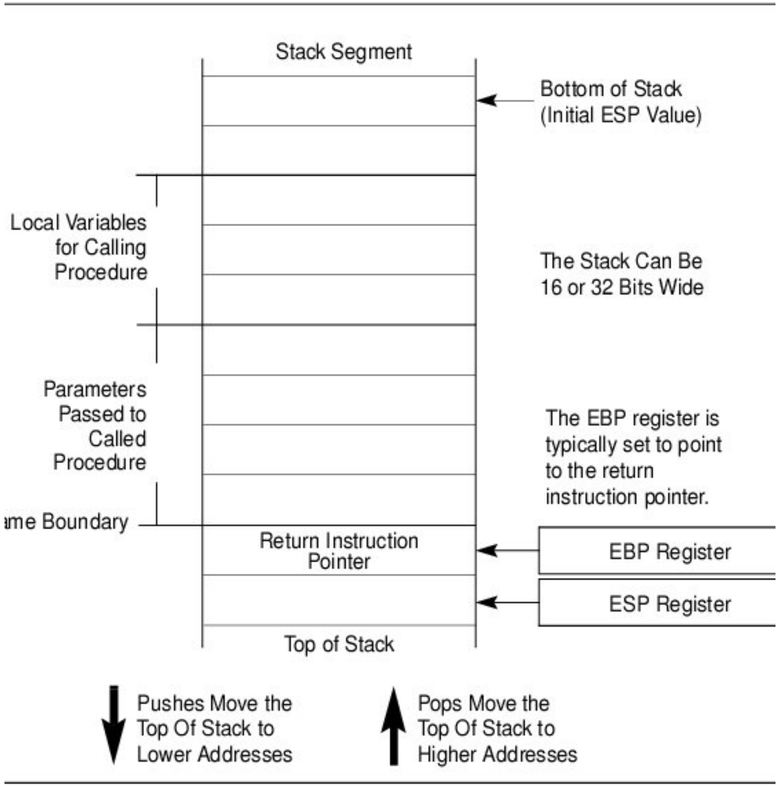
Memory state of registers

- Ideally should be verified in the interpreter space
- Cant be applied to the library dis-assembled code as lib code is generally position independent.
  - Since pos independent, verification will be difficult

Memory state of global variables

- Signature extracted by looking at portions of code that tend to
  - Read/write to “.bss” section
  - Read access from “.rodata” section

Memory state of function variables



- Function stack will be used to generate the stack frame
- The state is calculated using the references by using the pattern [ebp + xxx]

Allowed set of transitions

- Used to track the jmps/calls in the binary address range.
- Cant effectively mark the valid transitions for library code.
- Can be verified by the interpreter when the control reaches the interpreter space

Allowed set of system calls

- System calls are generally implemented by libraries.
- They can be analyzed by the presence of “int” instruction
- Static analysis of the system calls is very difficult because the system call is acted by the values present in various registers
- Extracting the values of registers before the int instruction requires the processing a lot more up the stack

Allowed sequence of system calls

- The most complex form of signature generation
- There are loops and conditionals before the actual system call point or state is reached.
- It becomes difficult because of “call” instructions

Commands

- Sometimes it becomes difficult to verify a state until some information is given to kernel.
- A command gives a directive to the kernel to collect state information so that it can be verified at a later point in time.
- Ex: A file write operation might verify based on file open operation.

UseCases

- Each usecase is triggered by the calling of a function
- The tool asks the high-level function that triggers the functioning of the usecase
- The tool then builds the tree of code that can be called from this point including the library code chunks.
- It builds the various signatures as mentioned previously for each usecase.

The interpreter

- Based on the dynamorio framework
- A code caching framework
- Effort involved in building the library that implements the hooks
- The interpreter is used to primarily check
  - Register signatures
  - Permissible transitions

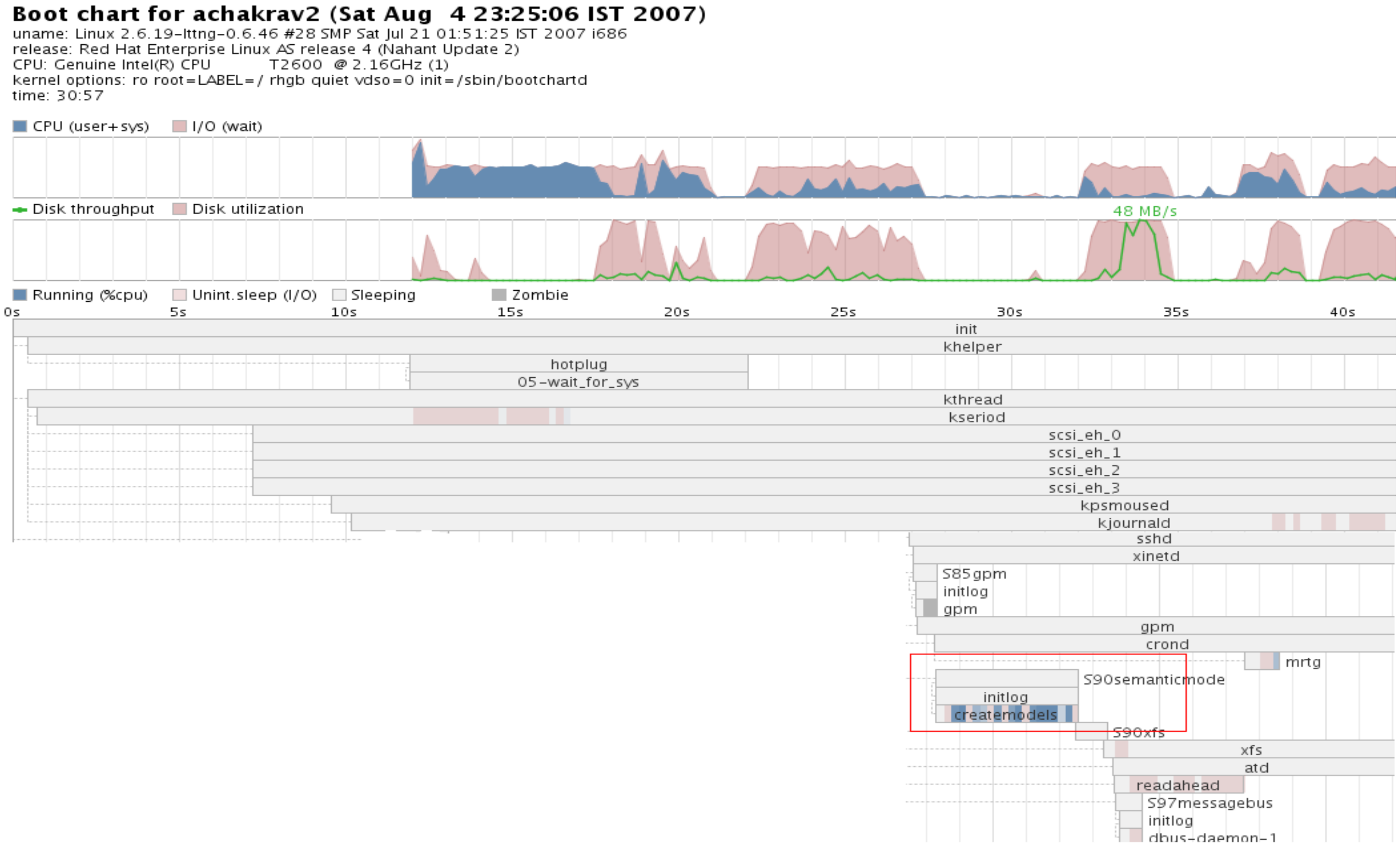
Modified kernel

- Additions to task_struct
  - History_node
  - Static description (as generated by the tool)
  - Runtime description (commands collected)
- A new set of system calls for
  - Interpreter to call for
    - Storing information
    - Triggering verification when the use case has been completed (as per address transition)
  - The model loader at boot time

Modified Kernel

- The verification runs as a parallel thread.
- The interpreter triggers the verification
- The verification can also be done for priority states
  - For example, opening a socket, opening a file

Some observations



Boot chart for achakrav2 (Sat Aug 4 23:25:06 IST 2007)

uname: Linux 2.6.19-lttng-0.6.46 #28 SMP Sat Jul 21 01:51:25 IST 2007 i686
release: Red Hat Enterprise Linux AS release 4 (Nahant Update 2)
CPU: Genuine Intel(R) CPU T2600 @ 2.16GHz (1)
kernel options: ro root=LABEL=/ rhgb quiet vdso=0 init=/sbin/bootchartd
time: 30:57

Performance

- System yet to be completed hence complete statistics not yet available.
- Performance hit observed. (around 100 % decrease in performance for some binaries)
- Need to optimize on
  - Number of verifications
  - Deductible verification

Thank you

- Q & A