Automatic Profile generation for live Linux Memory analysis.

Arkadiusz Socała and Michael Cohen

University of Warsaw / Google Inc.

DFRWS EU 2016

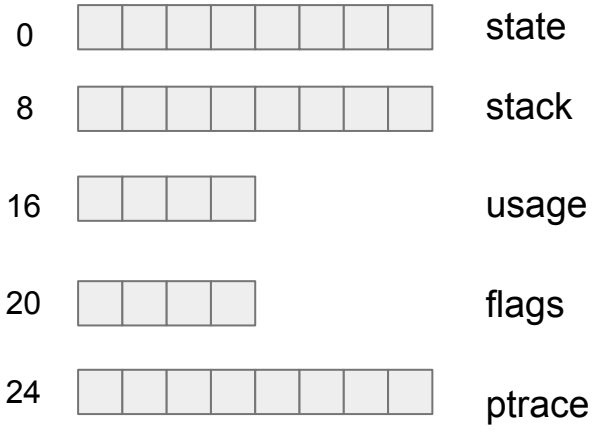
Proprietary + Confidential

Memory Analysis - What is it?

Trying to make sense of memory.

```
struct task_struct {
        volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags;     /* per process flags, defined below */
        unsigned int ptrace;

#ifdef CONFIG_SMP
        struct llist_node wake_entry;
        int on_cpu;
        struct task_struct *last_wakee;
        unsigned long wakee_flips;
        unsigned long wakee_flip_decay_ts;

        int wake_cpu;
#endif
        int on_rq;

        int prio, static_prio, normal_prio;
        unsigned int rt_priority;
        const struct sched_class *sched_class;
        struct sched_entity se;
        struct sched_rt_entity rt;
```

C Code



Profile

Proprietary + Confidential

What is a profile?

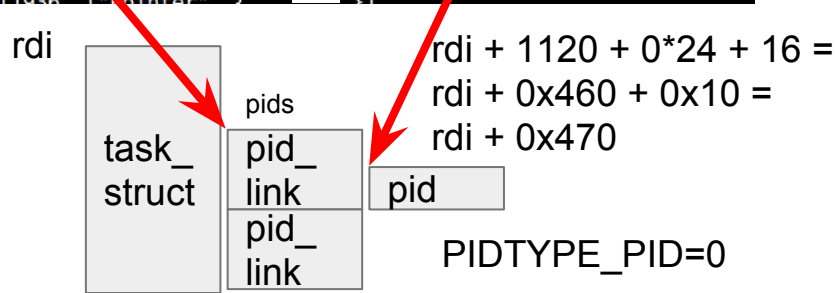
- A profile is the template that allows us to know where each field member begins and ends in memory.
- The compiler will normally plan how to lay each struct member in memory and generate code that access this struct:

```
545 struct pid_namespace *task_active_pid_ns(struct task_struct *tsk)
546 {
547         return ns_of_pid(task_pid(tsk));
548 }
549 EXPORT_SYMBOL_GPL(task_active_pid_ns);
550
1887 static inline struct pid *task_pid(struct task_struct *task)
1888 {
1889         return task->pids[PIDTYPE_PID].pid;
1890 }
```

```
"pids": [1120, ["Array", {
 "count": 3,
 "target": "pid_link",
 "target_args": null
}]],
```

```
"pid_link": [24, {
 "node": [0, ["hlist_node"]],
 "pid": [16, ["Pointer", {
  "target": "pid",
  "target_args": null
 }]]
```

```
[1] Ubuntu14.10_virtualbox.aff4 23:57:06> dis "linux!task_active_pid_ns"
                                        -> dis("linux!task_active_pid_ns")
Address      Rel            Op Codes                  Instruction                    Comment
------ ------------ ------------------ ------------------------------------------ -------
------ linux!task_active_pid_ns ------: 0xffff81090b40
 0xffff81090b40           0x0 6666666690             nop
 0xffff81090b45           0x5 488b8770040000         mov rax, qword ptr [rdi + 0x470]
 0xffff81090b4c           0xc 55                     push rbp
 0xffff81090b4d           0xd 4889e5                 mov rbp, rsp
 0xffff81090b50          0x10 4885c0                 test rax, rax
 0xffff81090b53          0x13 7413                   je 0xffff81090b68
```



Proprietary + Confidential

Take away:

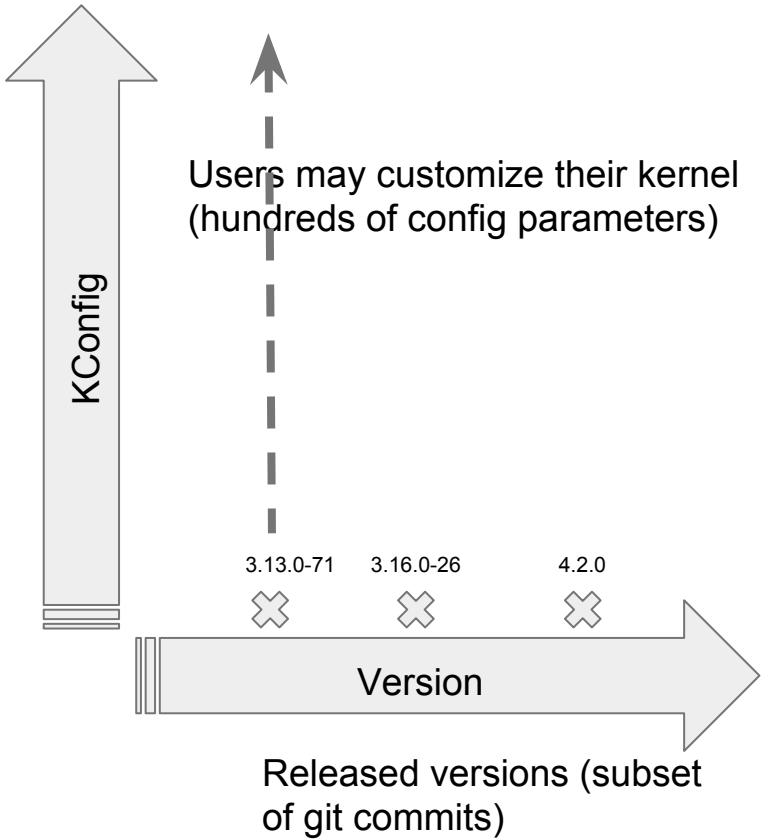
- Compiler generates code that uses the planned struct layout.
- The struct layout is not actually written explicitly anywhere in the binary code.
- The compiler can write down struct layout in debugging information (e.g. DWARF).
- It **is** possible to guess how the compiler will lay out the struct after a bit of experience.
- It is difficult to go from assembly code to profile because sometimes assembly code is the result of several offsets combined.
- Every kernel version may have a different profile with different values!

Sources of struct variability

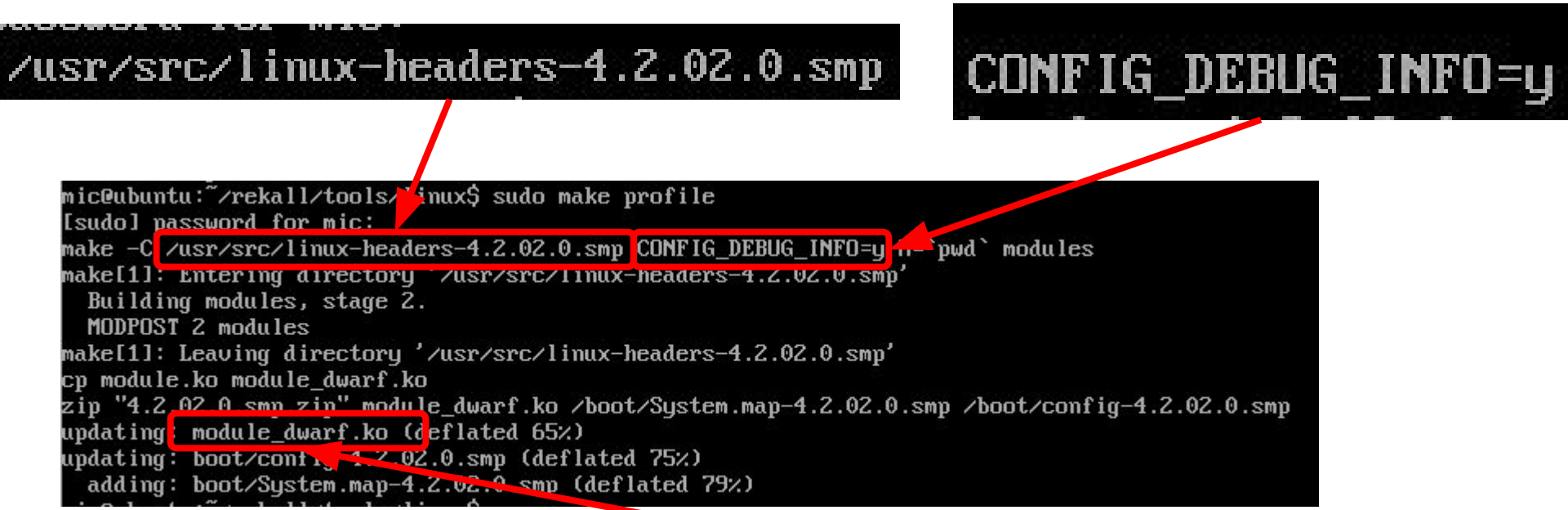
Two orthogonal sources of variability

1. Kernel versions
2. kernel configuration.

Number of potential unique profiles -> huge!



Current state of the art - Step 1



Compile test module with debug information generated - stored in zip file.

Current state of the art - Step 2

Use profile for live analysis

Parse DWARF data into json.

```
mic@ubuntu:~/rekall/tools/linux$ rekal convert_profile 4.2.02.0.smp.zip 4.2.02.0.smp.json
mic@ubuntu:~/rekall/tools/linux$ sudo make pmem
make -C /usr/src/linux-headers-4.2.02.0.smp M=`pwd` modules
make[1]: Entering directory '/usr/src/linux-headers-4.2.02.0.smp'
  Building modules, stage 2.
  MODPOST 2 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.2.02.0.smp'
mic@ubuntu:~/rekall/tools/linux$ sudo insmod pmem.ko
mic@ubuntu:~/rekall/tools/linux$ sudo rekal -p 4.2.02.0.smp.json -f /dev/pmem pslist --pid=1
 task_struct           Name          PID    PPID   UID    GID        DTB                Start Ti
------------- -------------------- ------ ------ ------ ------ -------------- ----------------
WARNING:rekall.1:mnt_devname has no offset in object vfsmount. Check that vtypes has a concret
0x88007c880000 systemd                    1      0      0      0 0x00003502d000     1969-12-30
```

Compile test module with debug information generated.

This is annoying for Incident Response/Live analysis

- Before we can respond and analyze memory we need to:
  - Find the kernel headers package.
  - If the user modified the kernel we actually need the full sources so that autoconf.h can be generated.
  - We need to build the debug module on a similar system (or the same system we want to analyze if we can't find a similar clean system).

chort [US Person]
@chort0

Follow

I will pay someone $1000 if they find a way to do Linux memory forensics w/o building a fscking profile for every gd kernel.

RETWEETS 14 LIKES 10

6:39 PM - 13 Mar 2015

Reply to @chort0

Andrew Case @attrc · 13 Mar 2015
@chort0 I have it 80% solved with modifications to LiME, but the last 20% suck

1 1

chort [US Person] @chort0 · 13 Mar 2015
@attrc Wooo Linux!!! Maybe you could pay a grad student $200 and collect the remaining $800 ;)

Barry Anderson @z3ndrag0n · 13 Mar 2015
@chort0 LOL - good luck!

View other replies

There has to be a better way!

- Current technique is not scalable and not automated.
- There is a high probability that it will not work:
  - Kernel header package is not available.
  - Kernel was custom compiled.
- Compilers are often not installed or available on the system we want to respond to!
  - Not forensically sound to install compilers, kernel headers and start many processes on compromised systems.
  - Typically we want to avoid changing the system under investigation.

Current state of the art - Rekall

- Rekall has an extensive profile repository for every released Linux kernel by distributions.
  - An automated pipeline installing kernel headers from distributions, and compiling Rekall profiles from DWARF symbols.
- This means that for standard Linux versions, Rekall has it covered already!
  - Rekall uses the /proc/kallsyms file to automatically identify the correct profile from a profile index.

Fully automated profile detection - no need to specify the profile explicitly.

```
Dev)root@scudette-glaptop:/home/scudette# rekal --live pslist
task_struct          Name          PID    PPID   UID    GID        DTB             Start Time          Binary
------------ -------------------- ------ ------ ------ ------ ------------ ---------------------- ------
016-03-14 10:53:25,495:WARNING:rekall.1:mnt_devname has no offset in object vfsmount. Check that vtypes has a concrete definition for it.
x8804285f0000 init                     1      0      0      0 0x0004258b9000     2016-03-12 21:10:50Z /sbin/init
x8804285f1800 kthreadd                 2      0      0      0 -                  2016-03-12 21:10:50Z -
x8804285f3000 ksoftirqd/0              3      2      0      0 -                  2016-03-12 21:10:50Z -
x8804285f6000 kworker/0:0H             5      2      0      0 -                  2016-03-12 21:10:50Z -
x880428619800 rcu_sched                7      2      0      0 -                  2016-03-12 21:10:50Z -
x88042861b000 rcuos/0                  8      2      0      0 -                  2016-03-12 21:10:50Z -
x88042861c800 rcuos/1                  9      2      0      0 -                  2016-03-12 21:10:50Z -
x88042861e000 rcuos/2                 10      2      0      0 -                  2016-03-12 21:10:50Z -
```

Jordi Sanchez

GitHub This repository Search Explore Features Enterprise Pricing Sign up Sign in

google / rekall-profiles Watch 17 Star 21 Fork 13

Code Issues 2 Pull requests 3 Pulse Graphs

Branch: gh-pag... rekall-profiles / v1.0 / Linux / Ubuntu / 12.04 / x86_64 / New file Find file History

parkisan Added all 64bit source profiles for Redhat 6.7 and Ubuntu precise, tr... Latest commit 405d435 on Feb 9

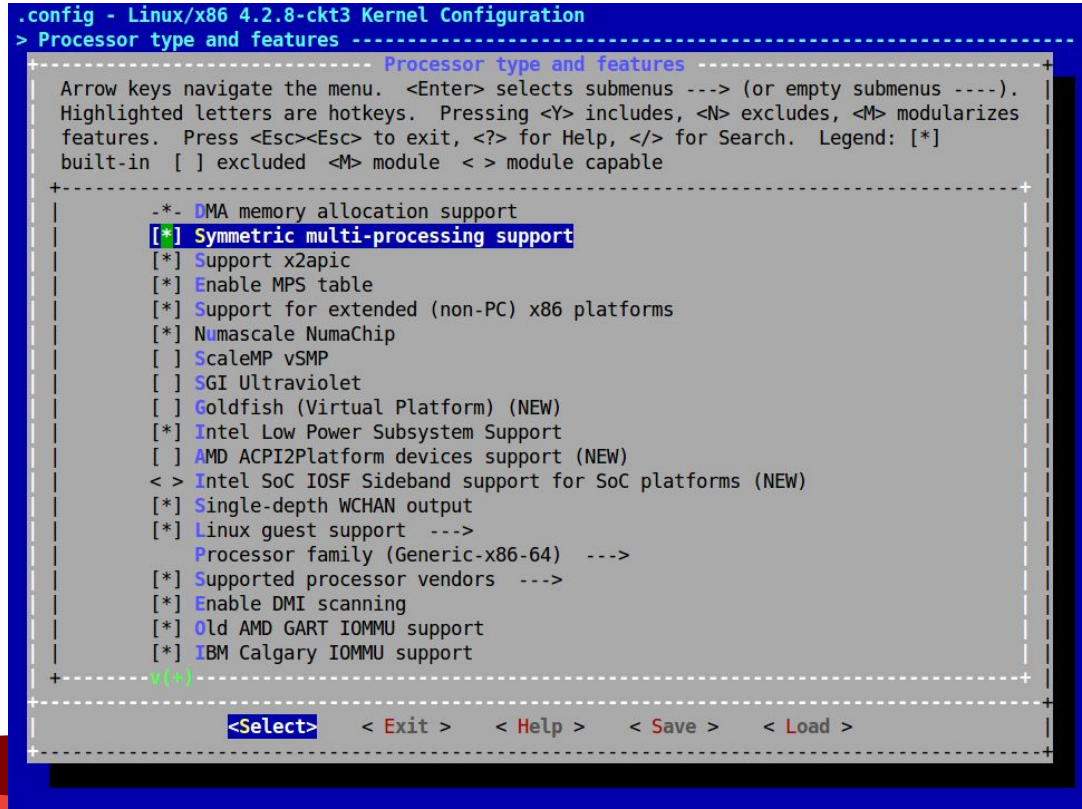
```
…usty, wily and vivid, as of today. These now include pmem as well.

Added all the Rekall JSON profiles for the above versions. Profiles have
been deduped.
Added an index of all deduped Linux Rekall profiles.
```

..		
3.11.0-13-generic.gz	Added all 64bit source profiles for Redhat 6.7 and Ubuntu precise, tr...	a month ago
3.11.0-14-generic.gz	Added all 64bit source profiles for Redhat 6.7 and Ubuntu precise, tr...	a month ago
3.11.0-15-generic.gz	Added all 64bit source profiles for Redhat 6.7 and Ubuntu precise, tr...	a month ago
3.11.0-17-generic.gz	Added all 64bit source profiles for Redhat 6.7 and Ubuntu precise, tr...	a month ago
3.11.0-18-generic.gz	Added all 64bit source profiles for Redhat 6.7 and Ubuntu precise, tr...	a month ago
3.11.0-19-generic.gz	Added all 64bit source profiles for Redhat 6.7 and Ubuntu precise, tr...	a month ago
3.11.0-20-generic.gz	Added all 64bit source profiles for Redhat 6.7 and Ubuntu precise, tr...	a month ago
3.11.0-22-generic.gz	Added all 64bit source profiles for Redhat 6.7 and Ubuntu precise, tr...	a month ago
3.11.0-23-generic.gz	Added all 64bit source profiles for Redhat 6.7 and Ubuntu precise, tr...	a month ago
3.11.0-24-generic.gz	Added all 64bit source profiles for Redhat 6.7 and Ubuntu precise, tr...	a month ago

What if the user rebuilds the kernel?

- Linux kernel is highly configurable.
- There is a fancy configuration system.
- Selecting different options results in setting various #define macros.

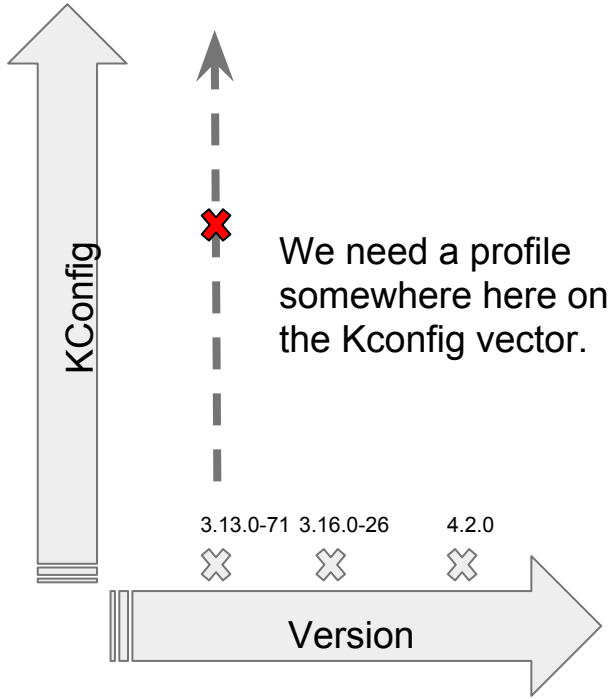
```
#define CONFIG_TOUCHSCREEN_MMS114_MODULE 1
#define CONFIG_I2O_LCT_NOTIFY_ON_CHANGES 1
#define CONFIG_SMP 1
#define CONFIG_FB_KYRO_MODULE 1
#define CONFIG_DVB_ZL10353_MODULE 1
```



Struct layout depends on configuration!

```
struct task_struct {
        volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags;     /* per process flags, defined below */
        unsigned int ptrace;

#ifdef CONFIG_SMP
        struct llist_node wake_entry;
        int on_cpu;
        struct task_struct *last_wakee;
        unsigned long wakee_flips;
        unsigned long wakee_flip_decay_ts;

        int wake_cpu;
#endif
        int on_rq;

        int prio, static_prio, normal_prio;
        unsigned int rt_priority;
        const struct sched_class *sched_class;
        struct sched_entity se;
        struct sched_rt_entity rt;
```

C Code



Can we guess the struct layout from C code?

- Common wisdom is that it is not reliable to do this.
- This is not exactly true:
  - The compiler must produce a consistent memory layout scheme in order to enable cross module linking. There is a platform Application Binary Interface (ABI) that specifies how structs should be laid out in memory.
- Unfortunately the ABI is not very detailed on a number of corner cases (e.g. bit fields).
- Some compiler directives override the ABI:
  - Compilers can do whatever they want if the user overrides the ABI.
  - Example __attribute__(packed), __attribute__(aligned)
- We need to do experimentation to deduce the layout strategy and the interaction of different directives.
- In reality layout is pretty stable and predictable because otherwise you can't link shared objects built with different versions of gcc.

How does the C compiler derive the layout?

- C code is pre-processed by the C pre-processor:
  - Expand macros.
  - Define macros (#define FOOBAR).
  - Conditionally include code based on macros (#ifdef).
  - Include other files (#include)
- Pure C code is fed to a C parser.
  - Generates the Abstract Syntax Tree.
  - Typedefs
  - Struct definitions
  - Generate Code (we don't need this).
- AST is fed to a layout engine
  - Calculate alignment requirements.

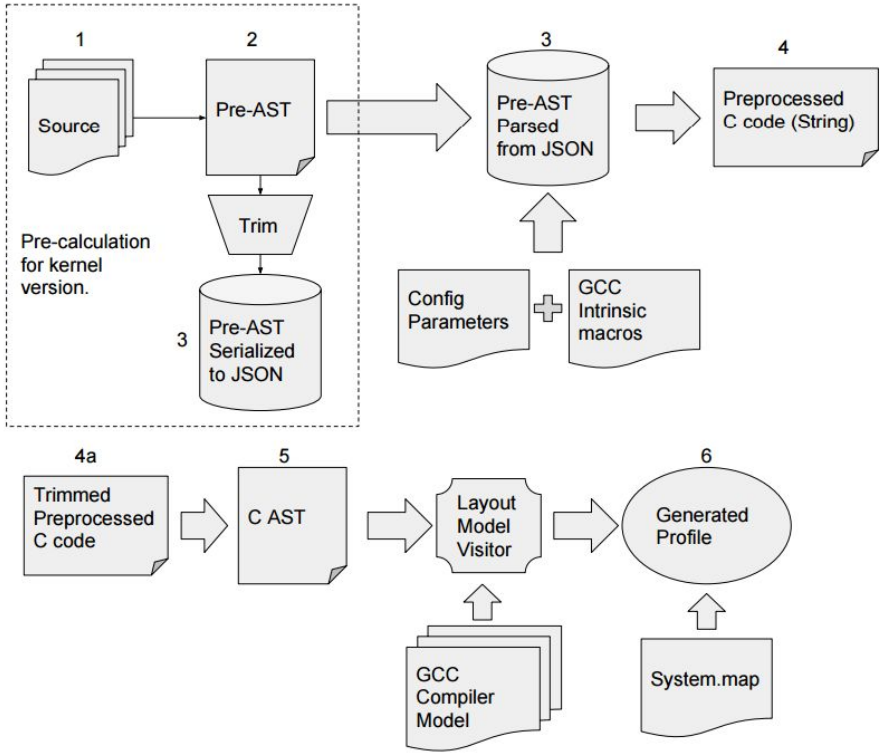
How does the Layout Expert work?

- Emulate the compiler tool chain
  - Take shortcuts wherever we can:
    - We don't actually need to generate code so we can simply remove much of the source text.
    - Only keep as much as needed to influence the final profile layout calculation.
- We have 2 separate parsers:
  - C preprocessor parser (emulates GNU CPP).
    - Builds a preprocessor AST.
  - C struct parser
    - Only smart enough to understand structs and typedefs - throws away everything else.

Layout Expert Overview

Two phases:

1. Preparation - prior to live analysis:
   a. Build a Preprocessor AST for kernel version.
2. During live analysis:
   a. Apply local system's Kconfig and System map to the Pre-AST.
   b. Produce a working profile.



Preparation - Pre AST

- Pre-AST contains all combinations of code (e.g. both #ifdef/#else branches).
- Macros are **not** evaluated in this phase (evaluation of macros requires the configuration file).
- The Pre-AST is trimmed as much as possible to reduce unnecessary text.
- Pre-AST is serialized into JSON for further consumption at runtime.
- Note that we must ignore autoconf include because it contains CONFIG_* macros.

Apply local config file to Pre-AST

- Pre-AST file contains both branches of #ifdef.
- The config file decides which branch should be considered.
- Extract configuration options from Kconfig file.
- Evaluate all macros into a **preprocessed (pure) C file**.
- We only care about struct definitions, so we **Trim** the pure C file into type definitions.
- Parse each type definition to build a C - AST.
- Use the layout model visitor to calculate layout for all struct members.
- Write a Rekall profile.

Proprietary + Confidential

Profit! Use the new profile to analyze the live system.

Future work

- Automate the whole process:
  - Rekall can find the config file and System map from the live system by itself.
  - Rekall can apply these to the pre-AST file, and generate its own profile.
- Reduce the size of the pre-ast file:
  - By making some assumptions about kernel configuration we may be able to partially resolve the pre-ast (e.g. CPU architecture).
- The ability to calculate memory layout on the fly allows us to begin work on more applications:
  - Tune binaries at runtime without recompilation:
    - We can create the pmem acquisition module for live analysis.

Questions?

Try it out:

```
$ pip install rekall-layout-expert
```

http://www.rekall-forensic.com/