Flying Under the Radar:
Maintaining Control of Kernel without Changing Kernel Code or Persistent Data Structures

Jinpeng Wei

Florida International University
weijp@cs.fiu.edu

Calton Pu

Georgia Institute of Technology
calton@cc.gatech.edu

Keke Chen

Wright State University
keke.chen@wright.edu

7th Annual Cyber Security and Information Intelligence Research Workshop (CSIIRW)

Oak Ridge National Laboratory
October 12 - 14, 2011

Smart Power Grid and Security

Source: http://www.renewablepowernews.com/wp-content/uploads/smart-grid-doe-illustration.jpg

- Cyber-spies could use their access to take control of power plants during a time of crisis or war
- But they need to hide first; they rely on stealthy malware (e.g., rootkits) to stay hidden before the actual strike
- If we are to defeat such cyber-spies, we must better understand their hiding capabilities

The Botnet Threat

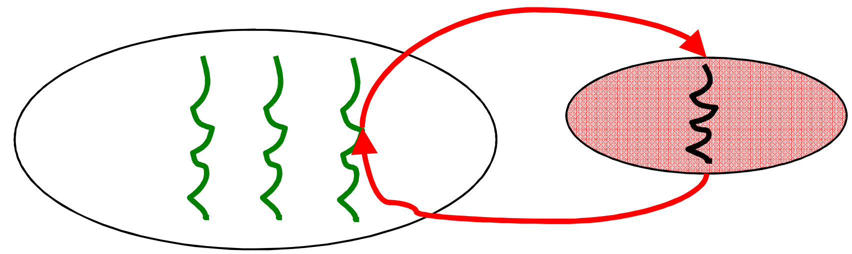
- A network of compromised computers under the control of a bot master
- Command-and-control infrastructure seems ideal for managing cyber-spies
- Already one of the major security threats
- It is desirable and feasible for the bots to achieve stealthy hiding of malware in the *kernel* space

Outline of the Talk

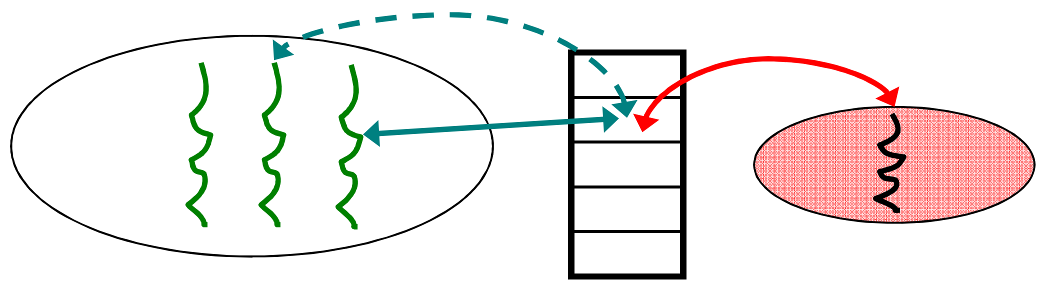
- Overview of kernel control flows
- Kernel-queue driven control flow attacks
- Two case studies
- Possible defenses
- Conclusion

Classification of Stealthy Control Flow Attacks in the Kernel

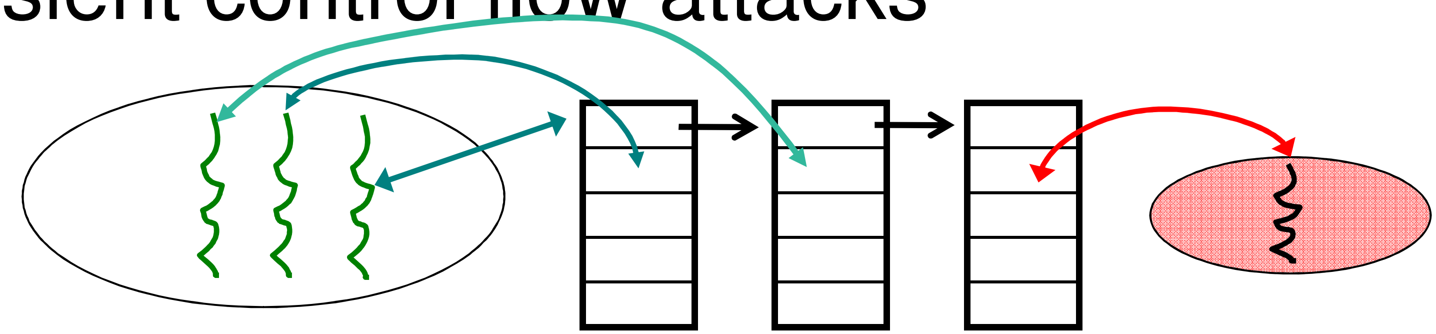
- Detour attacks



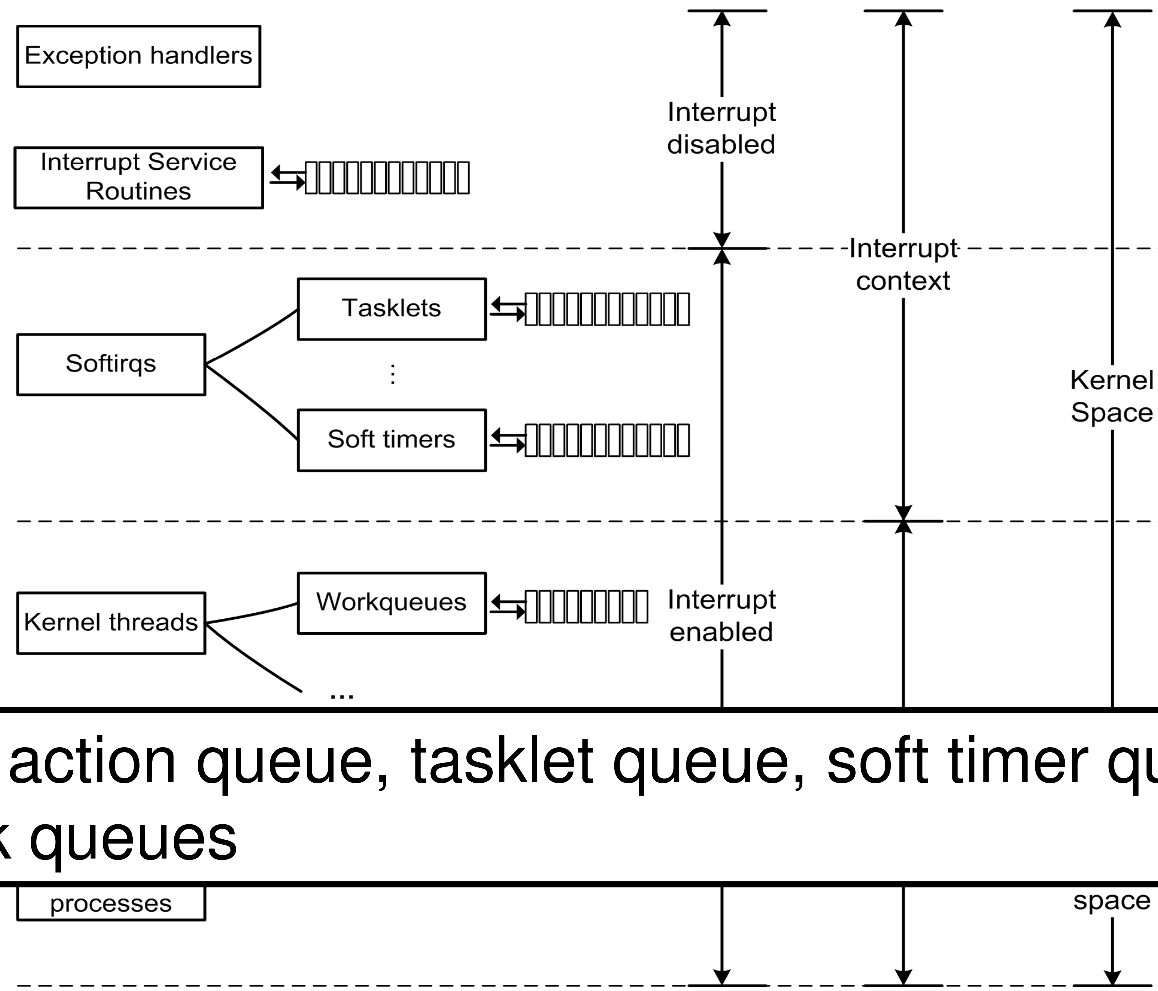
- Persistent control flow attacks



- Transient control flow attacks



Kernel Control Flows

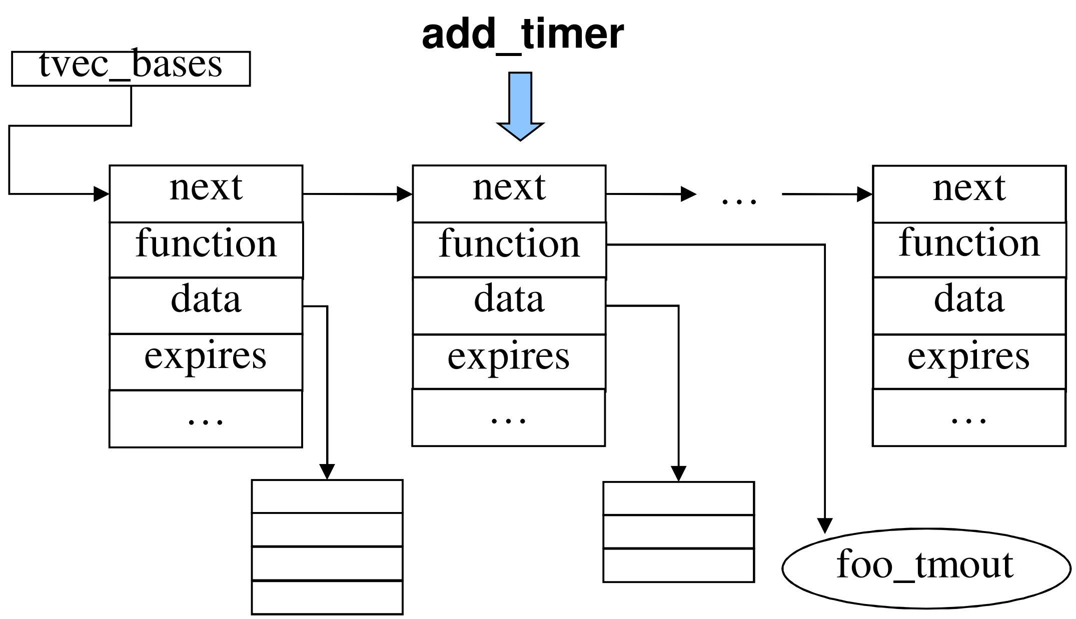


IRQ action queue, tasklet queue, soft timer queue, work queues

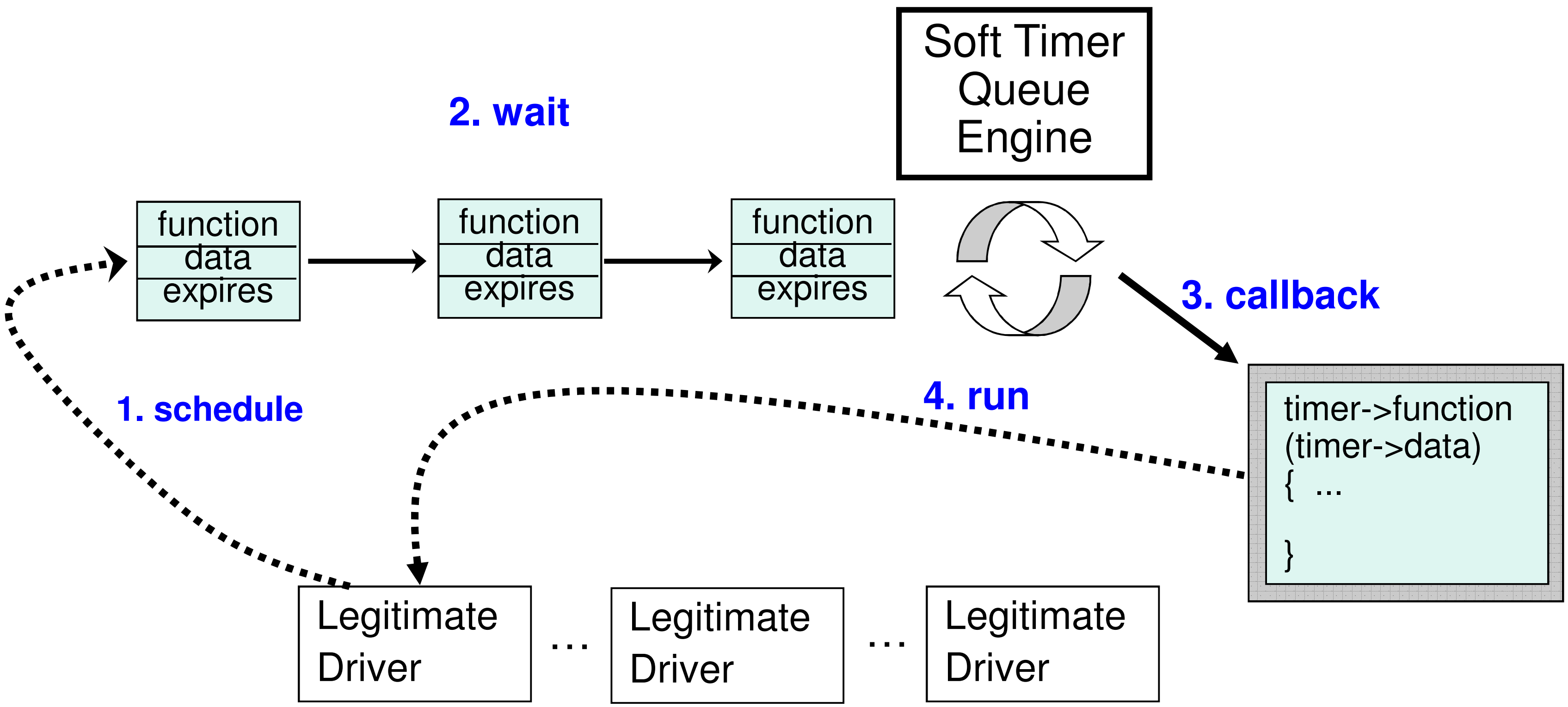
K-Queues (Kernel Schedulable Queues)

- Dynamic schedulable queues in the kernel
- Examples: IRQ action queue, tasklet queue, soft timer queue, work queues

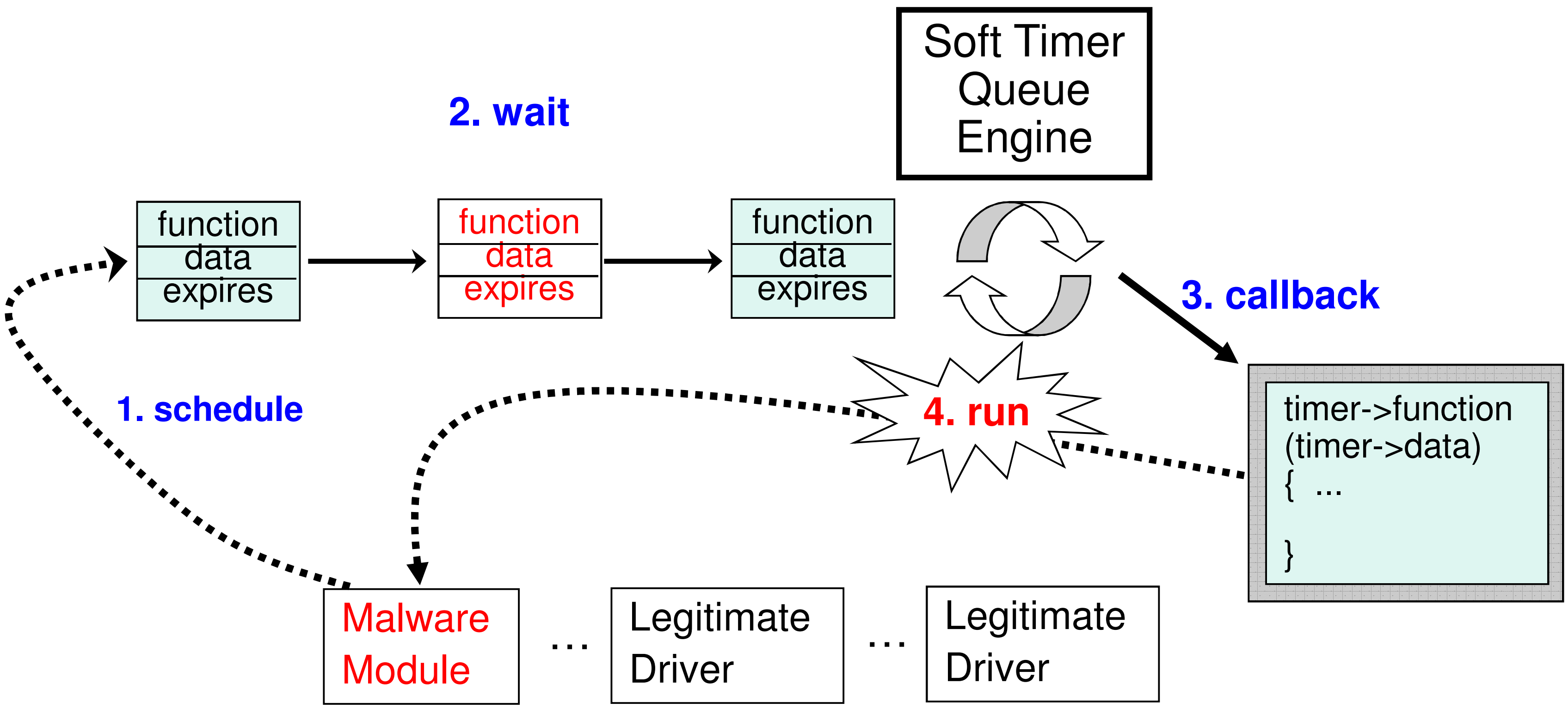
The soft timer queue:



Soft-timer-driven Transient Control Flow Attacks



Soft-timer-driven Transient Control Flow Attacks



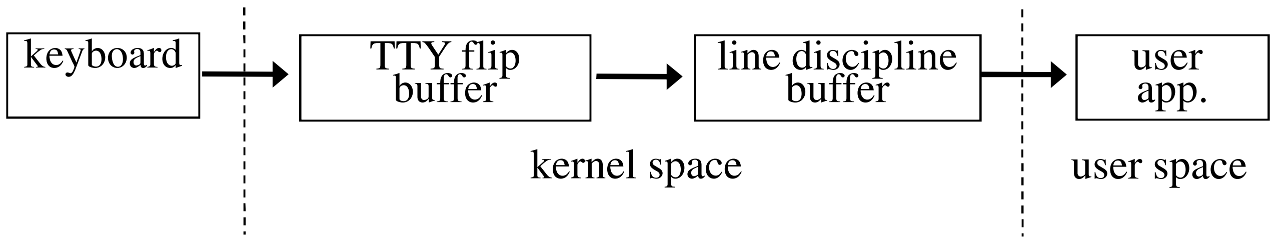
K-Queue-driven Malware in Reality

- The Rustock.C spam bot relies on two Windows kernel timers to check whether it is being debugged/traced
- The Storm/Peacomm spam bot invokes PsSetLoadImageNotifyRoutine to register a malicious callback function that disables security products
- Proof-of-concept malware

Proof of Concept Malware

- How do they work?
  - Request the first tasklet to interpose on the kernel control flow at break-in
  - Execute when the first tasklet callback function is invoked
  - Before giving up control, schedule the next tasklet
  - Wait for the next callback to happen
- What can they do?
  - Collect confidential information (stealthy key logger)
  - Mount a DoS attack (stealthy cycle stealer)

The Stealthy Key Logger



- Runs in Linux kernel 2.6.16
- Uses a tasklet
- The callback function reads the TTY line discipline buffer in the kernel, which can keep a history of up to 2,048 keystrokes
- Triggered every one second

Code Skeleton of the Key Logger

```
DECLARE_TASKLET(keylogger_tasklet, log_it, 0);

static void log_it(unsigned long arg){
    dump_keybuffer();
    keylogger_timer.expires = jiffies + (HZ);
    add_timer(&keylogger_timer);
    return;
}

struct timer_list keylogger_timer  =
    TIMER_INITIALIZER(sched_me, 0, 0);

static void sched_me(void){
    tasklet_schedule(&keylogger_tasklet);   return;
}
```

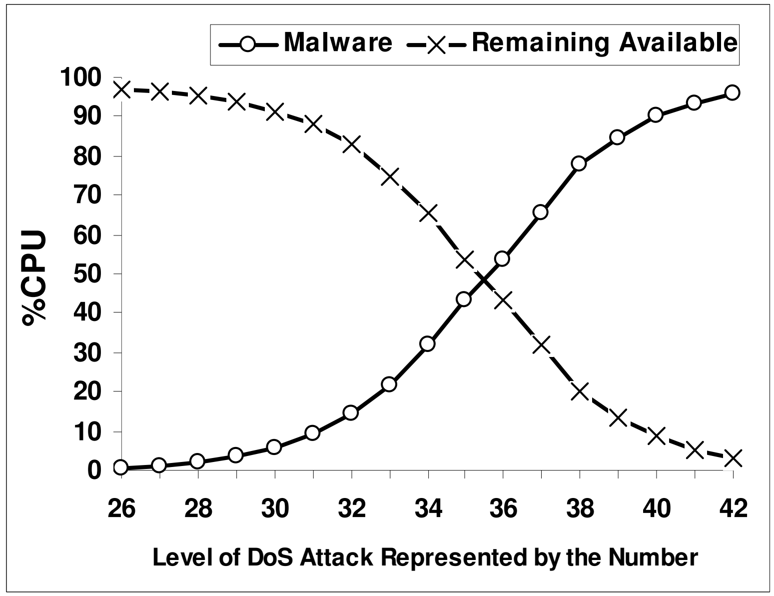
Schedule the next tasklet

The Stealthy Cycle Stealer

- Compute the factorial of a given number in the callback function
- Adjust the value of the number and the callback frequency to obtain different slowdown factors

Slowdown Factors of the Stealthy Cycle Stealer

- **Timer-driven:**



Frequency: one callback per second

- **Tasklet-driven:**

- ❑ When the number is 41, about 1/3 of total CPU time is consumed by the malware
- ❑ The CPU is saturated when the number reaches 48
- ❑ Tested on an Intel Xeon at 2.93GHz with 196MB memory and 6GB hard disk

The Stealthy Cycle Stealer

- Compute the factorial of a given number in the callback function
- Adjust the value of the number and the callback frequency to obtain different slowdown factors
- Manipulate the kernel accounting data to hide CPU time wasted

Outline of Possible Defense

- Idea: a legitimate K-Queue callback function and all functions that it calls transitively should always conform to a predetermined control flow graph

- Complete mediation of K-Queue execution
  - Check the callback function against a whitelist of legitimate K-Queue callback functions
  - The whitelist can be built from a static analysis of the kernel

Conclusion

- Maintaining a stealthy control over the kernels in the power grid cyber space has become an important strategy for the adversaries
- Transient kernel control flow attacks manipulate dynamic schedulable kernel queues (K-queues) to achieve continual malicious function execution
- Two illustrative examples show the feasibility and potential effectiveness of such attacks