# POSTER – Sechduler: A Security-Aware Kernel Scheduler

Parisa Haghani
Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
haghani1@illinois.edu

Saman Zonouz
Electrical and Computer Engineering
University of Miami
s.zonouz@miami.edu

## ABSTRACT

Trustworthy operation of safety-critical infrastructures necessitates efficient solutions that satisfy both realtimeness and security requirements simultaneously. We present Sechduler, a formally verifiable security-aware operating system scheduler that dynamically makes sure that system computational resources are allocated to individual waiting tasks in an optimal order such that, if feasible, neither realtime nor security requirements of the system are violated. Additionally, if not both of the requirements can be satisfied simultaneously, Sechduler makes use of easy-to-define linear temporal logic-based policies as well as automatically generated Büchi automaton-based monitors, compiled as loadable kernel modules, to enforce which requirements should get the priority. Our experimental results show that Sechduler can adaptively enforce the system-wide logic-based temporal policies within the kernel and with minimal performance overhead of 3% on average to guarantee high level of combined security and realtimeness simultaneously.

**Categories and Subject Descriptors:** D.4.6 [Operating Systems]: Security and Protection

**Keywords:** Real-time security; formal temporal verification; intrusion detection and prevention; operating system security.

## Sechduler

Maintenance of the safety-critical infrastructures, e.g., nuclear power plants and avionics systems, is extremely crucial because a failure to meet a single requirement may lead to a catastrophic consequence such as an explosion or an accident leading to loss of life. In particular, the realtime scheduling of tasks in those infrastructures such that individual timing requirements are met reliably is often a challenging endeavor. Furthermore, to guarantee core functionalities, those systems need to be secure and intrusion resilient as they operate in possibly adversarial environments. Currently many commercial and open-source security solutions are available that can monitor different security aspects of the systems. Clearly, the most comprehensive security level will be achieved by running a set of those security sensors in parallel; however, this would result in computationally intensive security analyses and hence overconsumption of the system's limited resources. Therefore, the system's core realtime functionality requirements could be violated as the system's critical tasks are deprived of the resources. This signifies the fact that to ensure timely accomplishment of the core system functionalities, the deployed security solutions need to be resource aware and satisfy the system-wide realtime requirements, i.e., *realtime security*. The same rationale justifies an urgent need for solutions to guarantee the *secure realtimeness* property provided by realtime solutions, e.g., realtime schedulers, that are aware of the system security requirements according to the high-level organizational objectives.

Previous efforts in designing realtime and security solutions have fallen short in several major aspects. There have been many theoretical as well as heuristic scheduling algorithms such as the Linux kernel 3.X Completely-Fair Scheduler [4], RTLinux [6] attempt to allocate the system CPU(s) to individual waiting tasks such that the likelihood of task starvations and deadline misses are minimized. Although the abovementioned solutions can be employed to ensure timely accomplishments of safety-critical and realtime applications, none of them take into account the existence possibility of malicious activities, e.g., an adversarial unfinished task waiting for execution. Security and privacy researchers have proposed numerous host-based intrusion prevention and detection solutions, e.g., Samhain [7], as well as forensics and root-cause analysis algorithms and tools, such as Backtracker [3], and FloGuard [8], in order to detect and terminate ongoing malicious misbehaviors with minimum amount of performance overhead on the target system. Even though the abovementioned security solutions attempt to minimize the overhead as a best effort to terminate attacks before it gets too late, e.g., confidential data is sent out to network, there is currently no generic and theoretically sound solution that considers the system's overall realtime requirements and guarantees timely reaction against the ongoing intrusions.

We present Sechduler, a formally verifiable security-aware operating system scheduler that guarantees simultaneous satisfaction of the system-wide realtimeness as well as security requirements. In particular, Sechduler accomplishes its objectives through three major steps. First, during a one-time offline phase, system security policies are defined that determine how the security vs. realtimeness tradeoffs should be resolved. These policies can be designed following whitelisting (deny by default), blacklisting (allow by default) or other more generalized paradigms. Second, during an online phase while the system is operating, Sechduler selects the appropriate subset of policies, given the current security state of the system, and generates the corresponding single logic-based conjunctive policy predicate. Sechduler then converts the policy to an extended finite state machine-based monitor automatically. Finally, Sechduler enhances the kernel scheduler with the generated monitor dynamically for runtime monitoring and verification of the system computational resource allocations. Consequently, Sechd-


*CCS'13,* November 4–8, 2013, Berlin, Germany.
ACM 978-1-4503-2477-9/13/11.
http://dx.doi.org/10.1145/2508859.2512527.

uler modifies the kernel's resource allocation schedule actively if it is about to violate any of the predefined system-wide security policies.

More specifically, Sechduler makes use of an easy to understand formal language, namely *three-valued linear temporal logic* that facilitates formulation of comprehensive temporal system-wide security policies for the system administrators. Needless to mention, the designed policies can be reused across systems (analogous to the SE-Linux access control policies). The employed three-value logic, i.e., *true:* policy-compliant, *false:* policy violation, and *inconclusive:* insufficient information, enables Sechduler to use the designed policies for accurate verification and reconfiguration of the kernel task scheduling dynamically based on the observed scheduling trace. , i.e., the past and current (to be) scheduled tasks. Sechduler considers the trace formally as a finite prefix of the (potentially) infinite task scheduling sequence in the future. For the kernel to understand and enforce the policies, Sechduler converts the logic-based policies automatically to an extended finite state machine, so-called Büchi automaton, working with the three-value logic. The Büchi automaton monitors the kernel scheduling trace and determines whether the policy is about to be violated. If so, the scheduler is reconfigured and the system CPU is allocated to the next non-policy-violating waiting task with an urgent need for execution. It is noteworthy that the automated conversion algorithm in Sechduler results in an automaton with provably minimum number of states ensuring that the performance overhead of the runtime monitoring and verification is minimized. Consequently, using a realtime and security-aware scheduling algorithm through continuous optimization for timely resource allocations and discrete logic-based monitoring for security verifications, Sechduler makes sure to provide both realtimeness and security guarantees simultaneously if feasible depending on the available time and resources.

In summary, the contributions of our work are as follows: 1) We propose an easy-to-understand logical formulation formalism to declare the system security requirements for different system security states; 2) We introduce a three-value logic-based automated algorithm to construct security formal monitors dynamically for runtime verification and temporal policy enforcement; 3) We propose a hybrid operating system task scheduling algorithm using continuous task ranking optimization and discrete logic-based formal verification techniques; and 4) We validate the Sechduler framework on a real-world host system through implementation and deployment of a working prototype of the proposed algorithms. It is also important to mention what Sechduler does not contribute to. In particular, Sechduler does not present a new intrusion detection sensor and automatic logic-based policy generation algorithm. Instead, Sechduler makes use of those solutions to provide the runtime verification capability to maintain the system security and realtime requirements and avoid potential violations of the previously defined temporal policies.

**Sechduler Overview.** Initially, the security administrators write system security temporal policies using the easy-to-understand formalism in Sechduler. This phase is very similar to writing access control policies for firewalls or host-based SE-Linux systems; however, in Sechduler, administrators concentrate on timing- and scheduling-related security concerns instead. Briefly, each policy determines the scheduling constraints that need to be held at a system security state by the operating system to guarantee that the system-wide security is maintained. Although, we assume that the policy writing is completed as a one-time manual effort, Sechduler could be extended and make use of the recent (semi-)automated policy writing algorithms and tools [5].

During the system's operational mode, we assume that appropriate host-based intrusion detection systems are deployed and are

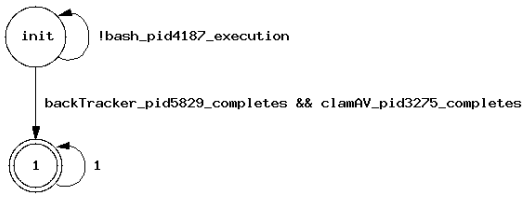



**Figure 1: The Büchi Automaton for the Predicate**

**G** ((receive_request ∧ ¬send_response ∧ **F** send_response) → (sensitive_file_access → (¬send_response **U** (security_check ∧ ¬send_response))) **U** send_response)

**Figure 3: A Sample Temporal Security Policy**

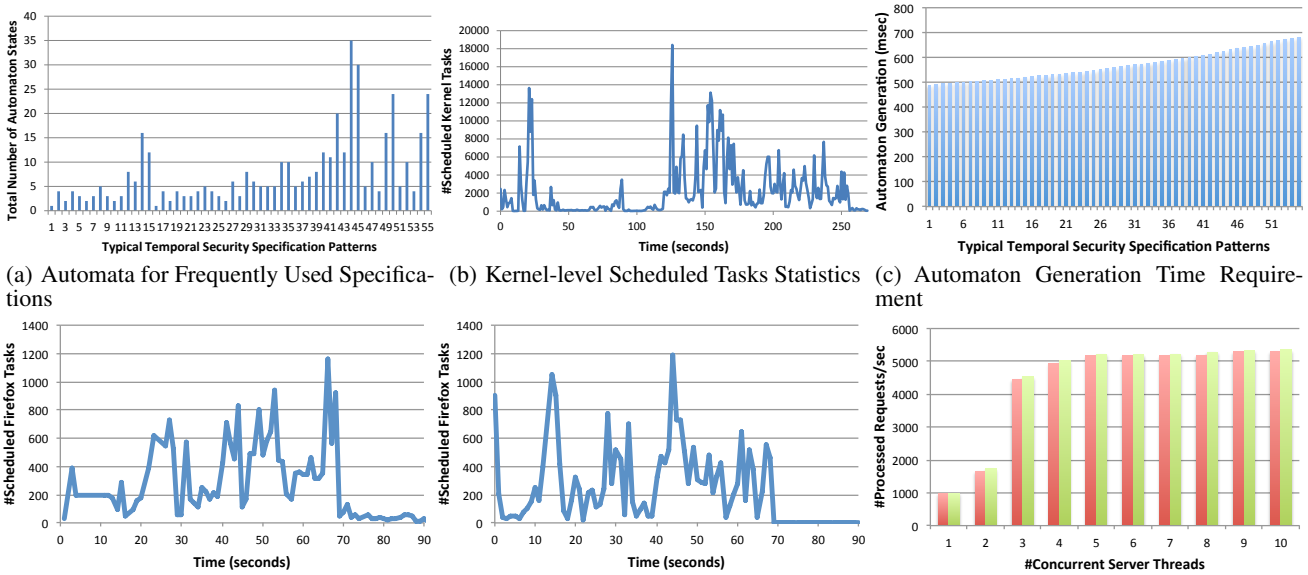
monitoring important aspects of the target system, such as the filesystem integrity using, for instance, periodic hash function-based scans [7]. In case a malicious activity is identified, Sechduler receives the triggered intrusion detection system alerts that cooperatively report the system's current security state. Sechduler goes through its policy repository dynamically and selects the relevant (possibly empty) subset of policy rules that correspond to the system's current state. Sechduler then constructs a single system-wide temporal logic-based predicate using the collected policy rules, and converts the predicate into a Büchi automaton-based monitor automatically. The automaton is compiled as a loadable kernel module and inserted into the running operating system kernel. The modified kernel scheduler notices the inserted module, and from then on verifies its individual task scheduling decisions using the loaded monitor. Additionally, if needed, it enforces the policies by reconfiguring the system's resource allocations, i.e., scheduling decisions, adaptively.

## Preliminary Results

We deployed Sechduler in a testbed environment and evaluated various aspects of its operation. Figure 1 shows the $B(\varphi)$ automaton in a *never claim* format in Promela [2].

As Sechduler verifies whether each scheduled tasks should be given CPU access, we collected statistics of the kernel-level scheduled tasks during a normal host computer usage session. Figure 2(b) shows the number of scheduled tasks for each second during the session. In particular, the session included a Web browser launch followed by an Office document editor application spawn. As demonstrated, the number of scheduled tasks can go up to $18K$ per second during an normal computer usage session. We measured the time requirements for the policy-to-automata conversion for the typical linear temporal logic-based system specification policies [1]. Figure 2(c) shows the results for individual temporal security policies. As shown in the figure, Sechduler completed the conversion for individual temporal requirements in approximately 0.58 seconds on average. This suggests that Sechduler can scale well for real-world settings where many requirements may be involved in the final logic-based predicate.

**Case Study: Sensitive File Modification.** We show how Sechduler protects a target host system once the system is hit by a sensitive file modification attack. Samhain was deployed as the attack consequence detection system. Specifically, we modified its configuration, i.e., `/etc/samhain/samhainrc`, to monitor the files and directories in which we are interested, and configured it to report events with at least `crit` severity level. Initially, we created its initial database, i.e., `/var/state/samhain/samhain_file`, using `samhain -t init`, and its database was updated, using `-t update`. During the operational mode of the system, Samhain was configured to check the marked sensitive files and directories




(a) Automata for Frequently Used Specifications (b) Kernel-level Scheduled Tasks Statistics (c) Automaton Generation Time Requirement

(d) Scheduling Statistics of the Trojan Firefox (e) Statistics on a Sechduler-Enabled Kernel (f) Overall Sechduler Performance Overhead

**Figure 2: Sechduler Evaluation Results**

against its database and fire an alert upon identifying a modification or access (depending on the policy defined in the configuration file).

To simulate an attack, we implemented a trojan Firefox that modified sensitive user files that had been marked to be monitored by Samhain. Figure 2(d) shows the malware's scheduling activity statistics within a non-Sechduler aware kernel. Consequently, Samhain fired the alert, and Sechduler performed three tasks. It 1) called the `setsec` system call and lowers the Firefox's security level variable within the kernel; 2) spawned a comprehensive ClamAV virus scan on the Firefox's executable; and 3) compiled the triggered alert's corresponding policy module and loaded it on the kernel dynamically.

Enforcing the loaded policy, Sechduler manipulated the task selection procedure within the kernel scheduler to ensure that (from its point of view) the potentially malicious Firefox process did not get CPU access and waited for the ClamAV's green light. However, in our experiments, ClamAV triggered an alert denoting that the executable contains malicious content. Consequently, the suspended Firefox process was terminated by Sechduler and its executable was removed. Figure 2(e) a different run of the trojan Firefox on a Sechduler-enabled Linux kernel. As shown on the graph, Sechduler denies its requests for execution since the 69-th seconds and finally terminates the process. We implemented the process termination as a single countermeasure action; however, more complicated actions can be defined by policies and implemented.

It is important that Sechduler performs the runtime system security verification efficiently such that the system's overall throughout is not affected significantly. We measured the Sechduler's overall performace overhead on our testbed system's overall throughput. In particular, we employed the `ab` Apache Webserver benchmarking toolset to measure the system throughout. To make the webpage processing more CPU-intensive, we designed a very simple HTML webpage. For our server system, we define the overall performance measure as the number of requests that can be processed per second. Figure 2(f) shows how the system's throughput is affected by the runtime verification of individual task scheduling decisions. We believe that the overall performance overhead of the Sechduler solution can be further reduced by optimizing our code. For instance, several data structures that are searched frequently, with $O(n)$ complexity, can be redesigned for logarithmic search, and overal system performance improvement.

## Acknowledgments

The authors would like to thank the Office of Naval Research (Grant N000141210462) for their support.

## 1. REFERENCES

[1] DWYER, M. B., AVRUNIN, G. S., AND CORBETT, J. C. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st international conference on Software engineering* (New York, NY, USA, 1999), ICSE '99, ACM, pp. 411–420.

[2] JIANG, K., AND JONSSON, B. Using spin to model check concurrent algorithms, using a translation from c to promela. In *Proc. 2nd Swedish Workshop on Multi-Core Computing* (2009), Department of Information Technology, Uppsala University, pp. 67–69.

[3] KING, S. T., AND CHEN, P. M. Backtracking intrusions. In *Proceedings of the Nineteenth ACM symposium on Operating systems principles* (2003), vol. 37, pp. 223–236.

[4] PABLA, C. S. Completely fair scheduler. *Linux J. 2009*, 184 (Aug. 2009).

[5] ROUTRAY, R., ZHANG, R., EYERS, D., WILLCOCKS, D., PIETZUCH, P., AND SARKAR, P. Policy generation framework for large-scale storage infrastructures. In *IEEE Symposium on Policies for Distributed Systems and Networks* (2010), pp. 65–72.

[6] SATO, H., AND YAKOH, T. A real-time communication mechanism for rtlinux. In *Annual Confjerence of the IEEE Industrial Electronics Society* (2000), vol. 4, pp. 2437 –2442 vol.4.

[7] WOTRING, B., POTTER, B., RANUM, M., AND WICHMANN, R. *Host Integrity Monitoring Using Osiris and Samhain*. Syngress Publishing, 2005.

[8] ZONOUZ, S. A., JOSHI, K. R., AND SANDERS, W. H. Floguard: cost-aware systemwide intrusion defense via online forensics and on-demand ids deployment. In *International conference on Computer safety, reliability, and security* (2011), pp. 338–354.