NC STATE UNIVERSITY

A Secure Resilient Real-Time Recovery Model, Scheduler, and Analysis

Abdullah Al Arafat[1], Sudharsan Vaidhun[2], Bryan C. Ward[3] and Zhishan Guo[1]

[1] Department of Computer Science, North Carolina State University
[2] Department of Electrical and Computer Engineering, University of Central Florida
[3] Department of Computer Science, Vanderbilt University

Outlines

- Background
- Problem and Assumptions
- Model
- Scheduler
- Analysis
- Experimental Results
- Conclusion

Background-Security (Embedded and Connected systems)

Fig: Attack at Smart Vehicle

Background-Security (Embedded and Connected systems)

Fig: Attack at Smart Vehicle

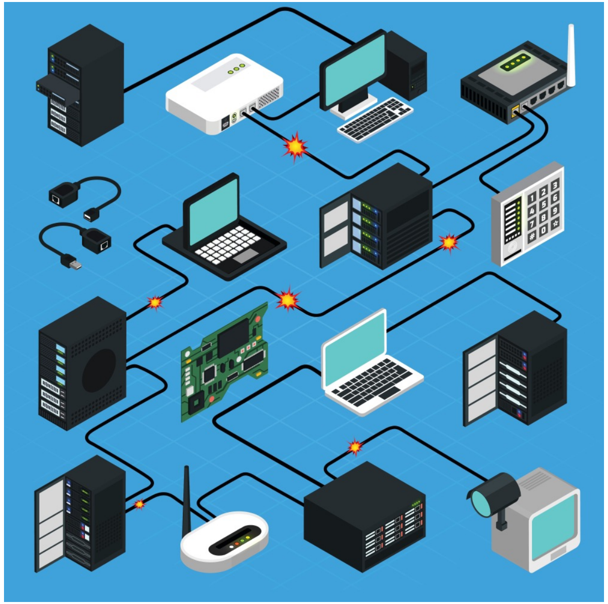


Fig: Attack through network

Background-Security (Embedded and Connected systems)

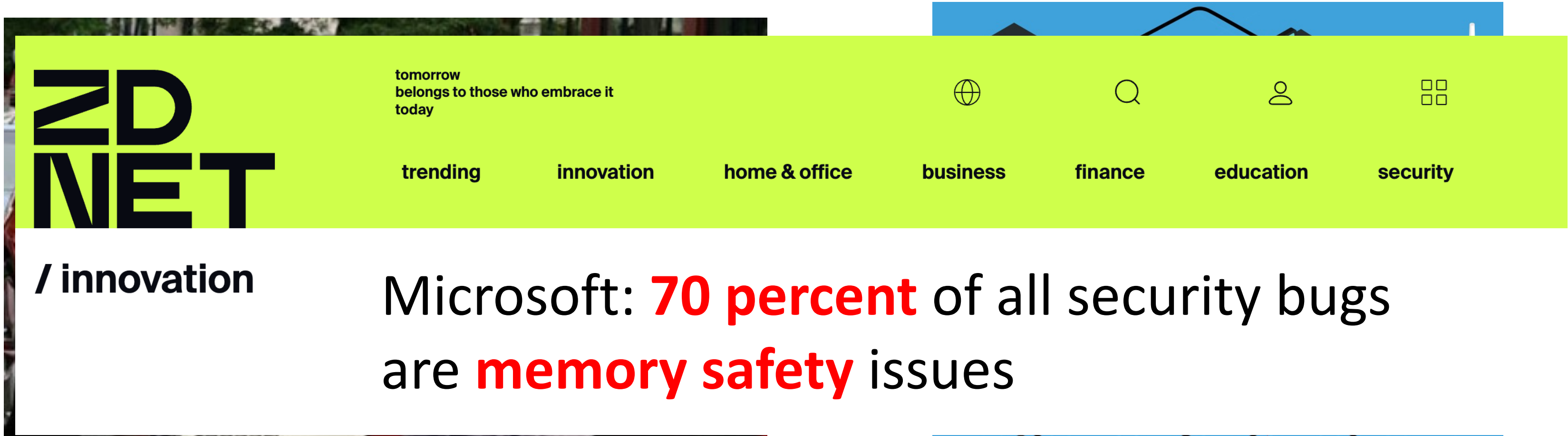


Fig: Attack at Smart Vehicle

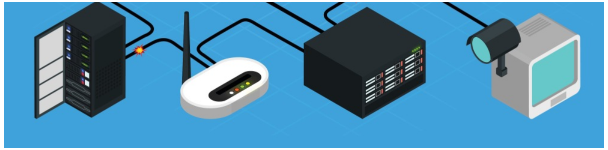
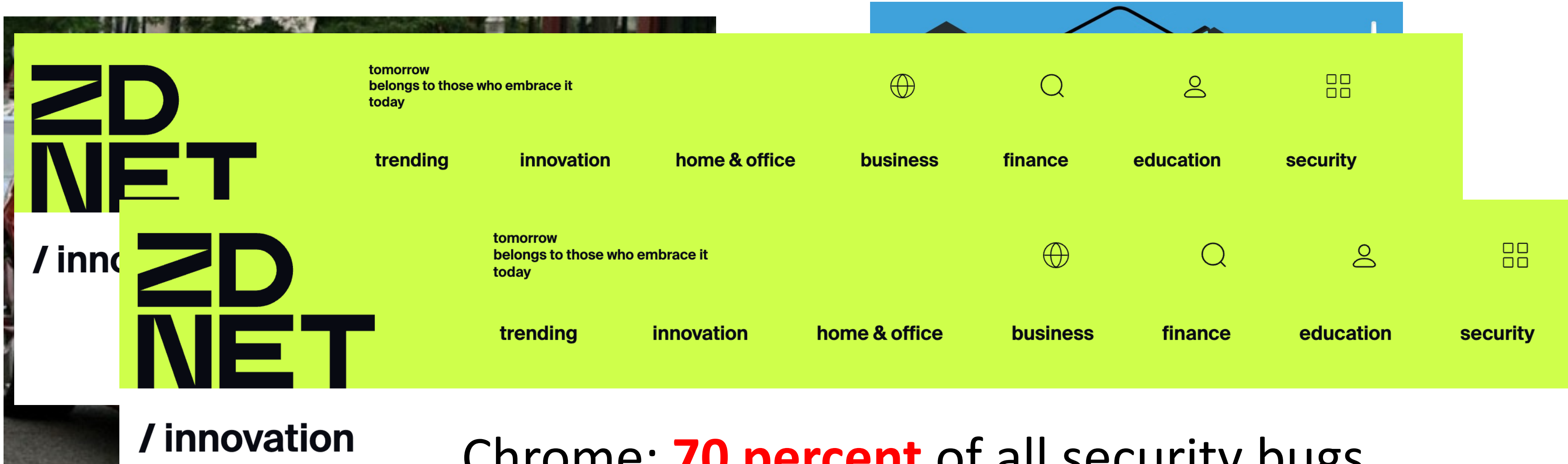


Fig: Attack through network

Background-Security (Embedded and Connected systems)



Chrome: **70 percent** of all security bugs are **memory safety** issues

This Photo

Fig

Fig: Attack through network

Background—Real Time Systems (RTS)

- Modern Design
  - Heterogeneously Platform
  - Hierarchical implementation with real-time kernel
    - Often require to share memory/resource with non-real-time processes

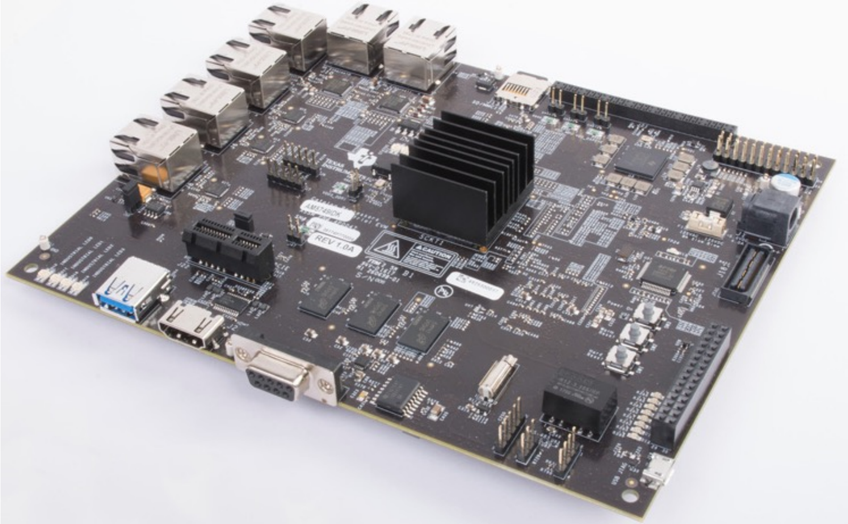


Fig: Heterogeneous Platform

Background—Real Time Systems (RTS)

- Modern Design
  - Heterogeneously implemented with non-real-time components (due to SWaP-C constraint)
  - Hierarchical implementation with real-time kernel
  - Connectivity (e.g., CPS, IIoT)

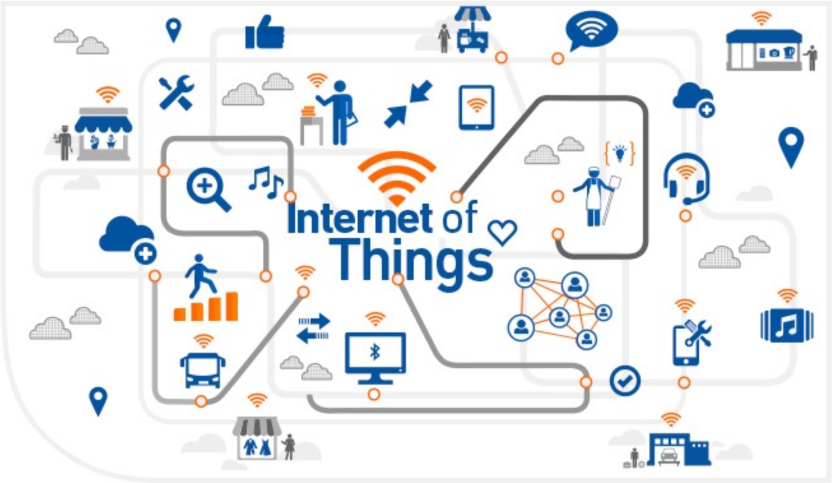


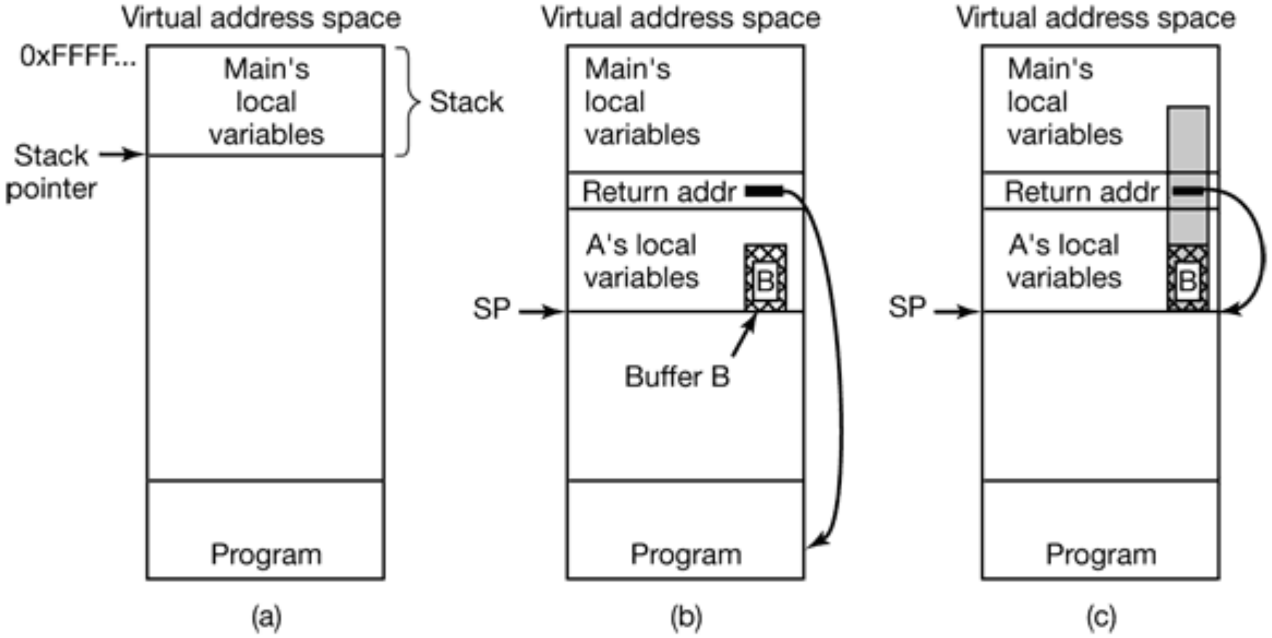
Fig: IoT

Background—Security Threats

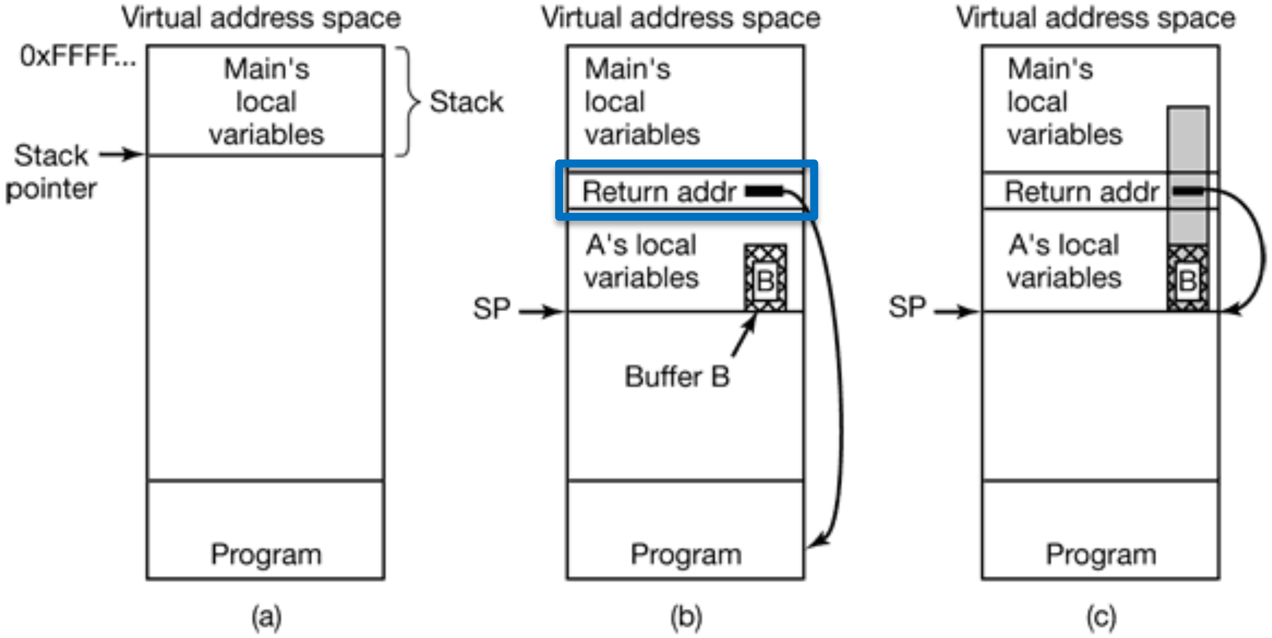
- Security concerns of RTS
  - Modern RTSs are vulnerable to security threat
  - Memory-based attacks, e.g., Control Flow Hijacking[1] and network-based attacks e.g., Mirai Botnet[2]

[1] “Control-flow integrity for real-time embedded systems”-ECRTS’19
[2] “Understanding the mirai botnet”-Usenix Security’17

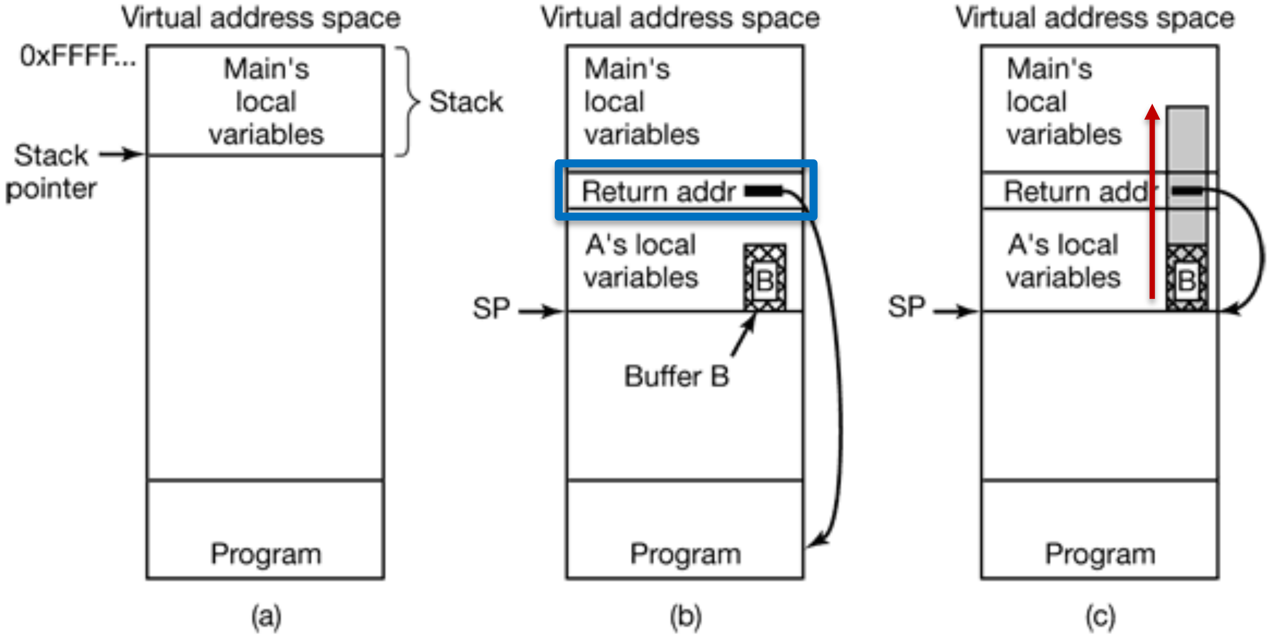
Background—Memory-based Attack (example – Buffer Overflow)



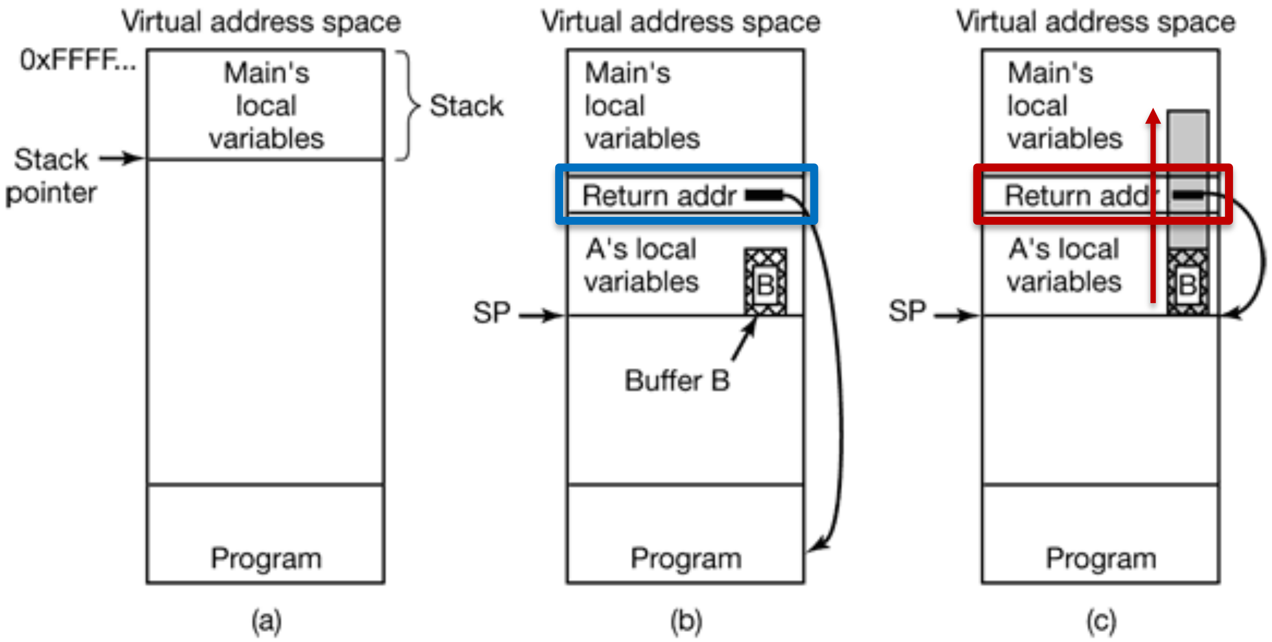
Background—Memory-based Attack (example – Buffer Overflow)



Background—Memory-based Attack (example – Buffer Overflow)



Background—Memory-based Attack (example – Buffer Overflow)



Background—Defense

- Runtime Defense, e.g., CFI[1], DFI[2]
  - **Instrumented** with real-time tasks
    - No need to schedule separately
    - Detect anomaly in real-time
    - Prevent the threat by crashing/killing the attacked task

[1] “Control-flow integrity for real-time embedded systems”-ECRTS’19
[2] “RT-DFI: Optimizing data-flow integrity for real-time systems”-ECRTS’22

Problem

- Can we develop **Security-Resilient** RTS model implemented with a Runtime Defense without crashing a critical task under attack?

- This work:
  - A Resilient System **Model**
  - **Scheduler** of the proposed model
  - **Analysis** of the proposed scheduler

Assumptions

- RTOSs are **trustworthy**[1]

[1] "RT-TEE: Real-time system availability for cyber-physical systems using ARM TrustZone"-S&P'22

Assumptions

- RTOSs are **trustworthy**[1]
- Security threats are exploited through **Memory Corruption**

[1] “RT-TEE: Real-time system availability for cyber-physical systems using ARM TrustZone”-S&P’22

Assumptions

- RTOSs are **trustworthy**[1]
- Security threats are exploited through **Memory Corruption**
- RTS are implemented with a **Runtime defense** technique
- Security event *can be detected at or before the completion* of attacked task

[1] "RT-TEE: Real-time system availability for cyber-physical systems using ARM TrustZone"-S&P'22

Assumptions

- RTOSs are **trustworthy**[1]
- Security threats are exploited through **Memory Corruption**
- RTS are implemented with a **Runtime defense** technique
- Security event *can be detected at or before the completion* of attacked task
- Workload can be classified as **security critical** and **non-critical**
  - ‘Security’ as a *new dimension* of criticality in MCS
  - Less security-critical tasks can be dropped during a security event
    - Minimize attack threat surface

[1] “RT-TEE: Real-time system availability for cyber-physical systems using ARM TrustZone”-S&P’22

Assumptions

- RTOSs are **trustworthy**[1]
- Security threats are exploited through **Memory Corruption**
- RTS are implemented with a **Runtime defense** technique
- Security event *can be detected at or before the completion* of attacked task
- Workload can be classified as **security critical** and **non-critical**
  - 'Security' as a *new dimension* of criticality in MCS
  - Less security-critical tasks can be dropped during a security event
    - Minimize attack threat surface
- After the detection of security event, system goes through critical mode
  - Take necessary actions to recover the system

[1] "RT-TEE: Real-time system availability for cyber-physical systems using ARM TrustZone"-S&P'22

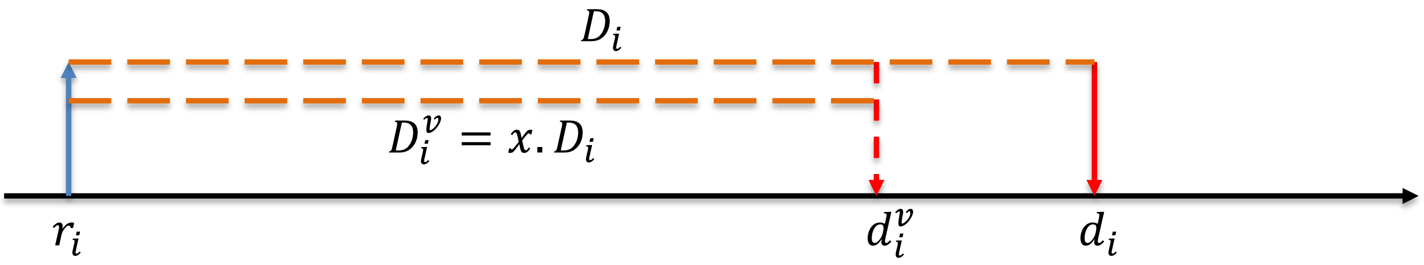
Model

- Workloads [$\tau' = \{\tau_1, \tau_2, \dots, \tau_n\}$, and $\tau_i = \{C_i, D_i, T_i, \varsigma_i\}$]
  - Security critical ($\varsigma_i = 1$) and non-critical ($\varsigma_i = 0$)
  - A **recovery task** ($\tau_R = \{C_R, T_R\}$) for each security-critical task
  - Attacked task will get a full-execution by its deadline

Model

- Workloads [$\tau' = \{\tau_1, \tau_2, \dots, \tau_n\}$, and $\tau_i = \{C_i, D_i, T_i, \varsigma_i\}$]
  - Security critical ($\varsigma_i = 1$) and non-critical ($\varsigma_i = 0$)
  - A **recovery task** ($\tau_R = \{C_R, T_R\}$) for each security-critical task
  - Attacked task will get a full-execution by its deadline
- System
  - **Uniprocessor** system
  - Only one task can be attacked at a time instant
    - **Note:** Any tasks can be under attack, however, one task can be exploited by the attacker using code pointer
  - Two operating system modes: **regular** and **recovery** mode

Scheduler



- Normal mode:
  - Calculate **virtual deadline** ($D_i^v = x\ D_i$) for each security-critical task
  - All **security tasks** are executed by their **virtual deadline** and **non-security** tasks by their **original deadline**

Scheduler

- Normal mode:
  - Calculate **virtual deadline** ($D_i^v = x\, D_i$) for each security-critical task
  - All **security tasks** are executed by their **virtual deadline** and **non-security** tasks by their **original deadline**
- Recovery mode:
  - All **security-critical** tasks (except targeted task) continue to receive **normal execution** budget and meet their **original deadline**
  - Targeted task receives full **re-execution** from mode-switch instant to its original deadline
  - Recovery task executes and meets deadline

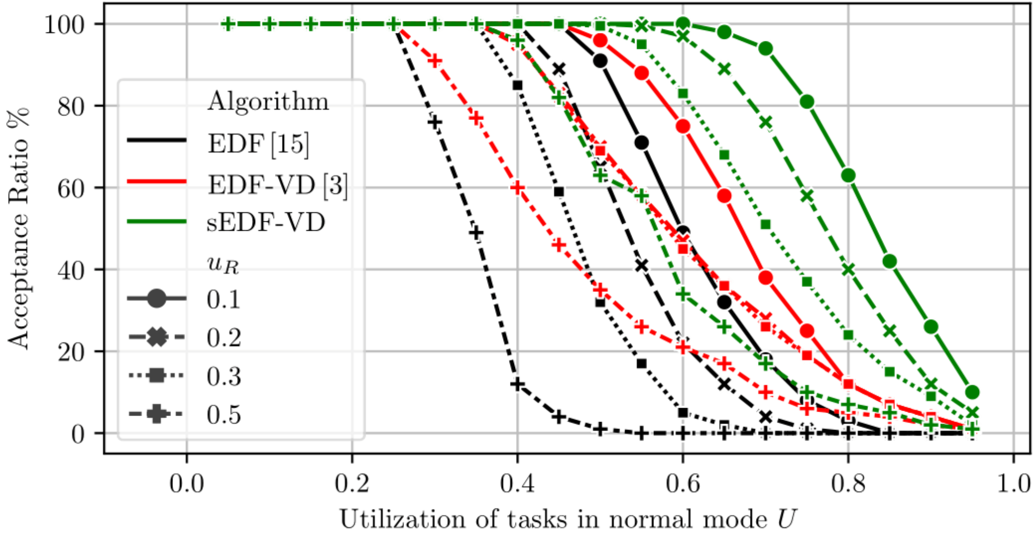
Analysis

- Utilization-based test
  - **Normal Mode:** $U_{\varsigma} + \frac{U_{\sim\varsigma}}{x} \leq 1$
    - U_{ς} –utilization of security-critical tasks
    - $U_{\sim\varsigma}$ –utilization of non-security-critical tasks
    - x –deadline shrinkage parameter
  - **Recovery Mode:** $xU_{\sim\varsigma} + U_{\varsigma} + u_t + u_R \leq 1$
    - u_t –utilization of targeted task
    - u_R –utilization of recovery task

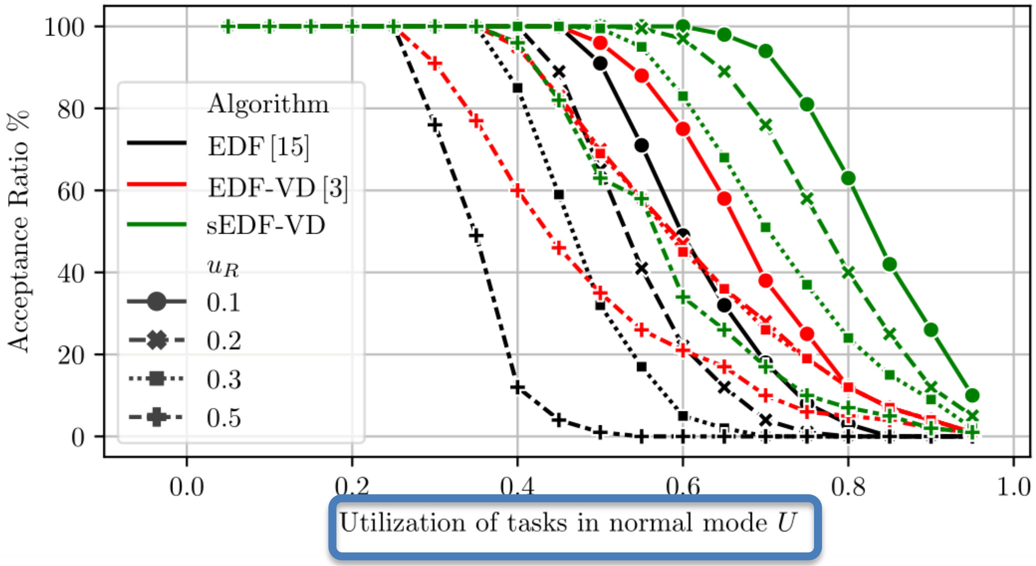
Performance Evaluation

- Baselines
  - EDF
    - Doubled the execution of security-critical tasks
  - EDF-VD
    - Model the workloads as MC workloads by doubling the execution-time of security-critical tasks
  - sEDF-VD (Ours)

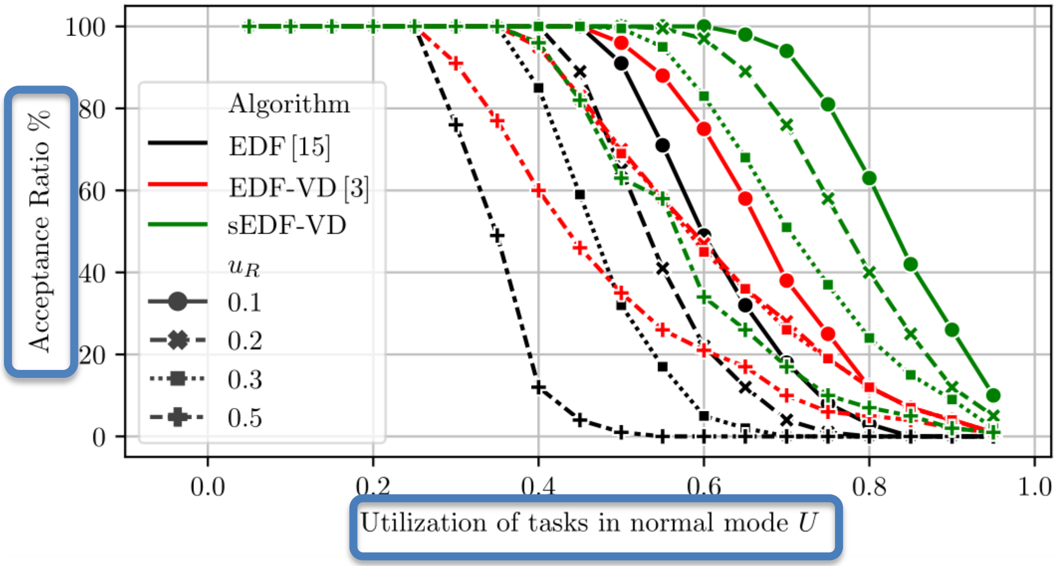
Performance Evaluation



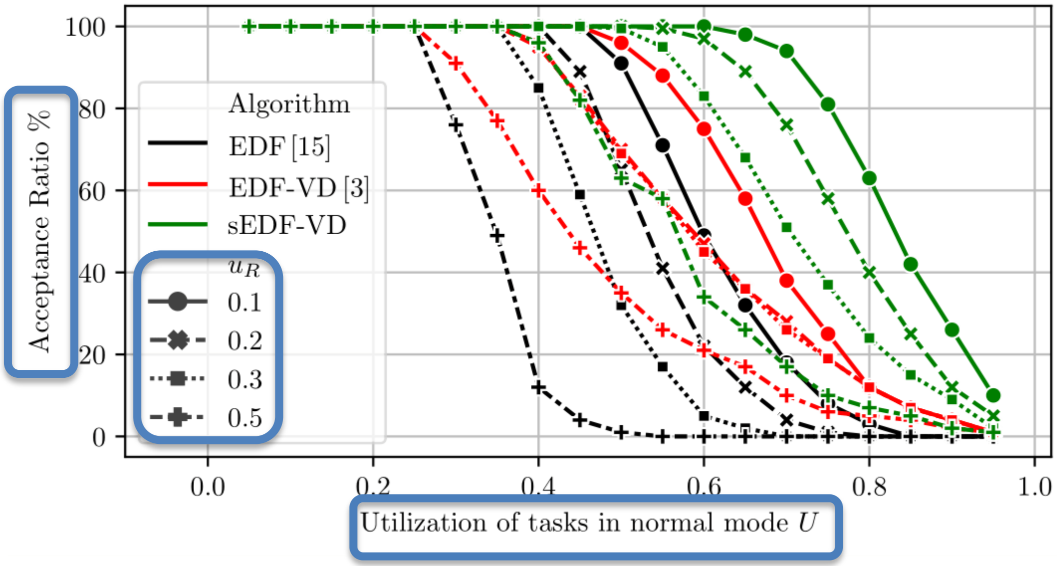
Performance Evaluation



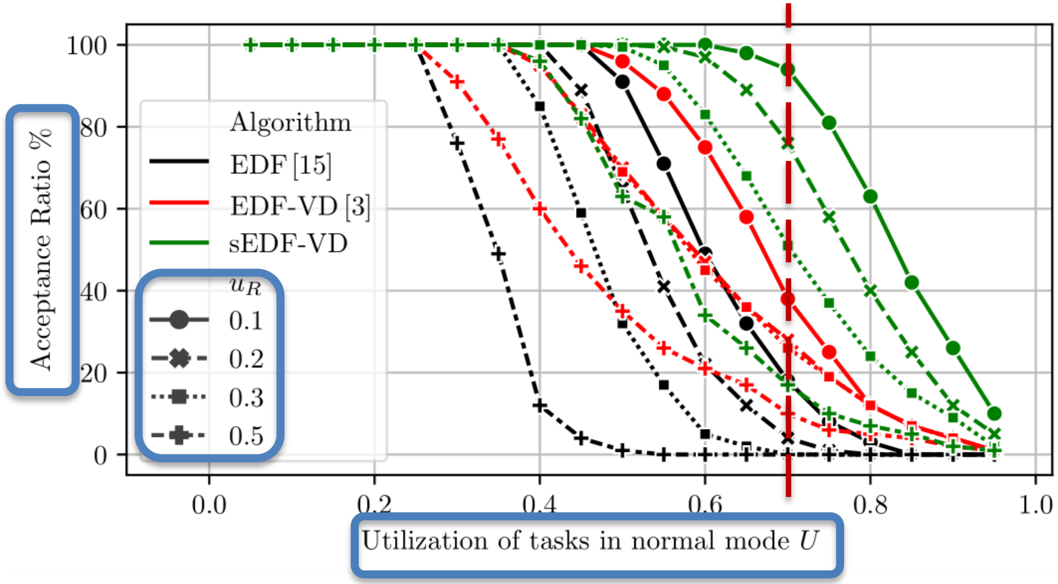
Performance Evaluation



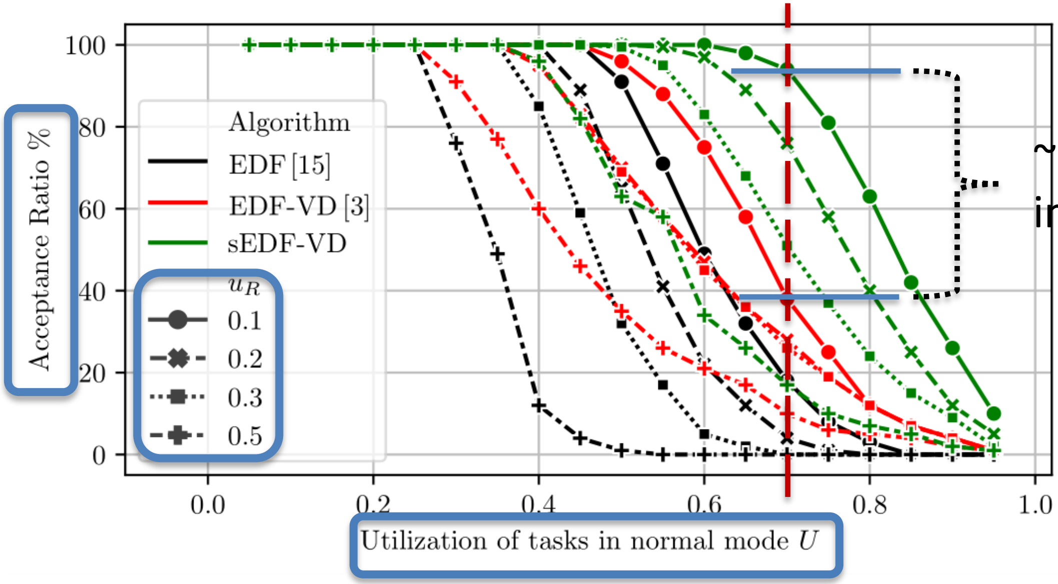
Performance Evaluation



Performance Evaluation



Performance Evaluation



~50% improvement

Related Works (Defense Approaches)

- Intrusion Detection Systems
  - Monitor security activity and potentially detect the security threat
    - Do not prevent the threat
  - Several Important works added additional security tasks, e.g., Contego[1]
    - Scheduling Overhead—Need to schedule the security tasks along with regular task
    - Non-real time—detection of security event before compilation of attacked task is not guaranteed

[1] “Contego: An adaptive framework for integrating security tasks in real-time systems.”-ECRTS’17

Conclusion

- Proposed a resilient real-time model that can protect security-critical operations
- Developed efficient deadline-based scheduler for the proposed model
- Presented utilization-based schedulability analysis for the scheduler
- *Future works:* efficient analysis, and system implementation