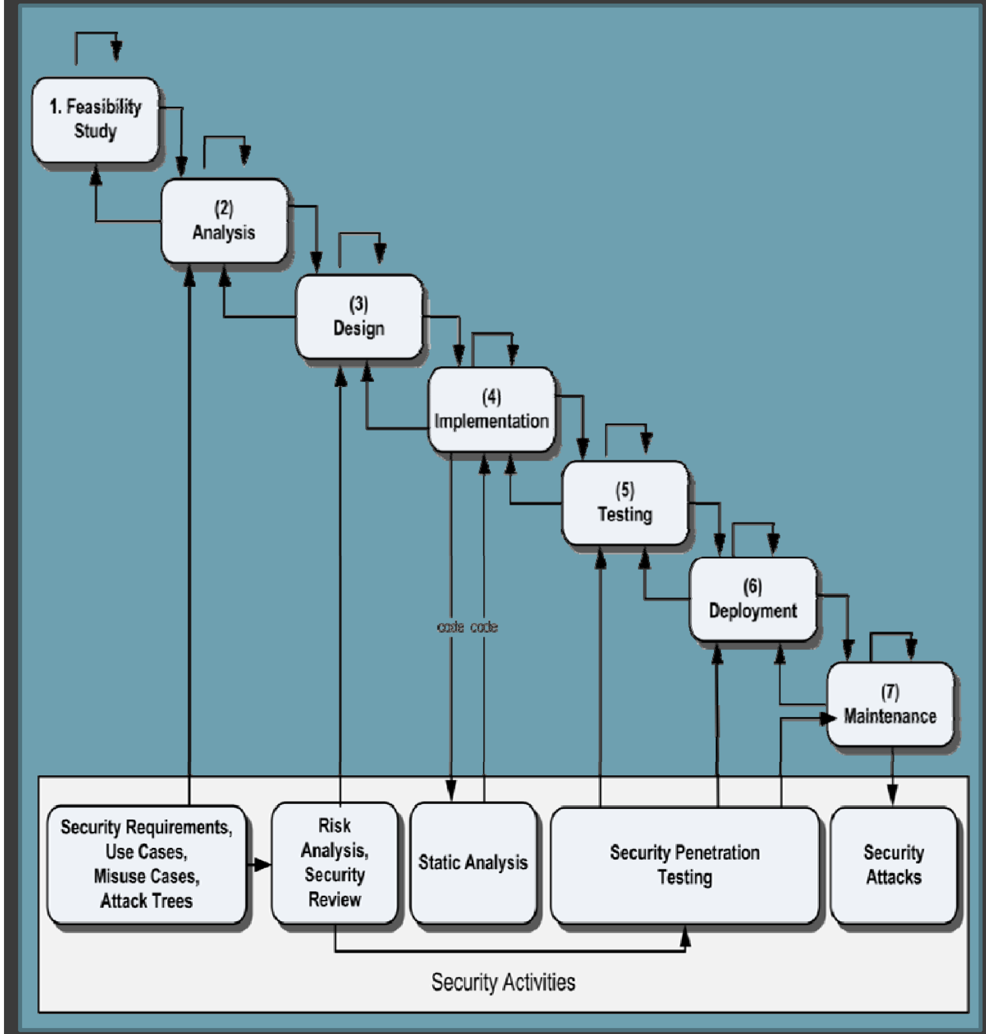
INTEGRATING CONTRACT-BASED SECURITY MONITORS IN THE SOFTWARE DEVELOPMENT LIFE CYCLE

Alexander M. Hoole, Isabelle Simplot-Ryl, Issa Traore

Introduction

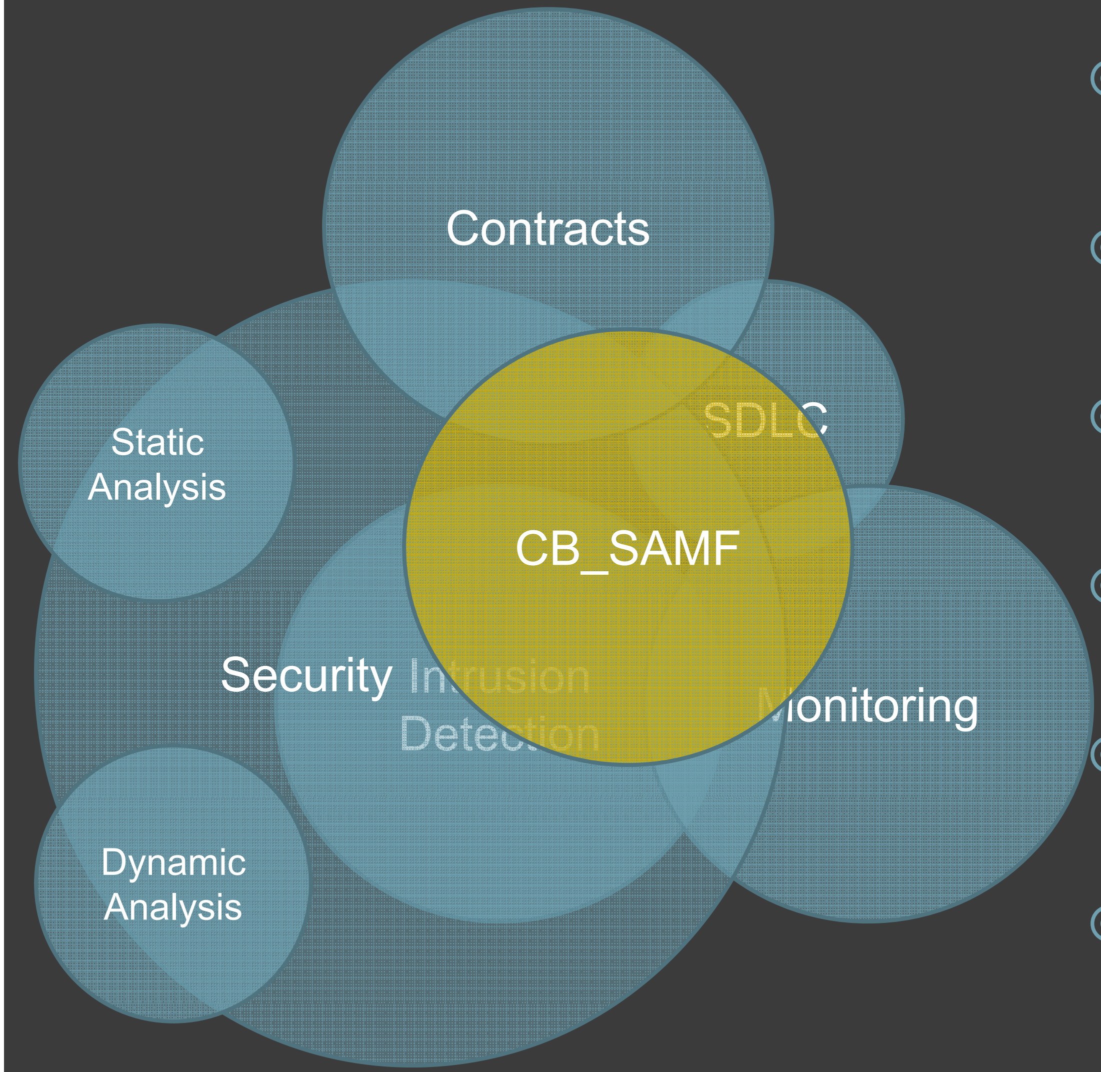


- Security vulnerabilities are growing:
  - *Connectivity*
  - *Extensibility*
  - *Complexity*
- We can no longer rely on Tiger teams/penetrate & patch
- Integrate security into every phase of SDLC:
  - Work has begun…
  - Much remaining to be done…
- How can we reduce vulnerability defects?

Motivation

- **Software containing vulnerabilities**
  - Need for improved software engineering practices
    - Safety, reliability, dependability, quality, **SECURITY**, …
  - Need methodologies and tools for:
    - Identification, monitoring, and verification of defects
    - Repairing and removal of defects
- **Existing approaches do not adequately address the issue**
  - Firewalls, IDSs, IPSs, static analysis, etc
- **Need for a unifying framework for:**
  - Specification of assertions for security requirements
  - System-wide monitoring framework for assertions
  - Assertion testing framework
  - Software security metrics

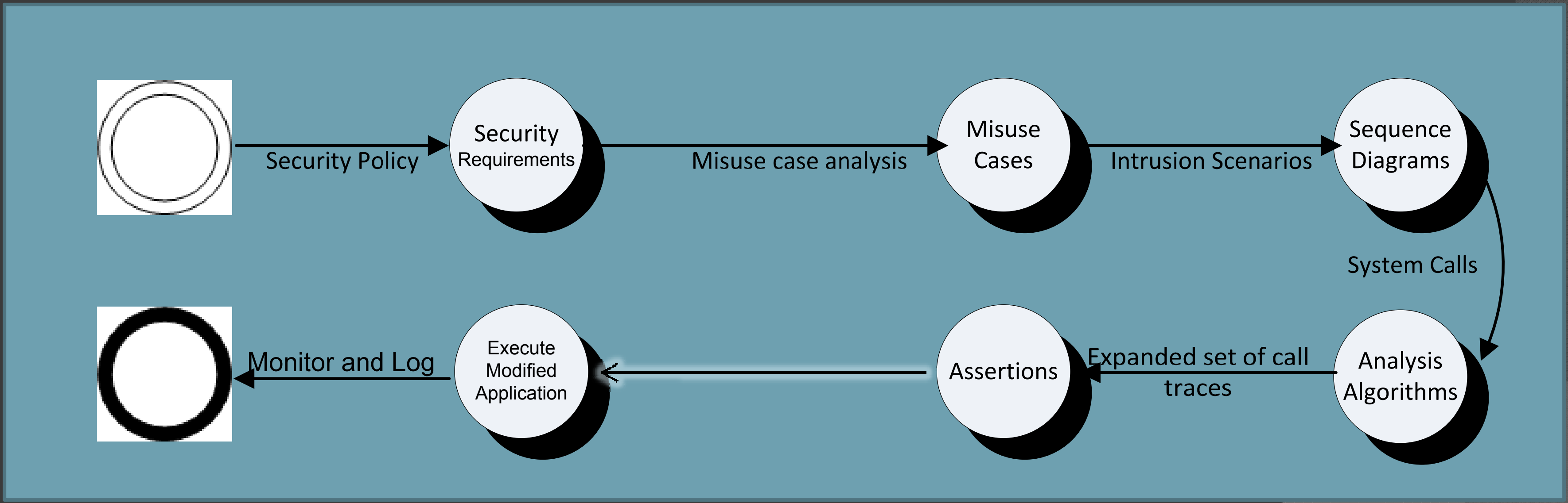
Context and Research Objectives



- Vulnerabilities exist at any software layer
- Monitoring applied as IDSs for security
- Security integration into SDLC has begun
- Analysis has begun to identify vulnerabilities
- Contracts appropriate for specification
- CB_SAMF unifies approach

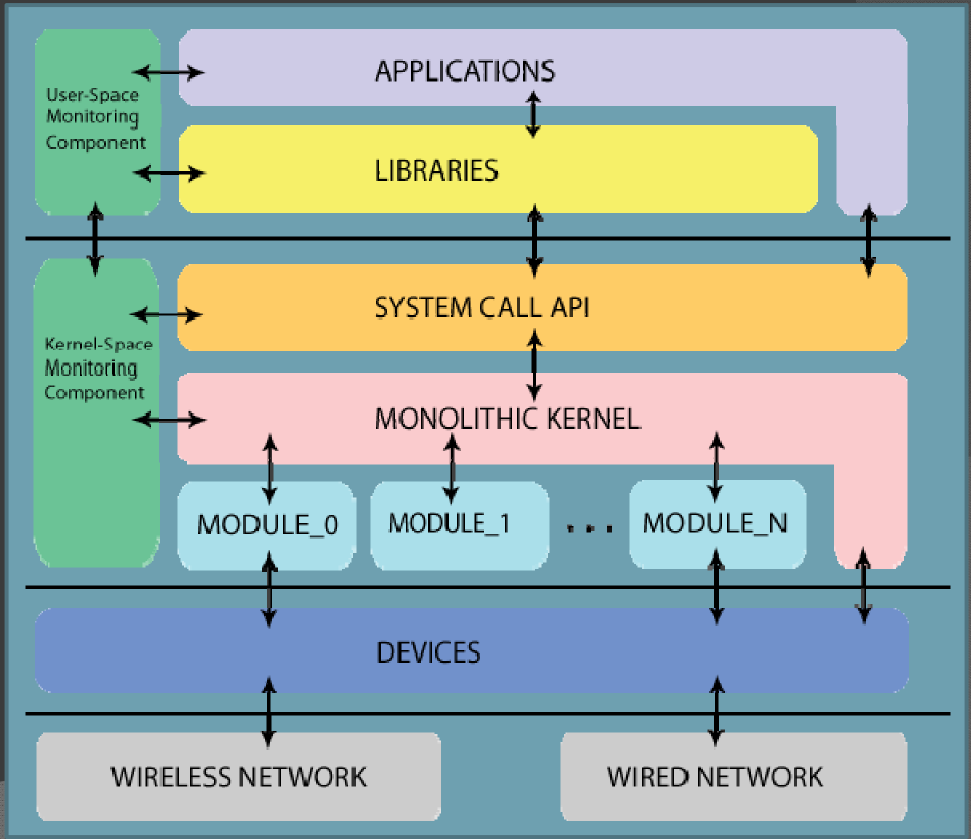
Proposed Approach

- CB_SAMF focuses on the last three phases of the following methodology:



Architectural Considerations

- Contract-Based Security Assertion Monitoring Framework (CB_SAMF)
  - Only as secure as weakestlink
  - Contracts for multiple layers
  - Monitoring at multiple layers
  - Security evaluation of system



EAGLE

- Framework that includes a range of finite traces monitoring logics
- Implemented for Java
- Example:

“Whenever P occurs then Q must occur within 10 seconds”

```
max Always (Form F) = F ∧ ○ Always (F)
minEventuallyAbs (float t, Form F)= currentTime()
   ≤ t ∧ ((¬F) → ○ EventuallyAbs(t,F))
minEventuallyRel (float t, Form F) =
   EventuallyAbs(currentTime()+t,F)
mon M = Always (P→ EventuallyRel(10,Q))
```

EAGLE

- Very rich and covers a lot of other approaches
- Lacks features for security monitoring, for example:
  - Vulnerabilities that involve environmental resources
  - Vulnerabilities that involve multiple layers
  - Real runtime monitoring
  - Reaction framework for collecting data when vulnerabilities occur
  - …

Contracts for security

"Pre ⇒Post"

- Not sufficient for security

Proposition

- Requirements – as preconditions
- Guarantees – as postconditions
- References – as invariants
- Context – as environmental information
- History – as knowledge about the past
- Response – as reactive measure

Model – extension of EAGLE

- **C := B (A{E}) {A{E}};**
- **E := {CONT} | {HIST} | {RESP};**
- **A := {R}{M};**
- **R := {max|min} N(T1x1, ..., Tnxn) = F;**
- **M := mon N = F;**
- T := Form | primitive type;
- B := symbol | HEX address;
- F := exp|true|false|$\neg$F|F1 $\wedge$ F2|F1 $\vee$ F2|F1 $\Rightarrow$ F2|$\bigcirc$F|$\ominus$F|F1 · F2|N(F1, . . . , Fn)|xi;
- CONT := env N | res N;
- HIST := trace N | runningsum N | runningavg N;
- RESP := core N | term N | kill N | log N;

Monitors for contracts

- To check contracts at runtime
- Contracts are associated with breakpoints
- Monitors can also evaluate context and history

```
max R(string s1, string s2)
  =¬Always({env_contains(s1,s2)})
E = env R("PATH", ".")
```

- Monitors can react with the defined response

Application of Contract

Buffer overflow vulnerability at: *symbol_bp*

E = log*bu f f er_log*
min*R*(*int* k) = *Sometime*(y == k)
mon*M* = *Always*(x > 0 → R(x) ^ x ≤ y)
C = *symbol_bp M E*

GENERATE PROBES

insmod catch_buffer_probe.ko
breakpoint=0xe0930000
buffer_addr=0xe09305e4

1. Identification of vulnerabilities
2. Representation in LTL/contract
3. Generation of probes
4. Execution of system and insertion of probes

Monitor with Probe

```
char *breakpoint; /*parameter for breakpoint*/
char *buffer_addr; /*parameter for buffer*/
module_param(breakpoint, charp, 0400);
module_param(buffer_addr, charp, 0400);
…
unsigned long *bp; /*breakpoint address*/
char *bad_buffer; /*buffer*/
unsigned long addr; /*temporary holder for incoming addr*/
struct kprobe kp; /*kprobe*/
…
int j_write_target(struct file *file, const char *buffer,
                   unsigned long count, void *data)
{
        int len = 0;
        ...
        len = strlen(bad_buffer);
        printk("The length of the target buffer is: %d\n", len);
        if (count > len) {
        /* Security Violation Reaction Here */
        printk("VIOLATION!!!\n");
}
jprobe_return();
/*NOTREACHED*/
return 0;
}
…
```

1. Identification of vulnerabilities
2. Representation in LTL/contract
3. Generation of probes
4. Execution of system and insertion of probes
5. Monitor execution for violations

Evaluation of Contract

SOP = Use-case profile + Misuse-case profile

Stress system using SOP identified fault-injection

…
 # echo -n "0123456789012345678901234567890123456789012345678901234567890" > /proc/target
….

Collect and store metric data

Resolve verified vulnerabilities and repeat
Use metric to compare relative pre/post effect on security

1. Identification of vulnerabilities
2. Representation in LTL/contract
3. Generation of probes
4. Execution of system and insertion of probes
5. Monitor execution for violations
6. Identification of security operational profile
7. Execution of fault-injection framework
8. Application of metrics
9. Resolve any verified vulnerabilities and repeat

Conclusion

- A methodology and tool set for improving security during development and testing
- A theoretical model for an assertion-based security monitor based on contracts and probes
- A low performance-cost prototype monitoring framework that is able to detect and respond to several well known attacks
- Potential to monitor, track, and counteract security related assertions through all software layers in the system
- Potential discovery of additional metrics to assess security of monitored systems

QUESTIONS?

This work is partially supported by NSERC, MITACS, and CPER Nord-Pas-de-Calais/FEDER Campus Intelligence Ambiante.