D&I in Cyber Security & Complex Software

J. Sukarno Mertoguno
sukarno.mertoguno@navy.mil
Vision

- **Secure, Dependable and Resilient Information and Information Infrastructure**
  - Self-aware
  - Leak-proof, and leak-aware
  - Not susceptible to external disruption, such as DDoS
  - Operational under attack, graceful degradation

- **Minimal Manual Intervention**
  - Automated, requires little manual intervention
  - Self-diagnostic, self-healing

Secure and Dependable Information Infrastructure with Minimal ‘Baby Sitting’ is an Essential Component for Achieving Successful Navy’s Missions
Roots of Issues

- **Cyber warfare → Asymmetric**
  - **Defender**
    → relatively large footprint & static in configuration & properties
    → large static target
  - **Attacker**
    → smaller footprint, dynamic, opportunistic, have time, persistent
    → small moving target

- **Most security breaches stem from exploitable system vulnerabilities**
  The source of vulnerabilities lays with the
  - user,
  - protocol and policy,
  - software application and operating system
  - and occasionally the hardware subsystem.

- **COTS based hardware and software exacerbate the problem**
ONR’s Proposed Approaches to Cyber Security Challenges

- **Addressing Asymmetric (static, large footprint):**
  - Artificial diversity → mostly host, potentially network
  - Dynamic re-configuration → host and network
  - Dis-information → host

- **Addressing Vulnerable Systems & Inherently Insecure COTS:**
  - Security aware software development environment
  - Active detection, active defense
  - Fight through damage: failure recovery, damage control, graceful degradation
  - New sw & hw architecture developed with security “baked-in”
    → GOTS (new architecture or modification of COTS)
    → Can DoD influence COTS development ?
    → Should any IA/IO features be strategically kept out of COTS ?
Applied (& Basic) Science Research Direction

F1: Hardening Hosts - System
- Robust Autonomic Host (monitoring, self-aware, self-healing, graceful degradation, ex-filtration detection)
- Secure Web, and Web Applications (browser, scripting, etc)
- Secure Software Development Environment (development, verification)
- Data Ex-filtration Awareness

F2: Hardening data sharing, and integrity (process-process, user-machine, user-process) - Data
- Trust management model and calculus
- Confidentiality in collaboration
- Mobility and Collaboration
- Automatic Crypto Function Generation

F3: Hardening the Network
- Toward automated dynamic re-configuration (mitigating DDOS)
- Automated network device configuration & policy management
- Enhancing malicious traffic detection, and source tracking
F1: Hardening The Host

Thrusts:

T1: Robust Autonomic Host
  (monitoring, self-aware, self-healing, graceful degradation, etc.)

T2: Secure Web, and Web Applications
  (browser, scripting, etc)

T3: Secure Software Development Environment
  (development, verification)

T4: Data Exfiltration Awareness
  (need more investigation, not emphasized in this phase)
Intertwined sub-thrusts:

a. Self aware computing

b. Automated reasoning and defense for security and integrity

Develop as an Integrated Framework with a Modular Architecture, for platform longevity
Robust Autonomic Computing System
Integrated Framework with Modular Architecture

Foundation:

- Flexible and capable system reasoning infrastructure
- Robust and accurate system reasoning and model
- Capable & stealthy data acquisition & control
- Comprehensive monitoring,
- Robust and capable event detection system

Above The Foundation (of Event Detection and System Reasoning)

- Self-healing system (system recovery via repair)
- Graceful degradation (recovering functionalities via salvaging & remapping)
- Artificial diversity (multiple and/or overlapping implementation of functions)
- Dis-information (Passive-Aggressive → defense & offense)
Dis-information: Deceptive Cyber Moat
Another Layer of Dis-information

Operation & Assumptions:
• The Cyber Moat presents a **mirage** of a LAN, complete with mirage **hosts** & mirage **network devices**
• The **actual host** is only an item **within** this **mirage LAN**
• The configuration and **topology** of the mirage LAN may **change dynamically**
• Many cyber Moats within the actual LAN may collaborate for consistency of dis-information
• The **system** (actual host) behind the moat has its **own** independent **defense**

**A Morphing Cyber Trap** where **intruders** who enter the mirage environment (Cyber Moat)
• spend significant time **futilely exploring**
• and are **susceptible** to any **deception & dis-information** fed by the moat
Robust Autonomic Computing System

Research Topics

- Disinfo
- Self-Healing
- Graceful Degrad
- Artificial Diverse

Data Acquisition & Monitoring

- Detection
- Model & Reasoning
- Monitoring
- Data Acquisition

Syst. Reasoning & Artif. Diversity

- Detection
- Model & Reasoning
- Monitoring
- Data Acquisition

Self-Aware

System Reasoning & Resiliency

Collaborative Disinformation
Toward Stealthy Monitoring and Self-Aware Computing

- Self-aware computing system is a system capable of
  - sensing its own states,
  - observing the behavior of the systems components and applications running in the systems,
  - setting virtual trip wires, detecting and
  - reasoning about potential anomalies.

- Why
  - Self-Aware for robust security with Minimum “Baby-Sitting”
  - Stealth for dealing with adversarial situation

- Enabling
  - Active detection & active defense
  - Dis-information

- Potential approach
  - Hardware assisted system,
  - Virtual Machine,
  - Software instrumentation via re-write, etc.
Stealthy Monitoring

- Need to cover all possible abstraction levels
- May require semantic bridging to all levels
- Reasonable overhead for practicality & stealth (latency & cost)

An example for Stealthy Monitoring

Sequence of activities for normal event

Sequence of activities for critical event
Self-Aware Computing (SAC)

SAC components

- **Data acquisition**
  - Sensors within program execution, providing data to monitor

- **Monitor**
  - Collecting data, aggregating low-level events into higher-level ones, bridging levels of event/semantic,
  - Simple composition rules
  - Maintain state of event(s)

- **Detector**
  - Knowledge, models and algorithms, for recognizing event of interest.
  - Maintain state of system/model/algorithm

**Dis-information**

- Algorithm for dis-information strategy
- Maintain state for dis-information consistency within host over time \(\rightarrow\) for stealth reason
- May need to coordinate with peers, for global dis-information consistency \(\rightarrow\) for offensive
- Or may be just completely random, and inconsistent \(\rightarrow\) not trying to be stealthy, just confusing
Enabling Automated Reasoning on System Security and Integrity

- **Why**
  - To achieve automated self-defending, self-mitigating and self-healing system
  - By providing foundation for reasoning for system security and integrity; what tasks, what class of data, and what activities can safely be serviced, by which applications or functions.

- **Foundation for**
  - Failure recovery, damage control & graceful degradation
  - Artificial diversity & dynamic reconfiguration at the host

- **Important parameter**
  - Granularity of the module

- **Potential approach**
  - System software architecture with metadata for system component providing description and role (analogous to semantic web)
  - Other approaches to be solicited under BAA
Enabling Automated Reasoning on System Security and Integrity

Sequence of events & actions
- Suspicious event or anomaly detected
- Investigate, diagnose & classify event
- Assess compromise & damage → Aware of own states

Further actions:
Temporary fix, Repair, Remap, Scavenge

Requires
- Infrastructure that accommodates storing and inter-linking descriptors/models (metadata) for module and/or sub-module → integration space for complete system description/model
- Intelligent algorithm for analysis and assessment of security and integrity
Enabling Automated Reasoning on System Security and Integrity

- Metadata for integrity evaluation
- Configuration controller manage and monitor security and integrity
- Current system has configuration file, such as MSWindows registry file, not enough for reasoning

Repository:
- Centralized?
- Distributed?
- Hierarchical?
**Self-Healing and Graceful Degradation**

**Graceful Degradation**
- OK for particular cases $f(\text{user,object,state})$

**Self-Healing**
- Fixed & Immuni. Module

---

**Graceful degradation**
- Partial functionality on original code
- Granularity?
  - Composition
  - Spec-based

**Function remapping** (dynamic reconfig.)
- Full or partial functionality by using other code(s)
- Composition may be difficult for some applications
Artificial Diversity within system software (dynamic reconfig)

- Many implementations of a module
- Alleviate vulnerability due to implementation
- Diverse configuration reduce compromise propagation
- Re-configuration can mitigate compromise
- May or may not need event detection
Secure Web and Web Application

Why

- Web Interface (HTML & XML) becomes the popular and unifying interface for many applications
- External/downloaded program (script) executions are allowed & common
- Similar security issues parallel to that of system software
- Web browser code base has grown to the size of operating system’s
- Potential for compromise abound.

Related to T1:

- Monitoring,
- Malicious event detection,
- Prevention from security breaches

@ Web Environment Level,

- and Developing more secure scripting language
Why separate

- Many processes on web environment are often isolated from underlying operating system, by virtual machines
- Huge semantic gaps, tedious to bridge, can be handled locally
- Often less performance sensitive, and less critical
- Easier to recover, often clean restart is acceptable

Against separating

- Occasional need for correlating event at web layer to system layer
Software Development Environment for Secure
System Software & Applications

Why

- Breaches stem from exploitable vulnerabilities, mostly from software
- Deploying software devoid of vulnerabilities significantly improve system security
- Poor discipline and security awareness on software developers, especially in commercial* software
  (* where COTS are developed, rush to the market attitude)
- Creeping featuristis* drives complexity
  (* tendency for adding more and more features to an application)
- After thought features lead to less than proper implementation
- Legacy issues

*Integrated development environment for secure & correct-by-construction software with support for system reasoning*
Sub-thrusts

a. Software development environment for reducing programmer induced software vulnerability

b. Software verification methods for reducing vulnerability

Seamless integrated development & verification environment offers the biggest benefit
Security Aware Software Development Environment

Security aware software development environment

- Recognize and identify vulnerability in code
- Suggest correction for identified vulnerability
- Generate executable devoid of vulnerability

Potential approach

- Security & vulnerability aware compiler
- High-level design entry, with automated secure code generation
- High-level design entry, with automatic insertion of security primitive

Challenge with new development environment

- Software developer acceptance for new methods, and limitations
**Current Focus on New 6.2 Program**

Develop and demonstrate tools and an environment that lead to the generation of robust and secure codes for both the system (or operating system) level, as well as at the software application level.

a) Security and vulnerability aware compilers with automatic insertion of constructs that guarantee robustness and security of codes written in unsecured languages.

b) Automated generation of secure and robust codes from high-level description (design-entry) of function that leads to software this is both readable and efficient.

c) Methods that automatically capture and utilize workflow, thought/design-decision, and documentation during software coding that lead to functioning code that meets performance and security requirements.
Improving Software Robustness and Efficiency

Driving force
- Processor core clock speed reach practical limit ~4GHz (power issue)
- Percentage of sustainable # of active transistors decrease; Increase in # of transistors is not followed by comparable reduction in power
- Natural limit on parallelizability of computation
- Future projection → Heterogeneous Multicore

What future software needs
- Integrated programming environment for heterogeneous multicore
- Dealing with data movement among cores – for exploiting parallelization, and improving performance & efficiency
- Efficiency become more important in future software

Potential research topics
- Software development environment and deployment/installation strategy & tools toward enhancing software efficiency
- Integrated programming environment for heterogeneous (& homogeneous) multicore
- Performance aware (latency predictability) multicore software development environment – important for CPS
Integrated programming env. for hetero-/homo- geneous multicores

- Automated analysis for uncovering parallel construct for highly threaded applications – including time constrains (if in CPS), for both homogeneous & heterogeneous multi-cores
- Sw Devl environment for heterogeneous multi-cores, including the environment for exploring mapping of functions to various cores, and analysis & optimization for toward the most efficient implementation given the application and available resources.
- Dealing with data movement among cores – for exploiting parallelization, and improving performance & efficiency

Performance and latency predictability for multicore software

- An integrated into software development environment for CPS with control over data transfer sizes, buffer sizes, and latency characterization and support for tradeoff analysis between performance and predictability
- Additional signaling and preemption construct can be introduced into the above to potentially improve throughput, while still maintaining reasonable latency guarantee.
- Software performance estimation/modeling may also help analysis, prediction and help in controlling the latency important for real time CPS

Software development & deployment strategy for enhancing efficiency

- Next slide
Current Software development & deployment practice

• Encourage complexity and bloat
• Focuses on programmer productivity – maximizing reuse
• Layer upon layers of abstraction, libraries, frameworks & API – inefficient execution
• Libraries, frameworks and API are designed to be general purpose – large percentage of dead code

Consequences

• Unnecessary complexity -- difficult to formally verified
• Wasting CPU cycle and runtime memory – unnecessary slow down
• Provide fertile ground for return oriented programming (ROP)

Rube Goldberg approved software application is analyzed in:
http://lcsd05.cs.tamu.edu/papers/sevitsky_et_al.pdf

Sevitsky et. al. (IBM TJ Watson Research Center) on framework based applications:
• In every application we looked at, an enormous amount of activity was executed to accomplish simple tasks.
• For example, a stock brokerage benchmark executes 268 method calls and creates 70 new objects just to move a single date field from SOAP to Java.
Software development & deployment strategy for enhancing efficiency

- Automated redundant layers collapsing and bloat removal during compilation, installation, dynamic runtime, via compilation, binary rewriting, just-in-time compilation or runtime binary rewriting, for smaller, simpler, leaner, and more efficient executable.

- Software deployment strategies, which reduce the use of dynamic link library (DLL); selectively compiles in only the required part of the library statically (or derive specialized library), by just-in-time compilation or binary rewriting. **

- Software development environment and methodology with support for ease of layer collapsing, while preserving the benefit for software reuse

- Better software maintenance control & tracking methods for reducing potential for redundant logics, and software bloat.

** The assumption taken in this research is that the required storage size for code is relatively small as compare to data, and current storage is relatively cheap, and hence there is not much of a reason for optimizing for code-storage by using DLL (storage was expensive in the past). This assumption applies to various but not all computing environments.
Post Compile time specialization beyond single executable

- Dynamic & Interpreted Code
- Protocol ‘subset’-ing/specialization
- Framework specialization
- Balancing ~ knowing when to specialize and when not to.
- Strategy ~ whole, partial, none
- What else … ?

Improving Efficiency w/o Being Disruptive to Development Methodology
Software verification for reducing vulnerability

- Security & vulnerability specific verification techniques
- Identify and isolate potential system vulnerability

Potential approach

- Static (including formal methods)
- Dynamic
- Hybrid

Challenges with the above approaches

- Accuracy & reliability (false positive & false negative)
- Large size & complexity preventing exhaustive evaluation/verification (static, dynamic)
- Continuous evaluation?, → keep on fixing, when to stop, limits applicability (dynamic)
- Hybrid inherits the strength & weaknesses of both, only less severe in limitations/weaknesses
Data Exfiltration Awareness

What

- Methods and techniques for detecting improper/illegal data transfer via regular and covert channels
  - Methods for retrieving data/document from adversary’s covert channels → implies recognizing covert channels
  - Methods for identifying document/part of document in transit
  - Methods for identifying origin of document fragment

Difficult topics

- Identifying and analyzing covert channels
- Associating fragment(s) of a document to original document
- Blind preventive techniques
  → Potential approach: reduce potential for covert channel, such as always reformat (transform) images to avoid steganography, etc…
F2: Hardening Interaction Among Processes & Users

Thrusts:

T1: Trust management, model and calculus
T2: Confidentiality in collaboration
T3: Mobility and collaboration
T4: Automatic Crypto Function Generation
Hardening Interaction Among Processes & Users

Users, process and data in distributed environment need:

- Robust authentication of users & credentials
- Reliable and sound trust propagation methods
  - Providing confidentiality
  - Preventing security breach

Conventional: \( \text{Access} \approx f(\text{user, data}) \)

For security: \( \text{Access} \approx f(\text{process, user, data}) \)
or even \( \text{Access} \approx f(\text{state, process, user, data}) \)

- In cloud application, need the ability to verify that the returned result is indeed the outcome of the requested process on the prescribed data
- In field application, also need to deal with disconnection
Crypto Factory: Toward automatic crypto-function generation

- Unexplored membership of set of secure* crypto functions (*as secure as current crypto functions)
- Currently → manual crypto function exploration
- The membership can be explored with the help of computer, if:
  - Security proofing can be automated (hard problem)
  - Candidate for secure crypto function can be computer generated (not so hard problem)
- Can a more secure crypto function be built from composite of secure crypto functions without relying on ever increasing key length?
  (The strictest answer → NO, but world isn’t as strict)

Typical structure of crypto function

Generalized diagram of Crypto Factory
**F3: Hardening The Network**

**Thrusts:**

- **T1:** Toward dynamic re-configuration (mitigating DDOS)
- **T2:** Automated dynamic network device configuration & policy management
- **T3:** Enhancing malicious traffic detection & Attribution
- **T4:** Collaborative framework for cross-domain network management
- **T5:** More Secure Communication & Routing Architectures & Protocols
- **T6:** Artificial diversity at network level
Dealing with legitimate traffic DDOS (brute force attack)

- Brute force attack does not rely on (fixable) vulnerability
- No easy way to deal with (similar to radar jamming), short off modifying internet protocols (such as in T5).
- Problem w/ changing internet protocols, every parties in the route need to change.
Detection, Redirection and Mitigation
(T1,T2,T3,T4)

- Detection of the attack
- Filtering, redirection & dropping
- Trace-back to source of attack
- Pushing filtering & redirection progressively closer to the source of attack reduces wasted bandwidth
- Requires cross-administrative-domain coordination
- Need for efficient (preferably automated) way for cross-domain collaborative network management (state of the art: manual over phone line)
  → Issues with policy for confidentiality and privacy
Hardening The Network

Out-Run adversary \((T1,T2,T6)\)

- Quick network configuration and topology reconfiguration cycle which is quicker than the cycle needed for the adversary to recognize the change and re-probe and understand the new configuration & topology

Questions to be addressed include:

- What re-configuration matters?, and what it takes?, How much disruption to (our) benign traffic?
  - IP Address, and TCP/UDP ports → prevents adversarial LAN mapping → dynamic NAT reconfiguration. In LAN already researched → ODNI has experimentation??
  - DNS mapping → prevents brute-force DDOS, → how to minimize disruption to our traffic
  - LAN topology, WAN topology??
The End