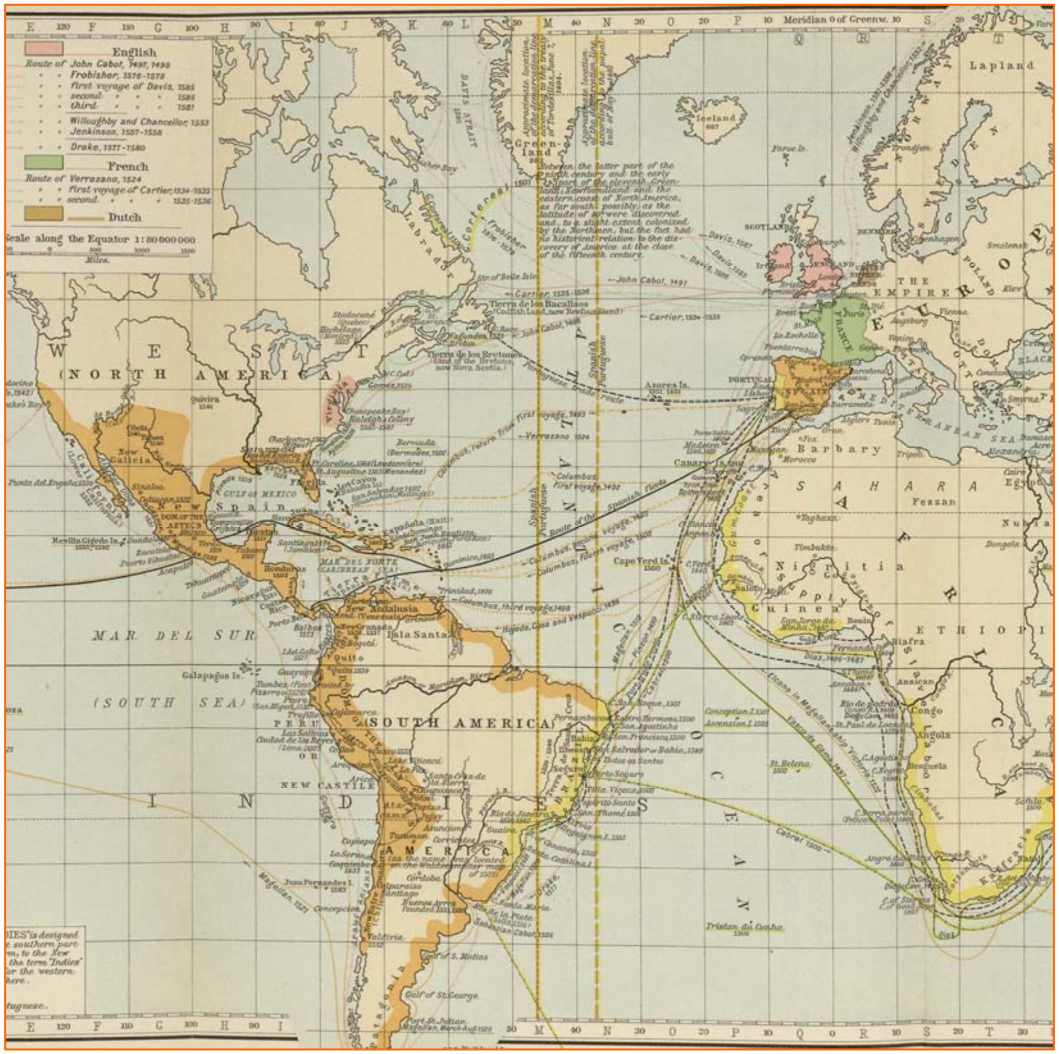
A Map for Security Science

Fred B. Schneider*

Department of Computer Science
Cornell University
Ithaca, New York 14853
U.S.A.

*Funded by AFOSR, NICECAP, NSF (TRUST STC), and Microsoft.

Maps = Features + Relations



- **Features**
  - Land mass
  - Route
- **Relationships**
  - Distance
  - Direction

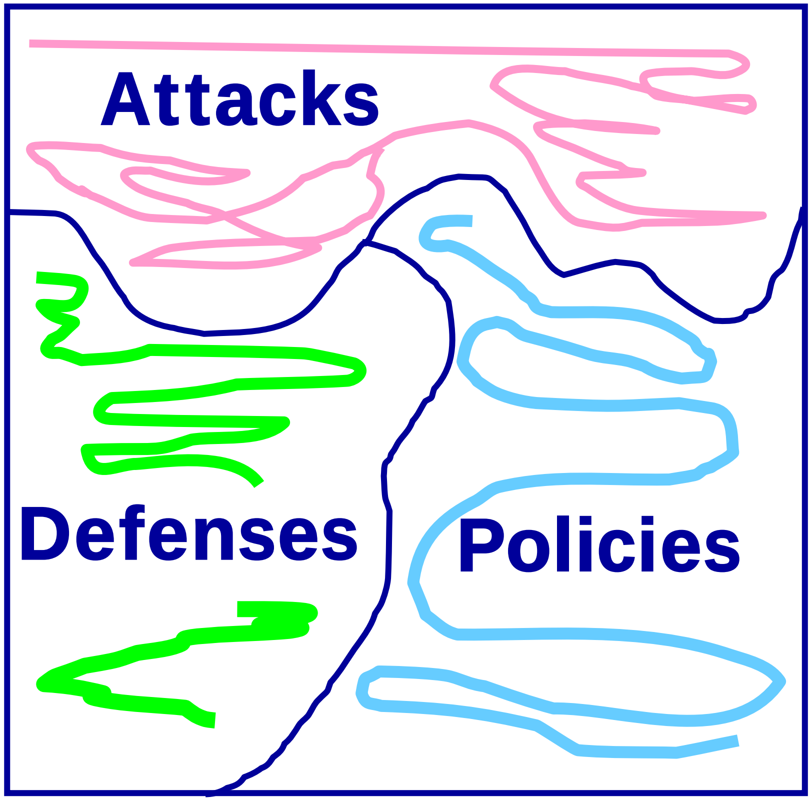
Map of Security (circa 2005)

Features:

- Port Scan
- Bugburg
- Geekland
- Bufferville
- Malwaria
- Root kit pass
- Sploit Market
- Valley of the Worms
- Sea Plus Plus
- Sea Sharp
- …

Reproduced courtesy Fortify Software Inc

Map of Security (circa 2015?)



Features:

- Classes of attacks
- Classes of policies
- Classes of defenses

Relationships:

"Defense class D enforces policy class P despite attacks from class A."

Outline

Give examples to demonstrate:

- map features: -- policy, defense, attack classes
- relationships between these “features”

Discuss scope for term “security science”.

“If everybody is special, then nobody is.”

-Mr. Incredible

“Good” work in security might not be “security science”.

Give example and non-obvious open questions in “security science.”

Oldspeak:
Security Features: Attacks

Attack: Means by which policy is subverted. A threat exploits a vulnerability.

- Attacks *du jour* :
  - E.g. buffer overflow, format string, x-site scripting, …
- Threat models have been articulated:
  - E.g. insider, nation-state, hacker, ….
  - E.g. 10 GByte + 50 Mflops, …
  - *Threat model → Attacks?*

Oldspeak:

Security Features: Policies

Policy: What the system should do; what the system should not do:

- **Confidentiality**: Who is allowed to learn what?
- **Integrity**: What changes are allowed by system.
  … includes resource utilization, input/output to environment.
- **Availability**: When must service be rendered.

Usual notions of "program correctness" are a special case.

Oldspeak:

Security Features: Defenses

Defense Mechanism: Ensure that policies hold. Example general classes include:

- Monitoring (reference monitor, firewall, …)
- Isolation (virtual machines, processes, sfi, …)
- Obfuscation (cryptography, automated diversity)

Oldspeak:

Security Features: Relationships

Attack $\leftrightarrow$ Defense

Secure System Pragmatics:

- Attacks exploit vulnerabilities.
  - Vulnerabilities are unavoidable.
- Assumptions are potential vulnerabilities.
  - Assumptions are unavoidable.

… All non-trivial systems can be attacked.

- ? Can a threat of concern launch a successful attack ?

Classes of Attacks

- Operational description:
  - “Overflow an array to clobber the return ptr…”
- Semantic characterization:
  - A program…
    - RealWorld = System || Attack (Dolev-Yao, Mitchell)
  - An input…
    - Causes deviation from a specification.
    - Causes different outputs in diverse variants.

Classes of Policies

System **behavior** t: an infinite trace

$t = s_0\ s_1\ s_2\ s_3\ \ldots\ s_i\ \ldots$

System **property** P: set of traces

$P = \{\ t \mid \text{pred}(t)\ \}$

System S: set S of traces (its behaviors).

System S **satisfies** property P: $S \subseteq P$

Safety and Liveness [Lamport 77]

Safety: Some “bad thing” doesn’t happen.

Liveness: Some “good thing” does happen.

Safety and Liveness [Alpern+Schneider 85,87]

Safety: Some "bad thing" doesn't happen.

- Proscribes traces that contain some irremediable prefix.

Liveness: Some "good thing" does happen.

- Prescribes that prefixes are not irremediable.

Thm: Every property is the conjunction of a safety property and a liveness property.

Thm: Safety properties proved by invariance.

Thm: Liveness properties proved by well-foundedness.

Execution Monitoring (EM) [Schneider 2000]

Execution monitor:

- Gets control on every policy-relevant event
- Blocks execution if allowing event would violate policy
- Integrity of EM protected from subversion.

Thm: EM only enforces safety properties.

Examples of EM-enforceable policies:

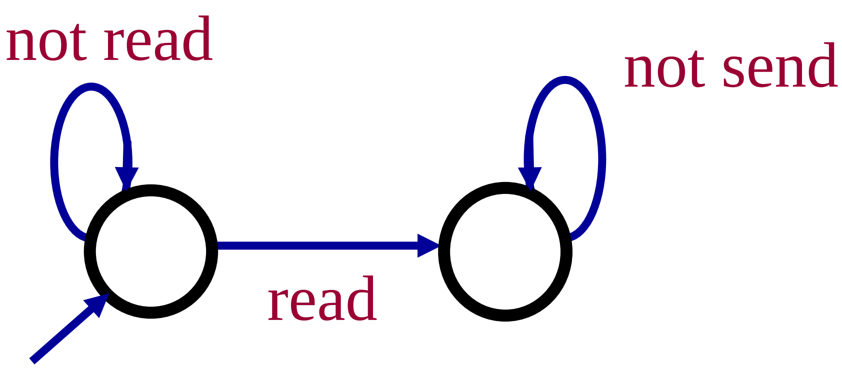
- Only Alice can read file F.
- Don't send msg after reading file F.
- Requests processing is FIFO wrt arrival.

Examples of non EM-enforceable policies:

- Every request is serviced
- Value of x is not correlated with value of y.
- Avg execution time is 3 sec.

New EM Approaches

Every safety property corresponds to an automaton.

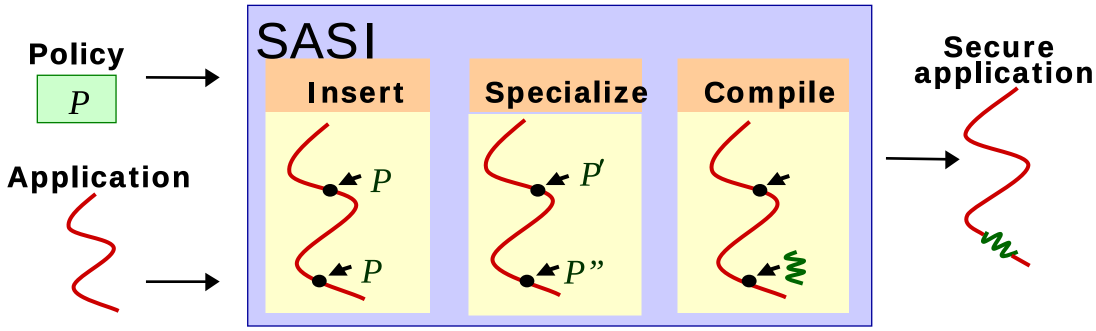


$\square(\text{read} \Rightarrow \square\neg\text{send})$

Inlined Reference Monitor (IRM)

New approach to enforcing EM policies:

1. Automaton → Pgm code (case statement)
2. Inline automaton into target program.



Relocates trust from pgm to reference monitor.

Proof Carrying Code

New approach to enforcing EM policies:

- Code producer:
  - Automaton A + Pgm S → Proof S **sat** A
- Code consumer:
  - If A suffices for required security then check:

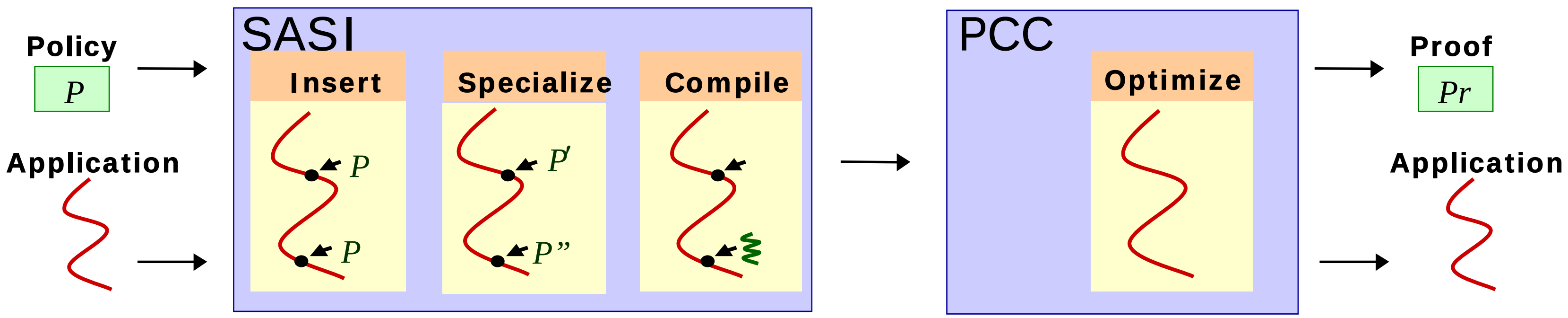
    Proof S **sat** A

(Proof checking is easier than proof construction.)

Relocates trust from pgm and prover to proof checker.
Proofs more expressive than EM.

Proof Carrying Code

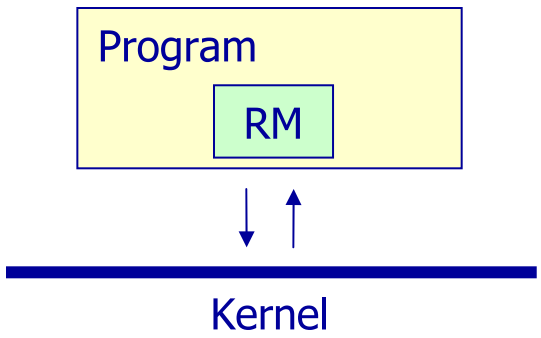
PCC and IRM…



Virtues of IRM

When mechanism inserted into the application ...

- Allows policies in terms of application abstractions.
- Pay only for what you need.
- Enforcement without context switches into kernel.
- Isolates state of enforcement mechanism.



Security ≠ Safety Properties

Non-correlation: Value of L reveals nothing about value of H.

Non-interference: Deleting cmds from H-users cannot be detected by cmd exec by L-users.
[Goguen-Meseguer 82]

Properties, safety, liveness not expressive enough!

EM not powerful enough.

Hyper-Properties [Clarkson+Schneider 08]

Hyper-property: set of properties
= set of sets of traces

System S **satisfies** hyper-property HP: $S \in HP$

Hyper-property [P]: $\{P' \mid P' \subseteq P\}$

Note:

- ($P \in HP$ **and** $P' \subseteq P$) $\Rightarrow$ HP not required.
- Non-interference is a HP.
- Non-correlation is a HP.

Hyper-Safety Properties

Hyper-safety HS: "Bad thing" is property M comprising finite number of finite traces.

- Proscribes tracing containing irremediable observations.

Thm: For safety property S, [S] is hyper-safety.

Thm: All hyper-safety are refinement closed.

Note:

- Non-interference is a HS.
- Non-correlation is a HS.

Hyper-Safety Applications

2SP: Safety property on program S composed with itself (with variables renamed). [Terauchi+Aiken 05]

S; S′

2SP transforms information flow into a safety property!

K-safety: Safety property on program

S^K: S || S′ || … || S″

K-safety is HS.

Thm: Any K-safety property of S is equivalent to a safety property on S^K.

Hyper-Liveness Properties

Hyper-liveness HL: Any finite set M of finite traces has an augmentation that is in HL.

Prescribes: observations are not irremediable.

- Examples: possibility, statistical performance, etc.

Thm: Every HP is the conjunction of HS and HL.

Hyper Recap

Safety Properties ↔ EM enforceable:
→ New enforcement (IRM)

Properties not expressive enough:
→ Hyper-properties (-safety, -liveness)
→ K-safety (reduces proving HS to a prop).

Q: Verification for HS and HL?
Q: Refinement for HS and HL?
Q: Enforcement for HS and HL?

Obfuscation: Goals and Options

Semantics-preserving random program rewriting…

Goals: Attacker does not know:

- address of specific instruction subsequences.
- address or representation scheme for variables.
- name or service entry point for any system service.

Options:

- Obfuscate source (arglist, stack layout, …).
- Obfuscate object or binary (syscall meanings, basic block and variable positions, relative offsets, …).
- All of the above.

Obfuscation Landscape

[Pucella+Schneider 06]

Given program S, obfuscator computes **morphs**:
T(S, K1), T(S, K2), … T(S, Kn)

- Attacker knows:
  - Obfuscator T
  - Input program S
- Attacker does not know:
  - Random keys K1, K2, … Kn

    … Knowledge of the Ki would enable attackers to automate attacks!

Will an attack succeed against a morph?

- Seg fault likely if attack doesn't succeed.

  integrity compromise → availability compromise.

Successful Attacks on Morphs

All morphs implement the same interface.

- **Interface attacks**. Obfuscation cannot blunt attacks that exploit the semantics of that (flawed) interface.
- **Implementation attacks**. Obfuscation can blunt attacks that exploit implementation details.

Def. implementation attack: An input for which all morphs (in some given set) don't **all** produce the same output.

Effectiveness of Obfuscation

Ultimate Goal: Determine the probability that an attack will succeed against a morph?

Modest goal: Understand how effective obfuscation is as compared with other defenses?

- Obvious candidate: Type checking

Type Checking as a Defense

Type checking: Process to establish that all executions satisfy certain properties.

- Static: Checks made prior to exec.
  - Requires a decision procedure
- Dynamic: Checks made as exec proceeds.
  - Requires adding checks. Exec aborted if violated.

Probabilistic dynamic type checking: Some checks are skipped on a random basis.

Obfuscation versus Type Checking

Thesis: Obfuscation and probabilistic dynamic type systems can "defend against" the same attacks.

From "thesis" → "theorem" requires fixing:

- a language
- a type system
- a set of attacks

Obfuscation approximates typing

Theorem: Type error signaled if and only if ressistible attack relative to T() and keys K1, K2, …, Kn for type systems:

- "pointer de-ref sanity" types.
  - Implied by usual notion of "strong typing".
  - Is a stronger type system than necessary. E.g.

    ```
    if x[i] = x[i] then skip
    ```

    is not type-safe but is not affected by T.
- "tainting" type system (=info flow)
  - Better approximation than "pointer de-ref sanity" types.
  - Low integrity value: can vary from morph to morph

Type Systems / Obfuscator Bad News

Theorem: There is no computable type system that signals a type error iff attacks relative to address obfuscation and some finite set of keys K1, K2, ..., Kn.

Pros and Cons of Obfuscation

- Type systems:
  - Prevent attacks (always---not just probably)
  - If static, they add no run-time cost
  - Not always part of the language.
- Obfuscation
  - Works on legacy code.
  - Doesn’t always defend.

Recap: Features + Relationships

- Defined: Characterization of policy: hyper-policies
  - Linked to semantics + orthogonal decomp
- Relationship: Class of defense (EM) and class of policies (safety):
  - Provides account of IRM and PCC.
- Relationship: Class of defense (obfusc) and class of defense (type systems).
  - Uses “reduction proof” and class of attacks

A Science?

- **Science**, meaning focus on **process**:
  - Hypothesis + experiments → validation
- **Science**, meaning focus on **results**:
  - abstractions and models, obtained by
    - invention
    - measurement + insight
  - connections + relationships, packaged as
    - theorems, not artifacts
- **Engineering**, meaning focus on artifacts:
  - discovers missing or invalid assumptions
    - Proof of concept; measurement
  - discovers what are the real problems

A Security Science?

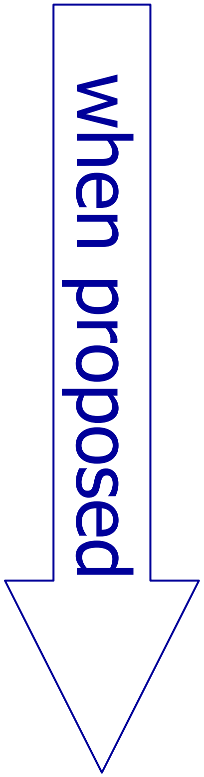
Address questions that transcend systems, attacks, defenses:

- Is “code safety” universal for enforcement?
- Can sufficiently introspective defenses always be subverted?
- …

A Security Science?

SSR* seeking relationships:

- Absolute security vs Risk Management
- Prevention vs Accountability
  - Role of Authentication + Authorization
- Perfection vs Diversity
  - Specification of behavior –vs-
  - Independence wrt attacks
- Enforcement vs Relocation of Trust



*SSR: Single Security Researcher