Unified Configuration Modeling Infrastructure

Denis Gopan,

Vlad Folts, Rebecca Swords, Thomas Wahl, Lucja Kot

May 19, 2022

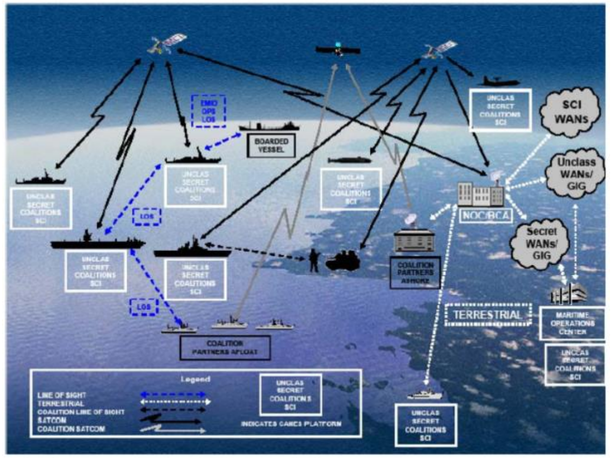
Modern Systems and Their Configuration

- Built from COTS / open-source components
  - General purpose components: support diverse usage scenarios
  - Must be properly configured to interoperate and provide desired functionality

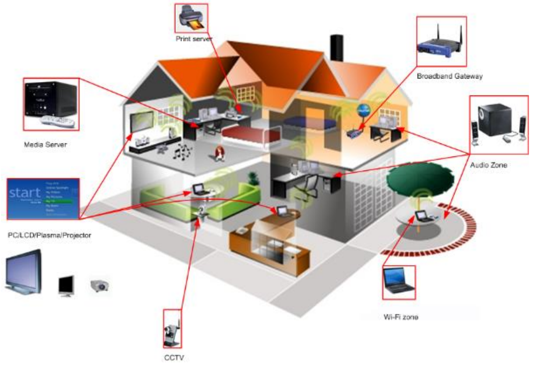
DoD Systems

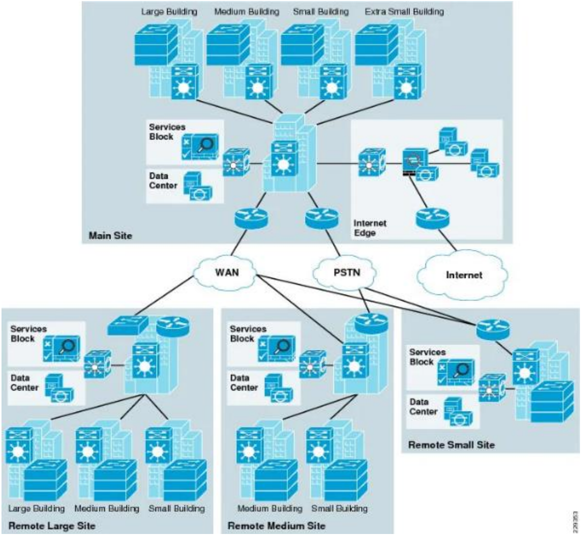
CONSOLIDATED AFLOAT NETWORKS
CANES
AND ENTERPRISE SERVICES

PEO C4I



Home Networks



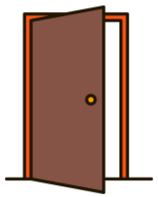


Enterprise Systems

GRAMMATECH

The Importance of Proper Configuration

- Misconfiguration constitutes a significant security threat
  - Easier to exploit than conventional vulnerabilities
  - Extend attack surface



vs.

Misconfiguration

Conventional Vulnerability

- Verizon's Data Breach Investigations Report (DBIR) Report, 2021
  - Overall, around 8% of breaches are due to misconfiguration
  - http://verizon.com/dbir/

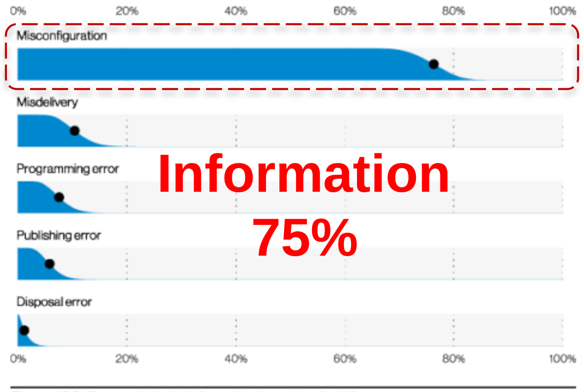


Figure 106. Error varieties in Information breaches (n=111)

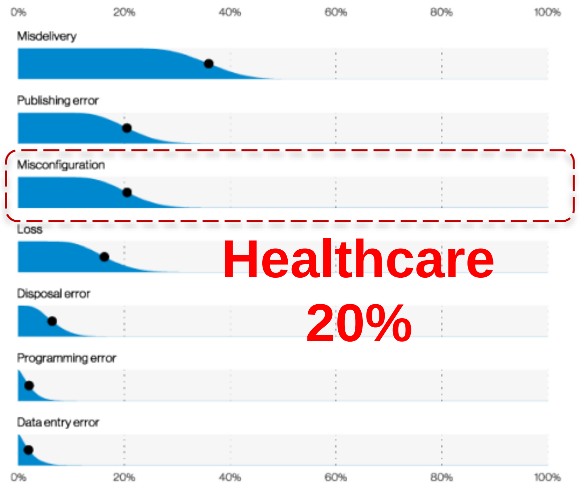


Figure 105. Error varieties in Healthcare breaches (n=70)

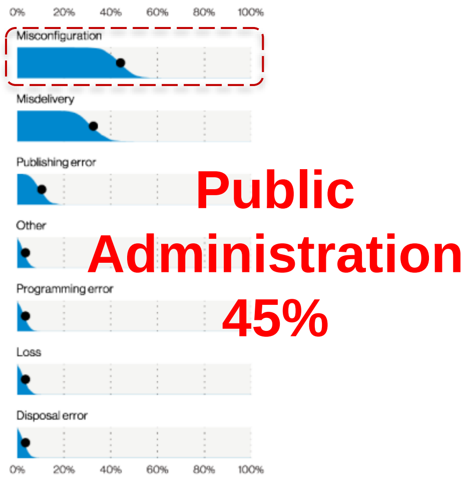
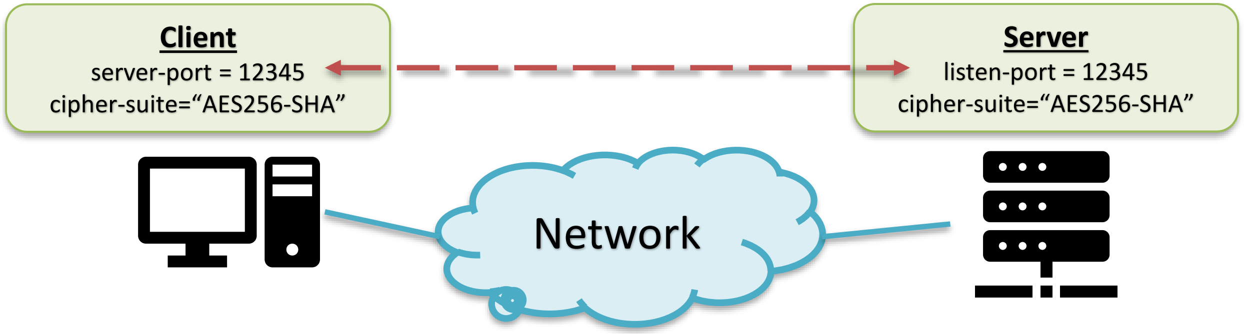


Figure 118. Top Error varieties in Public Administration breaches (n=86)

GRAMMATECH

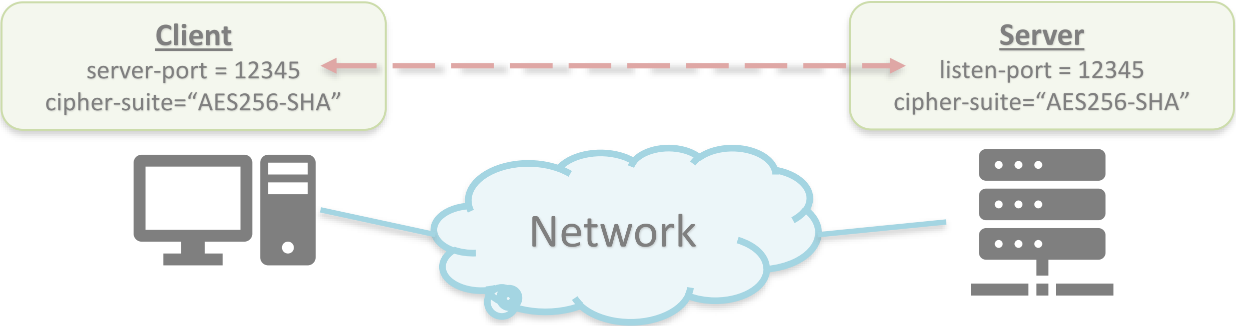
Why is Configuration Tricky?

- **Global relationships among configuration parameters**
  - Configuration must agree for the communication end-points
  - Local policies (e.g., STIGs), while useful, often fail system-wide constraints

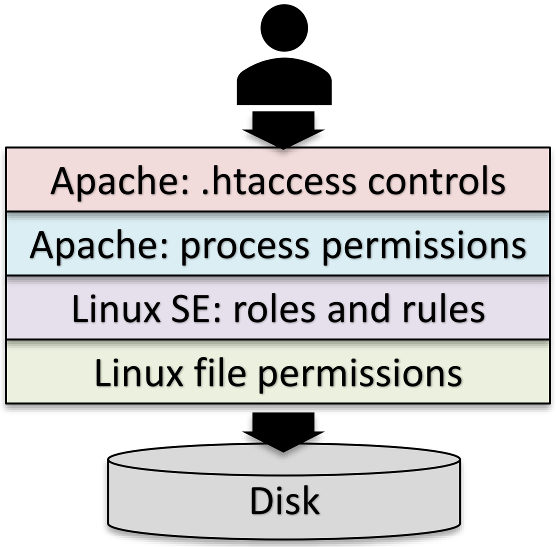


Why is Configuration Tricky?

- **Global relationships among configuration parameters**
  - Configuration must agree for the communication end-points
  - Local policies (e.g., STIGs), while useful, often fail system-wide constraints

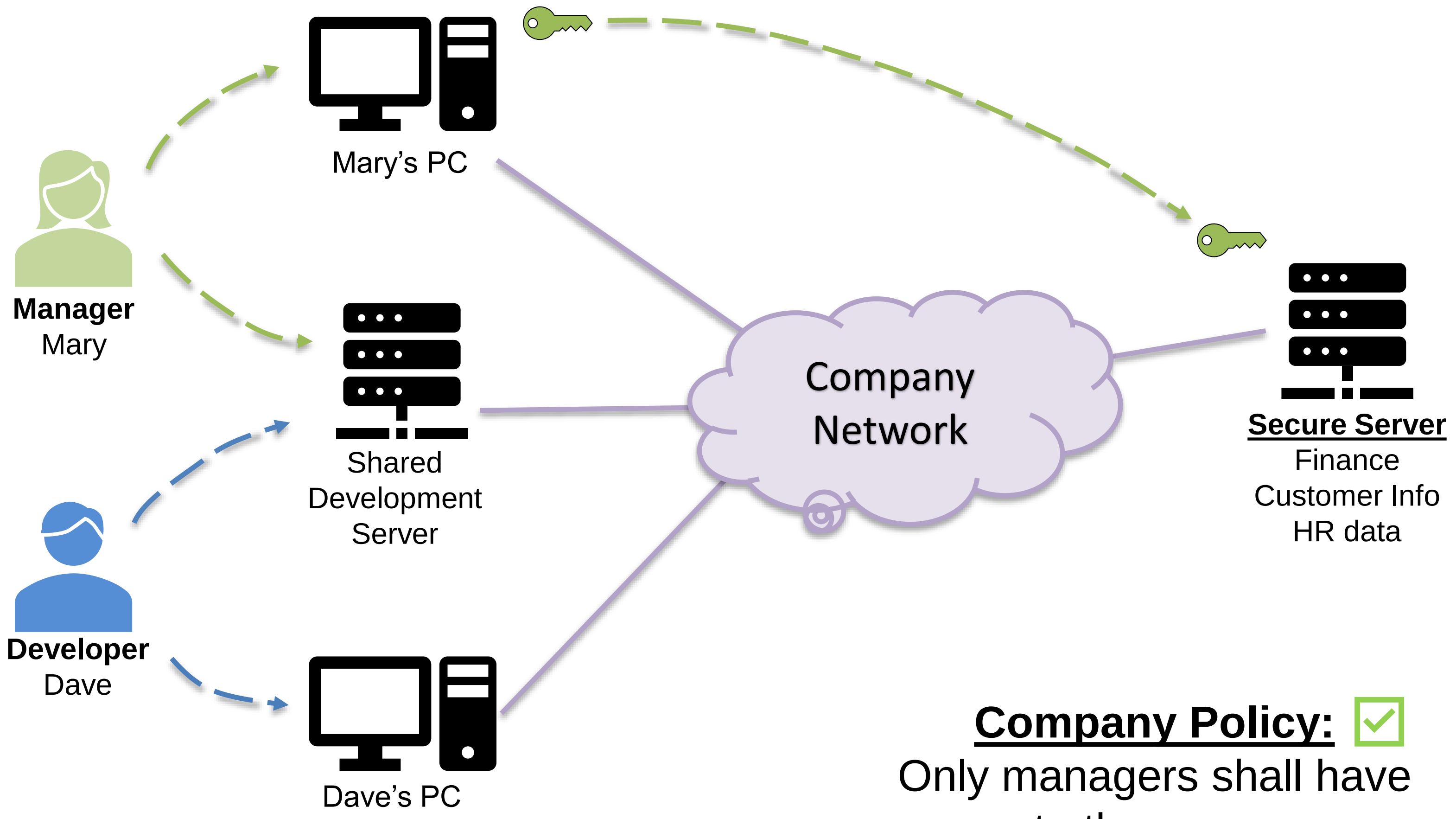


- **Configuration stacking for multi-layer systems**
  - Each layer is configured independently
  - Layer configurations are combined
  - *Overall end-to-end effect is hard to perceive*



GRAMMATECH

Misconfiguration: Security Breach Scenario



Human Actors:

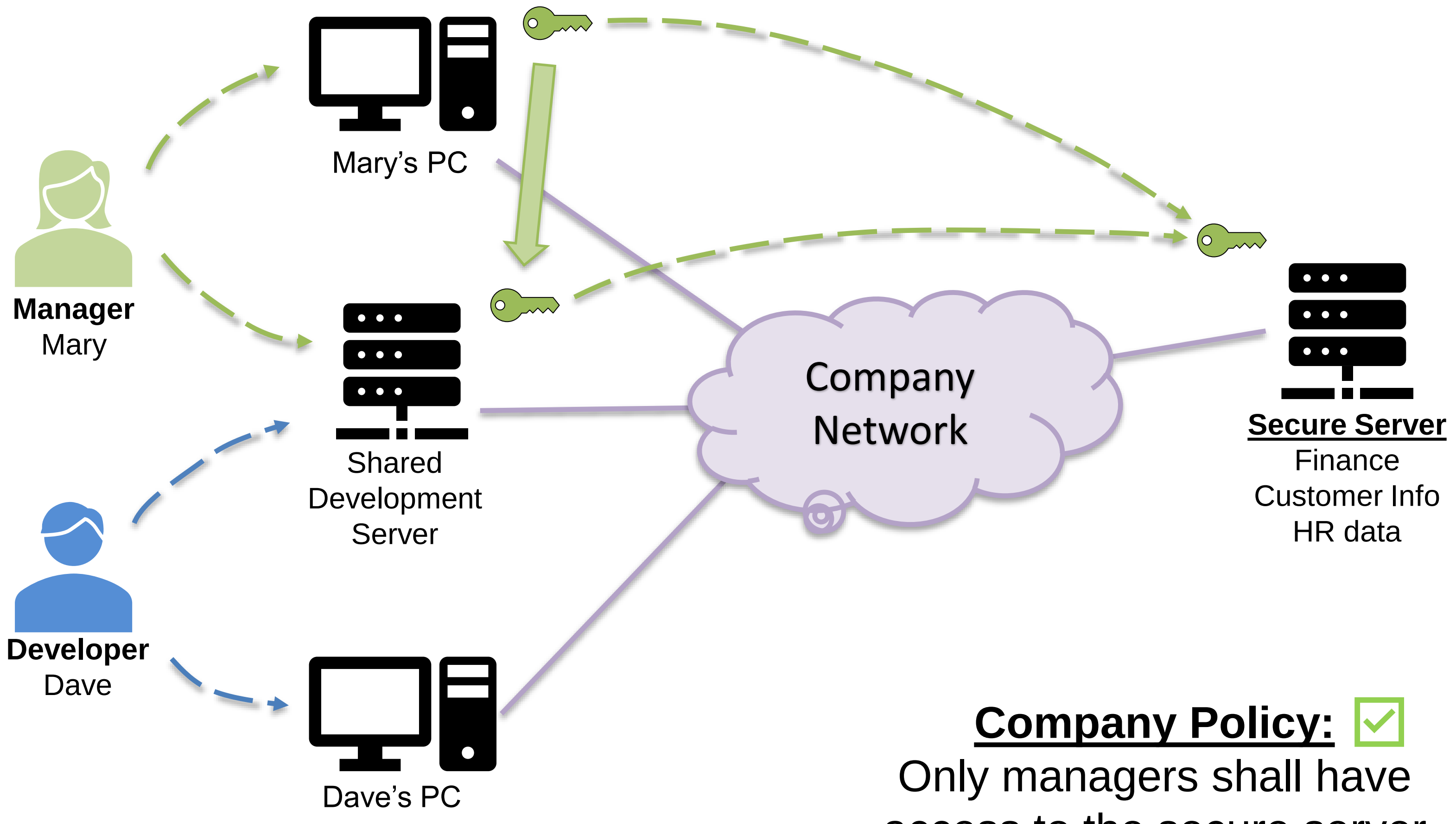
- Mary: manager
- Dave: developer

System Access:

- Both Mary and Dave:
  - Respective PCs
  - Development server
- Mary only:
  - Secure server via PKI

Company Policy: ☑
Only managers shall have access to the secure server

Misconfiguration: Security Breach Scenario



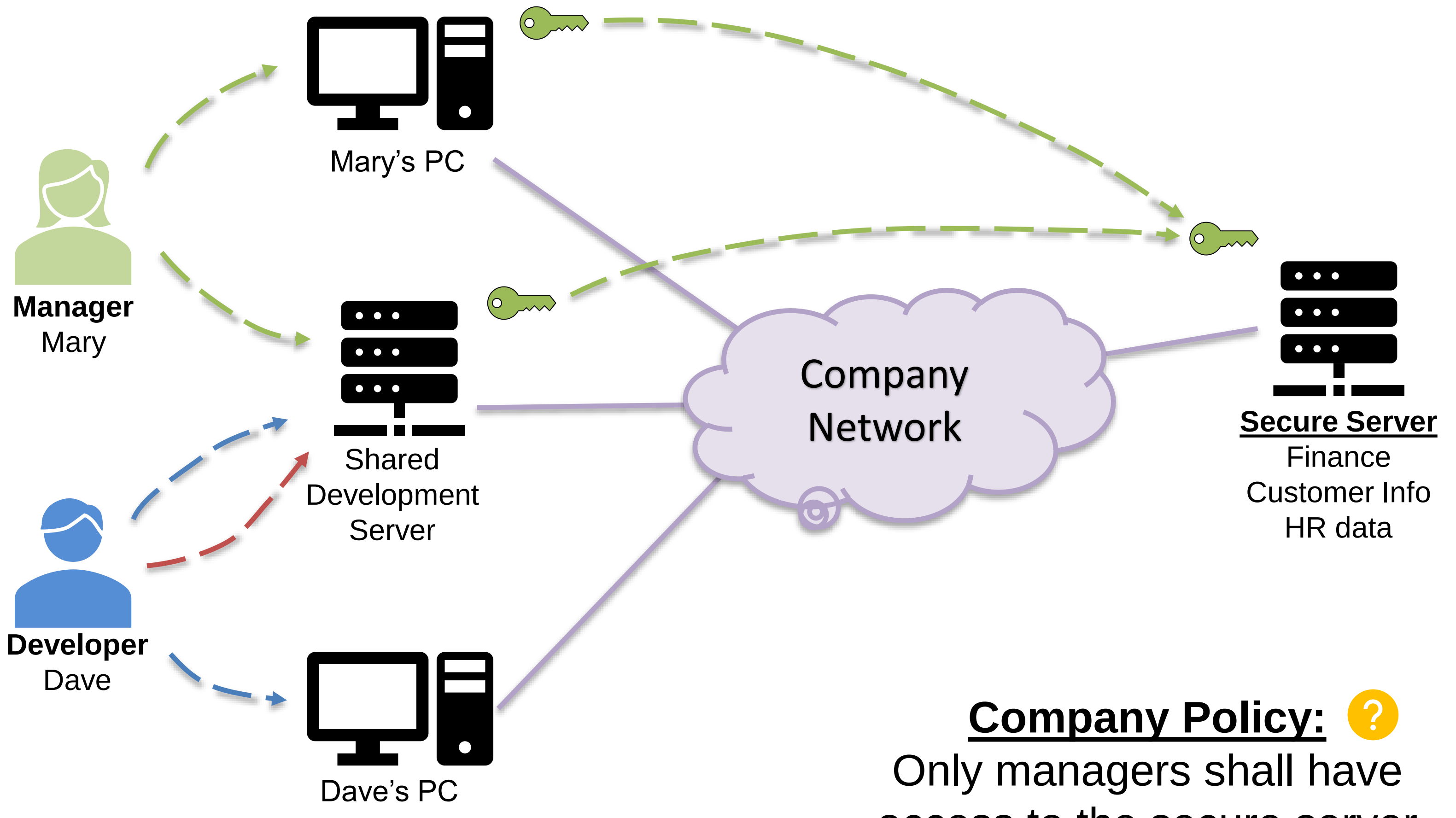
Mary:

- Wants to access the secure server from the development server
- Copies her private key (or sets up a new one)

Company Policy: ☑
Only managers shall have access to the secure server

GRAMMATECH

Misconfiguration: Security Breach Scenario



Company Policy: ?
Only managers shall have access to the secure server

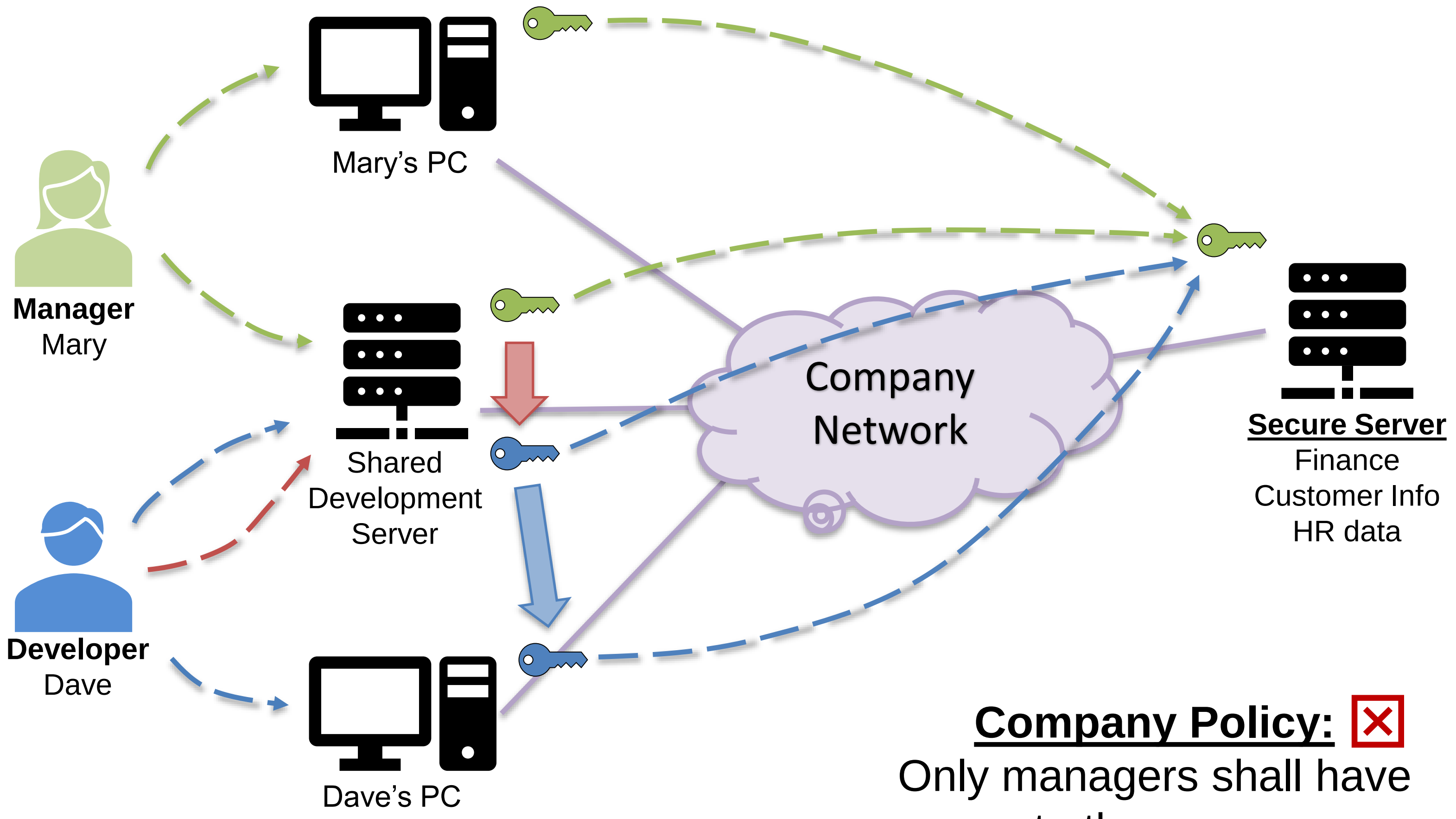
Mary:

- Wants to access the secure server from the development server
- Copies her private key (or sets up a new one)

Dave:

- Wants extra privileges (to install new software)
- Admin gives him root-level access (using naïve "sudo" configuration)

Misconfiguration: Security Breach Scenario

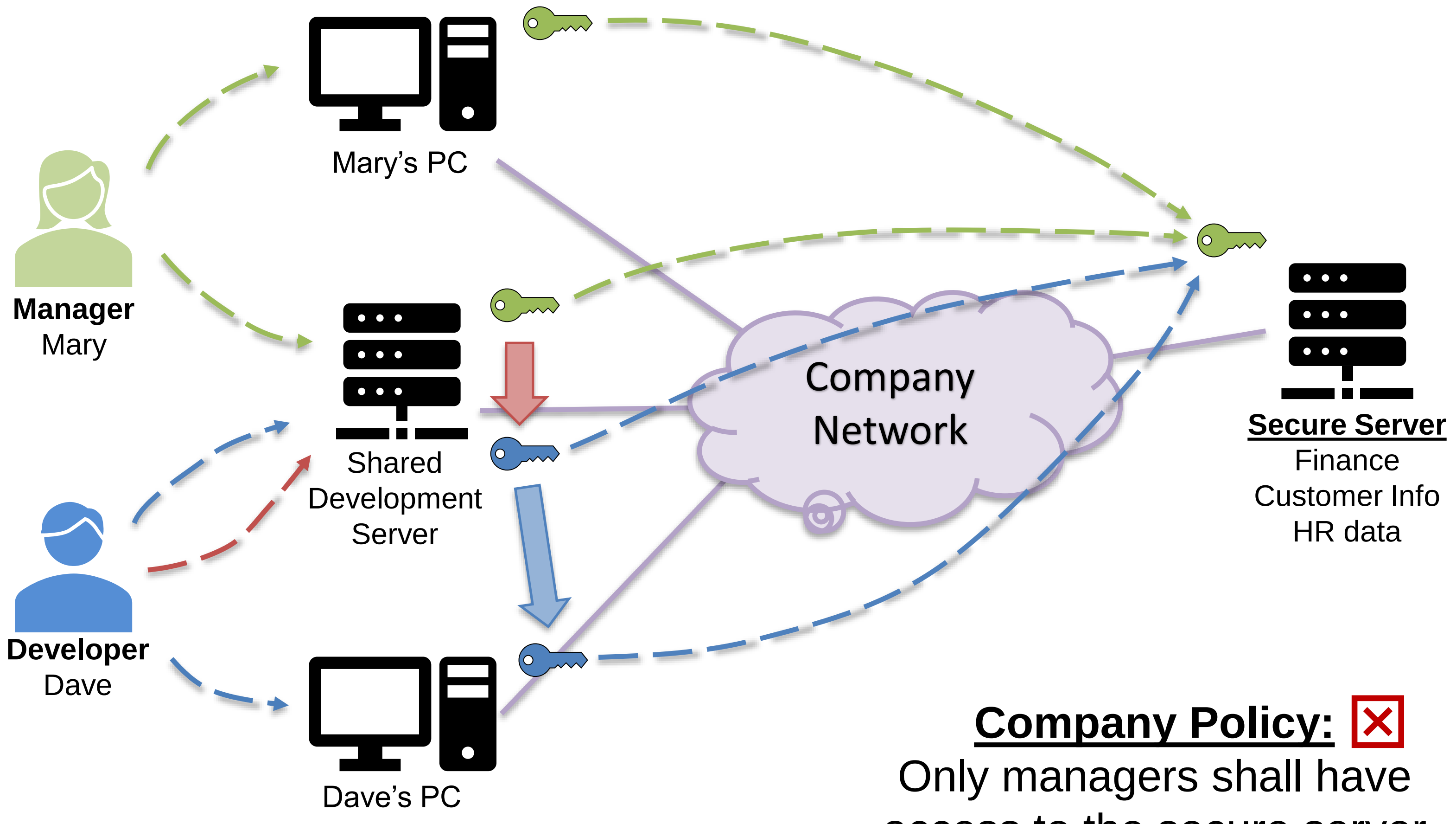


Company Policy: ☒
Only managers shall have access to the secure server

Security Breach:

- Dave copies (steals) Mary's private key using his root-level access privileges
- Gets access to the secure server by impersonating Mary
- ***Policy is violated!***

Misconfiguration: Security Breach Scenario



Company Policy: ☒
Only managers shall have access to the secure server

Observations:

- Local configuration changes may have global effect that is hard to predict / anticipate
- Fixes (e.g., implementing a more fine-grained sudo policy for Dave) further increase configuration complexity

Formal Reasoning to the Rescue

- System configuration modeling
  - Formal, self-contained description of the system
  - Logically links system behavior to the values of its configuration parameters
  - **Abstract:** reduces the complexity of the actual system
  - **Functional:** provides an effective way to query system properties
  - **Initial focus:** *access-control properties*

Can a system user with certain level of physical access and certain knowledge perform a certain operation on certain resource?

Formal Reasoning to the Rescue

- System configuration modeling
  - Formal, self-contained description of the system
  - Logically links system behavior to the values of its configuration parameters
  - **Abstract:** reduces the complexity of the actual system
  - **Functional:** provides an effective way to query system properties
  - **Initial focus:** *access-control properties*

Dave's PC

Dave's Credentials

Can a system user with certain level of physical access and certain knowledge perform a certain operation on certain resource?

Log into

Secure Server

Configuration Modeling: Practical Applications

- Automated configuration synthesis:
  - Find config that enables system functionality while minimizing attack surface
  - Formal modeling enables setting this up as ***an optimization problem***
  - Initial motivation for the research
- Symbolic configuration analysis:
  - Check non-trivial security and functionality properties
  - Assess how local configuration changes affect system-wide behavior
  - ***Example:*** validate that refined "sudo" configuration
    - Allows Dave to perform necessary development server maintenance
    - Respects relevant company's security policies

Configuration Modeling: Practical Applications

- Penetration testing and red teaming:
  - Can an outsider with minimal knowledge gain access to system's sensitive data?
  - If so, how? Can we get a sequence of operations to execute?
- Internal threat minimization:
  - Find system users with privileges that are not required for system functionality
  - Ensure access privileges are properly revoked
- Forensic analysis:
  - The system is breached!
  - What is the breach perimeter? What system data can we still trust?

Configuration Modeling: Overview

- Supports essential programming-language features
  - *Modular:* define and hierarchically compose modules
  - *High-level datatypes:* enums, records, collections, variants
  - *Strict static typing:* catch and debug errors

- *Current status:* language is bytecode / IR like
  - Aimed for machine consumption, rather than for human use
  - Working on a more human-friendly front-end language

Modeling Language

Declarative; supports modularity, high-level data types

GRAMMATECH

Configuration Modeling: Overview

Primitives:
- Network interfaces
- Linux users, groups
- Linux files, directories, symlinks
- Services (e.g., sshd, httpd)
- Public / private keys
- SSL certificates
- ….

Concepts:
- Network access
- Service access
- Local and remote host access
- Local and remote file access
- ….

Predicate Library

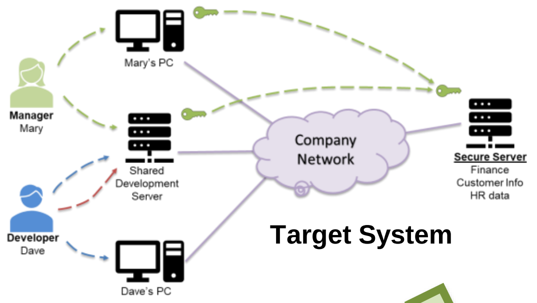
Logical definitions that describe generic system-building primitives and concepts

Definitions express semantics in terms of configuration parameters (*free model variables*)

Modeling Language

Declarative; supports modularity, high-level data types

Configuration Modeling: Overview



Target System

Predicate Library

Logical definitions that describe generic system-building primitives and concepts

Behavior

Instantiates predicates to capture system topology and device setup

Configuration

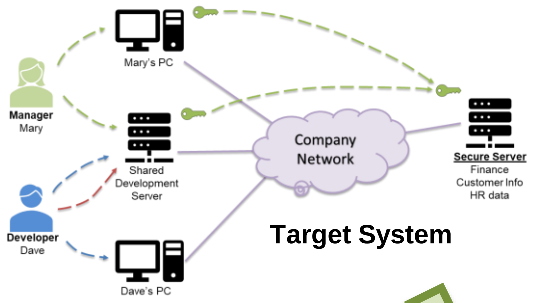
Assigns values to configuration parameters (*model variables*)

Modeling Language

Declarative; supports modularity, high-level data types

GRAMMATECH

Configuration Modeling: Overview



Target System

Predicate Library

Logical definitions that describe generic system-building primitives and concepts

Behavior

Instantiates predicates to capture system topology and device setup

Configuration

Assigns values to configuration parameters (*model variables*)

Modeling Language

Declarative; supports modularity, high-level data types

Querying System Properties

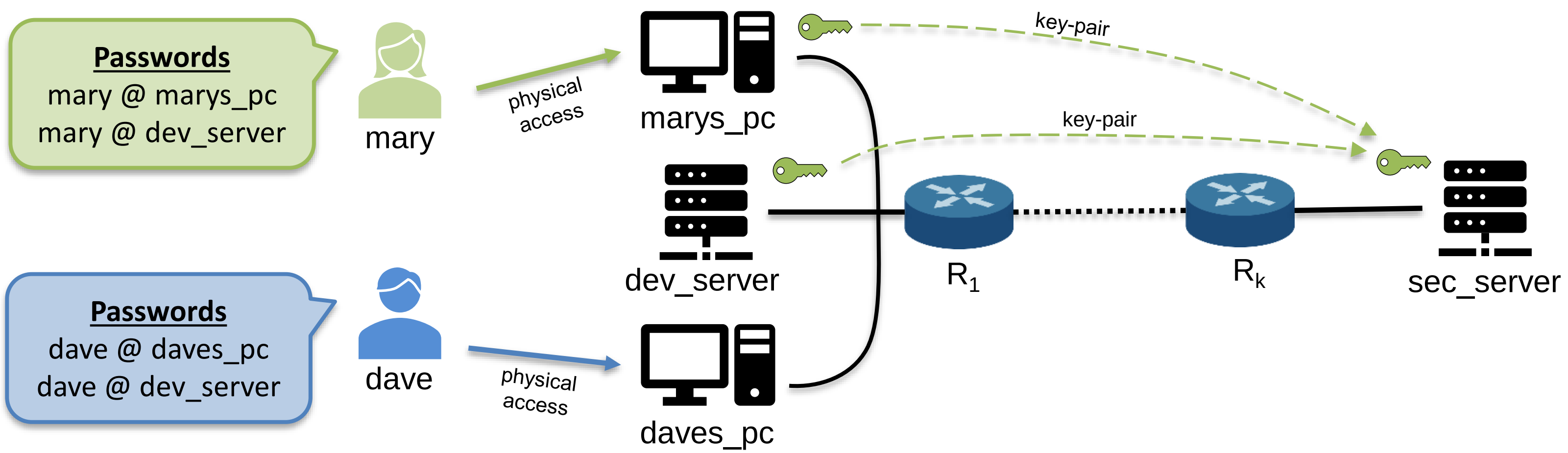
Compile into Prolog program and use off-the-shelf interpreter (better alternatives are possible)

Deductive Database

Facts and Rules

GRAMMATECH

Configuration Modeling: Deep(er) Dive

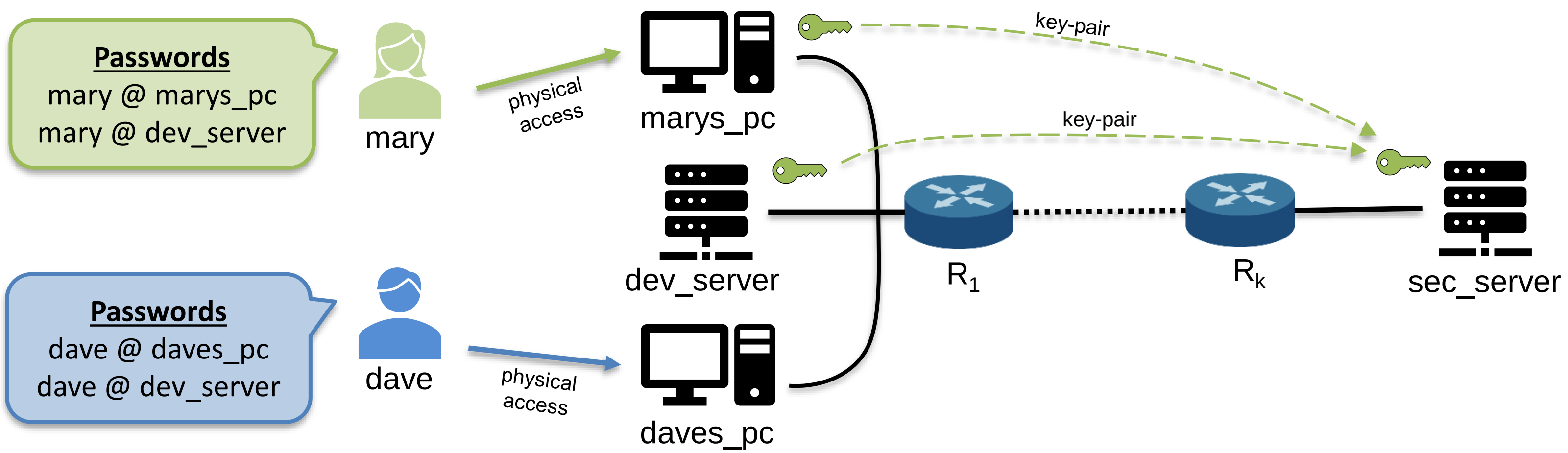



```
actorAccess(dave, sec_server, User).
```

***Query:** What user can Dave login as to the secure server?*

***Note:** User is a free variable; a value will be assigned to it if the query is satisfiable*

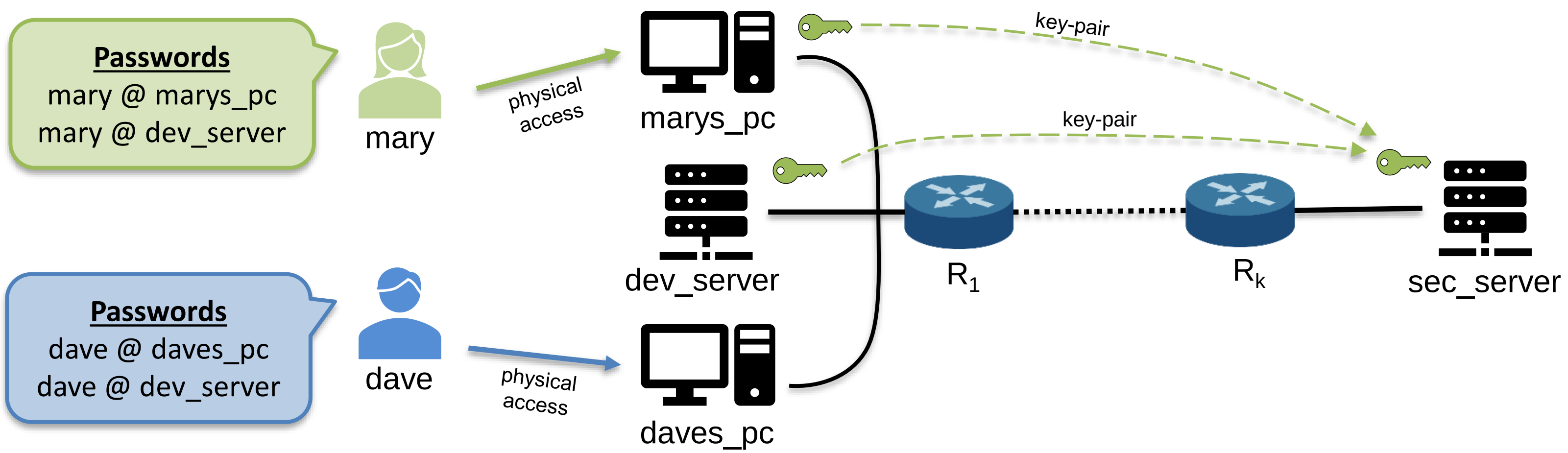
Configuration Modeling: Deep(er) Dive




```
actorAccess(dave, sec_server, User) :-
   hasPhysicalAccess(dave, Host),
```

What computers can Dave physically access?

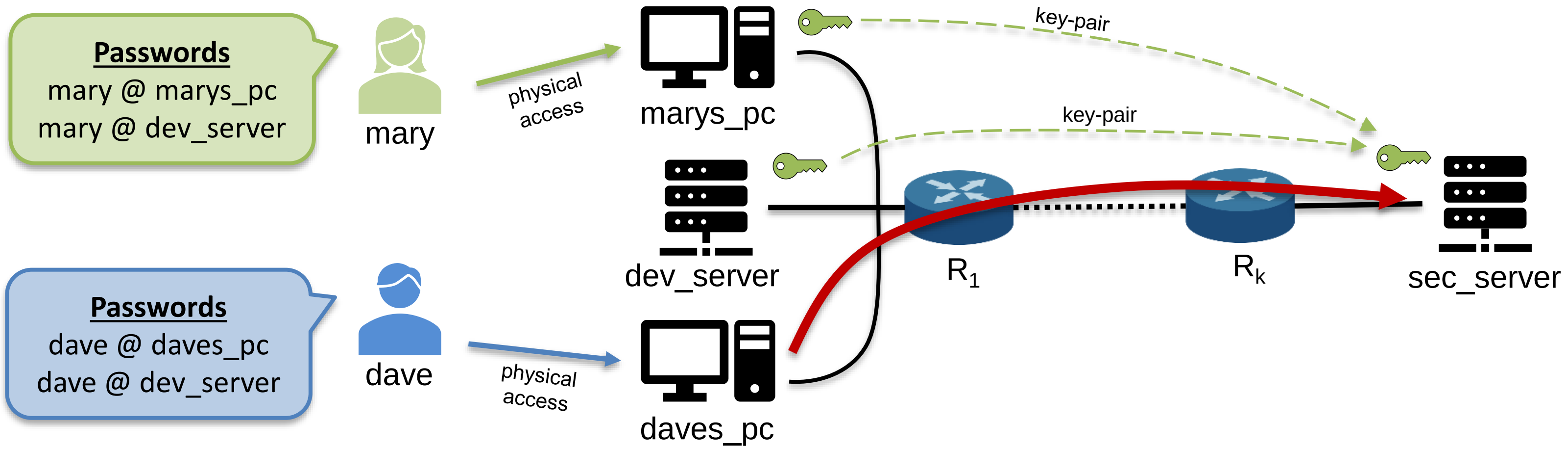
Configuration Modeling: Deep(er) Dive




```
actorAccess(dave, sec_server, User) :-
   hasPhysicalAccess(dave, daves_pc), ☑
```

What computers can Dave physically access?
(derived trivially from the actor set up)

Configuration Modeling: Deep(er) Dive



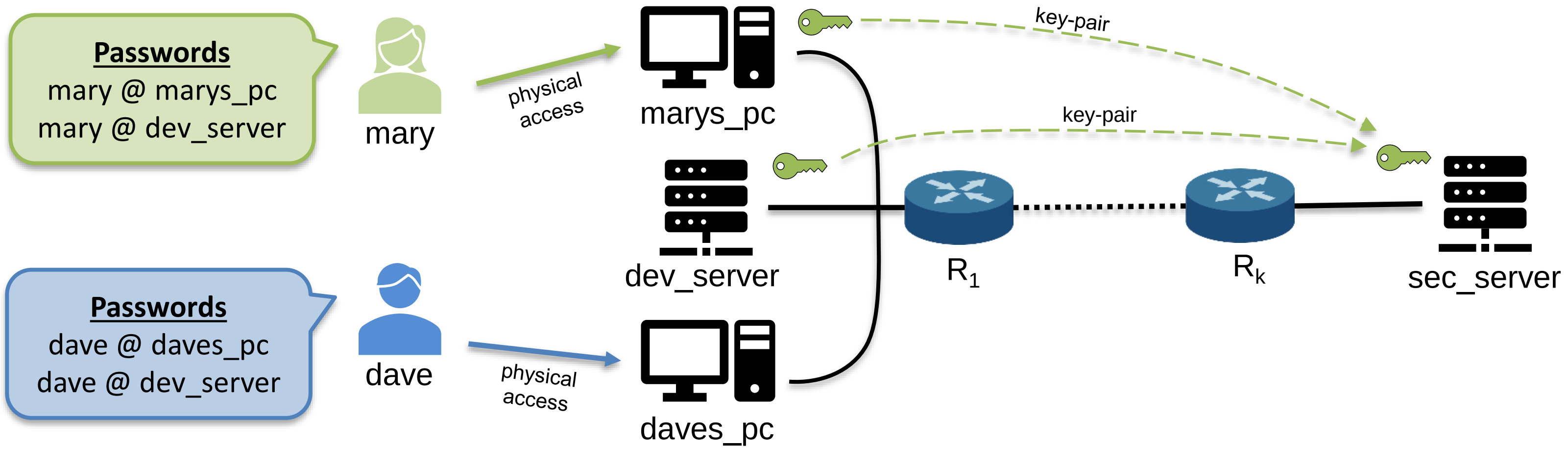

```
actorAccess(dave, sec_server, User) :-
   hasPhysicalAccess(dave, daves_pc),
   netAccess(daves_pc, sec_server, 22, tcp), ☑
```

Is there network connectivity between Dave's PC and secure server to support an SSH session?

Relevant configs: *routing, firewall configuration for routers R_1 through R_k*

For brevity: we assume it is possible, and will not expand netAccess definition

Configuration Modeling: Deep(er) Dive



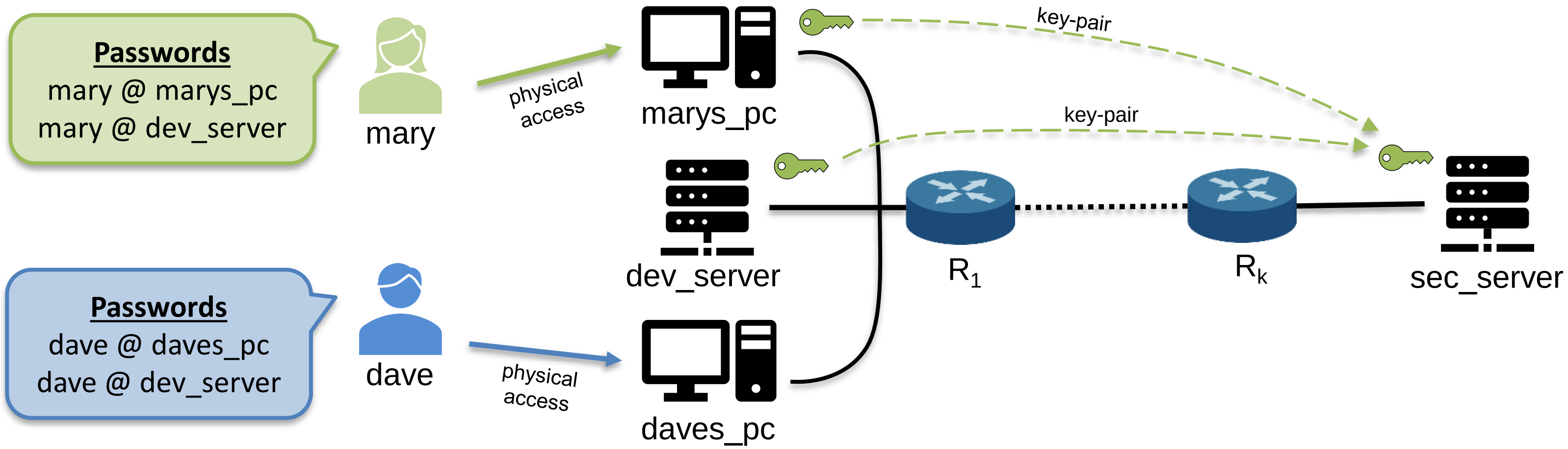

```
actorAccess(dave, sec_server, User) :-
   hasPhysicalAccess(dave, daves_pc),
   netAccess(daves_pc, sec_server, 22, tcp),
   remoteLogin(dave, sec_server, User).
```

Can Dave login into the secure server?

Relevant configs: *SSH configuration, user configuration (i.e., /etc/passwd, /etc/shadow contents) on the secure server.*

GRAMMATECH

Configuration Modeling: Deep(er) Dive



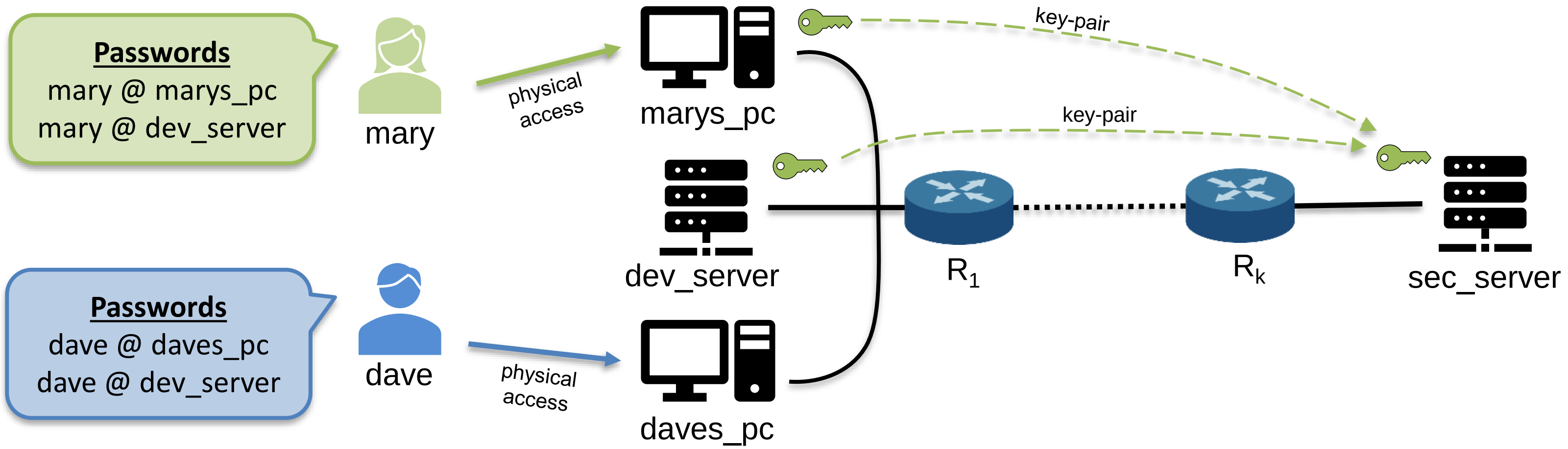

```
actorAccess(dave, sec_server, User) :-
   hasPhysicalAccess(dave, daves_pc),
   netAccess(daves_pc, sec_server, 22, tcp),
   remoteLogin(dave, sec_server, User) :-
      runsServiceSSH(sec_server, Service),
      sshPasswordAuthentication(Service),
      userPasswd(User, Passwd),
      actorKnowsPassword(dave, Passwd).
```

Does the secure server run SSH service ***and***

Does SSH support password-based login ***and***

Does Dave know the password for a sec_server user?

Configuration Modeling: Deep(er) Dive




```
actorAccess(dave, sec_server, User) :-
   hasPhysicalAccess(dave, daves_pc),
   netAccess(daves_pc, sec_server, 22, tcp),
   remoteLogin(dave, sec_server, User) :-
      runsServiceSSH(sec_server, Service),
      sshPasswordAuthentication(Service),
      userPasswd(User, Passwd),
      actorKnowsPassword(dave, Passwd). ☒
```

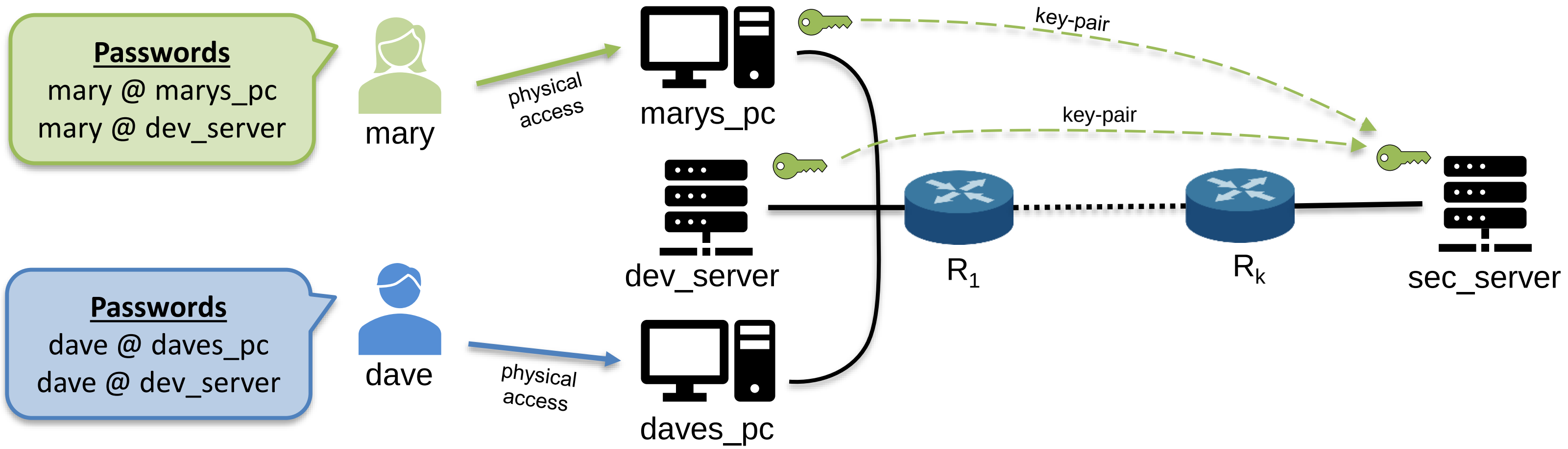
*Does the secure server run SSH service **and***

*Does SSH support password-based login **and***

Does Dave know the password for a sec_server user?

***Fails:** Dave does not know any relevant passwords*

Configuration Modeling: Deep(er) Dive




```
actorAccess(dave, sec_server, User) :-
   hasPhysicalAccess(dave, daves_pc),
   netAccess(daves_pc, sec_server, 22, tcp),
   remoteLogin(dave, sec_server, User) :-
      runsServiceSSH(sec_server, Service),
      sshPubkeyAuthentication(Service),
      sshPubkeyAccess(Service, User, PublicKey),
      keyPair(PublicKey, PrivateKey),
      fileContains(Host, Path, PrivateKey),
      actorFileRead(dave, Host, Path).
```

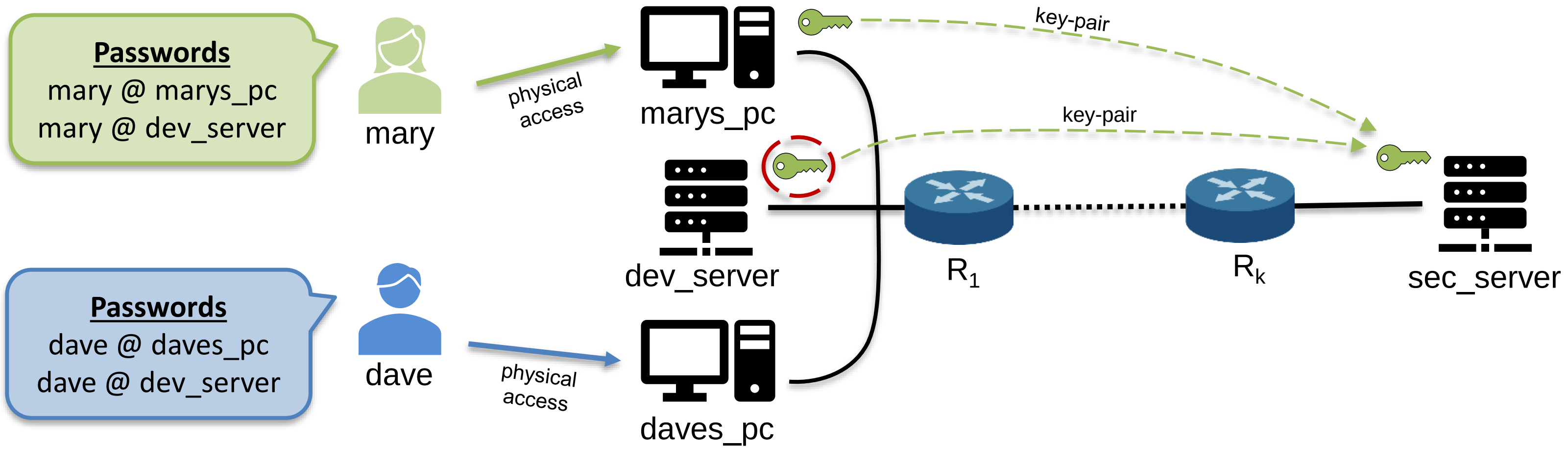
Alternative: PKI Authentication

*Does SSH support public-key-based login **and***

Can Dave get a private key needed for access

(e.g., can Dave read a file that stores a private key)?

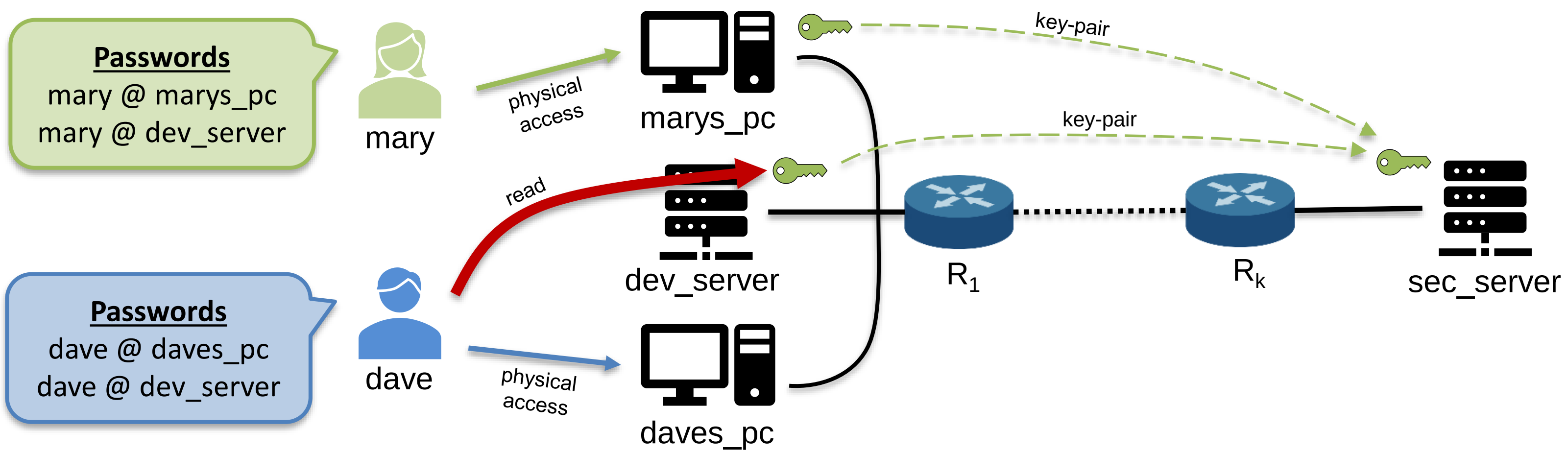
Configuration Modeling: Deep(er) Dive



Focus on Mary's private key stored on the development server:

```
actorAccess(dave, sec_server, User) :-
   hasPhysicalAccess(dave, daves_pc),
   netAccess(daves_pc, sec_server, 22, tcp),
   remoteLogin(dave, sec_server, User) :-
      ...,
      actorFileRead(dave, dev_server, "/home/mary/.ssh/id_rsa").
```

Configuration Modeling: Deep(er) Dive




```
actorAccess(dave, sec_server, User) :-
   hasPhysicalAccess(dave, daves_pc),
   netAccess(daves_pc, sec_server, 22, tcp),
   remoteLogin(dave, sec_server, User) :-
      ...,
      actorFileRead(dave, dev_server, "/home/mary/.ssh/id_rsa") :-
         actorAccess(dave, dev_server, dave@dev_server), ☑
         userSwitch(dave, dave@dev_server, root@dev_server), ☑
         localFileRead(
            dev_server, root@dev_server, "/home/mary/.ssh/id_rsa"). ☑
```

Focus on Mary's private key stored on the development server:

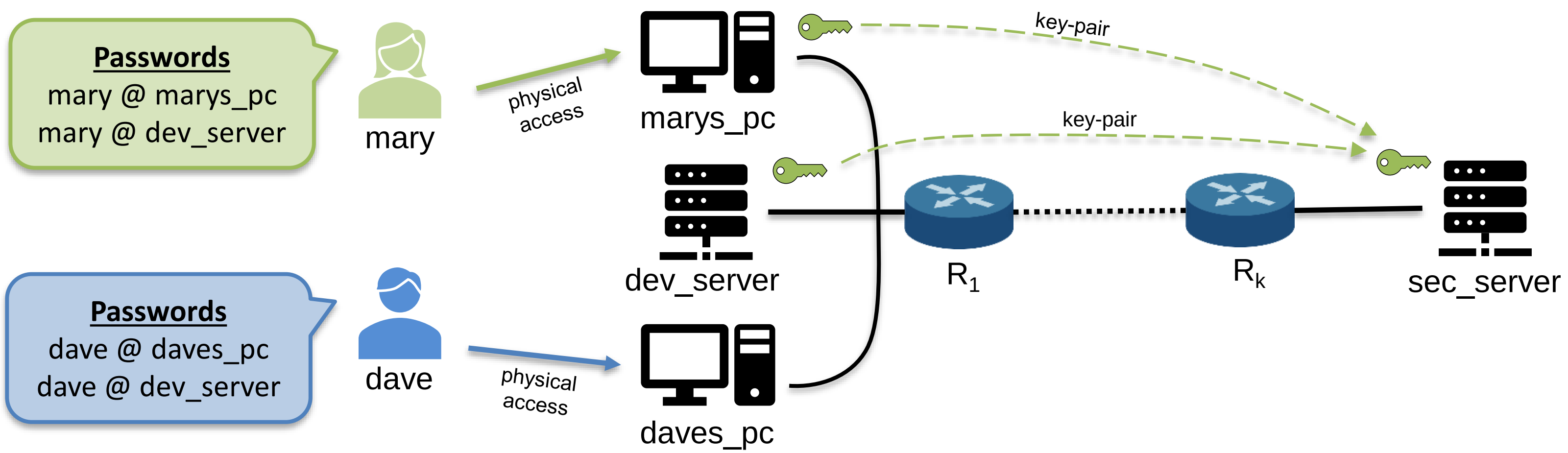
Dave can access dev_server
(recursive actorAccess definition)

Dave can elevate privileges to root
(due to weak sudo config)

Root can read Mary's files

GRAMMATECH

Configuration Modeling: Deep(er) Dive

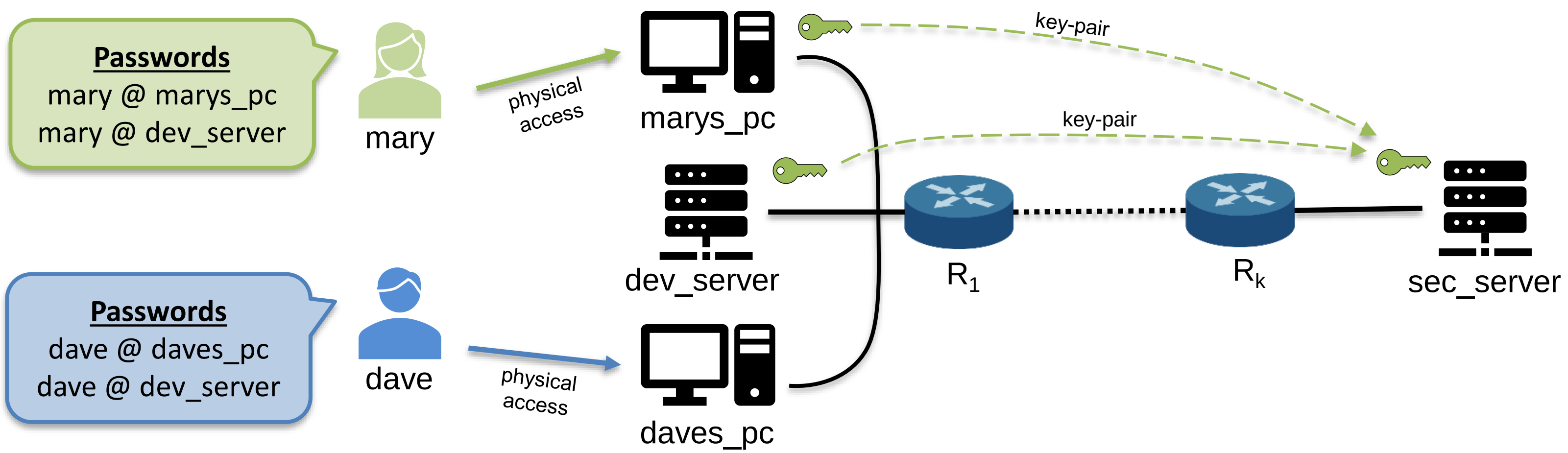


`actorAccess(dave, sec_server, mary@sec_server).` ☑

***Result:** Dave can log into Mary's account on the secure server*

(violation of the security policy)

Configuration Modeling: Deep(er) Dive



`actorAccess(dave, sec_server, mary@sec_server).` ☑

Result: *Dave can log into Mary's account on the secure server*
(violation of the security policy)

- Configuration modeling:
  - Systematically captures possible resource-access mechanisms
  - Supports parametric queries, not just Boolean ones
  - Tracks derivation to construct the sequence of steps for accessing the resource

NAWC Capture The Flag (CTF) System for Cyber Training

- Modeled CTF subset:
  - 20 devices connected to 6 or so subnets
- Resulting model stats:
  - 2K configuration parameters:
    - Firewall rule lists (for routers)
    - User to group mapping
    - Ownership and permissions for selected files
    - Subsets of options for ssh, tomcat, docker services
  - 4K facts and 200 rules
- Performance of Prolog-based solver:
  - 10K queries in under 2 minutes

<u>Sample Functional Requirements:</u>

- Web administrators shall be able to edit tomcat configuration files on the web server.
- Web developers shall be able to inspect tomcat logs.
- …

<u>Security Requirements</u>

- The principle of least privilege.

Configuration Modeling: Vision

- ***Challenge:*** diversity
  - Multitude of existing (and future) platforms, services, configuration surfaces

- ***Inspiration:*** re-targetable binary analyses
  - The use of uniform micro-code to specify semantics for various ISAs

- <u>Design principles:</u>
  - General formalisms for capturing semantics of diverse platforms / services
  - Semantic encoding that is agnostic to specific target analyses / backend solvers
  - Semantic encoding that enables diverse querying mechanisms
  - Extensible library of reusable modules

Further Research

- ***Automation:*** streamline library-extension efforts
  - Generate models for services, software (semi-)automatically
- ***Validation:*** boost modeling accuracy
  - Validate component models against real-world behavior
  - *Possible approaches:* testing, runtime monitoring, formal verification
- ***Scope:*** model properties beyond access control
  - *Privacy:* e.g., track which system data is encrypted and how
  - *Trust:* e.g., track how SSL certificates are set up and used
  - *More:* social engineering, service response times, user convenience?

Conclusion & Acknowledgements

- System Configuration Modeling Approach
  - ***Unified:*** uniform representation of various system-configuration aspects
  - ***Formal:*** well-understood systematic logical encoding (deductive database)
  - ***Modular:*** models are built from reusable building blocks
  - ***Extensible:*** formalisms and methodology for effective library extension

- ***Acknowledgements***
  - Research funded by DARPA under "Configuration Security" (ConSec) program

    The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.
  - Special thanks to other ConSec performers: STR, Peraton Labs, Siemens, PARC, Radiance, and Lockheed Martin for productive discussions and feedback!

GRAMMATECH