5. Access Control

Jin Hong

jin.hong@uwa.edu.au

Access Control

- To restrict access to places and resources
- Main features of Access Control includes:
  - (Authentication)
  - Authorization
  - Implementation and Audit
- Authentication verifies the user's identity, and access control uses this identity to grant or deny access.
  - So Authentication may not be needed in all circumstances
  - E.g.,?
- Access control uses the authorization process to either grant or deny access to systems or data
  - i.e., authorization defines policies on what a user or service may access.
  - Access control enforces these policies.

Access Control

- Authentication (password/crypto/etc.)
  - Who are you?
- Authorization (Access control)
  - What are you allowed to do.
  - Focus is policy
- Enforcement Mechanism
  - How its policy implemented/enforced

Night Club Example

- Authentication
  - ID Check
- Access Control
  - Over 18 - allowed in
  - Over 21 - allowed to drink
  - On VIP List - allowed to access VIP area
- Enforcement Mechanism
  - Walls, Doors, Locks, Bouncers

Online Liquor shop Example

- Authentication
  -
- Access Control
  -
  -
  -
- Enforcement Mechanism
  -

Other Examples

- Social Networks: Access to personal information.
- Web Browsers:
- Operating Systems:
- Memory Protection:
- Firewalls:

Controlled Sharing

- Express information access policies.
  - Who can access my files, who can't, how.

- Examples:
  - **Sandboxing**: Minimize damage inflicted by malicious program.
  - **Privilege Separation**: Allow programs to be written to minimize damage in case of failure.
    - Think about compartments in a ship

Defense-in-Depth

- programmers make mistakes, protection systems fail, redundancy always a good thing!!

Basic Design Space

- All security architectures are Isolation + Controlled Sharing
- Just Isolation
  - Unplug the network cable! (airgap provides *almost* perfect isolation).
  - Segment your network physically
    - Put red tape on one set of wires and machines, black tape on another, blue on another.
    - Make sure colors always match!
  - Military exploits this approach to separate classified, secret, topsecret networks.
- Most of us need to share stuff…what do we want from sharing?

Challenge of Controlled Sharing

All of Saltzer and Schroeder's principles are vital, these two are hard.

…

- Least privilege
  - Every program and every user of the system should operate using the least set of privileges necessary to complete the job.

…

- Psychological acceptability
  - It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly.

…

Reconciling these two is a fundamental challenge.

https://en.wikipedia.org/wiki/Saltzer_and_Schroeder%27s_design_principles

Access Control vs. IDS and AV

- Soundness (can I make a definite allow/deny decision)
  - Difference between access control and intrusion detection
  - An IDS makes a probabilistic statement i.e. this action is probably bad - allows catching a broader range of behaviors, at the cost of enforcement.
  - An IDS that is 100% accurate should be doing enforcement, it is then an IPS (access control system)
- Completeness (do I catch every bad action)
  - Access control vs. AV
  - Access control systems should (in theory enforce some property) -- e.g enforce these rules on information flow.
  - AV systems often rely more on un-sound blacklist approach, hard to make strict statements about what your getting
- In the real world, distinctions often unclear
  - Your AV product may do some access control

Subjects and Objects

- Subjects
  - can be processes, modules, roles
- Objects
  - can be files, processes, etc.

- Authentication often used to bootstrap subjects, but not necessary.
  - e.g. process assumes identity of one subject, then another.

Elementary Forms

- Authentication = Authorization
  - e.g. safes
- Whitelists/Blacklists (Single object, multiple Subjects)
  - Examples: Spam prevention
  - Blacklists:
    - Default on (fail open)
    - Hard to reason about who can access system
  - Whitelists:
    - Default off (fail closed)
    - Have to deal with adding whitelist entries
- Challenges
    - Hard to manage if rapidly changing set of principles
    - Both can grow quite large

Access Control Methods

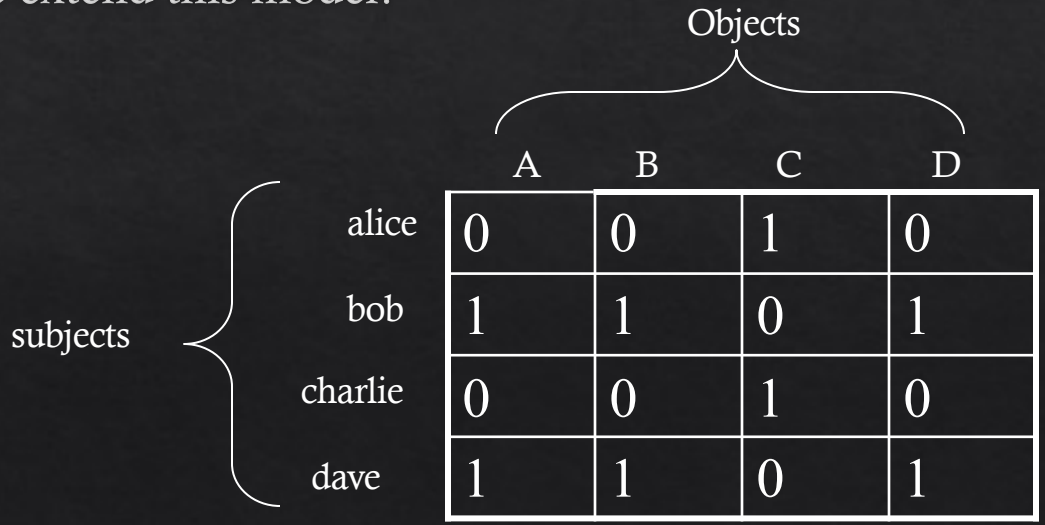
- Discretionary access control (DAC)
- Mandatory access control (MAC)
- Role-based access control (RBAC)
- Attribute-based access control (ABAC)
- Zero Trust

DAC

- DAC allows access rights to be propagated at subject's discretion
- often has the notion of owner of an object
- used in UNIX, Windows, etc.
- Let S be the set of all subjects, O the set of all objects, and P the set of all permissions. The description of access control can be given by a set $A \subseteq S \times O \times P$.
- When new permissions are added, triplets are added to A; when they are removed (revoked), triplets are deleted.

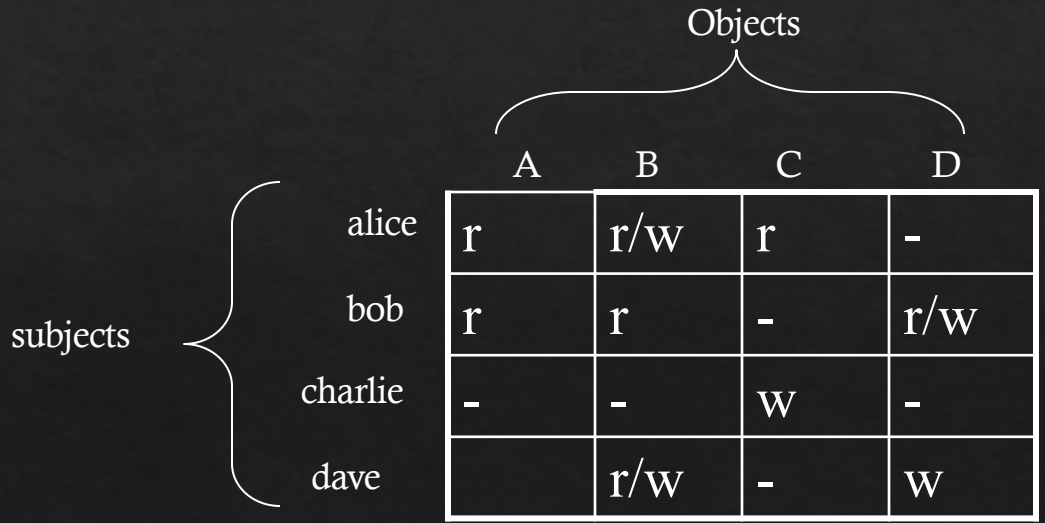
Access Control Matrix

- Instantaneous protection state of a system
- Dynamically Changing!
- How can we extend this model?



Adding Access Rights

- Access Rights
  - e.g. Simple: Read, Write
  - e.g. Complex: execute, change ownership



Grouping

- Subjects
  - Groups e.g. staff = {alice,dave}, students = {bob, charlie}
- Objects
  - Types e.g. system_file = {A,B}, user_file = {C,D}

File system ACL's

- Basic Problem
  - I want to do a project with a new group, how do I share files with only them without involving my sysadmin?
- Ghetto style
  - Create a directory accessible but not readable by everyone
  - Make directory (or files) in that directory with secret name
  - Hand out name as a capability
  - Gross and many limitations…
- ACL's a better solution
  - User created groups, user set-able list of groups/users on files/directories
- Almost everywhere now
  - AFS, NTFS, NFSv4, FreeBSD, Solaris, ZFS, etc.

ACL's

- What if I break my matrix down by columns?
  - Each object has a set of <user, right> tuples
  - A {<bob, r/w>, <alice,w>}
- Properties
  - Good for many applications (file systems)
  - Can grow quite large

ACL's

- What if I break my matrix down by rows
  - Alice {<A,r/w>, <B,w>, <C,r>}
- Properties
  - Natural model for delegation (rights coupled to object)
- Each tuple can be viewed as a handle to an object

- Let's look at UNIX example…

Operating Systems

- Most heavily studied area for access control
- First computers where single user/no network
- Timesharing systems introduced need for access control
  - studied heavily in the 70's and 80's.
- Still an open research area, why?
  - First 20 years: restrict what a user can do with data, focus on military problems, thought problem was malicious users
  - Since then: Malicious applications the primary problem.

- Another answer: Right Access control policy dictated by Usage models, Threat Model, Applications -- we still have lots to learn about how programs are built, how people use them.

Unix Resource

- Files
  - Includes devices, IPC (domain sockets, named pipes, mmap shared memory)
- Network
  - TCP/UDP port address space
- Other
  - Sys V IPC has its own name space
  - ability to load kernel modules

Unix Identities

- Each process has set of identities used to make access control decisions.
  - Conceptually uid/euid, gid/egid most important
  - In practice, a bunch of other details.
- euid used for access control decisions
  - Allows process to drop and resume privilege temporarily
- Changing uid/euid allows dropping privilege permanently.

File system access control

- "-rwxrwxrwx" Owner group name
- "-rw------- talg talg vimrc"
- "-r-sr-xr-x root wheel chpass"

- New files created according to "umask"
- setuid bit for executables - changes uid to uid of file owner on exec() of file.
- Execute on directory implies access to directory.

What can you do with identities

- Lots of hard coded rules
- Highest privilege uid is root (uid = 0), has most privilege,
  - Access privileged ports, override file permissions, load kernel modules, etc.
  - Once you have root, you own the system
  - this is a source of many problems.
- Process with a given euid can send signals to other processes with that uid, ptrace() those processes, etc.
  - Basically, no protection between processes with same uid.

Dropping privilege

- Only available to root
- To prevent programmer errors
  - Assume role of less privileged user for most operations (seteuid)
    - Let OS do its job
  - Only keep privilege long enough to do what you need.
  - Temporary, program can resume root privilege at will
  - Essential for some things in Unix (e.g. race free file open)
  - NEVER try to impersonate another user as root!
    - You will fail
- To prevent malicious code from inflicting damage
  - Drop privilege permanently (often accompanied by fork()).
  - Works well if there is a before-time (with privilege)/after-time(without privilege) model. e.g. daemons that need root to listen to privileged port.

Separating Time of Check and Time of Use A Really Bad Thing

- Example: access() system call.

```
int access(const char *path, int mode);
#From the man page:
#Access() is a potential security hole and should never be used.
```

- Problem: as soon as you get a result from access(), its invalid. Permissions could have changed.
  - Checking Permission /obtaining resource must be atomic!
- Solution: Never use access(), instead look up programming guidelines for your OS.

Hint: Never make up your own solution!

Confused Deputy Problem (Priv. Esc.)

- Pay-to-play compiler service (its an old problem)

1. Compiler service takes file and compiles it for user
2. User hands compiler path for compilers own billing file
3. Compiler overwrites billing file – but the user didn't have the privilege to overwrite!

Compiler was a confused deputy - acting on behalf of user.

- Solutions:
  - Drop privilege to that of user to open file (more error prone, common source of bugs)
  - Have user pass capability to compiler (less error prone).
  - Capability advocates love to point this out as reason to use capabilities.

Capabilities

- A group of operations in Linux
- Controlled independently on a per-thread basis, for example:
  - CAP_CHOWN: make arbitrary changes to file UIDs and GIDs
  - CAP_DAC_OVERRIDE: bypass read/write/execute checks
  - CAP_KILL: bypass permission checks for sending signals
  - CAP_NET_ADMIN: network management operations
- Each thread has zero or more capabilities tracked:
  - Permitted
  - Inheritable
  - Effective
  - Ambient

File descriptor passing: the poor mans capability system

- File descriptors are capabilities
- Ability to delegate files by passing descriptors over a unix domain socket (sendmsg(), recvmsg())
- Supports privilege separation
  - One process runs as root, opens a unix domain socketpair.
  - Forks() another process, drops privilege
  - More privilege process can pass descriptors to less privilege process, also do other stuff e.g. manage crypto keys.
  - Can apply this paradigm ad naseum i.e. multiple unprivileged processes.
- Trade-offs
  - Positive: Great way to make more robust software
  - Negative: *sometimes* very hard to do after the fact (don't believe the hype), requires pretty clueful programmers up front.

Capabilities have lots of nice properties

- Natural model for delegation
  - Just pass handle to object
- No ambient privilege
  - Access control explicit not implicit
- Separate access checking from use
  - E.g. fd = open() vs. read(fd), write(fd)
  - *Useful design pattern for ammoritizing the cost of permission checks*
- Powerful extensibility in languages that support capabilities natively
  - Wrap one object in another object.

Still More…

- Cryptographic Capabilities
  - Nonce as index into table (stateful)
  - {<permissions>, objectid, MAC(msg)} (stateless)
  - No OS support need, good for distributed systems
- Limitations of capabilities
  - No easy means of revocation
  - Can’t easily control delegation in a pure model (mostly a red herring)
  - Doesn’t always suit your problem

Dangers of bad access control model

- Too much privilege!
  - Original Unix architecture suffers severely
  - Windows world not any better
  - Once ISV's (developers) expect this, very hard to go back!
- Wrong granularity
  - To coarse grain => too much privilege
  - To fine grain => too many knobs, no one knows how to use it
    - (e.g., SELinux struggles with this, its gotten better, but still not for the faint of heart)
  - Compromise
    - Protection Profiles, Roles, etc. - use fine grain permissions to build up common case profiles.

A few words on Policy Languages

- Policy language at heart of rationale for access control
  - Look at policy spec instead of entire program/system
  - Clarity/simplicity key
- Even ACLs have a policy language
- Different requirements than programming language
  - Often non-expert users
  - Must be right the first time!
- Tension between flexibility/expressiveness and simplicity
  - Compare ACL's, to AppArmor to SELinux

DAC issues

- The underlying philosophy in DAC is that subjects can determine who has access to their objects.
- There is a difference, though, between trusting a person and trusting a program.
- The copies of file are not controlled

- Solution: use MAC ☺

MAC

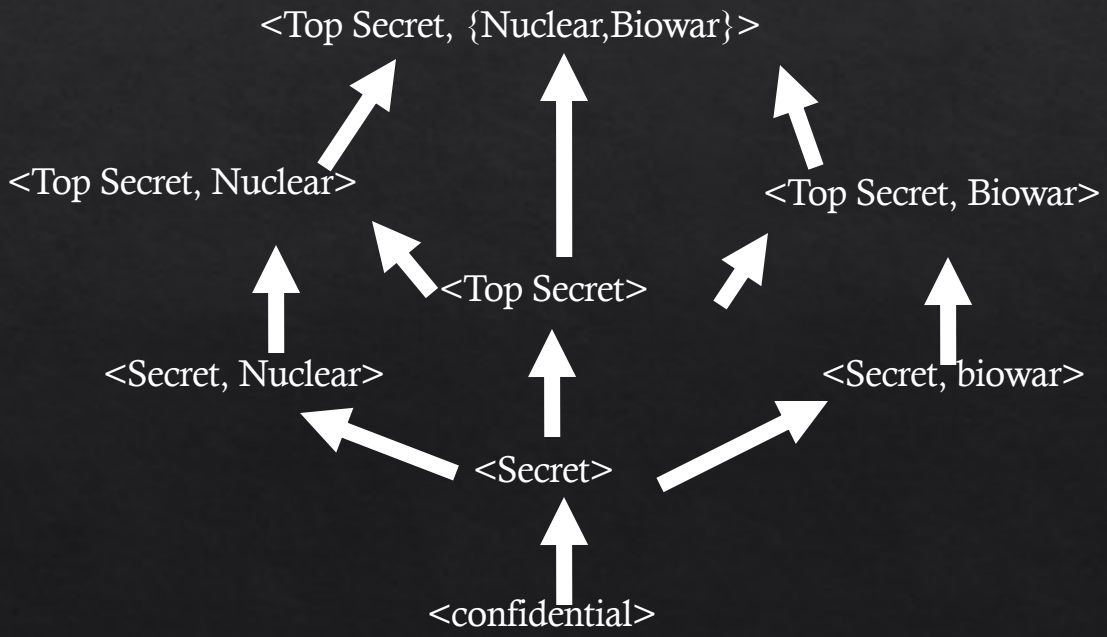
- Assigning access rights based on regulations by a central authority
- Implemented using a “reference monitor”
- Small Trusted Computing Base (TCB) [John Rushby, 1981, OSP]
- Implemented using Virtualization
- TOCTTOU (Time Of Check To Time of Use) problem:
  - authority checks access to an object
  - unknowingly to him, attacker replaces object with another one
  - privileged subject operates on attacker controlled object!

Classical military model

- Vertical classification levels
  - Confidential < Secret < Top Secret
- Horizontal compartments
  - Nuclear, SIGINT, Biowar, etc.
- User has a level, can't read above that level (violates secrecy), can't write below that level (leaks data).
- Process can begin with
  - Upperbound (read),lowerbound (write)
  - Lower bound raised every time information read from above.

Bell-Lapadula

- No read up, no write down



MLS in the world

- Classification level added by tagging files
- Everything tends to flow up
  - Ends up being very cumbersome in practice
- Declassification
  - Don’t know how to automate this, human in the loop
- Almost no one uses this in practice
- Tagging data with sensitive labels a cool idea, maybe could be used to help protect your personal data from malware
  - Ongoing research area.

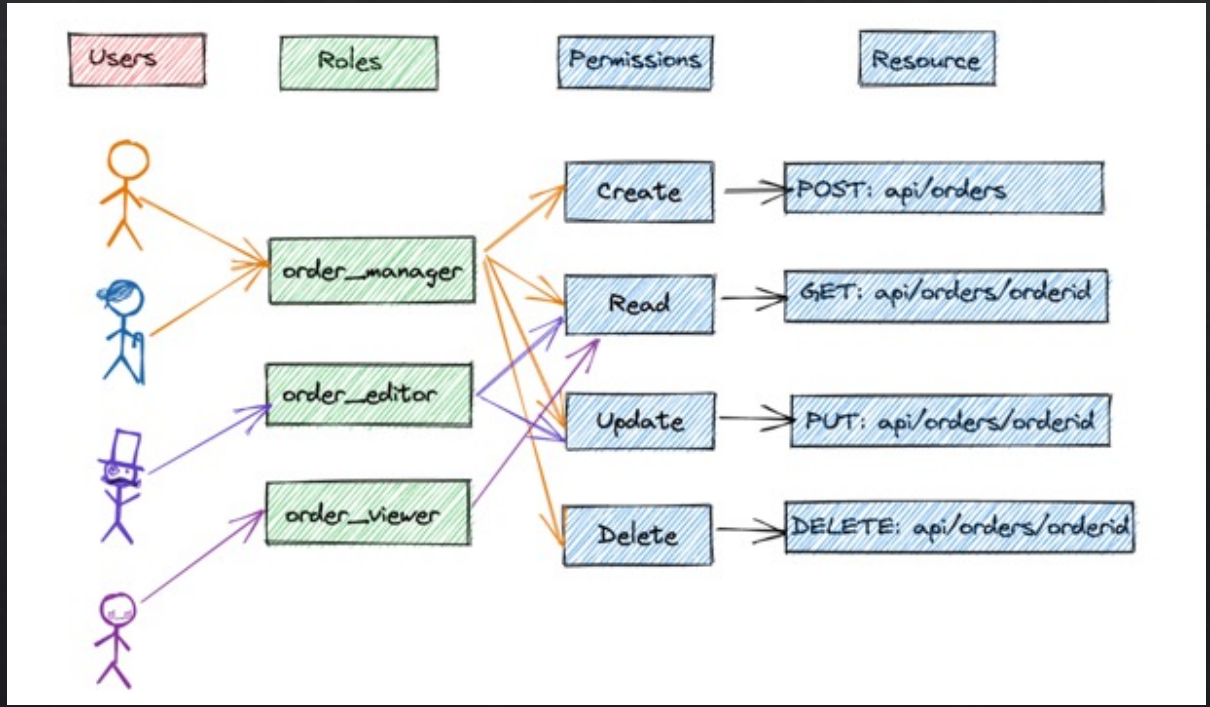
MAC issues - The Confinement Problem

- In real life, really really hard to prevent collaborating processes from communicating (say in order to write down).
  - Often due to timing channels
- Communication path called a “covert channel”
- Real world systems that care (very very few) try to limit bandwidth.
- Lampsons “Notes on the confinement Problem” a must read.

RBAC

- In the real world, security policies are dynamic.
- E.g., a user promotes at his job, therefore his rights must change (deleted, added, etc.)
- Changes the underlying subject--object model
  - a policy is a relation on roles, objects, and rights
- Subjects are now assigned to roles;
  - role assignment
- Roles are hierarchical

RBAC


Users
Roles
Permissions
Resource
order_manager
order_editor
order_viewer
Create
Read
Update
Delete
POST: api/orders
GET: api/orders/orderid
PUT: api/orders/orderid
DELETE: api/orders/orderid

RBAC

- Roles can be used to enforce policies
  - A role brings together
    - a collection of users and
    - a collection of permissions
  - These collections will vary over time
  - A user can be a member of many roles
  - Each role can have many users as Each role can have many users as members

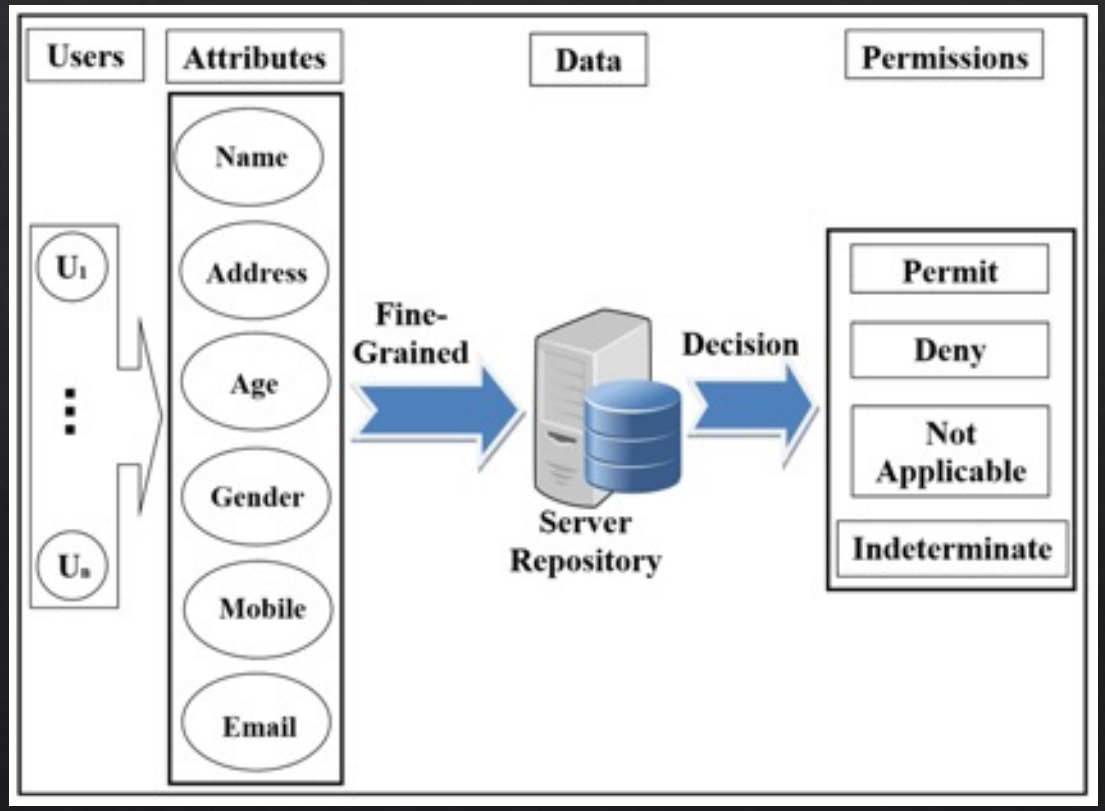
RBAC limitations

- Role granularity is not adequate leading to role explosion
- Role design and engineering is difficult and expensive
- Assignment of users/permissions to roles is cumbersome
- Adjustment based on local/global situational factors is difficult

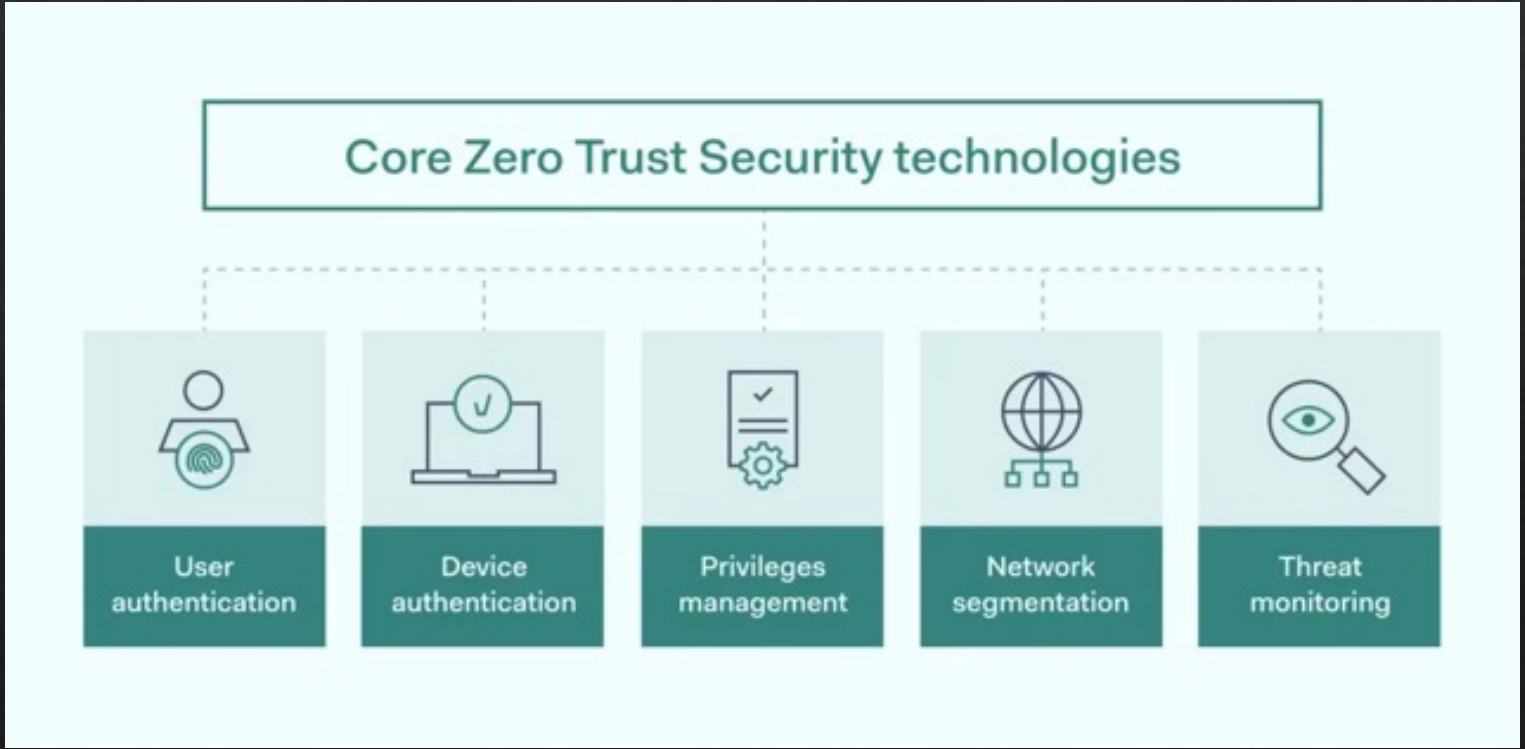
ABAC

- Manages access rights by evaluating a set of rules, policies and relationships using the attributes of users, systems and environmental conditions.
- This allows more dynamic approach to access control:
  - Time of day
  - Location
  - Device ID
  - User behaviour
  - Etc.

ABAC

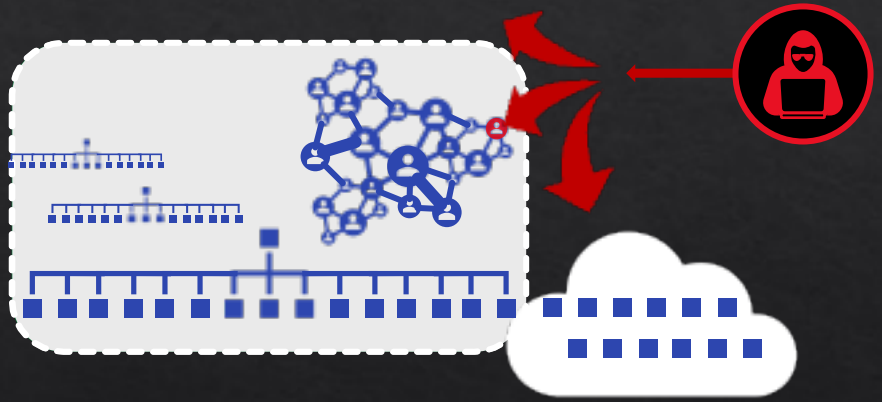

Users
Attributes
Data
Permissions
Name
Address
Age
Gender
Mobile
Email
U1
Un
Fine-Grained
Server Repository
Decision
Permit
Deny
Not Applicable
Indeterminate

Zero Trust


Core Zero Trust Security technologies
User authentication
Device authentication
Privileges management
Network segmentation
Threat monitoring

Zero Trust – the MS approach

1. IT Security is Complex
   - Many Devices, Users, & Connections
2. "Trusted network" security strategy
   - Initial attacks were network based
   - Seemingly simple and economical
   - Accepted lower security within network
3. Assets increasingly leave network
   - BYOD, WFH, Mobile, and SaaS
4. Attackers shift to identity attacks
   - Phishing and credential theft
   - Security teams often overwhelmed



Zero Trust - Principles

Verify Explicitly

Always authenticate and authorize based on all available data points, including user identity, location, device health, data classification, and anomalies.

Least Privilege

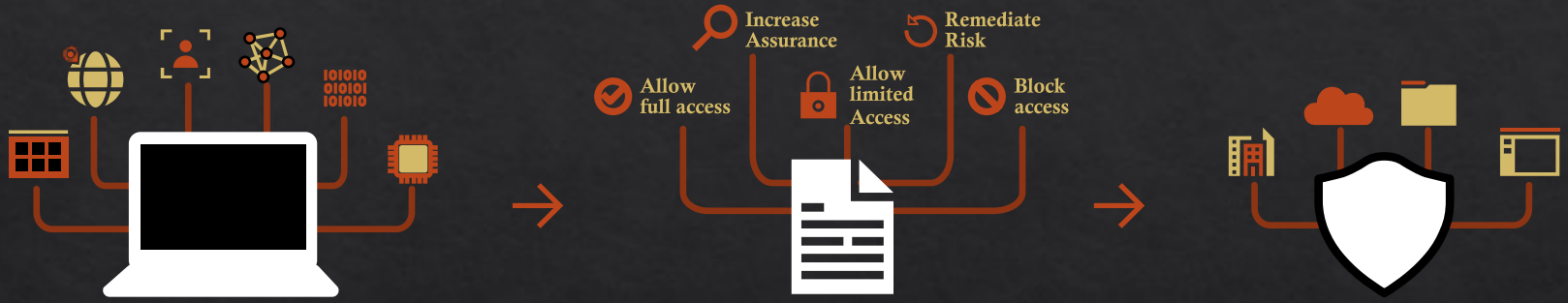
Minimize user access with Just-In-Time and Just-Enough Access (JIT/JEA), risk-based adaptive polices, and data protection which protects data and productivity.

Assume Breach

Minimize scope of breach damage and prevent lateral movement by segmenting access via network, user, devices and application awareness. Verify all sessions are encrypted end to end. Use analytics to get visibility and drive threat detection.

Zero Trust Access Control Strategy

Never Trust. Always verify.



Signal

to make an informed decision

Device Risk
- Device Management
- Threat Detection
- and more…

User Risk
- Multi-factor Authentication
- Behavior Analytics
- and more…

Decision

based on organization's policy

Apply to inbound requests

Re-evaluate during session

Enforcement

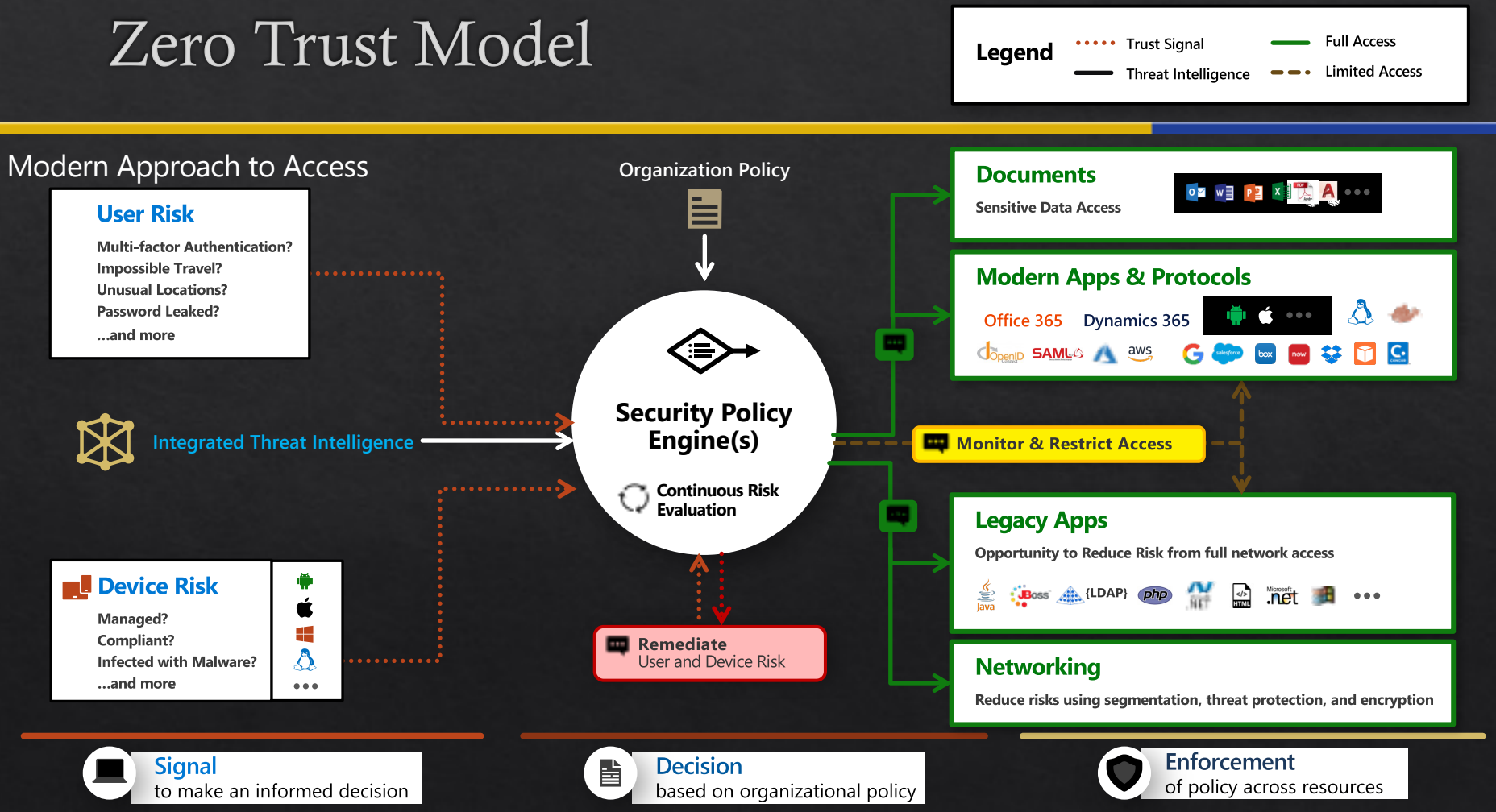
of policy across resources

Modern Applications
SaaS Applications
Legacy Applications
And more…

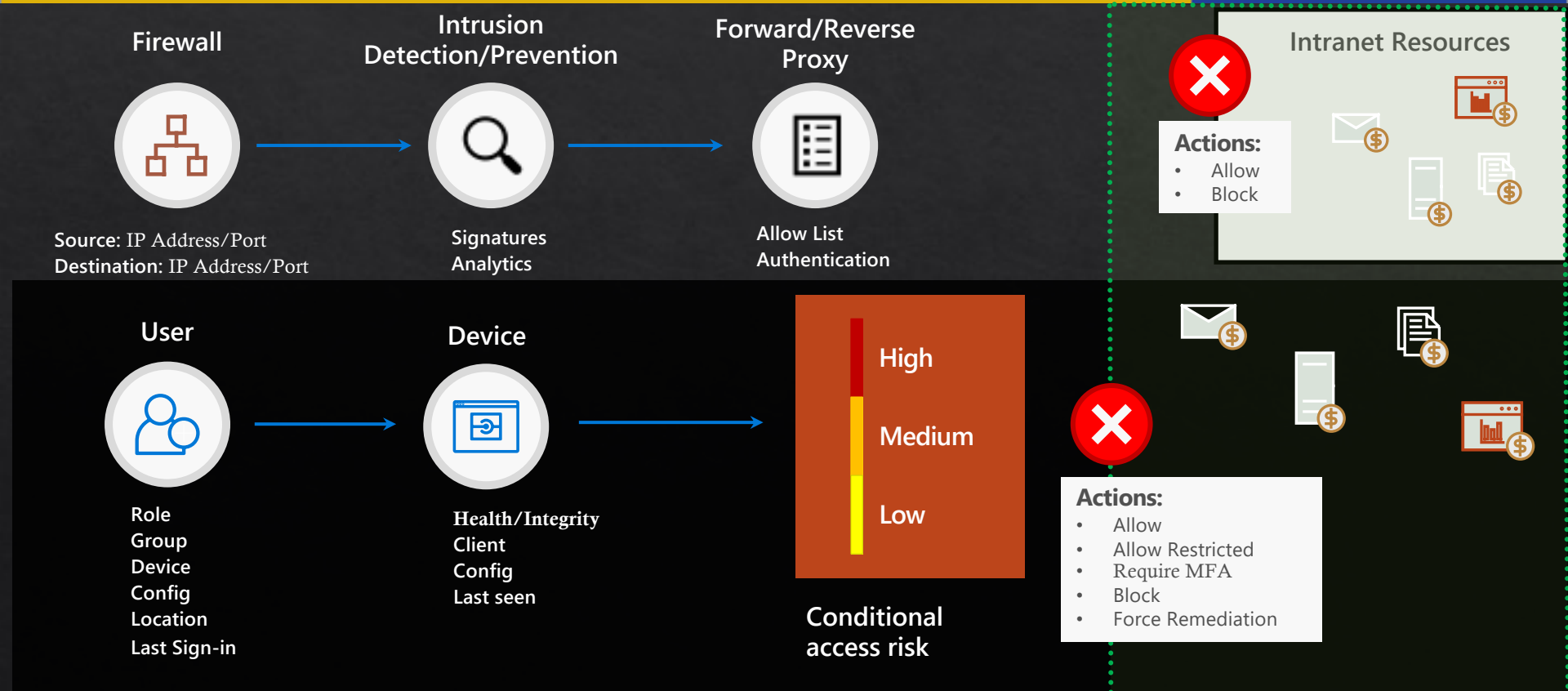
Zero Trust Access Control Paradigms

	Network	Identity
Control Plane	*Apply Zero Trust Policy to **network connections***	*Apply Zero Trust Policy to **access requests***
Industry Proponents	Network Security Vendors	Identity Vendors
Overall Effect	**Microsegmentation** enhances existing network perimeter by shrinking "trusted network" to each server / IP address.	**Dual Perimeter –** Adds an identity perimeter where "inside" is defined by authentication and authorization. *Coexists with network perimeter*
Applicability/Scope	**Limited to networks** controlled by customer. Doesn't protect modern SaaS and PaaS assets. *Microsegmentation approach varies by vendor*	**Applies to all assets –** • Natively protects modern cloud assets • Protects legacy intranet assets via proxy
Differentiation	**Scope of assets** where zero trust is enforced **Threat Intelligence** signal Integration	Integration of **Behavior Analytics (UEBA)** risk signal Use of **ML** across large datasets decisions Microsoft focuses on protecting modern and legacy assets as well as integration of ML, UEBA, and massive diverse threat intelligence
Common Components	Evaluate trust signals for Devices & User Identities with per application policy	

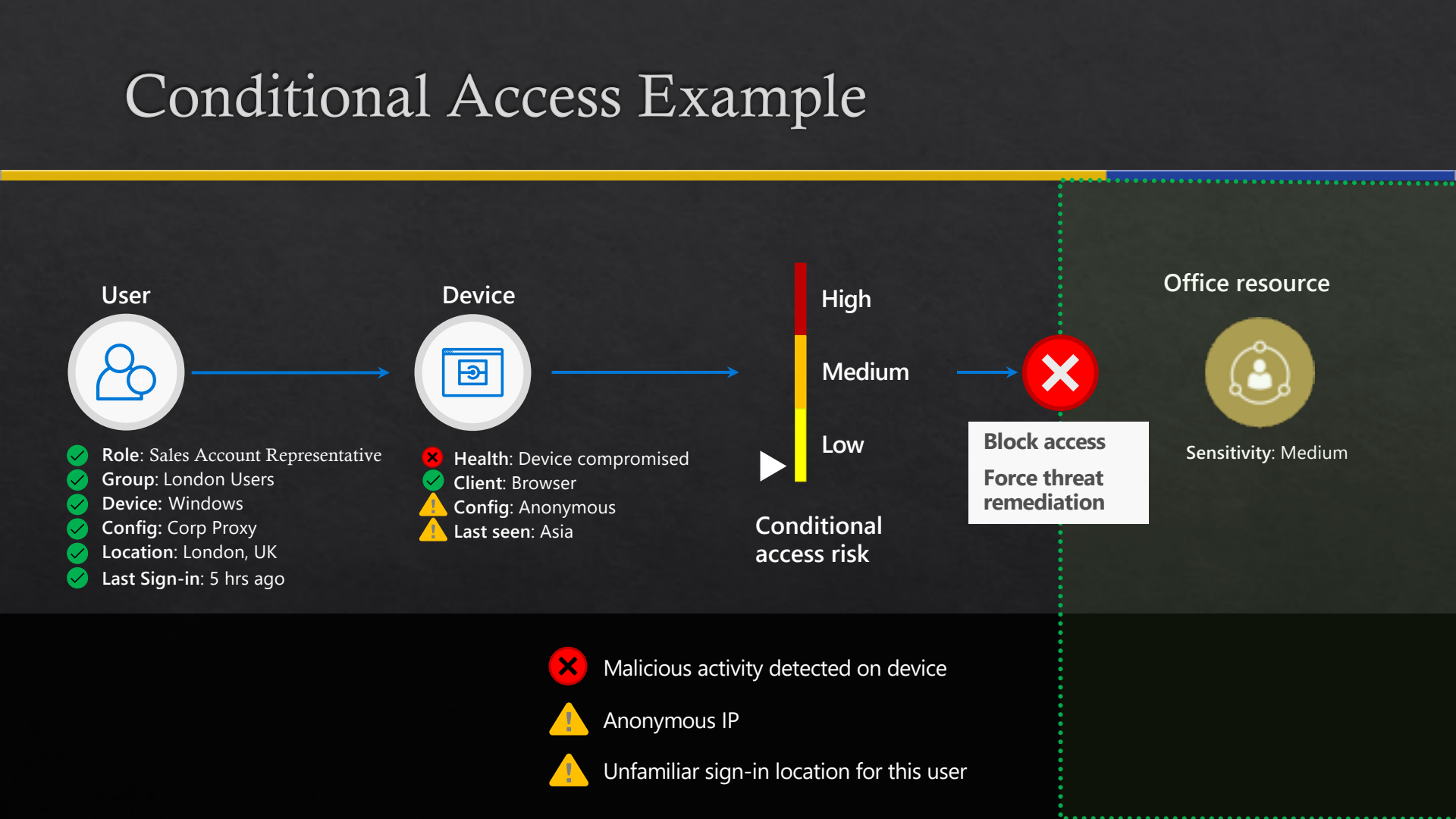
Zero Trust Model



Visibility and Control



Conditional Access Example



Zero Trust - Limitations

- Difficult to buy off-the-shelf all-in-one product
  - Why?
- Interoperability with legacy system
- Security vs Performance
- Complex system – gaps for threats

Conclusion

- Many attacks seen in the wild could have been rendered moot if software used access controls properly.
- Learn the access control system of your language runtime and OS
  - Learn their strengths and pitfalls.
- Build systems according to saltzer and schroeders principles.
  - Be conservative in your design, assume things will fail, attempt to minimize harm
- Building things right requires a bit more effort.
- Fixing things later requires far more
  - sometimes impossible or impractical.

References

- ISC Security Crunch
- Tal Garfinkel
- Microsoft Cybersecurity Solutions Group

Additional Resources

- Capabilities
  - https://people.cs.rutgers.edu/~pxk/419/notes/confinement.html
- Azure AD and ADFS best practices
  - https://cloudblogs.microsoft.com/enterprisemobility/2018/03/05/azure-ad-and-adfs-best-practices-defending-against-password-spray-attacks/
- Microsoft Password Guidance
  - https://aka.ms/passwordguidance
- NIST Updated Password Guidance

- Ignite Session: Azure Active Directory risk-based identity protection
  - https://channel9.msdn.com/events/Ignite/Microsoft-Ignite-Orlando-2017/BRK3016