What is security?

Different things to different people:

On June 8, as the investigation into the initial intrusion proceeded, the response team shared with relevant agencies that there was a high degree of confidence that OPM systems containing information related to the background investigations of current, former, and prospective Federal government employees, and those for whom a Federal background investigation was conducted, may have been compromised.

Computer Security

- Protecting my interests that are under computer control from malign threats
- Inherently subjective
  - Different people have different interests
  - Different people face different threats
- Don’t expect one-size-fits-all solutions
  - Grandma doesn’t need an air gap
  - Windows alone is insufficient for protecting TOP SECRET (TS) classified data on an Internet-connected machine

Claiming system “security” only makes sense with respect to well-defined security objectives:

- Identify threats
- Identify set of secure system states
State of OS Security

• Traditionally:
  – Has not kept pace with evolving user demographics
    o Focused on e.g. Defence and Enterprise
  – Has not kept pace with evolving threats
    o Focused on protecting users from users, not apps they run
• Is getting better in some ways
  – Eg smartphone OSes implement stricter security than desktops
  – Less blatant holes in mainstream Oses
• Is getting worse in others:
  – OSes are still growing in size, functionality and complexity
  – Too few people understand how to write secure code
  – More people know how to attack

Good security mechanisms

• Are widely applicable
• Support general security principles
• Are easy to use correctly and securely
• Do not hinder non-security priorities (e.g. productivity, generativity)
  – Principle of "do not pay for what you don’t need"
• Lend themselves to correct implementation and verification

OS Security

• What is the role of the OS for security?
• Minimum:
  – provide mechanisms to allow the construction of secure systems
  – that are capable of securely implementing the intended users'/administrators' policies
  – while ensuring these mechanisms cannot be subverted

Security Design Principles

• Saltzer & Schroeder [SOSP '73, CACM '74]
  – Economy of mechanism – KISS
  – Fail-safe defaults – as in good engineering
  – Complete mediation – check everything
  – Open design – not security by obscurity
  – Separation of privilege – defence in depth
  – Least privilege – aka principle of least authority (POLA)
  – Least common mechanism – minimise sharing
  – Psychological acceptability – if it’s hard to use it won’t be
Common OS Security Mechanisms

- Access Control Systems
  - control what each process can access
- Authentication Systems
  - confirm the identity on whose behalf a process is running
- Logging
  - for audit, detection, forensics and recovery
- Filesystem Encryption
- Credential Management
- Automatic Updates

Security Policies

- Define what should be protected
  - and from whom
- Often in terms of common security goals (*CIA properties*):
  - **Confidentiality**
    - X should not be learnt by Y
  - **Integrity**
    - X should not be tampered with by Y
  - **Availability**
    - X should not be made unavailable to Z by Y

Security vs Safety

- **Security**
  - Confidentiality
  - Integrity
  - Availability

- **Safety**
  - Timeliness

Policy vs. Mechanism

- Policies accompany mechanisms:
  - **access control** policy
    - who can access what?
  - **authentication** policy
    - is password sufficient to authenticate TS access?
- Policy often restricts the applicable mechanisms
- One person’s policy is another’s mechanism
Assumptions

- All policies and mechanisms operate under certain assumptions
  - e.g. TS-cleared users can be trusted not to write TS data into the UNCLASS window
- Problem: implicit or poorly understood assumptions
- Good assumptions:
  - clearly identified
  - verifiable

Risk Management

- Comes down to risk management
  - At the heart of all security
  - Assumptions: no absolute security, what risks we are willing to tolerate?
  - Cost & likelihood of violation vs. cost of prevention
  - Gain vs cost for attacker
- Other risks:
  - we mitigate (using security mechanisms)
  - or transfer (e.g. by buying insurance)
- Security policy should distinguish which is appropriate for each risk
  - Based on a thorough risk assessment

Trust

- Systems always have trusted entities
  - whose misbehaviour can cause insecurity
  - hardware, OS, sysadmin ...
- Trusted Computing Base (TCB):
  - the set of all such entities
- Secure systems require trustworthy TCBs
  - achieved through assurance and verification
  - shows that the TCB is unlikely to misbehave
  - Minimising the TCB is key for ensuring correct behaviour

Assurance and Formal Verification

- **Assurance**:
  - systematic evaluation and testing
  - essentially an intensive and onerous form of quality assurance
- **Formal verification**:
  - mathematical proof
- Together trying to establish correctness of:
  - the design of the mechanisms
  - and their implementation
- **Certification**: independent examination confirming that the assurance or verification was done right
Covert Channels

- Information flow not controlled by security mechanisms
  - Confidentiality requires absence of all such channels
- **Storage Channel:**
  - Attribute of shared resource used as channel
    - time stamp on file, existence of file
  - Controllable by access control
- **Timing Channel:**
  - Temporal order of shared resource accesses
  - Outside of access-control system
  - Much more difficult to control and analyse
- **Other physical channels:**
  - Power draw
  - Temperature (fan speed)
  - Electromagnetic emanation
  - Acoustic emanation

Covert Timing Channels

- Created by shared resource whose timing-related behaviour can be monitored
  - network bandwidth, CPU load, memory latency ...
- Requires access to a time source
  - anything that allows processes to synchronise
  - Generally compare relative occurrence of two event sequences (clocks)
- Critical issue is channel bandwidth
  - low bandwidth limits damage
    - why DRM ignores low bandwidth channels
  - beware of amplification
    - e.g. leaking passwords, encryption keys etc.

Covert Channels vs Side Channels

- Trojan intentionally creates signal through targeted resource use
- Worst-case bandwidth
- Attacker uses signal created by victim’s innocent operations
- Much lower bandwidth

Summary: Introduction

- Security is very subjective, needs well-defined objectives
- **OS security:**
  - provide good security mechanisms
  - that support users’ policies
- Security depends on establishing **trustworthiness** of trusted entities
  - **TCB:** set of all such entities
    - should be as small as possible
  - Main approaches: assurance and verification
- The OS is necessarily part of the TCB
ACCESS CONTROL PRINCIPLES

AC Mechanisms and Policies

- AC Policy
  - Specifies allowed accesses
  - And how these can change over time
- AC Mechanism
  - Used to implement the policy
- Certain mechanisms lend themselves to certain kinds of policies
  - Some policies cannot be expressed using your OS’s mechanisms

Access Control

- **who** can access **what** in which ways
  - the "who" are called **subjects** (or **agents**)
    - e.g. users, processes etc.
  - the "what" are called **objects**
    - e.g. individual files, sockets, processes etc.
    - includes all subjects
  - the "ways" are called **permissions**
    - e.g. read, write, execute etc.
    - are usually specific to each kind of object
    - include those meta-permissions that allow modification of the protection state
      - e.g. own

Protection State

**Access control matrix** [Lampson’71] defines the **protection state** at particular time

<table>
<thead>
<tr>
<th></th>
<th>Obj1</th>
<th>Obj2</th>
<th>Obj3</th>
<th>Subj2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subj1</td>
<td>R</td>
<td>RW</td>
<td></td>
<td>send</td>
</tr>
<tr>
<td>Subj2</td>
<td>RX</td>
<td></td>
<td></td>
<td>control</td>
</tr>
<tr>
<td>Subj3</td>
<td>RW</td>
<td></td>
<td>RWX</td>
<td>recv</td>
</tr>
</tbody>
</table>

Subjects are also objects
Storing Protection State

- Not usually as access control matrix
  - too sparse, inefficient, dynamic
- Two obvious choices:
  - store individual columns with each object
    - defines the subjects that can access each object
    - each such column is called the object’s **access control list**
  - store individual rows with each subject
    - defines the objects each subject can access
      - aka subject’s **protection domain**
    - each such row is called the subject’s **capability list**

Access Control Lists (ACLs)

- Subjects usually aggregated into classes
  - e.g. UNIX: owner, group, everyone
  - more general lists in Windows, recent Linux
  - Can have negative rights (e.g. to overwrite group rights)
- Meta-permissions (e.g. own)
  - control class membership
  - allow modifying the ACL
- Implemented in almost all commercial OSes

Capabilities

- A **capability** [Dennis & Van Horn, 1966] is a capability list element

<table>
<thead>
<tr>
<th>Subj1</th>
<th>Obj1</th>
<th>Obj2</th>
<th>Obj3</th>
<th>Subj2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R</td>
<td>RW</td>
<td></td>
<td>send</td>
</tr>
</tbody>
</table>

  - **Names** an object to which the capability refers
  - **Confers** permission over that object
  - Capability is **prima facie authority** to perform an operation
    - System will perform operation if appropriate capability is presented
  - Less common in commercial systems
    - IBM System/38→AS/400→i-Series
    - KeyKOS (VISA transaction processing) [Bomberger et al, 1992]
    - More common in research: EROS [Shapiro’99], Cheri, seL4

Capability-Based Access Control

- Any system call is invoking a capability: `err = method(cap, args);`
- **Cap = Access Token:** Prima-facie evidence of privilege
  - Eg. thread, address space
  - Eg. read, write, send, execute...

Capabilities provide
- Fine-grained access control
- Reasoning about information flow
Capabilities: Implementations

- Capabilities must be unforgeable
  - Traditionally protected by hardware (tagged memory), eg System-38
  - Can be copied etc like data
- On conventional hardware, either:
  - Stored as ordinary user-level data, but unguessable due to sparseness
    - Contains password or secure hash: PCS [Anderson'86], Mungi
    - “sparse” capabilities
  - Stored separately (in-kernel), referred to by user programs by index/address, eg Mach [Accetta’86], EROS [Shapiro’99], seL4
    - “partitioned” or “segregated” capabilities
    - Like UNIX file descriptors
- Sparse capabilities can be leaked more easily
  - Huge amplification of covert channels!

ACLs and Capabilities: Duals?

- In theory:
  - Dual representations of access control matrix
- Practical differences:
  - Naming and namespaces
    - Ambient authority
    - Deputies
  - Evolution of protection state
  - Forking
  - Auditing of protection state

Duals: Naming and Namespaces

- ACLs:
  - Objects referenced by name
    - Eg: open("/etc/passwd",O_RDONLY)
  - Require a subject (class) namespace
    - Eg: UNIX users and groups
- Capabilities:
  - Objects referenced by capability
  - No further namespace required
  - Cannot even name object without access

Duals: Confused Deputies

- ACLs: separation of object naming and permission can lead to confused deputies
  - Problem is dependence on ambient authority
    - Deputy uses its own authority when performing action on behalf of client
  - Capabilities are both names and permissions
    - You can’t name something without having permission to it
    - Presentation is normally explicit (not ambient)
Duals: Evolution of Protection State

• ACLs:
  – Protection state changes by modifying ACLs
    o Requires certain meta-permissions on the ACL
• Capabilities:
  – Protection state changes by delegating and revoking capabilities
  – Fundamental properties enable reasoning about information flow:
    o A can send message to B only if A holds cap to B
    o A can obtain access to C only if it receives message with cap to C
  – Right to delegate may also be controlled by capabilities
    o e.g. A can delegate to B only if A has a capability to B that carries appropriate permissions
    o A can delegate X to B only if it has grant authority on X

Duals: Auditing of Protection State

• Who has permission to access a particular object (right now)?
  – ACLs: Just look at the ACL
  – Caps: hard to determine with sparse or tagged caps, or for partitioned
• What objects a can particular subject access (right now)?
  – Capabilities: Just look at its capabilities
  – ACLs: may be impossible to determine without full scan
• “Who can access my stuff?” vs. “How much damage can X do?”

Duals: Forking

• What permissions should children get?
• ACLs: depends on the child’s subject
  – UNIX etc.: child inherits parent’s subject
    o Inherits all of the parent’s permissions
    o Any program you run inherits all of your authority
  – Bad for least privilege
• Capabilities: child has no caps by default
  – Parent gets a capability to the child upon fork
  – Used to delegate explicitly the necessary authority
  – Defaults to least privilege

Interposing Object Access

Caps are opaque object references (pure names)
• Holder cannot tell which object a cap references nor the authority
• Supports transparent interposition (virtualisation)

Usage:
• API virtualisation
• Security monitor
  – Security policy enforcement
  – Info flow tracing
  – Packet filtering...
• Secure logging
• Debugging
• Lazy object creation
  – Initial cap to constructor
  – Replace by proper object cap
Duals: Saltzer & Schroeder Principles

<table>
<thead>
<tr>
<th>Security Principle</th>
<th>ACLs</th>
<th>Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Economy of Mechanism</td>
<td>Dubious</td>
<td>Yes!</td>
</tr>
<tr>
<td>Fail-safe defaults</td>
<td>Generally not</td>
<td>Yes!</td>
</tr>
<tr>
<td>Complete mediation</td>
<td>Yes (if properly done)</td>
<td>Yes (if properly done)</td>
</tr>
<tr>
<td>Open design</td>
<td>Neutral</td>
<td>Neutral</td>
</tr>
<tr>
<td>Separation of privilege</td>
<td>No</td>
<td>Doable</td>
</tr>
<tr>
<td>Least privilege</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Least common mechanism</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Psychological acceptability</td>
<td>Neutral</td>
<td>Neutral</td>
</tr>
</tbody>
</table>

Mandatory vs. Discretionary AC

- **Discretionary Access Control:**
  - Users can make access control decisions
    - delegate their access to other users etc.
- **Mandatory Access Control (MAC):**
  - enforcement of administrator-defined policy
  - users cannot make access control decisions (except those allowed by mandatory policy)
  - can prevent untrusted applications running with user’s privileges from causing damage

MAC

- Common in areas with global security requirements
  - e.g. national security classifications
- Less useful for general-purpose settings:
  - hard to support different kinds of policies
  - all policy changes must go through sysadmin
  - hard to dynamically delegate only specific rights required at runtime
- Becoming more of a mainstream issue recently
  - Can I prevent a browser plugin from leaking data?

Bell-LaPadula [1966] (BLP) Model

- **MAC Policy/Mechanism**
  - Formalises National Security Classifications
- **Every object assigned a classification**
  - e.g. TS, S, C, U
  - may also have orthogonal security compartments
    - Support need-to-know
- **Classifications ordered in a lattice**
  - e.g. TS > S > C > U
- **Every subject assigned a clearance**
  - Highest classification they’re allowed to learn
BLP: Rules

- **Simple Security Property** ("no read up"):
  - s can read o iff clearance(s) >= class(o)
  - S-cleared subject can read U,C,S but not TS
  - standard confidentiality
- **★-Property** ("no write down"):
  - s can write o iff clearance(s) <= class(o)
  - S-cleared subject can write TS,S, but not C,U
  - to prevent accidental or malicious leakage of data to lower levels
- In practice need exceptions
  - allow trusted entity to write down
  - de-classify

Confidentiality + Integrity

- BLP+Bibra allows no information flow across classes
- Practicality requires weakening
  - Assume high-classified subject to treat low-integrity info responsibly
  - Allow read-down
- **Strong ★-Property** ("matching writes only"):
  - s can write o iff clearance(s) = class(o)
  - Eg for logging, high reads low data and logs

Biba Integrity Model

- Bell-LaPadula enforces **confidentiality**
- Biba: Its dual, enforces **integrity**
- Objects now carry **integrity** classification
- Subjects labelled by **lowest** level of data each subject is allowed to learn
- BLP order is inverted:
  - s can read o iff clearance(s) <= class(o)
  - s can write o iff clearance(s) >= class(o)

Boebert's Attack

"On the inability of an unmodified capability cache to enforce the ★-property" [Boebert 1984]
- Shows an attack on capability systems that violates the ★-property
  - Low passes cap to write buffer to High, which can then write down
  - Where caps and data are indistinguishable (sparse, tagged)
  - Does not work against **partitioned** capability systems
Boebert’s Attack: Lessons

- Not all mechanisms can support all policies
- Many policies treat data- and access-propagation differently
  - Eg explicit grant capability (Take-grant model)
  - Cannot be expressed using sparse capability systems
- This does not mean that capability systems and MAC are incompatible in general

Decideability

- Boebert’s attack highlights the need for decideability of safety in an AC system
- Safety Problem: given an initial protection state s, and a possible future protection state s’, can s’ be reached from s?
  - i.e. can an arbitrary (unwanted) access propagation occur?
- Harrison, Ruzzo, Ullman [1975] (HRU):
  - undecidable in general
  - equivalent to the halting problem

Decideable AC systems

- The safety problem for an AC system is decideable if we can always answer this question mechanically
- Most capability-based AC systems are decideable:
  - instances of Lipton-Snyder Take-Grant access control model [1977]
  - Take-Grant is decideable in linear time
- Less clear for many common ACL systems

Summary: AC Principles

- ACLs and Capabilities:
  - Capabilities tend to better support least privilege
  - But ACLs can be better for auditing
- MAC good for global security requirements
- Certain kinds of policies cannot be enforced with certain kinds of mechanisms
  - e.g. ★-property with sparse capabilities
- AC systems should be decideable
  - so we can reason about them
ASSURANCE AND VERIFICATION

Common Criteria

  - ISO standard, for general use
  - evaluates QA used to ensure systems meet their requirements
  - Developed out of the famous US DOD “Orange Book”:
    Trusted Computer System Evaluation Criteria [1985]
- Target of Evaluation (TOE) evaluated against Security Target (ST)
  - ST: statement of desired security properties based on Protection Profiles

Assurance: Substantiating Trust

- Specification
  - unambiguous description of desired behaviour
- System design
  - justification that it meets specification
    - by mathematical proof or compelling argument
- Implementation
  - justification that it implements the design
    - by proof, code inspection, rigorous testing
- Maintenance
  - justifies that system use meets assumptions

Common Criteria: EALs

- 7 Evaluated Assurance Levels
  - higher levels = more thorough evaluation
    - higher cost
    - not necessarily better security

<table>
<thead>
<tr>
<th>Level</th>
<th>Requirements</th>
<th>Specification</th>
<th>Design</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAL1</td>
<td>not evaluated</td>
<td>Informal</td>
<td>not eval</td>
<td>not evaluated</td>
</tr>
<tr>
<td>EAL2</td>
<td>not evaluated</td>
<td>Informal</td>
<td>Informal</td>
<td>not evaluated</td>
</tr>
<tr>
<td>EAL3</td>
<td>not evaluated</td>
<td>Informal</td>
<td>Informal</td>
<td>not evaluated</td>
</tr>
<tr>
<td>EAL4</td>
<td>not evaluated</td>
<td>Informal</td>
<td>Informal</td>
<td>not evaluated</td>
</tr>
<tr>
<td>EAL5</td>
<td>not evaluated</td>
<td>Semi-Formal</td>
<td>Semi-Formal</td>
<td>Informal</td>
</tr>
<tr>
<td>EAL6</td>
<td>Formal</td>
<td>Semi-Formal</td>
<td>Semi-Formal</td>
<td>Informal</td>
</tr>
<tr>
<td>EAL7</td>
<td>Formal</td>
<td>Formal</td>
<td>Informal</td>
<td>Informal</td>
</tr>
</tbody>
</table>

EAL 1–4 “not for use in hostile environments”
Common Criteria Protection Profiles (PPs)

- Controlled Access PP (CAPP)
  - standard OS security, up to EAL3
- Single Level Operating System PP
  - superset of CAPP, up to EAL4+
- Labelled Security PP
  - MAC for COTS OSes
- Multi-Level Operating System PP
  - superset of CAPP, LSPP, up to EAL4+
- Separation Kernel Protection Profile (SKPP)
  - strict partitioning, for EAL6-7

COTS OS Certifications

- EAL3:
  - Mac OS X
- EAL4:
  - 2005: SuSE Enterprise Linux
  - 2006: Solaris 10 (EAL4+)
    - against CAPP (an EAL3 PP!)
  - 2007: Red Hat Linux (EAL4+)
- EAL6
  - Green Hills INTEGRITY-178B (EAL6+)
    - against SKPP
    - relatively simple hardware platform in TOE

SKPP on Commodity Hardware

- SKPP:
  - OS provides only separation
- One Box One Wire (OB1) Project
  - Use INTEGRITY-178B to isolate VMs on commodity desktop hardware
  - Leverage existing INTEGRITY certification
    - by “porting” it to commodity platform
  - Conclusion [NSA, March 2010]:
    - SKPP validation for commodity hardware platforms infeasible due to their complexity
    - SKPP has limited relevance for these platforms
  - NSA subsequently dis-endorsed SKPP

Common Criteria Limitations

- Very expensive
  - rule of thumb: EAL6+ costs $1K/LOC
- Too much focus on development process
  - rather than the product that was delivered
- Lower EALs of little practical use for OSes
  - c.f. COTS OS EAL4 certifications
- Commercial Licensed Evaluation Facilities licenses rarely revoked
  - Leads to potential “race to the bottom” (Anderson & Fuloria, 2009)
Formal Verification

- Based on mathematical model of system
- Two approaches:
  - Automated techniques based on model checking / abstract interpretation
  - Theorem proving (manual or partially automated)

Automatic Analyses

- Algorithms that analyse code to detect certain kinds of defects
  - Usually static analysis
- Cannot generally “prove” code is correct
  - Only certain properties
  - False positives
  - False negatives
- Can be sound: guaranteed to detect all potential bugs of a kind
  - No false negatives
- Relatively cheap, often highly scalable (but then typically not sound)
  - Tradeoff between completeness and cost

Static Analysis and Linux: A Sad Story

- Static analysis of Linux source [Chou & al, 2001]
  - Found high density of bugs, especially in device drivers
- Re-analysis 10 years later [Palix & al, 2011]
  - Density of bugs detectable by static analysis had not dropped a lot!

And the Result?

Unsafe at any clock speed: Linux kernel security needs a rethink

Ars reports from the Linux Security Summit—a final call to arms that needs to be done.