Guaranteeing Dependency Enforcement in Software Updates

Luigi Catuogno[1], **Clemente Galdi[2]** and Giuseppe Persiano[1]

[1]Università di Salerno

[2]Università di Napoli “Federico II”

Outline

- Motivating scenario
- Software update models
- The proposed solution
- More complex scenarios
  - Improved solutions and limitations

Motivating Scenario

- A flock of drones is accomplishing a mission
- Some of the drones' missions needs to be reconfigured automatically by the Central/Regional Operations Office.
- The set of drones to be reconfigured depend on their configuration/status.
  - Availability of specific tools/weapons
  - Fuel level
  - Role in the current mission

Underlying requirements

- On-the-fly mission reconfiguration
  - No human intervention/authorization
  - Mission specification is a sensitive information!
- Drones might be captured
  - Exposed material should not compromise future missions
- No or minimal hardware support for security

Software Update scenario

- *Hardware components* are specified by means of *software* modules (drivers)
- *Missions* are specified by means of *software* modules
- *Drones' selection* is defined by the *installation policy*
  - *A mission can be installed if and only if the configuration meets the installation policy!*
- Updates might be:
  - Human assisted: Installation of new hardware component
  - Automatic: On-the-fly mission reconfiguration.

Existing solutions

- Different software update models/systems are available:
  - Centralized *a-la* iOS
  - Semi-decentralized: Android
  - Fully distributed: Linux
- Different tools to secure updates:
  - Authentication: Package Signature, Code Signing
  - Confidentiality: Channel encryption
  - Dependency enforcement: optional!

The Actors: Linux philosophy

- (Multiple) Distribution Server: Trusted component
  - Produces properly formatted software packages
- Mirror Server: Untrusted component
  - Used by the DS to distribute software packages
- Device: Partially trusted
  - If captured discloses key material
- The adversary
  - Controls the communication channel
  - Has full access to captured devices.

Updaticator [ABCSS14]

- Idea: Each *device* is identified by *static attributes*
  - O.S., CPU Type, Speed, etc.
- Uses CP-ABE to protect software confidentiality
  - Installation policy is embedded in the ciphertext
  - Each device holding the proper set of attributes can decrypt the software and install it.
- Problem: Attributes updates requires one of the following:
  - Existence of a single centralized key authority
  - Complete system profiling (transfer of all keys to an update authority)
  - Interactive protocol for key updates.

Our requirements

- Attributes are *dynamic* in nature
  - Each new installed software enhances the device with a new attribute
- Multiple 'key' generation authorities
  - Each device might install software from different vendors
- Non-interactive key-updates
- Enforce installation policy in a "*strong sense*"
  - It is not possible to bypass it
- Typical security requirements
  - Software authenticity, integrity, confidentiality and freshness

A hidden assumption

If a package A depends on package B, the software vendor has already installed package B

- Otherwise, how can the vendor test software A ?
- When the package is being assembled, the vendor knows the *attributes* of all the required packages.

The idea

- Each software has two associated random keys
  - An encryption key, used to encrypt it
  - A package key, the 'attribute key'
- Package creation:
  - Encrypt (software, package key) using the encryption key
  - Share encryption key using the installation policy as an access structure.
  - Encrypt share for 'software j' with the package key of software j.
- Installation is possible iff the set of installed packages satisfies the installation policy.

DIE TI. UNI NA

The Package creation protocol

1. *Generate a random encryption key r*
2. $(s_1,\ldots,s_m) \leftarrow Distribute(r, A_\phi)$
3. For j=1 to m do
4. $\quad e_j \leftarrow Encrypt_{kj}(s_j)$
5. *Generate a random package key k*
6. Package p = (name, timestamp, Δ, metatada, package key k ,software)
7. $E \leftarrow \langle\langle(n_1,e_1),\ldots,(n_m,e_m)\rangle, A_\phi, Encrypt_r(p)\rangle$
8. $M_E \leftarrow CreateMetaData(n, t, \Delta, M_s, E)$
9. $\sigma_M \leftarrow Sign_{SkV}(M_E)$
10. $\sigma \leftarrow Sign_{SkV}(E)$
11. Send (E, σ), (M_E, σ_M) to Mirror Servers

Package Installation 1/2

Obtain (E,σ) and (M_E,σ_M) (possibly from different servers)

/* Authenticity, Integrity and Freshness Verification */

1. if $(\mathrm{Verify}_{PkV}(E,\sigma) = \perp) \vee (\mathrm{Verify}_{PkV}(M_E,\sigma_M) = \perp)$
   then Reject
2. if (E does not Match M_E) $\vee$ (M_E Not Fresh)
   then Reject

/* Decryption Key Reconstruction */

1. Parse E as $\langle((n_1, e_1), \ldots, (n_m, e_m)), A_\varphi, \mathrm{Encrypt}_k(p)\rangle$
2. for i = 1,...,m do
   if $(n_i,k_i) \in$ Installed Packages then $s_i = \mathrm{Decrypt}_{ki}(e_i)$
   else $s_i = \perp$
3. $r \leftarrow \mathrm{Reconstruct}(A_\varphi, (s_1, \ldots, s_n))$

Package Installation 2/2

/* Software Installation and Key Database update */

1. $p \leftarrow Decrypt_r(E)$
2. $M_s \leftarrow ExtractMetadata(M_E)$

   if $(p \neq \perp) \wedge$ (p matches M_s) then

   Parse p as (n,t,Δ,M_s,k,s)

   Install s

   Add (n,k) to the set of Installed Packages

Security Properties

- Confidentiality: guaranteed by package encryption
- Integrity, Authenticity: Enforced using standard tools
  - Hash functions, signatures, certificates…
- Freshness: Defined by the software vendor
- Policy enforcement: guaranteed by impossibility of decrypting the package without the knowledge of the proper keys.

Advantages

- Each package brings its own attribute, the package key.
- Attributes' updates are executed non-interactively
  - "Save the package key".
- Multiple-vendor updates are possible.

More complex scenarios

- What if the installation policy is not monotone ?
  - Software policies contain the ‘conflict’ clause
  - Only monotone access structures are possible for secret sharing schemes!

	#packages	#packages with conflicts	Percentage
CentOs	8652	377	4,4%
RHEL 7.1	4432	299	6,7%
openSUSE 13.2	5334	242	4,5%
Fedora 21	2477	127	5,1%

More complex scenarios

- What if installation policy itself is sensitive ?
  - Our first solution assumes the access structure to be transferred in clear to the device

The Package creation protocol

1. *Generate a random encryption key r*
2. $(s_1,\ldots,s_m) \leftarrow Distribute(r, A_\phi)$
3. For j=1 to m do
4. $\quad e_j \leftarrow Encrypt_{kj}(s_j)$
5. *Generate a random package key k*
6. Package p = (name, timestamp, Δ, metatada, package key k ,software)
7. $E \leftarrow \langle\langle(n_1,e_1),\ldots,(n_m,e_m)\rangle, A_\phi, Encrypt_r(p)\rangle$
8. $M_E \leftarrow CreateMetaData(n, t, \Delta, M_s, E)$
9. $\sigma_M \leftarrow Sign_{SkV}(M_E)$
10. $\sigma \leftarrow Sign_{SkV}(E)$
11. Send (E, σ), (M_E, σ_M) to Mirror Servers

Policy (Partial) Hiding Protocol

- The vendor *locally* anonymizes package names and policy
  - *Arbitrarily* maps each package name to an integer.
  - Describes installation policy in DNF over the set of anonymized package names

Policy (Partial) Hiding Protocol

- At installation time
  - For each clause in the DNF formula tries to decrypt each share using the keys in the local DB
  - Gets the key when it succeeds in decrypting all the shares in a clause
    - Depends on the number of packages installed on the device!

Policy (Partial) Hiding Protocol

- In theory anonymity comes to at a huge price: efficiency
  - A DNF formula might be exponentially longer than a compact representation
  - The device needs to blindly search in the proper key.
- In practice the impact might be affordable
  - DNF expansion is due to multiple version
  - On average the DNF formula is 25 time larger than 'compact' representation

Conclusions

- Presented a system that allows the enforcement of installation policies during software installation/updates
  - Multiple independent vendors
  - Non-interactive key updates
  - Installation policies depend on static properties, e.g., installed packages
- Presented an extension for partially hiding policies
- Started evaluating the performance of the scheme

Future Work

- Allow dynamic policies
  - Depending on ‘fuel level’, ‘current position’
- Consider non-monotone installation policies
- Reduce the impact of anonymization for the installation phase.