Secure Device Design via Protocol Analysis

By means of an example

Joshua D. Guttman John D. Ramsdell

High Assurance Software and Systems

Version of May 8, 2023

Example: Cryptographically Assured Information Flow

- Secure reprogrammable devices — Even remotely
  - End Cryptographic Units, Key loaders, Enterprise Mgt.
- Reprogrammable counts because:
  - Asymmetric algs may change — Digital Signatures
    e.g. in response to quantum threat
  - Application-level crypto too — ciphers + hashes
  - Other code may evolve — Key mgt

Minimal hardware to ensure
we control our programs and keys on device

Adapted from Trusted Execution Environments

MITRE

Adversary model for secure reprogrammability

If we can reprogram it,
maybe the adversary can too?

- Goals preserved even if adversary:
  - installs malicious software on my devices
    or modifies my software maliciously
- Must assure:
  - My data delivered only to my programs — confidentiality
  - My programs act only on my data — integrity
- Payoff for:
  - End Cryptographic Units, Data Transfer Devices, Enterprise mgt

CAIF mechanism: Services

- Services are programs with
  - Isolated address space
  - Unchanging executable code segment
  - Hash of code segment is service identity
- CAIF mechanism maintains hash of code segment
- CAIF uses code hash for:
  - Provenance:

    Who prepared this data for me?

  - Protection:

    Who can receive this data from me?

CAIF mechanism: Instructions

Two pairs of instructions
to control flow between services
as identified by code hash

- For protection + provenance:
  - `protect-for` and `retrieve-from`

    Symmetric authenticated encryption
- For provenance:
  - `attest-locally` and `check-attest`

    Message Authentication Codes
- Focus: `protect-for` / `retrieve-from`

MITRE

Instruction pair: protect-for / retrieve-from

$$\begin{array}{lll} \texttt{prot-for}\ v,\ dh & \text{encrypt}\ \{\!|\ v\ |\!\}_k & sh :=_{\text{caif}} ch(current) \\ \texttt{rtr-from}\ \{\!|\ v\ |\!\}_k,\ sh & \text{decrypt}\ v & dh :=_{\text{caif}} ch(current) \end{array}$$

- Device has a (purely local) intrinsic secret IS
- Keys derived via IS, current service and intended peer each identified by code hash $ch(svc)$

$$k \quad = \quad kdf(\texttt{"pf"},\ IS,\ sh,\ dh)$$

MITRE

Core questions

For a CAIF device d

Local to d: Can a service svc on d determine local service

1. src as source of data value v
2. dst as sole destination of data value v

Remote from d: Can principal not on d determine a service svc on d as the

1. source of incoming value v
2. sole destination of outgoing value v

"Assured remote execution" by svc on d

MITRE

Core questions
For a CAIF device d

Local to d: Can a service svc on d determine local service

1. src as source of data value v
2. dst as sole destination of data value v

Yes, by construction of protect-for / retrieve-from

Remote from d: Can principal not on d determine a service svc on d as the

1. source of incoming value v
2. sole destination of outgoing value v

"Assured remote execution" by svc on d

MITRE

Core questions

For a CAIF device d

Local to d: Can a service svc on d determine local service

1. src as source of data value v
2. dst as sole destination of data value v

Yes, by construction of protect-for / retrieve-from

Remote from d: Can principal not on d determine
a service svc on d as the

1. source of incoming value v
2. sole destination of outgoing value v

"Assured remote execution" by svc on d

More challenging: Requires device-rooted protocol analysis
despite an adversary that can run programs

MITRE

Anchor for assured remote execution method

- Via shared secret key k_s
- Assumption 1: Each device has a distinct, publicly known
  - Immutable ID $imid$
- Assumption 2: We can once at factory?
  - Run a known anchor program anc on device
  - Deliver a shared secret r securely
  - Compute $k_s = kdf(\texttt{"c1"}, r, imid)$

MITRE

Run Anchor initially

At start in safe environment

- Run "anchor" program anc on device, that does:
  - Receive $\langle imid, sh, dh, r, n_0 \rangle$
  - Warn unless (i) $imid$ is mine, (ii) $sh = ch(anc)$
  - Let

$$k_s = kdf(\texttt{"c1"}, r, imid)$$

  - Execute `prot-for` k_s, dh
  - Send confirmation n_0
- Hence:

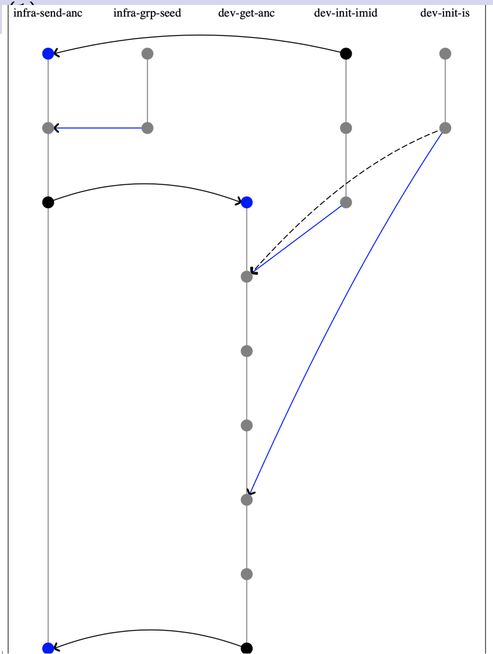
if any service svc gets k_s on device
then $dh = ch(svc)$

Mgt chooses one program to use k_s

MITRE

Where anchor key k_s goes to

Protocol analysis with CPSA shows (1)



Run Anchor initially

At start in safe environment

- Run "anchor" program anc on device, that does:
  - Receive $\langle imid, sh, dh, r, n_0\rangle$
  - Warn unless (i) $imid$ is mine, (ii) $sh = ch(anc)$
  - Let

$$k_s = kdf(\texttt{"c1"}, r, imid)$$

  - Execute `prot-for` k_s, dh
  - Send confirmation n_0
- Hence:

if any service svc gets k_s on device
then $dh = ch(svc)$

Mgt chooses one program to use k_s

- But: What should that program do with k_s?

Distributor program dtr

Use k_s to derive new per-service keys

- Distributor dtr, with $ch(dtr) = dtrh$, when run:
  - Retrieves k_s from anchor
  - Receives msg of form

$$\{\!|\, imid, (h, dtrh, ch(anc)), \ldots \,|\!\}_{k_s}$$

  - Sets $k_h = kdf(\texttt{"c2"}, k_s, h)$
  - Protects k_h for h
  - Exits, forgetting k_h
- For every svc:

  $k_{ch(svc)}$ is a shared secret between infrastructure with k_s and svc on device d

Distributor program dtr

Use k_s to derive new per-service keys

- Distributor dtr, with $ch(dtr) = dtrh$, when run:
  - Retrieves k_s from anchor
  - Receives msg of form

$$\{\!|\, imid, (h, dtrh, ch(anc)), \ldots \,|\!\}_{k_s}$$

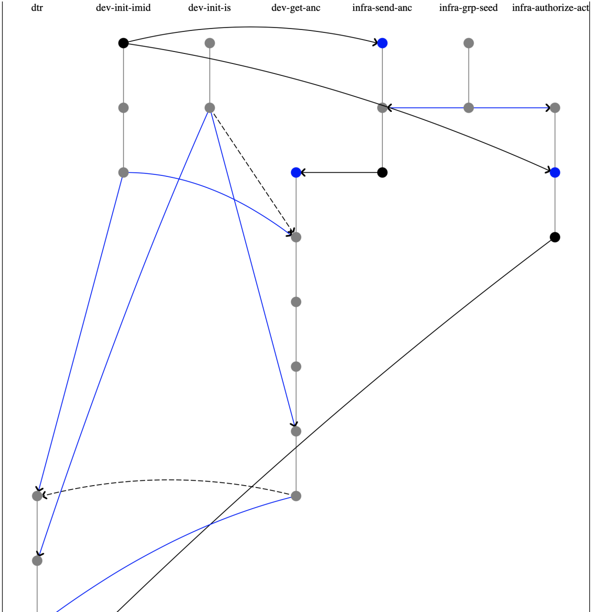
  - Sets $k_h = kdf(\texttt{"c2"}, k_s, h)$
  - Protects k_h for h
  - Exits, forgetting k_h
- For every svc:

  $k_{ch(svc)}$ is a shared secret between infrastructure with k_s and svc on device d

$(h, dtrh, ch(anc))$ is a trust chain

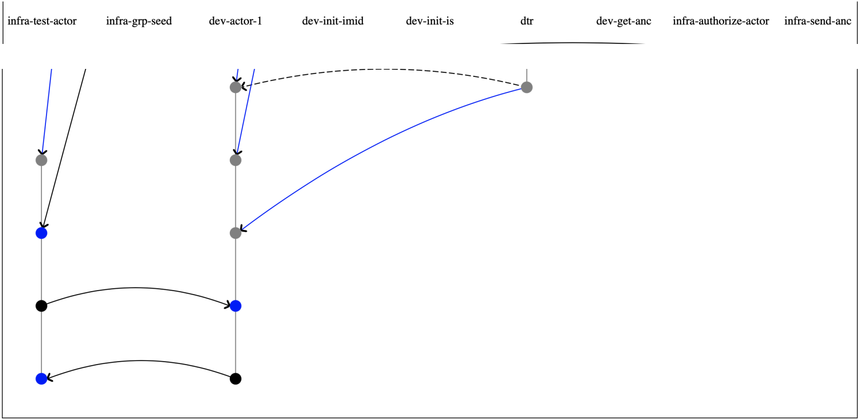
Where distributor command comes from

Protocol analysis with CPSA shows (2):



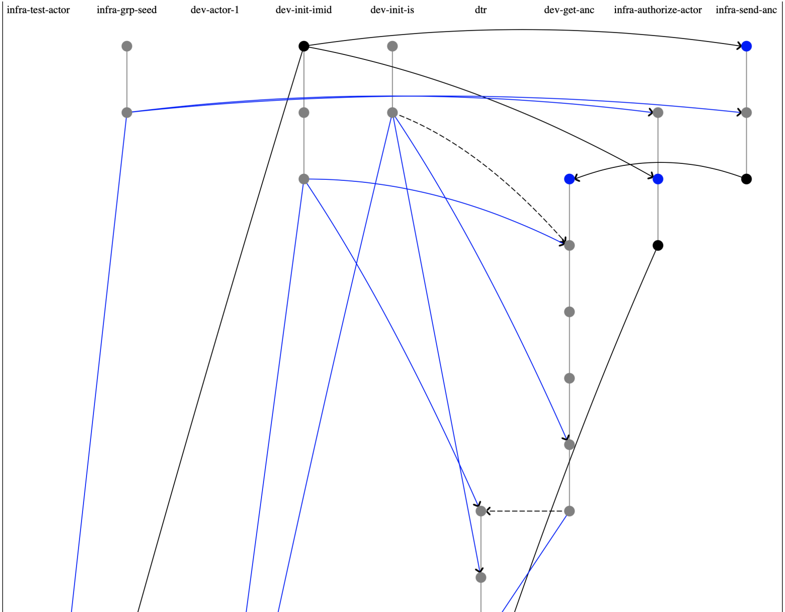
Testing an action keyed by distributor

Protocol analysis with CPSA shows (3):



Testing an action keyed by distributor (supplement)

Protocol analysis shows with CPSA (3):



Setting up trustworthy digital signatures

Distributor passes secret K_0 to service $SigGen$

- Setup phase:
  - Generate signature key pair (sk, vk) — Protect sk for myself
  - Prove possession of sk and K_0

$$\{\!|\ [\![\ldots,\quad ch(SigGen),\quad vk,\quad \ldots]\!]_{sk}\ |\!\}_{K_0}$$

  - Receive cert associating $ch(SigGen)$ and vk on $imid$

$$[\![\ldots,\quad imid,\quad ch(SigGen),\quad vk,\quad \ldots]\!]_{CA}$$

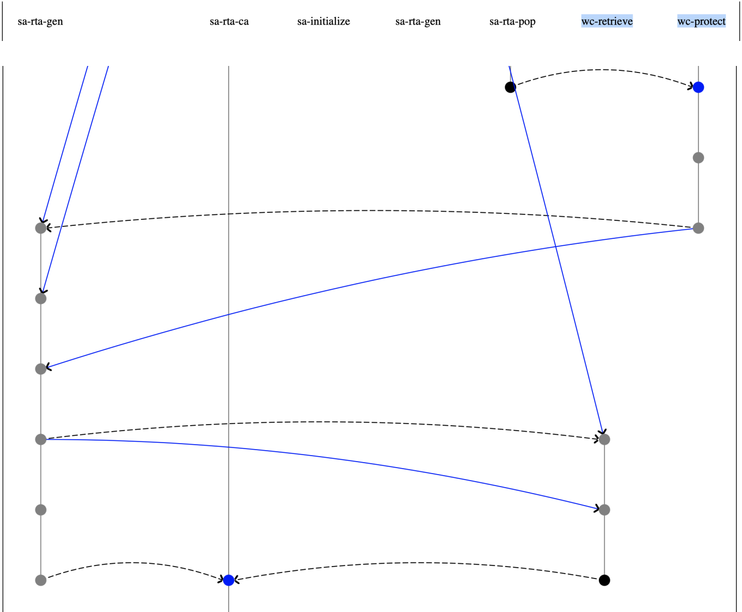
- Usage phase, for target service t:
  - Generate signature key pair tsk, tvk
  - Protect tsk for $ch(t)$ — Retrieve sk
  - Send cert associating $ch(t)$ and tvk on $imid$

$$[\![\ldots imid,\quad ch(t),\quad tvk\ldots]\!]_{sk}$$

MITRE

How to do this wrong

Against a powerful adversary that can run code on device



Setting up trustworthy digital signatures

Distributor passes secret K_0 to service $SigGen$

- Setup phase:
  - Generate signature key pair (sk, vk) — Protect sk for myself
  - Prove possession of sk and K_0

$$\{\!|\, [\![\ldots, \quad ch(SigGen), \text{trust-ch}, \quad vk, \quad \ldots]\!]_{sk} \,|\!\}_{K_0}$$

  - Receive cert associating $ch(SigGen)$ and vk on $imid$

$$[\![\ldots, \quad imid, \quad ch(SigGen), \text{trust-ch}, \quad vk, \quad \ldots]\!]_{CA}$$

- Usage phase, for target service t:
  - Generate signature key pair tsk, tvk
  - Protect tsk for $ch(t)$ — Retrieve sk
  - Send cert associating $ch(t)$ and tvk on $imid$

$$[\![\ldots imid, \quad ch(t), \text{trust-ch}, \quad tvk \ldots]\!]_{sk}$$

MITRE

Adversary model, 1: Wildcat protect

```
(defrole wildcat-protect
    ...
    (trace
        (load lis (is-entry d is))
        (recv val)
        ...
        (stor loc (cat d (prot-for val (mem-key is srch dsth))))))
```

Wildcat-protect instances are subject to an axiom:

Axiom

If an instance of wildcat-protect uses a srch

Then, for that srch, not(compliant(srch))

Adversary model, 2: Wildcat retrieve

```
(defrole wildcat-retrieve
    ...
    (trace
        (load lis (is-entry d is))
        (load loc (cat d (prot-for val (mem-key is srch dsth))))
        (send val)))
```

Wildcat-retrieve instances are subject to an axiom:

Axiom

If *an instance of wildcat-retrieve uses a dsth*

Then, *for that dsth, not(compliant(dsth))*

Subliminal message

Security protocol analysis
can help solve problems
you may not think of as
security protocols

Core questions

For a CAIF device d

Local to d: Can a service svc on d determine local service

1. src as source of data value v
2. dst as sole destination of data value v

Yes, by construction of protect-for / retrieve-from

Remote from d: Can principal not on d determine a service svc on d as the

1. source of incoming value v
2. sole destination of outgoing value v

"Assured remote execution" by svc on d

More challenging: Requires device-rooted protocol analysis

Protocol analysis enables answers,
about devices facing a powerful adversary

MITRE