Lecture Outline

• Announcements:
  – Homework for next week out by this evening
  – Guest lecture a week from Friday
    • Bill Marczak on *Abusive Surveillance*

• Today: broader notions relating to authentication
  – Architecting to resist subverted clients
  – Imprinting
  – Multi-party identities (Ecommerce, web advertising)
  – Bot-or-Not (CAPTCHAs)
Architecting to Resist Subverted Clients
Threats?

- Sniffing, MITM (network; app-level relay) ⇒ Theft of password and/or authenticator
- 3rd-party manipulation of automation
  - E.g. CSRF (browser fetching of images)
  - E.g. XSS (browser execution of JS replies)
- Password security
  - Blind guessing / bruteforcing
  - Reuse (breaches)
  - Phishing
- Compromised client: hijacking
Tackling *Transaction Generators*?

• How about using a separate system?
  – Very inconvenient

• Desired properties:
  – Compatible w/ existing legacy OS’s
  – Can run general web applications
  – No need to trust host OS
  – Small TCB: attestable via TPM
Trusted hypervisor mediates all physical I/O events
Cloud Terminal Architecture

Only hypervisor has access to user’s credentials: the user doesn’t know their own passwords
User signals Hypervisor to begin a secure session using a specific hardware keystroke combination ("Secure Attention Key")
Cloud Terminal Architecture

Upon this request, a “thin client” UI runs *in the hypervisor context* (= trusted)
Cloud Terminal Architecture

It’s locked down to only talk to services previously registered with the Hypervisor – user can’t be phished
After selecting which service to interact with (e.g., user’s bank), Hypervisor interacts using untrusted host OS for networking & storage.
Cloud Terminal Architecture

Doing so reduces TCB
End-to-end encryption obviates need to trust host OS; only threat is DoS
Cloud Terminal Architecture

Hypervisor ensures authenticity of remote service by (correctly) validating TLS certificate
Cloud Terminal Architecture

Hypervisor authenticates user to service using password in key store
Cloud Terminal Architecture

Remote service (e.g., user’s bank) renders interaction UI *locally* using kiosk software
UI is presented to user using VNC-style low-level frame buffer protocol
Cloud Terminal Architecture

Rendered directly into video hardware by RUI client + hypervisor
Cloud Terminal Architecture

Malware has no capability to observe what’s displayed
User interacts using physical hardware events mediated by hypervisor directly to RUI client.
Cloud Terminal Architecture

Malware has no opportunity to influence interaction
Tackling *Transaction Generators*

• Desired properties:
  – Compatible w/ existing legacy OS’s
    • They run in VMs controlled by hypervisor
  – Can run general web applications
    • Anything that remote site can render into VNC-style framebuffer protocol
  – No need to trust host OS
    • Hypervisor provides strong isolation
  – Small TCB: attestable via TPM
    • Working implementation: 22 KLOC
Architectural Elements?

• Abstractions:
  – Interactions via “dumb” separate terminal

• Placement of functionality:
  – Move rendering into controlled environment
  – Add trusted hypervisor w/ local secrets/smarts
  – Require interactions to come from physical hardware
  – Use E2E principle to leverage untrustworthy code

• State:
  – Isolated to trusted component

• Naming:
  – Use existing PKI system + TPM
Imprinting
Device Authentication: IOT

For IOT device + home controller, want secure, impermanent associations.

Impermanent: so you can sell your device but a thief cannot.
Resurrecting Duckling

Imprinting on Mother:
  Device shares key on 1st contact with controller

Metempsychosis:
  Upon death, soul progresses to a new body

Reverse metempsychosis:
  Upon death, new soul can enter the body

Resistance to assassination:
  Only mother can kill her ducklings

Escrowed seppuku:
  Manufacturer can kill too

⇒ Thief can’t “kill” device
  ⇒ no utility for anyone who buys it from them
1. **Two state principal** – the duckling can be in one of 2 states; imprintable and imprinted. In the imprintable state, anyone can claim to be the duckling’s mother. In the imprinted state, the duckling only obeys its mother.

2. **Imprinting principal** – imprinting happens when the mother duck sends an imprinting key to the duckling. This must be done over a channel that protects confidentiality and integrity of the key.

3. **Death principal** – the transition from imprinted to imprintable can only occur in specific circumstances defined by the model:
   - death by order of the mother duck (default)
   - death after a predetermined interval
   - death after the completion of a specific transaction

4. **Assassination principal** – The duckling must be constructed in a way that it is unfeasible for an attacker to force it into the imprintable state by means other than those stated in the death principal.

Imprinting in Other Contexts

• What is SSH’s PKI model?
  – It doesn’t have one: Leap-of-Faith

• Pros:
  – Ease of deployment

• Cons:
  – Security properties require users to *non-satisfice*

The authenticity of host 'diablo.icir.org (192.150.187.59)'
can't be established.
ECDSA key fingerprint is SHA256:uvJWTjlM5c74D5gp62GMeCk2ccBILukf91za1S2zl8k.
Are you sure you want to continue connecting (yes/no)?

  – No revocation model
  – Disaster if attacker gets there first
Persistent Ungrounded Identity

• Idea: systems generate (unanchored!) public key and consistently include it w/ (signed) messages
  – Provides recipient a lever for “this is the same entity I talked with previously” …
  – … even though actual identity (“persona”) not known

“Assurance through continuity”
Persistent Ungrounded Identity

• Idea: systems generate (unanchored!) public key and consistently include it w/ (signed) messages
  – Provides recipient a lever for “this is the same entity I talked with previously” …
  – … even though actual identity (“persona”) not known

• E.g.: consistently sign your email/texts
  – Recipient can associate reputation w/ each persona, use them for whitelisting
  – User can migrate persona to additional systems

• E.g.: use for SBGP instead of a PKI
  – Game theory result: deployment gains a network effect
Persistent Ungrounded Identity

- Idea: systems generate (unanchored!) public key and consistently include it w/ (signed) messages
  - Provides recipient a lever for “this is the same entity I talked with previously”
  - ... even though actual identity (“persona”) not known
- E.g.: consistently sign your email/texts
  - Recipient can associate reputation w/ each persona, use them for whitelisting
  - User can migrate persona to additional systems
- E.g.: use for SBGP instead of a PKI
  - Game theory result: deployment gains a network effect

Issues?

Key compromise is a disaster
No apparent handle for revocation
Multi-Party Identities
Cashier-as-a-Service (CAAS)

Abstract Ecommerce workflow:

1. Shopper surfs Merchant’s site
2. Shopper sends over .../place_order.html
3. Merchant sends back redir. to CAAS.com
4. Shopper interacts with CAAS
5. CAAS interacts with Merchant
6. CAAS redirects shopper back to Merchant
CAAS Attack #1?

S→M: place_order.html
   [M inserts ID and price into database; status=PENDING]

M→S→C: get_payment?orderID=X&price=Y
   [C records payment info, generates transaction # T]

C→S→M: finish?transID=T
   [M contacts C for identifier X associated w/ T]
   [M retrieves orderID=X from database;
    if order status = PENDING → mark as PAID; ship X]

Note: we view Merchant and Cashier as trustworthy. The Shopper, OTOH …
CAAS Attack #1!

S→M: place_order.html

[M inserts ID and price into database; status=PENDING]

M→S→C: get_payment?orderID=X&price=Y

[C records payment info, generates transaction # T]

C→S→M: finish?transID=T

[M contacts C for identifier X associated w/ T]

[M retrieves orderID=X from database;
   if order status = PENDING → mark as PAID; ship X]
CAAS Scheme #2?

S→M:  place_order.html
   [M inserts ID and price into database; status=\textit{PENDING}]

M→S→C:  get_payment?orderID=X&price=Y
   [C records payment info, generates transaction # \textit{T}]

C→S→M:  finish?transID=T
   [M contacts C for identifier X associated w/ \textit{T}]
   [M retrieves orderID=X from database;
    if order status = \textit{PENDING} → mark as \textit{PAID}; ship X]
CAAS Attack #2?

S→M: place_order.html
   \[ M \text{ inserts ID and price into database; status}=\text{PENDING} \]

M→S→C: get_payment?SIGN_M(ID=X, price=Y)
   \[ C \text{ verifies signature; records payment info, generates \# } T \]

C→S→M: finish?SIGN_C(ID=X, price=Y, PAID)
   \[ M \text{ verifies signature and \text{PAID} is indicated} \]
   \[ M \text{ retrieves orderID}=X \text{ from database; if order status } = \text{PENDING} \rightarrow \text{mark as } \text{PAID}; \text{ ship } X \]
S→M: place_order.html
   [M inserts ID and price into database; status=PENDING]

M→S→C: get_payment?SIGN_M'(ID=X,price=Y)
   [C verifies signature; records payment info, generates # T]

C→S→M: finish?SIGN_C(ID=X,price=Y,PAID)
   [M verifies signature and PAID is indicated]
   [M retrieves orderID=X from database;
    Shopper colludes with another merchant M' to get a signature on same identifier X for price Y … without having to ultimately pay]
Fix for CAAS Attack #2

S→M:  place_order.html  
       [M inserts ID and price into database; status=PENDING]

M→S→C:  get_payment?  
       \[ \text{SIGN}_M(\text{ID}=X, \text{price}=Y, \text{merch}=M) \]  
       [C verifies signature; records payment info, generates # T]

C→S→M:  finish?  
       \[ \text{SIGN}_C(\text{ID}=X, \text{price}=Y, \text{merch}=M, \text{PAID}) \]  
       [M verifies signature and PAID is indicated, etc.]  
       [M retrieves orderID=X from database;  
         if order status = PENDING → mark as PAID; ship X]
Better Fix for CAAS Attack #2

S→M: place_order.html
   [M inserts ID and price into database]

M→S→C: get_payment?
   \[\text{SIGN}_M(\text{ID}=X, \text{price}=Y, \text{merch}=M, \text{shop}=S)\]
   [C verifies signature; records payment info, generates # T]

C→S→M: finish?
   \[\text{SIGN}_C(\text{ID}=X, \text{price}=Y, \text{merch}=M, \text{shop}=S, \text{PAID})\]
   [M verifies signature and \text{PAID} is indicated, etc.]
   [M retrieves orderID=X from database;
    if order status = \text{PENDING} \rightarrow \text{mark as PAID}; ship X]