Cloaking Malware with the Trusted Platform Module

Alan Dunn, Owen Hofmann,
Brent Waters, Emmett Witchel
University of Texas at Austin

USENIX Security
August 12, 2011

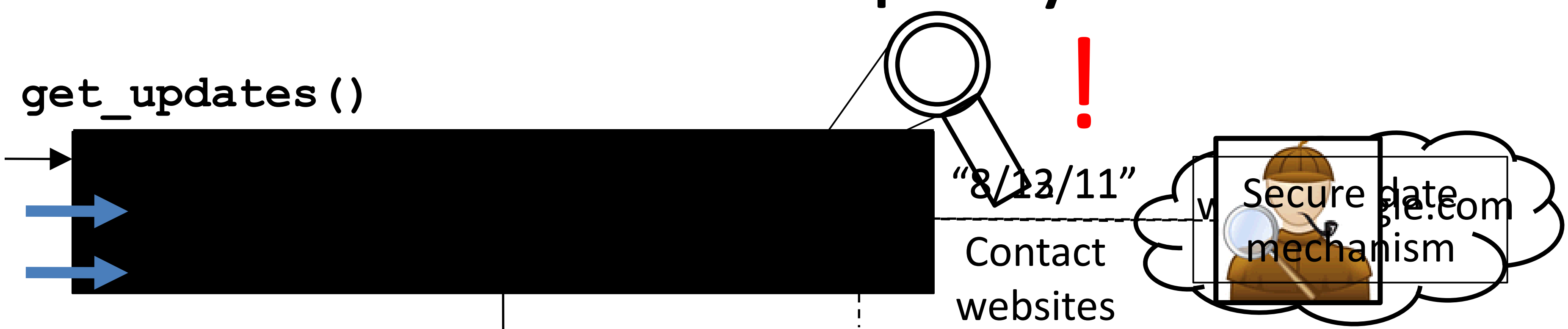
Trusted Computing

- Goal: Secure environment for computation
- Trust rooted in hardware
- Most familiar: Trusted Platform Module (TPM)
  - Standard by Trusted Computing Group (TCG)
  - IC in x86 machines connected to southbridge
  - Widely deployed (> 350 million TPMs)

Uses of Trusted Computing

- Typical: TPM provides hardware root of trust
  - Store cryptographic hash of executed software
  - Perform cryptography, store secret keys
  - Provide hardware-protected execution environment
- Ours: TPM provides hardware cloak for malware
  - Only run unmodified malware
  - Store malware secret keys
  - No monitoring/debuggers/virtualization

Conficker B Analysis



TPM can help malware writers achieve this goal: Execute computation securely in non-analyzable environment

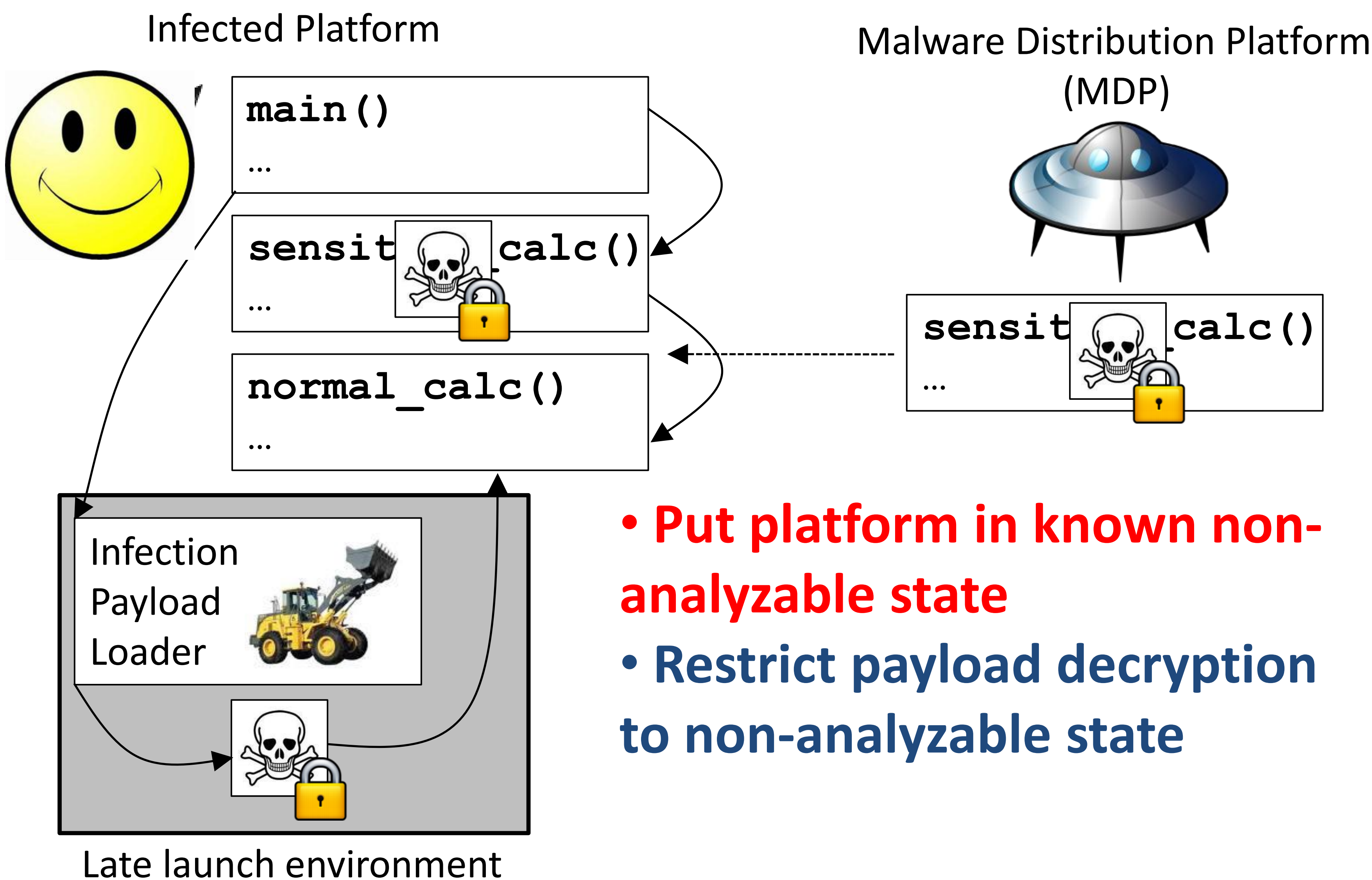
```
for domain in domains:
  content = fetch_content(domains)
  if (check_sig(content))
    apply_update(content)
```

Goal for malware writers: Secure and hidden malware sub-computation

Outline

- Protocol Overview
- Protocol
- Implementation
- Defenses

Protocol Overview

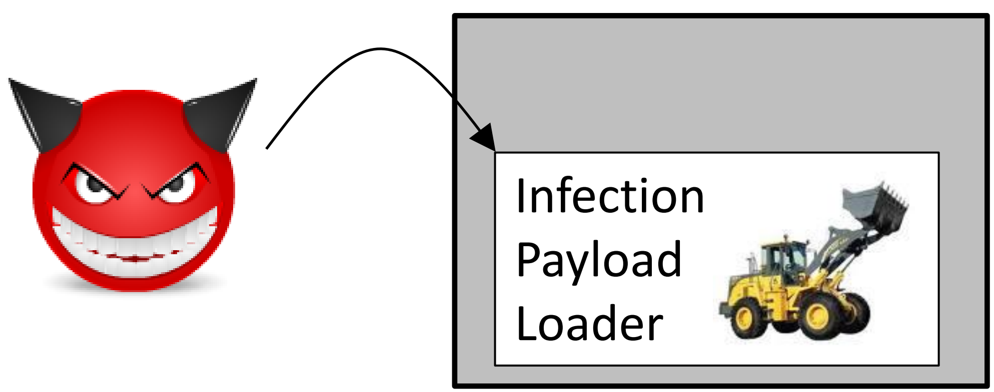


- **Put platform in known non-analyzable state**
- **Restrict payload decryption to non-analyzable state**

Put platform in non-analyzable state

- Suspend all system software, jump into known software state
- *Late launch* performs jump, records program jumped to via hash

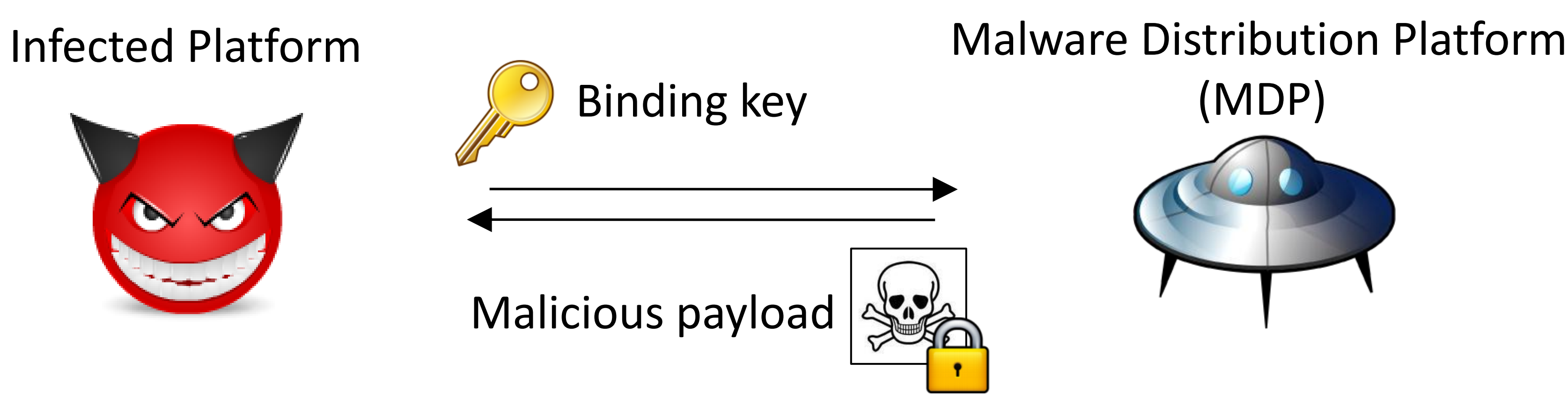
Infected Platform



Late launch environment

Restricting payload decryption

- TPM *controls* private key use for keypairs it generates
- Binding key *constrained* to use in non-analyzable state
- Certificates show Endorsement Key (EK) belongs to legitimate TPM
- Remote attestation proves binding key generated by same party as EK, so payload only decryptable in late launch

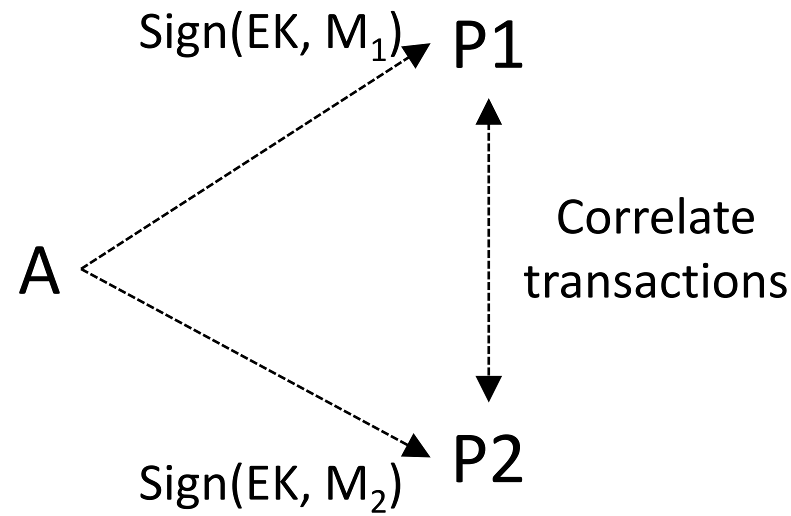


Late Launch

- `SENTER` instruction transfers control to binary, sets TPM register based upon cryptographic hash of binary
  - Allows binary to execute securely: stop other cores, turn off interrupts
- For malware:
  
  - Transfer control to Infection Payload Loader (IPL)
  - IPL hash satisfies key use constraint
  - IPL decrypts, transfers control to malicious payload

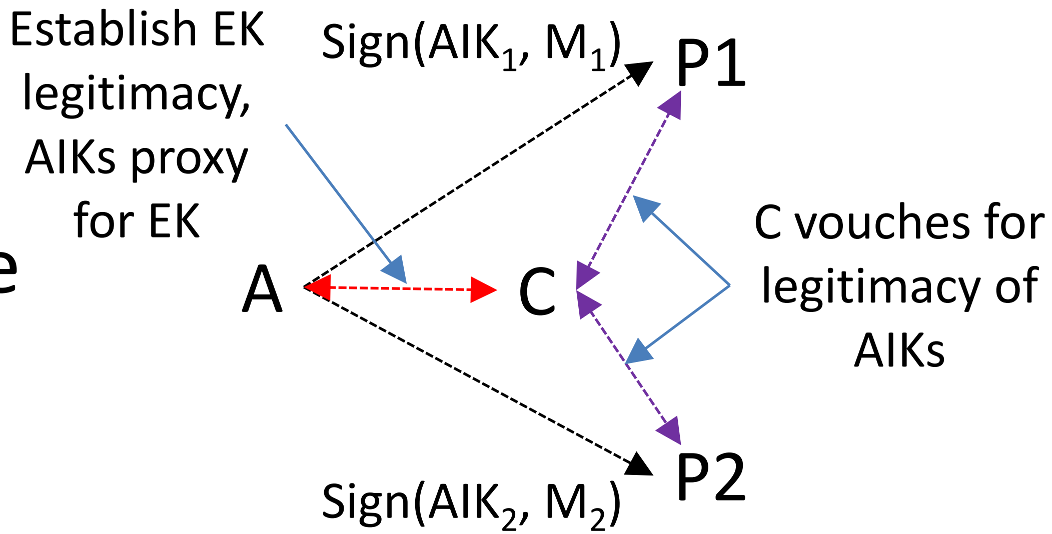
Validating the Binding Key

- Endorsement Key (EK) – unique identifying key, certified by TPM manufacturer
- Sign binding key with EK? Forbidden!
- EK identifying, compromises anonymity



TPM Identity (EK) with Indirection (AIK)

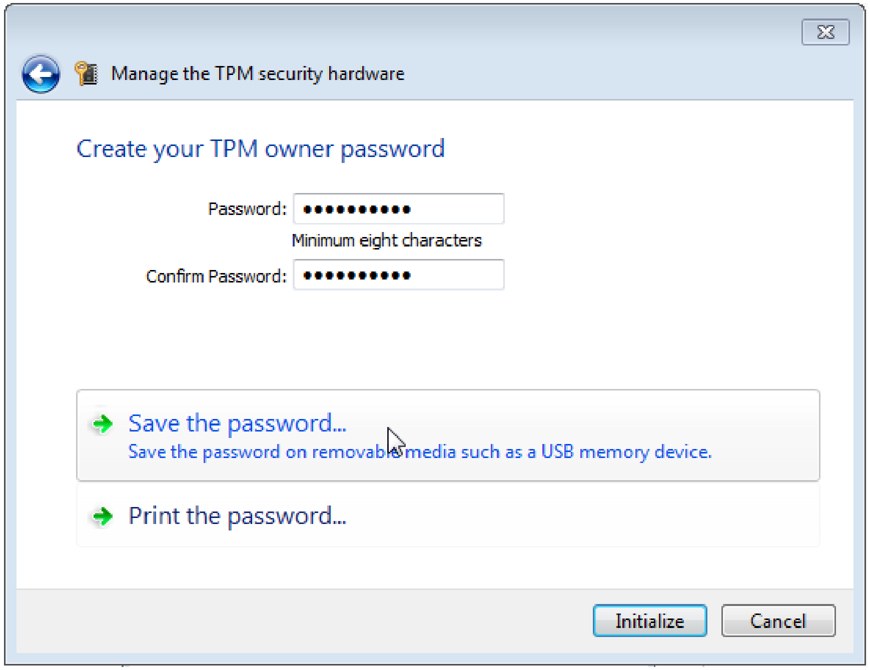
- Attestation Identity Keys (AIKs) fix anonymity
- *Privacy CA* vouches that AIK represents EK
- Problem: Privacy CAs don't exist
- Solution: Malware Distribution Platform acts as Privacy CA



C is a Privacy CA

Can malware generate an AIK?

- Owner AuthData required for AIK generation
- Owner AuthData not needed on platform, used rarely
- Capture from keylogging or from memory (Windows: cached for days)



Remote attestation details

Infected Platform

Malware Distribution Platform (MDP)

Phase 1: cred → AIK represents EK

1) Generate AIK

2) PK_{EK}, PK_{AIK}, Sign($SK_{manuf.}$,H(PK_{EK}))

3) Verify EK sig

4) Enc(PK_{EK}, cred || H(PK_{AIK}))

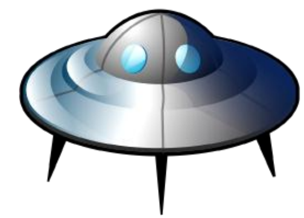
5) Activate AIK: if H(PK_{AIK}) matches AIK generated on that platform, TPM releases cred

Remote attestation details (cont'd)

Infected Platform

Phase 2: Prove binding key is from TPM that controls EK

Malware Distribution Platform (MDP)



1) Generate binding key with use constraint

2) PK_{bind}, key use constraint, cred, Sign(SK_{AIK},H(PK_{bind}||key use constraint))

3) Verify use constraint, cred

4) Send encrypted malicious payload

Malicious payload

5) Late launch, decrypt and run payload

Implementation

- Protocol until late launch (w/TrouSerS)
- Late launch (via Flicker v0.2) on Intel platforms
  - Infection Payload Loader (IPL): decrypt, execute payload
  - IPL run appears as 3 second system freeze on Infected Platform due to TPM key operations in late launch
- Three malicious payloads
  - Conficker B-like example
    - Secure time via Ubuntu package manifests
  - DDoS timebomb
  - Secret text search

Defense: Whitelisting late launch binaries

- Hypervisor-level whitelisting
  - Trap on `SENTER`, check late launch binary
    - List of hashes of whitelisted binaries
    - Digitally sign binaries, whitelist signing keys
- Problems
  - Requires hypervisor: tough for home users
  - Late launch binary updates
  - Signatures: Revocation, trust management (certificate chains)

Defense: Manufacturer Cooperation

- Manufacturer breaks TPM guarantees for analyst
- Fake Endorsement Key (EK)
  - Manufacturer produces certificate for EK that is not TPM controlled
  - Problem: EK leak can compromise TPM security properties
- Fake Attestation Identity Key (AIK)
  - Manufacturer uses EK to complete AIK activation for AIK that is not TPM controlled
  - Problem: AIK requests need manufacturer response online

Defense: Physical Compromises

- TPM compromise has been demonstrated
  - Simple: Grounding LPC bus allowed faking of TPM code measurement
  - Exotic: Etching away casing, probing around tamper-resistant wiring allowed EK recovery
- Industry incentives to fix
- Further discussion in paper (e.g. cold boot)

Conclusion

- TPM can cloak malware sub-computations, hiding them from analysts
- Concrete implementation of TPM-based malware cloaking
  - Remote attestation
  - Late launch
- Strengthening TPM guarantees makes attack more resilient