Unicorn: Two-Factor Attestation for Data Security

M. Mannan
Concordia University, Canada

B. Kim, A. Ganjali & D. Lie
University of Toronto, Canada
Unicorn – target systems

- **High value** data
  - overkill for casual Facebook profile

- **Requires:**
  - hardware support
  - PSD: personal security device
  - custom application package
Goal – malware

- Protect entire user session

- Not limited to login & other credentials
Goal – “phishing”

- Protect Unicorn credentials
- Less reliance on user diligence during authentication

www.bank.com

wwwbank.com
Goal – “automatic” security

- **Unicorn**: access to data depends on system state
- **Ideal**: functionality tied to system health – avoids the “unmotivated user” problem
Dealing with Phishing

- Use security token
  - generate one time passwords or do challenge response
  - adversary must steal/clone a token
  - or: hack RSA 😊

But no protection against malware!
Dealing with malware

- Use trusted computing hardware to verify the integrity of a computer (TPM, Intel TXT, AMD SVM)
  - compute hashes of BIOS, boot loader, OS, ...
  - sign hashes to prove that computer is not running unwanted code

Notifying the current system state to users isn’t easy

[Libonati, NDSS 2011]
Combine the two?

- **Unicorn:** security token + trusted computing
  - Unicorn applications are called uApps
  - uApp: small OS + one application
  - User OS: **open**, uApps: **closed**
Unicorn design

- Token is used to verify attestations
  - only releases authentication credentials if system is malware free
  - verifies: system memory + disk image

- Attacker must
  - clone or steal the token
  - physical access to user’s computer
  - assumption: no vulnerabilities in Unicorn
Unicorn example – setup

- Security tokens currently not capable of attestation
- Smartphone as PSD

User diligence is needed
Unicorn example — steps

Verify
Starting a uApp

- Suspend, save user OS state
  - option 1: reboot into uApp
  - option 2: use DRTM
    - User OS as the bootloader

- Resume:
  - direct transfer back to user OS
uApp usage scenarios

- Two example cases
  - data stored on a remote server
  - data stored encrypted on local machine

- Token is initialized with a secret key
  - remote server case: signing/MAC key
  - local machine: encryption sub-key
Remote server case

- **(MAC key)**
- **(TPM keys)**
- **Nonce, PCRs**
- **Quote(Nonce, PCRs)**
- **MAC(Quote(Nonce, PCRs))**
Local machine case

- Encryption key consists of two subkeys:
  - subkey 1: on smartphone
  - subkey 2: sealed into computer’s TPM
Unicorn prototype

- Extensions to Linux 2.6.34 kernel
  - one line change in user OS kernel

- Transfer directly to uApp OS via kexec
  - loads uApp kernel image into memory
  - saves state & transfer control to uApp loader

- uApp loader based on tboot package
  - sets up a measured launch environment (MLE)
  - measures the loaded kernel
  - kernel measures the disk image during boot
**uApp images**

- **uApp image should be small**
  - smaller attack surface & faster verification
  - we use a small Linux distro with a space efficient file system (squashfs)

- **uApp is network restricted**
  - communicate only with remote server
  - help against TPM relaying attacks
  - SSL cert: only one trusted CA
TPM relaying

- Attacker relays attestation requests to legitimate machine
  - but legitimate uApp will only communicate with remote server
Switching between commodity OS & uApp

- Majority of cost: suspending the user OS and kernel boot
- uApp loader is slow because TPM is slow

<table>
<thead>
<tr>
<th>Task</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suspend of user OS</td>
<td>11.16</td>
</tr>
<tr>
<td>uApp Loader</td>
<td>3.29</td>
</tr>
<tr>
<td>Kernel boot and Xserver startup</td>
<td>7.20</td>
</tr>
<tr>
<td>OS hash</td>
<td>3.85</td>
</tr>
<tr>
<td>Unicorn Total</td>
<td>25.50</td>
</tr>
<tr>
<td>Switching via reboot</td>
<td>47.70</td>
</tr>
</tbody>
</table>
Conclusion

- Unicorn for **two-factor data protection**
  - physical access to user PC & compromised token
  - for users: no passwords to remember or judge the safety of their computing environment
  - switching: skip hardware reset and BIOS
    - ~45% reduction in switching time vs. previous methods

- Enable “automatic” security
Questions, comments

Thank you!

mmannan@ciise.concordia.ca