STARK
TAMPERPROOF AUTHENTICATION TO RESIST KEYLOGGING
Chapter 1

Threat Scenario
Physical Access Threats

• 2008: “Airport Insecurity: The case of Missing Laptops” (Ponemon)
  – 12,255 laptops are lost at U.S. airports per week
  – 53% of traveling salesmen state to carry along sensitive data

• 2011: “The Billion Euro Lost Laptop Problem” (Ponemon)
  – 275 organizations in Europe
  – 8% of all company laptops are lost during their lifetime
    (in the 12-month study, 72,789 laptops were lost)

• 2012: “2011 Annual Study: U.S. Cost of a Data Breach” (Ponemon)
  – cost per data record: 194 USD
  – average cost per stolen laptop: ~45,000 USD
Solution: Disk Encryption

• protects data against *physical* loss and theft
• disk remains unreadable until a user enters the correct passphrase

• examples: BitLocker, FileVault, TrueCrypt, ...
  (most configurations based on AES)
Full Disk Encryption

• FDE = *full* disk encryption
• supported mode by TrueCrypt, BitLocker, ...
• encrypts a whole disk including the OS

• the whole disk?
• no!
  for *bootstrapping* reasons, at least a small part of the disk must be present unencrypted
Preboot Environments

- placed inside the master boot record (MBR)
- *uniform* (often text-based) password prompts

- problem: these prompts can easily be forged!
Evil Maid Attacks

• coined by Rutkowska, 2009: *Evil Maid goes after TrueCrypt*

• basically a keylogger placed inside the MBR

• a.k.a. *bootkit*

• scenario:
  1. salesman leaves hotel room
  2. an *evil maid* manipulates the MBR
  3. salesman enters PW and leaves again
  4. the evil maid reads out the logged PW
Consequences

- today's FDE does not protect against system subversion, but only against loss and theft
- TrueCrypt says that bootkits
  “require the attacker to have [...] physical access to the computer, and the attacker needs you to use the computer after such an access. However, if any of these conditions is met, it is actually impossible to secure the computer”
- so it's a matter of opinion if FDE should or should not protect against evil maid attacks
- in our opinion, it should!
BitLocker and the TPM

• BitLocker supports the Trusted Platform Module (TPM)
• TPM is used to “unseal” the data encryption key
• if MBR is manipulated, the correct key cannot be unsealed and the data cannot be decrypted

Windows BitLocker Drive Encryption Recovery Key Entry

Enter the recovery key for this drive.

___ ___ ___ ___ ___

Drive Label: TEST-LAB C: 21/12/2010
Recovery Key ID: 223039EA-B480-4CB6-A2A0-76FAF6DF423C

Use the function keys F1 - F9 for the digits 1 - 9. Use the F10 key for 0.
Use the TAB, SHIFT-TAB, HOME, END and ARROW keys to move the cursor.
The UP and DOWN ARROW keys may be used to modify already entered digits.
Tamper-and-Revert Attacks

- Türpe et al., 2009: “Attacking the BitLocker Boot Process”

- Tamper-and-Revert:
  1. *tamper* with the bootloader to introduce keylogging
  2. victim enters PW into forged prompt
  3. *revert* to the original bootloader
  4. reboot

- Most likely, the user enters PW again and proceeds working as usual
Chapter 2

STARK Authentication
Mutually Authentication

- STARK Tamperproof Authentication to Resist Keylogging
- idea: mutually authenticate users and PCs
  1. *PC proves to user that it is not manipulated*
  2. only then, user enters password
- how can the PC prove its *integrity*?
Personalized prompts

• only if the PC is not manipulated, a user-defined message can be unsealed (TPM)
• this message must be shown to the user before he enters the password

![TrueCrypt Boot Loader 6.3a](image)

Keyboard Controls:
[Esc] Skip Authentication (Boot Manager)

This is a secret authentication message

Enter password: _
Problems

• an evil maid can boot the machine, write down the unsealed message, and build a forged MBR

• Solutions?

PIN for the TPM (i.e., “pwd > msg > pwd”)?
No (two keylogging attacks)

Auth. message on USB drive?
No (MBR can clone the USB drive)
STARK

- mutual authentication in FDE by
  - trust bootstrapping (from USB drive)
  - one-time prompts (by so-called monces)

- “monce” = message used once
  (because humans cannot remember nonces)

- basic authentication scheme:
  1. PC auth. towards user by unsealing a monce (TPM)
  2. user auth. towards PC by a traditional password
  3. user updates the old monce with a new one
**STARK: Bootstrapping Phase**

<table>
<thead>
<tr>
<th>USER</th>
<th>COMPUTER</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>construct p and m₀</em></td>
<td><em>p, m₀</em></td>
</tr>
</tbody>
</table>
| | *construct t*
| | *save hash(p,t)*
| | *seal (m₀,t)*
| *sealedₜ(m₀,t)* | |
| *save sealedₜ(m₀,t) to USB* | |
### STARK: Authentication Sessions

<table>
<thead>
<tr>
<th>USER</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>plug in the USB drive</td>
<td>sealed(<em>c(m</em>{i-1},t))</td>
</tr>
<tr>
<td>authenticate the computer and only then enter (p)</td>
<td>(m_{i-1})</td>
</tr>
<tr>
<td>construct (m_i)</td>
<td></td>
</tr>
<tr>
<td>save &amp; unplug USB drive</td>
<td>sealed(_c(m_i,t))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COMPUTER</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sealed (m_{i-1},t)</td>
<td>unseal (m_{i-1},t)</td>
</tr>
<tr>
<td>(p)</td>
<td>calc hash((p,t)) to authenticate the user</td>
</tr>
<tr>
<td>ACK</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>seal ((m_i,t))</td>
</tr>
</tbody>
</table>
STARK Overview

1. is this your monce?  
   <a secret message>

2. please enter password:  
   > ********
   compromised

3. enter new monce:  
   <another secret msg>

   $ k = \text{pbkdf2}(\text{HMAC-SHA-256}, p, 4096, 256)$

   $ d = \text{decrypt}(\text{AES-256}, k, D)$

   decrypt. boot into OS...
STARK Characteristics

- **one-time boot prompts**
  - each auth. message is valid for only one boot

- **trust bootstrapping**
  - USB drive must be handled like a physical key

- additionally: **two-factor authentication**
  - USB drive is bounded to decryption process

►► evil maids catch outdated monces only!
Chapter 3

POTTS Implementation
POTTS

- Linux-based Implementation of STARK
- POTTS: *Prevents Opportunistic and Targeted Threat Scenarios*
  - *targeted*: system manipulation (by STARK)
  - *opportunistic*: physical loss and theft (by TRESOR)
TRESOR

• Academic disk encryption solution

• T. Müller, F. Freiling, A. Dewald (USENIX 2010)
  TRESOR Runs Encryption Securely Outside RAM

• Prevents *cold boot attacks*

(BitLocker, TrueCrypt, etc. are all vulnerable to cold boot attacks)
Setup Phase

POTTS - a STARK implementation

Choose

- **F1**  Boot Encrypted System
- **F2**  Setup
- **F3**  Change Container Path
- **F10** Boot Live System
Setup Phase
Setup Phase

POTTS - a STARK implementation

- get config: done
- set new passphrase: done

Enter a new nonce:

Secret message:

- generate or set DEK: (pending)
- encrypt / store DEK: (pending)
- test: (pending)
Setup Phase

POTTS – a STARK implementation

- get config: done
- set new passphrase: done
- set monce mB: done
- generate token t: done
- seal monce and token: processing (sealing token & monce)
- generate or set DEK: (pending)
- encrypt / store DEK: (pending)
- test: (pending)
Setup Phase

POTTS - a STARK implementation

get config: done

set new passphrase: done

set Disk Encryption Key

generate a random DEK <
enter the DEK manually (for recovery)
abort (return to main menu)

generate or set DEK: processing

encrypt / store DEK: (pending)

test: (pending)
Setup Phase

```
<table>
<thead>
<tr>
<th>Disk Encryption Key</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>DEK OK?</td>
</tr>
<tr>
<td>11 11 11 11 22 22 22 33 33 33 33 44 44 44 44</td>
</tr>
<tr>
<td>aa aa aa aa bb bb bb bb cc cc cc cc dd dd dd ee</td>
</tr>
</tbody>
</table>

ATTENTION: You need this key for recovery.
Write it down and store it at a secure location.
(This is your only chance to do so!)

> Yes. <
No, enter/generate a new one.
No, abort (go to main menu).
```
Setup Phase

POTTS – a STARK implementation

get config: done
set new passphrase: done
set mode:
generate t:
seal nonce and:
generate or set DEK: done
encrypt / store DEK: done

test: done

All steps were completed successfully.

> OK <
Authentication Session (Video)
Availability

- **STARK**: http://www1.cs.fau.de/stark/
- **POTTS**: http://13.tc/potts/

**POTTS: Manual**

- **Intended Audience**
- **Installation**
  - Preparing a USB Drive
  - Testing and Configuring the TPM
  - Installing POTTS
  - Installing Arch
  - Converting an Existing Non-Encrypted Installation
- **Daily Use**
  - Changing the Passphrase
  - Kernel Upgrade
- **Recovery**
  - Unsecting Fail
  - Hardware Failure
  - Repairs After Internet's Upgrade
  - Damaged/Unmountable Boot File System

**Intended Audience**

To use POTTS and this manual you should possess a basic understanding of Linux, the boot pro:

**Installation**

Preparing a USB Drive

Insert a USB drive that can be overwritten – all data on that drive will be lost! It should have a cap.

Find out which device name was assigned. Examples:

```
> dmesg | tail
> lsblk
> lsblk --fs
```

For our examples we assume your USB drive is `/dev/sdb`.

Write the image to the USB drive. Make sure that all data has been written to the drive before you unplug it:

```
> zcat potts-usb.img.gz | dd of=/dev/sdb
> sync
> eject /dev/sdb
```

Testing and Configuring the TPM

Download: `tg4d-testing.gz` (0.3 MB)

To install this image onto a USB drive (assuming `/dev/sdb`):

```
gzip -d < tg4d-test.img | dd of=/dev/sdb
```

When you try to boot your target machine with this stick, the screen should look like this:

![Image of the screen](image)

More details...

We have prepared a USB driver image that will let you create an encrypted partition and install Arch Linux into it. After the installation it is recommended that you use a two-stage boot process. TrustedGRUB is used as boot loader on the USB drive. It measures the Kernel and initramfs as well. It is decryptable only if the system is in a trusted state. If authentication is successful, the real system (stage 2) is booted.

The main components of our package are:

- POTTS (our sources interface)
- TrustedGRUB
- TraySert, tpm-tools
- TRCA (Linux kernel patch)
- archiso (Arch Linux live image)

**Screenshots**

![Screenshot 1](image)

**Videos**

![Video](image)
Chapter 4

Limitations and Future Work
Limitations

• POTTS enables users to *identify* a system compromise, but does not regulate which actions to take afterwards

• STARK defeats only traditional evil maid attacks
  
  [x] software-based boot manipulations
  
  [ ] hardware-based attacks
Future Work

- monces are hard to generate and remember
- problem: one communication end point is human (nonces are even harder to generate and remember).

- future: use active USB drives and real nonces
Questions?