Security in Wayland-Based DEs

Privileged Clients, LibWSM and Security UIs

Steve Dodier-Lazaro

s.dodier-lazaro@cs.ucl.ac.uk

Martin Peres

martin.peres@labri.fr

mupuf.org

// we are octopi

Outline

DRM: Update since our talk @ XDC2012

Drivers: Per-process virtual address space

- Intel: WIP; Nouveau & Radeon: done

X-Server

- DRI3: Use DMA-Buf instead of GEM flink for BO-passing
- but the X protocol is still unsecure **by design...**

Wayland/Weston

- Designed with security in mind from the ground up
- now uses DMA-Buf instead of GEM-flink to provide client isolation
- relies on DRM drivers for its security

But How to Build a Secure OS?

Some critical concepts

Complete Mediation	Requires client isolation in the HW, kernel, the display server *and* sandboxing
Least Privilege	Need for mandatory security within user sessions and means to identify privileged clients
Trusted Path	Unspoofable ways for user and trusted apps to communicate; Allows reading the user's intent

All of the above needed to prevent evil apps from hurting you!

[Saltzer and Schroeder, 1975, Loscocco et al., 1998]

Challenges of creating a secure Desktop 1/2

Some common GUI requirements are un-secure by design

- Clipboard monitoring
  - Acceptable: check that data can be pasted (for GUI toolkits)
  - Unacceptable: access sensitive data
- Key events monitoring
  - Acceptable: global hotkeys
  - Unacceptable: keylogging, reading your passwords
- Input-injection
  - Acceptable: visual keyboards / accessibility
  - Unacceptable: command injection

Challenges of creating a secure Desktop 2/2

- Focus raising
  - Acceptable: show apps requiring user input before a power down
  - Unacceptable: window stealing the user's input while authenticating
- Full-screen support
  - Acceptable: video, gaming, full-screen shell
  - Unacceptable: spoofing the greeter

Solutions by the X-Server vs Wayland/Weston

- X-Server: One valid use case $\rightarrow$ access granted to everyone
- Wayland: One invalid use case $\rightarrow$ access denied to everyone

Current solution on Wayland/Weston

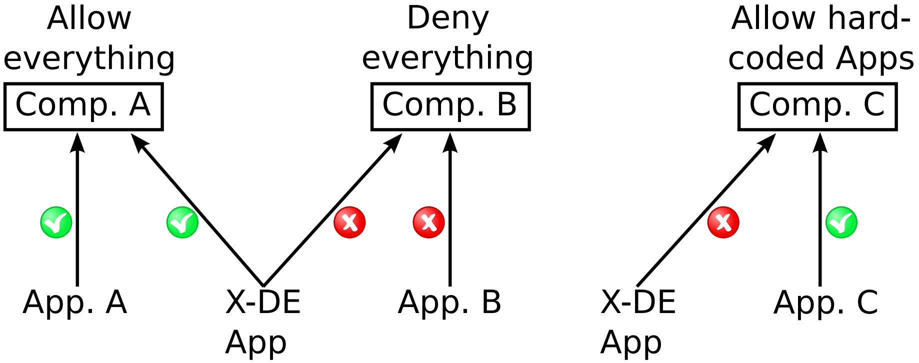
Wayland's privileged interfaces:

- Not defined yet, many discussions
- partially due to the security implications
- the compositor sometimes need the user intent (Trusted Path)
- users or packagers may want to work around that!

Example: Wayland/Clipboard

Reading the clipboard doesn't seem to be defined. Drag & Drop is however supported because the compositor gets the user's intent!

Example of policies for accessing a privileged iface



Challenges of defining policies

Challenges of defining policies

- Many Desktop Environments (Gnome, KDE, Tizen, XFCE, etc...)
- DEs won't agree on a single policy
- Cross-DE apps cannot ship with a policy for every DE
- Packagers need a simple policy interface

Possible solution?

- Abstract the decision process in a multi-backend library
- Create a generic policy and per-DE tweaks

Outline

Wayland Security Modules

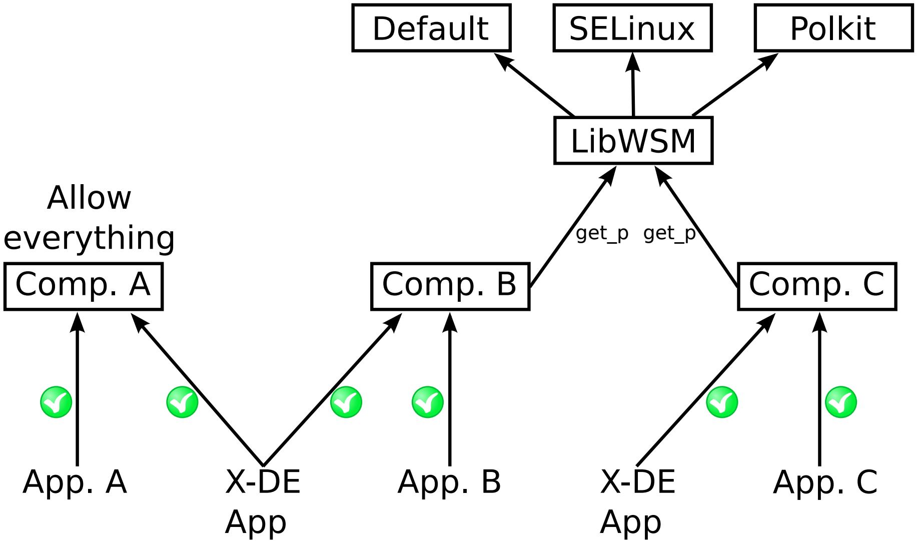
Goals

- Provide security decisions for Wayland privileged ifaces
- Help DEs store policy for their ifaces in a centralised way
- Support innovation and standardisation over time

How we do that

- Hooks on all privileged ifaces in the Wayland API
- Support for any backend/module: just a few symbols to export
- Simple: currently about 1100 LOCs w/ default backend
- Very extensible!

Example of policies for accessing a privileged iface



How to Use

Modifications to a compositor

1. `wsm_init()` on compositor start
2. `wsm_new_client()` on new client
3. Users choose a backend and write a policy – or use ours!
4. When implementing privileged ifaces, call `wsm_get_permission()`
5. Got custom semantics? Call `wsm_get_custom_permission()`
6. `wsm_client_free()` and `wsm_fini()` to clean up

Source

`https://github.com/mupuf/libwsm`

LibWSM Security Decisions 1/2

Four default semantics

Allow	Client explicitly allowed use of a privileged iface.
Soft Allow	Client allowed, but there could be issues. We recommend notifying the user.
Soft Deny	Client denied, but no particular concern. You could grant access via trusted UIs or prompts.
Deny	Client explicitly denied by policy, don't proceed.

If LibWSM answers something else, implement Deny.
If there is no policy, the default backend will reply Soft Deny.

LibWSM Security Decisions 2/2

Why the distinction?

Hard decisions	represent the actual security policy. Please respect it.
Soft decisions	are assumptions about what's best. Compositors can probably do better than just allow/deny.

Security notifications, Trusted UIs, User-driven access control and Permission prompts should come to mind with soft replies.

Extending LibWSM is Easy!

Support for custom capabilities...

Compositors should be let to innovate securely: custom ifaces like `_WESTON_FULLSCREEN` can be mediated.

Custom decision semantics...

If you implement specific behaviours, you can express them in the policy *e.g.*, “allow if no sensitive apps open” for `WSM_SCREENSHARING`.

Different policies per compositor...

Write per-DE policies just like in menu or autostart files, e.g.:

```
[GNOME]
WSM_RAISE_FOCUS=deny
```

LibWSM Capabilities 1/3

`WSM_SCREENSHOT`

Take a screenshot of the whole screen (Soft Deny)

`WSM_SCREENSHARING`

Record the screen continuously (Soft Deny)

`WSM_VIRTUAL_KEYBOARD`

Inject or filter input events on keyboard (Soft Deny)

`WSM_VIRTUAL_POINTING`

Modify the position of the pointer and simulate clicks (Soft Deny)

LibWSM Capabilities 2/3

`WSM_FULLSCREEN`

Use the entire screen (Soft Allow)

`WSM_GLOBAL_KEYBOARD_SEQUENCE` [obj: key sequence]

Receive keyboard sequences even when not active (Soft Deny)

`WSM_FORWARD_RESERVED_KEYBOARD_SEQUENCE` [obj: key sequence]

Receive reserved compositor sequences when active (Soft Deny)

`WSM_RAISE_FOCUS`

Raise the window and grab focus programmatically (Soft Allow)

LibWSM Capabilities 3/3

`WSM_CLIPBOARD_COPY`

Copy programmatically to the clipboard (Allow)

`WSM_CLIPBOARD_PASTE`

Paste from the clipboard (Soft Deny)

! Of course this list is provisional. Please suggest corrections/additions at `https://github.com/mupuf/libwsm/issues`.

The Default Backend

Policy managed with files

- default policy, app-specific policies, policy templates (WIP)
- system-wide policies can be customised by users

A single source of policy per app at any time

- more manageable for packagers and distributions
- better visibility (includes in SELinux are a recipe for trouble)

Example Default Policy

```
[Wayland Security Entry]
Exec=*
Version=1

[All Compositors]
WSM_FULLSCREEN=soft-allow
WSM_CLIPBOARD_COPY=allow
WSM_RAISE_FOCUS=soft-allow

[Paranoid Shell]
WSM_FULLSCREEN=deny
WSM_CLIPBOARD_COPY=deny

[GNOME]
_GNOME_USE_SHELL_API=basic-access

[Mupuf]
WSM_FULLSCREEN=soft-allow
_WESTON_FULLSCREEN=soft-allow

[Weston]
_WESTON_FULLSCREEN=implicit-deny
```

Example Policy for an Application

```
[Wayland Security Entry]
Exec=/usr/bin/example
Version=1

[All Compositors]
WSM_FULLSCREEN=allow
WSM_CLIPBOARD_COPY=deny

[Paranoid Shell]
WSM_FULLSCREEN=deny
WSM_CLIPBOARD_COPY=deny

[GNOME]
_GNOME_USE_SHELL_API=allow

[Mupuf]
WSM_FULLSCREEN=allow
_WESTON_FULLSCREEN=permanent-allow-if-frequent

[Weston]
_WESTON_FULLSCREEN=allow
```

This is Just a Backbone

We made it as flexible as possible.

Goal: positive user experiences

- Easy to conceptualise and edit policy, easy to add UI
- Soft decisions let DEs build their own UX

Need something more for your shell? Talk to us!

Potential Security Tasks & Interactions

GUIs might be necessary for

- Answering an app's permission request
- Installing an app which needs privileges
- Knowing what privileges an app has
- Revoking an app's privileges
- Prevent annoying apps from requesting privileges
    - ("why" may influence the design)

This is the job of DEs!

Outline

Introducing my research group...

I'm a PhD student at UCL Info Sec

We specialise in usable and productive security.

I'm investigating how Linux users manage their security!

- We don't know what makes security software work... yet
- We don't know how to change home user behaviour... yet
- But we know how to do user research!

Shameless advertising: `http://sec.cs.ucl.ac.uk`

Some Research Knowledge 1/3

User time is precious, respect it

- Nobody wants to perform security tasks! People *rationally* reject costly security advice
- Nobody has time for security! More security pressure decreases willingness to comply
- Most security warnings discarded in $2s$. One reason: people habituated to warnings, but there could be others

[Beautement et al., 2008, Herley, 2009, Krol et al., 2012, Akhawe and Felt, 2013]

Some Research Knowledge 2/3

Security is desirable

- People in corporate want insecure options *hidden by default*
- They still bypass or disable security to get stuff done
- And still deploy their own security strategies to compensate

[Bartsch and Sasse, 2013, Kirlappos et al., 2014] + private conversations with Sasse

Some Research Knowledge 3/3

Home users are not very concerned

- Home users think they will never be targeted. Telling them about opportunistic criminals best triggers concern
- Speak of malware as medical infections, it's better understood than other metaphors
- Some people think antivirus software keeps them safe
- Linux users feel Linux is more secure (but it's not)

[Camp, 2006, Wash, 2010, Krol et al., 2012] + market research and own observations

What Makes Security Work?

Warning: nobody really knows!

Some sanity principles for now:

1. Don’t put the cost of security on users!
   - Interaction cost of security
   - Initial cost of *‘opt-in’* security
   - Lost features $\rightarrow$ hinders practices
2. Don’t ask app devs to do the security
3. Complex policy = opaque breakdowns
4. Test with real people as often as possible

Outline

1 Trusted UIs

What, How

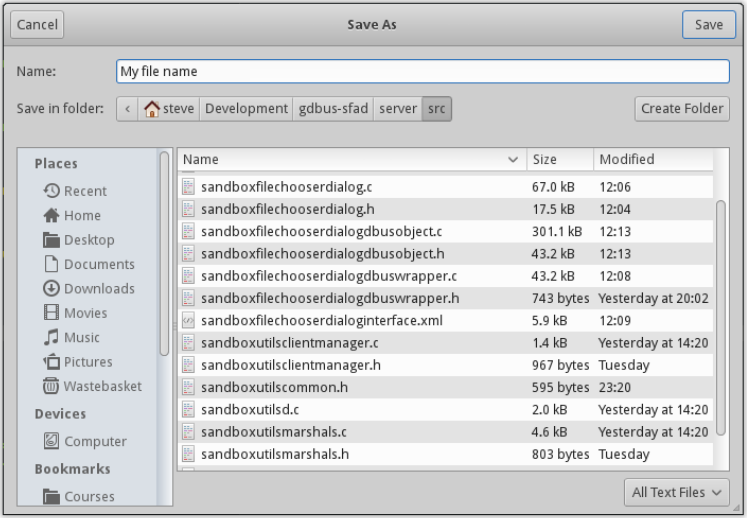
- UIs that are controlled by the OS/Compositor (not by apps)
- **Trusted path between Trusted UIs and user**
- Which means need for a UI embedding protocol

Goal

- Indicate user intent without policy when interacted with
  - Trusted File Dialog / Power boxes [Yee, 2004]
  - Secure “Take Photo” button on Android [Roesner et al., 2012]
- Some Trusted UIs in Windows 8 and OS X
- Sadly, they require API changes in toolkits...

PoC: File Chooser Dialog

Remediating the Need for Filename Autocompletion

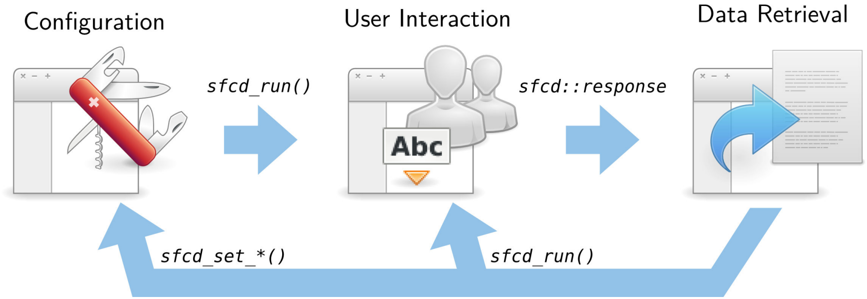


Name: My file name

Name	Size	Modified
sandboxfilechooserdialog.c	67.0 kB	12:06
sandboxfilechooserdialog.h	17.5 kB	12:04
sandboxfilechooserdialogdbusobject.c	301.1 kB	12:13
sandboxfilechooserdialogdbusobject.h	43.2 kB	12:13
sandboxfilechooserdialogdbuswrapper.c	43.2 kB	12:08
sandboxfilechooserdialogdbuswrapper.h	743 bytes	Yesterday at 20:02
sandboxfilechooserdialoginterface.xml	5.9 kB	12:09
sandboxutilsclientmanager.c	1.4 kB	Yesterday at 14:20
sandboxutilsclientmanager.h	967 bytes	Tuesday
sandboxutilscommon.h	595 bytes	23:20
sandboxutilsd.c	2.0 kB	Yesterday at 14:20
sandboxutilsmarshals.c	4.6 kB	Yesterday at 14:20
sandboxutilsmarshals.h	803 bytes	Tuesday

An Initial PoC for GTK+ File Chooser Dialog

- Mirror the “normal” usage of API: almost all apps portable
- Prevent attacks when user interacts w/ GUI
- Expose statefulness and error handling to apps



! Microsoft switched to a full async programming model: no error handling when permission is denied!

Is it really that simple?

Issues arising with new FCD

- File preview: should move to built-in DE routine
- **Custom widgets: needs a UI embedding protocol**

  would work for apps **currently** using such widgets
- Autocompletion of file extension when saving?

! Truth is both APIs and widgets need to be redesigned.
More complex workflows need entirely new UIs

Another Example: X11 Selections

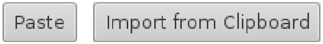
An actual security issue

- Apps can paste selection content w/out mediation
- They routinely check if they support the content type of selections

Is a Secure Clipboard Possible?

Apps should receive clipboard events

- Implication: need a reserved hotkey for pasting
- And Trusted UIs buttons and menu items



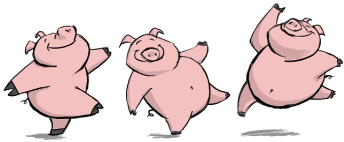
- **We do need a (two-way) UI embedding protocol**

Risk of issues down the road

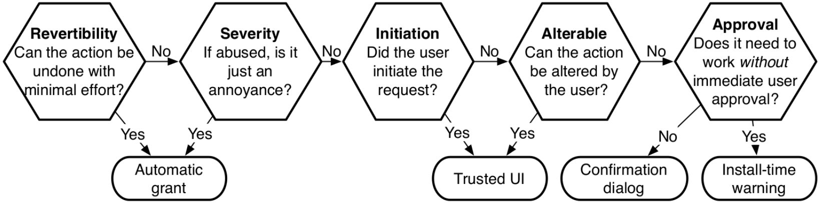
- What about apps that use other labels/keyboard sequences?
- Share your concerns/critics with us, it will help!

2 Permission Requests

Permission requests **don't provide security.**
Systematically ignored: lack of experience, disruptiveness, immediate gratification bias, **economic rationality / compliance budget**



How to ask for Permission [Felt et al., 2012]


Revertibility Can the action be undone with minimal effort?
No
Severity If abused, is it just an annoyance?
No
Initiation Did the user initiate the request?
No
Alterable Can the action be altered by the user?
No
Approval Does it need to work *without* immediate user approval?
Yes
Yes
Automatic grant
Yes
Yes
Trusted UI
No
Confirmation dialog
Yes
Install-time warning

Permission Requests (are totally offtopic)

You have 2 seconds per UI! No more!

- Don’t expect rational decisions, rely on interaction cost instead
- Make it useful and actionable
- Identify who is asking for permission
- Visualise what is asked for

[Reeder, 2011 on NEAT warnings, Day, 2014 on sandboxing]

! Good design is DE-dependant. No infrastructure work needed in Wayland for permission UIs.

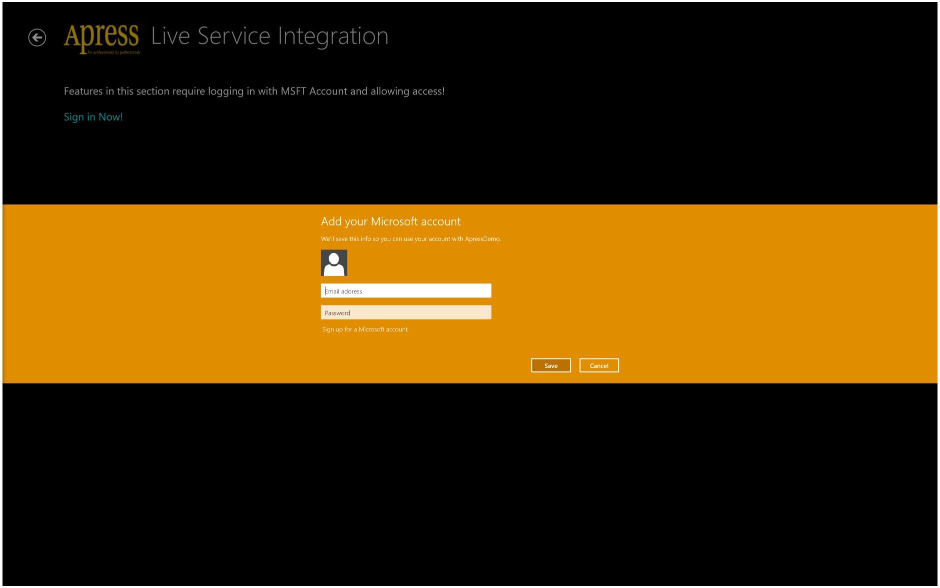
❸ Authentication UIs

Spoofing Auth UIs is highly rewarding... and easy

- Just fake the `polkit` UI!
- Or go fullscreen and fake:
  - The whole desktop environment
  - The greeter/lock screen
  - A Web login page

© Cyberoam

Please Pwn my Authentication UI



Apress Live Service Integration

Features in this section require logging in with MSFT Account and allowing access!

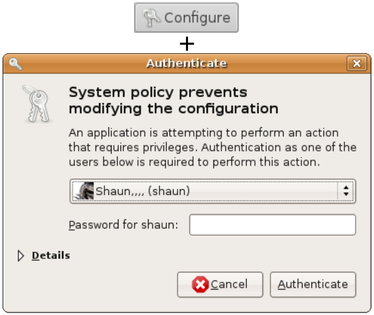
Sign in Now!

Add your Microsoft account

We'll save this info so you can use your account with ApressDemo.

Please Pwn my Authentication UI

When users open your rogue app’s settings, display this:



System policy prevents modifying the configuration

An application is attempting to perform an action that requires privileges. Authentication as one of the users below is required to perform this action.

Shaun,,,, (shaun)

Password for shaun:

Defences

Three (imperfect) approaches

Unspoofability	GFX effects only the shell can make
Indirections	add a user action to all auth (e.g., Windows UAC)
Mutual Auth	make the DE show the user a secret

! All three can be broken with a bit of user inattention/confusion.

Attacking Auth UI Defences

Unspoofable Effects

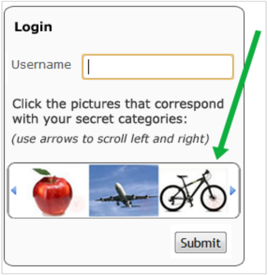
- *e.g.*, wobbly animation on background windows
- Not trivial to simulate but...
- Will users pay attention to the auth dialog or to the background?

Indirections

- Ctrl+Alt+Del? Bind each key individually and show your fake UI
- Or tell the user an error occurred and pop a new UI

Actual out-of-band communication is necessary!

Example: "Anti-Phishing" Mutual Auth


Login

Username

Click the pictures that correspond with your secret categories:

(use arrows to scroll left and right)

Ⓒ Confident Technologies

! Tell users “The image server is down” and they all proceed!

Lesson: cultural context matters. Errors expected on the Internet

Outline

Thanks for your Attention!

What we have

- A LibWSM prototype to enable privileged interfaces
- Open problems for security UIs design
- More detailled versions on `mupuf.org/blog`

What comes next

- Martin to finish and maintain LibWSM
- Steve to study Linux users' security practices in-the-wild
- All of us to review privileged APIs?

Steve Dodier-Lazaro
s.dodier-lazaro@cs.ucl.ac.uk

Martin Peres
martin.peres@labri.fr

Akhawe, D. and Felt, A. P. (2013).
Alice in warningland: A large-scale field study of browser security warning effectiveness.
In Proceedings of the 22Nd USENIX Conference on Security, SEC'13, pages 257–272, Berkeley, CA, USA. USENIX Association.

Bartsch, S. and Sasse, M. A. (2013).
How users bypass access control – and why: The impact of authorization problems on individuals and the organization.
In Proceedings of the 21st European Conference on Information Systems (ECIS 2013).

Beautement, A., Sasse, M. A., and Wonham, M. (2008).
The compliance budget: managing security behaviour in organisations.
In Proceedings of the 2008 workshop on New security paradigms, NSPW '08, pages 47–58, New York, NY, USA. ACM.

Camp, L. J. (2006).
Mental models of privacy and security.
Available at SSRN 922735.

Felt, A. P., Egelman, S., Finifter, M., Akhawe, D., and Wagner, D. (2012).
How to ask for permission.
In Proceedings of the 7th USENIX Conference on Hot Topics in Security, HotSec'12, pages 7–7, Berkeley, CA, USA. USENIX Association.

Herley, C. (2009).
So long, and no thanks for the externalities: the rational rejection of security advice by users.
In Proceedings of the 2009 workshop on New security paradigms workshop, NSPW '09, pages 133–144, New York, NY, USA. ACM.

Kirlappos, I., Parkin, S., and Sasse, M. (2014).

Learning from “shadow security”: Why understanding non-compliance provides the basis for effective security.
In Workshop on Usable Security, San Diego, California.

Krol, K., Moroz, M., and Sasse, M. A. (2012).
Don’t work. can’t work? why it’s time to rethink security warnings.
In Risk and Security of Internet and Systems (CRiSIS), 2012 7th International Conference on, pages 1–8.

Loscocco, P. A., Smalley, S. D., Muckelbauer, P. A., Taylor, R. C., Turner, S. J., and Farrell, J. F. (1998).
The inevitability of failure: The flawed assumption of security in modern computing environments.
In In Proceedings of the 21st National Information Systems Security Conference, pages 303–314.

Roesner, F., Kohno, T., Moshchuk, A., Parno, B., Wang, H. J., and Cowan, C. (2012).

User-driven access control: Rethinking permission granting in modern operating systems.
In Proceedings of the 2012 IEEE Symposium on Security and Privacy, SP '12, pages 224–238, Washington, DC, USA. IEEE Computer Society.

Saltzer, J. H. and Schroeder, M. D. (1975).
The Protection of Information in Computer Systems.

Wash, R. (2010).
Folk models of home computer security.
In Proceedings of the Sixth Symposium on Usable Privacy and Security, SOUPS '10, pages 11:1–11:16, New York, NY, USA. ACM.

Yee, K.-P. (2004).
Aligning security and usability.
Security Privacy, IEEE, 2(5):48 –55.