The Most Dangerous Code in the Browser

Stefan Heule, Devon Rifkin, Alejandro Russo, Deian Stefan

Stanford University, Chalmers University of Technology

Web Browsers Today

One of the most popular application platforms

- Easy to deploy and access
- Almost anything available as a web app
- Including very sensitive content (e.g., banking, email, passwords, health care)

Security built in

- E.g., website cannot steal locally stored photos
- Achieved through, e.g., same-origin policy (SOP)
- User does not need to worry about this

Browser Extensions

Users want more functionality

- Customize websites: content, behavior and display
- New functionality for websites
- Change browser

Browsers provide extension systems

Extension Security

Extensions are meant to interact with websites

- Challenging for user privacy and security

Firefox

- Extensions are powerful
  - Can change almost any aspect (and run native code)
- Can be installed from anywhere
- Web store: static analysis and human review

Chrome Extension Security

Split into extensions and plugins

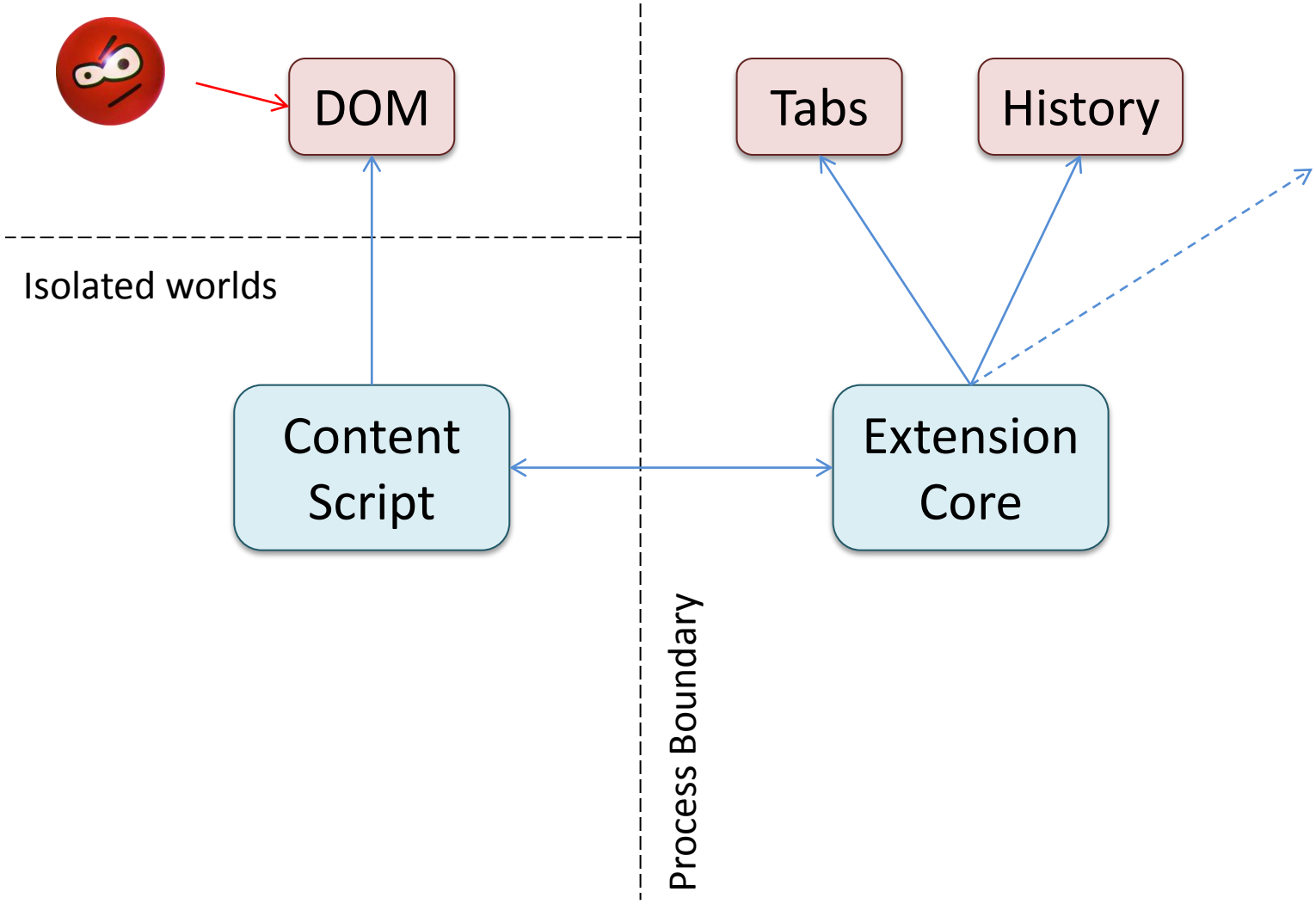
Plugins: native code

- Flash, Java, PDF, Silverlight
- Require manual review

Extensions: JavaScript based

- Vast majority are in this category
- Extension can only be installed from Chrome Web Store

Chrome Extension Architecture



Chrome Threat Model

Extensions are *benign-but-buggy*

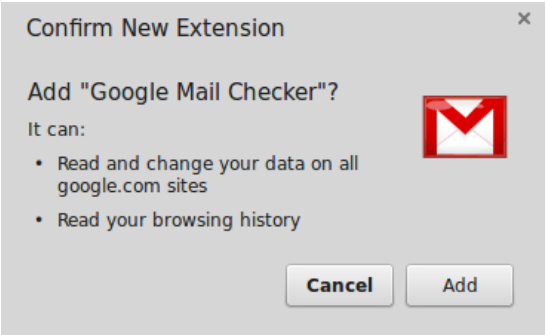
- Protect extensions from websites

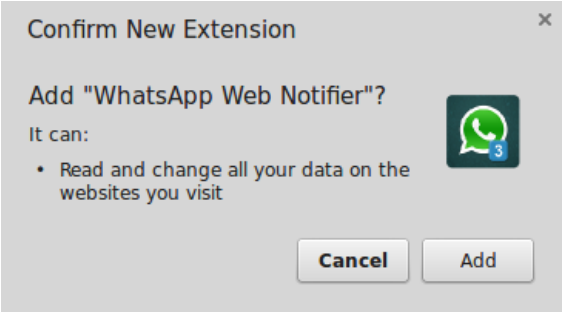
Principle of least privilege

- Extensions ask for permissions
- Typically asked for at install time

Permissions of Top 500 Extensions

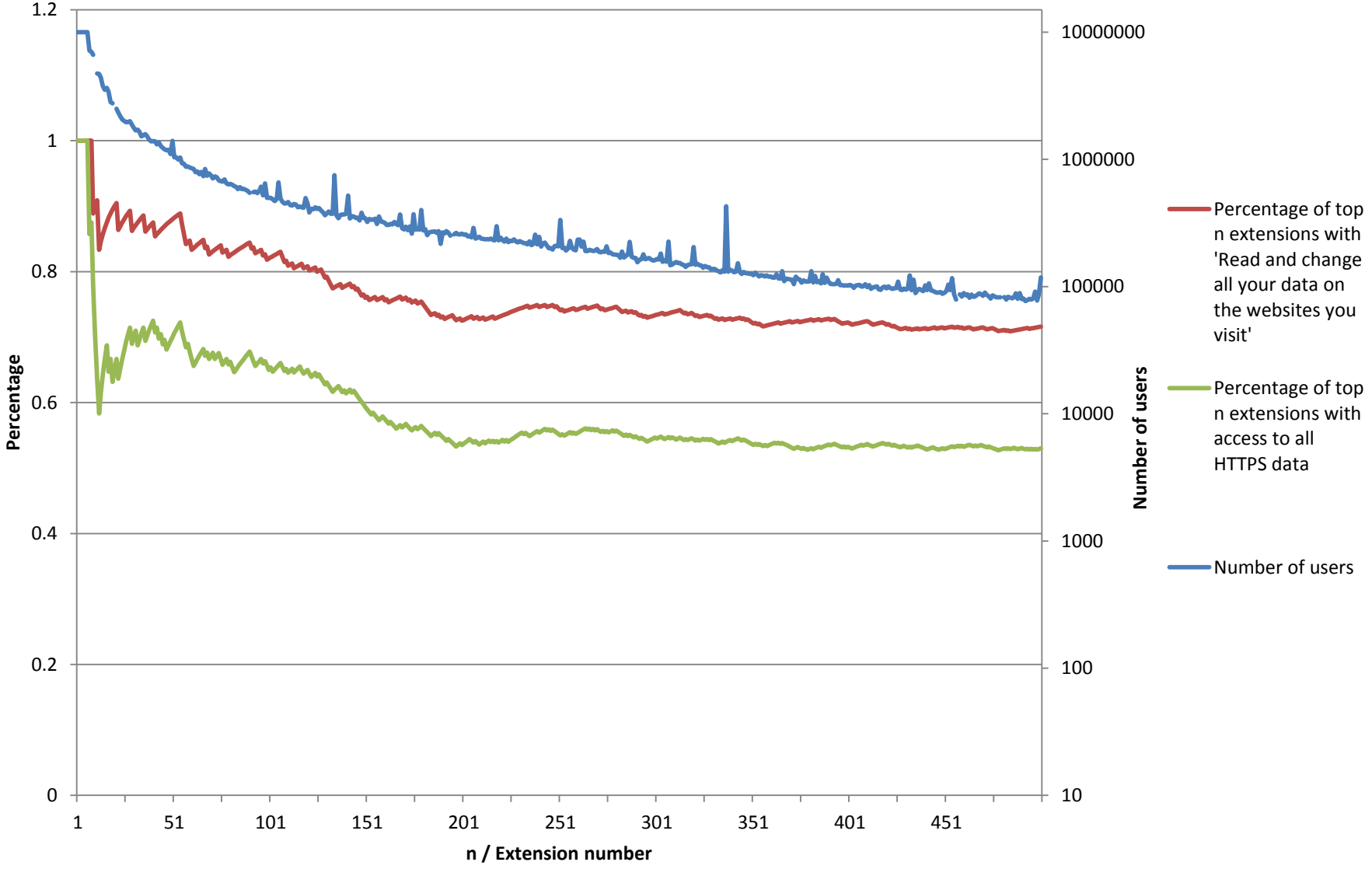
Permission	Count	Permission	Count
tabs	*75.6%*	*webRequestBlocking	*25.6%*
*storage	*38.4%*	*cookies	*24.6%*
http://*/*	*37.8%*	*unlimitedStorage	*20.4%*
https://*/*	*36.4%*	<all_urls>	*19.2%*
*contextMenus	*36.0%*	webNavigation	*16.6%*
*webRequest	*32.2%*	management	*14.6%*
*notifications	*30.4%*	history	*10.4%*





71.6% can “Read and modify all your data on all websites you visit”

Permissions are Meaningless



Problems

Permissions are broad and vague; without context

Users desensitized to permission requests

Incentives for developers to asks for too many permissions

- Adding permissions later requires user action

Attacker model assumes extensions to be benign

Attacks in the Wild

Google recently removed ~200 malicious extensions [Oakland'15]

- 5% of unique IPs accessing Google had at least one malicious extension
- Some injected ads, others steal personal information

Popular extension developers get contacted to sell extension

- And then update with malicious code

New Extension System: Goals

1. Handle mutually distrusting code
   - Extensions are protected from websites
   - Sensitive (website) user data is protected from extensions

   Attacker executes arbitrary extension to leak user data
2. Provide a meaningful permission system
   - Safe behavior should not require permission
   - Permissions should be *fine-grained* and *content-specific*
3. Incentivize safety
   - Many extensions should not require permissions

Preliminary Design

Reading sensitive data is safe

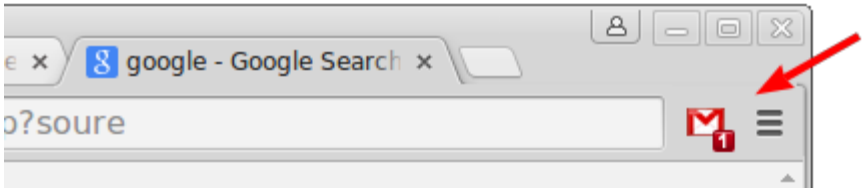
– if not disseminated arbitrarily

Mandatory access control (MAC) confinement

– Track sensitivity of information through application

Proposal: use coarse-grained confinement system like COWL [OSDI'14]

Example: Google Mail Checker



Extension reads unread count from gmail

- Gets tainted with mail.google.com
- No further communication with evil.com allowed

Not all extensions are this simple

- Need richer extension APIs

Explicit Sharing

Some users want to leak information

- Save snippet to Evernote
- Share webpage to Pintrest

Forbidden according to MAC

- Corresponds to information declassification

Leverage user intent with a sharing API

- Trusted UI, e.g. “Share with …” context menu

Encrypted Sharing

System allows labeled values

- Can freely be passed, only tainted when inspected

Encryption API takes labeled value, returns unlabeled encrypted value

- Can now be freely shared, e.g. sync to other device

Secure LastPass-style password manager

- Cloud only sees encrypted values, user controls master key
- When decrypted, passwords cannot leave browser due to MAC

More APIs

Declarative CSS API

– Change the display of a website

Networking API

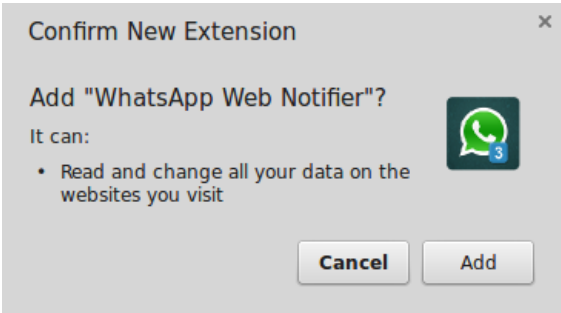
– E.g., to block undesired requests (AdBlock)

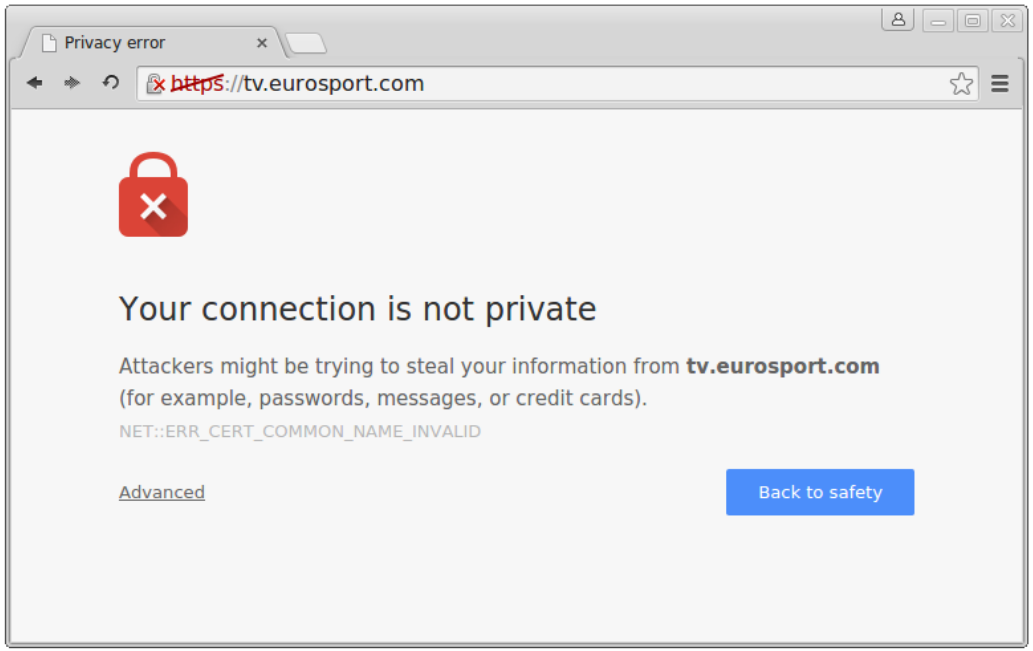
DOM access

– Isolate extension from website using shadow DOM

Safe by Default

When a large class of extensions can be written safely without permissions, warnings can become meaningful again





Conclusion

Extensions most dangerous to user privacy

– This need not be!

Strong guarantees of MAC-based confinement system allow many extensions to be safe

Meaningful permissions/warnings otherwise

– Fine-grained and content specific, at runtime

Thank you ☺

Questions?