

Enhancing web browser security against malware extensions

Mike Ter Louw Jin Soon Lim V.N. Venkatakrishnan

Department of Computer Science, University of Illinois at Chicago

Abstract

This work examines security issues of functionality extension mechanisms supported by web browsers. These extensions (or “add-ons”) in modern web browsers enjoy unrestrained access and thus are attractive vectors for malware.

We have taken advantage of the lack of security mechanisms for browser extensions and implemented a malware application for the popular Firefox web browser, that we call BROWERSPY, requiring no special privileges to be installed. BROWERSPY can observe all activity performed through the browser and is undetectable.

We then adopt the role of defenders to develop protection strategies against such malware. Our primary contribution is a mechanism that uses code integrity checking techniques to control the extension installation and loading process. We also explore techniques for runtime monitoring of extension behavior to provide a foundation for defending threats posed by installed extensions.

Introduction

People use web browsers for very personal activities where data confidentiality and content integrity are vital:

- web-based email
- online banking
- shopping
- professional business
- social networking

Malware running in an unprotected browser can observe and arbitrarily tamper with the data exchanged during these activities. It can apply contextual meaning to the data and access it as *plaintext*, even when transmissions are encrypted.

Thousands of people willingly install browser extensions without knowledge of the software’s capabilities. We show that it is trivial to develop a *malicious extension* able to conceal its rogue behavior. How will its users know they are safe from data theft and tampering?

A malware extension

To explore browser vulnerabilities and defense mechanism effectiveness, we created BROWERSPY — a malware extension with the following properties:

- Target:** Firefox web browser
- Development time:** Approximately 3 weeks
- Data theft of:** Web navigation history, submitted form data, saved passwords
- Concealment:** Hides presence in browser UI and file system (via injection into a trusted extension)

Defense strategy

We enhanced the security of Firefox in two general areas:

1. Trusted code base (TCB) integrity
2. User data confidentiality and integrity

These areas are depicted in Figure 1.

We protect TCB integrity by giving the user strict control over which extensions are allowed to execute in the browser. Before any extension can be loaded, it must be verified as uncorrupted and user authorized.

User data flow is controlled by a *runtime monitor* to restrict read and write access by extensions.

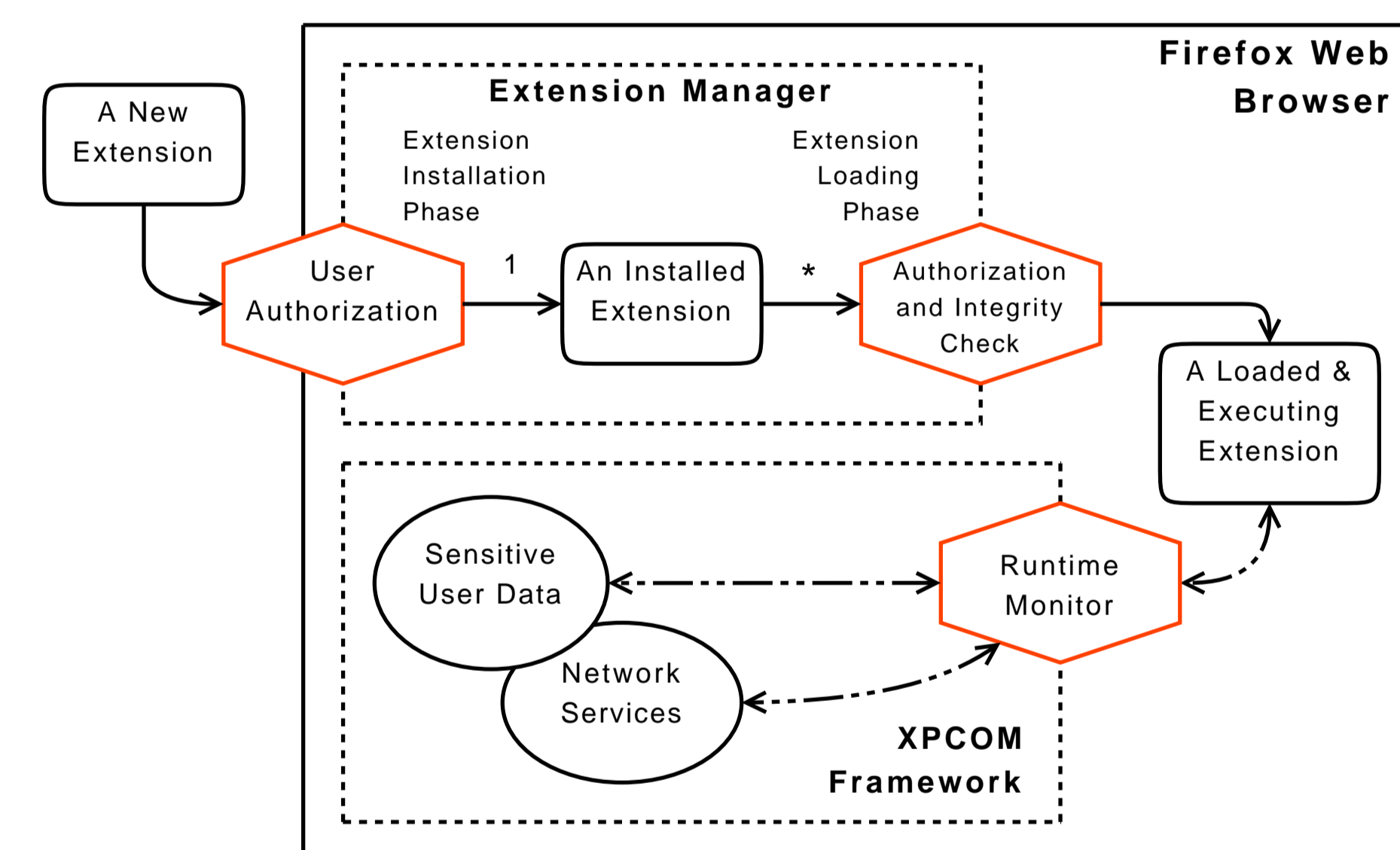


Figure 1: Overview of Firefox’s extensible architecture. Red areas represent security enhancements.

Code signing

Code signing of an extension by its creator is the typical mechanism by which code integrity assurances are provided. A cryptographic signature of the original program code is produced and is checked before loading the extension to verify its code has not been tampered with.

Although code signing can assure code base integrity, it can not tell us which extensions should be allowed in the browser’s TCB.

User-signed certificates

We propose a new technique that provides more comprehensive integrity assurance over the browser TCB: *user code signing*. As a user installs an extension into her browser, she explicitly authorizes new code into the browser TCB by the act of code-signing the extension.

To implement user code signing, our research project creates user-signed certificates as extensions are installed. The structure of a user-signed certificate is shown in Figure 2.

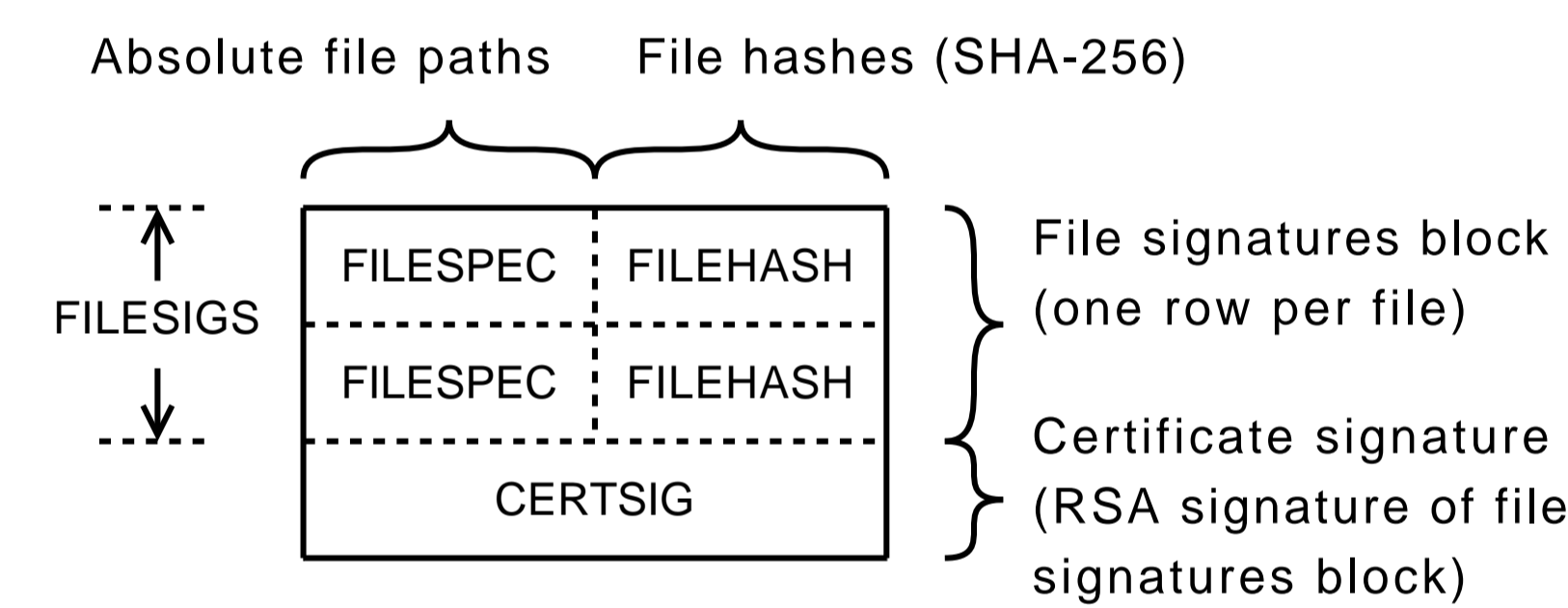


Figure 2: A user-signed extension certificate.

The user must authenticate using a valid password to decrypt the signing key needed to create the certificate. A viral extension therefore can not inject into the TCB.

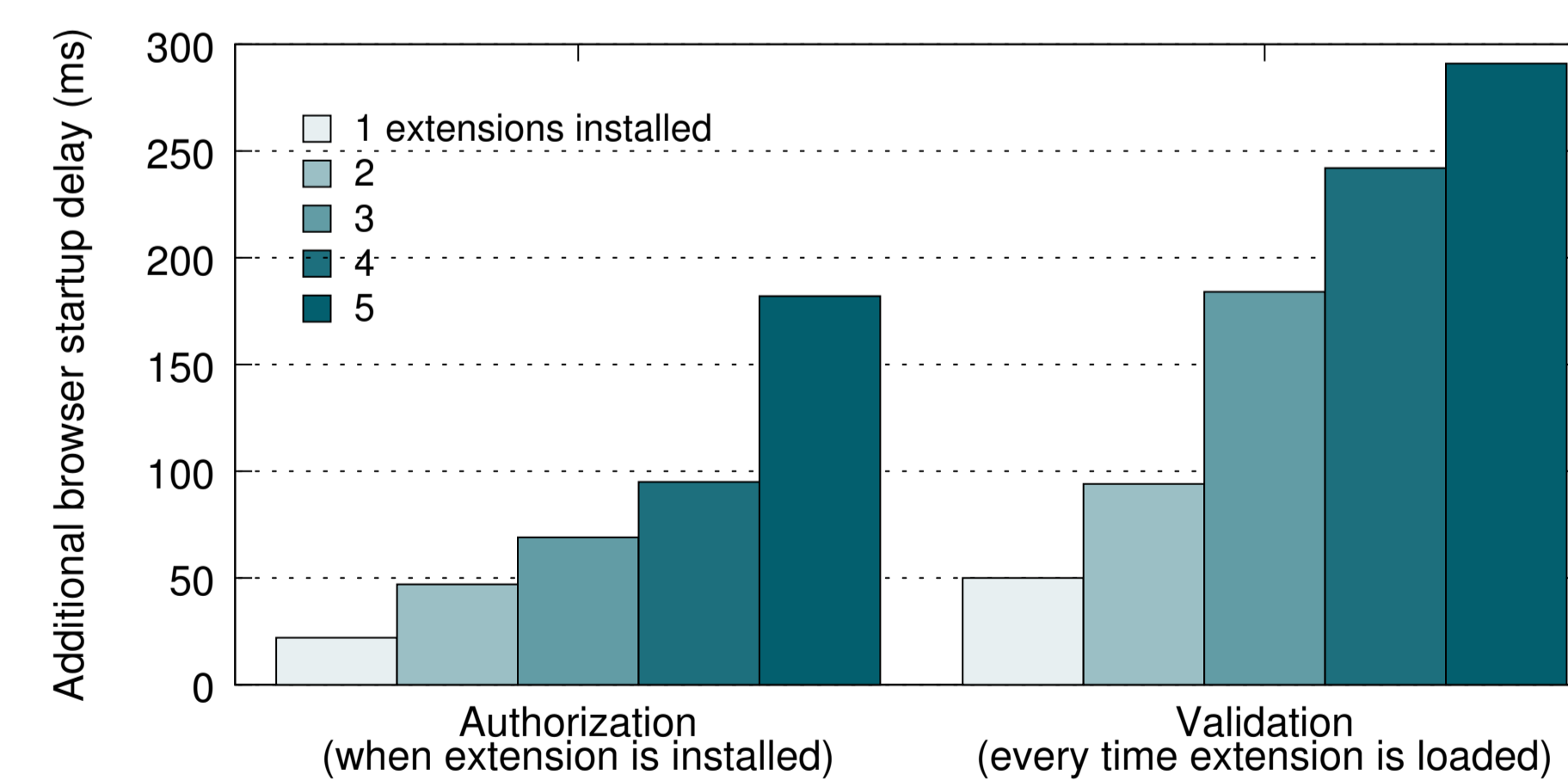


Figure 3: A user-signed extension certificate.

The performance impact of certificate creation and validation is almost imperceptible to the user when 5 extensions are installed:

Performance benchmark	Time (ms)
Total cert generation time	182
Average cert generation time	36.4
Total cert validation time	291
Average cert validation time	58.2

Runtime monitoring

Action attribution problem

To mediate an extension’s access to data we need the ability to determine if any given data read or write comes at the request of an extension. Firefox was not designed to easily facilitate this analysis when data is accessed. Our runtime monitor employs two methods to retrofit this ability onto Firefox, one of which is diagrammed in Figure 4.

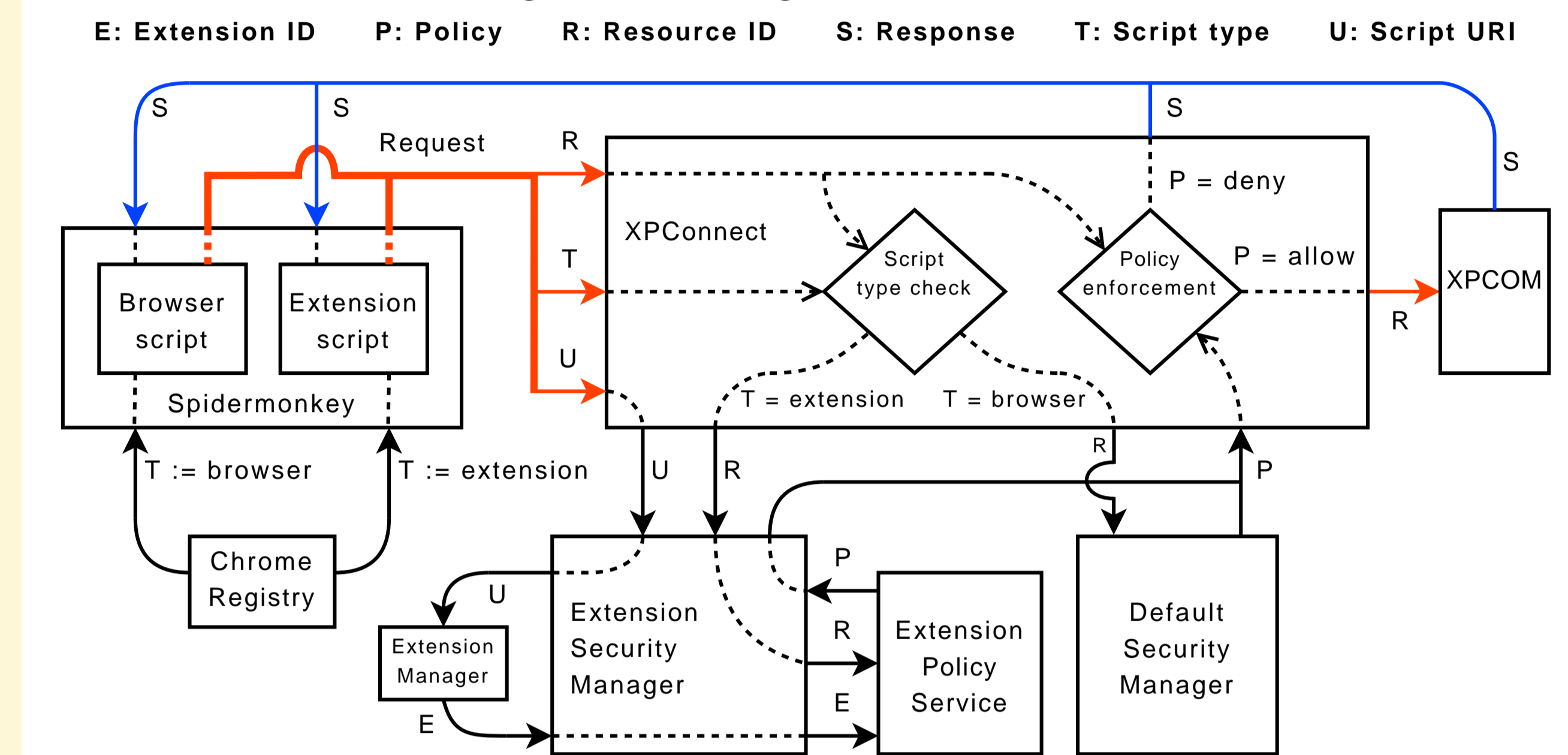


Figure 4: One policy enforcement mechanism: Extension code executing in Firefox must access resources (XPCOM) through a policy-enforcing intermediary, XPCConnect.

Policy enforcement

Our runtime monitor can enforce complex policies over all extensions or on a per-extension basis. Policies are tailored to each extension using the set of rules given in Table 2.

Policy name	What it does
XPCOM-ALLOW	Allow access to an XPCOM interface
XPCOM-DENY	Deny access to an XPCOM interface
SAME-ORIGIN	Allow “same-origin” network access
XPCOM-SAFE	Deny access to XPCOM while using SSL
PASS-RESTRICT	Deny access to the password manager
HISTORY-FLOW	Prevent URL history leaks to output streams

Table 2: Example policies created and tested with the runtime monitor.

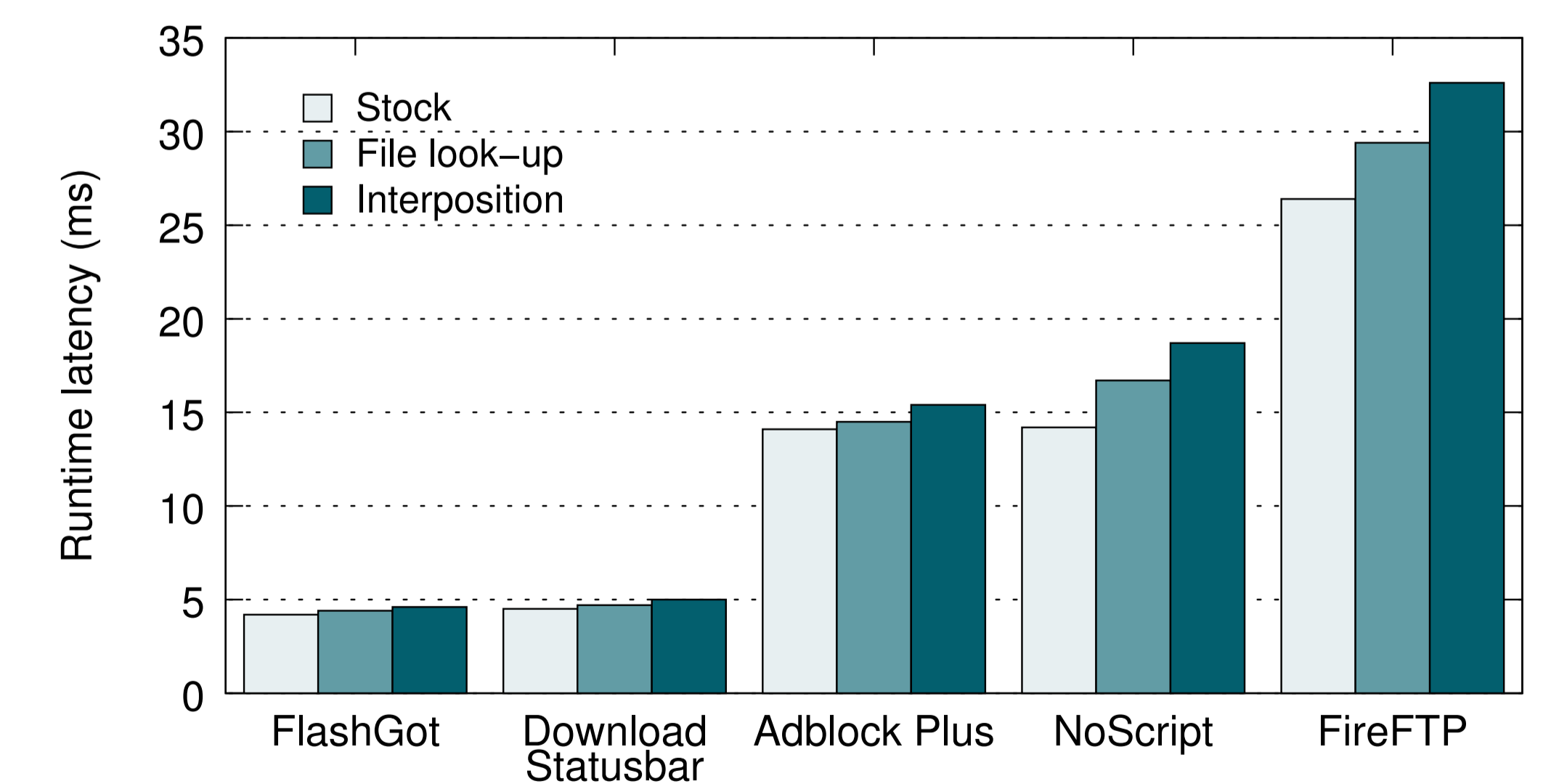


Figure 5: A user-signed extension certificate.

Conclusion

We authored a malware extension, BROWERSPY, that is representative of known attacks on the popular browser Firefox.

We protected Firefox’s trusted code base integrity from unauthorized ex-