Asset Protection via White Box Cryptography

Master thesis

Supervisors: Prof. Dr. Alexander Pretschner, Sebastian Banescu
Email: {alexander.pretschner, sebastian.banescu} @ in.tum.de
Phone: +49 89 289 – 17, 314
Starting date: immediately

Context

Many client-side software programs rely on locally stored asset files for customization and proper functioning. Such asset files are the target of malicious attackers who can gain benefits by changing software assets in particular ways. For instance, in the case of web-browsers, some examples of attackers are:

- Search engine providers, who wish to change the default search engine of users in order to gain market-share
- Plug-in providers, who wish to install their plug-in in the user's browser for advertisement and tracking purposes
- Botnet owner, who wish to control the host of the user and prevent him from finding security solutions online

These kinds of attackers generally create malicious software (malware) targeting a large user-base. Successful deployment of such malware generally causes user frustration and annoyance. This malware is very difficult to protect against, because it is a software process with normal user privileges running on the same host OS as the software it targets. Moreover, the target software may even be offline while the malware changes its asset file(s).

Protecting the asset files via cryptographic integrity checks is an intuitive solution. However, plain secret key storage is not secure on local systems, because an attacker could find the secret key in local memory. One alternative to plainly storing the key would be to hide it using the technique called White-Box Cryptography.

Goal

The goal of this thesis is to build a software component, which can be used to “sign” a digest of the asset file(s) which need to be protected. This digest must not be forgeable by an attacker who has administrative privileges on the platform on which the software using the asset files is running on.

The expected outcome of this thesis is to implement the aforementioned software component based on white-box cryptography techniques and integrate it into the Chromium open-source web-browser.

Workplan

1. Develop understanding of white-box cryptography techniques and attacks [2], [3], [4]
   a. Write state-of-art survey of several white-box cryptography techniques.
   b. Write security versus performance analysis of existing techniques.
2. Implement and integrate white-box cryptography technique(s) in Chromium
   a. Choose one or more white-box cryptography technique(s) through a coherent argumentation, i.e. security versus performance, which must be presented in written form.
   b. If 2 candidate techniques exist, both may be implemented
   c. Implement both as part of Chromium source code and as PNaCl executable
d. Implementation must be secure against code lifting and cryptanalysis attacks by an automated attack

e. A successful attack against one instance of the implementation should not be possible against another instance

f. The chosen technique(s) must be implemented and integrated with the Chromium open-source code project.

g. The design & implementation decisions must be documented in written form.

3. Evaluation of the implementation performance and its resistance to attacks

a. The performance impacts of the implementation must be recorded and compared to the original Chrome performance stats.

b. If different techniques were chosen in step 2, they shall also be compared with each other.

c. Analyse and discuss potential security and performance threats against the implemented solution. This should include resistance to known/published attacks [1].

4. The final thesis document must contain:

a. Description of the problem and motivation for the chosen approach

b. State of the art survey, including analysis of security and performance

c. Rationale for choosing certain technique(s) for implementation

d. Implementation description

e. Performance evaluation of implementation

f. Discussion on potential security and performance threats

g. Conclusions and future work.

Deliverables

- Virtual machine able to run a demo of the implementation, including instructions on how to run the demo.

- The VM should also include the source code of the implementation.

- Technical report with comprehensive documentation of the implementation, i.e. design decision, architecture description, API description and usage instructions.

- Final thesis report written in conformance with TUM guidelines.

References


