Detecting client-side e-banking fraud using a heuristic model

Tim Timmermans  Jurgen Kloosterman
tim.timmermans@os3.nl  jurgen.kloosterman@os3.nl

University of Amsterdam

July 4, 2013
E-banking malware;
Man-in-the-browser attack;
"Owns" the browser;
Not possible to detect malware with web techniques, i.e. JavaScript.
Figure 1: Normal banking web page
Servicemelding:

Geachte klant!

Tot onze spijt zijn momenteel alle servers van het Internetbankieren overbelast, waardoor u een kleine vertraging bij de toegang tot uw account en zijn functies kunt oplopen.

Wacht totdat het systeem uw aanvraag volledig uitgevoerd heeft, zodat u toegang kunt krijgen tot uw account.

Opmerking:
Het proces kan enkele minuten in beslag nemen afhankelijk van de mate van belastbaarheid van de servers van de bank.
Wij bieden u onze excuses aan voor dit tijdelijke ongemak!

Figure 2: Malicious banking web page
To what extend is it possible to detect maliciously injected code into a web page using a heuristic model in order to counteract fraud and what is the performance of such technique in terms of accuracy and execution time?
Current Solutions

- Pattern recognition;
- Cannot detect injections from unknown malware.
Related Work

- **CaffeineMonkey**: a method to analyse and detect malicious JavaScript (Feinstein et al.);

- **Prophiler**: a filter to examine millions of web pages for malicious content (Canali, Davide, et al.);

- **Zozzle**: a low-overhead solution that applies Bayesian analysis to detect JavaScript malware in the browser (Curtsinger, Charlie, et al.).
Approach (1)

- Supervised machine learning;
  - Labeling of benign and malicious pages
- Server-side detection mechanism;

**Goal:** detect injections from unknown malware and difficult to bypass.
Figure 3: Normal interaction with an e-banking web site.
Figure 4: Overview of fraud detection implementation.
Figure 5: Overview of the fraud detection model.
Method: feature extraction

Brief selection of features that are identified:

- iframes;
- inline styles;
- hidden elements;
- input fields;
- (obfuscated) Javascript;
- external Javascript, stylesheets and images.

Figure 6: Feature extraction component

A total of 26 relevant features are identified from HTML, Javascript and URLs
Method: preprocessor

- Transforms the feature data to a vector as input for the classifier;
- Assigns a maliciousness score based on the extracted URL features.

Figure 7: Preprocessor component
Method: classifier

- Naïve Bayes learning algorithm
- Two components
  - Trainer;
  - Classification.

Figure 8: Classifier components
Train the classifier on manual labeled malicious and benign pages.

Figure 9: Classifier - trainer component
Classifier: classification

- Classifies an unknown page against the training set using the Bayes’ theorem;
- Result consists of a probability between 0 and 100% for each class.

Figure 10: Classifier - classification
Results: performance

Mean execution time to classify an unknown page: 0.176 seconds.

Figure 11: Execution time performance
90% accuracy reached with \( \sim 32,000 \) instances in the training set.

**Figure 12**: Accuracy measurements
Experiment to validate the developed model:

1. Train classifier on page adapter by Zeus malware;
2. Classify a page adapted by Citadel malware.

Result: classified as malicious with a probability of 100%.
Classifier reaches an accuracy of 90% given the used dataset (needs validation with more complete set);
The developed model is able to counteract fraud, caused by (unknown) malware;
Classification process of a web page is performed with a mean of 0.176 seconds;
Improvement of the model may lower impact on resources and optimizing executing time.
References

