An Expert System for Automatic Software Protection

Leonardo Regano

Doctoral Program in Computer and Control Engineering
XXXI cycle
DAUIN – Department of Control and Computer Engineering

Tutor:
Prof. Antonio Lioy
Cataldo Basile, Ph.D.

Coordinator:
Prof. Matteo Sonza Reorda
Why is software security needed?

• software is used in (almost) every aspect of everyday life
  • e-banking, entertainment, e-government and many more

• attacks on weakly or non-protected software have a great impact on software companies
  • loss of intellectual property
  • loss of revenues: 46 billions $ in 2018

• using unlicensed software is dangerous
  • malware usually contained in pirated software
  • disclosure of sensitive data and/or identity theft

12018 BSA Global Security Survey: [https://gss.bsa.org/](https://gss.bsa.org/)
Software protection techniques

• objective: safeguard security requirements of software assets
  • assets: algorithms IP, license schemes, users’ data...
  • security requirements: confidentiality, integrity

• Man At The End (MATE) scenario
  • attacker has white-box access to application

• no perfect software protection exists
  • but protections can defer attacks
How to protect the software?

• protections decided and applied manually/empirically: several issues
  • long and complex vulnerability analysis
  • high expertise needed to choose the best protections
  • different platform+OS require different analysis

• an automatic approach? desirable
  • for the expert: can provide a good starting point
  • for the beginner: one click and do everything
Expert system for Software Protection (ESP)

- objective: provide an optimal *protection solution* for a given application
  - decide protections best able to safeguard the application assets
  - preserving the user experience

- can drive automatic protection tools
  - for a fully automated protection workflow

- implemented as a set of Eclipse plug-ins
Automated software protection workflow

Application source code
Source code analysis
Risk assessment
Asset protection
Asset hiding
Solution deployment
Protected binary

Knowledge Base
- Functions
- Variables
- Call graph
- Assets
- Protection solutions
- Attacks against assets
- Application structure
- Expert knowledge
Automated software protection workflow

- Application source code
- Source code analysis
- Risk assessment
- Asset protection
- Asset hiding
- Solution deployment
- Protected binary

Knowledge Base

- Protection techniques
- Protection tools
- Attacker profiles
- Attack tools
- Security requirements
- Attack step types
- Protection solutions
- Attacks against assets
- Application structure
- Expert knowledge
Software security meta-model\(^4\)

- formalizes all data handled by expert system
  - software security experts’ general knowledge
  - application-specific data
  - results of expert system

- OWL2 ontology
- classes and associations to describe:
  - application structure
  - assets and security requirements
  - attacks against assets
  - protections

\(^4\)C. Basile, D. Canavese, L. Regano, P. Falcarin, B. De Sutter, A meta-model for software protections and reverse engineering attacks, *Journal of Systems and Software, Volume 150, 2019*
Software security meta-model\textsuperscript{4}

- formalizes all data handled by expert system
  - software security experts’ general knowledge
  - application-specific data
  - results of expert system
- OWL2 ontology
- classes and associations to describe:
  - application structure
  - assets and security requirements
  - attacks against assets
  - protections

\textsuperscript{4}C. Basile, D. Canavese, L. Regano, P. Falcarin, B. De Sutter, A meta-model for software protections and reverse engineering attacks, \textit{Journal of Systems and Software, Volume 150, 2019}
Software security meta-model

- formalizes all data handled by expert system
  - software security experts’ general knowledge
  - application-specific data
  - results of expert system
- OWL2 ontology
- classes and associations to describe:
  - application structure
  - assets and security requirements
  - attacks against assets
  - protections

---

Risk assessment phase\(^5\)

- infers possible attacks
  - on the unprotected application
  - able to breach assets’ security requirements

- attack steps = simple attacker actions
  - expressed as Prolog inference rules

- attack paths = ordered sequences of attack steps
  - against actual assets

Risk assessment phase: attack paths

- integrity of license_check() is breached
  - license_check() is changed
    - statically change license_check()
      - statically locate license_check()
    - dynamically change license_check()
      - dynamically locate license_check()
  - skip license_check()
    - statically change main()
      - statically locate main()
    - dynamically change main()
      - dynamically locate main()
Asset protection phase

• infers the optimal protection solution best able to defer attack paths

• takes into account:
  • structure of application
  • assets+security requirements
  • attack paths from risk assessment phase
  • interactions among protection techniques
  • protected application slow-down

• decision based on
  • experts knowledge
  • quantitative asset metrics (e.g. cyclomatic complexity)
Asset protection phase: protections vs. attacks

execution correctness of license_check() is breached

license_check() is changed

- statically change license_check()
- statically locate license_check()

- dynamically change license_check()
- dynamically locate license_check()

skip license_check()

statically change main()

dynamically change main()

statically locate main()

dynamically locate main()

Anti-debugging license_check()

Anti-debugging main()
Asset protection phase: protections vs. attacks

- execution correctness of `license_check()` is breached
  - `license_check()` is changed
    - statically change `license_check()`
    - dynamically change `license_check()`
      - statically change `main()`
      - dynamically change `main()`
  - skip `license_check()`
    - dynamically change `main()`
    - dynamically locate `license_check()`
      - statically locate `main()`
      - dynamically locate `main()`

- Code mobility
  - `license_check()`
  - `main()`
Asset protection phase: protections vs. attacks

- execution correctness of license_check() is breached
  - license_check() is changed
    - statically change license_check()
      - statically locate license_check()
        - SW attestation license_check()
    - dynamically change license_check()
      - dynamically locate license_check()
  - skip license_check()
    - dynamically change main()
Asset protection phase: valid protection solutions

- must be able to defer all attack paths
- business logic of the application must remain unaltered
- ordering among protections applied on the same asset is important
- protected application slow-down must be below user-defined limits
Asset protection phase: game-theoretic approach
Asset protection phase: game-theoretic approach
Asset protection phase: game-theoretic approach
Asset protection phase: game-theoretic approach
Asset protection phase: game-theoretic approach
Asset protection phase: game-theoretic approach
Asset protection phase: game-theoretic approach
Asset protection phase: game-theoretic approach
Asset protection phase: game-theoretic approach
Asset protection phase: game-theoretic approach
Asset hiding phase

• problem: software protections might expose a "fingerprint"
  • fingerprints: code patterns, peculiar behaviors, etc.
  • attackers locate assets looking for protection fingerprints

• solution: Asset Hiding (AH) phase
  • apply protections to hide fingerprints
  • trade-off between fingerprint hiding and overhead
  • state of the art: manually obfuscate as much code as possible

---

Fingerprint example:

Control Flow Flattening

BB 2

BB 1

BB 3

BB 4

BB 5

...
Fingerprint example: Control Flow Flattening
Fingerprint example:
Control Flow Flattening
Asset hiding phase: strategies

- Asset Hiding strategies:
  - fingerprint replication
  - protected area enlargement
  - fingerprint shadowing

- deciding AH protections is difficult:
  - not all strategies are useful to hide all protections
  - some strategies may lower AP protections security
  - overhead must be taken into account
Asset hiding phase: approach

• objective: maximize the confusion index
  • confusion index: how much the attacker is expected do be delayed by the AH in finding the assets
  • applying an AH protection increases the confusion index

• custom Mixed Integer-Linear problem
  • based on the well-known Knapsack Problem
  • capacity constraints: overhead limits (e.g. CPU time, memory)
Validation by experts

• ESP tested on three real-life use-cases
  • OTP generator, application licensing scheme, DRM video player

• ESP results validated by software security experts
  • attack paths cover real attacks by tiger teams
  • protection solutions effectively block attacks
  • protection solutions leave applications business logic unaltered
  • protection solutions introduce limited overhead
Experimental results

<table>
<thead>
<tr>
<th>Application</th>
<th>SLOC</th>
<th>Functions</th>
<th>Assets</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>443</td>
<td>18</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>1029</td>
<td>47</td>
<td>15</td>
</tr>
<tr>
<td>C</td>
<td>3749</td>
<td>178</td>
<td>39</td>
</tr>
</tbody>
</table>
Conclusions and future work

• completely automated workflow for software protection
  • user must only identify assets and security requirements
  • infers attacks against assets
  • decides best protection to defer attacks
  • deploys protections by driving automatic protection tools

• results validated by software security experts

• future work: empirical assessment of software protections
  • master students asked to attack protected applications...
  • ...to assess how much attackers are deferred by protections
  • useful data to drive ESP reasoning processes
Thank you for your attention!

Questions?