Deemon: Detecting CSRF with Dynamic Analysis and Property Graphs

G. Pellegrino et al.
giancarlo.pellegrino@cispa.saarland

(presented by Martin Johns, SAP Security Research)
U WON’T BELIEVE WHAT DIS CAT IS DOIN’ !!!1!

Snowflake the Cat Reacts to RC Robot Spider

<!-- Change password to pwnd to access the image -->

```html
<img src="http://store.com/change_pwd.php?password=pwnd" width="0px" height="0px"/>
```
Cross-Site Request Forgery Attack

POST /login.php [...] user=Alice&pwd=secret

200 OK
Set-cookie: session=YBLqp32F

GET /video.html

GET /change_pwd.php?password=pwnd
Cookie: session=YBLqp32F

If credentials are valid, create and send a session cookies

If cookie is valid, then update password

Look at this cat video!
The Forgotten Sleeping Giant

• Popular vulnerability
  • Among top 10 security risks w/ XSS and SQLi
  • Discovered in popular websites, e.g., Gmail, Netflix, and ING

• Most of previous efforts spent on countermeasures:
  • Origin header, synchronizer tokens, and browser plugins

• A little has been done to provide techniques for the detection
  • Existing (semi-)automated techniques focus on input validation and logic flaws
  → Detection of CSRF via manual inspection

Giancarlo Pellegrino, gpellegrino@cispa.saarland
Building a tool to find CSRF

• CSRF is not overly hard to find for pen testers or security experts during dedicated security testing

• But
  • Bug pattern is unintuitive for developers
  • Security testing is often used in automated processes, such as Q-Gates or regression testing

• Hence,
  • Can we build a tool to find CSRF automatically?
So, why is it hard to detect CSRF automatically?

• Challenges (Operational):
  1) Application interaction
  2) Side-effect free testing

• Challenges (Detection):
  1) CSRF targets state transitions
  2) Attacker reliably create requests incl. parameters and values
  3) Not all state transitions are relevant
Challenge O1: Application interaction

• CSRF is rarely found on application entry pages
• Instead, in general it requires interaction with deeper functionality of the application
• Thus, “blind” black-box testing is unlikely to access all CSRF-relevant interfaces
Challenge O2: Side-effect free testing

• Remember: CSRF is all about causing lasting side-effects on the server-side
• But:
  • Testing for such side effects potentially causes... *side effects*
• Think:
  • Deletion of a shopping basket
  • Terminating an authenticated session
  • ...
• How can we ensure that our testing does interfere with our testing?

Giancarlo Pellegrino, gpellegrino@cispa.saarland
Challenge D1: CSRF Targets State Transitions

- Determine when a state transition occurs
- Not all operations change the state of a webapp
  - E.g., View user data vs reset user password
- Learning state transitions is possible
  - However, existing approach can be inaccurate or operation-specific

Giancarlo Pellegrino, gpellegrino@cispa.saarland
Challenge D2: Attacker Reliably Creates Requests

- Determine relationships between parameters and transitions
  - E.g., random security token may not be guessed by an attacker

- Existing techniques do not determine such a relationship
  - E.g., Web scanners match param names against list of predefined names (e.g., “token”)

Giancarlo Pellegrino, gpellegrino@cispa.saarland
Challenge D3: Not all State Transitions are Relevant

- Determine the relevance of a state transition
- State transitions can be the result of operations such as tracing user activities
  - They are state-changing operations but not necessarily security-relevant
- Easy for humans but hard for machines
Our approach: Deemon

• Approach: Guided grey-box testing
• Input: User generated interaction traces
  • E.g., Selenium scripts for regression/UI testing
• Infrastructure
  • HTTP observation
  • Instrumented server-side that monitors all state changes
Our Solution: Deemon

- Application-agnostic framework for developers and analysts
  1. Infer state transitions + data flow from program executions
  2. Property graphs for uniform and reusable model representation
  3. Graph traversals to select request candidates for testing
  4. Verify replay-ability of HTTP requests
Deemon: Architecture

Dynamic Trace Generation

Virtualized Env.

Login and change password

Giancarlo Pellegrino, gpellegrino@cispa.saarland
Reliable, repeatable workflow testing

• The architecture allows side-effect free testing
  • Set server VM into vanilla state
  • Run UI workflow and record all traffic & server-side effects
  • ...repeat

• Clear mapping between: UI interaction / HTTP requests / server-side effects
  • This allow the identification of single requests between traces

• Running the same UI workflow multiple times and comparing HTTP request parameters
  • With the same user -> session specific parameters
  • With different users -> user specific parameters
Deemon: Model Construction

Traces and Parse Trees

FSM

Data flow and types

Deemon: Model Construction

Traces and Parse Trees

FSM

Data flow and types

Giancarlo Pellegrino, gpellegrino@cispa.saarland
Deemon: Traversals

“Find all CSRF”

“Find all requests r such that:
1) r is state-changing
2) r can be created by an attacker
3) the state change is relevant”

∀n: request(n)
1) ∃tr, q_i, q_f: trans(tr, q_i, q_f)
   ∧ accepts(tr, n)
2) ∀v: variable(v)
   ∧ has(q_f, v)
   ∧ v.Types ∩ {“unguessable”} = ∅
3) relevant(r)”

[Query processor]

∀ v: variable(v) ∧ has(q_f, v) ∧ v.Types ∩ {“unguessable”} = ∅
Deemon: Testing

Graph Traversals

Requests

Queries

Test Execution

200 OK

GET

Virtualized Env.

Failed

Successful

Giancarlo Pellegrino, gpellegrino@cispa.saarland
Revisiting the Challenges

• O1) Application interaction
  • Guided testing via recorded workflows

• O2) Side-effect free testing
  • Removal of side effects via VM snapshots

• D1) CSRF targets state transitions
  • Monitoring of server-side effects

• D2) Attacker reliably create requests incl. parameters and values
  • Automated analysis of parameter roles and information flows

• D3) Not all state transitions are relevant
  • Removal of non-authentication and generic state transitions
Evaluation

- **Inputs:**
  - 10 Web apps from the Bitnami catalog (avg 600k LoC)
  - 93 workflows (e.g., change password, username, add/delete user/admin, enable/disable plugin)

- **1,380 requests**
  - 194 not st-ch
  - 1,186 st-ch
  - 1,022 not relevant

- **219 tests**
  - 29 succ.
  - 14 distinct CSRFs
  - 190 failed

- **Attacks:**
  - User account takeover in AbanteCart and OpenCart
  - Database corruption in Mautic
  - Web app takeover in Simple Invoices

Giancarlo Pellegrino, gpellegrino@cispa.saarland
Results Analysis: Awareness

1. **Complete Awareness**: all state-changing operations are protected
   - E.g., Horde, Oxid, and Prestashop

2. **Unawareness**: none of the relevant state-changing operations are protected
   - I.e., Simple Invoices

3. **Partial Awareness**
   - *Role-based*: only admin is protected
     - I.e., OpenCart and AbanteCart
   - *Operation-based*: adding data items is protected, deleting is not
     - I.e., Mautic
Takeaways

• Presented Deemon: Dynamic analysis + property graphs

• Deemon detected 14 CSFs that can be exploited to takeover accounts, websites, and compromise database integrity

• Discovered alarming behaviors: security-sensitive operations are protected in a selective manner

• Read all the gory details or play with Deemon:
  • https://github.com/tgianko/deemon

Giancarlo Pellegrino, gpellegrino@cispa.saarland