Distributed Password Cracking Platform

Dimitar Pavlov
Gerrie Veerman
UvA SNE Master Students
08-02-2012

Supervisors:
Marc Smeets
Michiel van Veen
The project

• Research Question:
  How can a scalable, modular and extensible middleware solution be designed for the purposes of password cracking, so that it is based on existing cracking tools and allows for the use of a dynamic and adjustable cracking strategy?

• Why: The need for a distributed password cracking system, which can work with both CPU and GPU capabilities

• Approach: -Formulate system requirements
  -Research and creation of system designs
  -Proof of Concept

• Related Work:
  • KPMG's previous research projects
  • Other work
Making the scope clear

• **What we did:**
  • Use existing cracking tools
  • Set requirements and make a distributed system design which is scalable, modular and extensible
  • Develop the basis for such a design

• **What we didn’t do:**
  • Create our own cracking tool
  • Design of cracking strategy
Research & Creation

• Distributed Systems
  • Architectures
  • Communication

• Cracking Tools
  • CPU
  • GPU
  • Both

• System Design
  • Technical
  • Functional

• Proof of concept
System Overview

Users

Website

Front-End

Submission/Response

Cracking Controller

Controller

CPU Box
CPU Cluster
GPU Box 1
GPU Box 2
Future Technology

Workers
User Requirements

Open Web-Interface

Provide Login Credentials

(a) Task: New Job
(b) Task: Get Status
(c) Task: Stop Job
(d) Task: Show History
(e) Task: Delete Job
System Requirements

• **Front-end Functionality**
  - User Job Input
  - Current Job Status
  - Job History
  - Stop Job
  - Delete Job

• **Worker Functionality**
  - Register a controller
  - Status request handling
  - Job processing
  - Cracking tool support

• **Controller Functionality**
  - User input and request handling
  - Worker nodes control
  - Dynamic cracking strategy
  - User notifications
System Design

- System Architecture
- Communication
- Existing Cracking tools
System Architecture Design

Front-End
- Website

Controller
- Communicator
- Database
- Dispatcher

Workers
- Common Code
- Tool specific code
- Common Code
- Tool specific code
Communicator Workflow

- Provide Status
- Submit New Job
- Show History
- Cancel Job
- Delete Job

Listener

- Notify User

Worker Side Requests
- Register Node
- Worker Response
- Worker Ready

Worker Side Requests

Crack Successful?
- Yes
- No

Stop Other Workers
Dispatcher Workflow

1. **Startup**
   - Contact Database
   - Jobs Available?
     - Yes: **Node Checker**
     - No: **Exit**

2. **Node Checker**
   - Nodes Available?
     - Yes: **Determine Strategy**
     - No: **Create Subjobs**

3. **Exit**
4. **Dispatching**
Submitjob Example

1. User

2. Listener Accepts Job

3. Check Credentials

4. Put Job in DB

5. Any Job Available?

6. Any Node Available?

7. Determine Strategy

8. Create subjobs

9. Dispatch Subjobs

10. Subjob 1

11. Start Cracking (CPU)

12. Intermediate Updates

13. Send Result back

14. Worker Clean Up

15. Send Result or Status Request To User

16. Stop Other Workers

17. Cancel Job

18. Stop Worker and Clean Up

Done
Communication

• **Paradigms**
  • Remote Procedure Calls (RPC)
  • Message-oriented communication

• **Protocol**

• **Data Structures**
Communication Messages & Data

- **Protocol**
  - Controller Messages – requestStatus, deleteJob, etc.
  - Worker Messages – requestStatus, stopJob
  - Asynchronous RPC – submitJob, sendResults

- **Data Structures**
  - Reply
  - Hash
  - Job
  - Subjob

---

Example: **Subjob data structure**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int</td>
<td>The identifier of this subjob</td>
</tr>
<tr>
<td>hashtype</td>
<td>string</td>
<td>The name of the hashtype used</td>
</tr>
<tr>
<td>method</td>
<td>string</td>
<td>The name of the cracking method used</td>
</tr>
<tr>
<td>alphabet</td>
<td>string</td>
<td>The name of the alphabet used</td>
</tr>
<tr>
<td>submitted</td>
<td>long</td>
<td>The time of submission (Unix timestamp format)</td>
</tr>
<tr>
<td>percentage</td>
<td>int</td>
<td>The percentage of completed checks</td>
</tr>
<tr>
<td>minlength</td>
<td>int</td>
<td>The minimum length of the password</td>
</tr>
<tr>
<td>maxlength</td>
<td>int</td>
<td>The maximum length of the password</td>
</tr>
</tbody>
</table>
Cracking Tools

• **Existing cracking tools**
  • John the ripper (CPU)
  • oclHashcat-plus (GPU)
Proof of Concept - Overview

<table>
<thead>
<tr>
<th>Component</th>
<th>Progress</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Website</strong></td>
<td>Very simple</td>
<td><code>&lt;HTML&gt;</code></td>
</tr>
<tr>
<td>• Frond-end:</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Controller</strong></td>
<td>Finished</td>
<td><code>&lt;PHP&gt;</code></td>
</tr>
<tr>
<td>• Communicator:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Dispatcher:</td>
<td>Very simple strategy</td>
<td><code>&lt;PHP&gt;</code></td>
</tr>
<tr>
<td><strong>Worker</strong></td>
<td>Finished</td>
<td><code>&lt;PHP&gt;</code></td>
</tr>
<tr>
<td>• Common code:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Tool specific:</td>
<td>Basic John the Ripper</td>
<td><code>&lt;PHP&gt;</code></td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td>Finished</td>
<td><code>&lt;MySQL&gt;</code></td>
</tr>
<tr>
<td>• Controller:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Worker:</td>
<td>Finished</td>
<td><code>&lt;SQLite&gt;</code></td>
</tr>
</tbody>
</table>
Proof of Concept

• **Demonstration**
  1. Adding new node
  2. Show database with jobs
  3. Starting dispatcher
  4. Intermediate hashes cracked
  5. Job ready (result?)
  6. Worker Clean up / Ready again
Conclusion

• What was the research question again? 😊
  • How can a **scalable**, **modular** and **extensible** middleware solution be designed for the purposes of **password cracking**, so that it is based on **existing cracking tools** and allows for the use of a **dynamic** and **adjustable cracking strategy**?

• Research
  • **Distributed Architecture**: Centralized
    • Transparency
    • Modularity
    • Concurrency
    • Simplicity
  • **Communication**: Message-Oriented / RPC
  • **Existing Tools**: John the Ripper (CPU) / oclHashcat (GPU)
Project Achievements

- **Functional Specification:**
  - System overview
  - Use-cases
  - System requirements

- **Technical Specification:**
  - User interface
  - Controller
  - Worker
  - Database
  - Communication

- **Proof of Concept:**
  - **Website:** very simple
  - **Controller:** working with simple strategy
  - **Worker:** working with John the Ripper
Future work

• Further development / fine tuning of the system modules

• Extending to support other architectures (Cloud, Cell, etc.)

• Implementing the following for the system:
  • Adding more tools and hashtypes
  • Tweaking for multiple OS’s (small changes needed)
  • Proper cracking strategy
  • Security for controller/node communication
  • Development of a proper front-end

• Testing / Benchmarking with many workers
Any Questions?