Lecture 5: Humans and usable security

Seyyedeh Atefeh Musavi / Mehdi Kharrazi
Department of Computer Engineering
Sharif University of Technology
Spring 1400

Acknowledgments: Some of the slides are fully or partially obtained from other sources. A reference is noted on the bottom of each slide to acknowledge the full slide or partial slide content.
There's an old joke that computers are actually easy machines to secure: just turn them off, lock them in a metal-lined room, and throw away the key. What you end up with is a machine that is very secure, just not very usable.
What are the human factors in cyber security? and why are they important?
From information security to cyber security

- Where are the human factors?
- Remember the three pillars of cybersecurity?
  - Technology, processes, and **people**.
- The assets cyber security aims to protect include an additional dimension which extends beyond the formal boundaries of information security.
- Both humans in their personal capacity and society at large can be directly harmed or affected by cyber security attacks
  - This is not necessarily the case with information security where harm is always indirect.

Human factors in cyber security

- Humans are consistently referred to as the weakest link in security.
- Human factors influence how individuals interact with information security technology.
- Dynamic and complex! Many factors:
  - Influence of individual differences
  - Personality traits
  - Cognitive abilities
  - Biases and heuristics that affect how individuals perceive risk
- Important because they help explain why individuals make certain decisions and why specific behaviors may be observed.

Jeep Shifter Example

• Jeep’s shift level doesn’t mechanically control the transmission, even though it looks and moves like a traditional shift lever.
  • Fundamentally a software switch that controls the transmission electronically.
• The "Monostable" design doesn’t provide any meaningful feedback about what gear you’re in — it returns to the center position after each shift.
• LEDs on the shifter (often covered by your palm) or the digital display in the instrument cluster displays current position.
• Confusion for thousands of people!
  • Over a hundred injuries, and now potentially a death.
  • Because of a design that prioritizes screens over switches.

Human Factor Errors

• Reasons for information security breaches:
  • Acts of omission, in which people forget to perform a necessary action.
    • For instance, the failure to regularly change passwords.
  • Errors are commonly acts of commission, in which people perform an incorrect procedure or action,
    • i.e. writing down a password.
  • Extraneous acts, which involves doing something unnecessary.
  • Sequential acts, which involve doing something in the wrong order.
  • Time errors, caused by people failing to perform a task within the required time.

Why users fail to show the required behavior?

• Impossible Demands
  • Most users today find it impossible to comply with standard policies governing the use of computer passwords.

• Awkward Behaviors
  • User locks the screen of his computer every time he leaves the office, even for brief periods.
  • His colleagues likely suspect that the user either has something to hide or does not trust them.

[Cranor, L. F., Garfinkel, S., Security and usability: designing secure systems that people can use, O’Reilly Media, 2005]
Why users fail to show the required behavior? (con’t)

• Beyond the User Interface
  • Why Johnny Can't Encrypt.
  • Users' perception of the task of encrypting email vs. the way that the PGP interface presents those tasks to users.
  • User-centered design of security mechanisms, however, is more than user interface design.
  • A cryptographic key does not function like a key in the physical world.
    • People's understanding of "public" and "private" is different from how these terms are applied to public and private keys.

[Cranor, L. F., Garfinkel, S., Security and usability: designing secure systems that people can use, O'Reilly Media, 2005]
Human Factor Errors (con’t)

- Security behavior can also be described using a two-factor taxonomy:
  - Intentionality
  - Technical expertise

[Cranor, L. F., Garfinkel, S., Security and usability: designing secure systems that people can use, O’Reilly Media, 2005]
Usability discussions discuss on these

Usability discussions also affects these

How to Balance Security and Usability
Security Is a Supporting Task

• Key to designing successful security applications.
  • Goals and tasks
  • Human behavior \(\rightarrow\) goal driven
  • Production tasks
    • Required to achieve the goal or produce the desired output.
  • Supporting tasks
    • Enable production tasks to be carried out in the long run.
    • Or be carried out more efficiently, but not essential.
  • Security tasks must be designed to support production tasks.
    • Should not conflict with production tasks (e.g. One’s performance)

[Cranor, L. F., Garfinkel, S., Security and usability: designing secure systems that people can use, O’Reilly Media, 2005]
Psychological Acceptability

• Security mechanism should not make accessing a resource more difficult, compared to when not present.

• In practice, a security mechanism should add as little as possible to the difficulty of the human performing some action.

[Image: Bauer, L., et al., Lessons learned from the deployment of a smartphone-based access-control system. SOUPS, 2007]

[Image: Cranor, L. F., Garfinkel, S., Security and usability: designing secure systems that people can use, O’Reilly Media, 2005]
Exploit Differences Between Users and Bad Guys

• Dots when typing a password to protect from "shoulder surfing".
• User would tend to pick an easier pass in order to avoid typos.
• Different perspectives:
  • The user is close to the screen
  • Eavesdropper is probably several feet away from the screen.
• Hence produce an interface that promotes complex passwords, while still leaving the eavesdropper in the dark.
• Done in design of Tresor 2.2

[Cranor, L. F., Garfinkel, S., Security and usability: designing secure systems that people can use, O'Reilly Media, 2005]
Case study: Tresor2

- Tresor is a high security file encryption application.
- Makes it easy to type a long “passphrase" even if you make typos.
- As you type the password dots appear with a delay.
  - Revealing the last few characters for a few seconds as the user types, long enough for the user to catch a typo.
- Pressing Delete would delete the last character and reveal one more so that three would always be visible.
- Users can have longer passwords more easily
What else?

- Exploit Differences in Physical Location
  - Our current "one size fits all" security systems tend to ignore that difference.
  - They arise from a single assumption: the bad guy may be standing behind you this minute!
- Vary Security with the Task.
- Increase Your Partnership with Users.
  - Trust the user.
  - Exploit the special skills of users.
  - Remove or reduce the user's burden.

[Cranor, L. F., Garfinkel, S., Security and usability: designing secure systems that people can use, O'Reilly Media, 2005]
What else?

• Balance Resource Allocation
  • For example resource allocation for Cybersecurity in the absence of physical security is useless.
  • Asking Emergency to fax a few pages from the record of a patient they had just sent up is refused today.
  • However, someone could steal the fax off the machine that sits right out in the hall before, with easy patient access.
  • The previous week, that was an acceptable risk. This week (New security regulations for hospitals ), it was against the law.
• Any new (security) project should be launched with a thorough field study of the people who will be using the system, the places where they will use it, and the nature of the tasks they will be accomplishing.

[Cranor, L. F., Garfinkel, S., Security and usability: designing secure systems that people can use, O'Reilly Media, 2005]
Usability & Visibility

• Visibility is a powerful tool for aligning security and usability.
• Hidden properties, functionality, or data storage that is part of a complex system can make it more complex to use (less usable).
• So what to do?
  • Teaching users about hidden aspects of a system with significant effort.
  • An attractive alternative is to remove the opportunities for a system's visible state to be inconsistent with its internal state.

[Cranor, L. F., Garfinkel, S., Security and usability: designing secure systems that people can use, O'Reilly Media, 2005]
Code smells

• Code smells are indicators that a piece of software code may be of lower quality than desired.
• The code may not be broken, but violating a design principle and may be fragile and prone to failure.

Examples

**Change Preventers**
These smells mean that if you need to change something in one place in your code, you have to make many changes in other places too. Program development becomes much more complicated and expensive as a result.

**Object-Orientation Abusers**
All these smells are incomplete or incorrect application of object-oriented programming principles.

**Couplers**
All the smells in this group contribute to excessive coupling between classes or show what happens if coupling is replaced by excessive delegation.

[https://refactoring.guru/refactoring/smells](https://refactoring.guru/refactoring/smells)
Usability smell

• A usability smell is an indicator that an interface may be difficult to use for its intended users.
• There have been multiple studies on usability smells in:
  • Graphical user interfaces
  • Library APIs (why?)
• The idea: the more usable API, The fewer questions about the basic usage
• Looking at a developer Q&A site, such as Stack Overflow
• 2,491 Stack Overflow questions to study about seven cryptographic libraries have been analyzed.

Thematic analysis

- Identifying 16 thematic issues and measure their prevalence across the different libraries.
- Relating these issues back to green and smith’s usability principles
- Identifying four usability smells

[Image: M. Green and M. Smith. Developers are not the enemy!: The need for usable security APIs. IEEE S&P, 2016]

The 16 issues identified through a thematic Analysis of Stack Overflow Questions

- **Usage issues**: Should I use this?  How should I use this?
- **System issues**: Build issue  Performance issue  Passing the buck  Lack of knowledge  Unsupported feature  Borrowed mental models
- **Not knowing what to do**
- **Not knowing if it can do**

**Missing information**
- Looking for example code
- Clarity of documentation

**Programming is hard**
- What's gone wrong here?
- API misuse
- Compatibility issue
- Deprecated feature

**Issues across time and space**

# Final usability smells

<table>
<thead>
<tr>
<th>Whiff</th>
<th>Issue</th>
<th>Abstract</th>
<th>Powerful</th>
<th>Comprehensible</th>
<th>Ergonomic</th>
<th>Intuitive</th>
<th>Failing</th>
<th>Safe</th>
<th>Testable</th>
<th>Readable</th>
<th>Explained</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need a super-sleuth</td>
<td>Missing Documentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Clarity of documentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Confusion reigns</td>
<td>Should I use this?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>How should I use this?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Abstraction issues</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Borrowed mental models</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Needs a post-mortem</td>
<td>What’s gone wrong here?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unsupported feature</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>API misuse</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Deprecated feature</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Doesn’t play well with others</td>
<td>Build issues</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Compatibility issues</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Performance issues</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Why Can't Johnny Fix Vulnerabilities

- Static analysis tools enable developers to detect issues early in the development process.
- How usable are they?
- Static analysis tools can help prevent security incidents.
  - They must enable developers to resolve the defects they detect.
  - Unfortunately, developers often struggle to interact with the interfaces of these tools.
- Leading to the proliferation of preventable vulnerabilities.

Prerequisite: Some methods to evaluate the usability

• Two popular methods to evaluate the usability of user interfaces:
  • Heuristic evaluations
    • Most informal form.
    • Some experts judge about usability.
    • There are some principles (e.g. visibility, user control/freedom) to judge.
  • Cognitive walkthroughs
    • Cost-effective testing.
    • Is a user able to do a task easy? (task-driven)
    • There are some questions as guideline.
• There are also other methods:
  • Consistency inspection, Pluralistic walkthrough, etc

[ Nielsen, J., Usability inspection methods. In Conference companion on Human factors in computing systems, 1994 ]
Methodology

• Two-phase method that combines the strengths of two usability evaluation techniques:
  • Cognitive walkthrough: evaluators simulate the tasks that real users would perform with a system.
  • Heuristic evaluation: evaluators systematically examine a system following a set of heuristics (as opposed to the task-driven approach in a cognitive walkthrough).
• Two evaluators
• User-study

Phase 1: Task-Oriented Evaluation

- Particular task in mind: fixing as many errors as possible in a limited time.
- Following guidelines have been used:
  - Choose a vulnerability to inspect first.
  - Determine whether it is a true positive or a false positive.
  - Propose a fix to the vulnerability.
  - Assess the quality of the fix.
- To help us think critically about each tool, we used Sears’ list of guiding questions (look in the paper)
Phase 2: Free-Form Evaluation

- Where evaluators freely explore an entire system using a set of usability heuristics to identify issues.

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preventing &amp; Understanding Potential Attacks</td>
<td>Information about how an attack would exploit this vulnerability or what types of attacks are possible in this scenario.</td>
</tr>
<tr>
<td>Understanding Alternative Fixes &amp; Approaches</td>
<td>Information about alternative ways to achieve the same functionality securely.</td>
</tr>
<tr>
<td>Assessing the Application of the Fix</td>
<td>Once a fix has been selected and/or applied, information about the application of that fix or assessing the quality of the fix.</td>
</tr>
<tr>
<td>Relationship Between Vulnerabilities</td>
<td>Information about how co-occurring vulnerabilities relate to each other.</td>
</tr>
<tr>
<td>Locating Information</td>
<td>Information that satisfies &quot;where&quot; questions: Searching for information in the code.</td>
</tr>
<tr>
<td>Control Flow &amp; Call Information</td>
<td>Information about the callers and callees of potentially vulnerable methods.</td>
</tr>
<tr>
<td>Data Storage &amp; Flow</td>
<td>Information about data collection, storage, its origins, and its destinations.</td>
</tr>
<tr>
<td>Code Background &amp; Functionality</td>
<td>Information about the history and the functionality of the potentially vulnerable code.</td>
</tr>
<tr>
<td>Application Context / Usage</td>
<td>Information about how a piece of potentially vulnerable code fits into the larger application context (e.g., test code).</td>
</tr>
<tr>
<td>End-User Interaction</td>
<td>Information about sanitization/validation and input coming from users. Does the tool help show where input to the application is coming from?</td>
</tr>
<tr>
<td>Developer Planning &amp; Self-Reflection</td>
<td>Information about the tool user reflecting on or organizing their work.</td>
</tr>
<tr>
<td>Understanding Concepts</td>
<td>Information about unfamiliar concepts that appear in the code or in the tool.</td>
</tr>
<tr>
<td>Confirming Expectations</td>
<td>Does the tool behave as expected?</td>
</tr>
<tr>
<td>Resources &amp; Documentation</td>
<td>Additional information about help resources and documentation.</td>
</tr>
<tr>
<td>Understanding &amp; Interacting with Tools</td>
<td>Information about accessing and making sense of tools available, including, but not limited to the defect detection tool.</td>
</tr>
<tr>
<td>Vulnerability Severity &amp; Rank</td>
<td>Information about the potential impact of vulnerabilities, including which vulnerabilities are potentially most impactful.</td>
</tr>
<tr>
<td>Notification Text</td>
<td>Textual information that an analysis tool provides and how that text relates to the potentially vulnerable code.</td>
</tr>
</tbody>
</table>

## Results

<table>
<thead>
<tr>
<th>Theme</th>
<th>Subtheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Missing Affordances</td>
<td>Managing Vulnerabilities Applying Fixes</td>
</tr>
<tr>
<td>4.2 Missing or Buried Information</td>
<td>Vulnerability Prioritization Fix Information</td>
</tr>
<tr>
<td>4.3 Scalability of Interface</td>
<td>Vulnerability Sorting Overlapping Vulnerabilities Scalable Visualizations</td>
</tr>
<tr>
<td>4.4 Inaccuracy of Analysis</td>
<td>Mismatched Examples Immutable Code</td>
</tr>
<tr>
<td>4.5 Code Disconnect</td>
<td>Tracking Progress Batch Processing</td>
</tr>
</tbody>
</table>

Figure 5: Scalability of RIPS’ function view.

Figure 6: RIPS call graph visualization for more than 50 files.

Further reading