Initial Usable Security Audit Methodology

contact@simplysecure.org

March 2017

Simply Secure

Introduction

Security and usability share the property of being non-binary: it is rarely possible to say with certainty that a given software program is or is not secure, or that it is or is not usable.

This is in part because both the security and user-experience communities define success in relation other elements. We define security in function of a particular threat model: is a program secure against threats A, B, and C? Similarly, we define usability in function of a particular set of human users: are these particular people able to independently accomplish essential tasks within the software without becoming frustrated or burned out?

In defining a methodology for performing a usable security audit, we must therefore center the analysis on both the threats that the software seeks to protect against, and the people it seeks to help.

We focus this document specifically on the analysis of an individual software tool that already exists to some degree (even if just in mockup or prototype fund). We leave to other documents a description of the broader set of user-experience (UX) and design activities that can support open-source software teams that are developing tools in the Internet Freedom ecosystem[1], and earlier-stage activities such as needfinding[2].

Additionally, we focus this document on the user-experience / usability aspect of such an audit. The process of performing security audits (both at the architectural and code levels) is well understood and widely practiced. For tools in the Internet Freedom ecosystem, and more broadly for tools that consider security and privacy preservation to be a core part of their value proposition, we believe that a comprehensive security audit is essential.

Audit Goals

The ultimate goal of a usable-security audit is to help a software team make its tool more usable and useful for its target users, while at a minimum helping preserve – and ideally enhancing – those users' privacy and security.

This means:

1. Understanding who are the tool's desired, likely, and/or actual users (or "target" users)
2. Understanding relevant threats:
    a. The threats users are likely to face
    b. The threats that the tool seeks to mitigate

[1] E.g. our *Office Hours Process Proposal* to OTF, January 2017
[2] E.g. Second Muse's Needfinding Framework

    c. The threats that the tool cannot mitigate against, but that users might expect it to
3. Analyzing the tool's user experience heuristically (i.e. through an expert review)
4. Analyzing the tool's user experience empirically (e.g. through user studies)
5. Making actionable recommendations to the software team to help them improve the tool's design in response to review and study results

Phases and Outputs

A good review will not be transactional, but cooperative. This means that software team doesn't just receive a report of problems or things to fix, but that there is an iterative exchange of insights and improvements over time. Good software should be viewed not as a product, but as a process; a good UX analysis will support that process and flex to accommodate a team's changing understanding of their users and their goals.

Phase 1: Understanding background

The first phase is to understand the background goals of the tool. This includes, as noted in the previous section, identifying the tool's target users and its threat model. It also includes understanding the tool's position in its competitive landscape, and the development team's priorities for UX improvement.

Because the software team will often be more familiar with the niche their tool occupies, the analysis of this phase involves more of a cooperative dialogue with the team than the other two phases. While UX and security experts can offer advice on e.g. who their users are likely to be or what threats those users are likely to face, it is critical that the team agree and be fully bought in to the way these questions are answered. There will also be situations where the answer depends entirely on the team's priorities – e.g., choosing to focus on one geographic region over another.

Identifying target users

It is essential that the software and evaluation teams develop a clear picture of who the target users are for the tool. This includes most importantly the users that the team would like to encourage to use it. The definition of target users should additionally include users who are *likely* to use the tool (or who are currently using it), even if they are outside of the group that the team would like to focus on.

While the desired user population should be the highest priority in evaluating usability, entirely ignoring probable or actual user populations will result in frustration for the development team when they receive a high volume of complaints from this "extra" population. Ignoring likely but unexpected users of tools with a security or privacy focus can also cause these users to be endangered when the threats they face are not the threats the tool protects against, if the tool does not help them understand this.

In working with software teams we like to develop a wide-ranging list of potential user characteristics and assign priorities to different populations described by them. We then ask teams to assign priorities to each population, and collaboratively estimate places where lower-priority groups may be nevertheless likely to use the tool. (This process can be assisted by any data the team has about its user base if the tool has already been released.)

Relevant user characteristics include:

- Language and geographic locale
- Level of general technical expertise or willingness to independently troubleshoot
- Familiarity with technical or domain concepts relevant to operation of the tool
- Reasons or motivations for using the tool (users of a circumvention tool may have very different usability requirements if they primarily download scientific papers or if they primarily stream foreign TV shows)

Scoping threats

Internet Freedom tools focus on providing certain security and privacy guarantees to their users. Most development teams have thought explicitly about what these protections are (and what protections their tool does not offer), and it is usually easy for them to enumerate them. It is usually harder for them to make assertions about the threat models that target users are likely to have, but this step is important to understanding the usable security of the tool. If the tool does not meet the security expectations of the user, it doesn't matter how many other protections it does offer – users will either choose to not use the tool, or are potentially misled into thinking that they are safe against real and significant threats.

A usable security evaluation may turn up new security or privacy features that the developers should consider implementing to better accommodate their users' threat models, and is likely to result in recommendations for small changes that help users engage with the tool in a way that will produce the desired security and privacy properties. However, the area that usually includes the most recommendations is the communication with users about security and privacy concepts and features (i.e., text, images, and help documentation).

Benchmarking and competitive analysis

No software exists in a vacuum. Even if a tool is targeting a severely underserved user population or trying to do something technically unique, it is important to understand the ecosystem that the tool exists (or will exist) in. Even if it just helps understand expectations or experiences that users might bring, developers of even the most niche tools can identify relevant or competitive projects that are relevant to their work. Reviewing this set of comparative projects will help evaluators be fresh on current design patterns for the space, and help put them in the mindset of target users.

Evaluation priorities

Because it may not be possible to evaluate the entire tool, it is important that the software team and evaluators agree on what elements of the user experience are most important to review. Even if there is time and budget for a thorough evaluation, explicit agreement on priorities will help evaluators understand where developers are focused, and may provide an opportunity to discuss whether those priorities are appropriate given the target user population.

Top priorities will commonly include the onboarding experience for new users and the handful of most common actions taken within the app.

Phase 2: Heuristic evaluation (expert review)

Once background information has been established, the fastest, easiest, and most economical way to get feedback on a tool's usability with respect to security is to perform an expert review. Expert reviews generally consist of one or more UX professionals walking through the tool's various features and flows to identify points where they believe that users are likely to have various different types of trouble (aka "pain points").

An expert review can be highly structured according to formal heuristics, such as Nielsen's ten principles for interaction design (see below), or can be more informal and based on the reviewer's instinctive understanding of UX principles that they have developed through years of work in the field. Both approaches have their benefits and detriments. We generally tend toward a less formal approach, as it allows us to focus on delivering what we consider to be a rigorous prioritization of the most important (i.e. painful) elements of the UX, rather than an exhaustive catalog of all potential issues. That said, heuristics such as Nielsen's principles can be highly useful in helping the development team come up with a solution to specific problems, and in helping them develop an understanding of concepts that will in turn empower them to improve their UX independently.

Jakob Nielsen's 10 general principles for interaction design[3]:

1. Visibility of system status
2. Match between system and the real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition rather than recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Help users recognize, diagnose, and recover from errors

[3] https://www.nngroup.com/articles/ten-usability-heuristics/

10. Help and documentation

In performing an analysis not just of usability but of usable security, evaluators must additionally have expertise with both the relevant security or privacy design patterns and common pitfalls. This expertise, and the ability to review the app in the context of background information about the teams' and target users' threat models, is what distinguishes a usable security review from a traditional UX review.

Where possible the evaluation team should include members familiar with the target user populations, although a lack of this expertise can generally be compensated for by empirical evaluations with target users.

The output of a heuristic evaluation is a list of pain points. Each pain point should include information on the relative priority of the problem (e.g., "must fix before launch", "fix soon", "fix someday"), details on why it is painful ("users will expect X and be frustrated when they get Y"), and suggestions for improvement. In some cases it will not be possible for evaluators to directly suggest solutions, in which case they should aim to characterize what an improvement would look like (e.g., "this dialogue box should help the user understand A in addition to B").

We strongly recommend that evaluators and software teams take the time to meet and go through the results of an expert review rather than relying purely on a report. This makes sure that the developers are able to fully understand the results of the review. It also allows evaluators to talk through the reasons for their recommendations, and help to build empathy and insight into the probable experiences target users would have with the current design.

Phase 3: Empirical evaluation

After one or more expert reviews have been completed, the next step in evaluating and improving a user experience is to put it in front of real users. The possibilities for such work are potentially endless, so it is important to work with the software development team to identify top questions that they would like to answer. For evaluating the usable security of a project, this should include at least some focus on whether users accurately understand the threat model that the app addresses and whether they can perform key security-related tasks with comprehension. (For example, it's not enough for a user to be able to successfully generate a key if she does not understand what that key is for or how she should manage it in the future.)

There are a host of empirical methods that can be used to evaluate various aspects of an app (see below for some of the more common forms). We generally recommend starting with a cognitive walkthrough with a small number of users – about five is usually a good starting place, and definitely not more than ten to begin with. From there a set of task analyses or semi-structured interviews can probe more specific elements or features of the tool. Diary

and/or survey studies may prove useful, although can consume a lot of time and resources and should only be undertaken with specific goals in mind.

As with an expert review, the basic form of output for an empirical evaluation is a report. However, most forms of empirical evaluation present a tremendous opportunity for developers to get direct insight into users' experience using the tool. Watching videos or sitting in on a live interview (after a small amount of coaching on how to engage in active listening without being tempted to educate the user!) is the single best way to help developers get out of their own assumptions and into the shoes of their target users.

Example forms of user studies

- **Cognitive walkthrough** – Similar to a task analysis, but with a greater focus on the user's experience using the software rather than their structured outcomes. Requests the user to give voice to the interior monologue as they explore the software or perform specific tasks. Gives greater insight into their emotional state and why they experience problems. Particularly useful for probing a user's understanding of security concepts or features as presented in the software. Often augmented by screen capture and/or video capture.
- **Semi-structured or group interview** – Often used to get an even more qualitative understanding of users' reactions to or understanding of a piece of software and its individual features. Well suited for reviewing initial tool presentation (e.g. webpage or app store page) and specific static elements of a tool, such as a product's name, a diagram on a help screen diagram, or the wording of a dialogue box. Often augmented by video recording.
- **Task analysis** – A classic "usability" study, where users are asked to perform specific tasks with a live version of the software. Results can be more quantitative ("X user succeeded after Y seconds", "Z users failed", etc.) or qualitative ("Users seemed to expected functionality A to live in menu B" etc.). Often augmented by screen capture, video capture, and/or eye tracking software.
- **Diary / prompted in situ study** – Most useful for understanding how a small number of users experience a tool over a longer period of time. Asks users to regularly reflect and record on their experiences. Good for evaluating more than just initial impressions. Sometimes augmented with automated prompts to submit diary entries or answer questions, such as via text message or a special-purpose phone app.
- **Survey or questionnaire** – Useful for for exploring a variety of topics with a larger number of users. Quality of data can be lower, especially on longer surveys. Can be hard to recruit for. Well-suited to getting users' perceived preferences across alternative solutions (e.g. three different product names). Carefully-constructed surveys can be used to simulate task analysis or semi-structured interviews for remote participants.

Additional sources of data

In addition to a user study, there are other potential sources of direct information from users that a software team may have access to.

- **Instrumentation** – Where the software is designed to "phone home" to developers with information about how it is being used. This form of data-gathering is much more common outside of the Internet Freedom community where security and privacy expectations are somewhat less strenuous. If data is available from instrumentation, it is incumbent on the evaluators to make sure that the data is being collected and handled responsibly – including getting users' explicit consent before it is gathered – before using it to inform UX decisions.
- **Support channels and social media** – If a tool has been released for some time and has an existing user base, chances are good that someone somewhere on the internet is talking about it. Support channels, user forums, and social media accounts can be rich sources of information from actual users. However, if using volunteered data of this nature, evaluators must work with development teams to rigorously ascertain whether the picture it paints of users' experiences is reflective of 1) the user population as a whole, and 2) the desired user population. This is especially true if the support data is coming in through an expert-friendly channel like GitHub.

Notes on remote evaluations

Depending on the nature of the tool and the team's target user population, remove evaluations may be an essential part of an empirical usability analysis. This may be because members of the target population are hard to connect with in person, or because aspects of the tool's performance may depend on operating in a hard-to-reach locale.

When possible, we generally recommend asynchronous (e.g. survey- or diary-based) evaluations for remote studies, or a combination of asynchronous and synchronous (i.e., interactive interviews via phone or video chat) evaluations. Conducting remote studies purely via live synchronous interaction can be a frustrating experience for users in low-bandwidth situations, can make it difficult to communicate without high-fidelity screen sharing, and can be especially challenging in situations where there is a language barrier. A primarily asynchronous structure forces the software team and the evaluators to document the steps they would like the study participant to go through in a way that will be accessible to the study participant independent of their bandwidth access. It also encourages the team to employ in-person evaluations for general usability evaluation, and focus remote sessions on elements that are specific to the target population in question. A brief synchronous component – e.g., an introductory phone call before the user goes through the study and a brief phone interview when they have completed it – can help to personalize the process, increase participant commitment to completing the tasks, and make sure that they are comfortable with the process.

Depending on the nature of the tool, its stability, and the complexity of setup, we encourage teams to consider conducting remote evaluations on a web-based mockup or prototype rather

than the live tool. We have had reasonably good experiences using tools like Invision[4] to help users experience essential elements of a tool's interaction without having to go through a laborious installation process.

A note on order and timing

Software teams are regularly challenged by competing priorities, and aligning various activities can be difficult – especially when the outcome of one activity can result in changes to the codebase. This is certainly true of security reviews and user-experience reviews. Bearing in mind the types of changes that are likely emerge from each type of review, we recommend the following ordering of activities.

1. Iterative design review and redesign of the UX (or planned UX, for new projects)
2. Iterative security review and redesign of the (planned) software architecture
3. Design check-in on any changed elements of the UX
4. Prototype or interactive mockup of the new UX
5. Iterative user study and implementation of the new UX
6. Iterative security review and changes to the codebase

A Note On Openness and Tone

The Internet Freedom community considers open-source software to be more trustworthy than closed-source software because it is possible for people outside of the software team to identify problems in the codebase.

Well-resourced teams also regularly publish the results of security audits, which help the community build trust in the software even if they don't (or can't) review the code themselves. Similarly, Simply Secure believes that it is important for open-source software teams to be as open as possible about their design process and priorities. While UX design does not have the same impact on safety as a security review, it can help the community understand and appreciate the decisions the team has made, especially in cases where they feel like the tool does not work perfectly for them. (This is especially useful in cases of highly technical users being frustrated with design decisions that were made to support people with less technical expertise.)

At a minimum, we encourage teams to track evolving UX designs and discussions in their public bug-tracking system. We consider it ideal if the team is willing to publish all results of the UX reviews and user studies as well. However, we recognize that this can be an intimidating prospect, especially for technical teams that have no UX members on staff. It is thus important that audit reports be framed in such a way as to not embarrass or shame software developers if they are published publicly. This includes, for example, selecting excerpts from user interviews

[4] https://www.invisionapp.com/

that are reflective of users' experiences but that are not too negative or pejorative. As we seek to have empathy for users, so should we also seek to have empathy for the developers that are trying to serve them. User-experience design is not easy, and we should strive to reflect in our reports the hard work that developers do in that domain. In this manner we hope to encourage tool developers to make their UX processes transparent, and thus benefit the broader ecosystem through their example.