# STRAP: A Structured Analysis Framework for Privacy

Carlos Jensen, Joe Tullio, Colin Potts, Elizabeth D. Mynatt
Graphics, Visualization and Usability Center
Georgia Institute of Technology
Atlanta, GA 30332, USA
+1-404-385-1102

[carlosj, jtullio, potts, mynatt] @cc.gatech.edu

## ABSTRACT

Privacy is an important concern for users, and a difficult design challenge. Different user populations have different requirements and expectations when it comes to privacy; thus finding universally acceptable solutions is far from trivial. Design guidelines have been available for a number of years, but often fail to address the dynamic and impromptu nature of privacy management. These methods also fail to provide a robust and replicable procedure for identifying potential problems, leaving the design process more in the realm of art than science. We identify general requirements for privacy-aware design and review how existing methods and guidelines meet these requirements. We then introduce a light-weight method adapted from the requirements engineering literature for the structured analysis of privacy vulnerabilities in design and the iterative adaptation of preferences. We present a study of this method on a predictive group calendar system.

## Author Keywords

Design, Privacy, CSCW, Goal-oriented Analysis, Structured Analysis, Groupware, Calendar Systems, e-commerce.

## ACM Classification Keywords

H5.2 [**User Interfaces**]: Theory and Methods. D2.1 [**Requirements/Specifications**]: Methodologies. K.4.1 [**Public Policy Issues**] – Privacy

### INTRODUCTION

People are concerned about their online privacy, as reflected in numerous surveys [12, 14]. Such concerns are understandable given the growing number of privacy invasions [31], and the increasing pervasiveness of information capture and sharing between IT systems. This pervasiveness of information capture and sharing facilitated by today's networked systems has in part triggered a growing interest and awareness among users about privacy management, and an increase in the legislation aimed at protecting privacy [34, 35, 36].

These developments are important to us as designers because they impose limitations on the types of systems we may build, and impose requirements on how they must operate. This is especially important for computer supported collaborative work (CSCW), e-commerce, and online environments, where the very nature of the system requires some information disclosure. Privacy will always be an important consideration in designing such systems, and may seriously affect their adoption and ultimate success.

In addition to requiring the collection and dissemination of information, these systems present other challenges to privacy management; their boundaries are sometimes unclear, especially to the casual user. Who is collecting what information about them, what other information is this combined with and from what sources, what is done with their information, and who is it shared with? Information in computer systems is seldom ephemeral; it accumulates and aggregates without the users' knowledge.

Many of these challenges are fundamentally classic HCI problems: customization, situation awareness, decision support, and adequate controls. Whitten and Tygar found that in the context of security, well-engineered systems may fail or be circumvented because of inadequate usability [37]. The same is likely true for privacy management, which is seldom an active goal for the user. It is more often the case that privacy becomes a concern only after some violation has occurred or is suspected. This has important implications for design; we need to minimize the burden and distraction of doing privacy management. Failing to do so will result in the user neglecting this task in favor of their other goals, at least until the damage is done.

In response, heuristics and guidelines have been proposed to help guide design [6, 7, 21, 24]. As in security, non-trivial problems are often caused by high-level architectural problems that affect many aspects of a system. These architectural problems are hard to identify before a system is built, and both difficult and costly to address after the

fact [3]. Thus, it is important to identify potential privacy vulnerabilities at the design stage to minimize potential damage and maintenance costs.

Guidelines provide support to designers by giving them a framework for detecting and addressing potential privacy violations before the system is implemented and deployed. As we will show, these heuristics, though building on a wealth of experience, fall short in some respects The most important shortcoming is that they imply the existence, or the desirability of seeking a universally satisfactory solution.

Palen and Dourish describe privacy management as managing disclosure – allowing users to manage and maintain separate public and private spheres [28]. From a design perspective, not only do preferences and expectations vary greatly between individuals and cultures, but privacy management is a highly dynamic process. If one accepts this view of privacy, it follows that for non-trivial systems it will be impossible to derive a universally acceptable solution. The goal instead shifts to identifying potential vulnerabilities, identifying solutions where possible, and mitigation strategies where not possible.

In addition, heuristic-based approaches to privacy-aware design often depend on the expertise of the evaluator(s) in order to successfully identify vulnerabilities. Even with skilled evaluators, the unstructured form of heuristic analysis means that potential areas of concern may be overlooked or given less treatment than others. Moreover, evaluative skills may not transfer adequately from one type of system to the next. What is needed, then, is a structured, robust approach to privacy analysis that mitigates variance in expertise and knowledge transfer between systems.

In this paper we examine some of the most influential frameworks for privacy-aware design to see how they meet these challenges. We then present a light-weight structured analysis of privacy vulnerabilities (STRAP) that builds on these frameworks while borrowing methods from requirements engineering and goal-oriented analysis [13, 15, 30]. We discuss how STRAP fits into the software development cycle, including how it supports iteration and adaptation. We wrap up with a comparison between the performance of STRAP and other frameworks for a predictive group calendar system.

## BACKGROUND AND RELATED WORK

Computers became an important part of the privacy debate in the 1960's following the widespread adoption of computers, or "giant brains" by business and government in the US. In 1973, the U.S. Department of Health Education and Welfare's (HEW) Advisory Committee on Automated Personal Data Systems presented a report that defined for the first time a code of Fair Information Practices (FIPs) [10]. The FIPs have since widely influenced legislation and practice, especially privacy policies.

There have been a number of studies examining policies. The results of these studies show that privacy policies are increasingly popular among companies, yet provide little value to users. Policies were seldom consulted, even when accessible [22]. Possible explanations include the fact that policies did not focus on the issues users cared about [4], and used intimidating or difficult language [22]. In order to overcome the barriers of language, interpretation, and unnecessary burden, efforts have been made to develop machine-readable policies, such as P3P [11], and the automated agents which would act upon such policies [2]. Such policies could be automatically parsed by an agent, only interrupting the users if the policy is likely to concern or trouble the user. By filtering which policies, and even which elements of a policy require attention, users are more likely to be engaged.

Although external policy statements are often the only source of information on privacy practices available to users, it is also important to examine how policy is implemented. After all, it makes little difference how accessible and usable a policy is if the software does not follow the policies specified. The relationship between policy and code is explored in Lessig's CODE [25], where he describes source-code as an instantiation of policy. Code and programs are a set of rules which the computer, and by extension the user, must follow in order to accomplish a set of goals. Some of these policies are planned and intentional, while others are the product of ad-hoc and potentially inconsistent implementation decisions. Still others are artifacts of the tools and underlying infrastructure.

Because of the evolving nature of both policy and code, there have been efforts to decouple the two. The IBM Tivoli system uses the EPAL specification language [5] to allow policy makers to specify and enforce policies regarding data access in applications. In order to access data, the application must go through a gatekeeper, which ensures that the policy is enforced. Though this approach adds some overhead, it frees the developer from verifying that the policy is up to date and being enforced. Policy-makers are similarly free to modify data policies as needed without having to modify the applications.

CSCW systems by their very nature usually imply the disclosure of information by individuals for consumption by a group. Whether in the interest of maintaining awareness of coworkers, providing context for collaborative activities, or promoting distributed collaboration, disclosure of personal information is often necessary or adds value to the group. Grudin notes that in the case of networked applications, we introduce new possibilities for actions that are de-situated and may have unknown consequences beyond the user's control [17].

Identity, location, and activity are examples of common information types that can both facilitate group work and expose users to privacy invasions.

In ubiquitous computing applications, the sensing, recording and application of personal information, possibly without the users' knowledge, raises critical privacy concerns. Bellotti and Sellen stress the importance of feedback and control over information capture, access, purpose, and construction [6]. Similarly, Abowd and Mynatt describe the challenges of designing collaborative environments where actions and roles are dynamic [1].

Privacy has also been an important concern in groupware calendar systems (GCS), most extensively studied in the work of Palen [29], Grudin [16], and collaborations between the two [19, 28]. While their studies were not specifically aimed at studying privacy, they found that it has a significant impact on the adoption, use, and evolution of groupware calendar systems. We conclude this paper by performing a small evaluation on a predictive GCS.

Grudin found that users with no people-management responsibilities regarded calendar sharing as an invitation to be micromanaged, and therefore preferred to only share information about free/busy time. Executives refrained from sharing their schedules due to the sensitivity of the information they contained. On the other hand, managers and administrators exhibited the highest degree of sharing, characterized by high levels of trust and perceived benefit. These benefits include the ability to determine the availability of colleagues, the assimilation of organizational knowledge, and the ability to keep more flexible work hours. [16]

The success of groupware systems, as in most other things, seems to depend largely on the degree to which the individual benefits of contributing outweigh its costs [19]. Palen and Dourish describe these tradeoffs as the resolution of tensions between the need to be part of the world and receive some benefit, and the need to shield and protect oneself and ones personal life [27]. In the case of calendar systems, Grudin writes:

> Just as people who live in buildings with paper-thin walls may adopt a convention of ignoring what they cannot help overhearing, people who allow open access to their calendar details assume that people will access information only when needed and would be offended by an inquiry that revealed "snooping." Being able to block off a calendar entry or reserve a conference room is deemed an adequate balance. Privacy is ultimately a psychological construct, with malleable ties to specific objective conditions [19].

While workplace habits and individual experience may motivate users to share their schedules, mechanisms must still exist to manage and protect ones privacy. The frequency with which such mechanisms are used seems to be less important than the fact that they exist, and the impact they have on risk perception. Such mechanisms may be as simple as the ability to omit sensitive events, give cryptic or context-sensitive names to events, to enable reciprocity of access settings, or to schedule defensively to regulate interruption [29]. More explicit mechanisms such as access-control lists or levels of disclosure are also possible.

These GCS studies also found that the default settings for privacy preferences are not typically changed, and may over time become institutionalized. In one example, the ability to add events to a colleague's calendar, while surprising to new employees, was eventually accepted due to its widespread use. Grudin and Palen compared an organization whose shared calendar system defaulted to openness, with one whose calendar showed only free/busy blocks, finding they resulted in very different norms [18].

## PRIVACY DESIGN BY HEURISTICS

### Privacy Design Methods

The first set of privacy guidelines, the HEW FIPs [10], have survived and evolved over the years. These principles were originally derived from expert testimony and defined a set of ethical guidelines for developers and designers. The most recent version was given by the Federal Trade Commission (FTC) in their 2000 report on the state of online privacy [31]. This list, a simplification of the original principles given in the HEW report, defines the FIPs as the following 4 principles:

1. **Notice/awareness** *"Consumers should be given notice of an entity's information practices before any personal information is collected from them."*
2. **Choice/Consent** *"[…]giving consumers options as to how any personal information collected from them may be used."*
3. **Integrity/Security** *"[…] data [should] be accurate and secure."*
4. **Enforcement/Redress** *"[…]privacy protection can only be effective if there is a mechanism in place to enforce them."*

Though groundbreaking, these are by far the most high-level and abstract of guidelines, providing little practical guidance. This lack of specificity and detail may make these guidelines difficult to apply in the real world. This may explain why the FTC recently found that few websites were in compliance with these principles [31].

More detailed and applicable frameworks have been proposed, most embracing the FIPs while providing more detailed guidance. Bellotti and Sellen developed a framework for privacy-aware design in ubiquitous computing [6, 7]. A brief description of this method is provided in Box 1.

What this framework presents is a procedure designers may follow in order to evaluate a system, and a set of requirements for solutions. This is a definitive improvement over the high-level FIPs guidelines. Though

Bellotti and Sellen's framework has been the benchmark since it was introduced, there is no evidence in the literature of widespread use. A recent variation on this method was that proposed by Hong et al [20] (see Box 2).

The designer asks a set of questions about the proposed system meant to identify potential privacy problems:

- What information is captured and how?
- What happens with this information?
- How is this information made accessible to the user?
- What is the purpose of the information collected?

Given the answers to these questions, the designer then identifies core vulnerabilities and ways to address them. The following criteria are both guidelines for desirable design, and benchmarks for evaluating potential solutions:

- Trustworthiness
- Perceptibility
- Minimal intrusiveness
- Flexibility
- Meaningfulness
- Low cost
- Data quality
- Use limitation
- Openness
- Accountability
- Appropriate timing
- Unobtrusiveness
- Fail safety
- Low effort
- Learnability
- Collection and limitation
- Purpose specification
- Security safeguards
- Individual participation

**Box 1: Bellotti and Sellen framework**

The designer starts by asking a set of analytical questions:

- Who are the users?
- What kinds of information are shared?
- What are the relationships between data subjects and observers?
- How much information is shared?
- How much information is stored?
- How is the personal information collected and shared?
- Are there malicious data observers?

A vulnerability should only be remedied if the cost is lower than the product of the likelihood of a violation and the damage it would cause, providing a set of priorities. A set of questions guide the discovery of potential solutions:

- How does the unwanted disclosure take place?
- How much choice, control, and awareness do data sharers have?
- What are the default settings?
- Is it better to prevent unwanted disclosures or prevent them?
- Are there ways for data sharers to maintain plausible deniability?
- What mechanisms for recourse or recovery are there?

**Box 2: Hong et al framework**

#### Method Review

These frameworks are relatively inexpensive to use; they are not very time-consuming nor do they require a lot of expertise. Ideally, these traits should promote adoption and real world use. Heuristics are often derived from case studies and observation, a large part of their appeal. Despite these benefits, Bellotti and Sellen's framework, and even the FIPs, have seen relatively little adoption (Hong et al. is too new to evaluate).

These heuristic frameworks address fairly high-level concepts, they don't address implementation or technology issues such as what protocols to use or programming pitfalls. They derive an analysis independent of implementation. Critics may of course say that they are too abstract, that important problems are being glanced over by keeping the analysis at this level. This is an important critique because these frameworks do not give a way of getting from design to requirements and implementation. A number of decisions which may impact the final design need to be made on the way down to this level.

Another problem with these methods is that they do not take iteration into account, an important part of the design process. Changes to one part of a system's design may affect multiple other parts in terms of privacy. These heuristics, producing a list of vulnerabilities, leave the designer no choice but to re-evaluate the entire system for each proposed change. There is no reliable way of leveraging previous analyses. This means that rather than becoming a part of the design process, these methods are more likely to be employed once at the end of the design cycle.

These heuristic forms of analysis assume that designers can, with the aid of some well-chosen questions, discover all or most vulnerabilities. Nielsen showed that given a small number of reviewers, heuristic analysis can be highly effective for general HCI problems [26]. This may not hold true for a domain such as privacy where designers have much less experience and bias abounds. These biases may have a large effect when it comes to hot-button topics such as privacy, and may lead the designer to overly focus on certain areas to the exclusion of others. This does not mean that heuristics will not work; it may just mean that a higher number of reviewers are needed to reach optimal performance.

Hong et al. include a cost-benefit analysis in their method to determine which problems are worth addressing. This takes into account the fact that not all vulnerabilities are worth addressing. It may even be the case that some vulnerabilities are impossible to address without crippling the overall system. In many ways, a vulnerability which does not have a cost-effective fix is sometimes more important to document, as it will live on in the system. Keeping track of hidden assumptions and vulnerabilities should therefore be an essential part of maintaining a system.

There are some problems with this cost-benefit analysis; this step depends on coming up with reliable numbers for the likelihood of a problem occurring as well as potential damages. Buttler and Fischbeck showed how you can do this when prior data is available [9], but it always includes

an element of guesswork. Because of this uncertainty, it may be more useful and honest to come up with non-numeric categorizations (e.g. critical, urgent, important, annoyance) and a set of objectives (e.g. eliminate all important vulnerabilities costing less than 100 man-hours to fix.)

### From Heuristics to Goal Analysis

To address some of the weaknesses of the heuristic approaches we turn to the goal-oriented analysis methods of requirements engineering [13, 15, 30]. In goal-oriented analysis, a domain is a collection of goal-achieving actors. Actors correspond not only to users who are pursuing their own goals (as in GOMS [23]) but also the major architectural components, described in terms of the user or customer goals they are designed to support. Goal analysis methods, such as ScenIC [30] provide guidelines for identifying goals, refining them into operationally defined tasks, and allocating tasks to actors. It should be stressed that no analysis method is fully algorithmic or replaces expert domain knowledge. Different analysts could produce functionally equivalent but structurally different goal hierarchies when applying these guidelines.

Goals, in contrast to requirements, are approximations of system properties. They may be vague or incomplete. They may be goals for the solution of a problem but not necessarily goals that will be wholly achieved by the proposed software system. Thus the system boundary may still be fluid, with the automation, partial automation, or assignment to users or external systems of tasks that achieve desired goals still being an unresolved issue.

Another difference between goals and requirements is that goals are idealizations that not necessarily fully achievable in their original form. The real world is not always as accommodating as a set of high-level goals assume. The "real world" may include human decisions outside the control of the designers, physical properties of the environment that may be predicted or influenced but not determined absolutely, other systems that may not behave as assumed, and the implementation of the proposed system itself, which may include defects. In the areas of privacy and security, the world includes actors with their own goals antagonistic to the systems'. In requirements engineering, it is often assumed that this refinement and allocation of goals is a rational and beneficent activity, but nothing in the approach requires this. Indeed, in a multi-actor domain, such as e-commerce or GCS, there is ample opportunity for conflicts or tradeoffs among the goals of users.

A goal may be blocked when there exists a hypothetical situation which makes the fulfillment of the goal impossible. We normally refer to this situation as an "obstacle". Such an obstacle is an "anti-goal," a set of events which make it impossible for the goal to be satisfied. In the domain of privacy, it is common to refer to obstacles as vulnerabilities, because they locate the potential for privacy violation. To identify vulnerabilities, we use the obstacle-identification heuristics of ScenIC [30] and KAOS [8].

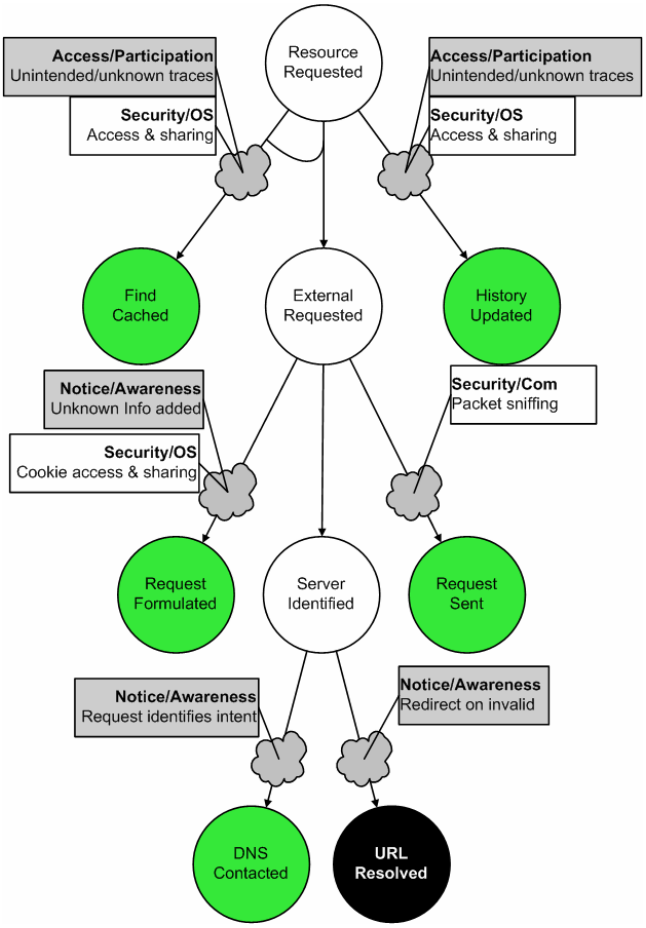


**Figure 1: A goal-tree with vulnerabilities**

As part of a goal-oriented analysis, one typically derives a goal-tree (see Figure 1). In the goal-tree, goals and sub-goals are drawn as circles, the top decomposed into lower level circles, as denoted by the arrows. Actors responsible for goals are typically identified by color-coding the nodes. Arches along the paths denote an 'or' operator, and while the ordering of the children left to right does not necessarily denote order of operation, we have attempted to accommodate that reading. Vulnerabilities are drawn as clouds with a callout describing them, and are placed on the path of the goal they block. Sub-goals sometimes refer to each other recursively. No new vulnerabilities are introduced or removed in these recursive branches. Therefore, for the sake of brevity, these nodes may be marked and the sub-trees omitted.

With STRAP we address some of the concerns associated with the heuristic frameworks. We leverage the real-world experience on which the heuristics are based, and structure and methods provided by goal-oriented analysis. This last

point gives us a method for bridging the gap between design, requirements and code.

Our goal with STRAP is to guide the analysis to ensure that we get a thorough, detailed, and unbiased evaluation. This is especially important when we are dealing with new domains or applications. In STRAP, we try to strike a balance between the costs of application against the utility of the method. We seek to provide enough detail to generate real value, while still being light-weight enough to see real use.

## STRUCTURED ANALYSIS OF PRIVACY

STRAP is a method in four steps:

1. Analysis
2. Refinement
3. Evaluation
4. Iteration

Each of these steps is explained in more detail below.

### Design Analysis

In order to address privacy vulnerabilities, we must be able to reliably find them. Depending on the complexity of the system, or the novelty of the domain, this may not be an easy process. Current frameworks rely on leading the designer through a set of analytical questions aimed at identifying potential trouble areas. This depends on the designers' ability to thoroughly examine all aspects of the application. The process of elicitation is not unreasonable, but the requirements literature shows that this type of analysis is difficult and error prone unless preceded by a systematic analysis of the system [3]. By omitting steps, ignoring problem areas or fixating on a specific component, important functions, hidden dependencies, and deep-rooted architectural problems may be overlooked.

We therefore start by performing a goal-oriented analysis of the system. As part of this analysis, the domain with its actors, goals and major system components is identified. Context information is collected; what the expectations of the different actors are with regards to privacy such as whether it is an open or closed system, whether there are expectations of privacy, of layers of access, default settings and behaviors, and of course the limitations of the physical and technological environment. This helps give the designers and users a shared vocabulary for discussing the system.

For each goal and sub-goal we ask a set of analytical questions, similar to those of [6, 20]:

- What information is captured/accessed for this goal?
- Who are the actors involved in the capture and access?
- What knowledge is derived from this information?
- What is done with the information afterward?

If information is captured or accessed, it presents a potential vulnerability. The user may wish to somehow be made aware of this, and consent may need to be collected. While the first question identifies vulnerabilities, the remaining help identify the appropriate actions to take. We mark all vulnerabilities in the goal-tree as clouds blocking the path to a goal, and keep a record of its context.

Once vulnerabilities have been identified, we look for common causes and duplicates. These often occur when one set of goals collect and store data needed to meet a different set of goals. The vulnerability will then appear in two or more places, with different contexts. These contexts must be merged in order for a full picture of the vulnerability to emerge. Vulnerabilities are then evaluated and categorized in terms of the risk they pose to the user, as described in section 3.2.

Final we categorize the vulnerabilities, which helps identify strategies to follow in order to negate them. It is important to note that these are not hard categories, but rather loose labels which suggest approaches to follow in negating the vulnerabilities. We derive these from the FIPS [31]:

- Notice/Awareness
- Choice/Consent
- Security/Integrity
- Enforcement/Redress

These categories overlap, particularly the Notice/Awareness category which is a catch-all, and prerequisite to most of the other categories. Once vulnerabilities have been identified, organized, evaluated and categorized, we are ready to enter the design refinement stage.

### Design Refinement

It is important to note that it may not be possible to address all vulnerabilities in a system. In some cases they may not be worth addressing, typically because the cost would be prohibitive, because any fix would introduce more serious vulnerabilities, or because any remedies would seriously undermine the utility of the system. Vulnerabilities resulting from dependencies on other systems, such as the operating system, are especially difficult to remedy, short of switching platforms. In such cases, the best one can do is to note the existence of these vulnerabilities so they are not overlooked later in the systems lifespan (maintenance and continued development).

The first step in the refinement process is to look at which vulnerabilities can be eliminated, and which can be mitigated. As an example: Data storage is often associated with potential theft or misappropriation of information. This can in part be avoided by encrypting the data, for which there are several low-cost implementation options (e.g. built in database functions). The benefits (e.g. avoiding wholesale compromise of database) therefore

clearly outweigh the costs of elimination (e.g. use built-in encryption functions.) The design document is then updated to reflect that this database must be encrypted, and that the vulnerability considered eliminated.

Vulnerabilities may also be eliminated by modifying the goal structure. A vulnerability may be considered important enough yet impossible to eliminate through implementation choices. This is an indication of an inadequate architecture or system design. To eliminate these vulnerabilities, the goal-tree must be re-examined to realign, remove, or modify the goals introducing this vulnerability so that the vulnerability disappears or a different implementation is possible. These are likely the most difficult vulnerabilities to address.

Vulnerabilities which cannot be eliminated, or which are deemed too costly to eliminate, may instead be mitigated. Mitigation is a strategy by which one tries to minimize the damage caused by a violation, and/or the likelihood of a violation occurring. The first is usually done by imposing limits on the information stored, processed or displayed; the latter by involving the user in the decision-making process. It is important to note that there are a large class of vulnerabilities which cannot be eliminated, only mitigated. This is especially true for Notice/Awareness and Choice/Consent types of vulnerabilities where there is room for nuance in terms of user preferences or sensitivities.

As we discussed earlier, and as Palen and Dourish point out, one is unlikely to find a single policy or design solution to fit all. A more successful strategy will in many cases be to involve the user in the privacy management process. This is the only real way to customize applications to meet the privacy requirements of diverse user populations. There are of course multiple strategies to employ at this point; we will simply present a few which we have found to be useful or promising.

The simplest strategy is one which we use every day, presenting the user with a dialogue, informing them of what the system is going to do, and ask them to consent or decline. Since the causes of every vulnerability are documented, warning and consent can be collected in context. This strategy, given the tendency to overwhelm users, is unlikely to succeed for complex systems.

We know that privacy is seldom a primary concern for users; privacy invasions are a potential side-effect which the user seeks to avoid while performing some other task. We have seen that when security or privacy systems interfere excessively with the tasks users wish to accomplish, these systems are often disabled. For more complex systems, users will likely need to specify policies about the use of their personal information so as to limit the clarification requests from a privacy management system.

While privacy management as a dynamic decision making process, high-level polices and plans can serve as the basis for basic risk assessment. These can serve as the basis for more advanced and less intrusive UI approaches such as mixed-initiative systems [21] or ramping interfaces [32]. These techniques try to minimize the distraction to the user by determining what the user needs or wants to know, and disclosing more or less information as needed.

In a mixed-initiative approach, a high-level policy is evaluated against the risks associated with the disclosure or withholding information. The cost of distracting the user also factors into the calculation [21]. The result is an expected utility for each of four possible actions: correct automatic disclosure, incorrect automatic disclosure, correct withholding, and incorrect information withholding. The utility values dictate what action to follow, or when the user should be prompted to make the decision (utilities too close or low to discriminate). Over time, user actions can inform the model, resulting in a more detailed, flexible, and individualized model, progressively becoming less invasive.

"Ramping interfaces" [32] can also be employed to let the user determine the amount of attention he/she wishes to devote to privacy management tasks. By progressively providing more details about the potential disclosure of information as the user increasingly interacts with it, this style of user interface facilitates the quick execution of straightforward decisions and the more involved, lengthier determination of difficult decisions.

These are only some of the possible strategies for mitigating vulnerabilities. Other techniques, such as attention-based interaction or peripheral user interfaces could also be employed to minimize the cost of interaction. Social solutions such as collaborative filtering could be used to inform decision-making for privacy. The application domain and its constraints will dictate which are feasible, or desirable in any given situation, and more are sure to be added as work continues in this area.

### Evaluation

As in any design process, several competing designs should be generated, where possible by independent designers. Competing designs should then be evaluated to identify the most successful design (or the synthesis of the most successful designs). Because individual or small groups of vulnerabilities will have similar causes and remedies, it is in theory possible to combine elements from multiple designs.

The first type of evaluation is to look at the risks assigned to each vulnerability and calculate the delta for the two design solutions. The design which results in the greatest decrease in risk is the best, from the perspective of privacy. This design may of course not be the best from other design perspectives. Other factors also need to be

considered, primarily how much the redesign affects the overall functionality or value of the resulting system.

Mitigation strategies will by their very nature not completely eliminate risks. For these we must instead perform an evaluation of the adequacy of the solution. For this we refer back to the classification of the vulnerability according to the FIPs categories. Each type of vulnerability presents certain unique challenges, derived from the definition of the FIPs categories and previous frameworks [31]. Proposed solutions need to meet as many of these as possible:

1. Notice/awareness
    a. Available, Accessible and Clear
    b. Correct, Complete and Consistent
    c. Presented in context
    d. Not overburdening
2. Choice/Consent
    a. Meaningful options
    b. Explicit consent
3. Integrity/Security
    a. Awareness of security mechanisms
    b. Transparency of transactions
4. Enforcement/Redress
    a. Access to own records
    b. Ability to revoke consent

In addition to these minimum requirements, the solution must of course meet requirements in terms of human factors. The evaluation factors in Bellotti and Sellen's privacy heuristics make up a good list of desirable properties for any solution [6].

### Iteration

Though the analysis and redesign processes are given structure, it is still beneficial for multiple designers to perform this analysis. Ultimately, the actual identification process is driven by the designers understanding and perspective. Jacobs, in his analysis of Heuristic evaluation noted that given a small set of analysts, their combined results quickly reach near-perfect detection [26]. While we expect this method to do better than purely heuristic approaches it should hold true that a larger number of critical eyes improve the analysis.

Most design processes are naturally iterative, and it is important to support this practice. We have seen different steps in this method benefit from multiple design phases and how these are evaluated and merged. This method also leaves a documentation trail of vulnerabilities found. This stays with the system through its lifecycle, documenting unaddressed vulnerabilities, assumptions, and the motivation behind design decisions. As the system evolves, this document needs to evolve. Successful designs start to manifest vulnerabilities when the system is used in contexts that violate hitherto justifiable assumptions [3]. It is essential to document assumptions so that they may be re-checked whenever the design changes.

Before new features are added to the system, their impact on users' privacy needs to be evaluated. This is done by modifying the goal-tree to include the new objectives of the system. We then do the analysis step on the new part of the tree. If new information is collected, the rest of the graph must be examined for ripple-effects in terms of privacy. For each goal we ask how it is affected by the new vulnerability. New vulnerabilities may emerge as part of this process, or even disappear as new goals are added. This process, though time-consuming, is less so than re-performing the entire analysis from scratch.

## EVALUATION

To valuate STRAP, we present a comparative study against the Bellotti and Sellen method. For this analysis we chose to use the Augur calendar system developed by Tullio et al. [33] as the target of analysis. This system was chosen because it has a number of known privacy vulnerabilities and because the subjects were unlikely to be familiar with it. The goal of this evaluation was to see if STRAP would prove to be more cumbersome (prohibitively so) to use compared to the Bellotti and Sellen method and whether it would result in better analysis results (more vulnerabilities discovered, less false positives and noise). Given time resource and space constraints we did not seek to do a more exhaustive evaluation as in [26].

### Augur: A Shared, Predictive Calendar

The Augur calendar system is a web-based, shared calendar that provides additional predictive features intended to facilitate communication within a workgroup. These features include predictions on the attendance of colleagues, as well as information on who has scheduled the same events. These predictions are based on Bayesian networks and improve over time, learning from attendance patterns. With these features, users can identify events that are no longer attended, make informed decisions about which of several conflicting events will be attended, and determine who they will likely see at a particular event.

Users access Augur via secure login. Scheduled events are presented in a standard hour-by-hour, block format. This view is augmented with additional information indicating colleagues who have scheduled the same events and attendance probabilities for those colleagues. Events on a user's calendar have a colored bar to indicate the user's likelihood of attendance as predicted by Augur.

### Methodology

Similar to Nielsen's study [26], we recruited 32 college students from an HCI class. The students had completed their full semester, covering the usual HCI curriculum including Heuristic evaluation, GOMS and similar evaluation methods. They had not covered privacy as a

specific subject, nor read about Bellotti and Sellen's work. They had all completed significant project work as part of their class-work (50% of their overall grade).

The students were given a system description complete with screenshots of the Augur system in use. They did not have access to the system itself. They were randomly assigned into two groups, 16 subjects in the Bellotti and Sellen condition and 16 in the STRAP condition. The students were given a 2 hour lecture, roughly one hour on Augur, and 1 hour on the method they were assigned. The students were also given hardcopies describing their method, in the case of Bellotti and Sellen [7], and in the case of STRAP a draft of the relevant section from this paper.

The students then went off to do their analyses individually, though the students in the STRAP case were allowed to work out the goal-tree in groups of up to three students. The students submitted their results together with an estimated time-on-task. Students knew their performance on this experiment would not be linked to their grade in the class.

We expected the STRAP group to spend more time-on-task then the Bellotti and Sellen group given the overhead of performing the goal-oriented analysis. We also expected to find that the STRAP group performed better both in terms of the number vulnerabilities discovered and the quality of the analysis (fewer false positives).

### Results

31 students returned their assigned analysis (1 was missing), and 26 returned data on the time spent on the analysis. The full data is reported in Table 1.

| | Time on task (minutes) | Total Notes | Vulnerabilities | General HCI issues |
|---|---|---|---|---|
| **STRAP** | 88.77 *(25.82)* | 6.86 (2.28) | 5.14 (2.14) | 1.00 (0.96) |
| **Bellotti & Sellen** | 101.18 (43.46) | 6.53 (2.50) | 3.80 (1.74) | 2.13 (2.10) |
| | +13.49 | -0.33 | -1.34 | +1.13 |

**Table 1: Study Results (stdev in parenthesis)**

We did not find any statistically significant differences in terms of the time spent doing the analysis (n=24, t=0.831, p=0.418). In fact the students in the STRAP condition reported spending less time on the analysis, which surprised us. There was no significant difference in the total number of reported vulnerabilities (n=29, t= -0.364, p=0.718). These "vulnerabilities" were filtered to remove non-privacy issues.

In the case of the Bellotti and Sellen case a full 32.65% were found to be general HCI issues rather than privacy issues (compared to 14.58% for STRAP). We did find marginal significance in the number of "real" privacy vulnerabilities discovered (n=29, t= -1.845, p=0.077).

## CONCLUSIONS

In this paper we have presented a novel approach to designing for privacy, STRAP. STRAP is a light-weight structured analysis technique that incorporates heuristics from existing frameworks and borrows from the fields of requirements and goal-oriented analysis. Our approach provides an analytical structure for privacy-aware design and a method for deriving policy requirements from the analysis. We demonstrate that STRAP performs better than the standard Bellotti and Sellen heuristics, requiring as much time on task yet resulting in more privacy-related vulnerabilities discovered.

## FUTURE WORK

In the future we will conduct a larger and more detailed analysis of STRAP and how it performs both against the Bellotti and Sellen as well as the Hong et al. frameworks. We are especially interested in determining why the Bellotti and Sellen framework elicits such a large number of general HCI issues. Though we do not consider it to be a confound, we will involve more expert designers in these studies, and see whether we can find differences in terms of the kinds of vulnerabilities discovered through the different methods. We will also seek to apply STRAP through a real software development cycle to see if iteration and refinement are adequately supported.

## REFERENCES


1. Abowd, G. and Mynatt, E. D. Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction*, 7(1):29-58, March 2000.
2. Ackerman, M.S. and Cranor, L,. Privacy critics: UI components to safeguard users' privacy. In *ACM Conf. Human Factors in Computing Systems (CHI'99)*, 1999.
3. Anderson, R. J. Security Engineering: A Guide to Building Dependable Distributed Systems, Wiley 2001
4. Antón, A.I., He, Q. and Bolchini, D. The Use of Goals to Extract Privacy and Security Requirements from Policy Statements, *To appear in: 12th IEEE International Requirements Engineering Conference (RE'04)*.
5. Ashley, P. and Schunter, M. The Platform for Enterprise Privacy Practices; Information Security Solutions Europe, Paris, October 2002.
6. Bellotti, V. Design for Privacy in Multimedia Computing and Communications Environments, In Agre, P., & Rotenberg, M. Eds. Technology and Privacy: The New Landscape. MIT Press, Cambridge MA, 1997.
7. Bellotti, V. and Sellen, A. Design for Privacy in Ubiquitous Computing Environments. *ECSCW 1993*.

8. Bertrand, P., Darimont, R., Delor, E., Massonet, P., van Lamsweerde, A. *GRAIL/KAOS: an environment for goal drivent requirements engineering.* Proceedings ICSE'98 - 20th International Conference on Software Engineering, IEEE-ACM, Kyoto, April 98.
9. Butler S. A. and Fischbeck P. "Multi-Attribute Risk Assessment" *Proceedings of SREIS'02*, Raleigh, NC, 2002.
10. Code of Fair Information Practices (The), US. Department of Health, Education and Welfare, 1973.
11. Cranor, L., Langheinrich, M., Marchiori, M., Presler-Marshall, M., and Reagle, J. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. W3C. 16 April 2002. Online: http://www.w3.org/TR/P3P/
12. Culnan, M.J. Georgetown Internet Privacy Policy Survey: Report to the Federal Trade Commission. Washington, DC: Georgetown University, The McDonough School of Business, 1999.
13. Dardenne, A., Lamsweerde, A.V. and Fickas, S. Goal-directed requirements acquisition. Sci. Comp. Prog, 1993. 20(1-2): 3-50.
14. Earp J.B. and Meyer, G. Internet Consumer Behavior: Privacy and its Impact on Internet Policy, 28th *Telecommunications Policy Research Conference*, Sept. 23-25, 2000.
15. Evans, E. Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley 2003.
16. Grudin, J. Managerial Use and Emerging Norms: Effects of Activity Patterns on Software Design and Deployment. In Proceedings HICSS-37, 2004.
17. Grudin, J. Desituating Action: Digital Representation of Context. Human-Computer Interaction, 16, 2-4, (2001)
18. Grudin, J. and Palen, L. Emerging Groupware Successes in Major Corporations: Studies of Adoption and Adaptation. *International Conference On Worldwide Computing and Applications*, 1997.
19. Grudin, J. Groupware and social dynamics: Eight challenges for developers. Communications of the ACM, 37, 1 (1994).
20. Hong, J.I., J. Ng, S. Lederer, and J.A. Landay. Privacy Risk Models for Designing Privacy-Sensitive Ubiquitous Computing Systems. *Designing Interactive Systems (DIS2004)*. Boston, MA..
21. Horvitz, E. Principles of Mixed-Initiative User Interfaces. *Proceedings of CHI'99*, May 1999, pp.159-166.
22. Jensen. C. and Potts, C. "Privacy Policies as Decision-Making Tools: A Usability Evaluation of Online Privacy Notices" *In Proceedings of CHI'04* Vienna, Austria, 2004
23. John, B.E. and Kieras, D.E. The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast. *ACM Transactions on Computer-Human Interaction*, 3 (4). 320-351.
24. Langheinrich, M. "Privacy by Design - Principles of Privacy-Aware Ubiquitous Systems." *Proc. Ubicomp 2001*, pp. 273-291, Springer-Verlag LNCS 2201, 2001
25. Lessig, L. Code and Other Laws of Cyberspace. Basic Books, New York, 1999.
26. Nielsen, J. & Molich, R. (1990). Heuristic evaluation of user interfaces, Proceedings of ACM CHI'90 Conf. (Seattle, WA, 1-5 April), 249-256.
27. Palen, L. and Dourish, P. "Unpacking 'Privacy' for a Networked World." *Proceedings of CHI'03*, Ft. Lauderdale, FL. 2003
28. Palen, L. and Grudin, J. Discretionary Adoption of Group Support Software: Lessons from Calendar Applications. Implementing Collaboration Technologies in Industry. B. E. Munkvold, Springer Verlag.2002
29. Palen, L. Social, Individual, and Technological Issues for Groupware Calendar Systems. *Proceedings of CHI '99*, Pittsburgh, PA, pp. 17-24.
30. Potts, C. ScenIC: A Strategy for Inquiry-Driven Requirements Determination, IEEE Fourth International Symposium on Requirements Engineering (RE'99), University of Limerick, Ireland, pp. 58-65, 7-11 June 1999.
31. Privacy Online: Fair Information Practices in the Electronic Marketplace. A Report to Congress. Federal Trade Commission, 2000.
32. Rhodes, B.J. Margin notes: building a contextually aware associative memory, In Proceedings 5th International Conference on Intelligent User Interfaces (IUI'2000),.
33. Tullio, J., Goecks, J., Mynatt, E.D., and Nguyen, D.H. Augmenting Shared Personal Calendars, UIST 2002, 11-20.
34. U.S. Children's Online Privacy Protection Act of 1998, Public Law No. 105-277, October 21, 1998.
35. U.S. Gramm-Leach-Bliley Financial Modernization Act of 1999, Public Law No. 106-102, November 1, 1999.
36. U.S. Health Insurance Portability and Accountability Act of 1996, Public Law No. 104-191, August 21, 1996.
37. Whitten, A. and Tygar, J.D. "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0," in Proceedings of the 8th USENIX Security Symposium, August 1999.