Out of Sight, Out of Mind: UI Design and the Inhibition of Mental Models of Security

Eric Spero
Carleton University
Ottawa, Canada
eric.spero@carleton.ca

Robert Biddle
Carleton University
Ottawa, Canada
robert.biddle@carleton.ca

ABSTRACT

In this paper we make the case that UI design inhibits mental models of security by concealing most of the security-relevant aspects of software functionality. Users are frequently required to make decisions that have important security implications, that requires a mental model of software infrastructure to know what actions are 'safe' versus 'unsafe'. People build internal causal models of what they experience that have explanatory and predictive power, and therefore form the basis of the decision-making faculty. By concealing security information, user interfaces hinder the user from building the kinds of models that will keep them safer, and only the small minority who are willing to go beyond the interface will acquire this knowledge. We suggest increasing the visibility of some essential information about the security-relevant aspects of software functionality in a way that ordinary users will be able to make sense of, so that through normal interactions with software everyone develops the kind of knowledge needed to better support security. We review the cognitive science and cybersecurity literature on mental models, present three 'case studies' which embody the security concealment problem, and present preliminary suggestions for how UI design might amend this problem.

CCS CONCEPTS

• **Security and privacy → Usability in security and privacy**; • **Human-centered computing** → *HCI theory, concepts and models.*

KEYWORDS

mental models, usable cybersecurity, human-computer interaction

ACM Reference Format:
Eric Spero and Robert Biddle. 2020. Out of Sight, Out of Mind: UI Design and the Inhibition of Mental Models of Security. In *New Security Paradigms Workshop 2020 (NSPW '20), October 26–29, 2020, Online, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3442167.3442174

1 INTRODUCTION

Usability and security have a complex and subtle relationship. In this paper, we identify a new aspect of this relationship. An issue often discussed in security is that users have weak mental models, of security threats and of the mechanisms designed to defend against those threats. One of the most well-established approaches to the design of usable software is to create a representation that allows users to interact with software while concealing the inner workings involved. Users develop mental models in large part as a result of their interactions with the user interface, and whatever is concealed is prevented from becoming part of user understanding. This is acceptable and even desirable where understanding will not have a meaningful impact on the lives of users. But we argue that so much security information has been hidden from users that they cannot develop the kind of knowledge that will guide safe action. This issue needs to be addressed in order to better support user security.

In the security literature, the term "mental model" is typically used informally. In the psychology and cognitive science literature, however, there is serious consideration of the nature of mental models, how they are formed, and how they are used. In particular, it is suggested that mental models are formed from observable experience linking causes and effects, and this leads to expectations and predictions of phenomena [18].

One of the earliest and most influential approaches to human-computer interaction design stems from the observation that the internal workings of software need not be the way the software is presented to the user. Instead, a "system image" can be created that allows the user to control and interpret the software behaviour in a way that eases the interaction. Typically, this might use language already familiar to the user from the task domain, and would avoid any mention of any internal software technology.

The problem that arises is that attacks on the security of software often involve the internal levels of software that "good" interaction design has concealed from users. This means that users will typically not understand the attack mechanisms, and have difficulty with any defensive measures that require user involvement. Indeed, users may be unaware that attacks occur, and unaware of whether defences succeed or fail. This suggests a need for a new approach to user interaction design.

The rest of this paper proceeds as follows. First is a review of the relevant literature: about mental models in psychology and cognitive science, in cybersecurity, and finally from human-computer interaction. We then present our argument in detail: that concealing all implementation details from the user impedes good mental models of security. This is followed with several cases where security is weakened in this way. We conclude by offering suggestions for a more nuanced approach to user interaction design for security.

NSPW '20, October 26–29, 2020, Online, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8995-2/20/10...$15.00
https://doi.org/10.1145/3442167.3442174

2 MENTAL MODELS

2.1 In Cognitive Science

The "mental models" view regards the mind as a flexible 'imitator' of the external world [18]. When observing the world, special attention is paid to the causal relations between things, and this information is used to build-up abstract internal models that work in the same way. Because mental models have the same causal structure as the real-world entities and processes they imitate, they can be used to simulate the outcomes of various conditions. This predictive power of mental models is essential in decision-making. The greater the correspondence between the internal causal structure and the outside-world thing or process imitated, the more accurately one will be able to predict what will happen in as a result of some condition, and the more one is better able to align their actions with their goals.

This view was first articulated by Craik in *The Nature of Explanation* [18]. For Craik the ultimate test of a concept—in science, but also in everyday life—is not the exactness or consistency of its internal/intensional properties, but rather the degree to which they impose order on diverse instances and enable successful predictions. That is, what matters about concepts is whether they *work*. Useful concepts are created not by considering single instances in isolation, but by observing many instances, creating general principles that explain them, and updating these rules in light of later observations.

According to Craik, models are able to make predictions about the world by virtue of a three-part process: (1) translating some aspect of the external world into an internal symbolic representation; (2) manipulating the symbolic representation to arrive at other symbolic representations; and (3) translating the symbol representation back to the external world through the execution of a plan of action, or a recognition of a correspondence between the predicted and actual world [18]. Craik mentions a few physical devices which mimic real-world processes in this way, such as Kelvin's tide-predicting machine (pictured in Figure 1). A successful prediction requires that the symbolic representation bears the same "relation-structure" as the state of affairs it imitates; in other words, the two must *work in the same way.* As the tide-predicting machine shows, the model need not bear a pictorial resemblance to the thing it imitates.

Craik suggests that the mind, in its capacity to anticipate the future, works according to the same principle as Kelvin's tide-predicting machine. However, the mind is far more flexible, enabling humans to cope with the novelty and variability so common to human experience. This tool is used to test out possible actions before undertaking them: one can simulate the implementation of various designs for bridges, say, and run stress tests to find out which among them is best before actual construction begins; thus danger can be anticipated and avoided. Simulating the outcomes of conditions through manipulation of mental models is cheap, quick, and convenient. This ability seems to offer a survival advantage, and for Craik is the "definite function" of thought.

The term *mental models* was popularized by Johnson-Laird [28]. He developed a mental model theory of reasoning, which included the implementation of 'mental models' accounts for various mental phenomena as effective procedures. For Johnson-Laird, people reason not by application of internalized formal rules of logic, but by generating a representation of the events described by the premises: a mental model. This representation depends on the meaning of the premises, and on inferences from general knowledge.

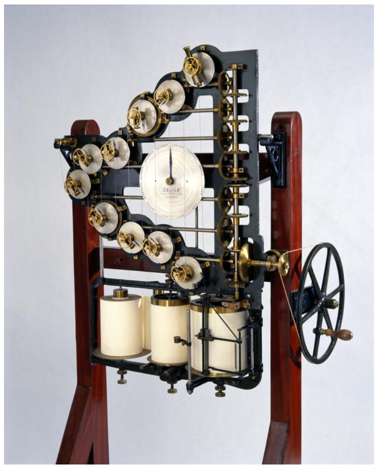


Figure 1: Craik likened mental models to prediction machines of his day, such as Kelvin's Tide-Predicting Machine [11]. It can predict the tide because it works in the same way: each gear corresponds to a tidal constituent, and the sum of these gears determines the position of a needle representing the tide level.

Johnson-Laird makes the point that more detailed mental models are not necessarily more useful, and the amount of accuracy needed to make good predictions depends on one's goals. For example, to successfully operate a piece of software like a word processor, a mental model relating user interface elements to the appearance and content in a written document is sufficient. Activities like troubleshooting software installation issues, writing word processor software, and designing computer architectures will each require different mental models of varying complexity. Having good mental models of CPU architecture will not help one compose better documents using a word processor software, even though the latter ultimately depends on the former. As long as one's model is sufficiently complex to successfully mimic the goal-relevant behaviour of some real-world system, it is adequate.

The mental models used in reasoning are representations (or simulations) of specific entities and events—tokens rather than types [5]. They are generated with the help of "deep" structures that make up general knowledge. These more abstract structures have been given different formulations under different theoretical frameworks, going by names such as *frames*, *prototypes*, and *schema* [5, 21, 28]. What matters to us is not the details separating these various formulations, but what unites them: that they provide layers of abstraction linking together disparate instances by their causal properties. We adopt the term "schema" here.

Schemas correspond to categories of experience, and consist of abstract entities and events and relationships between them. They are built up from memories of perceptual experience: whenever something is attended to, information from this experience is extracted and put toward the creation or elaboration of one or more schema (see Barsalou [5] for an account of how this system might be implemented computationally). The more instances of a category observed, the richer the schema, and thus the more 'true to life' the simulations it generates.

Humans have a natural ability for finding common causal structure across domains and situations which may vary widely in their surface features. In analogical reasoning, knowledge from a well-understood source domain is transferred to a less-understood target domain through mapping. Analogies provide a powerful method of coping with new situations, and are extensively relied upon in education, as in the familiar case of comparing the structure of an atom to that of a solar system (though it is really more complex). In order for the transfer to succeed, the two analogs must share a common schema. Gick and Holyoak [21] conducted numerous studies demonstrating the effectiveness of analogical transfer when solving new problems. Participants' ability to solve a difficult problem was dramatically improved by exposure to problem-solution pairs from radically different domains. For example, one target problem asked subjects how to treat a tumour with radiation, and were helped with a story about troops storming a fortress. Participants were even aided by analogs that took the form of simple diagrams depicting relations between entities.

Norman applied the idea of mental models to user interface design with his Theory of Action [40]. Users come to software with goals in mind (representations of desirable state of affairs), which tools like software help make manifest. Successful use of the interface requires developing a model of the causal structure relating the elements of the UI, and a mapping between the UI elements and the user's goals. This coupling between the physical state of the interface and the psychological state corresponding to the user's representation of the current goal-relevant state of affairs enables the user to track progress toward their goal. Anticipating user needs, the user interface designer tries to provide a straightforward correspondence between UI elements and users' goals. We discuss Norman's Theory of Action more in Section 3.1.

Mental models give meaning to UI elements and guide interactions with software. Interactions with software also feed back into the user's mental models, teaching the user about what is possible, and what the software is really doing. Anything software does that is not represented in the user interface may never become part of a user's working understanding of software. In Section 3.2 we argue that this has happened with the security-related aspects of software functionality, which have been backgrounded in favour of a focus on primary domain-oriented tasks (checking emails, composing documents, etc.). This keeps the user in the dark about how they are vulnerable, and prevents the kind of understanding that lead to safe behaviour.

Mental Model Definition. In this paper we adopt the Craikian definition of a mental model, which may be paraphrased as:

> *An internal representation that relates to something in the external world (e.g. an entity or process) by "imitating" its causal structure; i.e. it works in the same way. This isomorphism between mental models and the real world things they imitate gives rise to their essential function: simulating the consequences of various possible conditions for the purposes of guiding behaviour.*

2.2 In Cybersecurity

L. Jean Camp and colleagues' "mental model approach" [14] advocates for the use of knowledge from better-understood domains in communicating to users the risks involved in computer use. According to this approach, users conceptualize computer security in terms of other domains such as physical safety, medical infections, criminal behaviour, warfare, and markets [4], and each individual's mental model of computer security risks was seen as a multidimensional combination of one or more of these. These extra-domain schemas have different implications for behaviour: for example, the criminal behaviour schema emphasizes investigation and punishment from a central authority, and physical security emphasizes lock-down measures [9]. However, each extra-domain schema has incongruities with cybersecurity, and none on their own will perfectly communicate the relevant risks. It is recommended that different schemas be used to emphasize a particular dimension of cybersecurity risk at appropriate times.

Wash conducted a series of interviews with ordinary computer users to investigate their mental models of computer security [58]. His interviewees generally distinguished between 'hacker' and 'virus', considering them two distinct threats. All used the term 'virus' as a catch-all for malware, and also used related medical terms like 'infection' and 'immunity'. Wash asked respondents about the purposes, effects, and methods of transmission of malware, and found four patterns of responses, which for him indicate four mental models: (1) viruses are generically 'bad', (2) viruses are buggy software, (3) viruses cause mischief, (4) viruses support crime. He also asked about 'hackers', including what they did, why they did it, and who they would target. He also identified four mental models of hackers: (1) hackers are digital graffiti artists, (2) hackers are burglars who break into computers for criminal purposes, (3) hackers are criminals who target big fish, (4) hackers are contractors who support criminals. Some participants were in the possession of several of these mental models for hackers and viruses. Participants were given a questionnaire gauging participant opinion on the importance of following various security advice, and found a relationship between these beliefs and the eight mental models. He closes the paper by showing how malware like botnets defy these mental models, which he conjectures could be a contributing factor to their success.

Rader, Wash, and Brooks [43] found evidence that people came to have the mental models identified earlier by Wash as a result of stories heard from others. People share with each other accounts of security incidents which included information about the dangers involved and what was done to re-secure their computer. The victim will share these stories, typically with their friends and family, who will often re-tell this story to others. Participants reported that these stories had an influence on their future thought and behaviour about computer security.

There are numerous studies comparing the mental models held by technical users with those held by non-technical users. Bravo-Lillo et al. [10] were interested in how users processed and responded to computer security warnings. They conducted open-ended interviews with users asking them to describe the process of receiving warnings to taking action, and created a flow chart schematizing this process. They found that experts had more complex models, and proposed examples of how warnings could help compensate for weaknesses in the mental models of non-technical users. They note some misconceptions on the part of non-technical users, including a disregard of SSL warnings on banking websites because banks must be safe, a reliance on the appearance of the website when making security judgments, and the belief that opening a file is safer than saving it to local storage. Participants were confused by technical language such as “encryption” and “disk”. Kang et al. [29] asked participants to draw their understanding of how the internet worked. They found that non-technical users tended to demonstrate vaguer understanding, and relied more on metaphors (e.g. “cloud”, “main hub”) in their depictions. They showed awareness of only those organizations and services they had direct experience of. Asgharpour, Liu, and Camp [4] asked technical and non-technical users to assign extra-domain schemas (e.g. physical safety, medical infections) to various security risks, and found differences between the two groups.

Klasnja et al. [30] conducted diagramming exercises and interviews with participants to assess their knowledge of wifi, and related security and privacy threats. Their participants demonstrated good practical knowledge of wifi including a network’s range, signal strength, and signal propagation. They understood that wifi range extended outside the buildings containing the wireless access point, that signal strength weakened as distance from the wireless access point increased, that signal strength is impaired after passing through solid objects, and that weak signal strength is related to one’s ability to connect to a network. They demonstrated weak knowledge of how wifi and the technologies it uses worked, and this seemed to lead to a poor understanding of security and privacy issues related to wifi use. Their mental models of cybersecurity threats while using wifi consisted of hackers attempting to break into their computers to read their files and spy on them, or people overlooking their shoulders when in public spaces. They were not aware of encryption, or that wifi is a broadcast medium, and therefore they did not know that unencrypted data sent over public wifi could be intercepted by others on that network. In other words, their relevant mental models of cybersecurity were insufficient.

2.3 Discussion

The research on mental models in cybersecurity suggests that the typical non-technical user has a weak understanding of how computers, software, and the internet work, which impairs their ability to detect threats, and take appropriate measures to defend themselves. The source of users’ mental models is a combination of (1) their direct experience as users, (2) analogies to other domains, and (3) stories they hear from others.

The influential research by Camp, Wash, and Rader suggests that the mental models users have relating to cybersecurity are largely metaphorical/analogical in nature, borrowing information from better understood domains. The approach taken by these authors is generally pragmatic and descriptive: their goal is to understand how users think, so they can improve communications of cybersecurity concepts. While they point out shortcomings of these metaphor-heavy models, they are careful not to say they are ‘right’ or ‘wrong’. It is stressed that mental models need not be technically accurate, they just should support safe behaviour.

Metaphorical/analogical models perform the useful function of supporting behaviour during embryonic stages of learning. However, in all analogies there will be some incongruities between source and target, and these can lead to dysfunctional behaviour. For example, Whitten and Tygar [59] found the ‘key’ metaphor in PGP harmful because of mismatches with users’ everyday concept of keys. Demjaha et al. [19] tested seven different metaphors for end-to-end encryption, and found that all caused varying degrees of harm. In a domain as complex as cybersecurity these incongruities are immense, and as others have noted [4] each of these metaphorical schemas can only imperfectly cover but a single facet of cybersecurity. In an adversarial context these incongruities are equivalent to gaps in defences, which attackers work to exploit.

One solution to the problem of cybersecurity education for typical users is to impart knowledge to users facet-by-facet by making reference to models they have from other domains, and then attempt to remove or reduce the harmful effects of incongruities by pointing each of them out. However, the end-result of this type of education—a patchwork of overlapping models augmented with large lists of exceptions—seems unparsimonious, requiring more effort than considering security more on its own terms.

It seems there is no substitute for directly representing the domain of cybersecurity. At first glance this might seem unrealistic: we obviously cannot ask users to become experts on topics like TCP/IP and encryption. Success will depend on the careful use of *user interface encapsulation*. This technique has been successfully applied in a number of domains and contexts, enabling the use of complicated machinery with only high-level knowledge. In UI support for primary tasks it enables users to browse the web and send and receive email without understanding the HTTP or SMTP protocols; in automotive interface design it enables one to operate a motor vehicle without deep knowledge of automotive mechanics.

We could provide users with key information about the system’s security status and any relevant explanatory structures, in such a way that is intelligible to regular users and directly relatable to their security goals. In theory, this would make it possible for users to develop internal models of the system’s security status that are equivalent to those of technical users at a high level of abstraction. Just as the design of automobile interfaces has made it in principle possible for anyone to drive a car with performance levels meeting or exceeding those of a car designer or mechanic, it might be possible for regular users to operate a computer as safely as a security expert. We emphasize that what we are proposing is not a form of persuasion nor conditioning. Rather, we want to help users build the kind of conceptual machinery enables successful mapping of potential user actions to outcomes.

We think that mental models of cybersecurity are generally weak because UI developers have chosen to focus on supporting users’ primary domain-oriented goals, while backgrounding or outright ignoring security-related information. In the sections following, we begin to sketch how security aspects of software might be better represented in the user interface.

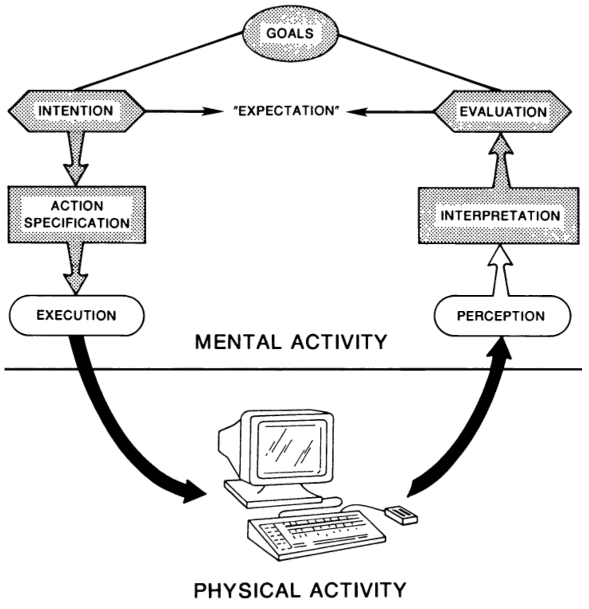


Figure 2: Seven stages of user activities involved in the performance of a task. From Norman [41].

3 INTERFACES AND SECURITY

3.1 Norman's Theory of Action

In a landmark paper from 1986, Norman [41] presents his Theory of Action: an "approximate theory" of goal-oriented interaction between a human user and a computer system. It describes the key problem facing users and UI designers as the gulf between user goals, which are *psychological* in nature, and the terms of the computer system, which are *material*. Users must be able to map their goals to the system controls and state so that they can manipulate the system state to bring about their goals, and to understand if these manipulations had the desired effect. The gulf between human and computer gulf can be narrowed from either side: the user learns to recast their goals and intentions in terms that are more system-like, and system designers create interfaces that are human-intelligible and match the user's needs.

The user and computer participate in an iterative, interactive loop (see Figure 2), consisting of the following stages:

(1) Establishing the Goal
(2) Forming the Intention
(3) Executing the Action
(4) Perceiving the System State
(5) Interpreting the State
(6) Evaluating the System State with respect to the Goals and Intentions

In short, users have a goal, use the interface provided to express how to achieve that goal, and then execute the appropriate actions. After executing the action, they monitor the system state as expressed through the UI, and evaluate the outcome based on whether it brought them closer to their goal or not.

Norman suggests this feedback loop involves an interplay between three representations of the system: the *design model*, the

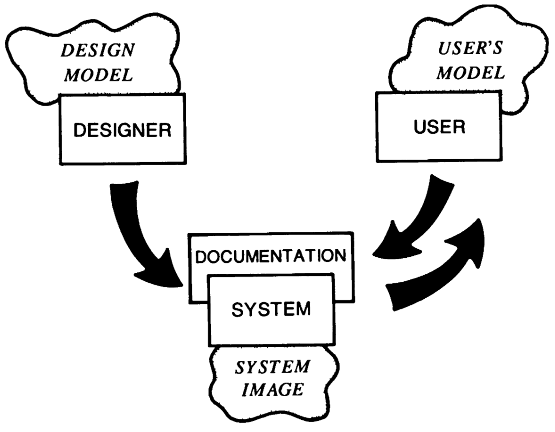


Figure 3: Three concepts involved in interaction: the design model, the user's model, and the system image. From Norman [41].

user model, and the *system image* (see Figure 3). The design model is the mental model of the system held by designers and implementers, which ideally contains the user's needs as a key component. The user model is the user's mental model of the system: what it is and how it seems to work. The system image is the part of the system the user sees and interacts with, including the UI and physical interaction devices. The user constructs their model of the system from the system image, and not the designer's model.

Norman calls for tools that reveal their conceptual structure to provide a strong sense of understanding and control. Systems that are too powerful (i.e. where the user has too little control) impair users' understanding of the system, leaving them feeling powerless. On the other hand, systems that are too simple, while offering much control, can fail to bridge the gulf between mind and system, preventing the user from mapping their goals into system states. The optimal amount of control given to the user versus the machine will vary based on the abilities of the user and the task at hand; this presents a difficult design challenge.

Any aspect of the underlying structure and functionality of software that is not made intelligible to users in the system image cannot become integrated into their mental model. We argue that this is what has happened with the security-relevant aspects of software. Most indications of the security status of the system state are obscured from most users, accessible only to those with the technical knowledge required to use arcane tools. It is certainly the case that users have security as only a secondary and ancillary goal, and even then software does not adequately support them. UI designers have instead chosen to focus only on primary domain-oriented goals such as sending emails and composing documents. We discuss this issue more, and begin to sketch our idea of possible remedies in Section 3.2.

3.2 (In)visibility of Security

Over 30 years after his landmark paper [41], Norman's dream for better interaction design has largely been realized. Software occupies a central role in daily life, and users of all technical capabilities are able to manipulate computers to bring about their goals. User interfaces elements are intelligible by ordinary users, and readily mappable to goals. Users have enough power to accomplish their goals, but not so much as to overwhelm them.

An important caveat, however, is that only *primary domain-oriented* goals are well-supported by software UIs—the kind of goals that bring people to their computers in the first place, such as writing text documents, sending emails, and sharing photos. Almost no support is provided for security—a *secondary* but important goal. Seldom does the user interface attempt to communicate to the user anything about the security status of the running program. When it does, it is often not effective. For example, browser certificate indicators have been a visible (though arguably inconspicuous) part of every browser user interface for years, yet users still appear to be largely unaware of web certificates or the assurances they offer. Security tools and concepts are notoriously difficult [59]. Cyber-attackers are aware of the user-computer gulfs when it comes to security, and in recent years they have increasingly shifted their focus toward the user [20].

A related problem is that whenever UIs do attempt to communicate security information to users, it is not done so in way that facilitates understanding by users who lack mental models rich enough to take the appropriate course of action. The system image of the security-relevant aspects of software functionality typically does not have a relation-structure—the user is not provided visibility into the relevant entities and processes and the causal relations between them—which makes it impossible to develop mental models rich enough to guide behaviour, unless they are prepared to seek external educational resources. If the average user is asked if they are "sure" that they want to download a piece of software they found on the internet, they will have no principled way of answering one way or the other.

In addition, we think this backgrounding of security issues and extreme foregrounding of primary-tasks promotes the idea that software does not have security issues. Humans build mental models based on their observations, so if users only see system feedback related to primary tasks, this will foster a notion that software is only concerned with performing primary task–related functions. When planning actions, if users only consider the primary-task–related consequences of their actions, they will be susceptible to any attack which presents itself as useful. Guided by such a mental model of software, why should the user ever practice caution?

When designing to defend against security threats, system designers generally favour a paternalistic approach: security matters are handled automagically by the underlying infrastructure, and the user is occasionally brought into the loop when absolutely necessary. Given the complexity of computer security issues, this may seem a sensible and effective approach. The user's input is required wherever there are gaps in the infrastructural solution—where it cannot be determined with certainty that a given action will cause the user harm, and the user must make a decision. For example, if a website offering a useful service to the user has a misconfigured certificate—or no certificate—what should be done? This site could be perfectly safe, and outright preventing the user from visiting the site could get in the way of them achieving a primary goal for no purpose. On the other hand, the typical user does not have the knowledge required to understand the risks they are taking by visiting the site.

One approach to this problem is "soft paternalism" [33] wherein the user is "nudged" toward making a conservative security decision in an uncertain situation. HTTPS warnings in web browsers are an example of this, where a warning message is displayed, along with a prominent 'back' button (labelled "Back to Safety"). The user may proceed to the site, but the link to do so is hidden until an inconspicuous "Advanced" link is clicked. While the "soft paternalistic" approach is more effective than providing no guidance at all, it is far from perfect. In the HTTPS warning example, there will be many situations where the user will be dissuaded from visiting a safe website, and others where the user clicks through the warning and visits an attack site. A "softer" paternalism yet is shown when operating systems provide vague warnings about running software from unknown sources: the user might be asked if they are *sure* they want to run this software. In these situations the user must draw on their understanding of the relevant issues to make a decision. But will their understanding be sufficient?

In cases like those just mentioned, where the infrastructure is unable to determine whether an action is safe or unsafe, it must be up to the user to make the right decision. There is in principle enough information 'out there' for the user to make good decisions, but at present it practically inaccessible to most users. It should be the responsibility of the system designer to write interfaces that enable the development of security mental models rich enough to support safe behaviour.

There are a number of challenges relating to security—and computer security in particular—which may explain why interfaces do not yet support effortless mapping. First, security has a *secondary* or *ancillary* status in the minds of users. Users typically do not begin interacting with a computer with a security goal in mind, and frustration may set in if their progress toward achieving a primary goal is unduly interrupted by secondary concerns. Second, security issues are complex, and it may be difficult to separate what should be conveyed to users from what should be left concealed. Computer security is particularly difficult because users cannot as readily bring their everyday physical world mental models to bear on problems as they can in the case of physical security. One can easily leverage everyday knowledge to envision some ways in which our home security can be compromised, which also suggests how it may be defended. However, most of us do not have the kind of everyday experience which will enable the configuration of, say, a firewall, which requires some domain-specific knowledge of topics like TCP/IP. Third, security is essentially *adversarial*: security incidents are carried out by skilled attackers who actively aim to undermine defensive efforts. This adds an extra layer of psychological complexity, as interface designers not only need to be sensitive to the goals and intentions of users, as is usually the case, but also to how attackers might exploit these goals and intentions to benefit themselves. Attempts to help improve user vision into system security can also be exploited.

While we call security goals *secondary* and *ancillary*, this is not to say they are unimportant. Everyone has security goals and places high value on them; nobody wants to have their credentials copied or be exploited for money. User interfaces should make it easy for users to map the system state to their security goals, just as it has for primary domain-oriented goals. This will enable the construction of good mental models of system security, which will allow users to make decisions that align with their goals.

UI designers will still have to practice extreme care in deciding what information is made visible, how it is expressed, and what information is concealed.

4 CASE STUDIES

In this section we present case studies of three cybersecurity attack methods—malware, phishing email, and fraudulent websites—which embody the problem of security (in)visibility just introduced. We show how they are presently supported in system images, and make preliminary suggestions for how this might be improved. These case studies focus on desktop, mobile, and web platforms, but the problem extends to IoT devices as well [47].

In general, these attack methods succeed where there are misalignments between the user's mental model of what the software is doing, and what it is actually doing. The user believes they are taking actions that are wholly beneficial to them, but in reality they benefit an attacker, and can be harmful to the user.

User interface design contributes to this problem in the short-term by failing to provide intelligible cues to the security state of the system during key moments (such as when software is about to be installed), and in the long-term by concealing too much of the security-relevant aspects of software functionality. We claim the proper solution to the short-term problem depends on developing good high-level mental models of computer security.

4.1 Malware

Malware is software that runs on the user's device, and is in some way harmful to the user (e.g. exfiltrating their personal information; hijacking resources to mine bitcoin). In other words, malware is software that is misaligned with, or counter to, the user's goals.

Being affected by malware is a two-step process. First the malware has to be installed on the host machine. Second, the malware has to run.

Installing malware can happen in a number of ways, such as installing software from dubious sources, connecting to dangerous websites, and opening malicious email attachments. If the attack is to succeed, each of these actions typically presents themselves as consistent with the user's goals. Software updates and real-time antivirus protection can help protect the user from installing malware. Besides these, it is helpful if the user knows whether they can trust the software or not; support for trust in software is provided by digital certificates, which we discuss next.

4.1.1 Software Certificates. The user can avoid installing malware by knowing something about the trustworthiness of the code they execute. Properly used, code signing and digital certificates provide just this assurance. Code signing uses public key infrastructure and cryptographic hash algorithms to ensure that the program has not been modified by someone other than the original author. Certificate Authorities (CAs) are trusted third parties who can oversee the issuing of keys and/or certificates. In an OS context, the CAs are the OS provider (e.g. Microsoft, Apple). CAs can perform identity checks on the author of the code, which in turn provides the user an indication of their trustworthiness. For example, developers are asked to provide uniquely identifying information to CAs [34], and organizations seeking certificates must be bound to a legal entity (e.g. [3]), enabling recourse when there are problems. In other words, digital certificates can help give users confidence about the integrity and authenticity of the software they consider installing.

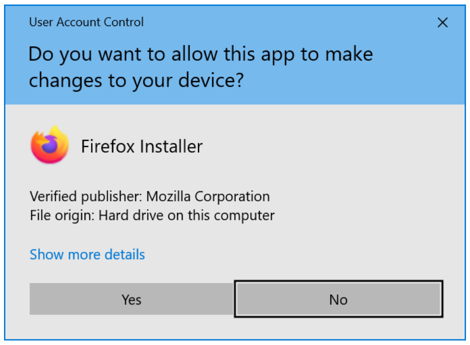
Ideally, each time the user installs new software on their device they are provided some intelligible information about the integrity and authenticity of the code. If all trustworthy developers participate in CA-mediated code signing, and all untrustworthy authors are denied from participating in this process, the information shown to users will provide a very strong signal of the safety of the code they are considering installing and running.

Desktop operating systems communicate information about the trustworthiness of software through prompts displayed to the user before they run the software. In Windows 10, a "User Account Control" prompt is shown each time they run a program which requires administrator rights, such as installers. These ask if the user wants to allow the program to "make changes" to their device. The prompts differ depending on whether the software is "verified" or not. Being "verified" means the software has a signed certificate from a member of the Microsoft Trusted Root Program [36]. Verified app prompts are shown with a blue banner, and the publisher's name and logo are shown (e.g. Figure 4a). Prompts with "unknown" publishers are shown with a yellow banner (e.g. Figure 4b).

In macOS Catalina, the default security settings allow users to install software from the macOS App Store, and from outside the App Store provided it has been developer-signed and "notarized" by Apple. When developers submit their apps for "notarization", the app is automatically checked for code signing issues, and scanned for malicious content [2]. If notarization is successful, a "ticket" is generated which the developer must "staple" to their software. A copy of the "ticket" is also sent to "Gatekeeper", the macOS security feature which ensures that apps have been notarized before running. Apple attempts to communicate whether an app has been notarized or not through warning prompts such as those shown in Figure 5 (which never explicitly refer to "notarization" or "signing").

If the user attempts to open an app from outside the App Store that has been signed and notarized, they are shown the prompt in Figure 5a. This prompt says that the app was downloaded from "the Internet" and that it has been scanned for malware by Apple, and asks if they are "sure" they want to open it. When attempting to run an app that has *not* been signed or notarized, the prompt in Figure 5b is shown, saying that the app cannot be opened because the publisher is "unverified". In order to run the app, the user can control-click the app icon before selecting "Open", which creates an exception for the app in security settings. After this is done, they are shown the prompt in Figure 5c, warning the user about the potential security and privacy dangers of running unverified software, and asking if they are "sure" they want to open it.

In Windows, the only differences between attempting to run apps from "verified" versus "unknown" publishers are the colour of the banner prompts, the inclusion of "verified" or "unverified" signifiers, and the presence or absence of a logo. The sequence of actions they need to take is identical. The user may never take notice of the difference between these two prompts. If they are paying close attention and notice the "verified" language, one wonders what significance it would have. No attempt has been made to build the user's mental models of Microsoft's verification process, so the user

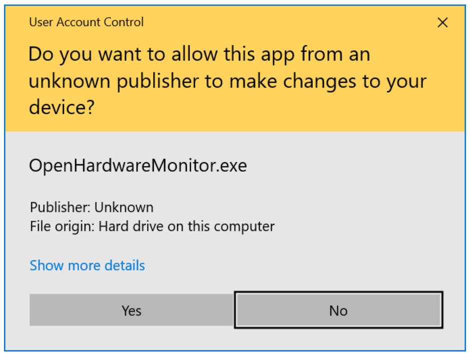

User Account Control

Do you want to allow this app to make changes to your device?

Firefox Installer

Verified publisher: Mozilla Corporation
File origin: Hard drive on this computer

(a) Requesting write access for software from a "verified" publisher, i.e. the software has a signed certificate from a trusted authority.


User Account Control

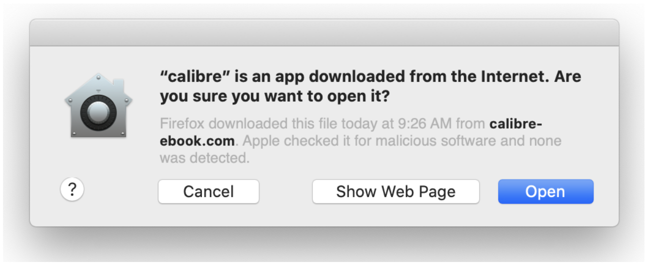
Do you want to allow this app from an unknown publisher to make changes to your device?

OpenHardwareMonitor.exe

Publisher: Unknown
File origin: Hard drive on this computer

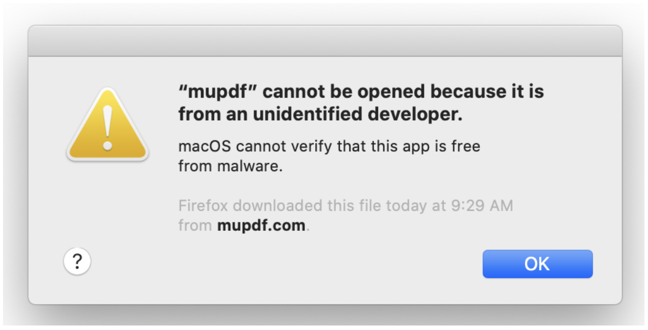
(b) Requesting write access for software from an "unknown" publisher.

Figure 4: Windows 10 User Account Control prompts for "Verified" (4a) and "Unknown" (4b) publishers


"calibre" is an app downloaded from the Internet. Are you sure you want to open it?

Firefox downloaded this file today at 9:26 AM from calibre-ebook.com. Apple checked it for malicious software and none was detected.

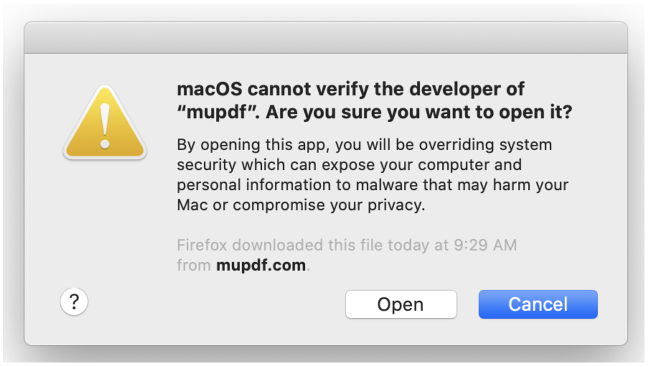
(a) Attempting to open an app from outside the App Store that has passed the "Gatekeeper" check, i.e. it is developer-signed and "notarized" by Apple.


"mupdf" cannot be opened because it is from an unidentified developer.

macOS cannot verify that this app is free from malware.

Firefox downloaded this file today at 9:29 AM from mupdf.com.

(b) Attempting to open an app that has failed the "Gatekeeper" check.


macOS cannot verify the developer of "mupdf". Are you sure you want to open it?

By opening this app, you will be overriding system security which can expose your computer and personal information to malware that may harm your Mac or compromise your privacy.

Firefox downloaded this file today at 9:29 AM from mupdf.com.

(c) Attempting to open the same app shown in Figure 5b after having made an exception.

Figure 5: macOS software certificate warnings

cannot see the assurances it provides. Indeed, a user study on Windows User Account Control prompts by Motiee et al. [37] showed a general lack of their effectiveness. Only 13% of participants recognized "unverified" software as being potentially dangerous or malware, 77% had an incorrect understanding of the purpose of the prompts, and 49% consented to a random fake prompt.

macOS more clearly distinguishes between software that has passed the OS legitimacy tests from those that haven't. By disabling the running of unidentified apps in the normal way, macOS makes it difficult for users to miss the distinction between the two. The warning shown when the user tries to open the non-notarized app makes explicit reference to malware, which may help give the user a basic impression of the significance of their verification process. It is not obvious how to bypass this feature, and to find this out the user will have to seek external resources. Many will be dissuaded from running the app, and may seek out alternatives instead. For those who do find out how to run apps from unidentified developers, the prompt shown when they try to do so warns the user of potential harmful consequences (i.e. harm to Mac; compromise of privacy).

Each of the prompts shown in Figures 4 & 5 asks the users to make a decision, but it will be difficult for many to know how to respond. Both Windows and macOS aim to express a distinction between apps which have been 'checked' in some way from those that have not, but the meaning of this distinction depends on a basic model of this checking process and what it means for user trust and safety. For example, if the process involves some identity checks on the developer and a careful inspection of the software for possible security issues, it will be evident to the user that this check has high relevance to their decision. On the other hand, if there are no background checks, there are no malware scans, or if the malware scans are easily defeatable, these checks will have less relevance for the user. Without attempting to inform the user of the checking process, the user does not know how much trust they can put in this process, and therefore the software. (This is of course assuming that the user even recognizes a difference to begin with, which they quite possibly will not in the case of Windows.)

An attempt should be made to impart a schema of the certification process to users both in the case of running non-certified software, and when running certified software as well. A mental model of the certification process will help users envision the consequences for their actions in a way that is not presently possible. We think the reference to malware scans in the macOS prompt is a step in the right direction, but more should be done.

In addition to these issues of communication, there are wider systemic issues with software certificates and CAs. It is common for seemingly legitimate developers to forego the OS certification process—Open Hardware Monitor (Figure 4b) and MuPDF (Figure 5b) are two examples. There are a number of potential reasons for this relating to challenges relating to the notarization process and cost. The macOS process is easiest with Xcode, which in turn requires macOS (and therefore Mac hardware) to run. Developers using other SDKs must invest development time into building their own tools and scripts [25]. Developers have noted frustrating bugs with Xcode and the notarization process [38], and in getting notarization for apps that use external libraries [25, 38]. Notarizing also imposes additional "hardening" restrictions [1] which can add extra development time (e.g. [24]). Finally, membership in the Apple Developer Program—a requirement for notarization—costs 99 USD annually, which appears to be a barrier for some (e.g. [60]). If many useful and legitimate apps from reputable developers forego the notarization process, it substantially weakens the ability for software certificates to signal trustworthiness.

4.1.2 Detect and Mitigate. If the user happens to install malware, they must detect it before any measures can be taken to remove it. The difficulty is that users are provided very little visibility into the underlying functionality of software, which makes it easy for malware to carry out malicious functions undetected.

Many users rely on antivirus software to perform this task for them [26], but their actual effectiveness is highly questionable. One study reported that 70% of malware today exists only once, and 80% disappears after one hour [12]—much too quickly for antivirus threat libraries to keep up. On Linux-based systems, AppArmor [27] defines what processes can be accessed on a per-application basis, which can limit the power of malware once it is installed on the user's system.

There are tools that can help improve the visibility of the activity of running processes, which can be used to find evidence of malware. For example, Little Snitch [42] reports the network requests made by each application, and allows users to restrict network access per-application. However, a significant amount of technical knowledge, attention, and fine-tuning is required to make use of such tools.

4.1.3 Primary Task Tunnel Vision. We suspect that the extreme backgrounding of security issues in user interfaces in favour of a near-exclusive focus on primary tasks fosters a dangerously naive conception of software in which security-relevant consequences for software use are simply not considered at all, and users instead think of software as something that essentially supports primary tasks. Such a view would lead users to think only of the primary task–related consequences for their action, increasing their susceptibility. For example, running an app is 'good' if it appears to align with primary goals, and 'bad' only if it does not. Attackers can exploit such a view by bundling malware with useful software, which is a common technique (e.g. 'cracked' software).

Indeed, Spero et al. [51] found evidence that users tended to think of malware in primary-task terms. Their study featured a diagramming exercise where participants drew their understanding of how a malware-affected word processor worked, and they found that many participants focus exclusively on the capacity for malware to disturb productivity. Malware destroyed and corrupted documents and devices, and interrupted the user's input, but seldom did it exploit the user or their device for someone else's benefit.

4.2 Phishing Email

In a phishing attack, the user is solicited to perform an action that the user believes is consistent with their goals (e.g. recovering their online identity or bank account, sending money to a friend in need, or responding to an urgent request). In reality, these actions typically lead to handing something over to an attacker (e.g. banking credentials, money). A key component of this scam is the attacker "spoofing" their identity (impersonating another). The attacker may pretend to be from a reputable organization, or a friend of the victim. In "spear-phishing" attacks, attackers use information acquired about the target to make their pitch more effective. For example, attackers may study the email communications of an employee for a while, and then at some opportune moment send an email spoofing the identity of a trusted other asking for a transfer of funds, which will be deposited into the attacker's account.

Phishing attacks are often attempted using email. In email, user interfaces provide users with relatively little identifying information about their conversational partner: mainly the `From` header and body text. The `From` header lists the email address and given name of the sender. Both of these pieces of information can be easily modified to make the message appear as if it is coming from someone other than the actual source. The 'real' source of the email message is not clearly visible in the main UI, and one has to know where to find it. Attackers use the body of the email to make their pitch to the user, which involves adopting the writing conventions of the person or role they wish to impersonate. They may make mistakes in grammar or style which can lead to suspicion, but more careful attacks can seem quite genuine.

DMARC (Domain-based Message Authentication, Reporting & Conformance) authentication checks provide the best-available infrastructural protection from fraudulent modifications to the `From` header [16]. DMARC includes other authentication mechanisms Sender Policy Framework (SPF) and DomainKeys Identified Mail (DKIM). SPF authenticates the `Mail-from` header—the true source of the email—and DKIM verifies digital signatures attached to the email. When performing a DMARC check on an incoming email, the receiving server queries the domain in the `From` header for its DMARC policy, which provides a list of domains which may use it in the `From` header. The receiving server then compares this list to the domains provided by SPF and DKIM, and if there is no match the DMARC check fails.

Unfortunately, there are some implementation issues with DMARC. A recent paper by Chen et al. [16] shows that there are inconsistencies between the various components involved in DMARC authentication, and also in their implementation by mail servers and clients, which leads to a number of vulnerabilities. According to Microsoft, only 9% of Fortune 500 companies publish strong DMARC policies [35]. Even where DMARC is enabled, users may be vulnerable to `From` header forgeries and other fraudulent misuse.

Email providers defend against phishing emails by using authentication mechanisms like DMARC, and through analysis of message content including attachments and the body text. When an incoming email is determined to be malicious it is typically directed to a "junk" or "spam" folder, which are rarely checked by users, and it is therefore likely that the message will never be seen. We think this is a lost opportunity to give users insight into the phishing email threat. These phishing email could be shown to users along with the factors that the anti-spoofing procedures used to determine it was fraudulent, allowing users to start building mental models of how these scams work, so they are better prepared to act safely if they ever have to make these determinations themselves.

Some spoofing tricks fall outside the scope of anti-phishing measures, such as changing the given name component of the `From` header, and creating email addresses from legitimate providers which look like they might be owned by the individual they are impersonating (e.g. John.Smith1@gmail.com impersonating John.Smith@gmail.com). At present, the user is offered no support for attacks of these types.

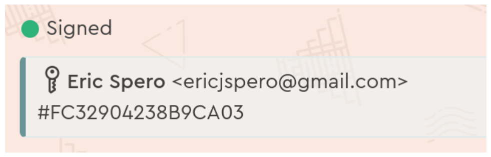
In general, we think increasing users' vision into the true source of the message would help them build the mental models of email necessary to identify fraudulent email whenever they are required to do so. This means more pieces of identifying information, provided in a way that might be intelligible to ordinary users.

We think that making the `Mail-from` (the true origin of the email) more visible would be beneficial, along with some information about the `Mail-from` domain. In addition, it would be helpful if users could be provided some information about this domain, including any cues regarding its trustworthiness.

Digital signatures would also be a useful additional piece of identifying information, providing assurance of the authenticity and integrity of an email through the use of public key cryptography. Just like software, email can be signed, and incoming messages can be accompanied by an indication that the message has or has not been signed by someone they trust (see Figure 6 for an example of this from browser-based PGP implementation Mailvelope [32]). Digital signatures add an extra layer of difficulty to the task of impersonating another, as the attacker now must also acquire a private piece of data owned by person they with to impersonate.

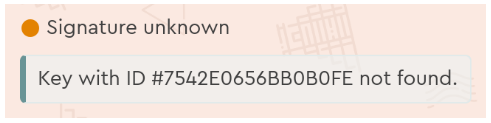
Email clients could in principle pre-trust email signatures from known trustworthy domains, including those who are most impersonated in phishing email (e.g. banks, tech companies). Users would then manually add people they regularly interact to their list of trusted accounts. Email from outside sources would be flagged as untrusted—similar to Figure 6b, but perhaps more obvious—which should put the user into a vigilant mindset. They could then be guided through a process of verifying the unknown sender to see if they should be trusted.

Support for digitial signatures in email are available today through many implementations of PGP. Unfortunately, the UIs of these implementations of PGP have long been notorious for their usability


● Signed

Eric Spero <ericjspero@gmail.com>
#FC32904238B9CA03

(a) Mailvelope indicator for a known signature


● Signature unknown

Key with ID #7542E0656BB0B0FE not found.

(b) Mailvelope indicator for an unknown signature

Figure 6: For signed emails, Mailvelope performs an automatic check to see if the signature matches a key in the user's keychain, and shows the results under the body text.

issues [59] which prevent its adoption by non-technical users. Mailvelope [32] is a browser extension that augments web email clients with PGP, allowing users to send signed and encrypted emails through providers like Gmail and Outlook.com. Mailvelope represents a step forward for usable PGP, but a basic mental model of concepts involved in public key encryption is still required to make use of it, and to understand how users can trust that a signed message comes from a particular individual. The app does not attempt to impart this mental model to users, and therefore remains truly usable only for a technical audience.

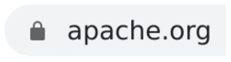
4.3 Fraudulent Websites

Similar to phishing email, fraudulent websites try to convince the user that they are offering a legitimate service in order to take advantage of the user. They are often used as part of phishing attacks to do things like capture passwords, accept payments, and deliver malware. Users might also find them through web search, or by mistyping a URL.

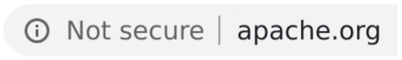
X.509 certificates [17] provide infrastructural assurances about the legitimacy of a website, and are part of the HTTPS protocol. These "web certificates" can be signed by CAs, who perform identity checks as part of the signing process. Browsers come pre-configured with a set of trusted CAs, and others can be added by the user.

CAs offer three validation standards to authenticate websites corresponding to different amounts of rigour in the checking process. Domain Validation (DV) offers the least assurance of identity, ensuring that the domain is controlled by the individual applying for the certificate. Organization Validation (OV) performs further checks that the organization exists, for example by checking government registries. The Extended Validation (EV) standard ensures that the website is controlled by a legal entity, and CAs take steps to establish the legitimacy of the business operating the website [13]. EV provides the most assurance.

Web browsers provide a visual indication of the website's certificate next to the URL. The indicators have changed over time, and vary between browsers; the discussion here reflects the current (mid-2020) version of Google Chrome. A "lock" symbol denotes that the website has a certificate that has been signed by a CA



(a) DV / OV / EV certificate



(b) No certificate

Figure 7: Web certificate indicators in Google Chrome (mid-2020). HTTPS connections are shown with a "lock" symbol meaning they are "secure" (e.g. Figure 7a), and HTTP connections are shown as "Not secure" (e.g. Figure 7b).

(Figure 7a), and the words "Not Secure" are shown if the site has no certificate (Figure 7b). Clicking on the certificate indicator opens a menu in which more detailed information about the certificate, including the validation standard, can be seen.

A number of issues in the communication of web certificate information to users impair their ability to meaningfully signal a website's authenticity and trustworthiness. For example, the "lock" symbol shown with DV, OV, and EV certificates is meant to indicate "secure" [22], but no attempt is made to explain to the user what "secure" means. Understanding what these certificates mean for user security requires working knowledge of the X.509 infrastructure, including CAs, encryption, and the client-server model.

A related issue is that a single "lock" symbol is now used for all three validation levels, despite large variations in the respective assurances they offer. DV certificates offer next to no assurances about identity: they are easily, freely, and quickly obtainable by anyone with a registered domain—malicious actors included: over 50% of malicious websites now use HTTPS [39]. With DV certificates, users know nothing about where their data is going. EV certificates, on the other hand, require substantial checks on the website owner and their affiliated business, cost hundreds of dollars, and the process takes days to complete. If a website has an EV certificate, users can have vision into who they are connected to, and also the knowledge that they have been vetted by a trusted authority—provided they have the appropriate mental models of web certificates.

What the three validation levels have in common is that they ensure an encrypted channel of communication between client and server. These connections are "secure" from man-in-the-middle attacks. However, if the endpoint of the encrypted channel is malicious, encryption will do little to protect the user.

There is empirical evidence that the current way of signalling identity and web certificate information is ineffective. In two separate studies by Stojmenović et al. [52, 53], eye-tracking was used to show that the vast majority of participants never look at the certificate indicator when making trust decisions, even when they are specifically asked to look for things which could be help them identify the page. Until late 2019 browsers distinguished EV indicators from OV/DV by showing the legal entity tied to the website to the right of the "lock" icon, but a large-scale study of Chrome usage by Google researchers showed that this way of signalling EV did not have a meaningful effect on user behaviour, and likely did not help users defend against fraudulent websites [54]. However, users did click on "Page Info" (accessible by clicking the "lock" icon) significantly more often when the EV indicator was present. It seems that users do not understand EV certificates, or what the EV indicator meant, or both. This is hardly surprising as little effort has been made to develop the user mental models involved. In a study on URL parsing, Reynolds et al. [46] found evidence of weak mental models of URL structure, and that users could not determine the correct identity of obfuscated URLs in 60% of cases.

There are at least two dimensions of security concerning connections to websites (encryption and identity), and the presence or absence of a single "lock" icon is incapable of communicating this status. Indeed, Biddle et al. [8] showed that the separate presentation of these dimensions in UI indicators improved user comprehension and feelings of safety. The larger problem is that the user does not know what these dimensions of security status *mean*, and they cannot develop an understanding without seeking external resources. We think that UI indicators for website certificate information should be rich enough so that over time users can develop good high-level mental models of certificates and the infrastructure surrounding them. Once they have a basic understanding of how certificates work, they can know what they signify, and therefore their relevance for the trust decisions they make online.

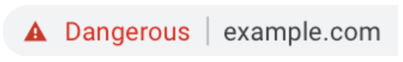
Besides web certificates, exclusion lists of known malicious URLs can help protect users from fraudulent websites. Perhaps the best known use of these exclusion lists in the context of web browsing is Google Safe Browsing, which is integrated with all major web browsers (i.e. Chrome, Firefox, Safari). When users attempt to visit a malicious URL, Google Safe Browsing shows a full-screen warning before the connection is made. Figure 8b shows a "Deceptive site" warning displayed by Google Chrome, which might be shown for URLs tied to known phishing sites. The format is similar to HTTPS warnings, and so are the options available to the user: they can abort visiting the URL by clicking the "Back to safety" button, or they can bypass the warning by clicking the "DETAILS" link, and then a clicking a link labelled "I understand the risks". The security status is shown as "Dangerous" to the left of the URL (Figure 8a).

According to a 2016 report Google Safe Browsing is the most effective exclusion list web browsing service, yet it only detects a small fraction of malicious websites [56]. In an informal analysis, we visited 20 active phishing domains from the Phishing Domain Database [31] at random, and none were flagged by Google Safe Browsing. There is some evidence that phishing sites are quite short-lived [6], and perhaps Safe Browsing faces similar challenges to those faced by antivirus software.

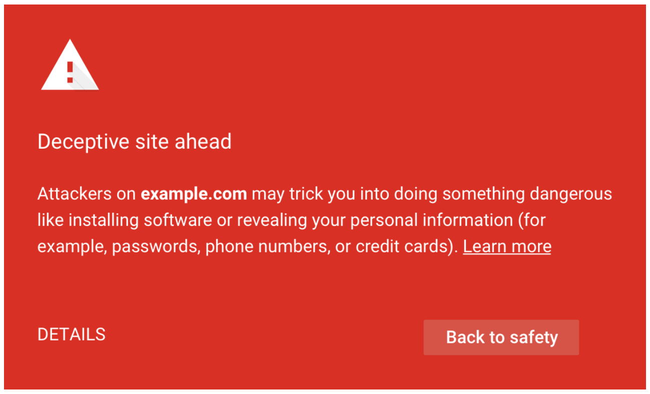
The warning page shown to users provides only generic information about the threat is provided. For example:

- The site ahead contains malware
- This page is trying to load scripts from unauthenticated sources
- Deceptive site ahead
- The site ahead contains harmful programs
- Suspicious site [23]

The goal is to alarm the user and turn them back around to prevent them from falling victim to an attack. This is good, but providing the user with more information would be beneficial for the development of richer mental models, which we think would make for a better long-term solution. Landing on this warning page is an



(a) "Dangerous" security status


Deceptive site ahead

Attackers on **example.com** may trick you into doing something dangerous like installing software or revealing your personal information (for example, passwords, phone numbers, or credit cards). Learn more

(b) Deceptive site warning

Figure 8: Google Safe Browsing uses an exclusion list of known malicious URLs to protect users from fraudulent sites, and is integrated with all major web browsers. When a user attempts to visit an unsafe site in Google Chrome they are shown a full-page warning (e.g. Figure 8b) and a "Dangerous" security status indicator (e.g. Figure 8a).

opportunity for education, and given that the user's own actions have brought them to a potentially unsafe situation they may in a particularly receptive state. We should provide the user with more detail about the nature of the threat to help prepare them for situations where Google Safe Browsing or similar defensive measures do not catch the problem. This could include communicating the evidence Google used to determine that the site is malicious. For example, was it deceptive domains, suspicious code execution, dubious claims made on the website (such as affiliations with major companies)? It might also be helpful to show the user how the suspicious page looks, as they might be surprised that an attack site can have a professional look-and-feel.

5 POSSIBLE SOLUTIONS

Wherever there are imperfections in infrastructural cybersecurity defences, it is up to the user to make the right decision. The user's ability to protect themselves depends on mental models of the situations they are in that are accurate and detailed enough to guide safe action. They must decide if they want to install a piece of software, visit and enter sensitive information into a website, and respond to requests for action made through an email. In these examples, the decision rests on user ascertainment of the *identity* or *origin* of something. The user must also be able to detect and remove any malicious software running on their device, which requires a model of the activity of applications.

There is infrastructural support for assurances about identity and origins in the form of digital certificates and signatures (though there are systemic and implementation issues), along with their UI indicators, and for understanding process activity, but the relevant information is not being properly communicated to ordinary users.

Understanding is model-based, consisting of elements bound together in a causal relation structure. If the aim is to provide users with enough information about the state of their system to inform their behaviour, it is not enough to say that software or websites are "verified" or "unknown", "secure" or "not secure". Users need to develop a gist representation of the real-world situation(s) underpinning these labels to know what they really mean.

UI designers must help users build the knowledge to generate accurate security-relevant representations. Ordinary users will not read documentation or seek external knowledge sources to build their understanding, so the source of this knowledge must be the user interface. This is a non-trivial challenge, especially considering that much of this knowledge will be domain-specific, which means the development of new knowledge structures. However, this same challenge has already been faced and met for support of mental models for primary domain-oriented use of software, and we see no reason why the same can't be done for security.

A unique challenge for the support of security models is to avoid major distractions from the primary task that led the user to use software in the first place. Economy of presentation will be key, and improvements will likely be very gradual. We identify two general strategies for meeting the goal of improved security understanding in light of the challenges faced. The first is to leverage analogy wherever possible, which includes references to structurally isomorphic knowledge from more-familiar domains, and visualizations in which causal relations are expressed through pictorial analogies. The second is to provide user with model-building information not just when there is a potential security risk, as is typically the case at present, but when the user is safe as well. For example, provide an indication of the verification process when software is deemed trustworthy as well as dubious. Increasing the number and range of situations in which this model-supporting information is provided is a way of coping with the relatively low amount of information that can be conveyed at a time.

In the remainder of this section we discuss some approaches which we think will be useful in devising a solution to this problem.

5.1 Ecological Interface Design

The issue of interface complexity is not new, long predating software systems. For example, the levers, knobs, and dials of steam engines required substantial training and experience to master. In the context of modern complex system engineering, an established approach developed by Rasmussen and Vicente is that of Ecological Interface Design (EID) [57].

The origins of EID include earlier work on mental models [44] but there is a specific focus on the needs of operating complex engineering systems, including the need for fault diagnosis and managing human error. EID involves an operation model distinguishing skills, rules, and knowledge (SRK), and establishing an abstraction hierarchy based both on physical elements and functional purposes. In this way, simple skill-based interfaces can be used for normal operation, with the deeper levels quickly available when needed.

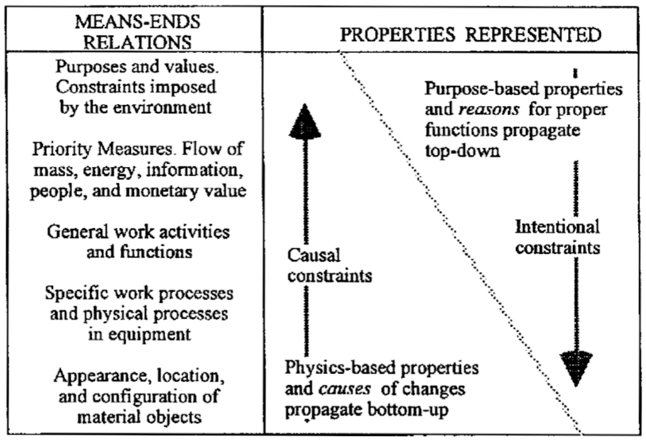
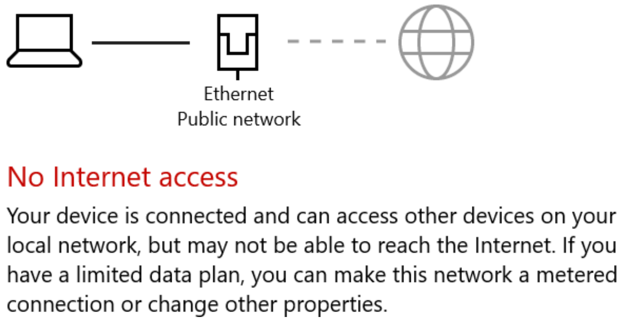

MEANS-ENDS RELATIONS
PROPERTIES REPRESENTED
Purposes and values. Constraints imposed by the environment
Priority Measures. Flow of mass, energy, information, people, and monetary value
General work activities and functions
Specific work processes and physical processes in equipment
Appearance, location, and configuration of material objects
Purpose-based properties and *reasons* for proper functions propagate top-down
Intentional constraints
Causal constraints
Physics-based properties and *causes* of changes propagate bottom-up

Figure 9: Systems can be described at several levels of a means-ends hierarchy. From Rasmussen [45]

This general structure, the abstraction hierarchy, is familiar (in different ways) to both end-users and to programmers. To end-users, it is sometimes called "progressive disclosure", and allows simple screens that handle most things, with an "Advanced" button or similar to allow handling of less common needs. The structure therefore limits cognitive load much of time, without prohibiting access to more detailed control. For programmers, the structure resembles levels of decomposition, whether in structured programming, object-oriented, or functional programming. The structure enables reuse, but also supports program comprehension, maintenance, and debugging. EID certainly has a place in addressing security. For operations centres, for example, it supports oversight of systems or networks, allowing easy ascertainment of normal states, while providing access to levels of detail in abnormal and potentially problematic states [7].

Applying the approach to software security for end-users, however, would present some different challenges. The reasons involve the nature of security understanding and the SRK model. From the perspective of the end-user, at the top of the abstraction hierarchy, there would be two common situations. In one, the situation detected is known to be harmful and is therefore forbidden. For example, access over a network to private data. In the other, the situation is potentially harmful, but needs detail consideration of the implementation levels of the software. End-users will not have, and likely not want, knowledge to engage—indeed, engagement might even make matters worse, for example by approving insecure operations such as unexpected requests to access interprocess communication sockets. Considering the SRK model, the end-users might be seen as having the skills to operate the software normally, but not the rules or knowledge of the implementation and security. Stated more carefully, the issue is that the end-users may well have mastery of rules and knowledge for their application domain, but not of the implementation or of the security domain.



No Internet access

Your device is connected and can access other devices on your local network, but may not be able to reach the Internet. If you have a limited data plan, you can make this network a metered connection or change other properties.

Figure 10: Windows 10 network status visualization.

5.2 Visualization for Understanding

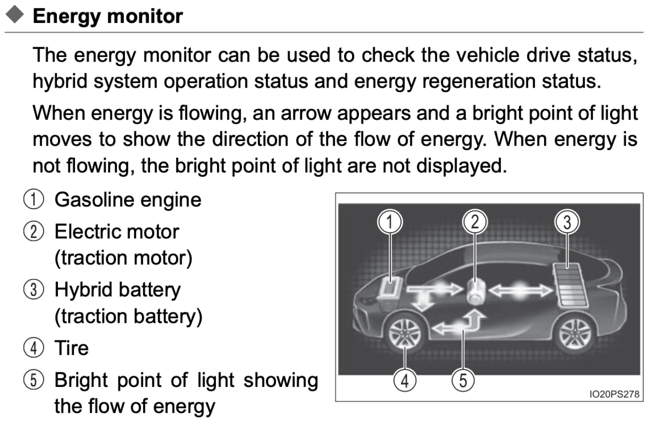
In day to day life, people see various displays that indicate aspects of the inner states of systems, even when they do not have specific control of those inner states. Simple examples include battery meters, fuel gauges, and, network activity monitors.

A more complex example is the Windows 10 display to help users understand network connectivity issues (Figure 10). The idea seems to be to help users understand where the problem is, for example with a network connection to their own machine, or with a more distant connection issue. In the case of a more distant problem, there might be little the user can do, but at least they know that is the location of the problem, so they may contact support personnel.

More complicated yet is a dashboard element on the Toyota Prius, a popular hybrid car that involves both a gasoline and an electric engine. The display is called the "Energy Monitor": see Figure 11. The diagram shows the elements of the drive system, including both engines and the battery pack, and shows which is currently being used to provide power; it also shows when the battery is being charged by regenerative braking. The driver of the car does not have precise control over these elements, but the display helps the user understand what is happening, and thus helps guide driver expectations about driving performance and economy. Similar visualizations have been proposed for "smart home" environments [15].

The common approach in all these examples is that they do not support directly controlling the system internals, but do reflect the impact of external events. They can therefore illustrate states that the user will understand, including the effects of errors or even malice. For example, even a simple battery meter on a smartphone or laptop computer can show an unexpected strong drain, potentially indicating the presence of malware.

If we were to apply this approach to all software systems, the issues then are what we could show, whether users could understand the implications, and how easily the approach could be defeated by attackers. Moreover, if the only way of involving users is a visual display, then damage may well be done before a user detects anything anomalous. If more active user involvement is required, then the situation resembles that of web browsers or smartphones, where software is tightly restricted, but allows access to resources with user permission on an as-needed basis. The interaction often required ensures user attention, and the timing means the user has some context for the request. However, apparently reasonable



◆ **Energy monitor**

The energy monitor can be used to check the vehicle drive status, hybrid system operation status and energy regeneration status.

When energy is flowing, an arrow appears and a bright point of light moves to show the direction of the flow of energy. When energy is not flowing, the bright point of light are not displayed.

① Gasoline engine
② Electric motor (traction motor)
③ Hybrid battery (traction battery)
④ Tire
⑤ Bright point of light showing the flow of energy

Figure 11: 2019 Toyota Prius Owner's Manual entry about the Energy Monitor dashboard display [55].

requests can be fraudulent, and repeated requests will be annoying and likely approved without appropriate consideration.

Visualization can also be used to aid end-user understanding by showing users the effects of their interactions with the system. Again, there many simple examples. Some are very primitive, e.g., volume knobs or buttons that indicate the directions for loud and quiet, and are typically standarized. Some are a little more sophisticated, such the diagrams on gearshift levers in standard transmission cars. These do vary between car models, but once the driver has learned the pattern, the diagram is seldom needed. In such cases, we might consider the diagram as having helped the driver build a mental model for controlling the transmission.

Visualization can also be a powerful method for delivering metaphors/analogies. As we mentioned in Section 2, humans rely on metaphor/analogy particularly for coping with novelty, by 'bridging-in' existing knowledge. UI elements can be made to pictorially resemble real-world entities or, more abstractly, causal relations, which could help the user understand something about system functionality. This technique enabled the desktop UI metaphor, which gave many users a newfound understanding of computer usage and capabilities [48].

6 A NEW SECURITY PARADIGM

We have identified issues in user interfaces that inhibit the development of security mental models, provided cases demonstrating the problem in context, and suggested how these might be resolved. In Section 6.1 we draw the main ideas in this paper together as a new security paradigm that we call Security Attention Visibility and Evaluation (SAVE). In Section 6.2 we describe this paradigm's major principles.

6.1 Security Awareness Visibility and Evaluation (SAVE)

When deciding how to act, people simulate the outcomes of various actions and conditions using mental models (Section 2.1). The simulations are accurate to the extent that the mental model and the thing it represents have the same causal structure. Building good causal structures requires repeated observation of the real world thing as it participates in various cause-effect events. When coping with novelty, there are situations where it is helpful to 'borrow' structure from similar models via analogical transfer. Using analogy is not strictly necessarily, but where it is not employed, high quality observations have an even greater importance.

In Norman's Theory of Action (Section 3.1), the observable parts of the system (the "system image") should serve as a bridge between the user's model of the system and the system's deeper engineering design; between the user's goals and the system's state. Users must be able to map the system image to their goals, and to the system state. The system often requires users to make decisions with security implications that depend on richer mental models than what they have. We suggest the weakness of security mental models is the result of an impoverishment of security information in the system image (Section 3.2).

Systems frequently ask users to make security-sensitive decisions while failing to provide the information they need to build the kind of models that can make those decisions. In software installation (Section 4.1), email (Section 4.2), and web browsing (Section 4.3), users are asked to make trust decisions without (a) understanding what the possible consequences of misplacing trust might be, or (b) knowing anything about the trustworthiness of the source.

We propose the SAVE paradigm as a solution to this problem, which is defined as follows:

> **SAVE paradigm**: *User security can be greatly improved by building into the system image better bridges between the user's security goals and the system's security state. Security information must be made salient in the user interface, in terms that users find intelligible, while making frugal use of user attention. These security bridges must prepare users to make informed security decisions whenever they are required by the system to do so.*

SAVE is not a form of persuasion nor conditioning; rather, it aims to build users' understanding of system security so that they are able to successfully map potential actions with outcomes.

In the following section we present four main principles for the SAVE paradigm. These capture the essential qualities interfaces must have to avoid the 'impoverishment' problem just mentioned—providing richer system images of security, which will help users build richer mental models of system security. We are flexible on details regarding *how* these principles may be implemented: Our main objective in this paper has been to articulate the nature of the problem and the general form of the solution. We plan to consider solutions and their implementation more thoroughly in future work.

However, we do identify some existing UI paradigms that we expect will play an important role in the implementation of SAVE: Ecological Interface Design (EID; Section 5.1), Visualization (Section 5.2), and metaphor/analogy (Section 2 & 5.2). EID supports investigating activity by emphasizing functional layers, allowing users to drill down layer by layer. *Progressive disclosure* can allow for a frugal communication of security issues by making details available on demand. Visualization supports users by displaying functional models of system activity, showing consequences of user actions (Section 5.2). Visualizations also provide an effective and efficient way of communicating causal relationships between entities in a system, which will be a major advantage for mental model development. Metaphor/analogy can help improve intelligibility by allowing users to bridge-in existing knowledge (e.g. the desktop metaphor). While useful, metaphors/analogies are a partial solution at best, and they can easily mislead (e.g. [59]); designers must carefully consider potential issues. None of these paradigms has been much employed to support security in the way we suggest, though we feel they all have significant potential.

6.2 Principles for SAVE

The following principles are minimal requirements for SAVE-abiding interfaces. We think these principles could be useful as UI evaluation heuristics, and could serve as helpful scaffolding for other HCI researchers who may wish to implement their own particular solutions.

Principle of Visibility*: Make the system's security state, and the security consequences of actions, visible to users.* Where there exists infrastructural support for a user's security decision (e.g. X.509 certificates; PGP signatures), make sure it is easy to find in the system image. The way it is presented in the system image must be in terms the user will understand in terms of their goals, the actions and the consequences—see below. Where the user's actions have consequences for security, make these visible as well. If this information is absent in UIs, there is little opportunity for the user to develop understanding.

Principle of Intelligibility*: Make it possible for users to map visible UI elements to security goals, regardless of their technical background.* It is not enough for the security state and consequences to be visible: the information must be presented in a form that is human-intelligible for all users, and it must be relatable to the user's security goals.

Principle of Frugality*: Treat attention as a scarce resource.* Attention is capacity limited, and managing users' attention is always important. Security is a secondary task, so it is especially important that frugality is practised when communicating security information to users.

Principle of Preparedness*: Make sure the user is able to make the security decisions required of them.* The ultimate test of whether a UI succeeds or fails is whether the system has adequately prepared the user (through mental model building) to make informed decisions whenever they are called upon by the system to do so.

6.3 Challenges

Our proposition is essentially to improve security behaviour through education, where the content is delivered through UI widgets representing relevant aspects of the underlying system state. We take the challenges that come with such a proposal seriously. We understand that computer security concepts are inherently complex and alien to most users. We consider *intelligibility* as the core principle of this paradigm, and demonstrating that users can make sense of what is being communicated to them will be the highest criterion for success. We also understand that security matters have a secondary status for users, meaning that users have less patience for being asked by the system to consider security. This constrains the bandwidth of communication between us and the user, exacerbating the challenges of intelligibility. We do not expect to find a single solution that will work perfectly for all users in all cases. However, by adhering to the principles just described, we do think that that practical gains can be achieved with at least a subset of users.

Another potential challenge is that the security system images could have the unfortunate consequence of helping attackers plan their attacks. The UI enhancements we propose would let attackers would know what aspects of the system security users are aware of, and they would then know to focus their attacks on other aspects of the system. This could create an 'arms race' where UI developers are making continual UI changes to respond to new vulnerabilities. While a plausible scenario, the implications for user security would never be worse than the current state of affairs, where users are kept mostly in the dark and attackers need not be concerned about being detected by users.

7 FUTURE WORK

The ultimate aim of this project is to create user interface elements that help build users' understanding of the system security state. We expect that these will make use of techniques borrowed from the existing UI paradigms visualization and Ecological Interface Design (particularly the abstraction hierarchy), and perhaps others.

We plan to make prototypes of these security-mental-model-building UI elements aimed at solving real-world usable security problems such as those described in our case studies (Section 4). We will test these prototypes with real users and compare the security behaviour and knowledge outcomes of those who use the prototype against a baseline (e.g. those who use the existing, 'unenhanced' system).

To help us design intelligible security interfaces we will first study the knowledge people have about everyday, physical world security. This work is already underway: we conducted a focus group on everyday security behaviour [50], and we are in the midst of conducting a series of one-on-one interviews on the same topic. Our main finding in this work so far is that people tend to cluster their real-world security behaviour around certain key contexts, and we think these contexts map well to the digital domain. We think that designing interfaces with people's existing real-world security tendencies in mind can help improve security. In recent work, we described a number of UI design patterns that conform to our findings regarding everyday security behaviour [49], and these patterns will inform the design of our implementations of SAVE.

8 CONCLUSIONS

Many papers on usable security have commented on the problem of users having weak mental models of security issues. In this paper, we reviewed the major cognitive science literature on mental models to provide a situating framework for our discussions of what information users need, and we examine how this idea has been used in cybersecurity research. We presented three case studies of security problems which are the result of this (in)visibility of security problem, and suggest how it might be resolved through UI design. Finally, we discuss three approaches which we think will be useful in the development of more comprehensive solution.

Users must make decisions with security consequences, but their internal representations of the security-relevant aspects of software are not rich enough to enable safe behaviour. We suggest that user interfaces should provide better vision into the security-relevant inner workings of software to better support security behaviour. There are several challenges: we must not over-burden the user, and we must not expect the user to become a security expert. Instead, we must expose security considerations in a way that relates to the user's goals, and over time provide observable cause and effect relationships that will build helpful mental models. Where users bear responsibility for making security decisions, they must have the understanding to make those decisions, and it should be the design of the user interface that helps them develop that understanding.

REFERENCES

[1] Apple Inc. 2020. *Hardened Runtime.* Apple Inc. Retrieved May 23, 2020 from https://developer.apple.com/documentation/security/hardened_runtime

[2] Apple Inc. 2020. *Notarizing macOS Software Before Distribution.* Apple Inc. Retrieved May 22, 2020 from https://developer.apple.com/documentation/xcode/notarizing_macos_software_before_distribution

[3] Apple Inc. 2020. *What You Need To Enroll.* Apple Inc. Retrieved May 17, 2020 from https://developer.apple.com/programs/enroll/

[4] Farzaneh Asgharpour, Debin Liu, and L. Jean Camp. 2007. Mental Models of Security Risks. In *Financial Cryptography and Data Security* (Scarborough, Trinidad and Tobago) *(FC '07)*, Sven Dietrich and Rachna Dhamija (Eds.). Springer, Berlin, Germany, 367–377.

[5] Lawrence W. Barsalou. 1999. Perceptual symbol systems. *Behavioral and Brain Sciences* 22, 4 (1999), 577–660. https://doi.org/10.1017/S0140525X99002149

[6] Simon Bell and Peter Komisarczuk. 2020. An Analysis of Phishing Blacklists: Google Safe Browsing, OpenPhish, and PhishTank. In *Proceedings of the Australasian Computer Science Week Multiconference* (Melbourne, Australia). ACM, New York, NY, USA, 1–11.

[7] Kevin B. Bennett, Adam Bryant, and Christen Sushereba. 2018. Ecological Interface Design for Computer Network Defense. *Human Factors* 60, 5 (2018), 610–625. https://doi.org/10.1177/0018720818769233 PMID: 29741960.

[8] Robert Biddle, P. C. van Oorschot, Andrew S. Patrick, Jennifer Sobey, and Tara Whalen. 2009. Browser Interfaces and Extended Validation SSL Certificates: An Empirical Study. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security* (Chicago, IL, USA) *(CCSW '09)*. ACM, New York, NY, USA, 19–30. https://doi.org/10.1145/1655008.1655012

[9] Jim Blythe and L Jean Camp. 2012. Implementing mental models. In *2012 IEEE Symposium on Security and Privacy Workshops* (San Francisco, CA, USA). IEEE, New York, NY, USA, 86–90.

[10] Cristian Bravo-Lillo, Lorrie Faith Cranor, Julie Downs, and Saranga Komanduri. 2011. Bridging the Gap in Computer Security Warnings: A Mental Model Approach. In *IEEE Symposium on Security and Privacy* (Oakland, CA, USA) *(SP '11)*. IEEE, New York, NY, USA, 18–26. https://doi.org/10.1109/MSP.2010.198

[11] British Association for the Advancement of Science (per Lord Kelvin). 1876. Thomson's (Lord Kelvin) First Tide Predicting Machine, 1876. https://collection.sciencemuseumgroup.org.uk/objects/co53901/thomsons-lord-kelvin-first-tide-predicting-machine-1876-tide-predictor

[12] Zheng Bu and Rob Rachwald. 2014. *Industry Perspectives: Ghost-Hunting with Anti-Virus.* FireEye, Inc. Retrieved May 20, 2020 from https://www.fireeye.com/blog/executive-perspective/2014/05/ghost-hunting-with-anti-virus.html

[13] CA/Browser Forum. 2018. Guidelines for the Issuance and Management of Extended Validation Certificates. https://cabforum.org/wp-content/uploads/CA-Browser-Forum-EV-Guidelines-v1.6.8.pdf

[14] L. Jean Camp. 2009. Mental Models of Privacy and Security. *IEEE Technology and Society Magazine* 28, 3 (2009), 37–46. https://doi.org/10.1109/MTS.2009.934142

[15] Nico Castelli, Corinna Ogonowski, Timo Jakobi, Martin Stein, Gunnar Stevens, and Volker Wulf. 2017. What Happened in my Home? An End-User Development Approach for Smart Home Data Visualization. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, CO, USA) *(CHI '17)*. ACM, New York, NY, USA, 853–866. https://doi.org/10.1145/3025453.3025485

[16] Jianjun Chen, Vern Paxson, and Jian Jiang. 2020. Composition Kills: A Case Study of Email Sender Authentication. In *29th USENIX Security Symposium* (Boston, MA, USA) *(USENIX Security '20)*. USENIX Association, Berkeley, CA, USA. https://www.usenix.org/conference/usenixsecurity20/presentation/chen-jianjun

[17] D. Cooper, S. Santesson, S. Farell, S. Boeyen, R. Housley, and W. Polk. 2008. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* RFC 5280. RFC Editor. https://www.rfc-editor.org/info/rfc5280

[18] Kenneth Craik. 1943. *The Nature of Explanation.* Cambridge University Press, Cambridge, United Kingdom.

[19] Albese Demjaha, Jonathan M Spring, Ingolf F Becker, Simon Parkin, and M Angela Sasse. 2018. Metaphors considered harmful? An exploratory study of the effectiveness of functional metaphors for end-to-end encryption. In *Workshop on Usable Security* (San Diego, CA, 2018-02-18) *(USEC '18)*. ISOC, Reston, VA, USA, 1–11.

[20] Simson Garfinkel and Heather R. Lipford. 2014. *Usable Security: History, Themes, and Challenges.* Morgan & Claypool, San Rafael, CA, USA. https://doi.org/10.2200/S00594ED1V01Y201408SPT011

[21] Mary L. Gick and Keith J. Holyoak. 1983. Schema Induction and Analogical Transfer. *Cognitive Psychology* 15, 1 (1983), 1 – 38. https://doi.org/10.1016/0010-0285(83)90002-6

[22] Google Chrome Help. 2020. *Check if a site's connection is secure.* Google. Retrieved May 17, 2020 from https://support.google.com/chrome/answer/95617?hl=en

[23] Google Chrome Help. 2020. *Manage warnings about unsafe sites.* Google. Retrieved May 17, 2020 from https://support.google.com/chrome/answer/99020?

[24] hoakley. 2019. *Last Week on my Mac: Notarization devalued?* The Eclectic Light Company. Retrieved May 14, 2020 from https://eclecticlight.co/2019/09/08/last-week-on-my-mac-notarization-devalued/

[25] hoakley. 2020. *Hardening and notarization finally arrive in Catalina.* The Eclectic Light Company. Retrieved May 14, 2020 from https://eclecticlight.co/2020/02/03/hardening-and-notarization-finally-arrive-in-catalina/

[26] Iulia Ion, Rob Reeder, and Sunny Consolvo. 2015. "...no one can hack my mind": Comparing Expert and Non-Expert Security Practices. In *Proceedings of the Eleventh Symposium On Usable Privacy and Security* (Ottawa, ON, Canada) *(SOUPS '15)*. USENIX Association, Berkeley, CA, USA, 327–346.

[27] John Johansen. 2020. *AppArmor Wiki: Home.* AppArmor. Retrieved May 20, 2020 from https://gitlab.com/apparmor/apparmor/-/wikis/home

[28] Philip N. Johnson-Laird. 1983. *Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness.* Harvard University Press, Cambridge, MA, USA.

[29] Ruogu Kang, Laura Dabbish, Nathaniel Fruchter, and Sara Kiesler. 2015. My Data Just Goes Everywhere: User Mental Models of the Internet and Implications for Privacy and Security. In *Proceedings of the Eleventh Symposium On Usable Privacy and Security* (Ottawa, ON, Canada) *(SOUPS '15)*. USENIX Association, Berkeley, CA, USA, 39–52.

[30] Predrag Klasnja, Sunny Consolvo, Jaeyeon Jung, Benjamin M. Greenstein, Louis LeGrand, Pauline Powledge, and David Wetherall. 2009. When I am on Wi-Fi, I am Fearless: Privacy Concerns & Practices in Everyday Wi-Fi Use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Boston, MA, USA) *(CHI '09)*. ACM, New York, NY, USA, 1993–2002.

[31] Mitchell Krog and Nissar Chababy. 2020. Phishing Domain Database. https://github.com/mitchellkrogza/Phishing.Database

[32] Mailvelope. 2020. *Mailvelope Homepage.* Mailvelope GmbH. Retrieved May 18, 2020 from https://www.mailvelope.com/en

[33] Nathan Malkin, Arunesh Mathur, Marian Harbach, and Serge Egelman. 2017. Personalized Security Messaging: Nudges for Compliance with Browser Warnings. In *2nd European Workshop on Usable Security* (Paris, France) *(EuroUSEC '17)*. Internet Society, Reston, VA, USA.

[34] Microsoft. 2017. *Digital Certificates.* Microsoft. Retrieved May 17, 2020 from https://docs.microsoft.com/en-us/windows-hardware/drivers/install/digital-certificates

[35] Microsoft. 2020. *Anti-spoofing protection in EOP.* Microsoft. Retrieved May 20, 2020 from https://docs.microsoft.com/en-us/microsoft-365/security/office-365-security/anti-spoofing-protection

[36] Microsoft. 2020. *Program Requirements - Microsoft Trusted Root Program.* Microsoft. Retrieved May 17, 2020 from https://docs.microsoft.com/en-us/security/trusted-root/program-requirements

[37] Sara Motiee, Kirstie Hawkey, and Konstantin Beznosov. 2010. Do Windows Users Follow the Principle of Least Privilege? Investigating User Account Control Practices. In *Proceedings of the Sixth Symposium on Usable Privacy and Security* (Redmond, WA) *(SOUPS '10)*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/1837110.1837112

[38] NeoFinder Developer Team. 2020. *The Gates to Hell: Apples Notarizing.* NeoFinder blog. Retrieved May 14, 2020 from https://www.cdfinder.de/guide/blog/apple_hell.html

[39] Patrick Nohe. 2019. *58% of Phishing Websites Now Use HTTPS.* The SSL Store. Retrieved May 17, 2020 from https://www.thesslstore.com/blog/58-of-phishing-websites-now-use-https/

[40] Don Norman. 2013. *The Design of Everyday Things: Revised and Expanded Edition.* Basic Books, Inc., Hachette, NY, USA.

[41] Donald A. Norman. 1986. Cognitive Engineering. In *User Centered System Design: New Perspectives on Human-computer Interaction*, Donald A. Norman and Stephen W. Draper (Eds.). CRC Press, Boca Raton, FL, USA, 266–290.

[42] Objective Development Software. 2020. *Little Snitch.* Objective Development Software GmbH. Retrieved May 5, 2020 from https://obdev.at/products/littlesnitch/

index.html

[43] Emilee Rader, Rick Wash, and Brandon Brooks. 2012. Stories as Informal Lessons about Security. In *Proceedings of the Eighth Symposium on Usable Privacy and Security* (Washington, D.C., USA) *(SOUPS '12)*. ACM, New York, NY, USA, Article 6, 17 pages. https://doi.org/10.1145/2335356.2335364

[44] Jens Rasmussen. 1987. Mental Models and the Control of Actions in Complex Environments. In *Selected papers of the 6th Interdisciplinary Workshop on Informatics and Psychology: Mental Models and Human-Computer Interaction 1* (Schärding, Austria) *(Risø-M, 2656)*. North Holland Publishing Co., Amsterdam, Netherlands, 41–69.

[45] Jens Rasmussen. 1999. Ecological Interface Design for Reliable Human-Machine Systems. *The International Journal of Aviation Psychology* 9, 3 (1999), 203–223. https://doi.org/10.1207/s15327108ijap0903_2

[46] Joshua Reynolds, Deepak Kumar, Zane Ma, Rohan Subramanian, Meishan Wu, Martin Shelton, Joshua Mason, Emily Stark, and Michael Bailey. 2020. Measuring Identity Confusion with Uniform Resource Locators. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Virtual conference). ACM, New York, NY, USA, 1–12.

[47] F. Sharevski, P. Treebridge, and J. Westbrook. 2019. Experiential User-Centered Security in a Classroom: Secure Design for IoT. *IEEE Communications Magazine* 57, 11 (2019), 48–53. https://doi.org/10.1109/MCOM.001.1900223

[48] David Canfield Smith, Eric Harslem, Charles Irby, Ralph Kimball, and Bill Verplank. 1982. Designing the Star user interface. *Byte* 7, 1982 (1982), 242–282.

[49] Eric Spero and Robert Biddle. 2019. Home and Away: UI Design Patterns for Supporting End-User Security. In *Proceedings of the 2020 European Conference on Pattern Languages of Programs* (Virtual conference) *(EuroPLoP '20)*. ACM, New York, NY, USA, 1–9. Forthcoming.

[50] Eric Spero and Robert Biddle. 2019. Security Begins at Home: Everyday Security Behaviour and Lessons for Cybersecurity Research. In *Proceedings of the 26th Conference on Pattern Languages of Programs* (Ottawa, ON, Canada) *(PLoP '19)*. ACM, New York, NY, USA, 1–9. Forthcoming.

[51] Eric Spero, Milica Stojmenović, Sonia Chiasson, and Robert Biddle. 2019. Control and Understanding in Malware and Legitimate Software. In *2019 APWG Symposium on Electronic Crime Research* (Pittsburgh, PA, USA) *(eCrime '19)*. IEEE, New York, NY, USA, 1–11.

[52] Milica Stojmenović and Robert Biddle. 2018. Hide-and-Seek with Website Identity Information. In *2018 16th Annual Conference on Privacy, Security and Trust* (Belfast, United Kingdom) *(PST '18)*. IEEE, New York, NY, USA, 1–6.

[53] Milica Stojmenović, Eric Spero, Temitayo Oyelowo, and Robert Biddle. 2019. Website Identity Notification: Testing the Simplest Thing that Could Possibly Work. In *17th Annual Conference on Privacy, Security and Trust* (Fredericton, NB, Canada) *(PST '19)*. IEEE, New York, NY, USA, 310–316.

[54] Christopher Thompson, Martin Shelton, Emily Stark, Maximilian Walker, Emily Schechter, and Adrienne Porter Felt. 2019. The Web's Identity Crisis: Understanding the Effectiveness of Website Identity Indicators. In *28th USENIX Security Symposium* (Santa Clara, CA, USA) *(USENIX Security '19)*. USENIX Association, Berkeley, CA, USA, 1715–1732.

[55] Toyota Motor Corporation. 2019. 2019 Prius Owner's Manual (OM47C35U). https://www.toyota.com/t3Portal/document/om-s/OM47C35U/pdf/OM47C35U.pdf

[56] Liam Tung. 2016. *Google Safe Browsing beats rivals but still only flags up 10 percent of hacked sites*. ZDNet. Retrieved May 17, 2020 from https://www.zdnet.com/article/google-safe-browsing-beats-rivals-but-still-only-flags-up-10-percent-of-hacked-sites/

[57] Kim J. Vicente and Jens Rasmussen. 1992. Ecological Interface Design: Theoretical Foundations. *IEEE Transactions on Systems, Man, and Cybernetics* 22, 4 (7 1992), 589–606. https://doi.org/10.1109/21.156574

[58] Rick Wash. 2010. Folk Models of Home Computer Security. In *Proceedings of the Sixth Symposium on Usable Privacy and Security* (Redmond, WA) *(SOUPS '10)*. ACM, New York, NY, USA, Article 11, 16 pages. https://doi.org/10.1145/1837110.1837125

[59] Alma Whitten and J.D. Tygar. 1999. Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. In *8th USENIX Security Symposium* (Washington, D.C., USA) *(SSYM '99, Vol. 348)*. USENIX Association, Berkeley, CA, USA, 169–184.

[60] [winglett]. 2019, October 26. *Can we talk about Apple and macOS notarization?* Reddit. Retrieved May 22, 2020 from https://www.reddit.com/r/gamedev/comments/dnjc17/can_we_talk_about_apple_and_macos_notarization/