User Account Access Graphs

Sven Hammann
Department of Computer Sciene
ETH Zurich
Switzerland
sven.hammann@inf.ethz.ch

Saša Radomirović
School of Science and Engineering
University of Dundee
UK
s.radomirovic@dundee.ac.uk

Ralf Sasse
Department of Computer Sciene
ETH Zurich
Switzerland
ralf.sasse@inf.ethz.ch

David Basin
Department of Computer Sciene
ETH Zurich
Switzerland
basin@inf.ethz.ch

ABSTRACT

The primary authentication method for a user account is rarely the only way to access that account. Accounts can often be accessed through other accounts, using recovery methods, password managers, or single sign-on. This increases each account's attack surface, giving rise to subtle security problems. These problems cannot be detected by considering each account in isolation, but require analyzing the links between a user's accounts. Furthermore, to accurately assess the security of accounts, the physical world must also be considered. For example, an attacker with access to a physical mailbox could obtain credentials sent by post.

Despite the manifest importance of understanding these interrelationships and the security problems they entail, no prior methods exist to perform an analysis thereof in a precise way. To address this need, we introduce *account access graphs*, the first formalism that enables a comprehensive modeling and analysis of a user's entire setup, incorporating all connections between the user's accounts, devices, credentials, keys, and documents. Account access graphs support systematically identifying both security vulnerabilities and lockout risks in a user's accounts. We give analysis algorithms and illustrate their effectiveness in a case study, where we automatically detect significant weaknesses in a user's setup and suggest improvement options.

CCS CONCEPTS

• **Security and privacy → Formal security models**; **Authentication**.

1 INTRODUCTION

There is a rich body of work on authentication methods for user accounts. An often ignored aspect is that most accounts do not exist in isolation, but are linked directly or indirectly to other accounts. For example, websites often allow password resets via e-mail, thereby linking control of the website account to control of the user's e-mail account. Other links between a user's accounts are reused passwords or single sign-on services such as *Sign In With Google*. The links between a user's accounts result in security problems that are not apparent when considering each account in isolation.

The links for accessing and recovering accounts extend into the physical world. Devices commonly have active account sessions or may be used for two-factor authentication. Thus, the physical location of a device and how that location is physically secured is an important factor in determining whether an attacker can obtain access to an account. Other aspects of the physical world must also be considered. For example, when a user loses access to an online banking account, a common recovery option is a code sent via physical mail. Obtaining this code then requires access to the mailbox, which, in turn, may require a physical key. Furthermore, some recovery options may require personal identification in the real world. Then, physical credentials such as identity documents combined with a person's appearance become relevant.

A user's entire setup of both digital accounts and credentials as well as physical documents, keys, and devices form an intricate web of connections. Attacks can arise from any weak link in these connections. A single weakly secured account, device, or physical element can be used as a *backdoor*, allowing an attacker to circumvent other strong authentication mechanisms.

Thus, from a security perspective, account recovery should be restrictive. However, users become frustrated if it takes too much time or effort to regain access to one of their accounts [16]. There is a conflict between providing security against adversaries and the risk of locking users out of accounts. There is no single optimal trade-off between these two aspects, since the context of the account within the user's entire setup must be considered.

Both security vulnerabilities and lockout risks in account connection setups arise due to links between accounts from different service providers, and thus cannot be detected locally by any particular service provider. Identifying such weaknesses requires stepping back and analyzing the big picture. However, despite the practical importance and ubiquity of account recovery mechanisms, there is a lack of systematic understanding of how to perform such an analysis. To find weaknesses within account connection setups and to manage their complexity, we need a precise model of the links

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3354193

between a user's credentials and accounts, encompassing both the digital and physical world.

Approach. We present *account access graphs*, a formalism for evaluating account setups with respect to security and recoverability. An account access graph is a directed graph. Each vertex represents either a credential or an account. An edge from a credential to an account denotes that the credential is used in an authentication mechanism of that account. An edge from one account to another denotes that the first account can be used to access the second, using for example a recovery method or single sign-on.

We provide a method for analyzing the security of accounts. First we present an algorithm to compute an account's *access base*, which contains all minimal sets of credentials necessary to obtain access to the account. Then, we show how to compute security scores from this access base. For this, we define *scoring schemes* to compute scores for each account based on initial scores assigned to credentials. These initial scores may be derived from a risk analysis, and should reflect the difficulty of compromising each credential. Our scoring schemes are not limited to numerical scores, and may take values from any partially ordered domain, depending on the properties that we wish to express. In particular, we give a security scoring scheme that measures how vulnerable accounts are to attackers of different types and skills.

We also provide a method for analyzing the recoverability of accounts. Namely, we give an algorithm to compute an account's *lockout base*, which contains all minimal sets of credentials that the user must lose to be locked out of her account permanently. We show how to use this to compute recoverability scores.

Finally, we define *predicates* that identify concrete weaknesses in an account setup based on the scores assigned by a scoring scheme. In particular, we formalize the concept of account *backdoors*.

Contributions. We provide the first method for the analysis of a user's entire account connection setup. This setup includes digital accounts and credentials, such as passwords, as well as physical components, such as devices, keys, mailboxes, and identity documents. We model all possibilities how one or more such components provide access to other components. This allows us to systematically identify subtle weaknesses in the overall setup that were previously hard to detect.

We show how to identify both security vulnerabilities and lockout risks for a wide range of scenarios. This includes attackers that can compromise credentials and directly exploit vulnerabilities of services, users that can lose credentials, and services that can become unavailable. Our formalism is general and thereby can be easily adapted to analyze the particular aspects of account connection setups that one is interested in.

We have implemented all presented algorithms. The code is available for download at [10] and can be run to directly reproduce all presented results.

Finally, we illustrate the effectiveness of our method in a case study showing that even seemingly simple parts of a user's setup contain weaknesses that are not apparent without rigorous analysis.

Outline. In Section 2, we describe the basic aspects of our model, namely accounts, credentials, and connections between them. We develop our formalism in Sections 3-6. In Section 3, we define account *access sets* upon which we compute *security scores* in Section 4. In Section 5, we define account *lockout sets* upon which we compute *recoverability scores*. In Section 6, we define *predicates* based on scoring schemes that identify weaknesses in an account connection setup. We present a case study in Section 7. We compare with related work in Section 8 and draw conclusions in Section 9.

2 SYSTEM MODEL

We explain the concepts that we model, and how we model them.

2.1 Basic concepts

Users are persons that interact with services and have *accounts* with these services. An account can be accessed by providing *credentials*. We shall take a broad view of these concepts to also model the physical world. An account provides access to a service or resource, which can be digital or physical. Credentials can also be digital or physical, and must be presented to access, or unlock, an account. For example, we can model a physical key unlocking a room by modeling the key as a credential, and physical access to the room, or a device located in the room, as an account.

A set of credentials *provides access* to an account if presenting these credentials is sufficient to access the account. Losing all credentials in a set of credentials *locks* a user *out* of an account if at least one of these credentials is necessary to access the account.

2.2 Graph model

We model a user's account setup as an *account access graph*. Vertices represent credentials or accounts. We model the relation of providing access to a vertex v as follows. When v represents an account, we consider the access mechanisms of v. We draw edges of the *same color* from vertices (credentials or accounts) that are *all* needed to access the account (i.e., multi-factor authentication). We draw edges of *different colors* from vertices that represent *alternative* access methods. When v represents a credential, we draw an edge to v when an account provides access to that credential. For example, a password manager is an account that provides access to its stored passwords.

Note that the semantics of colors is local to each target vertex: whether two edges have the same color is only relevant if they are edges to the same target vertex. We reuse colors for simplicity in our examples for different target vertices. We use different types of lines (dashed, dotted, and solid) for different colors to distinguish them also in the black-and-white version of this paper.

Definition 1. *An* account access graph *is a directed graph* $G = (V_G, E_G, C_G)$*, where* V_G *are vertices,* C_G *are colors, and* $E_G \subseteq V_G \times V_G \times C_G$ *are directed colored edges.*

Example 1. *We introduce our running example in Figure 1. There is a webshop account* $\mathrm{acc}_{\mathrm{shop}}$ *that can be accessed with password* $\mathrm{pwd}_{\mathrm{shop}}$ *or recovered from the e-mail account* $\mathrm{acc}_{\mathrm{mail}}$*. The e-mail account requires two-factor authentication with password* $\mathrm{pwd}_{\mathrm{mail}}$ *and a* code*. The* code *is generated by an authentication app on a (mobile)* device*. The* device *can be unlocked using either a fingerprint (*finger*) or a* PIN*.*

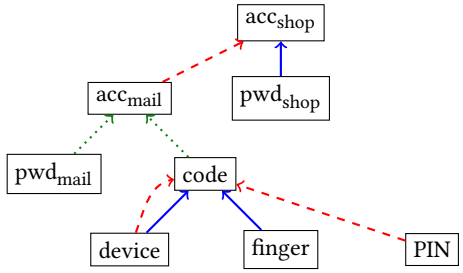

Figure 1: Same colors denote conjunction (all necessary for access), different colors denote disjunction (alternative)

3 ACCESS SETS

In this section, we define an account's access sets, which denote the minimal sets of credentials that are sufficient to provide (possibly transitively) access to the account. An account's access sets model the possibilities an attacker has to compromise the account. In Section 4, we use an account's access base to compute a security score for that account, enabling a detailed security evaluation.

We first define what it means that a set of vertices *provides access* to another vertex. A set of vertices V directly provides access to a vertex v if it includes v or includes, for some color c, all vertices with a c-colored edge to v.

Definition 2. *For an account access graph,* $\text{In}_c(v)$ *is the set of all vertices that have a* c*-colored edge to* v*. Formally, for an account access graph* $G = (V_G, E_G, C_G)$*,*

$$\text{In}_c(v) := \{v' \in V_G \mid e = (v', v, c) \in E_G\}.$$

We next define accessFrom(V) as the set of vertices that can directly or transitively be accessed from a set of vertices V.

Definition 3. *For an account access graph* G*, the set* accessFrom(V) *for a vertex set* $V \subseteq V_G$ *is the smallest set that satisfies* $V \subseteq \text{accessFrom}(V)$ *and is closed under the rule*

$$\frac{\exists c \in C_G : \emptyset \subsetneq \text{In}_c(v) \subseteq \text{accessFrom}(V)}{v \in \text{accessFrom}(V)} .$$

Example 2. *In the graph from Figure 1,*

$$\text{accessFrom}(\{\text{pwd}_{\text{mail}}, \text{device}, \text{PIN}\}) =$$
$$\{\text{pwd}_{\text{mail}}, \text{device}, \text{PIN}, \text{code}, \text{acc}_{\text{mail}}, \text{acc}_{\text{shop}}\} .$$

We now use accessFrom(V) to define AccessTo(v) as the sets of vertices that provide access to a vertex v.

Definition 4. AccessTo(v) *is the set of all sets of vertices that provide access to a vertex* v*, that is*

$$\text{AccessTo}(v) := \{V \subseteq V_G \mid v \in \text{accessFrom}(V)\} .$$

We call an element of AccessTo(v) *an* access set *of* v*.*

We use uppercase names for sets of sets, such as AccessTo(v), and lowercase names for sets, such as accessFrom(v). Constants, e.g., acc_{mail}, use lowercase except for acronyms, e.g., PIN.

Example 3. *For the graph in Figure 1,* AccessTo(acc_{mail}) *contains these sets, and all of their supersets:*

$$\{\text{acc}_{\text{mail}}\}, \{\text{pwd}_{\text{mail}}, \text{code}\}, \{\text{pwd}_{\text{mail}}, \text{device}, \text{finger}\},$$
$$\{\text{pwd}_{\text{mail}}, \text{device}, \text{PIN}\} .$$

As this example illustrates, AccessTo(v) may contain many elements, but we are often only interested in the *minimal* sets that provide access to a vertex.

Definition 5. *The* minimal access sets *of a vertex* v *are all minimal sets that provide access to* v*. Formally,* MinAccessTo(v) *:=*

$$\{V \subseteq V_G \mid V \in \text{AccessTo}(v) \wedge (\forall\, V' \subsetneq V : V' \notin \text{AccessTo}(v))\} .$$

For the graph in Figure 1, MinAccessTo(acc_{mail}) contains the sets listed in Example 3, but not any of their supersets.

We are particularly interested in those minimal access sets that suffice to gain access to a target account v and only contain vertices from a given set of initial vertices V_{init}. This set contains all vertices that a user may initially have access to, or that an adversary may compromise directly. That is, we assume that an adversary can only directly compromise the vertices in V_{init}, and must derive access to any other vertex using the interconnections modeled in the account access graph. In most of our examples with acyclic graphs, V_{init} is the set of all leaves. However, we show in Section 4.5 that V_{init} can also include intermediate vertices to model that an adversary could compromise accounts directly, i.e., without presenting any credentials, by exploiting a vulnerability in the underlying service.

Definition 6. *The* access base AccessBase(v, V_{init}) *of a vertex* v *with respect to a set of initial vertices* V_{init} *consists of the minimal access sets* V *that only contain vertices from* V_{init}*.*

$$\text{AccessBase}(v, V_{\text{init}}) := \{V \in \text{MinAccessTo}(v) \mid V \subseteq V_{\text{init}}\} .$$

An account's access base models all possibilities that an adversary has to compromise that account by compromising credentials or accounts in V_{init}. Each access set contained in the access base models one such possibility.

Example 4. *Let* V_{init} *be the set of all leaves in the graph from Figure 1.* AccessBase($\text{acc}_{\text{shop}}, V_{\text{init}}$) =

$$\{\ \{\text{pwd}_{\text{shop}}\}, \{\text{pwd}_{\text{mail}}, \text{device}, \text{finger}\}, \{\text{pwd}_{\text{mail}}, \text{device}, \text{PIN}\}\ \} .$$

Note that account access graphs may have cycles. We illustrate this with the following example.

Example 5. *The account access graph from Figure 2 contains an account* $\text{acc}_{\text{backup}}$ *on a backup platform. This account can be accessed using the password* $\text{pwd}_{\text{backup}}$ *and a* code *generated by a two-factor authentication app on a* device*. The account* $\text{acc}_{\text{backup}}$ *then provides access to the password* $\text{pwd}_{\text{manager}}$ *for a password* manager*, and the* seed *to reset the two-factor authentication app, usable to obtain a* code *without access to the* device*. The password* manager *provides access to the password* $\text{pwd}_{\text{backup}}$*.*

Note that the graph contains only a single leaf, device*. Setting* V_{init} *to all leaves would result in an empty access base for* $\text{acc}_{\text{backup}}$*: it is impossible to obtain access to that account starting from* device *only. A more natural initial set to consider is*

$$V_{\text{init}} := \{\text{pwd}_{\text{backup}}, \text{pwd}_{\text{manager}}, \text{device}, \text{seed}\} .$$

It contains all vertices that represent credentials in V_{init} but does not include vertices that represent accounts. Then,

$$\text{AccessBase}(\text{acc}_{\text{backup}}, V_{\text{init}}) = \{\{\text{pwd}_{\text{backup}},\ \text{device}\},\ \{\text{pwd}_{\text{backup}},\ \text{seed}\},\ \{\text{pwd}_{\text{manager}},\ \text{device}\},\ \{\text{pwd}_{\text{manager}},\ \text{seed}\}\ \}\ .$$

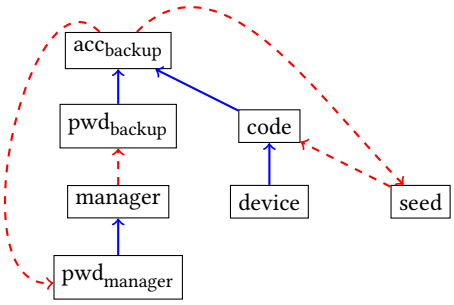

Figure 2

Algorithms. The set accessFrom(V) can be computed by starting from V and computing the least fixpoint of the rule from Definition 3. For a query $v \in^{?}$ accessFrom(V), computation can stop if $v \in$ accessFrom(V) is deduced before a fixpoint is reached. We compute AccessBase(v, V_{init}) as follows. We compute access sets of increasing size, skipping supersets of access sets computed previously. That is, we only perform a query of the form $v \in^{?}$ accessFrom(V) if there is no previously computed access set V' such that $V \supseteq V'$. The algorithms are described in more detail in Appendix B.

We also show in Appendix B how an account access graph can be translated into a theory of Horn clauses, and a query $v \in^{?}$ accessFrom(V) into an entailment problem $V \rightarrow^{?} v$. This enables us to leverage algorithms from the area, e.g., the linear time algorithm by Dowling and Gallier [7].

4 SECURITY SCORING SCHEMES

The security of a user's account setup depends on many factors. Not all accounts are equally valuable. Some users share their credentials with friends or family and other users are at particular risk of having their physical devices stolen. A security evaluation of a user's account setup should thus depend on the user's threat model. There can therefore be no single evaluation method that fits all use cases. For regular users, it may be sufficient to point out critical flaws in a setup that could easily be exploited. For users where the risk of targeted attacks is higher, such as reporters or politicians, analyzing security with respect to both local and remote attackers would be appropriate. Moreover, our account access graphs are general enough to be applicable not only to individual users but also to organizations. These organizations may want to perform an even more fine-grained analysis, incorporating our methods into their threat modeling process. They can also combine our methods with insights obtained from existing risk analysis.

To allow for a flexible yet expressive security analysis, we introduce *security scoring schemes*. These schemes provide a general method to evaluate an account's security using its access base. Recall that an account's access base consists of the credential sets that are sufficient to compromise that account. From these sets, security scoring schemes compute a security *score* for the account. A *score* is a general concept that can be defined over various domains. These domains are (partially) ordered, and a higher score denotes a more secure account.

We give three examples of scoring schemes of increasing complexity to illustrate the flexibility of our method. The first is a simple numerical score. The second is a generalization of the first to sets of multisets; this scheme is more expressive and allows for a more nuanced security analysis. The third is non-numerical, and considers the type of attacker and skill required to compromise an account or credential.

While there is no single security scoring scheme that is suitable for all account access graphs, we require that every scoring scheme must satisfy a simple soundness condition: If access to account A implies access to account B then A's score must be at least as high as B's. This condition supports what-if analyses on account access graphs. Even very basic schemes can provide useful feedback in the form of a simple "sanity check" that the user's highly valued accounts do not receive lower security scores than lower value accounts.

Formally, a security scoring scheme assigns a score to each vertex v based on that vertex's access base. Given initial scores of vertices in V_{init}, the scoring scheme evaluates each access set in v's access base, assigning it an intermediate score. It then combines the intermediate scores of all access sets in v's access base into a single score for v. Here, and in the remainder of the paper, we use $\mathcal{P}_{\mathcal{M}}(S)$ to denote the set of multisets over S, and $\mathcal{P}(S)$ to denote the powerset of S. Moreover, $\{\!\!\{\ \}\!\!\}$ denotes a multiset.

Definition 7. *A* security scoring scheme *for an account access graph G is a 6-tuple* $(D, \preceq, V_{\text{init}}, \text{Init}, \text{Eval}, \text{Combine})$*, where:*

- *D is the domain over which scores are defined.*
- *$\preceq$ is a partial order relating elements in D.*
- *$V_{\text{init}} \subseteq V_G$ is called the set of* initial vertices.
- *$\text{Init} : V_{\text{init}} \rightarrow D$ maps initial vertices to* initial scores.
- *$\text{Eval} : \mathcal{P}_{\mathcal{M}}(D) \rightarrow D$ maps a multiset of initial scores (of vertices in an access set) to an* intermediate score *(for that access set).*
- *$\text{Combine} : \mathcal{P}(D) \rightarrow D$ maps a set of intermediate scores (of access sets in a vertex's access base) to a* score *(for that vertex).*

Given Eval*, we define an auxiliary function* $\text{EvalSet} : \mathcal{P}(V_{\text{init}}) \rightarrow D$ *that directly maps an access set to its intermediate score by first computing the initial scores of its vertices and then applying* Eval.

$$\text{EvalSet}(S) := \text{Eval}(\{\!\!\{\text{Init}(v') \mid v' \in S\}\!\!\})\ .$$

We then define the following score function $\text{Score} : V_G \rightarrow D$*, which directly maps a vertex to its (final) score:*

$$\text{Score}(v) := \text{Combine}(\{\text{EvalSet}(S) \mid S \in \text{AccessBase}(v, V_{\text{init}})\})\ .$$

We usually instantiate Eval with a maximum or sum function, which captures an *and*-semantics: All credentials in an access set are necessary to access an account. We usually instantiate Combine with a minimum function, which captures an *or*-semantics: Any

one access set is sufficient to access an account. We formalize the expected semantics for Eval and Combine in Section 4.2.

4.1 A simple scoring scheme

We next give a simple scoring scheme that assigns a single natural number to each vertex.

Definition 8. *The* sum-then-min *scoring scheme is given as*

$$D = \mathbb{N},\ \preceq = \leq,\ \text{Eval}(M) = \sum_{m \in M} m,\ \text{Combine}(S) = \min_{s \in S}(s),$$

where $\leq$ *is standard comparison of natural numbers.*

Example 6. *We illustrate the scoring scheme with an example in Figure 3. We extend the running example introduced in Figure 1 by assigning initial security scores to the leaves.*

$$\text{Score}(\text{acc}_{\text{shop}}) = \text{Combine}(\text{EvalSet}(\{\text{pwd}_{\text{shop}}\}), \text{EvalSet}(\{\text{pwd}_{\text{mail}},\ \text{device},\ \text{finger}\}), \text{EvalSet}(\{\text{pwd}_{\text{mail}},\ \text{device},\ \text{PIN}\})\ ) = \min(\{1, 5, 4\}) = 1\ .$$

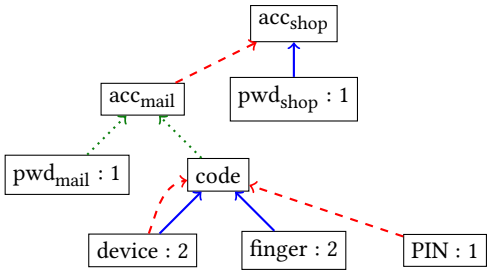

Figure 3

Unfortunately for many setups, this scoring scheme is too simplistic. We give an example that illustrates this.

Example 7. *In the graph in Figure 4, account* acc_A *requires two passwords, while* acc_B *requires a* code *generated by a* device. *We assign initial security scores to the leaves, and get*

$$\text{Score}(\text{acc}_\text{A}) = \text{Score}(\text{acc}_\text{B}) = 2\ .$$

Whether obtaining two passwords is more or less difficult than obtaining a device depends on many factors, and assigning the same score to both accounts oversimplifies the situation.

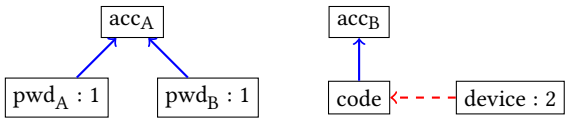

Figure 4

To solve this problem, we will give a more fine-grained scoring scheme in Section 4.3. Despite this problem, the sum-then-min scoring scheme respects the expected semantics of Eval and Combine mentioned before. We next formalize the necessary conditions for enforcing these semantics.

4.2 Requirements for scoring schemes

A security scoring scheme should meaningfully measure the security of accounts. In particular, whenever one account is at least as secure as another, a scoring scheme should assign at least as high a score to the first account as to the second. In this section, we formalize this requirement.

We define the notion of an account being *at least as secure* as another by implication. If an attacker that can access an account vertex v_B can also always access another account vertex v_A, then we say that v_B is at least as secure as v_A.

Example 8. *In Figure 5, consider an account where some features are available after only logging in with a password* pwd_acc. *We denote this basic access to the account by* acc_basic. *For security-critical features, a second factor* code *generated by a* device, *is necessary. We denote access to these features by* acc_full. *For* $V_\text{init} = \{\text{pwd}_\text{acc},\ \text{device}\}$,

$$\text{AccessBase}(\text{acc}_\text{basic}, V_\text{init}) = \{\{\text{pwd}_\text{acc}\}\},\ \textit{and}$$
$$\text{AccessBase}(\text{acc}_\text{full}, V_\text{init}) = \{\{\text{pwd}_\text{acc},\ \text{device}\}\}\ .$$

Whenever we can access acc_full, *we can also access* acc_basic. *Therefore,* acc_full *is at least as secure as* acc_basic.

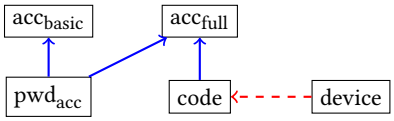

Figure 5

Based on this intuition, we next define a relation on access bases. We will then use this relation to define a soundness condition for scoring schemes.

Definition 9. *Let* v_A, v_B *be vertices in an account access graph* G. *An access base for a vertex* v_B *is* at least as secure *as that for* v_A *if and only if any set of vertices* V *that provides access to* v_B *also provides access to* v_A. *Formally,*

$$\text{AccessBase}(v_A, V_\text{init}) \preceq \text{AccessBase}(v_B, V_\text{init}) :\Leftrightarrow$$
$$\forall\, V \subseteq V_\text{init} : v_B \in \text{accessFrom}(V) \rightarrow v_A \in \text{accessFrom}(V)\ .$$

We also write $A \prec B$ to denote $A \preceq B \wedge A \neq B$. We now state a necessary and sufficient condition for $A \preceq B$.

Theorem 1. *Let* v_A, v_B *be vertices in an account access graph* G *and let* $V_\text{init} \subseteq V_G$ *be given. Let* $A := \text{AccessBase}(v_A, V_\text{init})$ *and* $B := \text{AccessBase}(v_B, V_\text{init})$. *Then*

$$(\forall B_i \in B\ \exists A_j \in A : A_j \subseteq B_i) \Leftrightarrow A \preceq B\ .$$

The proof is given in Appendix A.

Example 9. *In Example 8,*

$$\text{AccessBase}(\text{acc}_\text{basic}, V_\text{init}) \prec \text{AccessBase}(\text{acc}_\text{full}, V_\text{init})\ ,$$

since $\{\text{pwd}_\text{acc}\} \subseteq \{\text{pwd}_\text{acc},\ \text{device}\}$, *where* $\{\text{pwd}_\text{acc},\ \text{device}\}$ *is the only element in* $\text{AccessBase}(\text{acc}_\text{full}, V_\text{init})$.

Based on this, we define soundness of a scoring scheme.

Definition 10. *A security scoring scheme* $(D, \preceq_S, V_{\text{init}}, \text{Init}, \text{Eval}, \text{Combine})$ *with score function* Score_S *is* sound *if, for any two vertices* v_A *and* v_B

$$\text{AccessBase}(v_A, V_{\text{init}}) \preceq \text{AccessBase}(v_B, V_{\text{init}}) \Rightarrow \text{Score}_S(v_A) \preceq_S \text{Score}_S(v_B) .$$

We next give sufficient conditions for the Eval and Combine functions for a scoring scheme to be sound.

Theorem 2. *A security scoring scheme* $(D, \preceq, V_{\text{init}}, \text{Init}, \text{Eval}, \text{Combine})$ *is sound if the following two conditions hold.*

$$(1)\ \forall A, B \subseteq V_{\text{init}} : A \subseteq B \Rightarrow \text{EvalSet}(A) \preceq \text{EvalSet}(B) .$$

$$(2)\ \forall S, T \in \mathcal{P}(D) : (\forall T_i \in T\ \exists S_j \in S : S_j \preceq T_i) \Rightarrow \text{Combine}(S) \preceq \text{Combine}(T) .$$

The proof is given in Appendix A. All scoring schemes that we present in this paper are sound. The soundness proofs are given in Appendix A as well. We next present a scoring scheme that is more expressive than the previous one, addressing the problem illustrated in Example 7.

4.3 A multiset-based scoring scheme

We present a scoring scheme where each score is a set of multisets S. Each multiset in an account's score S represents an alternative access method for that account. Thus, this scoring scheme considers more details about an account's access sets than the previous scheme, which reduced the information from all different access sets to a single number.

Consider a multiset M that is associated with one access method. Each element in the multiset represents a credential as a single numeric value, where a higher number represents a credential that is more difficult to compromise. For example, $\{\!\{1, 1\}\!\}$ denotes an access method that requires two credentials with value 1 each. Then, the score S of an account represents all different access methods for that account, including (transitively) the different access methods for accounts linked to S. For example, a score of $\{\{\!\{1, 1\}\!\}, \{\!\{2\}\!\}\}$ denotes that the account can (transitively) be accessed either by two credentials with value 1 each, or a single credential with value 2.

For comparing scores, we first define a partial order on multisets. For two multisets M and N, $M \preceq N$ holds if and only if each number in M can be injectively mapped to a number in N that is at least as high. Then, M represents a less (or equally) secure access method than N.

Definition 11. *For two multisets* M *and* N *over* $\mathbb{N}$, $M \preceq N$ *if and only if* $k = |M| \leq |N| = n$, *and there exists an indexing of their elements such that* $M = \{\!\{m_1, \ldots, m_k\}\!\}$, $N = \{\!\{n_1, \ldots, n_n\}\!\}$, *and* $\forall\ 1 \leq i \leq k : m_i \leq n_i$.

Example 10. $\{\!\{1, 1\}\!\} \prec \{\!\{1, 2\}\!\}$ *and* $\{\!\{1, 2\}\!\} \prec \{\!\{1, 1, 2\}\!\}$, *but* $\{\!\{1, 2\}\!\}$ *and* $\{\!\{1, 1, 1\}\!\}$ *are incomparable.*

Note that this multiset ordering is specialized to our needs and differs from standard multiset orderings [5].

We next define a partial order on *sets of multisets* based on the same idea of implication for comparing access bases explained in Theorem 1. Consider a set of multisets S that represents a score for an account. Recall that each multiset in S is associated with an *alternative* access method. Thus, a set of multisets S_1 represents a lower score than another set S_2 if and only if, for every multiset in S_2, there is a multiset representing a less secure access method in S_1.

Definition 12. *For two sets of multisets* S_1, S_2, $S_1 \preceq S_2$ *if and only if* $\forall\ N \in S_2\ \exists M \in S_1 : M \preceq N$.

Note that the empty set is the highest possible score, since it denotes an account that cannot be accessed at all.

Example 11. $\{\{\!\{1, 1\}\!\}, \{\!\{1, 2\}\!\}\} \prec \{\{\!\{1, 2\}\!\}, \{\!\{1, 1, 2\}\!\}\}$, *but* $\{\{\!\{1, 2\}\!\}\}$ *and* $\{\{\!\{1, 1\}\!\}, \{\!\{1, 1, 1\}\!\}\}$ *are incomparable.*

To define our scoring scheme, we will employ a definition of distributed product from [15] for Eval:

Definition 13. *The distributed product of two sets of multisets* S_1 *and* S_2 *is defined as*

$$S_1 \otimes S_2 = \{M \uplus N \mid M \in S_1, N \in S_2\} ,$$

where $\uplus$ *denotes multiset sum (union). Moreover, we define* $\bigotimes_{S_i \in S}$ *as the generalization of* $\otimes$ *with unit element* $\{\emptyset\}$, *where* S *is a multiset containing sets of multisets.*

Example 12.

$$\bigotimes \{\!\{\{\{\!\{1\}\!\}\}, \{\{\!\{2, 2\}\!\}\}, \{\{\!\{3\}\!\}, \{\!\{4\}\!\}\}\}\!\} = \{\{\!\{1, 2, 2, 3\}\!\}, \{\!\{1, 2, 2, 4\}\!\}\} .$$

Definition 14. *The* sets of multisets *scoring scheme is given as*

- $D = \mathcal{P}_{\mathcal{M}}(\mathbb{N})$, *the sets of multisets over* $\mathbb{N}$.
- $\preceq$ *is given according to Definition 12.*
- $\text{Eval} = \bigotimes$.
- $\text{Combine}(S) = \text{minimal}(S_\cup)$, *where* $S_\cup := \bigcup_{S_i \in S} S_i$, *and*

$$\text{minimal}(S_\cup) := \{M \mid M \in S_\cup \wedge \neg(\exists M' \in S_\cup : M' \prec M)\} .$$

Eval takes the distributed product of sets of multisets given in Definition 13. Combine then takes the minimal values of the union. For example, given a set of two multisets $\{M_1, M_2\}$ with $M_1 \prec M_2$, M_1 is a minimal value but M_2 is not, so $\text{minimal}(\{M_1, M_2\}) = \{M_1\}$.

This scoring scheme addresses the problem illustrated in Example 7, where two accounts with very different setups were assigned the same score.

Example 13. *Consider the graph in Figure 6.*

$\text{Score}(\text{acc}_\text{A}) = \text{Combine}(\text{EvalSet}(\{\text{pwd}_\text{A}, \text{pwd}_\text{B}\})) = \{\{\!\{1, 1\}\!\}\}$, *and*
$\text{Score}(\text{acc}_\text{B}) = \text{Combine}(\text{EvalSet}(\{\text{device}\})) = \{\{\!\{2\}\!\}\}$.

Note that $\{\{\!\{1, 1\}\!\}\}$ and $\{\{\!\{2\}\!\}\}$ are incomparable. Thus, the scoring scheme assigns incomparable scores to setups that should be considered incomparable.

While this scoring scheme is more fine-grained, it still requires assigning initial numerical scores to credentials. These numbers can be obtained from a risk analysis, but may not always be precise. We next present a scoring scheme with a non-numerical domain based on an attacker model, illustrating the flexibility of our formalism.

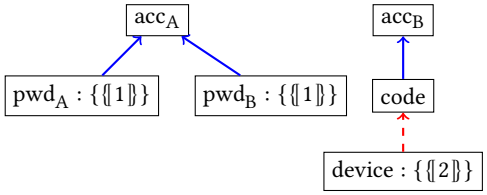

Figure 6

4.4 An attacker model scoring scheme

An important measure of an account's security is which kinds of attackers could potentially compromise the account. We consider the *possibility* of compromise by different kinds of attackers. We associate each account with the weakest attacker that could potentially compromise that account. Note that possibility here only means that such a compromise could occur, but not necessarily that it is likely. For example, an account that is only protected by a password could be compromised by a *remote* attacker, while an account that requires authentication from a device not connected to the Internet could only be compromised by a *local* attacker.

We formalize these concepts with a security scoring scheme based on n attribute sets $A_1, \ldots, A_n$ modeling attacker capabilities. We model an attacker as an n-tuple $(a_1, \ldots, a_n) \in A_1 \times \cdots \times A_n$.

Example 14. *Consider the following attribute sets:*

- Location = {rem, loc}, *with* rem < loc .
- Skill = {none, some, exp}, *with* none < some < exp .

For Location, rem *means the attacker acts remotely, e.g., from another country, while* loc *means that the attacker is local, and may thus obtain access to physical devices owned by the user. The relation* rem < loc *means that a local attacker has more capabilities than a remote one, that is, a local attacker could always also act like a remote attacker. For* Skill, none *means that the attacker has no special skills,* some *means that he has special skills, e.g., he can exploit known vulnerabilities, and* exp *denotes an expert hacker. For example, the tuple* (rem, some) *models a remote attacker with special skills.*

We next give an ordering on such attacker tuples.

Definition 15. *Let $A_1, \ldots, A_n$ be totally ordered attribute sets with order relations $\leq_1, \ldots, \leq_n$. Then, for $t, u \in A_1 \times \cdots \times A_n$, we define the following partial order $\preceq$ on tuples: $t \preceq u$ if and only if each component of t is less than or equal to the corresponding component in u. That is:*

$$t \preceq u :\Leftrightarrow \forall 1 \leq i \leq n : proj_i(t) \leq_i proj_i(u) \,,$$

where $proj_i(t)$ maps t to its i-th component.

We will define a scoring scheme that denotes the minimal attribute values an attacker must have to compromise a credential or account. The values assigned to vertices by the scoring scheme are *sets of attribute tuples.* Each tuple denotes one kind of attacker who could compromise the account. An attacker who has only better attributes as denoted in a tuple could also compromise the account.

Example 15. *An account with a score of* {(rem, some), (loc, none)} *could be compromised by a remote attacker with special skills, represented by* (rem, some)*, or by a local attacker without any special skills, represented by* (loc, none)*. An attacker stronger than at least one of these options, for example a remote attacker with expert skills,* (rem, exp)*, or a local attacker with some skills,* (loc, some)*, could also compromise the account. However, a remote attacker without special skills,* (rem, none)*, cannot compromise the account.*

We can also compare sets of attribute tuples as follows. Consider a set of attacker tuples S_2. Another set S_1 is smaller or equal to S_2 if and only if for each attacker tuple in S_2, there is a tuple representing a weaker (or equal) attacker, in S_1. That is, any attacker that could compromise the account with score S_2 could also compromise the account with score S_1.

Definition 16. *For $S_1, S_2 \in \mathcal{P}(A_1 \times \cdots \times A_n)$, $S_1 \preceq S_2$ if and only if $\forall\, u \in S_2\ \exists\, t \in S_1 : t \preceq u$.*

This ordering is analogous to the ordering for sets of multisets. The empty set is the highest possible score, since it denotes that *no* attacker could ever compromise the account.

Definition 17. *Let $A_1, \ldots, A_n$ be totally ordered attribute sets. We define the* attacker attribute *security scoring scheme based on these attribute sets as follows:*

- *$D = \mathcal{P}(A_1, \ldots, A_n)$, sets of attribute tuples.*
- *$\preceq$ is given according to Definition 16.*
- *$\text{Eval}(M) = \text{compMax}(M_\cup)$, where $M_\cup := \bigcup_{M_i \in M} M_i$, and* compMax *is a singleton set that contains the component-wise maximum of tuple set $M_\cup$.*
- *$\text{Combine}(S) = \text{minimal}(S_\cup)$, where $S_\cup := \bigcup_{S_i \in S} S_i$, and*

$$\text{minimal}(S_\cup) := \{t \mid t \in S_\cup \wedge \neg(\exists t' \in S_\cup : t' \prec t)\} \,.$$

Note that minimal is defined analogous to the sets of multisets scoring scheme given in Definition 14.

Example 16. *In Figure 7, we extend the graph from Figure 1 by assigning attacker tuple security scores to the leaves.*

$\text{Score}(\text{acc}_{\text{shop}}) = \text{Combine}(\text{ EvalSet}(\{\text{pwd}_{\text{shop}}\}),$
$\text{EvalSet}(\{\text{pwd}_{\text{mail}}, \text{device}, \text{finger}\}), \text{EvalSet}(\{\text{pwd}_{\text{mail}}, \text{device}, \text{PIN}\}))$
$= \text{Combine}(\{(\text{rem}, \text{some})\}, \{(\text{loc}, \text{exp})\}, \{(\text{loc}, \text{some})\}) =$
$\text{minimal}(\{(\text{rem}, \text{some}), (\text{loc}, \text{exp}), (\text{loc}, \text{some})\}) = \{(\text{rem}, \text{some})\} \,.$

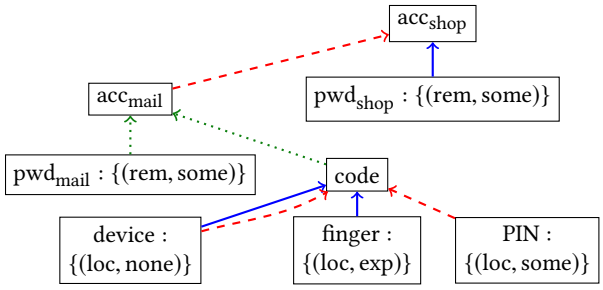

Figure 7

That is, a remote attacker with special skills could potentially compromise the acc_{shop} account.

4.5 Inherent account vulnerabilities

Our modeling and analysis so far was based on the assumption that an account can be accessed only by presenting credentials. In reality, services may have vulnerabilities that allow an attacker to access an account without credentials. We can model such vulnerabilities by also including intermediate vertices representing accounts in V_{init}, and assigning initial scores to them. An account's initial score denotes how difficult it is for an attacker to compromise it by means other than compromising the user's credentials.

Example 17. *In Figure 8, we extend the example given in Figure 3 by assigning initial security scores also to the intermediate vertices, setting* $V_{\text{init}} = V_G$. *Note that* $\text{AccessBase}(v, V_G) = \text{MinAccessTo}(v)$. *We assign an initial score of* 3 *to* acc_{shop} *and* acc_{mail}, *measuring the risk of direct account compromise through a service vulnerability. We assign an initial score of* 5 *to* code, *which means that we deem it difficult to compromise the* code *without access to the unlocked* device. *We then compute the access bases, which also contain intermediate vertices:*

$$\text{AccessBase}(\text{acc}_{\text{mail}}, V_G) = \{\{\text{acc}_{\text{mail}}\}, \{\text{pwd}_{\text{mail}}, \text{code}\}, \{\text{pwd}_{\text{mail}}, \text{device}, \text{finger}\}, \{\text{pwd}_{\text{mail}}, \text{device}, \text{PIN}\}\ \} .$$

$$\text{AccessBase}(\text{acc}_{\text{shop}}, V_G) = \{\{\text{acc}_{\text{shop}}\}, \{\text{acc}_{\text{mail}}\}, \{\text{pwd}_{\text{shop}}\}, \{\text{pwd}_{\text{mail}}, \text{code}\}, \{\text{pwd}_{\text{mail}}, \text{device}, \text{finger}\}, \{\text{pwd}_{\text{mail}}, \text{device}, \text{PIN}\}\ \} .$$

From these sets, we then compute the final security scores.

$$\text{Score}(\text{acc}_{\text{mail}}) = \min(\{3, 6, 5, 4\}) = 3 .$$

$$\text{Score}(\text{acc}_{\text{shop}}) = \min(\{3, 3, 1, 6, 5, 4\}) = 1 .$$

The score of 3 *for* acc_{mail} *is less than the score of* 4 *obtained previously.* $\text{Score}(\text{acc}_{\text{mail}}) = \text{Init}(\text{acc}_{\text{mail}})$, *i.e., the final score computed for* acc_{mail} *is equal to its initial score. This shows that the highest risk for the e-mail account is direct compromise, without compromising the user's credentials.*

The score of 1 *for* acc_{shop} *is the same as the score obtained previously.* $\text{Score}(\text{acc}_{\text{shop}}) < \text{Init}(\text{acc}_{\text{shop}})$, *i.e., the final score computed for* acc_{shop} *is lower than its initial score. This shows that compromise of the credentials is the highest risk for the web shop account (in particular, the credential* pwd_{shop}*).*

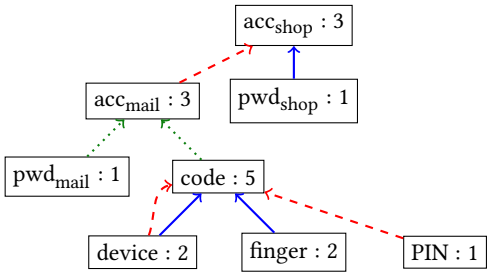

Figure 8

5 LOCKOUT SETS AND RECOVERABILITY

In the previous sections, we showed how to analyze the security of accounts against compromise. However, if security were our only concern, then accounts should not contain any recovery methods at all. In reality, security must be balanced against the risk of locking users out of their accounts. Thus, we now introduce a way to analyze *lockout risk*. We say that an account with low lockout risk achieves high *recoverability*. We can then analyze both security and recoverability of an account setup and understand whether the setup achieves a good balance between the two.

Lockout analysis is analogous to access analysis, and the main idea of the analogy is the following. To obtain access, an attacker must compromise *all* the necessary credentials for *one* access method. To be locked out, a user must lose *one* of the necessary credentials for *each* access method.

5.1 Definitions

We define the *minimal lockout sets* for a vertex. A lockout set for v is a set of vertices V such that, if the user does not have access to any credential in V and is locked out of all accounts in V, the user cannot access v. We define $\text{lockoutFrom}(V)$ for a set of vertices V analogous to $\text{accessFrom}(V)$. We assume that the user is initially locked out of all vertices in V. Then, $\text{lockoutFrom}(V)$ also contains all vertices that the user can no longer transitively access.

We construct $\text{lockoutFrom}(V)$ as follows. For a vertex v, consider all colors for which v has at least one incoming edge. When $\text{lockoutFrom}(V)$ contains the source vertex of an edge for each such color, then this means that the user lacks one vertex for each possible access mechanism of v. Thus, she cannot access v, so $v \in \text{lockoutFrom}(V)$.

Definition 18. *For an account access graph* G, *the set* $\text{lockoutFrom}(V)$ *for a vertex set* V *is the smallest set that satisfies* $V \subseteq \text{lockoutFrom}(V)$ *and is closed under the following rule:*

$$\frac{\exists c \in C_G : \text{In}_c(v) \neq \emptyset \wedge \forall c \in C_G : (\text{In}_c(v) \neq \emptyset) \rightarrow (\text{In}_c(v) \cap \text{lockoutFrom}(V) \neq \emptyset)}{v \in \text{lockoutFrom}(V)}$$

Example 18. *In Figure 8,* $\text{lockoutFrom}(\{\text{pwd}_{\text{shop}}, \text{device}\}) = \{\text{pwd}_{\text{shop}}, \text{device}, \text{code}, \text{acc}_{\text{mail}}, \text{acc}_{\text{shop}}\}$.

We next define the set $\text{Lockout}(v)$ in terms of $\text{lockoutFrom}(V)$ analogous to the relation between $\text{AccessTo}(v)$ and $\text{accessFrom}(V)$, and then define minimal lockout sets and lockout bases.

Definition 19. *The set of* lockout sets *of a vertex* v, $\text{Lockout}(v)$, *is defined as* $\text{Lockout}(v) := \{V \subseteq V_G \mid v \in \text{lockoutFrom}(V)\}$.

Definition 20. *We define the set of* minimal lockout sets *of a vertex* v *as* $\text{MinLockout}(v) :=$

$$\{V \subseteq V_G \mid V \in \text{Lockout}(v) \wedge (\forall\, V' \subsetneq V : V' \notin \text{Lockout}(v))\} .$$

Definition 21. *The* lockout base $\text{LockoutBase}(v, V_{\text{init}})$ *of a vertex* v *with respect to a set of initial vertices* V_{init} *consists of the minimal lockout sets that only contain vertices from* V_{init}.

$$\text{LockoutBase}(v, V_{\text{init}}) := \{V \in \text{MinLockout}(v) \mid V \subseteq V_{\text{init}}\} .$$

For lockout analysis, V_{init} denotes the vertices that a user might directly get locked out of. It should contain any credential that the user could lose or forget. If V_{init} also contains accounts, this models accounts that could be unavailable even if the user has all required

credentials. This models that the service provider shut down or is unavailable. We give examples for this in Section 5.2.

Example 19. *Let* V_{init} *be the set of all leaves in the graph from Figure 8.* $\text{LockoutBase}(\text{acc}_{\text{shop}}, V_{\text{init}}) =$

$$\{\ \{\text{pwd}_{\text{shop}}, \text{pwd}_{\text{mail}}\}, \{\text{pwd}_{\text{shop}}, \text{device}\}, \{\text{pwd}_{\text{shop}}, \text{finger}, \text{PIN}\}\ \}\ .$$

Algorithms. The algorithms for computing lockout sets are analogous to those for access sets. A vertex's lockout base can be computed analogously to computing an access base by answering queries of the form $v \in^{?} \text{lockoutFrom}(V)$. These queries can be answered by computing the least fixpoint of the rule given in Definition 18. A translation to Horn clauses analogous to that for access sets to leverage linear time algorithms is also given in Appendix B.

Scoring Schemes. Most concepts from the previous sections can naturally be adapted to apply to recoverability rather than security by replacing the access base with the lockout base. We define recoverability scoring schemes and their soundness. A higher recoverability score for an account denotes that a user is less likely to get locked out of the account.

Definition 22. *A* recoverability scoring scheme *is defined analogously to a security scoring scheme, with the score function operating on the lockout base rather than the access base:*

$$\text{Score}(v) := \text{Combine}(\{\text{EvalSet}(S) \mid S \in \text{LockoutBase}(v, V_{\text{init}})\})\ .$$

Definition 23. *Let* v_A *and* v_B *be vertices in an account access graph* G*. A lockout base for* v_B *is* at least as recoverable *as that for* v_A *when being locked out of* v_B *implies being locked out of* v_A*.*

$$\text{LockoutBase}(v_A, V_{\text{init}}) \preceq \text{LockoutBase}(v_B, V_{\text{init}}) :\Leftrightarrow$$
$$\forall\ V \subseteq V_{\text{init}} : v_B \in \text{lockoutFrom}(V) \rightarrow v_A \in \text{lockoutFrom}(V)\ .$$

Definition 24. *A recoverability scoring scheme* $(D, \preceq_R, V_{\text{init}}, \text{Init}, \text{Eval}, \text{Combine})$ *with score function* Score_R *is* sound *if, for any two vertices* v_A *and* v_B*,*

$$\text{LockoutBase}(v_A, V_{\text{init}}) \preceq \text{LockoutBase}(v_B, V_{\text{init}}) \Rightarrow$$
$$\text{Score}_R(v_A) \preceq_R \text{Score}_R(v_B)\ .$$

The numerical scoring schemes given in Definition 8 and Definition 14 can also be interpreted as recoverability scoring schemes. In this case, the initial scores assigned to credentials express how easily the user could lose the credential. A credential that could more easily be lost receives a lower score.

5.2 Inherent lockout risk

We mentioned that we consider inherent lockout from accounts, for example, due to a service provider shutdown. We can also reflect these possibilities with recoverability scoring schemes by including intermediate vertices in V_{init}. In Section 4.5, we considered additional *compromise* possibilities by giving the attacker more options: he might compromise an account without possessing sufficient credentials. Now, we consider additional *lockout* possibilities by giving the user fewer options: she might not be able to access an account despite having sufficient credentials. The difference can be subtle, and we illustrate it with the following examples.

Example 20. *In the graph from Figure 9, there is an account* acc_{SSO} *with a service that provides single sign-on, and an account* acc_{shop} *with a web shop using this single sign-on. For* $V_{\text{init}} := \{\text{pwd}_{\text{SSO}}, \text{acc}_{\text{SSO}}\}$,

$$\text{AccessBase}(\text{acc}_{\text{shop}}, V_{\text{init}}) = \text{LockoutBase}(\text{acc}_{\text{shop}}, V_{\text{init}}) =$$
$$\{\{\text{pwd}_{\text{SSO}}\}, \{\text{acc}_{\text{SSO}}\}\ \}\ .$$

Note the different interpretation of the set $\{\text{acc}_{\text{SSO}}\}$ *in the access base and the lockout base. In the access base, the set denotes that an attacker who directly compromises* acc_{SSO} *can access* acc_{shop}*. In the lockout base, it denotes that, when* acc_{SSO} *is (inherently) unavailable, the user is locked out of* acc_{shop}*.*

We apply the sum-then-min scoring scheme as a recoverability scoring scheme, and assign a score of 1 *to* pwd_{SSO}*, and a score of* 2 *to* acc_{SSO}*. Then, the final score for* acc_{shop} *is* 1*. This means that the highest lockout risk is due to the user losing or forgetting the password, and not due to a service provider shutdown. If we added recovery mechanisms to* acc_{SSO}*, then this could change, and service provider shutdown could become the greatest lockout risk.*

Figure 9

Example 21. *In the graph from Figure 10, there is an account* acc_{A} *with a password* pwd_{A} *that is saved in a password* manager*. For* $V_{\text{init}} := \{\text{manager}, \text{pwd}_{\text{A}}\}$,

$$\text{AccessBase}(\text{acc}_{\text{A}}, V_{\text{init}}) = \text{LockoutBase}(\text{acc}_{\text{A}}, V_{\text{init}}) =$$
$$\{\{\text{manager}\}, \{\text{pwd}_{\text{A}}\}\ \}\ .$$

There is again a different interpretation of the set $\{\text{pwd}_{\text{A}}\}$ *in the access base and the lockout base. In the access base, the set denotes that an attacker who directly compromises* pwd_{A} *can access* acc_{A}*. In the lockout base, however, it reflects the possibility that* pwd_{A} *could be unavailable such that it could not be retrieved* even with *access to the password* manager*. This models the accidental deletion of* pwd_{A} *from* manager*'s database. If this scenario is considered unlikely, then* pwd_{A} *should be assigned a higher initial recoverability score than passwords that are not stored in a password manager, and could thus be forgotten. If this scenario is even considered impossible, then* pwd_{A} *should not be included in* V_{init} *for computing lockout sets, and thus not be assigned an initial recoverability score at all.*

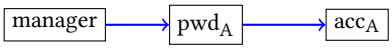

Figure 10

6 IDENTIFYING ACCOUNT SETUP WEAKNESSES

In this section, we show how to leverage scoring schemes to answer concrete questions about an account setup, such as which accounts contain *backdoors*, or whether more important accounts are always better secured than less important accounts. That is, we discover critical weaknesses without manually analyzing each score.

Formally, we define *predicates*, which are Boolean-valued functions on vertices that evaluate to true when there is a potential weakness. We will give example predicates for concrete weaknesses, but note that this is a general definition that can be instantiated to answer many relevant questions about account setups.

Definition 25. *An* n-vertex predicate *is a function* $P_{S,\text{Add}} : V_G^n \to \{\text{true}, \text{false}\}$ *that is defined with respect to a scoring scheme* S *and additional information* Add*, and takes as input an* n*-tuple of vertices (with* $n \geq 1$*, where a 1-tuple is a single vertex).*

6.1 Recovery paths and backdoors

One kind of weakness in an account access graph is a *backdoor*. An account has a backdoor when it can be accessed more easily using recovery access methods than using its primary authentication method.

We formalize this notion with a predicate. For this, we first explicitly define the subset of edges that are associated with recovery methods. We then consider for an account access graph G a reduced version G' of that graph that does not contain the recovery edges. An account has a backdoor with respect to a security scoring scheme if its score in G is lower than its score in G', i.e., it is easier to access this account by using at least one recovery method.

Definition 26. *Let* S *be a security scoring scheme with scoring function* Score *and* $E_{\text{rec}} \subsetneq E_G$ *a set of edges used in recovery methods. Let* G' *be the graph obtained from* G *by removing the edges in* E_{rec}*,* $G' := (V_G, E_G \setminus E_{\text{rec}}, C_G)$ *. Then, we define the following predicate.* $\text{HasBackdoor}_{S,E_{\text{rec}}}(v) := \text{Score}_G(v) < \text{Score}_{G'}(v)$.

Example 22. *In the account access graph* G *given in Figure 11, we evaluate the predicate* $\text{HasBackdoor}(\text{acc}_{\text{bank}})$*. We apply the sum-then-min security scoring scheme* S *with* V_{init} *containing all leaves. Let* $E_{\text{rec}} := \{(\text{acc}_{\text{mailA}}, \text{acc}_{\text{bank}}, \text{red}), (\text{acc}_{\text{mailB}}, \text{acc}_{\text{mailA}}, \text{red})\}$ *. Then, the backdoor predicate is evaluated as follows.*

$$\text{AccessBase}_G(\text{acc}_{\text{bank}}, V_{\text{init}}) = \{\{\text{pwd}_{\text{bank}}, \text{device}\}, \{\text{pwd}_{\text{mailA}}, \text{device}\}, \{\text{pwd}_{\text{mailB}}\}\} .$$

$$\text{AccessBase}_{G'}(\text{acc}_{\text{bank}}, V_{\text{init}}) = \{\{\text{pwd}_{\text{bank}}, \text{device}\}\} .$$

Thus, $\text{HasBackdoor}_{S,E_{\text{rec}}}(\text{acc}_{\text{bank}})$ *since*

$$\text{Score}_G(\text{acc}_{\text{bank}}) = 1 < 3 = \text{Score}_{G'}(\text{acc}_{\text{bank}}) .$$

The backdoor of acc_{bank} is not directly due to its recovery method, using $\text{acc}_{\text{mailA}}$, but due to the recovery method of $\text{acc}_{\text{mailA}}$, using $\text{acc}_{\text{mailB}}$. Thus, the predicate also identifies indirect backdoors.

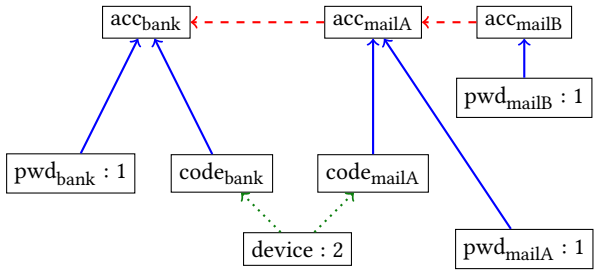

Figure 11

From a recoverability point of view, we are interested in the effectiveness of recovery methods. The analogous weakness to backdoors are *ineffective* recovery methods, namely those that do not actually improve an account's recoverability.

Definition 27. *Let* R *be a recoverability scoring scheme with scoring function* Score *and* $E_{\text{rec}} \subsetneq E_G$ *a set of edges used in recovery methods. Let* G' *be given as in Definition 26. Then, we define the predicate* IneffectiveRec *as follows:*

$$\text{IneffectiveRec}_{R,E_{\text{rec}}}(v) := \text{Score}_G(v) \preceq \text{Score}_{G'}(v) .$$

This predicate identifies accounts whose recoverability scores in G, the graph that contains recovery methods, are not better than in G', the graph without any recovery methods. Thus, the account's recovery methods are ineffective.

Example 23. *In the graph in Figure 12, the* device *provides access to two-factor codes as well as to a password* manager*. Let* $E_{\text{rec}} := \{(\text{acc}_{\text{mail}}, \text{acc}_{\text{shop}}, \text{red})\}$ *,* $V_{\text{init}} := \{\text{pwd}_{\text{mail}}, \text{pwd}_{\text{shop}}, \text{device}\}$ *. We apply the sum-then-min recoverability scoring scheme. We assign an initial score of* 1 *to* pwd_{mail} *and an initial score of* 2 *to the* device*. However, we assign an initial score of* 3 *to* pwd_{shop}*, as its initial score reflects the unlikely scenario of the password being deleted from the password* manager*'s database, as explained in Example 21.*

We obtain the following lockout bases.

$$\text{LockoutBase}_G(\text{acc}_{\text{shop}}, V_{\text{init}}) = \{\{\text{device}\}, \{\text{pwd}_{\text{mail}}, \text{pwd}_{\text{shop}}\}\} .$$

$$\text{LockoutBase}_{G'}(\text{acc}_{\text{shop}}, V_{\text{init}}) = \{\{\text{device}\}, \{\text{pwd}_{\text{shop}}\}\} .$$

This results in $\text{Score}_G(\text{acc}_{\text{shop}}) = \text{Score}_{G'}(\text{acc}_{\text{shop}}) = 2$ *and thus* $\text{IneffectiveRec}_{R,E_{\text{rec}}}(\text{acc}_{\text{shop}})$.

The reason is that device loss is the highest lockout risk with or without the recovery method. The recovery method only helps in case pwd_{shop} would be deleted from the password manager's database, a scenario we deemed unlikely.

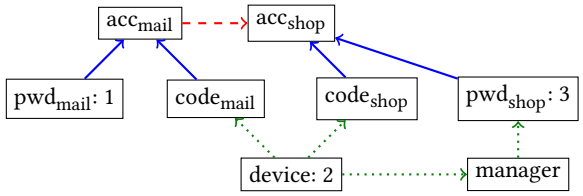

Figure 12

6.2 Account importance

Different accounts are of varying value and importance for the user. For example, an online banking account may be more valuable than other accounts. A risk analysis may indicate that a lower score is acceptable for a less important account. When importance values are assigned to each account, we can then combine this information with a scoring scheme to discover inconsistencies.

Definition 28. *Let* S *be a security or recoverability scoring scheme with a scoring function* Score *and let* $I : V \to D_I$ *be a function that*

assigns to a vertex an importance value from a partially ordered domain D_I. Then, the predicate Inconsistent *is given as follows.*

$$\text{Inconsistent}_{S,I}(v_1, v_2) := I(v_1) \succ I(v_2) \wedge \text{Score}(v_1) \preceq \text{Score}(v_2) .$$

That is, v_1 represents a more important account than v_2, but receives the same or a lower score value.

The definition is sensible for both security and recoverability scoring schemes, since it only depends on more important accounts receiving higher scores.

Example 24. *In Figure 13, we use the attacker security scoring scheme S given in Definition 17 and $D_I = \{\text{low}, \text{med}, \text{high}\}$ with the expected ordering. There are two accounts, which both require two-factor authentication. However,* acc_{bank}*, the more important account, can also be accessed with the* answers *to security questions.*

$I(\text{acc}_{\text{bank}}) = \text{high} \succ \text{med} = I(\text{acc}_{\text{shop}})$*, but*

$\text{Score}(\text{acc}_{\text{bank}}) = \{(\text{rem}, \text{some})\} \prec \{(\text{loc}, \text{some})\} = \text{Score}(\text{acc}_{\text{shop}})$.

Thus, $\text{Inconsistent}_{S,I}(\text{acc}_{\text{bank}}, \text{acc}_{\text{shop}})$.

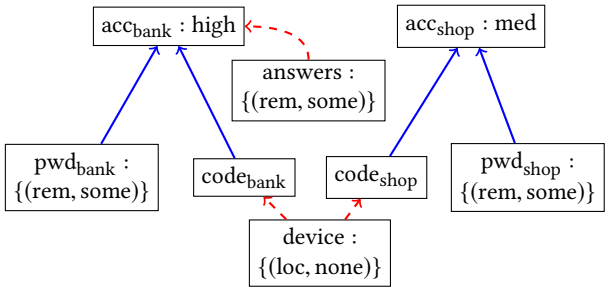

Figure 13

That is, even though acc_{bank} is the more important account compared with acc_{shop}, it is less secure.

7 CASE STUDY

We have performed an extensive case study on one of the author's account connection setups. We present here a reduced version of this case study, considering only a subset of the setup, for reasons of space and simplicity. The full version is presented in Appendix C. Even in the reduced setup, we still find several weaknesses, that we report upon here.

7.1 Setup

We consider a user who has a *Google* account as well as an account with the cryptocurrency exchange *Binance*. The user's account access graph is given in Figure 14.

Account structure. The user has a smartphone, a work laptop, and a home PC. The smartphone can be unlocked by entering a PIN or providing a fingerprint. The work laptop requires a password to be unlocked. For clarity, we here explicitly denote the unlocked version of a device d by d^*.

Login to the Google account requires two-factor authentication with an authenticator app or SMS, and a recovery e-mail address has been set up. Additionally, the user is logged into his Google account on his phone, his work laptop, and his home PC. We denote access to an existing session by $\text{Google}_{\text{basic}}$. Some functionality, such as changing security settings, requires authentication again. We denote access to this functionality by $\text{Google}_{\text{full}}$.

Furthermore, we discovered (in December 2018) that an old password and a code sent to the recovery e-mail address was sufficient to access the user's Google account via Tor [6]. While the same computer was used as for previous logins, the use of a different browser and Tor makes it unlikely (if not impossible) that the computer was used as an authentication factor. Thus, we model the possibility of logging in to Google with the old password and the code only.

Next, consider the user's Binance account. $\text{Binance}_{\text{basic}}$ denotes access to the account, which is protected with a password and a second-factor authentication code. $\text{Binance}_{\text{full}}$ additionally denotes access to cryptocurrency withdrawals, for which a verification link sent to the user's Gmail address must be followed. The account password can be reset by an e-mail sent to the user's Gmail address, so Gmail access can replace the password for accessing $\text{Binance}_{\text{basic}}$.

Scoring schemes. We use two different security scoring schemes and one recoverability scoring scheme. We include passwords, devices, and physical keys in V_{init}, that is

$$V_{\text{init}} := \{\text{homeKey}, \text{officeKey}, \text{phone}, \text{PIN}, \text{finger}, \text{pwd}_{\text{Google}}, \text{pwd}_{\text{Binance}}, \text{oldpwd}_{\text{Google}}, \text{pwd}_{\text{mail}}, \text{pwd}_{\text{laptop}}\} .$$

The first security scoring scheme is a simple attacker attribute scoring scheme according to Definition 17, with only the $\text{Location} = \{\text{rem}, \text{loc}\}$ attribute. We assign an initial score of loc to devices and keys, and rem to passwords.

The second security scoring scheme is a multiset-based scoring scheme according to Definition 14. The weakest credential, receiving a score of $\{\!\{1\}\!\}$, is $\text{oldpwd}_{\text{Google}}$. The reason is that an old password should not be considered secure; the user might have changed the password because the old one was compromised. We then assign a score of $\{\!\{2\}\!\}$ to other passwords and the phone, and a score of $\{\!\{3\}\!\}$ to the keys. This models that the user is more likely to leave his phone unattended than his keys.

The recoverability scoring scheme is also multiset-based, using similar scores. One difference is that we assign an initial score of $\{\!\{2\}\!\}$ to the keys only, modeling that the user is just as likely to lose his keys as his phone.

Additional information. The following edges belong to recovery methods.

$$E_{\text{rec}} := \{(\text{oldpwd}_{\text{Google}}, \text{Google}_{\text{full}}, \text{green}), (\text{acc}_{\text{mail}}, \text{Google}_{\text{full}}, \text{green}), (\text{Gmail}, \text{Binance}_{\text{basic}}, \text{green}), (\text{code}_{\text{Binance}}, \text{Binance}_{\text{basic}}, \text{green})\} .$$

Furthermore, we assign importance med to $\text{Binance}_{\text{basic}}$ and $\text{Google}_{\text{basic}}$ as well as importance high to $\text{Binance}_{\text{full}}$ and $\text{Google}_{\text{full}}$.

7.2 Evaluation

We evaluated the HasBackdoor and IneffectiveRec predicates on $\text{Binance}_{\text{basic}}$, $\text{Binance}_{\text{full}}$, $\text{Google}_{\text{basic}}$, and $\text{Google}_{\text{full}}$. We also evaluated the Inconsistent predicate on the pairs

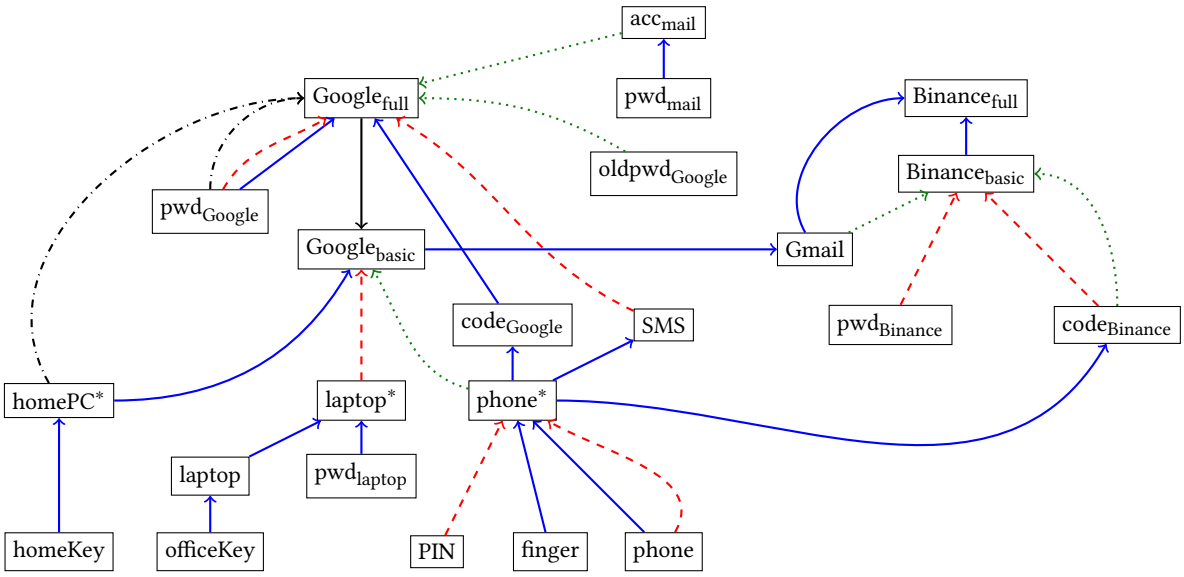

Figure 14

(Binance$_{\text{full}}$, Binance$_{\text{basic}}$) and (Google$_{\text{full}}$, Google$_{\text{basic}}$). We evaluated each predicate with respect to both given scoring schemes. Evaluation of all predicates including access and lockout base computation took an average of 60 seconds over 10 runs on a laptop computer with an Intel i7-6600U processor and 8 GB of RAM running Windows 10. We next present the most important identified weaknesses. The full results can be reproduced by the Haskell program available at [10].

We identified backdoors in Google$_{\text{full}}$ and Google$_{\text{basic}}$ with respect to both scoring schemes. For Google$_{\text{full}}$, the algorithm computed the following access bases. G is the original graph and G' is the graph after removing the edges belonging to recovery methods.

$$\text{AccessBase}_{G'}(\text{Google}_{\text{full}}, V_{\text{init}}) = \{\{\text{pwd}_{\text{Google}}, \text{phone}, \text{finger}\}, \{\text{pwd}_{\text{Google}}, \text{phone}, \text{PIN}\}, \{\text{pwd}_{\text{Google}}, \text{homeKey}\}\ \} .$$

$$\text{AccessBase}_{G}(\text{Google}_{\text{full}}, V_{\text{init}}) = \text{AccessBase}_{G'}(\text{Google}_{\text{full}}, V_{\text{init}}) \cup \{\{\text{oldpwd}_{\text{Google}}, \text{pwd}_{\text{mail}}\}\ \} .$$

For the attacker attribute scoring scheme, this results in

$$\text{HasBackdoor}_{S, E_{\text{rec}}}(\text{Google}_{\text{full}}), \text{ since}$$
$$\text{Score}_{G}(\text{Google}_{\text{full}}) = \{\text{rem}\} \prec \text{Score}_{G'}(\text{Google}_{\text{full}}) = \{\text{loc}\} .$$

That is, the primary authentication mechanisms could only be compromised by a local attacker, whereas the recovery mechanism is potentially vulnerable even to a remote attacker. The recovery mechanism thus constitutes a backdoor into the account. The user could fix this issue by removing the recovery e-mail option or by additionally securing the recovery e-mail account with a second factor device.

Furthermore, we identified an inconsistency between Binance$_{\text{full}}$ and Binance$_{\text{basic}}$. The accounts' access bases are equal.

$$\text{AccessBase}(\text{Binance}_{\text{full}}, V_{\text{init}}) = \text{AccessBase}(\text{Binance}_{\text{basic}}, V_{\text{init}}) = \{\ \{\text{pwd}_{\text{Binance}}, \text{phone}, \text{finger}\}, \{\text{pwd}_{\text{Binance}}, \text{phone}, \text{PIN}\}\ \} .$$

Thus, for any security scoring scheme S,

$$\text{Inconsistent}_{S, I}(\text{Binance}_{\text{full}}, \text{Binance}_{\text{basic}}), \text{ since}$$
$$I(\text{Binance}_{\text{full}}) \succ I(\text{Binance}_{\text{basic}}) \wedge$$
$$\text{Score}(\text{Binance}_{\text{full}}) = \text{Score}(\text{Binance}_{\text{basic}}) .$$

That is, the additional verification code sent to the Gmail account for withdrawing cryptocurrency does not add any security. The two-factor authentication already requires access to the unlocked smartphone, and there is an active Google account session on the phone, which is sufficient to access Gmail as well. The user could fix this issue by configuring a different e-mail address on Binance for which he does not have an active session on his phone or by logging out of the active Gmail session on his phone.

This case study illustrates how our analysis identifies concrete weaknesses of an account connection setup, and we also show how the user could improve his setup.

8 RELATED WORK

We consider three kinds of related work: (i) graph-based threat modeling, (ii) logic programming, and (iii) end-user authentication and account recovery in general.

First, there are many graph-based threat modeling formalisms that are related to our model, originating from safety engineering techniques such as fault trees [21]. The most popular model used for threat modeling today is attack trees [18, 20]. Different formalizations of attack trees have been given, such as that by Mauw et al. [15]. The survey by Kordy et al. [12] provides an extensive overview of attack and defense models based on directed acyclic graphs (DAGs). In contrast, account access graphs are general directed graphs, and our formalism naturally handles cycles. Furthermore, we model many concepts that are specific to the domain of account recovery and the evaluation of account security

and recoverability. Many existing techniques focus on a single system, and hierarchically refine a top-level event. Account access graphs do not have this hierarchical structure.

Second, logic programming [13] is also well-suited for problems of the kind we consider. In particular, its stable model semantics [8, 9], which is the basis of answer set programming [1, 14], is commonly used for problems of a similar form as our access and lockout set computation. For example, a query of the form $v \in^{?} accessFrom(V)$ can be answered using Horn clause resolution. Nevertheless, our domain-specific formalism offers many advantages. It facilitates visualization, and is well suited to directly analyze both access and lockout. The translation to Horn clauses given in Appendix B requires different sets for access and lockout, respectively, and also requires the introduction of auxiliary variables that do not directly correspond to any particular account or credential. In contrast, in our formalism, each vertex directly corresponds to an account or credential.

Third, there is extensive previous work on end-user authentication with services; the survey by Bonneau et al. [3] provides a good overview. However, there is substantially less work on account recovery, and existing work mostly belongs to one of two categories: empirical studies of existing systems and proposals for new systems. Since these are not directly related to our work, we only give a few examples. Bonneau et al. [2] and Rabkin [17] have conducted studies on security questions. Social recovery systems based on trustees have been proposed by Brainard et al. [4] and Schechter et al. [19]. Jakobsson et al. [11] suggest the use of personal preference questions as a replacement for traditional security questions. To our knowledge, there is no previous work analyzing a user's entire account connection setup.

9 CONCLUSION

We have presented account access graphs, the first formalism that enables the systematic analysis of a user's accounts encompassing the user's entire digital and physical context. Our formalism facilitates the discovery of account setup weaknesses: (i) with respect to security that could allow an attacker to compromise an account, and (ii) with respect to recoverability that could result in a user being permanently locked out of an account.

Our case study illustrates the complex and subtle nature of identified weaknesses. For example, we identified a weakness that is due to interconnections between a user's device, an active account session on that device, an account using the device for two-factor authentication, and a confirmation e-mail sent to one of the user's e-mail accounts. A user's entire account connection setup is typically much larger, and is likely to contain even more subtle weaknesses.

Our formalism is precise, simple, and general enough to support the representation of organizations', developers', and users' account setups and enables the evaluation of bespoke security and recoverability scoring schemes. It thus paves the way to a large-scale study of account graphs and the subsequent understanding and mitigation of account setup vulnerabilities.

REFERENCES

[1] Chitta Baral. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

[2] Joseph Bonneau, Elie Bursztein, Ilan Caron, Rob Jackson, and Mike Williamson. 2015. Secrets, Lies, and Account Recovery: Lessons from the Use of Personal Knowledge Questions at Google. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015*. International World Wide Web Conferences Steering Committee, 141–150.

[3] Joseph Bonneau, Cormac Herley, Paul C van Oorschot, and Frank Stajano. 2012. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *IEEE Symposium on Security and Privacy, SP 2012*. IEEE, 553–567.

[4] John G Brainard, Ari Juels, Ronald L Rivest, Michael Szydlo, and Moti Yung. 2006. Fourth-factor authentication: somebody you know. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006*. ACM, 168–178.

[5] Nachum Dershowitz and Zohar Manna. 1979. Proving termination with multiset orderings. *Commun. ACM* 22, 8 (1979), 465–476.

[6] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. *Tor: The second-generation onion router*. Technical Report. Naval Research Lab Washington DC.

[7] William F Dowling and Jean H Gallier. 1984. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *The Journal of Logic Programming* 1, 3 (1984), 267–284.

[8] Michael Gelfond and Vladimir Lifschitz. 1988. The Stable Model Semantics for Logic Programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, Vol. 88. MIT Press, 1070–1080.

[9] Michael Gelfond and Vladimir Lifschitz. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9, 3-4 (1991), 365–386.

[10] Sven Hammann, Saša Radomirović, Ralf Sasse, and David Basin. 2019. Haskell code for Account Access Graphs. https://infsec.ethz.ch/research/software/account_access_graphs.html.

[11] Markus Jakobsson, Erik Stolterman, Susanne Wetzel, and Liu Yang. 2008. Love and authentication. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 197–200.

[12] Barbara Kordy, Ludovic Pièlre-Cambacédès, and Patrick Schweitzer. 2014. DAG-based attack and defense modeling: Don't miss the forest for the attack trees. *Computer Science Review* 13 (2014), 1–38.

[13] John W Lloyd. 2012. *Foundations of logic programming*. Springer Science & Business Media.

[14] Victor W Marek and Miroslaw Truszczyński. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm*. Springer, 375–398.

[15] Sjouke Mauw and Martijn Oostdijk. 2005. Foundations of attack trees. In *International Conference on Information Security and Cryptology*. Springer, 186–198.

[16] Ron Miller. 2017. That time I got locked out of my Google account for a month. https://techcrunch.com/2017/12/22/that-time-i-got-locked-out-of-my-google-account-for-a-month/. Accessed: 2019-05-15.

[17] Ariel Rabkin. 2008. Personal knowledge questions for fallback authentication: Security questions in the era of Facebook. In *Proceedings of the 4th symposium on Usable privacy and security*. ACM, 13–23.

[18] Chris Salter, O Sami Saydjari, Bruce Schneier, and Jim Wallner. 1998. Toward a secure system engineering methodolgy. In *Proceedings of the 1998 workshop on New security paradigms*. ACM, 2–10.

[19] Stuart E Schechter, Serge Egelman, and Robert W Reeder. 2009. It's not what you know, but who you know: a social approach to last-resort authentication. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems, CHI 2009*. ACM, 1983–1992.

[20] Bruce Schneier. 1999. Attack trees. *Dr. DobbâĂŹs journal* 24, 12 (1999), 21–29.

[21] William E Vesely, Francine F Goldberg, Norman H Roberts, and David F Haasl. 1981. *Fault tree handbook*. Technical Report. Nuclear Regulatory Commission Washington DC.

A PROOFS

A.1 Requirements for scoring schemes

We prove Theorem 1.

Proof. We first show $\Rightarrow$: We want to show that, given the left-hand side, we have $A \preceq B$, that is:

$$\forall\, V \subseteq V_{\text{init}} : v_B \in \text{accessFrom}(V) \rightarrow v_A \in \text{accessFrom}(V)\ .$$

Consider any $V \subseteq V_{\text{init}}$ such that $v_B \in \text{accessFrom}(V)$. Then, V must contain all elements of some set in the access base B, i.e., we must have $\exists B_k \in B : B_k \subseteq V$. Thus, from the left-hand side, we have

$$\exists A_j \in A : A_j \subseteq B_k \subseteq V \Rightarrow A_j \subseteq V\ .$$

Thus, V contains all elements of A_j, an element of the access base of A, giving us $v_A \in \text{accessFrom}(V)$.

To show $\Leftarrow$, consider

$$\neg(\forall B_i \in B\ \exists A_j \in A : A_j \subseteq B_i) \Rightarrow \neg(A \preceq B)\ .$$

The left-hand side can be rewritten as:

$$\exists B_k \in B\ \forall A_j \in A : \neg(A_j \subseteq B_k)\ .$$

Consider such a B_k. We clearly have $v_B \in \text{accessFrom}(B_k)$ since B_k is a set in v_B's access base. We also have $\neg(v_A \in \text{accessFrom}(B_k))$ since B_k is not a superset of any set in v_A's access base. Thus,

$$v_B \in \text{accessFrom}(B_k) \wedge \neg(v_A \in \text{accessFrom}(B_k))$$

and hence

$$\exists V \subseteq V_{\text{init}} : v_B \in \text{accessFrom}(V) \wedge \neg(v_A \in \text{accessFrom}(V))\ ,$$

which is exactly $\neg(A \preceq B)$. □

We now prove Theorem 2.

Proof. Let v_A, v_B be two vertices in G. Let $A := \text{AccessBase}(v_A, V_{\text{init}})$ and $B := \text{AccessBase}(v_B, V_{\text{init}})$, and let $A \preceq B$. Let $S = \{\text{EvalSet}(A_i) \mid A_i \in A\}$ and $T = \{\text{EvalSet}(B_i) \mid B_i \in B\}$. We want to show

$$\text{Score}(v) = \text{Combine}(S) \preceq \text{Combine}(T) = \text{Score}(v')\ .$$

We show that, given conditions (1) and (2), as well as $A \preceq B$, that $\text{Combine}(S) \preceq \text{Combine}(T)$.

From $A \preceq B$ and Theorem 1, we have

$$\forall B_i \in B\ \exists A_j \in A : A_j \subseteq B_i\ .$$

Combining this with condition (1) yields

$$\forall T_i \in T\ \exists S_j \in S : S_j \preceq T_i\ .$$

From condition (2), this yields $\text{Combine}(S) \preceq \text{Combine}(T)$. □

A.2 Soundness proof for scoring schemes

We prove that all scoring schemes we presented are sound by showing that they fulfill the conditions given in Theorem 2.

Theorem 3. *The sum-then-min scoring scheme given in Definition 8 is sound.*

Proof. We show that the conditions of Theorem 2 are fulfilled.
Condition (1): Let $A \subseteq B$. We show that $\text{EvalSet}(A) \leq \text{EvalSet}(B)$:

$$\text{EvalSet}(A) = \sum_{v \in A} \text{Init}(v) \leq$$
$$(\sum_{v \in A} \text{Init}(v)) + (\sum_{v \in B \setminus A} \text{Init}(v)) =$$
$$\sum_{v \in B} \text{Init}(v) = \text{EvalSet}(B)\ .$$

The first step here is justified since all $\text{Init}(v)$ are non-negative, and the second step holds since, for $A \subseteq B$, we have $B = A \cup (B \setminus A)$.

Condition (2): When we have

$$\forall T_i \in T\ \exists S_j \in S : S_j \preceq T_i\ ,$$

then in particular, $\exists S_j \in S : S_j \preceq \min(T)$. Furthermore, $\min(S) \leq S_i$ for any $S_i \in S$. Thus,

$$\text{Combine}(S) = \min(S) \leq S_j \leq \min(T) = \text{Combine}(T)\ .$$

□

Theorem 4. *The set of multisets scoring scheme given in Definition 14 is sound.*

Proof. We show that the conditions of Theorem 2 are fulfilled.
Condition (1): Let $A \subseteq B$. We show that $\text{EvalSet}(A) \leq \text{EvalSet}(B)$.

$$\text{EvalSet}(A) = \bigotimes_{v \in A} \text{Init}(v) \leq$$
$$(\bigotimes_{v \in A} \text{Init}(v)) \otimes (\bigotimes_{v \in B \setminus A} \text{Init}(v)) =$$
$$\bigotimes_{v \in B} \text{Init}(v) = \text{EvalSet}(B)\ .$$

The first step holds since we have $S_1 \preceq S_1 \otimes S_2$ for any sets of multisets S_1 and S_2. This can be seen as follows: Any element of $S_1 \otimes S_2$ is of the form $M \uplus N$ with $M \in S_1$ and $N \in S_2$, and $M \uplus N \preceq M$. The second step holds since, for $A \subseteq B$, $B = A \cup (B \setminus A)$.

Condition (2): We assume

$$(*)\ \forall T_i \in T\ \exists S_j \in S : S_j \preceq T_i$$

and want to show

$$\text{Combine}(S) = \text{minimal}(S_\cup) \preceq \text{minimal}(T_\cup) = \text{Combine}(T)\ .$$

To show $\text{minimal}(S_\cup) \preceq \text{minimal}(T_\cup)$, we must show:

$$\forall N \in \text{minimal}(T_\cup)\ \exists M \in \text{minimal}(S_\cup) : M \preceq N\ .$$

Consider such a multiset $N \in \text{minimal}(T_\cup)$. There is a set of multisets $T_i \in T$ such that $N \in T_i$. Then, due to assumption (*), there is a set of multisets $S_j \in S$ such that $S_j \preceq T_i$. Thus, by definition of $\preceq$ on sets of multisets, there is a multiset $M_k \in S_j$ with $M_k \preceq N$. Then, either $M_k \in \text{minimal}(S_\cup)$, or $\exists M'_k \in \text{minimal}(S_\cup)$ such that $M'_k \preceq M_k \preceq N$. In either case, $\exists M : M \preceq N$. □

Theorem 5. *Let $A_1, \ldots, A_n$ be totally ordered attribute sets. Then, the attacker attribute scoring scheme given in Definition 17 based on these attribute sets is sound.*

Proof. We show that the conditions of Theorem 2 are fulfilled.

Condition (1): Let $A \subseteq B$. Let $M_A := \{\!|\text{Init}(v) \mid v \in A|\!\}$ and $M_B := \{\!|\text{Init}(v) \mid v \in B|\!\}$. Then,

$$\begin{aligned}\text{EvalSet}(A) = \text{compMax}_{t \in M_{A\cup}}(t) \preceq \\ \text{compMax}_{t \in M_{A\cup} \cup (M_{B\cup} \setminus M_{A\cup})}(t) = \\ \text{compMax}_{t \in M_{B\cup}}(t) = \text{EvalSet}(B)\,.\end{aligned}$$

The first step holds since, for each component, considering additional elements can only increase the maximum. The second step holds since, for $A \subseteq B$, we have $B = A \cup (B \setminus A)$, and thus also

$$M_{B\cup} = M_{A\cup} \cup (M_{B\cup} \setminus M_{A\cup})\,.$$

The proof for condition (2) is analogous to the proof of Theorem 4 for sets of multisets, since the proof only depends on the definition of Combine, which is analogous to the one for sets of multisets. □

B ALGORITHMS

We give details on our algorithms and their complexity.

B.1 Fixpoint computation for access and lockout

The function for computing whether $v \in^? \text{accessFrom}(V)$ for a set of vertices V is described in Algorithm 1, where

$$\begin{aligned}\text{accessFromStep}(V) := V \cup \\ \{v \mid \exists c \in C_G :\ \emptyset \subsetneq \text{In}_c(v) \subseteq V\}\end{aligned}$$

applies a single step of the rule given in Definition 3. Since $\text{accessFrom}(V)$ is defined as the smallest set closed under that rule, it can be computed as the least fixpoint of iterating this stepwise function.

Data: $G, V \subseteq V_G, v \in V_G$
Result: $v \in^? \text{accessFrom}(V)$
if $v \in V$ **then**
  return true ;
end
$V_{\text{next}} := V$;
repeat
  $V_{\text{prev}} := V_{\text{next}}$;
  $V_{\text{next}} := \text{accessFromStep}(V_{\text{prev}})$;
  if $v \in V_{\text{next}}$ **then**
    return true ;
  end
until $V_{\text{prev}} = V_{\text{next}}$;
return false ;

Algorithm 1: Algorithm for answering a query $v \in^? \text{accessFrom}(V)$

This algorithm runs in $O(n^2)$, where $n = |V_G|$, as accessFromStep(V) must consider all vertices, and $O(n)$ calls to accessFromStep(V) are necessary in the worst case. $v \in^?$ lockoutFrom(V) is computed in an analogous way, using a single step of the rule given in Definition 18.

We use Algorithm 2 for computing access bases. Lockout bases are computed analogously by replacing $v \in \text{accessFrom}(V)$ with $v \in \text{lockoutFrom}(V)$.

Data: $G, V_{\text{init}} \subseteq V_G, v \in V_G$
Result: AccessBase(v, V_{init})
AccessBase := $\emptyset$;
for $v' \in V_{\text{init}}$ **do**
  if $v \in \text{accessFrom}(\{v'\})$ **then**
    AccessBase := AccessBase $\cup$ $\{\{v'\}\}$;
  end
end
for $i := 2$ **to** $|V_{\text{init}}|$ **do**
  for $V \subseteq V_{\text{init}}$ *with* $|V| = i$ **do**
    for $V' \in$ AccessBase **do**
      if $V \supsetneq V'$ **then**
        skip V ;
      end
    end
    if $v \in \text{accessFrom}(V)$ **then**
      AccessBase := AccessBase $\cup$ $\{V\}$;
    end
  end
end
return AccessBase ;

Algorithm 2: Algorithm for computing an access base

These are the algorithms used in our implementation, and they are suitable for graphs the size of our case study. We next show how a better worst-case complexity can be achieved by translating account access graphs into Horn clauses, where different sets of Horn clauses are required for access and lockout, respectively.

B.2 Translation into Horn clauses

An account access graph can be translated into Horn clauses. This allows leveraging Horn clause resolution algorithms for access and lockout set computation. Different sets of Horn clauses are required for access and lockout computation, respectively.

Definition 29. *Let G be an account access graph. $\mathcal{V}(G)$ is called the set of G's* equivalent variables*, and is given as follows. For each vertex $v \in V_G$, we introduce a variable $v \in \mathcal{V}(G)$ and an additional auxiliary variable $v_c \in \mathcal{V}(G)$ for each color for which v has at least one incoming edge. That is, the set of variables is*

$$\mathcal{V}(G) := V_G \cup \{v_c \mid v \in V_G \wedge c \in C_G \wedge \text{In}_c(v) \neq \emptyset\}\,.$$

Definition 30. *Let G be an account access graph. $C_{\text{Access}}(G)$ is called the set of G's* equivalent access-clauses*, and is given as follows over the variables in $\mathcal{V}(G)$.*

$$\begin{aligned}C_{\text{Access}}(G) := \\ \{v_c \to v \mid v \in V_G,\ \text{In}_c(v) \neq \emptyset\} \cup \{\bigwedge_{v' \in \text{In}_c(v)} v' \to v_c\}\,.\end{aligned}$$

Theorem 6. *Let G be an account access graph. In the set of Horn clauses $C_{\text{Access}}(G)$, a variable v that corresponds to a vertex is entailed by the variables V corresponding to vertices, written $V \to v$, if and only if $v \in \text{accessFrom}(V)$.*

Proof. We first show that $v \in \text{accessFrom}(V)$ implies $V \to v$ for the Horn clause variables. We show this by structural induction

on the derivation tree of $v \in \text{accessFrom}(V)$ using the rule from Definition 3. The base case is $v \in V$. In this case, $v \in V$ also over the variables $\mathcal{V}(G)$ and thus trivially $V \to v$.

For the induction step, consider an application of the rule with conclusion $v \in \text{accessFrom}(V)$. From the induction hypothesis, for any previous deduction of $v' \in \text{accessFrom}(V)$, $V \to v'$ for the Horn clause variables. Now, consider the current rule's premise $\exists c \in C_G : \emptyset \subsetneq \text{In}_c(v) \subseteq \text{accessFrom}(V)$. Let c' be a fixed color such that $\emptyset \subsetneq \text{In}_{c'}(v) \subseteq \text{accessFrom}(V)$. For any $v' \in \text{In}_{c'}(v)$, $v' \in \text{accessFrom}(V)$, and thus, due to the induction hypothesis, $V \to v'$. We can therefore apply the Horn clause $\bigwedge_{v' \in \text{In}_{c'}(v)} v' \to v_{c'}$ to obtain $v_{c'}$, and then apply $v_{c'} \to v$ to obtain v.

We now show the converse by structural induction on the derivation of v by applying Horn clauses. The base case is again $v \in V$, in which case $v \in V$ also for the graph and thus $v \in \text{accessFrom}(V)$.

For the induction step, consider the derivation step that yields v. This step must apply a clause of the form $v_c \to v$, since any other clauses only yield auxiliary variables of the form v_c. Note that $v_c \notin V$, since V does not contain any auxiliary variables. Thus, v_c must have been derived from a clause of the form $\bigwedge_{v' \in \text{In}_{c'}(v)} v' \to v_{c'}$, where $\text{In}_c(v) \neq \emptyset$. Thus, we must have $V \to v_i$ for all v_i on the left-hand side of that clause. Due to the induction hypothesis, also $v_i \in \text{accessFrom}(V)$ for all of these v_i. Thus, $\emptyset \subsetneq \text{In}_{c'}(v) \subseteq \text{accessFrom}(V)$, and therefore also $\exists c \in C_G : \emptyset \subsetneq \text{In}_c(v) \subseteq \text{accessFrom}(V)$. We apply the rule from Definition 3 and obtain $v \in \mathit{accessFrom}(V)$.

□

Definition 31. *Let G be an account access graph. $C_{\text{Lockout}}(G)$ is called the set of G's* equivalent lockout-clauses*, and is given as follows over the variables in $\mathcal{V}(G)$.*

$$C_{\text{Lockout}}(G) := \{ \bigwedge_{\text{In}_c(v) \neq \emptyset} v_c \to v \mid v \in V_G,\ \exists c \in C_G : \text{In}_c(v) \neq \emptyset \} \cup \{ v' \to v_c \mid v' \in \text{In}_c(v) \} .$$

Theorem 7. *Let G be an account access graph. In the set of Horn clauses $C_{\text{Lockout}}(G)$, , a variable v that corresponds to a vertex is entailed by the variables V corresponding to vertices, written $V \to v$, if and only if $v \in \text{lockoutFrom}(V)$.*

Proof. We first show that $v \in \text{lockoutFrom}(V)$ implies $V \to v$ for the Horn clause variables. We show this by structural induction on the derivation tree of $v \in \text{lockoutFrom}(V)$ using the rule from Definition 18. The base case is $v \in V$. In this case, $v \in V$ also over the variables $\mathcal{V}(G)$ and thus trivially $V \to v$.

For the induction step, consider an application of the rule with conclusion $v \in \text{lockoutFrom}(V)$. By the induction hypothesis, for any previous deduction of $v' \in \text{lockoutFrom}(V)$, $V \to v'$ for the Horn clause variables. Now, consider the current rule's premise

$$\exists c \in C_G : \text{In}_c(v) \neq \emptyset \wedge \forall c \in C_G : (\text{In}_c(v) \neq \emptyset) \to (\text{In}_c(v) \cap \text{lockoutFrom}(V) \neq \emptyset) .$$

As $\exists c \in C_G : \text{In}_c(v) \neq \emptyset$, there exists a clause of the form $\bigwedge_{\text{In}_c(v) \neq \emptyset} v_c \to v$. From the rule's premise, for all such c with $\text{In}_c(v) \neq \emptyset$, $\text{In}_c(v) \cap \text{lockoutFrom}(V) \neq \emptyset$. Thus, for each such c, there is a $v' \in \text{In}_c(v)$ with $v' \in \text{lockoutFrom}(V)$. By the induction hypothesis, $V \to v'$ for all of these v'. We can therefore apply the Horn clause $v' \to v_c$ to obtain v_c for each c with $\text{In}_c(v) \neq \emptyset$, and then apply $\bigwedge_{\text{In}_c(v) \neq \emptyset} v_c \to v$ to obtain v.

We now show the converse by structural induction on the derivation of v by applying Horn clauses. The base case is again $v \in V$, in which case $v \in V$ also for the graph and thus $v \in \text{lockoutFrom}(V)$.

For the induction step, consider the derivation step that yields v. This step must apply a clause of the form $\bigwedge_{\text{In}_c(v) \neq \emptyset} v_c \to v$ (with $\exists c : \text{In}_c(v) \neq \emptyset$), since any other clauses only yield auxiliary variables of the form v_c. Note that $v_c \notin V$, since V does not contain any auxiliary variables. Thus, all the v_c's must have been derived from clauses of the form $v' \to v_c$, where $v' \in \text{In}_c(v)$. Thus, for each c with $\text{In}_c(v) \neq \emptyset$, $V \to v'$ for at least one such v' with $v' \in \text{In}_c(v)$. By the induction hypothesis, $v' \in \text{lockoutFrom}(V)$ for all of these v'. Thus, $\text{In}_c(v) \cap \text{lockoutFrom}(V) \neq \emptyset$ for each such c. Since also $\exists c : \text{In}_c(v) \neq \emptyset$, we obtain

$$\exists c \in C_G : \text{In}_c(v) \neq \emptyset \wedge \forall c \in C_G : (\text{In}_c(v) \neq \emptyset) \to (\text{In}_c(v) \cap \text{lockoutFrom}(V) \neq \emptyset) .$$

We apply the rule from Definition 18 to obtain $v \in \text{lockoutFrom}(V)$.

□

These results allow for the use of efficient decision procedures for Horn clause entailment, such as the linear time algorithm by Dowling and Gallier [7], for computing access and lockout bases.

C FULL CASE STUDY

We present the full version of the case study, a reduced version of which was presented in Section 7.

C.1 Model

To manage complexity and reduce access and lockout base computation times, we split the account graph into different parts, which we call layers. When two layers of the graph are independent of each other, i.e., no edge exists from one layer to another, they can be analyzed independently. We define a *base layer*, which is always included. Any other layer may either only depend on the base layer, or on other layers. Any layers it depends on must then also be included for its analysis.

Base layer. We depict the base layer in Figure 15. It includes physical devices, keys, and documents, as well as a mail account used for many recovery methods. The homeKey, officeKey, and mailboxKey are keys held on a physical keychain. We model the keychain as a separate credential that gives access to the keys, and include only the keychain in V_{init}. This models that one either has access to the whole keychain or to none of the keys at all. The user has a smartphone, a work laptop, and a home PC.

The smartphone can be unlocked by entering a PIN or providing a fingerprint. The work laptop requires a password to be unlocked. For a device d, we denote the unlocked device by d^*. Accessing the work laptop or home PC requires access to the respective location, so they are not included in V_{init} but access must be derived from the respective keys. The smartphone could be stolen directly, so it is included in V_{init}.

The credential IdDocument models an official document sufficient to identify the user, such as a passport, or, in some parts of the world, a driver's license. The inPerson credential models that

the user must go somewhere in person and that another person compares his appearance, e.g., with his portrait on an IdDocument. Furthermore, the physicalMailbox credential represents access to physical mail sent to the user, which is used in some recovery methods and is unlocked with the mailboxKey.

Google layer. We depict the Google layer in Figure 16. This layer contains the same vertices and edges as already described in the case study from Section 7.

Binance layer. We depict the Binance layer in Figure 17. This layer is also the same as already described in Section 7. This layer depends on the Google layer due to the connection from Gmail. To analyze it, we must thus include the Google layer in the graph.

University layer. We depict the university layer in Figure 18. This layer includes the user's university account. The main authentication method is a password, which is also saved on the user's laptop. The following recovery methods exist. The $\text{answers}_{\text{uni}}$ to security questions can be combined with either a code entered by SMS or sent via physical mail. Alternatively, the university's service desk offers help. We model that this recovery path requires the user to present an IdDocument in person.

Online Banking layer. We depict the online banking layer in Figure 19. The user's online banking works as follows. The homePC, laptop, and phone are configured as trusted devices.

The account $\text{banking}_{\text{basic}}$ allows read access and transactions to trusted parties. It can be accessed by entering a password on one of the trusted devices. The account $\text{banking}_{\text{full}}$ additionally allows transactions to untrusted parties, which must be confirmed using a second trusted device. This can be any trusted device that is not used for the main session.

The password can be reset with an activationCode. However, activationCode does not give access to the current password, it allows one to reset the password. To model this accurately, we denote by $\text{currentpwd}_{\text{banking}}$ the current password, which gives access to $\text{pwd}_{\text{banking}}$, but $\text{pwd}_{\text{banking}}$ can also be accessed by activationCode. We could alternatively allow activationCode to replace the password in each authentication method, but this would greatly increase the number of edges. An activationCode can be obtained by physical mail after answering some personal knowledge questions, denoted by $\text{answers}_{\text{banking}}$. An activationCode can also be used in combination with one trusted device to add an additional trusted device. Thus, it can be used to replace the second trusted device to provide access to $\text{banking}_{\text{full}}$.

Scoring Schemes. The scoring schemes used for the case study are an extension of those presented in Section 7. In particular, the attribute scoring scheme also considers attacker skill. We assign an initial score of (rem, some) to passwords, but of (rem, none) to answers for security questions. This denotes that compromising an average password usually requires at least some skills, while answers to security questions can often be guessed or may be gleaned from a person's online presence. We assign an initial score of (loc, some) to the keychain but of (loc, none) to the phone, modeling, as mentioned in Section 7, that the user is more likely to leave his phone unattended than his keys. Furthermore, we assign an initial score of (loc, exp) to the inPerson credential, denoting that it would require significant skills to physically impersonate the user in real life. This difficulty is also reflected in the multiset scoring, where we assign an initial score of $\{\{\![5]\!\}\}$ to inPerson. The full list of scores is given in the Haskell code available at [10].

C.2 Evaluation

We highlight some results here. The full results can be reproduced by compiling and running the Haskell code available at [10].

We presented the results for the *Google* and *Binance* layer in Section 7. We additionally note that we discovered backdoors to $\text{Binance}_{\text{basic}}$ and $\text{Binance}_{\text{full}}$ with respect to the multiset scoring scheme, but not the attribute scoring scheme. This is unsurprising since the multiset scoring scheme is more fine-grained, and thus detects finer differences. The backdoors reflect that $\text{pwd}_{\text{Binance}}$ can be circumvented with access to the phone by sending a recovery code to the Gmail account, for which the phone has an active session. This constitutes additional evidence that the active Gmail session is a weakness of the setup.

In the *banking* layer, we also discovered backdoors to $\text{banking}_{\text{basic}}$ and $\text{banking}_{\text{full}}$ with respect to the multiset scoring scheme, but not the attribute scoring scheme. These backdoors are due to the $\text{answers}_{\text{banking}}$ being able to replace the $\text{pwd}_{\text{banking}}$. A positive result is that $\text{banking}_{\text{basic}}$ and $\text{banking}_{\text{full}}$ can both only be compromised by a local attacker, namely one with access to the keychain.

For the *university* layer, we obtain another positive result: acc_{uni} contains no backdoors with respect to either scoring scheme, and the recovery methods increase the recoverability score. That is, the recovery methods manage to improve recoverability without decreasing security.

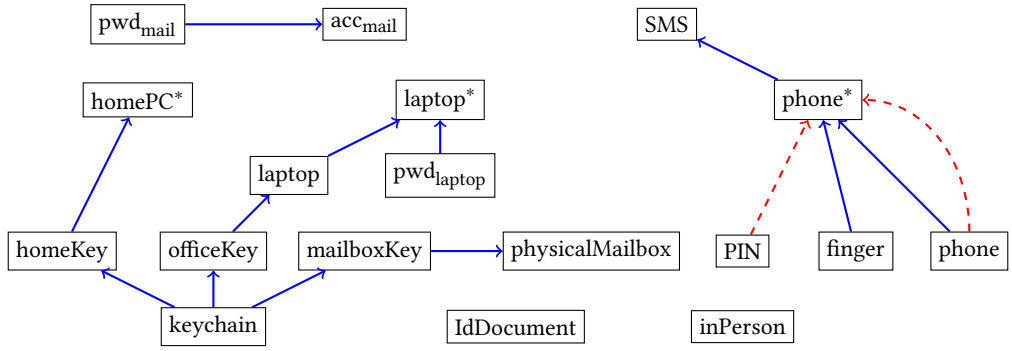

Figure 15: Base layer

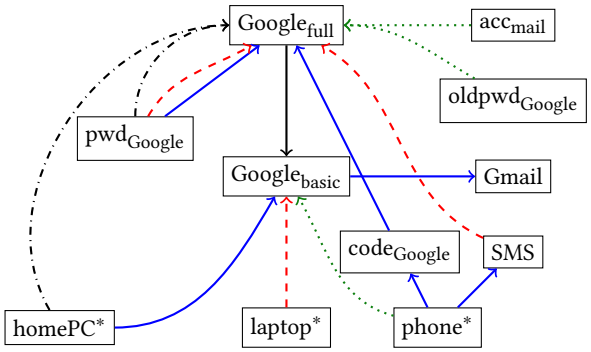

Figure 16: Google layer

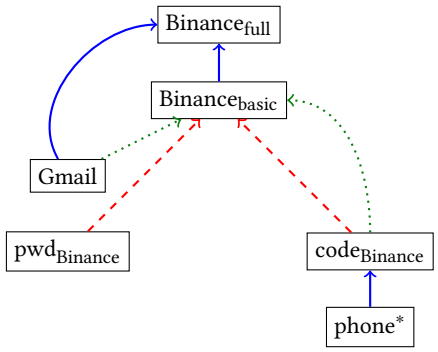

Figure 17: Binance layer

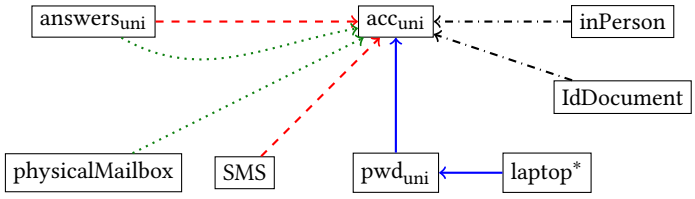

Figure 18: University layer

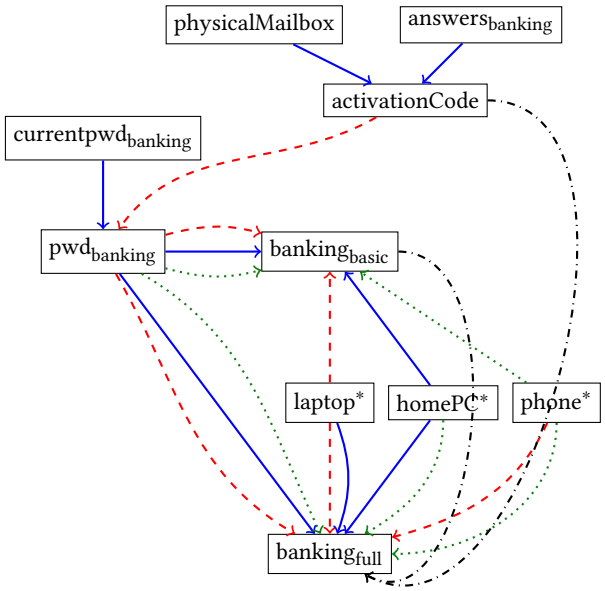

Figure 19: Online banking layer