A Logic-Based Reasoner for Discovering Authentication Vulnerabilities between Interconnected Accounts

Erisa Karafili[1], Daniele Sgandurra[2], and Emil Lupu[1]

[1] Department of Computing, Imperial College London
{e.karafili, e.c.lupu}@imperial.ac.uk
[2] Information Security Group, Royal Holloway, University of London
daniele.sgandurra@rhul.ac.uk

Abstract. With users being more reliant on online services for their daily activities, there is an increasing risk for them to be threatened by cyber-attacks harvesting their personal information or banking details. These attacks are often facilitated by the strong interconnectivity that exists between online accounts, in particular due to the presence of shared (e.g., replicated) pieces of user information across different accounts. In addition, a significant proportion of users employs pieces of information, e.g. used to recover access to an account, that are easily obtainable from their social networks accounts, and hence are vulnerable to correlation attacks, where a malicious attacker is either able to perform password reset attacks or take full control of user accounts.
This paper proposes the use of verification techniques to analyse the possible vulnerabilities that arises from shared pieces of information among interconnected online accounts. Our primary contributions include a logic-based reasoner that is able to discover vulnerable online accounts, and a corresponding tool that provides modelling of user accounts, their interconnections, and vulnerabilities. Finally, the tool allows users to perform security checks of their online accounts and suggests possible countermeasures to reduce the risk of compromise.

Keywords: Logic-Based Reasoner · Logic Analyzer · Authentication · Interconnected Accounts.

1 Introduction

With reliance on Information Technology (IT) growing, more and more services are being gradually digitised, and users are getting accustomed to use online accounts to handle almost all their digital life, from banking accounts to day-to-day communications – the so called "digital footprint" [6,10]. Typically, when registering for an online account, besides creating his/her credentials, a user is required to enter an email address for verification as well as providing answers to a set of predefined questions as a way for password recovery (also called *cognitive passwords*). Depending on the nature of the online account created, users may

also be asked to provide other pieces of information, such as a billing address, birthday, name of their pet and so on. However, different accounts end up being logically connected by virtue of having the same owner, e.g. because they share a common password, or because some pieces of information (e.g., date of birth) are shared across these accounts. This allows attackers to perform attacks such as *credential stuffing* cyber-attacks[3], where stolen account credentials (i.e., username/email address and passwords) are used to gain unauthorized access to user accounts on online services through large-scale automated login requests (e.g., using tools such as Sentry MBA [3]) on critical websites (e.g., to perform e-banking activities). The danger in this type of attacks comes from the fact that many users reuse credentials across many different websites and, therefore, the compromise of one account, typically on non-critical websites having few resources and/or motivations to protect those credentials, can have widespread repercussions to other accounts as well. Therefore, the automated analysis of security flaws within interconnected accounts is an important issue.

Another example that shows how interconnections between online accounts pose security threats to users are recovery security questions. Consider a user having an online account with a recovery security question "What is your date of birth?". However, empirical studies have found out that a significant proportion of users have security information which are easily attainable on social networks, and hence are vulnerable to reset attacks [20]. A real life example would be that of Mr. Mat Honan, who came under media scrutiny when his online accounts were hacked progressively, starting from his iCloud account. In fact, attackers obtained the last four digits of his credit card number through Amazon [18]. Then, the attackers were able to fool the Apple technical support staff into giving them a temporary password to Honan's iCloud account as they assumed his identity by providing the partial credit card number. From there, the attackers continued to gain access to Honan's Gmail and Twitter accounts using the iCloud account previously compromised. Furthermore, since iCloud Keychain keeps and remembers passwords for online and offline accounts across authorised devices, attackers were in turn able to gain access to all accounts linked to this service. This incident shows how conflicting security policies of different online services can be a disadvantage for users, and it also highlights the chain reactions among accounts that can occur after a single account is compromised.

Even if it is simple to devise measures to prevent attacks for a small number of accounts (e.g., avoiding the use of the same password across different account), however, an analysis of data from more than 20,000 users in 2015 found that the average user has 90 online accounts [4], and so even simple policies are difficult to be enforced in such large contexts. In addition, this analysis found that in the U.S. there is an average of 130 accounts assigned to a single email address. Hence, there are too many pieces of information to be remembered, and too many interconnections among accounts that are potential vulnerabilities. As such, there can be multiple ways of exploit for the attacker, and it is impractical and inefficient for a user to go through all of his/her accounts to find flaws.

[3] `https://www.owasp.org/index.php/Credential_stuffing`

Therefore, there is a need for an automatic reasoner that is able to analyse all the accounts and their interconnections looking for known vulnerabilities or potential ones.

This paper introduces a logic-based approach to identify security flaws in interconnected user accounts. To this end, we propose a reasoner to model accounts, identify threats and propose countermeasures. The proposed reasoner supports different types of online accounts and enables users to find compromised accounts as well as corresponding countermeasures to protect these accounts. We have developed a prototype tool that, given as input the user's accounts information, returns the compromised accounts and proposes countermeasures. In addition, further vulnerabilities or connections between accounts can be added to perform a 'what-if' analysis. Then, given the interconnected accounts and their related vulnerabilities, the tool shows the steps of the possible exploitation of an account. Finally, the tool focuses on the accounts where the exploits are simpler and easier to achieve, and provides a set of countermeasures to reduce the risk of account exploitation.

The main contributions of this paper are:

- We formalize the problem of vulnerabilities that derive from interconnected devices.
- We propose a reasoner to model user accounts and vulnerabilities, and the interconnections among different accounts.
- We implement a tool that, given the user accounts, finds the relations between accounts and shows the dependences between interconnected accounts that can be exploited by malicious users.

The rest of the paper is structured as follows. In Section 2 we describe related works. Then, in Section 3, we describe the proposed framework, which is used (i) to formally represent online accounts, (ii) to model existing vulnerabilities in interconnected accounts, (iii) to simulate interactions across accounts, (iv) to analyse security flaws, and (v) to generate corresponding countermeasures. In Section 4 we describe the implementation of the reasoner, while in Section 5 we report some preliminary evaluations of our framework. Finally, in Section 6 we conclude the paper.

2 Related Work

Model checking is a viable way to analyse security flaw in interconnected accounts as it supports partial verification [2]. There is no need to provide a complete specification and, hence, more focus can be put on essential properties which need to be fulfilled. Besides, when a property is invalidated, diagnostic information is provided through the trace and this is particularly useful when determining countermeasures for these vulnerabilities. There are, however, certain disadvantages to using model checking as a technique for verification. In particular, model checking suffers from the state space explosion problem and, if the number of states required is very large to be efficiently represented on the

computer memory, then the computation may take a long time [21]. MulVAL is a logic-based framework which models the interaction of software bugs with system and network configurations [17] and analyse security vulnerabilities of the network. MulVAL is implemented first by capturing the database of known vulnerabilities, then scanning the system for configuration information and also at the same time matching the known vulnerabilities to the system. These pieces of information are then encoded using Datalog, a subset of the Prolog language. MulVAL captures system interactions using a set of pre-defined rules, and analysis on the security level of the system is carried out once this preparation is done.

Attack graphs (AGs) have long been used as an effective way of assessing security threats in a network system. AGs provide a visual representations of how an exploit, or a series of exploits, can affect different hosts in a network in terms of node compromise. In the literature, there are mainly two types of attack graphs: the first one, state-based AGs, shows how an attack happens, while the second one, logical AGs, shows why an attack happens. State-based AGs [9,24] result in directed graphs, where each node represents the state of the whole network after a successful atomic attack. However, state-based AGs also suffer from state explosion issues, and empirical data has shown that graph generation procedure takes a much longer time compared to the model checking phase [23]. Moreover, these graphs contain duplicate attack paths that differ only in the order of the attack steps, which also increase the complexity of the graph, limiting the applicability of state-based representations to very small networks [8,16]. Therefore, state-based AGs do not scale well for a large number of accounts. The scalability problems of state-based representations are overcome with *logical AGs*, which are bipartite graphs that represent dependencies between exploits and security conditions [1,8]. These representations rely on the monotonicity principle: this principle states that an attacker never relinquishes privileges once obtained. Nevertheless, a suitable method to model the system needs to be chosen before generating the attack graph. The uncertainty about the attacker's behaviour makes Bayesian networks more suitable to model AG as well as to perform static and dynamic security risk assessment. For this reason, several techniques have also been proposed in the literature for performing inference on Bayesian attack graphs (BAGs). For example, forward-backward propagation is proposed in [19] to compute the unconditional probabilities. More recently, the JT algorithm was proposed in [13] for exact inference in BAGs to efficiently compute the exact unconditional probabilities by using a probabilistic message passing scheme. However, the applicability of JT to large networks is limited, especially when the AGs are dense. Therefore, the work in [14] shows how approximate inference techniques can be applied to attack graphs, so that the analysis scales linearly in the number of nodes for both static and dynamic analysis, making such analyses viable for larger networks.

3 A Logic-Based Reasoner to Model Online Accounts

In this section, we describe how the proposed reasoner models online accounts, simulates interactions across accounts, analyzes security flaws, and generates corresponding countermeasures.

3.1 Modelling of Online Accounts

Our logic-based reasoner represents the various accounts by firstly uniquely identifying them through their service provider and username, and then by associating them with other pieces of information regarding the user, e.g. password. In addition, the reasoner considers the user information associated to an account as either private or public, and allows users to select the type of authentication procedure used for every account, e.g. using a single sign-on or using the specific service provider. Another important entity modelled by our framework is the user, which is represented through the following attributes: name, gender, date of birth, mobile number, city, home-town, location, workplace, job and address (other fields can be easily added). To analyse possible vulnerabilities, every account is associated with an access policy which states who should be able to access the account. Anyone outside of this group of people/entities, who are allowed access to the account, should not be able to access the account, or it will be considered a violation of this policy. In case an entity/user, which is not part of the entities/users that are allowed to access the account, accesses the account, then this is considered a violation of the policy, and this is modelled in our reasoner as:

$$policyViolation(Account, Access) \leftarrow hasAccessed(Account, Access), \\ \textbf{not } allowAccess(Account, Access).$$

where the left side of "$\leftarrow$" represents the conclusion of the engine policy (also called *rules*), while the right side represents the preconditions that should be satisfied for the rule to be triggered and the conclusion to be satisfied. For the sake of simplicity and presentation purpose, in our rules' representations we do not use the AND logical connector ("$\wedge$"), but we substitute it simply with "," – we still use the OR logical connector and represent it with "$\vee$".

Finally, the reasoner is also used to represent generic attackers that carries out exploits by using stolen/retrieved/inferred account's information. In the reasoner, there are three main categories of exploits that an attacker can use to compromise an account:

- the attacker is able to find the credentials of the account;
- the account uses single sing-on verification and the attacker is able to exploit it;
- the attacker knows the username and is able to reset the password of the account.

3.2 Vulnerabilities of Accounts

We now describe the list of vulnerabilities that we have modelled in our reasoner. We consider vulnerabilities for individual accounts, as well as vulnerabilities that arise when linking different accounts. In the following, we describe these vulnerabilities together with their representation in our reasoner.

Vulnerabilities for Individual Accounts. The vulnerabilities for individual accounts are vulnerabilities independent from the connection between accounts, and these are:

- Publicly-available username: namely, when the chosen username can be obtained easily. This might happen in some online forums where the used nicknames or avatars replicates the username. This vulnerability is modelled as:

$$vulExists(publicUsername, X) \leftarrow account(X), username(X, U), public(U).$$

- Publicly-available email: usually service providers request an email address from the user, often used for password recovery. When a user forgets his/her password, a reset link is sent to this address to allow the user to change/reset the password. As a result, having this piece of information available means that the attacker can use some of his/her efforts in exploiting this email account as well. This vulnerability is modelled as:

$$vulExists(publicEmail, X) \leftarrow account(X), email(X, E), public(E).$$

- Commonly-used password: a list of commonly used passwords are published every year by several institutions [5]. If the user selects such a password, the attacker does not even need to carry out a dictionary attack to obtain the password of the particular account. This vulnerability is modelled as:

$$\begin{aligned} vulExists(commonPW, X) \leftarrow\ & account(X), pwAccount(X, pw), \\ & commonPW(pw). \end{aligned}$$

- Password contains name of user: it is not uncommon to find users using just their name as passwords. This vulnerability is modelled as:

$$\begin{aligned} vulExists(sameName\&PW, X) \leftarrow\ & account(X), nameofUser(X, N), \\ & pwAccount(X, pw), \\ & (N = pw \vee similar(N, pw)). \end{aligned}$$

  We use the $similar(N, Pw)$ predicate as the password can be a simple variation of the name, thus, they are not exactly the same but similar[4].
- Password contains username of user: even if some website disallows the use of the username as password, not every website enforces this rule. This vulnerability is modelled as:

$$\begin{aligned} vulExists(sameUsername\&PW, X) \leftarrow\ & account(X), username(X, U), \\ & pwAccount(X, pw), \\ & (U = pw \vee similar(U, pw)). \end{aligned}$$

[4] The predicate $similar(A, B)$ states that A and B are very similar to each other and by knowing A the attacker can infer easily B, and vice versa.

– Password is weak: while several websites provide a password robustness check, users are able to find (sometime ingenious) alternative ways to bypass these checks and provide a weak password. This vulnerability is modelled as:

$$vulExists(weakPW, X) \leftarrow account(X), pwAccount(X, pw), weakPW(pw).$$

– Password unchanged for too long: a password which has not been changed for long, if weak, can be easily found due to dictionary attacks. This vulnerability is modelled as:

$$vulExists(oldPW, X) \leftarrow account(X), pwAccount(X, pw), oldPW(pw).$$

Vulnerabilities due to Account Connections. In the following we describe those vulnerabilities that involve linkage across different accounts that are owned by the same user/entity, which are:

– Repeated passwords: many users use the same passwords across different accounts. This means that if an attacker knows (or is able to retrieve) the password to an account, he/she also can access all the accounts which reuses this particular password. This vulnerability is modelled as:

$$\begin{aligned} vulExists(repeatPW, X, Y) \leftarrow\ & account(X), account(Y), X \neq Y, \\ & pwAccount(X, pw_1), pwAccount(Y, pw_2), \\ & pw_1 = pw_2. \end{aligned}$$

– Repeated usernames: suppose an attacker knows the username of a specific account, the attacker can attempt to log in a different online account with the same username to check if the account exists. This vulnerability is modelled as:

$$\begin{aligned} vulExists(repeatUsername, X, Y) \leftarrow\ & account(X), account(Y), X \neq Y, \\ & username(X, U_1), username(Y, U_2), \\ & U_1 = U_2. \end{aligned}$$

– Information required for password reset available (publicly or through the use of another account). There are mainly two methods used for password reset: the first method entails sending a recovery link to the associated email address, while the second one requires the user to answer a set of security questions. In this last case, the information required for answering security questions may be available publicly or in another account that the attacker already has access to. This means that the attacker can use this piece of information to compromise the account of the user by resetting the passwords. This vulnerability is modelled as:

$$\begin{aligned} vulExists(publicRecoveryInfo, X) \leftarrow\ & account(X), recovery(X, Info), \\ & public(Info). \end{aligned}$$

$$\begin{aligned} vulExists(recoveryInfoInAcc, X, Y) \leftarrow\ & account(X), recovery(X, Info), \\ & account(Y), X \neq Y, \\ & inAccount(Y, Info). \end{aligned}$$

In Figure 1 we show a graphical representation of the dependencies between accounts and possible vulnerabilities that might allow attackers to get access to a particular account, which we have modelled in our reasoner.

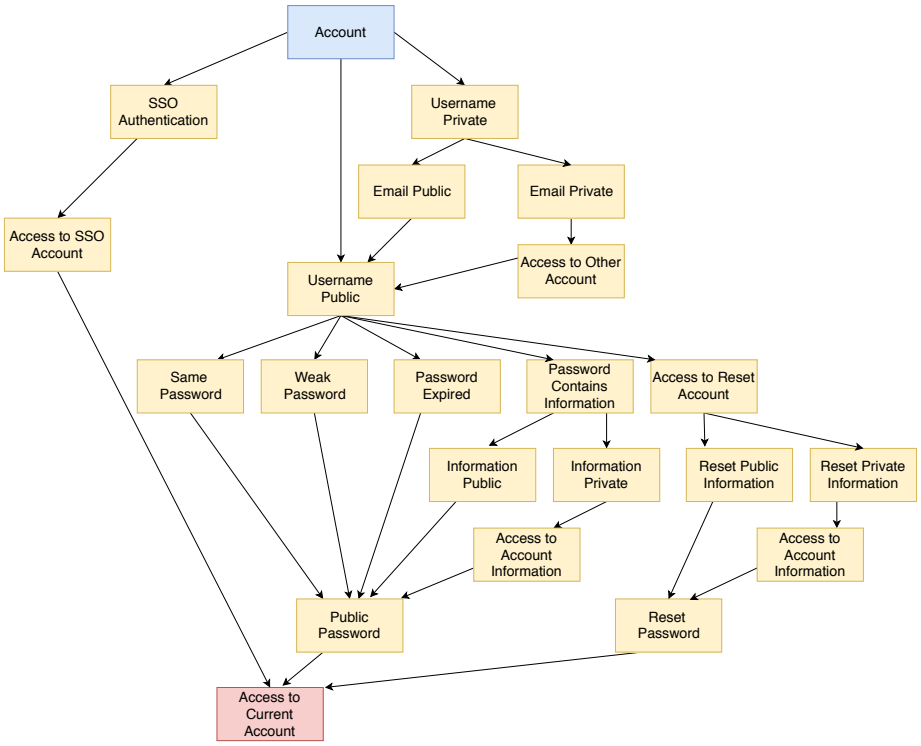

Fig. 1. Attack Tree of an Account based on the Attackers' Knowledge

3.3 Metrics for Account Compromise Analysis

Let us now give a brief overview of the metrics we have introduced in our reasoner to provide a measure of the extent of compromised accounts. The first two metrics are non-path analysis metrics while the remaining ones are metrics related to the paths that attackers can follow to access the account.

– An interesting metric to use is the *network compromise percentage* (NCP) [12], which measures the proportion of the network that was compromised/attacked successfully by the attacker. In our reasoner, this metrics represents the proportion of accounts that are compromised by the attacker. The reasoner allocates a higher weight to accounts that hold sensitive information, e.g. bank or card details, or to accounts that have information about other accounts.

- To analyse the generated graphs the reasoner uses the *weakest adversary* metric, which measures the security of the weakest component in the network [7]. Using this metric, the reasoner assumes that the set of accounts are as secure as their weakest component.
- The first path metric the reasoner checks is the *shortest path* [15], which represents the smallest number of steps leading to the attack goal. Knowing the shortest path helps the reasoner in developing a countermeasure, as we expect the attackers to choose the most efficient way to reach their target. Please note that the shortest path is not necessarily the easiest path. Therefore, this metric does not always provide a consistent evaluation of the attacks.
- Another metrics in the reasoner is the *number of paths*, which represents the number of different attacks/exploits that can be carried out to compromise the same account [15]. This metrics can also be used to define the account with the highest risk of being compromised, as the higher the number of exploits, the higher the number of vulnerabilities that can be used by the attacker. Having a higher number of vulnerabilities does not necessarily mean that the vulnerabilities have higher chances on being exploited as no information is provided on their difficulties on being exploited.
- The *mean path length* [11] is the arithmetic average of all the paths lengths that, starting from an initial state, bring to the goal account. This metric is relevant in determining the average level of effort required to compromise an account [9]. Its drawback is that the value might be the same even when there is an increase in the number of paths.
- To evaluate which are the outliers when it comes to different exploits, the reasoner uses the *standard deviation of path lengths*. This metrics is useful when generating countermeasures, as we want to focus on specific attack paths that are more likely to be exploited.
- The reasoner also uses the *mode* and *median* of path lengths as they are less likely to be influenced from outliers values. The *normalised mean path length* gives a better security estimate as it is based on both the number of paths as well as the path lengths, hence addressing the key criticism of the mean path length metric. The reasoner aggregates the functions of both the mean path length and the total number of paths.

The different metrics pertain to different types of conclusions a user can draw from the generated graph. In more detail, the number of paths, shortest path and mean path length metrics quantify the overall security level of the network or an individual account. Therefore, these metrics can be used as a basis of comparison before and after a countermeasure is implemented to find out the effectiveness of such a countermeasure. Instead, the mean path for an individual account and the standard deviation can be employed together to find which account requires the greatest attention. For instance, suppose the user wishes only to secure the most vulnerable accounts, the reasoner can present countermeasures only pertaining to accounts with mean path length below the network mean or below a certain percentage.

In the following we first describe the threat model used in the reasoner, and how the reasoner selects the countermeasures.

3.4 Threat Model and Countermeasure Analysis

In the threat model, we make the assumption that every attacker has all the possible pieces of information that it can obtain, which is not necessarily true as sometimes the attacker cannot find connections between accounts or cannot identify the public information needed for compromising an account. Similarly, we assume that an attacker knows a piece of information in case it is public, or if the information is stored in an account where the attacker has access to. In addition, we assume that an attacker can compromise an account if s/he has the login credentials or can reset the password or if s/he knows the login credentials of the single-sign-on account associated with it. Finally, we assume the attacker may retrieve the login credentials if there are existing vulnerabilities in the account, e.g. weak password or connections with other accounts. These strong assumptions make the vulnerability analysis more robust, as any countermeasure to these threats is able to cover also other threats with less resources.

At the end of the vulnerability analysis, the reasoner returns a set of countermeasures aimed at reducing the overall risk for the user' accounts. The proposed countermeasures are based on the *single action removal* principle, in which the reasoner evaluates the impact of removing a single step/move from the attacker strategy. Please note that some vulnerabilities are not as easy to remove as others. For example, let us suppose a set of accounts shares the same common password and username. Instead of changing all the passwords, it is more feasible to ensure that the username will never be leaked. Hence, the reasoner provides a *minimum critical set of countermeasures* ensuring that the vulnerabilities are eliminated using as less resources as possible. The minimum critical set is developed using a greedy algorithm. The reasoner provides different set of attack paths, each representing an "or" relation. In other words, for every possible exploit, the reasoner represents each step of the exploit as an element in a set, and the set represents a single exploit. There might be different sets representing the different independent ways an attacker can compromise an account, and the reasoner iteratively selects an element (attack step) that is present in the largest number of sets, until all of the sets are covered [9].

4 Implementation

We have developed a prototype tool to implement the proposed reasoner system. The reasoner is built upon XSB[5] and it includes all the definitions for modelling user details, account details, vulnerabilities, access policies, exploit rules, hypothetical 'what-if' vulnerability analysis as well as to generate countermeasures. We have defined a set of pre-defined Prolog rules, which are built into the reasoner to capture connections across the different accounts, to identify security

[5] `http://xsb.sourceforge.net/`

vulnerabilities, and to define and verify the threat model [22]. Similarly, we have created a set of pre-built queries that can be launched by users to perform attack simulations (e.g., "find out who can access the different accounts"). After performing these queries, the reasoner returns to the user a set of results, namely lists of vulnerable accounts and the steps to perform such attacks. The tool also provides metrics to give a quantitative view of the security status of the list of online accounts. This enables the optimisation of countermeasures and security improvements. For analysing the generated graphs, the proposed system implements the metrics introduced in Section 3.3. As a future development, we envision this tool as a standalone application running on the user computer, with a plugin integrated with the browser to interact with the user information (similar to a password manager), as to avoid the user entering the pieces of information manually in the engine. Of course, in this implementation, all the pieces of information need to be securely stored on the user computer and only accessed by the user (similar to a password manager).

4.1 Countermeasures Generation

As recalled previously, the system, after analysing the given accounts together with their vulnerabilities, proposes a set of countermeasures. Once the user has selected some of these countermeasures, the system can be re-run to perform a further analysis to check if further countermeasures should be proposed. For each vulnerability listed in Section 3.2, we have defined a set of applicable countermeasures[6], in particular:

- Password related vulnerabilities. When the user is using a bad/weak/common password, or contains publicly available information, the reasoner suggests the user to change the password with a stronger one.

$$countermeas(change, pw) \leftarrow account(X), pwAccount(X, pw), \\ (weakPW(pw) \vee badPW(pw) \vee \\ commonPW(pw) \vee \ pubInfoPW(pw)).$$

- Repeated passwords. When the user is using the same password for more than one account, especially when this account is connected or shares the same username, the reasoner proposes the user to change the password to a different one.

$$countermeas(change, pw_1) \leftarrow account(X), account(Y), X \neq Y, \\ pwAccount(X, pw_1), pwAccount(Y, pw_2), \\ (pw_1 = pw_2 \vee similar(pw_1, pw_2)), \\ ((username(X, U_1), username(X, U_2), \\ U_1 = U_2) \vee \ connected(X, Y)).$$

[6] We give together with the countermeasures their corresponding rules as represented by the reasoner.

– Password reset information available publicly or in another account. When a piece of information for resetting a password is publicly available or is available using another account, the reasoner proposes two countermeasures: (i) to change the recovery information, or (ii) to restrict/remove the access to the information needed for resetting the password.

$$countermeas(change, Info) \leftarrow account(X), recovery(X, Info), \\ public(Info) \vee (account(Y), \\ X \neq Y, inAccount(Y, Info)).$$

$$countermeas(makePrivate, Info) \leftarrow account(X), recovery(X, Info), \\ public(Info).$$

5 Preliminary Evaluation of the Framework

We tested our framework by designing a realistic use case in which we have considered all the possible vulnerabilities described in Section 3 that an attacker can exploit to access an account. The reasoner was then used to check for possible violations of the policies, and the proposed countermeasures were put in place and a further analysis was performed to verify the effectiveness of the countermeasures in eliminating the vulnerabilities exploited by the attacker.

In detail, we created a set of 35 accounts using the most common online accounts, e.g. Facebook, Twitter, Ebay and Amazon. For each of these websites and social networks, we have modelled their different password strategies realistically, as well as their authentication mechanisms. In addition, for each of the 35 accounts, we have provided user information, e.g. username, email, phone number, and set them as either public or private ones. We have also considered the reset password procedure for each account and the required information: in particular, we have associated those accounts with critical information, such as credit cards details, with stronger passwords than those accounts used in social settings, e.g. public forums. To make the use case more realistic, we have manually inserted some vulnerabilities into the accounts. For example, we have inserted few repetitions of the passwords across accounts as well as minor variations of the same password. Finally, we have also included several relations between different accounts, which mainly arise due to shared login credentials, or recovery email address, or public information used for the recovery.

On this use case, we then have run a what-if vulnerability analysis, where we have used a range of possible types of information available to the attacker. In particular, we have used the following different scenarios: (i) public information, e.g. assuming a username becomes public; (ii) information leaks, when some private pieces of information can be found on leaked database, e.g. of passwords; (iii) website vulnerabilities, e.g. the attacker has access to the account by exploiting a vulnerability of the website. By introducing these vulnerabilities, we then have modelled and evaluated how the amount of information available to an attacker corresponds to changes in the number of exploits and in the number of compromised accounts. As expected, we have seen that rendering some basic

pieces of information as public, such as the username or the type of account, is not as critical as disclosing the password of the account in terms of increase in the number of exploitation of accounts. In addition, "information leaks" is the what-if vulnerability scenario with the worse impact for user accounts, as it is the one that brings higher chances for an attacker of successful exploitation. Similarly, when it comes to introducing website vulnerabilities, the impact is very large when introduced in highly interconnected accounts and popular websites. Finally, the evaluation showed that the selection of the countermeasures is targeted on highly interconnected account and on those with a weak password.

6 Conclusion

Securing a large set of interconnected online accounts requires a huge effort from both the users and the service providers. The problem becomes harder when the security policy of one provider can influence or contrast the security policy of another provider. In this paper, we introduced a framework that can be used by users to assess the security and vulnerabilities of their interconnected online accounts. The proposed tool gives a representation of the dependencies between the accounts and proposes countermeasures to ensure their security. As a direction of future research we plan to fully automate the system, e.g. by implementing a browser plugin that securely provides the account information to the reasoner, as well as to increase the number of families of vulnerabilities that can be checked by the reasoner.

Acknowledgments

Erisa Karafili was supported by the European Union's H2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 746667. This work builds upon research funded by the Engineering and Physical Sciences Research Council (EPSRC) through grants EP/L022729/1 and EP/N023242/1.

References

1. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, Graph-Based Network Vulnerability Analysis. In: Procs. Conf. on Computer and Communications Security. pp. 217–224 (2002)
2. Baier, C., Katoen, J.P.: Principles of Model Checking (Representation and Mind Series). The MIT Press (2008)
3. Ben-Meir, E.: Sentry MBA: A Tale of the Most Popular Credential Stuffing Attack Tool. `https://blog.cyberint.com/sentry-mba-a-tale-of-the-most-popular-credential-stuffing-attack-tool` (2017)
4. Bras, T.L.: Online overload its worse than you thought. `https://blog.dashlane.com/infographic-online-overload-its-worse-than-you-thought/` (July 2015)
5. Data, S.: 100 worst passwords of 2017. `https://s13639.pcdn.co/wp-content/uploads/2017/12/Top-100-Worst-Passwords-of-2017a.pdf` (2017)

6. Gosling, S.D., Gaddis, S., Vazire, S.: Personality impressions based on facebook profiles. In: ICWSM (2007)
7. Idika, N., Bhargava, B.: Extending attack graph-based security metrics and aggregating their application. IEEE Transactions on Dependable and Secure Computing **9**(1), 75–85 (2012)
8. Jajodia, S., Noel, S., O'Berry, B.: Topological analysis of network attack vulnerability, pp. 247–266. Springer US, Boston, MA (2005)
9. Jha, S., Sheyner, O., Wing, J.: Two Formal Analyses of Attack Graphs. In: Procs. of the Workshop on Computer Security Foundations. pp. 49–63 (2002)
10. Kosinski, M., Stillwell, D., Graepel, T.: Private traits and attributes are predictable from digital records of human behavior. Proceedings of the National Academy of Sciences (2013). https://doi.org/10.1073/pnas.1218772110, `http://www.pnas.org/content/early/2013/03/06/1218772110`
11. Li, W., Vaughn, R.B.: Cluster security research involving the modeling of network exploitations using exploitation graphs. In: Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on. vol. 2, pp. 26–26 (May 2006)
12. Lippmann, R., Ingols, K., Scott, C., Piwowarski, K., Kratkiewicz, K., Artz, M., Cunningham, R.: Validating and restoring defense in depth using attack graphs. In: Proceedings of the 2006 IEEE Conference on Military Communications. pp. 981–990. MILCOM'06, IEEE Press, Piscataway, NJ, USA (2006)
13. Muñoz-González, L., Sgandurra, D., Barrere, M., Lupu, E.C.: Exact inference techniques for the analysis of bayesian attack graphs. IEEE Transactions on Dependable and Secure Computing **PP**(99), 1–1 (2017). https://doi.org/10.1109/TDSC.2016.2627033
14. Muñoz-González, L., Sgandurra, D., Paudice, A., Lupu, E.C.: Efficient attack graph analysis through approximate inference. ACM Trans. Priv. Secur. **20**(3), 10:1–10:30 (Jul 2017). https://doi.org/10.1145/3105760, `http://doi.acm.org/10.1145/3105760`
15. Ortalo, R., Deswarte, Y., Kaâniche, M.: Experimenting with quantitative evaluation tools for monitoring operational security. IEEE Transactions on Software Engineering **25**(5), 633–650 (1999)
16. Ou, X., Boyer, W., McQueen, M.: A scalable approach to attack graph generation. In: Procs. Conf. on Computer and Communications Security. pp. 336–345 (2006)
17. Ou, X., Govindavajhala, S., Appel, A.W.: Mulval: A logic-based network security analyzer. In: Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14. pp. 8–8. SSYM'05, USENIX Association, Berkeley, CA, USA (2005)
18. Pepitone, J.: Hack attack exposes major gap in amazon and apple security. `http://money.cnn.com/2012/08/07/technology/mat-honan-hacked/` (August 2012)
19. Poolsappasit, N., Dewri, R., Ray, I.: Dynamic Security Risk Management using Bayesian Attack Graphs. IEEE Trans. on Dependable and Secure Computing **9**(1), 61–74 (2012)
20. Rabkin, A.: Personal knowledge questions for fallback authentication: Security questions in the era of facebook. In: Proceedings of the 4th Symposium on Usable Privacy and Security. pp. 13–23. SOUPS '08, ACM, New York, NY, USA (2008)
21. Ritchey, R.W., Ammann, P.: Using model checking to analyze network vulnerabilities. In: Proceeding 2000 IEEE Symposium on Security and Privacy. S P 2000. pp. 156–165 (2000)

22. Sgandurra, D., Karafili, E., Lupu, E.: Formalizing threat models for virtualized systems. In: Ranise, S., Swarup, V. (eds.) Data and Applications Security and Privacy XXX. pp. 251–267. Springer International Publishing, Cham (2016)
23. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.: Automated Generation and Analysis of Attack Graphs. In: Procs. of the IEEE Symp. on Security and Privacy. pp. 273–284 (2002)
24. Sheyner, O., Wing, J.: Tools for Generating and Analyzing Attack Graphs. In: Formal Methods for Components and Objects. pp. 344–371 (2004)