Software Engineering and Applications 软件工程与应用, 2019, 8(5), 207-214
Published Online October 2019 in Hans. http://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2019.85025

Hans汉斯

A Honeyword Generation Method Based on Special Character Distance

Weiwei Jing[1], Jinku Cui[2], Youwen Zhu[2]

[1]Nanjing Research Institute of Electronics Technology, Nanjing Jiangsu
[2]Nanjing University of Aeronautics and Astronautics, Nanjing Jiangsu
Email: wwjingx@163.com

Received: Sep. 18th, 2019; accepted: Oct. 3rd, 2019; published: Oct. 10th, 2019

Abstract

Honeyword can be used to improve password storage security, and timely detect password data set disclosure. Nevertheless, existing honeyword generation schemes are of low security, and require large storage overhead. In this paper, we propose a virtual honeyword generation method based on special character distance. Additionally, our analysis shows that the proposed scheme can dramatically reduce the storage cost, and improve the security and detection probability of password.

Keywords

Identity Authentication, Password, Disclosure Detection, Storage Security

基于特殊字符间距的Honeyword生成机制

荆巍巍[1]，崔进库[2]，朱友文[2]

[1]南京电子技术研究所，江苏 南京
[2]南京航空航天大学，江苏 南京
Email: wwjingx@163.com

收稿日期：2019年9月18日；录用日期：2019年10月3日；发布日期：2019年10月10日

摘 要

Honeyword可以用于提升口令存储的安全性，并能够及时检测口令数据集的泄漏。然而，当前的honeyword生成机制依然存在安全性较弱、占用存储空间过大等问题。为此，本文提出了基于特殊字符间距的虚拟honeyword生成机制。分析结果显示该方案可以显著地减少存储空间开销，并提升安全性和

文章引用: 荆巍巍, 崔进库, 朱友文. 基于特殊字符间距的 Honeyword 生成机制[J]. 软件工程与应用, 2019, 8(5): 207-214. DOI: 10.12677/sea.2019.85025

口令集泄漏被检测的概率。

关键词

身份认证，口令，泄漏检测，存储安全

1. 引言

口令文件的泄露是非常严重的安全问题，其影响已经波及到百万用户和相关的公司，比如 Yahoo [1]，Dropbox [2]，Weebly [3]，Myspace [4]，LinkedIn [5]，eBay [6]，eHarmony，RockYou，Adobe [7]，以及国内的网易邮箱泄露事件、12306 用户帐户泄露等。今天，那些赫赫有名的网站被攻击、百万级别的用户口令流出已经不是什么新闻了。正是这些泄露的口令让用户成为了潜在的网络攻击目标。这些口令泄露事件也说明了一个严重的问题，很多网站依然在采用安全性较弱的方法来存储用户的口令。例如，LinkedIn 网站采用不加盐混淆的 SHA-1 算法对口令进行哈希后存储，类似的 eHarmary 网站采用的是未加盐混淆的 MD5 算法对口令进行哈希后存储[8]。实际上，一旦口令文件被盗，借助诸如 Weir 等人提出的口令破解算法[9]，很容易得到大多数口令的明文。尽管 MD5 在众多哈希算法中是安全性较高的，但是随着计算机计算能力的提高，以及王等提出的“杂凑碰撞算法”[10]，使得 MD5 的碰撞攻击成功概率大大提高，能够在较短时间内针对某个 MD5 哈希值进行碰撞。

更严重的是，探测到这些泄露事件常常是在口令刚被盗取后的几个月，有的甚至是数年。在此期间，攻击者已经挖掘、利用完数据中的信息、把口令文件发布在网上之后，或者是转手卖给了其他人。例如，最近在 2017 年 10 月报道的灾难性口令泄露事件，涉及到 Yahoo 所有 30 亿用户，然而泄密实际上是在 4 年前发生的[1]；2016 年 10 月揭露的 Weebly 泄露的口令涉及到 4300 万用户，要求用户修改他们的口令，然而泄密发生在 8 个月之前[3]；Dropbox 的 6800 万泄密发生在 2012 年，直到 4 年之后的 2016 年 5 月份[2]，用户才被要求修改他们的口令；Myspace 的 3.6 亿用户数据泄露 8 年后才被得知：数据库在 2008 年被窃取[4]，直到 2016 年 5 月有人在网上出售相关文件才知道。2016 年由 Version 公司做出的数据泄露报告指出，在调查的 2260 起泄露事件中，超过 85%的最先由第三方的组织发现，91%的事件几个星期之后才发现，70%的事件要在几个月甚至几年之后才被探测到[11]。因此，设计一种主动的、及时的口令泄露探测方案具有重要的应用价值。

鉴于上述问题，本文提出了一种基于伪造口令或者帐户的方式来实现一种简单可行、经济有效的口令安全存储和泄露预警方案。蜜罐(Honeypot)技术是众多确认口令数据库泄露中的一种。在蜜罐方案中，系统管理员蓄意的创建一些虚假帐户来诱惑攻击者，如果任一虚假帐户被使用了，就可以探测到口令的泄露[12]。Herley 和 Florencio [13]改造了这种技术，用来保护银行账户，免受暴力猜测的攻击。根据 Herley 和 Florencio 的研究，用每个用户的帐户和某些口令组合，进行错误登录的尝试会指向蜜罐帐户，也就是说，能够识别恶意行为。例如，长度为 8 的纯数字口令有10^8种可能，假设系统将 10,000 个错误口令与蜜罐帐户关联，那么攻击者进行 10,000 次暴力猜解就很可能命中一个蜜罐帐户。Bojinov 等人基于诱饵构建了口令防盗的模型[14]，被称为 Kamouflage。在此模型中，虚假口令和用户真正的口令放在同一个集

合中，因此攻击者必须进行大量的在线尝试才能得到正确的帐户信息。近来，Juels 和 Rivest 提出了一种可改善口令泄露问题的方法，引入诱饵口令(被称为“honeyword”)，能够探测到尝试使用破解的口令进行登录的攻击行为[15]。基本的思想是，对于每个用户，构建一个与之关联的“sweetword”集合，该集合中只有一个元素是正确的口令，其他的都是“honeywords”(decoypasswords)。因此，当攻击者尝试用 honeyword 进入系统时，就会触发警报、通知系统管理员口令文件已经泄露。此后，一些改进的 honeyword 生成机制被陆续提出。然而，目前基于 honeyword 的认证方案都存在或多或少的缺陷。最明显的一点是为了存储 honeyword，对存储空间有较大的需求。除了存储开销，也有很多与此技术相关的安全性和适用性问题。现有的方法只能解决这些问题当中的一部分，无法兼顾所有。我们提出的基于 honeyword 的认证技术正是为了解决这些问题。

概括地说，本文提出了一种新的技术——特殊字符间距机制(Special Character Distance Mechanism, SCDM)——来生成 honeyword。SCDM 不需要对系统的用户接口进行更改，也不需要用户记忆额外的字符。SCDM 的部署对用户来说是透明的，因此保证了系统的用户友好性和适用性。SCDM 运用少量特殊字符生成 honeyword，能够规避口令与用户名的关联问题(例如：bond007)、口令与用户名的关联问题(例如：用户名为 James，口令为 bond007)。采用我们提出的方案，系统只需要额外存储少量的信息，就可以生成虚拟的 honeyword 列表，因此极大程度上降低了存储开销。

论文其余部分安排如下。第 2 部分介绍了 honeyword 相关的基础知识和背景信息；第 3 部分详细地展示了我们所提方案的步骤；第 4 部分对本文所提方案进行了深入分析；第 5 部分总结了本文。

2. 背景知识

这里，我们对基于 honeyword 的认证技术进行简短的介绍，以及一些与此技术相关的问题，这有助于理解我们的方案。

2.1. 基于 Honeyword 的认证技术

在表 1 中，我们给出了一些本文使用的相关名词及其注解。

从本质上来说，honeyword 技术背后的思想是：生成虚假口令——被称为 honeyword——使这些口令与每个帐户都关联起来。当攻击者得到口令列表后，即便她将口令列表中的大部分或者全部哈希值逆转为明文，她依然无法确定列表当中的哪个口令才是用户真正的口令。因此，当攻击者贸然用 honeyword 尝试访问系统时，系统管理员就可以探测到这种恶意行为。

Table 1. Annotation
表 1. 注释表

符号	含义
$H(\ )$	用来计算口令哈希值的密码学哈希函数
u_i	第 i 个用户的用户名
p_i	第 i 个用户的口令
W_i	第 i 个用户的口令列表
k	W_i 中元素的个数
c_i	第 i 个用户的真正口令在列表 W_i 中的索引
$Gen(k)$	用来生成含有 k 个元素的列表 W_i 的方法
sweetword	W_i 中的每个元素
sugarword	W_i 中的正确口令
honeyword	W_i 中的每个虚假口令

Honeyword 的工作机制简述如下：对于每个用户 u_i，系统调用 honeyword 生成算法 $Gen(k)$ 生成一个 sweetword 列表，W_i。方法 $Gen(k)$ 输入参数为 k，即想要生成的 sweetword 的个数；输出结果为 sweetword 列表 $W_i=(w_{i,1},w_{i,2},\cdots,w_{i,k})$ 和 c_i，其中 c_i 为第 i 个用户的真正口令(sugarword)在列表 W_i 中的索引。其中，用户名 u_i 和 sweetwords 的哈希值组成一个元组 $\langle u_i,(v_{i,1},v_{i,2},\cdots,v_{i,k})\rangle$ 存在主服务器上，而 c_i 存在另一个被称为“honeychecker”服务器上。通过将系统中的私密信息多元化——将口令的哈希值存在一个服务器上，将 c_i 存在 honeychecker 上，攻击者就难以将整个系统作为一个整体攻破。换句话说，就是提供了基本形式上的分布式安全。需要强调的一点时，传统的口令认证方案中，每个帐户存储的是用户名 u_i 和口令 p_i 的哈希值 $H(p_i)$ 的键值对，即 $\langle u_i,H(p_i)\rangle$，而基于 honeyword 的认证方案中，数据库中存储的是用户名 u_i 和 sweetwords 的哈希值列表 V_i 构成的元组，即 $\langle u_i,V_i\rangle$，其中 $V_i=(v_{i,1},v_{i,2},\cdots,v_{i,k})$。honeyword 方案中用户的登录过程如下：

1) 用户 u_i 输入口令 g 登陆系统。

2) 服务器首先检查 $H(g)$ 是否在列表 V_i 中，如果不在，则拒绝访问。

3) 如果 $H(g)$ 在列表 V_i 中，系统则进一步检查 g 到底是 honeyword 还是 sugarword。

4) 假设 $v_{i,j}=H(g)$，即 $H(g)$ 在列表 V_i 中，且是 V_i 中第 j 个元素。j 的值通过确保安全的信道传送给 honeychecker。

5) honeychecker 检验 j 是否和 c_i 相等。如果相等，则返回 TRUE；如果不想等，则返回 FALSE 并根据系统既定的安全策略触发警报。

在进一步讨论 honeyword 生成方法之前，我们需要先对 honeyword 的生成算法 $Gen(\ )$ 进行说明。需要注意的是，honeyword 的优点、有效性和 $Gen(\ )$ 是如何构造的息息相关。因此，honeyword 的作者引入了一个新的定义——$Gen(\ )$ 的 flatness——来度量攻击者从 sweetwords 中选中正确口令的可能性大小。换句话说，如果 honeyword 生成方法是 $\epsilon-flat$，那么攻击者就有至少 $1-\epsilon$ 的可能选中 honeyword。例如，若 $\epsilon=1/4$，那么攻击者从 W_i 中选中正确口令 p_i 的可能性至多为 25%。简而言之，如果算法不够平滑(flat)，那么真正的口令与剩下的虚假口令相比就很显眼，攻击者就可以轻易的发现哪个是用户的原始口令。

2.2. 引入 Honeyword 的主要问题

基于 honeyword 的认证技术可以认为是对基于口令的认证技术的一种扩展。因此，人们对于基于 honeyword 的认证技术的接受程度主要取决于它如何在安全性和适用性上取得平衡。基于 honeyword 的认证技术需要额外的大量空间用于维护 honeywords，所以空间开销也对其接受程度产生了影响。下面，我们讨论了主要问题当中和存储空间、安全以及适用性相关的某些问题。

1) 存储开销：现有的 honeyword 生成方法中，大多数需要在系统中为每个用户的帐户存储 $k-1(k\geq 2)$ 个 honeyword。因此，一个拥有 N 个用户的系统需要额外存储 $N\times(k-1)$ 大小的信息。假设 h 是口令哈希值的字节数目。如果系统采用流行的哈希技术，譬如：SHA-1，那么 h 的值就是 20 字节。考虑到 Juels 和 Rivest 在[15]推荐的 k 大小为 20，那么每个用户的存储开销就要增加 380 个字节。近来的研究工作也都证实了存储开销是基于 honeyword 的认证技术的一大缺点[15] [16]。

2) 相互关联风险：在用户名和口令之间可能存在着关联。例如，如果用户选择“姚明”作为用户名，用“basketball”作为口令，那么着两者之间就存在着关联。在此种情境下，honeyword 起不到掩盖原始口令的作用。

3) 易辨识的知名口令模式：如果用户选择的口令是和一些众所周知的事物、事实相关的，也非常容易从 sweetword 列表中被辨认出来。属于此类的口令像 bond007，james007，007bond，007007 等。这些例子是从 10,000 个最常用的口令[17]中挑选出来的。

4) 拒绝服务式攻击抗性相关问题：在没有得到口令文件 F_p 的前提下，攻击者如果能够猜中任何honeyword，那么这种基于 honeyword 的认证方案就是一种拒绝服务式攻击抗性较弱的方案。为了猜测用户 u_i 的 honeyword，攻击者必须要事先知道用户 u_i 的真正口令。在已知正确口令信息的前提下，攻击者若能成功猜中 honeyword，那么攻击者就可以蓄意使用 honeyword 进行登录，让系统错误的以为口令文件 F_p 已经被盗取了，而事实并非如此。一旦系统感应到某个用户用 honeyword 尝试访问系统了，它可能会按照既定的预警策略对每个用户进行封锁，这样十分影响用户体验。为了发起拒绝服务式攻击，攻击者可能会先进行观察攻击[18]或者自己去注册一个帐户，从而得到原始的口令信息。

5) 多系统薄弱性相关问题：大部分人在不同的系统帐户中使用相同的口令，近来的关于口令泄漏事件的报道几乎都支持这一事实[19]。用户 u_i 提交了口令 p_i 用于注册之后，系统根据原始口令 p_i 生成列表 W_i，其中包含 honeywords 和用户的真正口令 p_i。即使用户在两个系统提交的口令 p_i 相同，但是任何 honeyword 生成算法的基本特性之一就是每次运行生成的 honeywords 是不能完全相同的。因此，即使使用相同的口令 p_i，不同系统上的 W_i 列表也是不同的。假设攻击者成功的获取了这两个系统的口令文件 F_p 对同一用户的两个 W_i 列表进行集合的交集运算，得到该用户真正口令的概率就非常大。这就是所说的基于 honeyword 的认证技术中的多系统薄弱性问题。

6) 更改系统用户界面：现有的基于 honeyword 的认证方案中，为了取得更好的 flatness，需要对系统原本的用户界面进行更改。比如：加尾巴、密码圆盘、基于问卷调查的方案等。若是现有的系统想要部署这些方案，就必须对系统界面进行更改，可能需要用户额外花费一些时间才能弄明白如何注册系统，以及应该记住哪些改动，以便下次能够成功登录。所以这些方案大大降低了方案的适用性，没能较好的平衡适用性和安全性。

7) 系统对用户的干扰和用户的记忆压力：这是两个和适用性相关的互补标准。在 Juels 和 Rivest 的工作中[15]，作者阐明为了满足 honeyword 方案的安全性要求，某些基于 honeyword 的方案会干扰用户对于口令的选择(例如，让用户选择距离满足距离测量函数的尾巴)，强制用户记忆系统产生的信息(额外记忆尾巴，记忆问卷的答案等)。这些额外的信息是被系统用来生成 honeywords 的。因此，一个基于 honeyword 的认证方案，一旦有系统扰动，就意味着要对用户增加额外的记忆压力。总的来说，此类系统已经明示了它们采用了 honeyword 技术，而且需要和用户进行额外的交互、影响用户对口令的选择。

3. 基于字符间距的 Honeyword 生成机制

我们提出的方案名为“特殊字符间距机制(Special Character Distance Mechanism, SCDM)”。该方案无需对用户界面做出任何更改，唯一的要求就是用户的口令当中需要含有两个不同的特殊的字符。所以，我们的方案既不会干涉用户对口令的自主选择，也不会对用户施加额外的记忆压力——我们的方案具有极好的适用性。此外，SCDM 中与 honeyword 相关的仅有特殊字符，而在绝大多数口令中，特殊字符与口令的其他部分、与用户名不存在语义上的关联，所以 SCDM 能够规避口令－用户名关联问题。SCDM 理应归属到传统 UI 类 honeyword 生成方法中，所以也具有传统 UI 类共同的有点，即由于不需要更改系统的用户接口，攻击者不会轻易的从系统界面上判断出是否采用了基于 honeyword 的认证方案。

3.1. 生成特殊字符链

随着 Unicode 的引入和广泛使用，系统能够接受和处理的特殊字符越来越多，我们的方案并不局限于某种特定的字符集合。不过为了分析之便，我们仅在美国信息交换标准代码(ASCII [20])上讨论我们的方案。在 ASCII 表中，可用于口令的可打印字符有 95 个，其中个特殊字符有 33 个，我们将这 33 个特殊字符以随机顺序排放在一个环形的链表中。为了叙述方便，将其称为“特殊字符链(Special Character Chain,

SCC)”。特殊字符链被系统用来生成 honeywords，为该系统所有用户共享，所以系统仅需存储一个特殊字符链即可，即只需存储 33 个特殊字符的一种排列。特殊字符链的某种排列如图 1 所示。

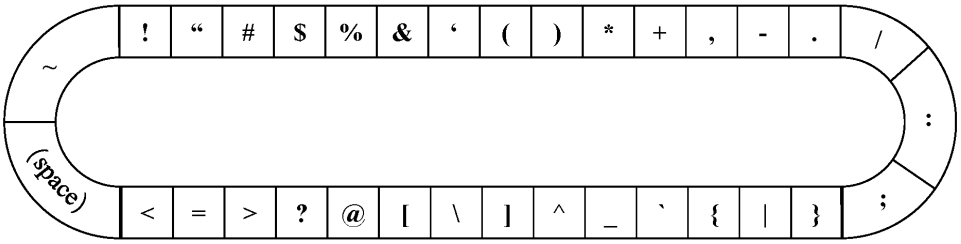

Figure 1. Special character chain
图 1. 特殊字符链

利用用户口令当中的两个不同的特殊字符，根据特殊字符链，SCDM 就可以计算出两个特殊字符的距离。在系统初始化时，一旦生成特殊字符链，就不再改变。除生成特殊字符链外，部署 SCDM 不需要进行用公开的口令文件建立概率模型等工作。即，SCDM 初始化非常简单便捷。

3.2. 虚拟 Honeyword

定义：特殊字符间距(Special Character Distance)为特殊字符 e_1 和 e_2 的间距 $SCD(e_1,e_2)$ 为从 e_1 沿特殊字符链的顺时针方向到 e_2 经过的节点的数目，其中 $e_1 \neq e_2$。

由上述定义可知，口令当中必须含有两个不同的特殊字符，而且特殊字符间距的值是大于 0 的。

在主服务器上，SCDM 需要保存用于用户认证的信息，除了用户名、去掉两个特殊字符的口令外，还需要保存特殊字符的距离。在 honeychecker 上，SCDM 则需要保存用户的用户名和口令中的第一个特殊字符。

假设某个用户u_i的用户名是“Ironman”，口令是“Revenge~2018!”。根据图 1，可以得到两特殊字符“~”、“!”的间距为 1。则用户 u_i 在口令文件 F_p 和 honeychecker 上的信息如表 2 和表 3 所示。

考虑到用户 u_i 的口令可能含有相同的特殊字符，系统管理员在部署 SCDM 时，可以自由选择使用某重复特殊字符第一次出现、或者最后一次出现的位置用来生成 Honeyword。譬如：对于口令“!Revenge~2018!”，若选取第一个“！”，则 SCDM 在 F_p 中存储的口令为“Revenge2018!”；若选取最后一个“！”，则 SCDM 在 F_p 中存储的口令为“!Revenge2018”。

Table 2. User u_i ’s information in F_p
表 2. 文件 F_p 中用户 u_i 的信息

Username	Password	FirstIndex	SecondIndex	Distance
Ironman	H(Revenge2018)	7	12	1

Table 3. User u_i ’s information in honeychecker
表 3. Honeychecker 上用户 u_i 的信息

Username	FirstChar
Ironman	~

接下来我们阐述特殊字符间距是如何在生成 honeywords 中发挥作用的。对于既定的字符间距，以任一特殊字符链中的字符作为开头，SCDM 可以产生 $|SCC|$ 个特殊字符对。其中，$|SCC|$ 表示特殊字符链中

的字符个数。

假设攻击者已经得到了文件 F_p，她可以看到的信息有用户名、去除了两特殊字符的口令，但是特殊字符间距会产生$|SCC|$ (此处为 33)个可能的口令，供攻击者挑选。因此，只需要额外存储特殊字符间距，SCDM 就能构造出含有 33 个 sweetwords 的虚拟列表，从而使攻击者感到迷惑。

4. 方案分析

4.1. 探测口令泄露的概率

用户登录时输入用户名 u_i 和口令 p_i，系统首先在文件 F_p 中查看用户 u_i 是否存在；若存在，系统从文件 F_p 中得到与 u_i 对应的口令哈希值，$H(p_i')$。接下来，系统按照 SCDM 的具体实现，从 p_i 中选取两特殊字符，去除后得到 g_i，然后比对 $H(g_i)$ 是否和 $H(p_i')$ 一致。如果不一致，则提示口令错误，拒绝登录；如果一致，则对两个特殊字符进行验证。首先，根据存储的特殊字符链，生成两特殊字符距离。只有生成的距离和 F_p 中存储的距离一致时，系统才和 honeychecker 进行交互；否则，拒绝登录，并记录该用户的登录行为——因为可能是攻击者仅仅得到文件 F_p 后，在不了解系统已经采用了 SCDM，而尝试进行登录。

由于攻击者可能猜到的特殊字符之间的距离和系统中存储的一致，所以系统和 honeychecker 进行下一步校验是有必要的。当系统和 honeychecker 进行校验时，系统只用将用户名和第一个特殊字符传送给 honeychecker。此处需要强调的是，只有当攻击者猜测的特殊字符完全正确，她的第一个特殊字符才能和 honeychecker 匹配成功。如果 honeychecker 找到正确匹配，则返回正面的反馈；否则，触发警报通知管理员——已经探测到口令泄露。

因此，即使攻击者得到了泄露的口令文件 F_p 并把所有的信息转化为明文，SCDM 依然能够通过虚拟出一个含有 33 个 sweetwords 的列表来迷惑攻击者。因此，仅仅通过要求用户在口令当中含有两个不一样的特殊字符，SCDM 就能以 32/33 (96.97%)之高的概率探测到攻击行为。

4.2. 安全性分析

泄露的口令文件 F_p 中，每个用户都有 k 个 sweetwords，攻击者因此而感到迷惑。然而有些时候攻击者可以从 W_i 中轻易的选出用户 u_i 的原始口令(例如，用户名和口令之间存在着关联)。一个完全 flat 的、基于 honeyword 的认证方案中，攻击者在从 W_i 中甄选用户 u_i 的口令时不应有任何优势。因此，一个完全 flat 的 honeyword 生成算法，从 W_i 中选出原始口令概率应该是 $1/k$。

SCDM 中特殊字符链是随机排列而成的，SCDM 在生成虚拟的 sweetword 列表时，每个 sweetword 出现的概率都是相等的；此外，在前面的例子中，用户名 Ironman 和口令 Revenge~2018!存在着关联，但是由于每个 honeyword 只是特殊字符不同，所以每个 honeyword 和用户名也存在这种关联，攻击者从 sweetwords 中选择时不具有任何优势。因此，SCDM 是可以生成完全 flat 的 sweetwords 的。

5. 结束语

本文分析了现有的 honeyword 生成机制依然存在安全性较弱、占用存储空间过大等问题。然后，提出了基于特殊字符间距的虚拟 honeyword 生成机制，并对所提方案进行了深入分析。分析结果显示本文所提方案可以显著地减少存储空间开销，并提升安全性和口令集泄漏被检测的概率。

致　谢

本文工作受到国家重点研发计划(项目号：2017YFB0802300)的资助。

参考文献

[1] Hackett, R. (2017) Yahoo Raises Breach Estimate to Full 3 Billion Accounts, by far Biggest Known. http://fortune.com/2017/10/03/yahoo-breach-mail/

[2] Heim, P. (2016) Resetting Passwords to Keep Your Files Safe. https://blogs.dropbox.com/dropbox/2016/08/resetting-passwords-to-keep-your-files-safe/

[3] Ragan, S. (2016) Weebly Data Breach Affects 43 Million Customers. http://bit.ly/2kP4EA2

[4] Weir, C. (2016) Cracking the Myspace List-First Impressions. http://reusablesec.blogspot.kr/2016/07 /cracking-myspace-list-first-impressions.html

[5] Contributors, W. (2012) 2012 Linkedin Hack. https://en.wikipedia.org/w/index.php?title=2012_LinkedIn_hack&oldid=722095159

[6] Khandelwal, S. (2014) Hacking Any Ebay Account in Just 1 Minute. https://thehackernews.com/2014/09/hacking-ebay-accounts.html

[7] Schneier, B. (2013) Cryptographic Blunders Revealed by Adobe's Password Leak. https://www.schneier.com/blog/archives/2013/11/cryptographic_b.html

[8] Brown, K. (2013) The Dangers of Weak Hashes. SANS Institute InfoSec Reading Room, MD, 1-22.

[9] Weir, M., Aggarwal, S., De Medeiros, B. and Glodek, B. (2009) Password Cracking Using Probabilistic Context-Free Grammars. 2009 30*th IEEE Symposium on Security and Privacy*, Berkeley, CA, 17-20 May 2009, 391-405. https://doi.org/10.1109/SP.2009.8

[10] Wang, X. and Yu, H. (2005) How to Break md5 and Other Hash Functions. Annual International Conference on the Theory and Applications of Cryptographic Techniques. In: Cramer, R., Ed., *Advances in Cryptology-EUROCRYPT* 2005. *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 19-35. https://doi.org/10.1007/11426639_2

[11] Enterprise, V. (2016) 2016 Data Breach Investigations Report. Verizon Enterprise. http://www.verizonenterprise.com/resources/reports/rp_dbir-2016-executive-summary_xg_en.pdf

[12] Almeshekah, M.H., Spafford, E.H. and Atallah, M.J. (2013) Improving Security Using Deception. Center for Education and Research Information Assurance and Security. Purdue University, West Lafayette, IN.

[13] Herley, C. and Florêncio, D. (2008) Protecting Financial Institutions from Brute-Force Attacks. IFIP International Information Security Conference. In: Jajodia, S., Samarati, P. and Cimato, S., Eds., *Proceedings of The Ifip Tc* 11 23*rd International Information Security Conference. IFIP-The International Federation for Information Processing*, Springer, Boston, MA, 681-685. https://doi.org/10.1007/978-0-387-09699-5_45

[14] Bojinov, H., Bursztein, E., Boyen, X. and Boneh, D. (2010) Kamouflage: Loss-Resistant Password Management. In: Gritzalis, D., Preneel, B. and Theoharidou, M., Eds., *Computer Security-ESORICS* 2010. *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 286-302. https://doi.org/10.1007/978-3-642-15497-3_18

[15] Juels, A. and Rivest, R.L. (2013) Honeywords: Making Password-Cracking Detectable. In: *Proceedings of the* 2013 *ACM SIGSAC Conference on Computer & Communications Security*, ACM, New York, 145-160. https://doi.org/10.1145/2508859.2516671

[16] Erguler, I. (2016) Achieving Flatness: Selecting the Honeywords from Existing User Passwords. *IEEE Transactions on Dependable and Secure Computing*, **13**, 284-295. https://doi.org/10.1109/TDSC.2015.2406707

[17] Burnett, M. (2011) 10,000 Top Passwords. https://xato.net/passwords/more-top-worst-passwords

[18] Kwon, T., Shin, S. and Na, S. (2014) Covert Attentional Shoulder Surfing: Human Adversaries Are More Powerful than Expected. *IEEE Transactions on Systems, Man, and Cybernetics*: *Systems*, **44**, 716-727. https://doi.org/10.1109/TSMC.2013.2270227

[19] Shen, C., Yu, T., Xu, H., Yang, G. and Guan, X. (2016) User Practice in Password Security: An Empirical Study of Real-Life Passwords in the Wild. *Computers & Security*, **61**, 130-141. https://doi.org/10.1016/j.cose.2016.05.007

[20] ASA X3.4-1963 (1963) American Standard Code for Information Interchange. American Standards Association.