NDSS'18-Krawiecka-Safekeeper: Protecting Web Passwords using Trusted Execution Environments

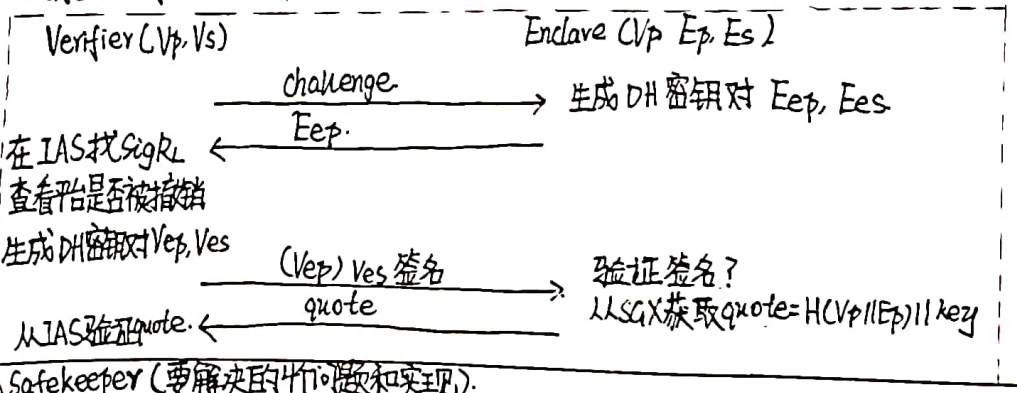
优点：①在抵抗钓鱼攻击和数据库泄漏的同时，考虑了服务器恶意的情况
②将设计分为4个部分，分别描述原理和具体实现，较好.
③对方案进行了安全性、可用性、性能和可部署性分析，很全面.

问题：①本文旨在解决钓鱼攻击，实际上只是保证泄漏口令无法用于攻击，不够清晰.
②在用户评估处，应与其它保护口令方案对比才能说明Safekeeper可在实际使用
③对于恶意脚本的所御最好通过CPS设置白名单，不允许使用脚本很可能降低目前网站的功能

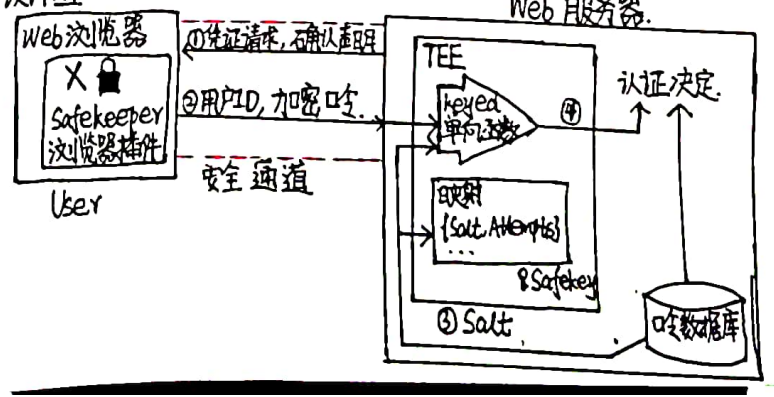
▲研究背景
1>在web中，根据安全性角度，口令是一个相对稳妥的选择，存在两个关键的安全问题：
①对用户口令的钓鱼攻击　②服务器端口令数据库的泄漏
▲由于口令重用增多，泄漏口令的危害超出了泄漏的网站本身. 服务器端
2>目前一些解决口令数据库泄漏的方案依赖于硬件的可信执行环境(TEE)，但对于恶意的服务器，仍无法保护口令的机密性.
3>设计抵抗恶意服务器的方案存在许多挑战.
①安全性：如何向认证服务器隐藏口令本身？如何限制服务器对口令的猜测频率？
②可用性：如何最小化用户的负担？如何支持多个设备登录？
③用户可验证性：如何告知用户方案处于激活状态？
④性能：如何高效地实现该方案？
⑤可部署性：如何简便、廉价地在流行网站框架中集成？
4>结合NIST对身份指南要求口令以.keyed.单向函数的形式存储，例如CMAC (cipher-based MAC)

▲研究问题
1>安全性目标：即使在恶意服务器的情况下，也可以防御钓鱼攻击，并防止口令数据库泄漏的攻击.
在安全性保证的前提下，保证可用性、性能和可部署性.
2>口令机密性被破坏的几种方式：①用户无意/错误地直接向敌对发送口令，例钓鱼攻击
②口令信息或口令本身从服务器泄漏，例数据库失窃　③服务器腐化
3>敌手模型(重点考虑服务器端敌手)
▲获取明文口令：利用口令重用访问其它网站；访问由口令生成密钥的用户数据；向其它恶意实体出售口令.
①访问口令数据库/恶意服务器或窃取的数据库.
②修改网页内容：可修改任何发送给用户的数据，包括客户端脚本的动态数据(如通过XSS攻击)
③访问C-S通信：获取所有发送到web服务器的数据.
④已知服务器端所有软件，并可以在服务器端执行任意软件代码.
⑤可执行(定向)钓鱼攻击
4>全面保护口令认证系统的解决方案需做到两点需求：
①口令保护：除了口令猜测没有更好的方案获取口令；离线猜测在计算上不可行；在线猜测受速率限制
②用户需要在服务器给出提示时，准确地确定输入口令是否安全.
5>可部署性目标
①不应给基于口令的认证带来明显的性能下降，无论是完成单次认证所需的时间，还是完成认证的总体速率.
②不需现有软件较大的改变即可将解决方案集成到软件系统中
③可以透明升级现有口令系统(例，不需要重置口令)，更改和重置口令的现有机制也不受影响

▲用到的知识-Intel SGX & TEE
1>TEE提供了安全敏感代码的隔离执行，在TEE中的代码才能访问其中的数据
2>SGX指支持允许用户空间应用建立硬件增强的TEE，enclave，在应用的虚址运行.
一旦初始化，只有enclave中的代码才能访问其中的数据.
→应用可以通过ecalls调用enclave函数；enclave数据存储在特定内存区域EPC，读取时由CPU对访问的key加密并进行完整性保护.
→在初始化阶段CPU衡量了enclave的代码和配置，组成enclave实体值，MRENCLAVE值
enclave的数据的加密的key，只能由同CPU且同MRENCLAVE的enclave访问.
3>SGX远程认证：verifier确认远程prover的硬件和软件的正确性.
SGX让verifier依赖于IAS来验证远程prover enclave的认证证据



▲Safekeeper (要解决的4个问题和实现).
1>设计图



2> 服务器端口令保护

① 将口令和相应盐值作为输入输出CMAC存储在数据库中.

为了保护CMAC key, Safekeeper在服务器端的TEE计算CMAC, key在TEE中.

因此, 在无CMAC key的条件下, 无法进行离线猜测攻击.

② 本文使用Intel SGX提供TEE. 使用enclave的ecalls中的init()生成CMAC key, 使用process()计算口令的CMAC值并返回结果.

3> 速率限制(在每个帐户基础上做速率限制, 保护每个的帐户且避免帐户间影响)

① ▲Safekeeper无法区分猜测攻击和用户正常的登录行为

CMAC运算中未添加用户ID, 故本文使用salt(帐户唯一)限制速率.

▲ 为了避免强制两次尝试间隔W分钟带来的可用性问题, 便在固定的时间间隔内允许固定次数的尝试, 这个时间后重置尝试次数并设置下一个时间节点.

② 在实现中, 除了web服务器的速率限制(例 Captcha), 调用process()时先查看此时间间隔内 attempts_i 的值.

→ 为了允许enclave重启, shutdown()函数在enclave外安全存储state信息, 包括CMAC key, salt, 尝试次数和 t_reset, 可通过init()恢复.

另外, 如果恢复失败, enclave则将t_reset设置成1个未来值, 并将尝试次数置为0

4> 远程认证.

① 为了安全传输用户口令, Safekeeper浏览器扩展必须通过远程认证Safekeeper. TA, 且扩展必须与TA建立安全信道. (扩展处存储服务器的白名单)

→认证过程包括密钥协商, 生成会话密钥加密口令

5> 客户端右确认机制

① 服务器在网页中指定哪些输入字段被加密, 浏览器插件解析该信息, 并加密输入的文本。用户手动点击插件图标才会显示高亮加密字段(secure attention sequence)

② 扩展包括后台脚本(实现主要功能)和内容脚本(注入网页来与网页交互)

扩展可以访问标签页数据, 使用浏览器内存, 捕获并修改特定web请求.

▲讨论.

1> Safekeeper的其它扩展和变体.

① 对右更件信息提供足够的备份和恢复机制, 采用unanimous approval的方式

② 扩展到多个服务器上, 利用上述①中的备份进行扩展.

③ 许多网站利用邮件作口令重置机制, 可利用保护口令类似的机制保护邮件防止篡改

④ 另一种对抗恶意脚本的方法(无需全部禁用): 弹出Safekeeper控制的窗口用于输入需保护的字段, 但会导致评估口令策略和口令强度评测无法执行.

2> 集成到口令管理器上

① 可以使PM与Safekeeper协商会话密钥, 保护客户端输入的口令

② 在PM处实现TEE, 密钥协商和加密可在TEE内完成.