KLASSP: Entering Passwords on a Spyware Infected Machine Using a Shared-Secret Proxy

Dinei Florêncio and Cormac Herley
Microsoft Research
One Microsoft Way
Redmond, WA

ABSTRACT

In this paper we examine the problem of entering sensitive data, such as passwords, from an untrusted machine. By untrusted we mean that it is suspected to be infected with spyware which snoops on the user's activity. Using such a machine is obviously undesirable, and yet roaming users often have no choice. They are in no position to judge the security status of internet cafe, airport lounge or business center machines. Either malice or negligence on the part of an administrator means that any such machine can easily be running a keylogger. The roaming user has no reliable way of determining whether it is safe, and has no alternative to typing the password.

We consider whether it is possible to enter data to confound spyware assumed to be running on the machine in question. The difficulty of mounting a collusion attack on a single user's password makes the problem more tractable than it might appear. We explore several approaches. In the first, we show how the user can embed a password in random keystrokes to confuse spyware, while leaving the actual login unaffected. In the second we employ a proxy server to strip random keys. In the third we again employ a proxy that inverts a key mapping performed by the user. We examine also several potential attacks.

1. INTRODUCTION

Attacks that target user information on client workstations are certainly not new. Spyware is a very diverse phenomenon [16] and includes a broad range of techniques that snoop on user activity, deploy Trojan downloaders, or infest with adware. However, the last few years have seen a surge in various ploys targeting information that can be directly exploited for financial gain. The popular and technical presses contain many accounts of the growth of this problem.

Keylogging is one of the most insidious threats to a user's personal information. Passwords, credit card numbers, and other sensitive or personally identifying information are potentially exposed. Writing a keylogger is a trivially easy task [11], there are numerous freeware offerings, and many of them make efforts to conceal their presence. For example, they will not show up in a process list. The incidence of keyloggers in-the-wild is growing rapidly. Keylogging is more worrisome in many respects than the related threat of phishing. In phishing the attacker lures the victim into typing a password into a malicious web-site. The social engineering aspect of the attack is important, since the attacker generally does not have any code (beyond possible browser scripts) running on the victims machine. In the problem of spyware, or keylogging, by contrast, the attacker is assumed to have compromised the machine in use.

Enterprize users can most likely trust their desktop systems provided their network administrators maintain good firewall and anti-virus regimes. Knowledgeable home users who keep their systems updated are probably almost as well protected. However home users who are less proficient, or confused, or who leave their systems unpatched are at greater risk. Roaming users who use their own laptops are subject to wireless and network snooping attacks, but are no more at risk from spyware if their perimeter defenses are good. However roaming users who use unfamiliar machines are in the worst case of all: the spyware infection status of public machines must be regarded as unknown. Safety requires knowing that the administrator is both competent and trustworthy. As things stand, a roaming user has no reliable way to determine if a machine is running a keylogger or not. In this environment every session on such a machine must be assumed to be logged. We call out system KLASSP: KeyLogger Avoidance using a Shared Secret Proxy.

1.1 The Problem and Limitations of Our Approaches

The problem we address is to determine what a user can do when she must login to a password protected account from an untrusted machine. One answer is that she should never do so. This is a simple advice, but unsatisfactory, and millions upon millions ignore it every day. If this advice were followed internet cafes, business centers and airport kiosks would have no customers.

Clearly, this is not the case. While every machine is an untrusted machine to some degree, and every user is at risk, we are principally interested in those most at risk: roaming users of public machines. Furthermore — while all data typed by the user may be sensitive — we are principally interested in protecting passwords. We assume that preventing a password falling into the wrong hands is more important than protecting the rest of the data from a login session. This last fact is important, since the solutions we explore will require additional steps or effort on the user's part when entering the password; this effort may be well worthwhile to protect a password, but not ordinary search queries for example.

We make several requirements of the solutions in order to be useful.

- No change to the login server. The techniques we explore must work with existing login servers; *i.e.* we don't expect that login servers are going to alter anything.
- No change to the browser or client software environment. We do not assume that the roaming user has installation privileges.
- User cares most about obscuring her password; it matters less if other activity is captured.
- A roaming user is unlikely to login more than once or twice to the same account from the same untrusted machine. We further assume that collusion among spyware infected machines is unlikely. For example a keylogger on a machine in the airport in San Francisco is unlikely to be able to collude with one running on an internet cafe machine in Milan.

The general approach we use is to obscure the password typed by the user, and employ a Proxy to map the typed keystrokes to the actual password. We have two general approaches. The first resembles a winnowing and chaffing [14] algorithm, where the actual password is typed in the clear, but embedded in random keystrokes. This makes the attackers job of extracting the actual password extremely hard. The second approach involves mapping the keys of the password using a mapping agreed in advance with the proxy server; this is described in Section 3.4. In the next section we describe previous and related work.

2. RELATED WORK

Much of the recent work on desktop computer security has been directed at securing the perimeter. Commercial anti-virus technologies attempt to detect files with malicious payloads. More recently there have been anti-spyware offerings that attempt to address the problem of code that maliciously spies and reports on a user's activity. Some commercial offerings attempt signature based identification of spyware applications on desktop machines. For an excellent overview of the range of threats and the technologies employed against them see [18]. Excellent systematic studies of spyware in the wild using honeypot techniques are [16, 20]. An interesting analysis of the difficulty of ensuring the integrity of user key and mouse events is given in [10]. We refer interested readers to [11, 18] for excellent accounts of the difficulties of this field.

Rather than have users key their passwords some web sites have experimented with on-screen keyboards as a method of secure data entry. These schemes can be attacked by having the keylogger do a screen capture at each mouse click event. The problem of password phishing has attracted lots of attention recently [15, 12, 9, 6]. Generally the target of a phishing attack is the password or other sensitive information such as credit card number of the victim. In contrast to the spyware case the machine being used is not generally compromised. Rather the user is directed to a web page that manipulates them into willingly typing their password into a malicious site (in the belief that the site is benign).

Password management systems are one approach to helping users with the risks and difficulties associated with managing multiple passwords. We distinguish between those that store the passwords on the client, *e.g.* [15, 9], and those that store in the cloud [8]. An early in-the-cloud example, proposed by Gaber *et al.* [8], used a master password when a browser session was initiated to access a web proxy, and unique domain-specific passwords were used for other web sites. Since users authenticated themselves by typing the master password, this clearly offers no defence against keyloggers. The same is true of other in-the-cloud password protected management systems such as Passport, where the user authenticates herself using a master password.

There are several client based password management systems. These are interesting, but less relevant to the problem of protecting roaming users. Ross *et al.* [15] propose a solution that, like [8], uses domain-specific passwords for web sites. A browser plug-in hashes the password salted with the domain name of the requesting site. Halderman *et al.* [9] also propose a system to manage a user's passwords. Passwords both for web sites and other applications on the user's computer are protected. While client password management schemes can help with the problem of phishing they are of little help if a keylogger captures the password, and offer roaming users no protection. Using bookmarklets (*i.e.* Javascript functions that can be stored as favorites in the browser) Zarate [1] describes how to automatically populate password fields, with domain specific passwords by typing only a master password. This of

course would not help roaming users.

The surge of spyware and keyloggers is a somewhat recent phenomenon, so there have been relatively few works on protecting the roaming users of untrusted machines. An interesting work by Tan *et al.* [19] addresses the question of minimizing the chances that a password entered using an on-screen keyboard is captured by an observer. This work addresses the "shoulder surfing" risk rather than the risk that the machine itself is running spyware, but has interesting analysis of the usability of various alternative password entering mechanisms.

The closest work to the system we propose here is the Impostor system of Pashalidis and Mitchell [13], which specifically addresses the question of entering passwords on compromised machines. This is a password management system where roaming users can access their credentials. Rather than have users authenticate themselves by typing a master password, a challenge response authentication is used. The user is assigned a large string that forms the secret. When requesting access the user is challenged to provide characters from randomly selected positions in the string, and is authenticated only if she responds correctly. In this way the user reveals only a small portion of the secret string to any compromised machine. A replay attack is not possible, since the challenge positions change each time the user contacts the proxy. In common with our system and [8] Impostor also runs as a Man-In-The-Middle (MITM) proxy, and the user must direct their browser to the proxy. The are several points of difference between KLASSP and [13]. First we do not store the user's password. This has the effect that KLASSP user must trust the proxy machine only when the alternative is trusting an internet café. More importantly, it removes the burden of maintaining up to date password information at the proxy. As with most password management systems the Impostor user must update their passwords at the proxy whenever one changes. By contrast the KLASSP user need register only once by giving (userid, url) pairs, and these seldom change. Since users are likely to use a proxy only when using untrusted machines (presumably a small portion of their total logins for most users), we believe that reducing the burden of maintaining the account is a key advantage. Also, a spyware attacker that discovers the secret used by Impostor gains access to all of the accounts managed, while an attacker who gains access to the KLASSP secret gains access only to accounts where he also observes a successful login. Second, Impostor requires that the user carry a piece of paper with the secret string. Of the methods we describe only that of Section 3.4 requires the user to carry something. Using the method of Section 3.2 the user need merely be able to distinguish picture uploaded by her from randomly chosen other images. Knowing the secret string allows access to all of the accounts maintained by the Impostor proxy. By contrast, knowing the KLASSP shared secret is worth little unless the attacker observes a login session. This reduces the required complexity of the shared secret considerably.

Cheswick [5] examines the use of Challenge-Response authentication mechanisms to evade spyware. The advantage of such systems is that a spy who observes a successful login session cannot perform a replay attack: the challenge will be different for each event and observing a single response helps the attacker very little. Cheswick reviews a number of approaches from the point of view of usability. The most secure approaches require a user to carry a hardware device or a piece of paper. The tradeoff between usability and security remains an open question.

Several one time password systems exist that limit the phisher's ability to exploit any information he obtains. SecureID from RSA gives a user a password that evolves over time, so that each password has a lifetime of only a minute or so. This solution requires that the user be issued with a physical device that generates the password. One time passwords can be based on an SKEY approach [17]. This solution requires considerable infrastructure change on the server side, and has not seen any significant deployment to general users.

Florêncio and Herley [7] describe a simple trick that users can employ to confound keyloggers by obfuscating their passwords. We review the method in Section 2.1, and an elaboration of it forms the basis of one of our approaches. Due to the relevance of that trick to KLASSP, we give a brief overview of that trick.

2.1 Embedding the Password in Random Keys

Extracting a password from a sequence of keystrokes is generally not hard. For example, here is a sequence of keys typed while the browser had focus:

```
hotmail.comsarahj7@hotmail.com<tab>snoopy2
```
(1)

Clearly, the first characters were typed into the address bar to navigate to the `hotmail` login page; the next characters give out the userid and password of this user.

In [7] we show one way to make the attacker's job harder: embedding the actual password in a sequence of random keys, typed outside the text field of the login page. Instead of the password `snoopy2` the keylogger now gets, for example:

```
laspqm5nsdgsos8gfsodg4dpuouuyhdg2
```
(2)

The password is now embedded in random keys, making extracting it a lot harder. Note however that the server is not confused by the extra characters; the browser will differentiate between the legitimate characters (typed when the password field has focus) and the random characters (typed somewhere else), and only forward

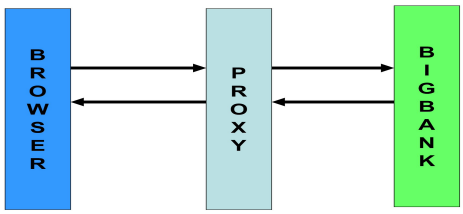

Figure 1: The basic design. All traffic from the browser on the untrusted machine flows through a MITM proxy. The user enters the password using an obscuring method; it is unobscured at the proxy. Once login is established the proxy reverts to a passive rôle.

the legitimate ones to the server.

This simply trick evades most current keyloggers. Unfortunately, as explained in [7], it is not a general solution: keyloggers could be modified to log additional information (e.g., mouse clicks or screen shots), that would allow the attacker to recover the original password.

3. USING A MITM PROXY

While the method of [7] has a number of weaknesses, the idea of embedding the actual password (signal) in a large number of random keystrokes (noise) is a valuable one. The problem is that all of the information the keylogger needed was present (albeit hard to get at) on the compromised machine (this is also true of on-screen keyboards and alternative login approaches). Simply taking a screenshot at each keystroke would allow capturing of the password. Equally, an attacker who instals a spyware plug-in in the browser would see all of the encrypted traffic, including passwords, in the clear.

To avoid leaving the password in any accessible form on the untrusted machine we now explore the possibility of logging in via a MITM proxy. All traffic between the user's browser and the end login server will flow through the proxy as in Figure 1. Our goal is that the user will enter an obscured version of the password, and the proxy will unobscure before passing to the login server. Thereafter the proxy reverts to a passive rôle, but remains in the middle. We'll explore two main approaches to obscuring the password: obfuscating it by embedding it in random characters, and mapping it using a simple encryption table. The key challenges in designing an obscuring method achieving a good tradeoff between usability and security.

It is worth emphasizing that the proxy does not act as a password management system; *i.e.* the user's password is not stored on the proxy (recall from Section 2 that password management systems offer no protection against keyloggers).

3.1 Obscuring Using a Prompt Table

As with the client embedding case in Section 2.1 the key question is how the proxy can determine which keys belong to the true password, and which are random additions. One possible solution is that the proxy prompts the user when a password key should be typed. For example, suppose the proxy displayed a prompt at every key typed by the user: a 0 meaning that the next key should be randomly chosen, and a 1 meaning that it should be the next key of the password. This can easily be accomplished using JavaScript on the proxy login page. The problem is that this is too easily defeated. If the spyware does a screen capture at each keystroke it will see the prompts, and it is a simple matter to then determine which keys belong to the password and which are fake. Anything we display on the untrusted machine must be assumed to be visible to the spyware.

Our scheme works as follows. The user sets up an account with the KLASSP server. She is assigned a shared-secret. For example, a simple shared-secret would be a symbol that will act as her prompt, and a position in a symbol table where she will look for it. She also enters the userids of the accounts that she may be accessing via the proxy server. Note that, after registration, access to the KLASSP webserver is not password protected: when the user is using the proxy to login (for example) to `hotmail` the userid `sarahj7@hotmail.com` combined with the target login domain suffices to uniquely identify her. This enables the proxy to determine the unique prompt symbol and symbol table position for that user.

Suppose we have M symbols and N positions in the table. When entering the password the entire symbol table will be refreshed every time a key is entered. When the user sees her assigned symbol in her assigned position she types the next key of her password; otherwise she types a random key. Now, even assuming that the spyware does a screencapture at every keystroke it is not obvious which keys are which. The evolution of the sample symbol table is shown in Figure 2. At each key the table changes. When the user sees the assigned symbol appear in the assigned position she types the next key of the password; otherwise she types a random key. After the last key of the password the user clicks a submit button to indicate that the password is complete (recall that neither the password nor its length are stored at the proxy).

3.1.1 *Analysis*

To prevent spyware from determining which symbol and which position in the table prompts the user to enter a true password key we must be careful of the statistics of the symbol changes. If we assume the average password is 8 characters long, and that an average of k random characters between successive keys is acceptable, then a total of $(k+1)8$ characters will be typed.

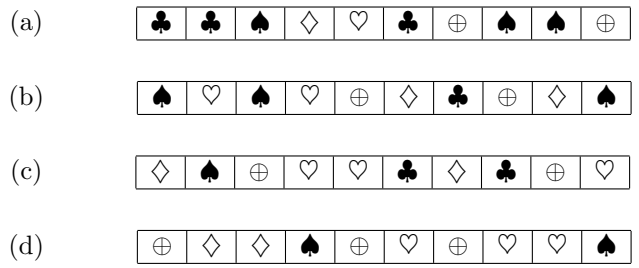

Figure 2: Symbol table as the user types, where the table is of size $N = 10$ and there are $M = 5$ symbols: ♣, ♢, ♡, ♠ and ⊕. The line (a) represents the prompt for the first key, (b) for the second and so on. A user who has been assigned position 7 and the symbol ♢ would type random characters for the first, second and fourth prompt and a true password key at the third. The sequence she sees is ⊕♣♢⊕$\cdots$.

The assigned symbol must appear in the assigned position 8 times. Any other symbol should appear the same number of times over the course of the $(k+1)8$ characters. Hence, we can allow only $M = k+1$ distinct symbols to avoid compromising the scheme statistically.

Since the spyware does not know which position is assigned to the user there will be a total of $N(k+1)$ possible passwords for the spyware to check. If we assume that an average of $k = 4$ random characters between successive password keys is acceptable, and a table of size $N = 100$ is not too large this leaves us with 500 passwords that the spyware must consider. We will examine in Section 3.3.1 what level of security this provides from an entropy point of view.

3.1.2 Other Shared-Secrets

In the simple shared secret example the size of the secret space equals the size of the table times the number of symbols; *i.e.* $N(k+1)$. We wish to have the largest possible shared secret space to prevent entropy or brute force attacks on the password (as in Section 3.3.1) while making the proxy use-model simple. Too large a table becomes difficult to display. There are a number of simple possibilities to expand the shared-secret space without enlarging the table. The first we employ is asking the user to look for her assigned symbol in two different positions: if it occurs in either she types the password key, and if not she types a random key. This increases the number of shared secrets to $N^2(k+1)$ where N is the table size. Using again four random characters on average between true passwords keys, and a size 100 table this increases the shared-secret space to 50000. In general if the user scans p positions for the symbol at each character we can achieve a space of size $N^p(k+1)$.

A second possibility is to fix a single position, but have the user look for a sequence of symbols rather than a single assigned symbol. For example, for an 8-character password the user might be instructed to look for the sequence ♣♠♢♡♣♠♢♣. That is she types the first character of the password on seeing ♣for the first time, the second on seeing ♠, and so on. The shared-secret space now becomes $N(k+1)^8$ for a length 8 password, which gives $39e6$ possibilities for $k = 4$ and a table of size 100.

3.2 Obscuring Using Known Images

There is a large number of possible passwords if we use the prompt table and one of the shared secrets of Section 3.1.2. However, entering the password becomes harder and more error prone as the size of the possible password space increases. We now explore the possibility of prompting the user with images with which they are familiar.

At registration time the user uploads a number of images, at least equal to the length of the longest password she will use, say L. These are her own personal images. They can be pictures of friends, objects, travel scenes; they can be taken by her personally or obtained elsewhere, the only requirement is that she be able to distinguish her images from other randomly chosen ones. Using this scheme the login procedure is much as in Section 3.1, but instead of being presented with a prompt table she is shown a new image after every keystroke. The images are chosen randomly from a collection of $5L$, where L are the images that she uploaded, and $4L$ are images with which she has no association. On seeing an image she types the next character of her password if the image is one of hers, and a random character otherwise. It has been our experience that users enter passwords more accurately and reliably using this method than using the prompt table approach.

3.2.1 Analysis

Assuming that an attacker has no cues to help him determine which images are the users there are $(k+1)^p$ possible passwords where p is the password length.

Obviously all $5L$ of the images must be stripped of all metadata that might allow an attacker to classify which L belong to the user. Further, the same $4L$ random images should be used over and over again as the user logs in many time. In a community of a large number of users employing this login method each user might be assigned the L images from each of 4 other users.

3.3 Attacks

The methods explored in Sections 3.1 and 3.2 rely upon the insertion of random characters to confuse a keylogger. This implies that the actual passwords characters are still all present, allowing an attacker to use entropy or collusion to try and recover the password. We now analyze these attacks.

3.3.1 Entropy Attacks

Using the fact that actual passwords chosen by users commonly have low entropy, an attacker may be able to narrow the search space for correct password by examining only the low entropy possibilities. For example, the choice `snoopy2` appears much more likely a password than `sdgsdio`.

Given that a spyware attacker will have as many passwords to try as the size of the shared-secret space we cannot afford to make his task even simpler. For this reason we favor the shared-secrets introduced in Section 3.1.2, which essentially avoid the attacker using second order entropy analysis. By using that, we increase the probability that low entropy passwords will withstand attack. An important exception exists for passwords that are numeric; *i.e.* consist of numbers only. If we assume that in numeric PINs all 10 digits are equally likely, and that the randomly typed characters are also numeric, the entropy attack gains very little. Thus a much simpler shared-secret can be employed for numeric passwords, since the attacker's main option would appear to be to attempt login.

3.3.2 Collusion/Averaging Attack

When trying to extract signal from noise multiple independent measurements can help reduce the noise. It is for this reason that we assumed that collusion among spyware machines was unlikely. For example the embedded string in (1) was:

```
laspqm5nsdgsos8gfsodg4dpuouuyhdg2.        (3)
```

On another occasion for the same user logging into the same account the string might be

```
wqsasdfnk4olou3dnsodgsjap1yheyjedrd2.     (4)
```

As more embedded strings are gathered the password keys are the only thing constant as everything else changes. A simple dynamic programming approach will likely reveal the password if it has access to even two of the embedded strings. For this reason spyware has a far simpler task if the first login attempt is unsuccessful and the user types it a second time. Equally, if a user realizes that she typed a random key instead of a password key and backspaces to correct it, she generally gives away one key of the password.

3.3.3 Man in the Middle Attack

A natural line of attack is for the spyware author to set up a login server, claim to be a user whose userid has already been captured, type a series of keys and watch to see which are relayed from the proxy to the login server. This would be enough to reveal which symbol and position in the table were assigned to that user. This doesn't work, since the proxy will relay the `sarahj7@hotmail.com` password only to `hotmail` and so on. Since the user enumerates the userid and domains she will be using, the attacker cannot induce the proxy to relay any information for that userid to any other domain.

3.4 Obscuring by Mapping Password Keys

In Sections 3.1 and 3.2, we obscured the password by inserting random keys. While that will certainly make recovering the password harder, the amount of randomness that can be inserted is limited by the fact that the password characters are still typed in the clear. A much stronger method to obscure the password is to perform a random mapping of the keys, and do the reverse mapping at the proxy. We will now examine this alternative.

In this approach, we assume at registration time the user gets a printed table that she will be use to encrypt (map) the password. The idea is that, since passwords are usually short, we can use a character-by-character encryption table, and ask the user to do the encryption herself.

More specifically, before a trip, the user goes to the KLASSP webserver site, and ask for an encryption table, similar to the one presented in Figure 3. Each table is random and a new table is generated for each user, each time. Each table has an ID number. Note that since we assume no collusion, she can print a table at one current (unsafe) internet cafe for use in her next (also unsafe) location. When logging in from an unsafe location, she goes to the proxy site, and types the desired target site (e.g., www.BigBank.com), and the ID. Figure 4 shows a simple version of the user interface. The proxy shows the target page in the lower frame. When the proxy detect that the focus is in a password field, the top frame instructs the user on which column in the table to use for entering the next password character. According to the user's choice, a new column in the table is used for every new session (recommended) or for every character (for maximum security). If the same table is used for too many sessions, the user may exhaust the table; if so the the last entry is used as many times as required. Note that re-using the last entry is simply a last-resort effort; the user should obtain a new table before the current one is exhausted.

If the user is using the one column per character approach, breaking the encryption is essentially impossible, since any key will be mapped to any key with same probability. Even if we use the same column for the whole session, only a few characters will have been typed, and no significant statistical information can be obtained to try and break the encryption table. Only when the last column is being used repeatedly (i.e., after the one time pad is exhausted, and until the user obtain a new one) there is any possibility of breaking the encryption. In this case, while the user is still safe from a direct decryption attack for several many characters, more efficient attacks are possible, as described

TABLE ID: 734955

	1	2	3	4	5	6	7	8	9	10
A	r	5	s	h	2	k	l	s	F	f
B	f	r	F	s	f	J	u	5	r	T
C	7	)	5	3	h	T	k	g	A	i
D	G	5	o	P	L	y	z	Z	d	x
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 3: A segment of sample encryption table, showing the mapping for characters A-D. Each column corresponds to a particular random mapping. For example, if instructed by the proxy to use column 6, and the next password character is "D", the user would type "y" instead.

below.

3.4.1 *Attacks and possible solutions*

The use of a one-time key removes the opportunity for any direct attack on the keys. Nevertheless, a few other attacks are still possible, if we assume the attacker has complete control over the compromised machine. In particular, a two-men-in-the middle attack, intercepting the keys before it reaches the proxy as well as on the way out would be very insidious. More specifically, a DNS redirection sends the keystrokes to redirect of the original proxy to a hacker site. The site then intercepts and changes the target site (e.g., from BigBank.com to FakeBank.com). The modified site will then receive the clear password. This attack can be avoided as long as the user remember to check the SSL certificate of the proxy. If the user is willing to declare in advance which domains will be visited (during the registration phase) this problem is avoided as the proxy will not send keystrokes to any site not on that list, or that is not known to be safe.

Another point of attack is by exhausting the keys in the table. More specifically, the compromised computer could change the typed characters so that you cannot login at the site. As the users keeps trying to login, the keys will be exhausted. The last key is then used for any subsequent entry. After the user leaves, the attacker can then submit the same keystrokes to a fake site and get the password in the clear. A solution similar to the one proposed above would also circumvent this attack. We could also instruct the user not to keep using the table once it reaches the last entry (note: we allow the user to keep using it, since is most likely still safer then typing the password in the clear).

4. IMPLEMENTATION

We now address some of the issues related to implementation of the proxy. This architecture illustrated in Figure 5 is common to each of the methods of obscuring the password described in Section 3. We first describe the sequence of events, and the flow of connections between the user's browser, the KLASSP proxy,

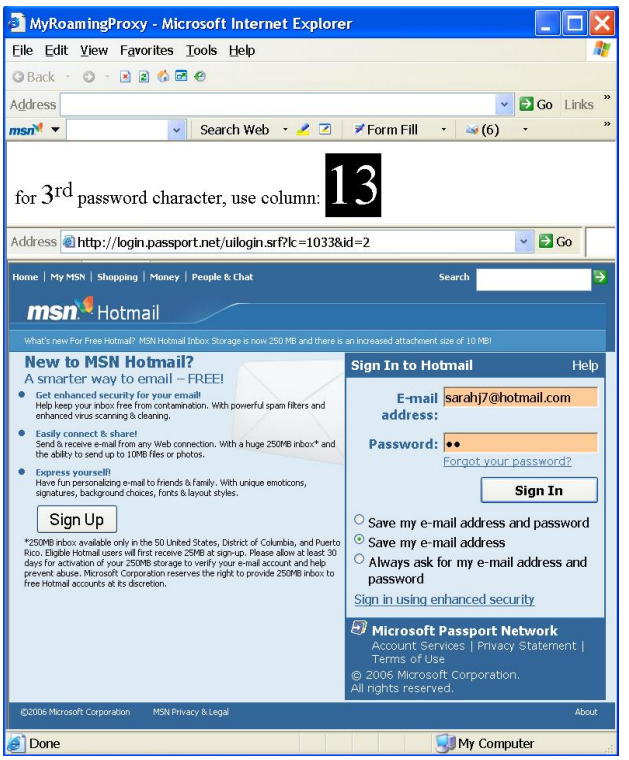

Figure 4: Sample screen capture of the proxy server interface. The upper frame tells the user which column of the table to use for the next character. The lower frame is the actual site to which the user is logging in.

the KLASSP webserver and the end login server.

In our implementation we rely on running Javascript scripts in the user's browser. We use ASP.Net scripting to handle the the actions to be performed at the web server. We use JScript scripting in the KLASSP proxy to alter certain requests and responses between the client browser and the login server.

4.1 Sequence of Events

1. User sets browser to point at the proxy
2. User navigates to the KLASSP webserver, enters the userid and url (*e.g.* `www.bigbank.com/login` and gets Shared Secret prompts
3. User enters obscured password, using one of the methods in Section 3, and submits to the KLASSP webserver
4. The KLASSP webserver server extracts the password
5. Browser is auto-transferred to request the the url
6. Proxy intercepts response and populates login form with userid and "roguepwd"
7. User receives pre-populated `actualLogin` page and clicks submit button.

8. Proxy intercepts request and replaces "roguepwd" with pwd XOR salt from the database. It deletes the entire record (userid, pwd XOR salt, actualLogin) from the database

9. Login proceeds and proxy reverts to MITM rôle.

4.2 User Experience

To use the service a user first points the browser at the proxy server. In both Internet Explorer and Firefox this is done in the Connection Settings tab of the options menu. By entering the IP address of the proxy we are sure that all connections flow through the proxy. Note that this does not require any installation or privileges that are not available to all users. For example a user at an internet café will be able to do this (even guest accounts on a Windows system have this privilege).

The user next navigates to KLASSP webserver and enters the obscured password using one of the shared secret methods described in Section 3. When the obscured password has been uploaded to the webserver the user's browser automatically opens `www.bigbank.com/login`. The user need type nothing further and merely clicks the submit button and login proceeds.

4.3 KLASSP webserver

The KLASSP webserver acts as the visible component. The user is first asked for the address `actualLogin` of the login site, and her userid at that site. At this point the webserver retrieves the shared secret for that user. The user now enters the obscured password using one of the shared secret methods described in Section 3. For the methods of Sections 3.1 and 3.2 all of the images are downloaded to the browser at once to avoid the possible delay of a roundtrip to the server at each keystroke. The images are labeled in the order in which they will be displayed, and thus reveal nothing of the shared secret. Using the `onkeydown` event handler a new image is displayed every time the use types a key. With the method of Section 3.4 no action is needed until the user submits.

In any of the three cases, when the user clicks the "Submit" button on the entire obscured password string gets uploaded to the webserver. This will be the password embedded in junk characters, or an encrypted version of the password, depending on the entry method. The webserver extracts the true password from the obscured string and stores it temporarily for retrieval by the proxy. The password is XORed with a user-specific salt that was assigned at registration time.

The last action of the webserver is to instruct the user's browser to open `actualLogin`: using the `onclick` event for the "Submit" button we can use, for example the Javascript command `http://www.bigbank.com/login`.

4.4 KLASSP proxy

Recall that all connections for the browser pass through the proxy. As our foundation we used the Fiddler debugging proxy version 2.0.5.0 [2], which allows interception of all sessions, including those that are SSL encrypted. Fiddler also provides a Jscript scripting mechanism that allows filtering and altering requests and responses. While we have used Fiddler, we point out that several other debugging proxies also allow modification of requests and responses; see, for example Paros [3] and BurpSuite [4].

Observe from Figure 5 that, while it sits as a MITM in all of the actions of the browser, it is only after the obscured password has been uploaded to the webserver that the proxy starts to play a vital rôle. When the user retrieves the page `actualLogin` (*e.g.* `http://www.bigbank.com/login`) the request and response both flow through the proxy (Steps 5 and 6). Recall, that `actualLogin` is the target login page, and thus contains a both a userid and password form field. At this point the proxy scripts populate these fields before passing them to the browser. The userid is populated with the actual userid which has been deposited in the database, while the password field is populated with the string "roguepwd." To replace the password we merely search for an replace the string `type=''password''` with `type=''password'' value=''roguepwd''`. This is done in the `onBeforeReponse` handler provided by Fiddler; this handler allows us to edit responses coming back from a server befire they are passed to the browser. To replace the userid value, we do similarly. However, the userid is merely a text field, and there may be several on the page, so the string `type=''text''` is not sufficient to indicate that we have found the right one. The userid field has an id that can be different for different sites; for example at PayPal it is `id=''login_email''` and at WellsFargo it is `id=''SSN''`. Rather than manually determine the string for each possible login site we maintain a cumulative list of the id of the userid fields for all of the login sites encountered so far. We find that the number of distinct labels is far fewer than the number of sites (*e.g.*the id "SSN" is common). If any of these labels is found we replace, for example `id=''login_email''` with `id=''login_email'' value='' userid ''`, where userid is the actual userid retrieved from the database. If this fails, and the userid field has a label we have not previously encountered, we enter the userid as `value` for every text field on the page. This will have the effect of populating every text field, including Search and any others present with the userid. Since only the login form will be submitted, these extra entries in other form fields will be ignored when the user submits the login form. Note that by populating the login form fields in the HTML response from the login server we avoid the difficulties of the Javascript same-origin policy.

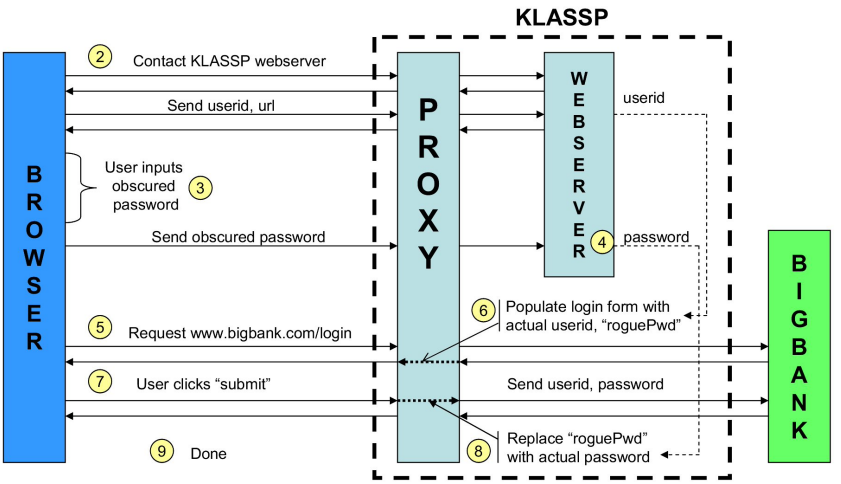

Figure 5: The sequence of steps logging in using the KLASSP proxy. See Section 4.1

Thus when `actualLogin` opens on the user's machine it is pre-populated with the correct userid, but a rogue value password. Obviously we could not place the actual password in the page returned to the user, as this would deliver the unobscured password. The reason for the rogue value password is that many login pages perform scripting checks to prevent submission of the form if the password field is empty. The user now clicks the submit button on `actualLogin`. This request again flows through the proxy, and on this step the proxy replaces the rogue value password with the password retrieved from the database (XORed once again with the user specific salt). This is done in the `onBeforeRequest` handler provided by Fiddler, which allows editting of requests as they are passwed from browser to server. From this point on the proxy merely acts as a MITM between the user's browser and the end site. It maintains an SSL connection to the user and another to the end site.

4.5 Certificates

When a browser is SSL connected it displays certificate information to the user. Generally this will give details of which Certificate Authority (CA) issued, and the details of the recipient institution. When our proxy is running as a MITM it also must act as a CA, and all of the certificates will show up as being issued by the proxy. The user will get warnings that about this, but can merely click "OK" to proceed. The only way of preventing this is to explicitly declare the proxy as a trusted issuer of certificates for that browser. This is a simple process, but has several steps. On an internet kiosk machine the user may not have privilege to declare the proxy CA trusted. Thus we view the certificate warnings as an unavoidable annoyance. The number of certificate warnings displayed to the user depends on the site being visited and the browser; it can as few as one, or as many as seven.

4.6 Registration

To use the service the user must register to establish the shared secret to be used to obscure and un-obscure the password. This is a separate service also hosted by the web server. At registration the user is assigned a shared secret, a position and symbol in the case of Section 3.1, and encryption table in the case of Section 3.4, or uploads personal images in the case of Section 3.2. The user also specifies the urls of the institutions where she will login, and her userids at each of these institutions. Any of the pairs userid and url uniquely identifies the user. This allows the web server to retrieve the correct shared secret without having to ask the user to authenticate herself (typing a password to get access to the service would defeat the whole purpose).

4.7 Burden of maintenance and Trust

A difference between our system and an in-the-cloud password management system is that the passwords do not have to be maintained on the server. We believe this carries two advantages. First, the burden of maintenance on the user is lower. Using a credential management system (CMS), the user must maintain all of the credentials; if she changes her PayPal password, she must then also update the record at the CMS. The user of our system by contrast is not required to maintain anything. She registers for the service once, and is assigned a shared secret. If she employs the image based interface she uploads images once. She need maintain nothing, and can still use the service after a gap of months or years so long as she can successfully distinguish her L images from the $4L$ randomly assigned ones.

Secondly, a CMS must be trusted much more than our system. A rogue employee at a CMS might have access to all of the credentials of all of the users of the system.

A rogue employee at a service running our system by contrast would have to wait for passwords as they come in one at a time. The user of a CMS must trust the service entirely with all of the credentials she uploads. This is true whether she subsequently uses the service or not. The user of our system must trust the system with only the passwords of sites that she logs into, and she does this only when the alternative is trusting an internet kiosk machine.

4.8 Self-Hosting

In-the-cloud password management systems such as Passport and LPWA have been hosted by large servers and served many users. There is no reason, however, that the entire KLASSP system cannot be hosted on a machine maintained by the user herself, and dedicated to serving only her. Exactly such a self-host model is employed by Impostor [13]. Using this approach a user who has a fixed IP address on her home machine might host both proxy and webserver there, and login to BigBank using her home machine as a MITM proxy. This entirely obviates the need to trust any intermediary, and removes the single point of attack that a popular centralized webserver might represent.

5. CONCLUSION

We have presented a new approach to entering a password on a spyware infected machine. We use a shared secret to obscure the password as typed, and a proxy to map back to the actual password. Obscuring the password can be done in various ways. We investigate two possible solutions. The first solution insert random characters between the true password characters by using a pre-agreed secret prompt. The secret prompt can be either a symbol and position in a table, or images uploaded by the user. The second solution employs manual encryption using a pre-printed table. In either case, a proxy is the used to compute the actual password, either by stripping the random keys in the first solution, or by inverting the key mapping in the second case. The second is more secure, but requires that the user carry and consult a printed substitution table.

To the question of whether one can enter passwords securely from a compromised machine thus the answer appears to be a qualified "Yes." The qualification is that each of the schemes we presented burdens the user with a more involved interface than the conventional web login. Nonetheless we think they are the most promising directions for simple password entry from untrusted machines. The difficulty of mounting a collusion attack on a password entered on a public machine makes this problem far more tractable that it appears on the first glance.

Acknowledgements: The authors thank Nikita Pandey for assistance in implementing a version of the proxy.

6. REFERENCES

[1] `http://labs.zarate.org/passwd/`.
[2] `http://www.fiddlertool.com`.
[3] `http://www.parosproxy.org`.
[4] `http://www.portswigger.net/proxy`.
[5] W. Cheswick. Johnny Can Obfuscate: Beyond Mother's Maiden Name. *In Proc. Usenix HotSec*, 2006.
[6] R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. *Symp. on Usable Privacy and Security*, 2005.
[7] Dinei Florêncio and Cormac Herley. How To Login From an Internet Café without Worrying about Keyloggers. *Symp. on Usable Privacy and Security*, 2006.
[8] E. Gaber, P. Gibbons, Y. Matyas, and A. Mayer. How to make personalized web browsing simple, secure and anonymous. *Proc. Finan. Crypto '97.*
[9] J. A. Halderman, B. Waters, and E. Felten. A convenient method for securely managing passwords. *Proceedings of the 14th International World Wide Web Conference (WWW 2005).*
[10] H. Langweg. With Gaming Technology towards Secure User Interfaces . *ACSAC*, 2002.
[11] S. McClure, J. Scambray, and G. Kurtz. *Hacking Exposed*. McAfee, fifth edition, 2005.
[12] P. Oorschot and S. Stubblebine. Countering identity theft through digital uniqueness, location cross-checking, and funneling. *Financial Cryptography*, 2005.
[13] A. Pashalidis and C. J. Mitchell. Impostor: A single sign-on system for use from untrusted devices. *Proceedings of IEEE Globecom*, 2004. `http://impostor.sf.net`.
[14] R. Rivest. Chaffing and Winnowing: Confidentiality without Encryption. 1998. `http://theory.lcs.mit.edu/~rivest/chaffing.txt`.
[15] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser extensions. *Proceedings of the 14th Usenix Security Symposium, 2005.*
[16] S. Saroiu, S. D. Gribble, and H. M. Levy. Measurement and Analysis of Spyware in a University Environment. *Proc. NSDI*, 2004.
[17] B. Schneier. *Applied Cryptography*. Wiley, second edition, 1996.
[18] E. Skoudis and L. Zeltser. *Malware: Fighting Malicious Code*. Prentice Hall, 2004.
[19] D. Tan, P. Keryana, and M. Czerwinski. Spy-resistant keyboard: more secure password entry on public touch screen displays. *CHISIG'05.*
[20] Y.-M. Wang, D. Beck, X. Jiang, and R. Roussev. Automated Web Patrol with Strider HoneyMonkeys. *MSR Tech Report*, 2005.