The following paper was originally published in the
Proceedings of the Fifth USENIX UNIX Security Symposium
Salt Lake City, Utah, June 1995.

One Time Passwords In Everything (OPIE): Experiences with Building and Using Stronger Authentication

Daniel L. McDonald and Randall J. Atkinson
U.S. Naval Research Laboratory, Washington,D.C.
Craig Metz
Kaman Sciences Corporation, Alexandria, Virginia

One Time Passwords In Everything (OPIE): Experiences with Building and Using Stronger Authentication

Daniel L. McDonald
Randall J. Atkinson
U.S. Naval Research Laboratory
Washington, D.C.
danmcd@itd.nrl.navy.mil
atkinson@itd.nrl.navy.mil

Craig Metz
Kaman Sciences Corporation
Alexandria, Virginia
cmetz@itd.nrl.navy.mil

Abstract

The U. S. Naval Research Laboratory's OPIE (One-time Passwords In Everything) Software Distribution is an enhancement of Bellcore's S/Key™ 1.0 package. OPIE improves on S/Key in several areas, including FTP service with one-time passwords, and a stronger algorithm for generating one-time passwords. OPIE diverges from S/Key in select design decisions and in the behavior of certain programs. While not a total security solution, OPIE can be an important part of one. OPIE and its evolutionary predecessors have been used for over a year in parts of NRL. Its use has taught the authors lessons on implementation, usability, deployment, and future directions for improvement.

1 Introduction

In the past decade, computer networks have grown at an explosive rate [Lot92]. In a wide range of environments, such networks have become a mission-critical tool. Organizations are building networks with larger scales than ever before, and many are connecting these networks to the global Internet. Along with this trend has come an explosion in the use of computer networks as a means of gaining illicit access to computer systems. In the past, intruders have used flaws in network software such as older versions of the BSD *sendmail(8)* program to gain entry into remote computer systems [Rey89]. As more vendors and more sites fix the known flaws in their network software, many crackers are now looking for other weaknesses to exploit.

One particularly widespread attack is to passively capture and replay passwords commonly used to authenticate users [CER94b]. Since so many protocols send their passwords in cleartext (that is, there is no encryption of any sort done on the password before it is sent out over the network), anyone who can read network traffic can gain access to whatever is protected by cleartext passwords [HA94]. Many UNIX™-based machines, as well as almost all PCs, allow their network hardware to read any packet transmitted on the attached network. Crackers exploit the ability to sniff packets to discover cleartext passwords, thereby gaining unauthorized access to systems using cleartext reusable passwords.

One solution to this problem is to encode the password in such a way that an encoded password can only be used once and cannot be used to generate any other encoded password. Such an encoded password is called a one-time password because it is usable exactly one time. If an attacker captures such a password from a stream of data sent over a network, he or she cannot use it to gain access to the target system either by using it again (the first condition) or by performing any new coding on it (the second condition). In practice, the second condition is guaranteed by computational infeasability rather than by impossibility – it would take an attacker an inordinately long time to discern any useful data from the intercepted one-time password.

Such an encoding was first devised by Lamport [Lam81], but one-time passwords gained popularity only recently with the development of S/Key [Hal94] by Bellcore. While the design of the S/Key system is not specific to any particular platform, the reference implementation released to the public as the Bellcore S/Key 1.0 Distribution was very specific to 4.3BSD UNIX systems. While it proved to be a valuable tool that has gained wide use in many environments, it left room for improvement.

In response to widespread passive attacks in the Internet reported in early 1994 [CER94b], we started refitting local UNIX machines with Bellcore S/Key. Subsequent additions to S/Key evolved into NRL OPIE (One-time Passwords In Everything). While it is certainly a work derived from the S/Key 1.0 Distribution, the OPIE Version 2[1] Software Distribution noticeably enhances its ancestor.

The development and testing of OPIE raised many issues, both old and new. Classic issues such as security vs. ease-of-use, circles of trust, and mundane portability headaches combined with problems like questioning what is sent in the clear and what subsystems should be guarded with one-time passwords. NRL has been using some form of one-time password (originally Bellcore's S/Key, followed by transient steps to what is now NRL OPIE) for about a year. The experience gained from use not only within NRL, but also elsewhere, has helped shape a better system.

2 What OPIE is Not

OPIE only defends against one specific type of attack, passively listening for passwords. By itself, OPIE does not provide a single login for a whole set of machines, nor can it authenticate services. Alternatives like Kerberos[SNS88] authenticate more than just user identity on a single machine.[2] OPIE is not secure against certain kinds of active attack, such as dictionary attacks. OPIE has not yet been specified using rigorous formal methods. The current OPIE implementation is also not formally verified, though it was developed using good software engineering methods.

OPIE should be used in combination with other protective measures for maximum effectiveness. The operating system itself still needs to be secure against legitimate users improperly gaining privileges or improper access to privileged parts of the system. TCP Wrapper software can and should be used to provide coarse-grained access controls to the Internet services provided by computer systems [Ven92]. Routers within the same administrative domain should be configured to filter out source-routed or obviously forged IP packets [CER95]. Stronger authentication of distributed services, provided by Kerberos or commercial products such as Sun's NIS+™ can also be important in risk reduction. Encryption of IP packets or of telnet, ftp or rlogin sessions might be desirable in some environments where the confidentiality is considered to be worth the consequent performance loss.

[1] NRL OPIE version 1 was sometimes colloquially called NRL S/Key.

[2] Kerberos domains that allow entry from non-Kerberos systems should use one-time passwords or other techniques to protect incoming logins originating on non-Kerberos systems.

3 Specific Improvements

The fundamental concepts behind S/Key have not changed in OPIE. A challenge still contains a sequence number and a public seed. A reply is computed locally, and only the computed reply is sent in the clear. The remote machine does not store any keys. The details of S/Key (and OPIE) fundamentals are left to Haller's S/Key paper and the recently released RFC on S/Key[Hal95].

3.1 Functional Improvements

Several improvements to the function of the S/Key authentication system were made early on in the development process. The most obvious is the replacement of the default cryptographic checksum used in S/Key, MD4 [Riv92a], with MD5 [Riv92b]. MD5 is believed to be cryptographically stronger, and is definitely slower, than MD4. These two properties decrease the feasibility of reverse-engineering or defeating the one-way function. This increases the assurance that a system running OPIE will not be compromised via brute force or cryptanalysis.

Another functional improvement is the restructuring of the OPIE challenge. Under OPIE, a challenge looks like:

```
[opie-md5 99 wi12351]
```

The entire challenge string is surrounded by square brackets. These can serve as an indicator to a local terminal emulator, or a co-resident challenge detector (e.g. an MS-DOS TSR) that an OPIE password is expected. The first string inside the brackets, `opie-md5` indicates that one-time passwords are required and which algorithm is in use. The complete first string can be used by challenge detectors to select the correct algorithm. It is also the correct command-line syntax to invoke the one-time password calculator.[3] The last two strings are the familiar S/Key-like sequence number and public seed.

[3] Of course, only a LOCAL machine should be used to generate one-time passwords. That issue is discussed later.

A source of password disclosure overlooked in the original S/Key distribution was FTP sessions. To address this, OPIE introduces an FTP daemon which is a direct modification of the 4.3BSD Net/2 release *ftpd(8)* program. We chose not to use the more popular WUarchive ftp daemon because its additional features and consequent code complexity made it harder to determine if other unknown vulnerabilites were present [CER94a][CER94c][CER93a]. The current implementation merely adds the square-bracketed challenge into the normal FTP password response, *331* [PR85]. The current implementation also accepts the reply using the standard PASS command. Most existing FTP clients work without change.

3.2 User Interface Improvements

Along with improvements in function, OPIE sports improvements in the way users interact with the OPIE software. Several are simple, and merely bring OPIE binaries up to the level of other UNIX packages. For instance, every program related to the OPIE software distribution starts with `opie` to clearly distinguish them from other programs. Also, every program has command-line flags to show the software version number and a quick usage summary, much like the Free Software Foundation's suite of tools.

The default configuration of the OPIE key calculator on UNIX asks the user to retype the secret password to help prevent typing errors. As a compromise with the people who are either used to older calculators, or who just cannot stand typing the same thing twice, a user can also just press return when prompted again for the password. This provides added protection to those who would like it without adding significant burden to those who do not. The double-password prompt can be removed as a compile-time option.

The original S/Key password initialization program *keyinit(8)* has undergone a facelift to become *opiepasswd(1)*. The name change brings it more in line with its UNIX counterpart *passwd(1)*, which should make both programs easier to remember for users. This program has also been modified to operate, by default, in a mode where the password change is done using a one-time password and a local calculator instead of cleartext passwords; the opposite was the old default. This new default was chosen to reduce the risk of disclosing a secret password over the network. *Opiepasswd(1)* now also generates an inital default sequence number and a seed without prompting the user. This should prevent confusion that could result in the re-use of a seed, as well as giving the novice user one less thing to think about. More experienced users can now specify both their seed and starting sequence number from the command line, allowing more flexibility. A shell script that behaves more like S/Key's *keyinit(1)* is included as a transition aid. The *opiepasswd(1)* command employs simple checks, such as examining the `DISPLAY` environment variable used by X11, to try to reduce the likelihood of accidental misuse of the command in "plaintext-mode" on a remote system. It is, however, difficult to prevent all forms of deliberate misuse.

The original Bellcore S/Key software supplied an S/Key-enhanced *keysu(1)*, but permitted one to *su(1)* without using the one-time password scheme. This is unwise in many environments because it means that anyone who could eavesdrop on the net could become root if they could get on the system (e.g. a legitimate user who was not authorised root privileges). Hence, OPIE's user switching program, *opiesu(1)*, always asks for a one-time password. This is an annoyance to users who are truly on the console. Given the difficulty of determining whether a particular tty or pty is trustworthy, security once again prevailed over convenience. In practice, the inconvenience is not that great because most users have a windowing system and can use "cut and paste" between the window that *opiesu(1)* is in and another window where the key generator is executed.

As mentioned previously, the OPIE challenge itself can form a complete command in a trustworthy command-line environment. An example of this is demonstrated in Figure 1. Under trusted conditions, logging in with OPIE differs from logging in with cleartext passwords only by inserting two copies-and-pastes, one before typing in the secret password, one after.

3.3 Other Improvements

One of the most serious deployment problems with the S/Key software was that it was very 4.3BSD-centric. This made it widespread installation and use difficult in heterogeneous computing environments. All of the 4.3BSD system dependencies have been isolated and protected by suitable **`#ifdefs`**. Many of the 4.3 dependencies involved *ioctl(2)* calls that were easily replaced with highly portable POSIX-compliant *termios(4)* calls. Other dependencies, such as the *utmp* and *wtmp* logging schemes have been rewritten in a very portable manner. Behaviors and features unique to particular dialects of UNIX have been isolated into compile-time op-

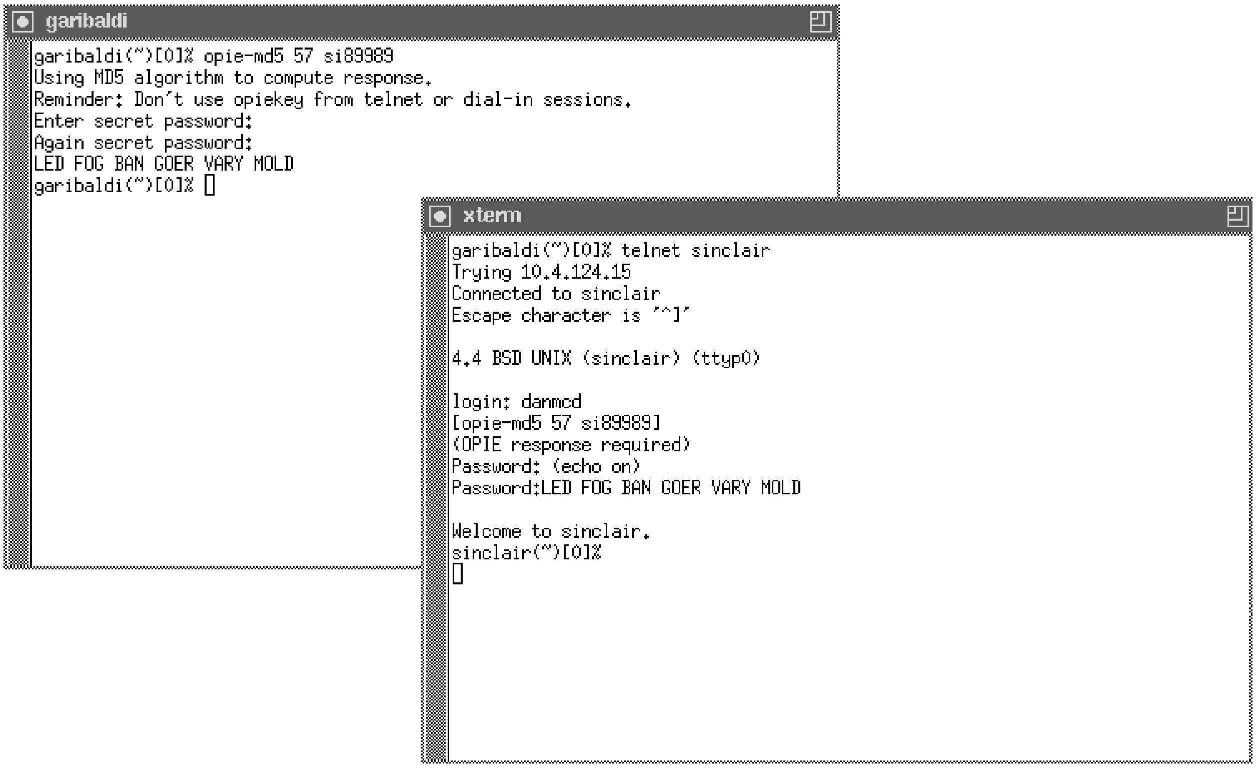

Figure 1: Using an OPIE challenge as a command.

tions. The software now works on most dialects of the UNIX operating system. The software also implements and fully supports many vendor extensions to the system programs replaced by OPIE counterparts. Furthermore, the OPIE source package is complete; no source licenses are needed for platforms which do not include source. Unfortunately, the software no longer supports a few older 4.3BSD systems that are not POSIX conformant. We believe that this was the right tradeoff because it will permit more systems to be protected with one-time passwords.

Programmers can now add support for OPIE authentication to their programs more easily. All of the OPIE routines that are available to client programs are isolated in one library, `libopie.a`, and all start with the prefix `opie` to prevent namespace conflicts. Information about limits, such as the size of a secret password, is explicitly provided along with other useful constants in a header file and every preprocessor symbol starts with the prefix `OPIE` to prevent namespace conflicts.

4 Design Decisions

During the transition from S/Key to OPIE, several design issues surfaced. Most of these issues received the same treatment in OPIE as they did in S/Key. With the addition of new features, new design issues also came up. The OPIE approach to most design decisions was to err on the side of increased security, even sometimes at the cost of usability.

4.1 Direct replacement of `/bin/login`

The OPIE design requires replacement of the *login(1)* program. Another approach to implementing one-time passwords at login time involves not directly replacing `/bin/login`. With the latter approach, a second level of authentication is introduced by invoking the second-level authenticator as a login shell after normal login with disclosing cleartext passwords succeeds. This second-level authenticator then invokes the normal user login sequence if the user passes. In these schemes, one-time passwords usually occur in the second-level authenticator.

The advantage of not having to replace an often system-dependent `/bin/login` is obvious, and some sites have implemented this two-level authentication scheme. On the other hand, two pieces of code have more potential for vulerabilities than one piece of code. Also, this practice does disclose one of the user's passwords and this might be considered to increase the security risk as compared with always using only the one-time passwords. Furthermore, it is simpler for the user to perform only one task with a replacement `/bin/login`. Hence, we believe replacing `/bin/login` is a better approach.

4.2 Security vs. Ease-of-Use

Except for the direct replacement of the original `/bin/login` with an OPIE login program, OPIE tends to impede users in the name of greater security. As mentioned in the **Improvements** section, many of the OPIE binaries default to more paranoid behavior. The *opiesu(1)* command will only accept one-time passwords because of the difficulty determining the trustworthiness of a tty. The *login(1)* command always forces the user to use a one-time password except when executed with a saved uid of root without the `-h` *hostname* flag, or when specifically used on `/dev/console`. These two restrictions eliminate a potential risk with using *login(1)* to switch user identities with cleartext passwords, but are not a general solution.

In S/Key, there is a host equivalence file which lists trusted remote machines that can log into a machine using ordinary disclosing UNIX passwords rather than S/Key one-time passwords. We have made this into a compile-time option. The default configuration in the NRL OPIE distribution does not enable this capability because we believe that it is generally an unacceptable security risk. However, some user communities choose by policy to balance security and convenience more in the direction of convenience or have a different threat environment, so we did not want to entirely remove the capability.

The *opiepasswd(1)* command defaults to a mode of operation where what is entered for the new OPIE password is an actual six-English-word response. This makes the default operation of *opiepasswd(1)* safe for use over the network, but assumes that the user has a secure one-time password calculator. If a user is sitting at a secure terminal (such as the console), however, there is a flag to override the default behavior.

4.3 Internals

OPIE contains a deliberate effort to avoid internal coding practices that may make programs vulnerable. The Internet Worm of 1988 [Rey89] exploited a string bounds overrun bug caused by use of *gets(3)* in *fingerd(8)*. OPIE generally uses numerically bounded string manipulations, such as *strncmp(3)* and *strncpy(3)* rather than *strcmp(3)* and *strcpy(3)* to reduce the risk of such subtle security problems.

5 Adding OPIE Authentication to Services and Clients

One-time Passwords In Everything should be more than a contrived acryonym. It should be a philosophy for hosts that want to foil password sniffing attacks. This section discusses how to add OPIE authentication to both programs which allow access (servers), and programs which take advantage of access (clients).

5.1 Functions Needed for OPIE Authentication

The OPIE library, `libopie.a`, offers two families of functions. The functions `opiechallenge()`, `opieaccessfile()`, and `opieverify()` are for authenticating users. The functions `opiekeycrunch()` and `opiehash()` are for generating one-time passwords, and `opiebtoe()` is for transmitting one-time passwords in readable form.

```
int opiechallenge(struct opie *mp, char
*name, char *cstring)
```

The first parameter references storage for a stateful OPIE server-side structure, which contains current OPIE login information for the user specified with `name`. `Opiechallenge()` initializes the storage referenced by `*mp`. The third parameter should point to enough memory to store an OPIE challenge string of the form, "`[opie-`alg. number seed`]`", which `opiechallenge()` writes out. 0 is returned if the lookup of a name is successful. A return value of 1 indicates a problem opening the OPIE password file. -1 is returned if the lookup is unsuccessful. If -1 is returned, a random challenge will be issued, so that a potential cracker is at least initally confused.

```
int opieaccessfile(char *hostname)
```

`Opieaccessfile()` looks in the host equivalence file, if enabled, and sees if the the host name is in this file. If so, `opieaccessfile()` returns 1, otherwise, it returns 0. If hostname points to an empty string (`""`), 0 is also returned. This function always returns 0 if support for host equivalence is disabled, which is the default.

`int opieverify(struct opie *mp, char *response)`

After `opiechallenge()` returns successfully, and a response has been issued to the server, `opieverify()` verifies the response by using the information in the OPIE server-side structure. The return values are 0 if successful, -1 if non-authentication errors occurs, and 1 if the authentication fails. The data inside the OPIE server-side structure is rendered invalid after this call, regardless of return value. Invalidating the server-side structure contents forces a call to `opiechallenge()` before a call to `opieverify()`, and also indicates that the OPIE response file has been updated.

`int opiekeycrunch(unsigned algorithm, char *result, char *seed, char *passwd)`

`int opiehash(char *x, unsigned algorithm)`

The one-time password schemes implemented in OPIE, as first described in [Hal94], compute a cryptographic checksum over a secret password and a public seed. The secret password and seed, along with the algorithm identifier, are the fourth, third, and first parameters of `opiekeycrunch()` respectively. The results of this, folded into an 8-byte result, are stored where the second parameter references. This result is passed into the `opiehash()` function, where the cryptographic checksum is computed over the result referred by x, and stored in the same location. Lamport's one-way function *F(x)* is what `opiehash()` implements. The `algorithm` parameter to both of these functions isolates algorithm dependencies so that new algorithms can be added simply by modifying these functions.

`char *opiebtoe(char *bytes,char *engout)`

A server accepts an OPIE response as six English words. The `opieverify()` routine performs a conversion into an 8-byte quantity internally. If a client computes one-time passwords internally, it needs to take the result from `opiehash()` and convert it into six English words. `Opiebtoe()` does that, with the 8-byte quantity referenced by `bytes`, storing the result in `engout`, and returning a pointer to the result.

5.2 Example Server Code

Assuming a server is well-modularized, and has a way of issuing an OPIE challenge, it is relatively painless to insert code to add OPIE authentication to that service. Two places in a server need to be modified. The first place is after user identification is given. The server would then call `opiechallenge()` and optionally `opieaccessfile()`.

```
...

/* I have determined the user's name.  */

if opiechallenge( &cookie, name, challenge)
                 != 0
   if Access file allowed &&
           opieaccessfile(hostname)
      Allow cleartext password.

/* I have a challenge, either actual, or
   random.  I also know if I can allow
   cleartext passwords or not.  */
...
```

The second place would be after the password, or OPIE response has been issued.

```
...

/* I have a valid user name, and a response
   from that user.  */

if opieverify(&cookie, response)
   Allow entry
else if Cleartext is allowed && Cleartext is good
   Also allow entry
else Deny entry
...
```

5.3 Example Client Code

It is possible for programs that interact with authentication-granting services to compute OPIE responses within the program itself. These programs can send back the response to the server while hiding some or all of the details from the user. The convienence gained from not having to consult an out-of-the-way calculator can both save time and reduce frustration.

Unfortunately, adding OPIE calculation to a client program can cause the very problem OPIE tries to foil. If an OPIE-generating Telnet client is run on a local machine, there is no problem. The intelligent Telnet client parses out the OPIE challenge, the user types in his or her secret password, and the local Telnet client sends the response to the remote /bin/login program. If this intelligent Telnet client is running on a remote machine, the remote Telnet detects an OPIE challenge, and asks for the user's secret password. Since the Telnet is running on a remote machine, the secret password is sent *in the clear*. Any client program that adds built-in one-time password generation should allow the one-time password itself to be entered by the user, or if possible, check if the program is running locally or not. These same safety tips apply to OPIE calculators themselves. Future work will try to develop higher assurance methods of determining whether an executable (e.g. Telnet client running in an Xterm) is local or remote so that we can improve ease-of-use without increasing risk of disclosing the secret password.

Modifications to OPIE-aware clients need only be made in one place, and that is immediately after a challenge is issued by the OPIE-guarded server.

```
...

/* The server has issued a password request,
   containing an OPIE challenge.  */

/* These next two may have support in
   libopie.a someday.  */
Parse out string between [ and ].
Determine algorithm, sequence, and seed from
[opie-algorithm sequence seed].

printf("One-time password requested.  ");
if Input stream not sniffable
   printf ("Please calculate locally and
           enter OPIE reply.");
   Obtain words from user.
else
   opiekeycrunch( algorithm, result, seed,
                  passwd);
   while (sequence-- != 0)
         opiehash(result, algorithm);
   opiebtoe(result,words);
Send words.

...
```

6 Deployment

Every machine that has one-time passwords is one less machine that can be broken into with a network-based passive attack. Sometimes, however, not every machine can run the one-time password software. Several obstacles hinder large-scale deployment, but even small uses of OPIE can significantly reduce the risk of penetration from passive attacks.

6.1 Example Deployment - A Small Cluster

Consider a small cluster of machines that allow a central machine to have privileged access via *rsh(1)* for the purpose of triggering backups. The central machine, `sinclair`, must also allow privileged access from every machine, because it has the tape drive every machine writes to. Notwithstanding other forms of attack, if any of these machines is not protected by one-time passwords, then the other machines can be compromised because of the trust allowed by the tape backup scheme. Figure 2 helps illustrate the circle of trust, and how the circle is only as secure as its weakest machine.

All of the machines except for `londo` have OPIE installed. If `londo` is compromised, a quick scan of the `/etc/hosts.equiv` file will reveal equivalence with `sinclair`. Using *rsh(1)*, an intruder can access `sinclair`, whose equivalence file contains all of the other machines. All of the machines on the network can now be compromised at the whim of the intruder.

This phenomenon is not only restricted to one machine not having OPIE, or another form of one-time passwords. If a machine in a circle of trust is missing any security precaution that the others have, that vulnerability can be exploited as shown in Figure 2. The solution to this problem is to either not allow trust at all, which means the convienience of this backup scheme is lost, or ensure that all machines in the circle take identical security precautions.

6.2 Example Deployment - Firewall

Another popular place to deploy one-time passwords is in a firewall gateway. Some packages [AR94] include one-time password software for this purpose. Figure 3 shows how a one-time password is installed on a firewall gateway machine. Users from outside the protected domain first log into the firewall with one-time passwords, then use normal cleartext passwords from the firewall to reach machines inside the domain.

The theory behind this approach is that even if the internal machines' passwords are sniffed, they will not be usable because the firewall will prevent unauthorized access. If the firewall is compromised, however, any sniffed passwords immmediately become useful as the intruder starts to play with the compro-

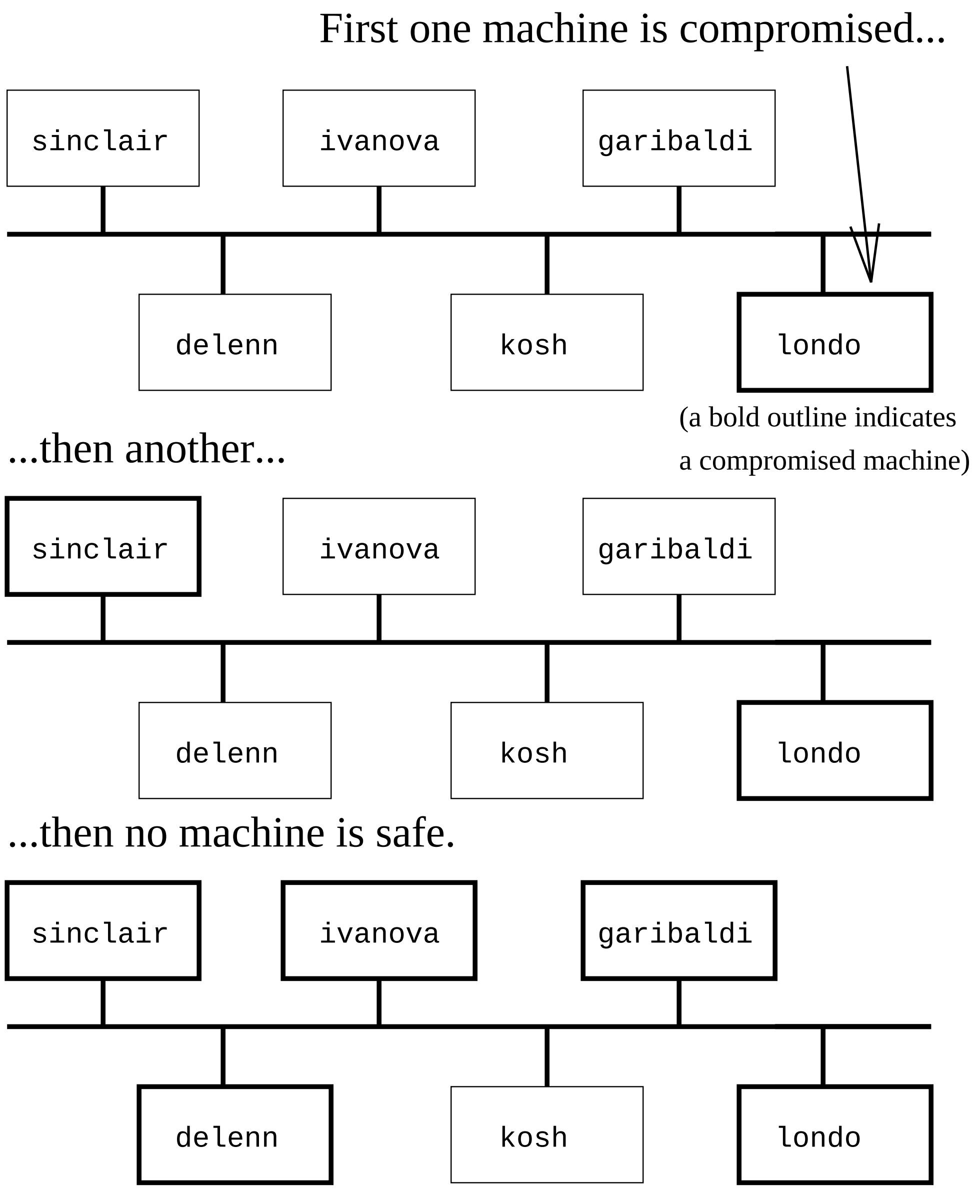

Figure 2: Breaking the circle of trust in a small cluster.

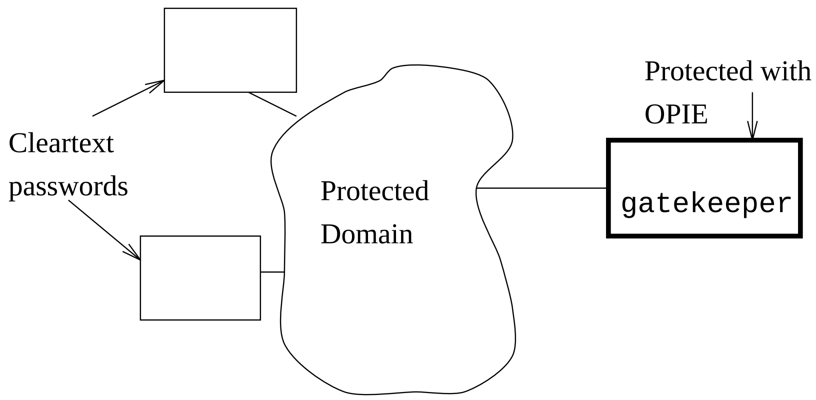

Figure 3: Firewall gateway with one-time passwords.

mised domain. There are known cases of corporate firewalls being breached and havoc subsequently being wreaked on the internal systems. In part this is because some sites place all of their trust in the security provided by the firewall. Defense in depth by implementing appropriate security precautions even on the internal machines is a wiser strategy.

6.3 Barriers to Deployment

Experience has shown three main issues with widespread OPIE deployment. The first, hinted at earlier, is introducing to users yet another level of complexity to do a simple login. As with most security, OPIE makes life somewhat less convenient for users. This issue is compounded by some client programs which make it impossible to either obtain an OPIE challenge, or give an OPIE response (which can be as large as 29 characters). Even if client programs did not prevent OPIE from working, some users might have to change which client program they use, which is often a painful exercise.

The second problem is scaling. Currently, OPIE has no good method for securely sharing its file of next-challenges across a cluster of machines. This implies that every machine has to have an individual password initialization. For a small cluster of machines, this is not a problem. For a campuswide system of workstations, individual password initialization is intractable. A combination of Kerberos for use among the campus machines and OPIE for access from outside the campus machines might be a good choice in such situations. Alternately, NIS+ using DES authentication could be used to share OPIE challenges.

The third problem is that OPIE requires users to always have a local computer system available for them to generate one-time passwords on or to have planned ahead and generated printed one-time passwords to carry around with them. This last issue is not a technical problem but an economic one. It is straightforward to build hand-held, battery-powered S/Key-compatible one-time password generators. It is not immediately clear whether there is a good business case for building such a product.

7 Future Work

One of the issues in any useful application that generates one-time passwords, either a mere calculator or in an intelligent client, is the determination of how trustworthy an environment is for entering one's secret password. Besides the low-level determination of a tty's trustworthiness, other potential holes in applications [CER93b] and environments may reveal secret password keystrokes to crackers. This problem extends beyond the scope of work with OPIE or other one-time password systems, but any solution will increase the effective use of OPIE.

While an improvement of Bellcore S/Key 1.0, OPIE can be improved further. Many of the difficulties mentioned earlier can be better addressed in future work. Integration of OPIE or other one-time password schemes into programs like terminal emulators and FTP clients needs to be done. Some have proposed Telnet or FTP options to hide the details of one-time passwords and make it easier for client programs to work well with either OPIE or conventional UNIX passwords.

If the scaling problem is resolved, OPIE can be deployed with greater ease on large campuses. As noted earlier, a common OPIE key file can be shared securely by using Sun's Network Information Service (NIS+) or the TIS Firewall Toolkit's Authentication Server. Another possible approach is to keep the key file on some central fileserver, and use secure RPC protocols to avoid tampering.

Another area of potential improvement is in the method of calculating keys themselves. Currently users have two choices, either MD4 or MD5 checksums over the public seed and secret password, followed by continued MD4 or MD5 checksums over a number of iterations. Marcus Ranum suggested a method where the initial checksum (i.e. `opiekeycrunch()`) be modified to use the secret password to unencrypt a random file using DES, and add that randomly decrypted file to what is initally checksummed. Subsequent iterations work as before. This approach defeats dictionary brute-force attacks, but requires that an auxilliary file be stored with the one-time password calculator. This last approach can also be enabled as a compile-time option in OPIE. Also, any new and stronger one-way functions will strengthen OPIE. Support for NIST's Secure Hash Algorithm (SHA) will be added in a future release of OPIE.

8 Summary

One-time Passwords In Everything should be a rule for machines that wish to defeat password sniffing attacks. The NRL OPIE distribution has improved upon earlier work in one-time passwords, as well as bringing it to more platforms. Experience with our software has pointed out better ways of doing things, as well as what still needs to be done. OPIE, while not a complete security solution, precludes a widely used class of attacks on networked computer systems.

9 Availablility

NRL OPIE version 1 is available now in the directory `ftp://ftp.nrl.navy.mil/pub/security/nrl-opie/`. NRL OPIE version 2 will be available at the same directory soon.

10 Acknowledgments

We would like to thank Mike Harrison and Tim McChesney of the Information Security Program Office of the US Space and Naval Warfare Systems Command for sponsoring this work. Neil Haller has had a strong influence on all of this work, not only in his efforts with Bellcore S/Key that our work is derived from, but also in ongoing discussions about open issues, possible approaches, and future directions for S/Key-compatible one-time password systems. We would also like to thank two others who have been particularly helpful, Marcus Ranum, for his work to improve resistance to dictionary attacks, and Marshall Rose, for showing us that putting basic support for one-time password generation into client software is not difficult.

References

[AR94] Frederick Avolio and Marcus Ranum. A Network Perimeter with Secure External Access. In *Proceedings of the Symposium on Network & Distributed Systems Security*. Internet Society, February 1994.

[CER93a] WUarchive ftpd vulnerability. Computer Emergency Response Team, April 1993. CA-93:06.

[CER93b] `xterm` Logging Vulnerability. Computer Emergency Response Team, April 1993. CA-93:17.

[CER94a] ftpd Vulnerabilities. Computer Emergency Response Team, April 1994. CA-94:08.

[CER94b] Ongoing Network Monitoring Attacks. Computer Emergency Response Team, February 1994. CA:94:01.

[CER94c] WUarchive ftpd Trojan Horse. Computer Emergency Response Team, April 1994. CA-94:07.

[CER95] IP Spoofing Attacks and Hijacked Terminal Connections. Computer Emergency Response Team, January 1995. CA-95:01.

[HA94] Neil Haller and Randall Atkinson. On Internet Authentication, October 1994. RFC-1704.

[Hal94] Neil M. Haller. The S/Key One-Time Password System. In *Proceedings of the Symposium on Network & Distributed Systems Security*, San Diego, CA, February 1994. Internet Society.

[Hal95] Neil Haller. The S/KEY One-Time Password System, February 1995. RFC-1760.

[Lam81] Leslie Lamport. Password Authentication with Insecure Communication. *Communications of the ACM*, 24(11):770–772, November 1981.

[Lot92] Mark Lottor. Internet Growth (1981-1991), January 1992. RFC-1296.

[PR85] Jon Postel and Joyce K. Reynolds. File Transfer Protocol, October 1985. RFC-959.

[Rey89] Joyce K. Reynolds. The Helminthiasis of the Internet, December 1989. RFC-1135.

[Riv92a] Ronald L. Rivest. The MD4 Message-Digest Algorithm, April 1992. RFC-1320.

[Riv92b] Ronald L. Rivest. The MD5 Message-Digest Algorithm, April 1992. RFC-1321.

[SNS88] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Proceedings of the Winter Usenix Conference*, Dallas, TX, 1988. USENIX Association.

[Ven92] Wietse Venema. TCP WRAPPER: Network monitoring, access control and booby traps. In *Proceedings of the Third Usenix UNIX Security Symposium*. USENIX Association, September 1992.