Apparent Weaknesses in the Security Dynamics Client/Server Protocol

Preliminary Version

Adam Shostack
adam@homeport.org
October 1996

Abstract. The protocol used by Security Dynamics has substantial flaws which appear to be exploitable and reduce the security of a system using Security Dynamics software to that of a username and password. In this paper, we explain the protocol in some detail, show the outline of how the attack would work, and offer some defenses against the attack. The attack has not yet been implemented. There do not seem to be easy defenses available within the current protocol without some form of firewall. We also suggest some additional areas for future research.

I. Introduction

The SecurID system is a popular form of two factor authentication that involves a small handheld card distributed to users, client software for a variety of systems and server software for centralized authentication and management.

The card contains an 8 bit microprocessor, a clock, an LCD, a battery and possibly a pressure sensitive keypad. The unit is housed in a tamper resistant steel case, and is designed to erase its memory if opened. The card generates a stream of apparently unrelated numbers of fixed length (referred to as cardcodes) by combining a factory programmed seed with the time, and running those through what is commonly thought of as 'the' 'SecurID hash.' Security Dynamics actually has two hashes, the card hash and a second hash called F2. See below for some details of F2. The length of the cardcode can be from 6-8 digits, and does not vary for a given card. If the card has a keypad, the card can combine a pin (of 4-8 digits, configurable by the administrator) with the output of the hash. The pin is added to the output of the hash by no carry addition. If the card does not have a keypad, the user is expected to prepend his pin to the cardcode when sending it to be authenticated. Security Dynamics has announced a software token, which is not considered here.

The client software consists of a modification to a host's authentication system so that it can communicate with an ACE/Server. Software exists for a wide variety of systems, from AppleTalk Remote Access to Cisco routers, Annex Terminal servers, and UNIX workstations. We will be focusing on the case of a UNIX workstation running Security Dynamics' sdshell program. That program is assigned to be the user's shell. Once the user has authenticated to /bin/login, they are passed to the sdshell, which communicates with the ACE/Server to obtain authentication information for the user. Although the attacks described here focus on the sdshell implementation, it is likely that they affect all implementations of the protocol. Version 1.2.4 of the sdshell is 49k. Later versions are reputed to have grown to as much as 80k for version 2.2. (All sparc binaries running on SunOS 4.1.4). There is also a client API which allows SecurIDs to be integrated into a new product on a supported platform without much involvement from Security Dynamics.

The server is comprised of a daemon, a database, and software to administer both. The server can be run on a variety of platforms, and has support for running over TCP/IP, Novell 3.11 and 3.12, and AppleTalk. Authentication can also be done via TACACS, XTACACS, and TACACS+. We focus on the TCP/IP implementation which is likely to be in use in most internet connected networks.

The protocol is stateless, and runs over UDP. The daemon listens on UDP port 124 by default. This has been changed in version 2. The database in version 1.2.4 was a proprietary flat database, and was replaced in version 2 by a Progress DBMS. The database and log files are stored DES encrypted, and can only be manipulated by the sdadmin program, or read by sdlogmon, respectively. The database stores information for each token: its planned death, the last successful login (in theory), the last attempt, the pin, an estimate of the clock drift, and the card's seed. It also stores information about groups of users, and groups of machines, and which groups or users are allowed to login to each machine or group of machines. The system supports two servers, a master and a slave. We do not examine their interaction, and consider only the case of a single master, because the nature of the attack described here is not affected by the master-slave connection.

II. The Ace protocol

This paper examines the protocol as implemented in version 1.2.4 of the software. The purpose of the Ace protocol is to allow the token seed database and the card algorithm to remain on a single central machine which can then be well defended. The protocol, it is claimed, offers 'four successive levels of overlapping encryption' and 'mutually authenticates' the client and server (FAQ, 4.13). This claim of mutual authentication will be shown to be false.

The following terminology will be used through the textual and graphical (Figure 1) description of the protocol:

c is a DES key previously generated by the client and sent to the server.

E_k(plaintext) refers to the DES encryption of the plaintext with key K. The protocol does not use an IV, and only encrypts single blocks in ECB mode.

IP is the IP address of the Ace client.[1]

[1] Note that the IP being tied to the host caused dual homing (more than one network card in a ma to fail to work with Version 1 of the software, since machines were indexed by IP; no hostna transmitted, and if two routes were available, the wrong one might be chosen. This may al DHCP to fail, if the aceserver is conservatively configured to not use the DNS. If machine sent instead, then this would not be an issue.

P is the passcode that the user would like to have authenticated.

λ is the passcode a particular token + pin is expected to produce

λ+1, λ-1 are the expected passcodes if the card's clock has drifted one quantum forward or back.

T is a timestamp

F2 is a hash developed by Security Dynamics. For the protocol to work as designed, it needs to be infeasible to invert (compute its inputs from its output) and be collision resistant (infeasible to find two inputs that produce the same output). (See (Schneier) chapter 18 for detailed explanation) If it could be easily inverted, then a card's seed could be determined by an attacker. It needs to be collision resistant because the server claims to not accept previously used tokencodes (FAQ 4.12). F2 produces 256 bits of output from its input. F2 can be derived from examining the "sdshell" program, which is available to the computer underground.

WP refers to a 'Workstation Passcode,' which is 64 bits of the 256 bits of output of F2. WP[1] refers to the first 64 bits.

The protocol begins when a user causes some server process (login, in.rshd, etc) to spawn their shell.

1. sdshell signals the server with a hello message (Figure 2).
2. aceserver responds with a timestamp, T (Figure 3).
3. sdshell prompts the user 'Enter Passcode:'
4. The user sends back the appropriate current passcode, P
5. sdshell calculates F2(IP, T, P), and splits it into wp[1-4]
6. sdshell sends a UDP packet with (username, E_c (wp[1])). (Figure 4)
7. aceserver decrypts the packet, and calculates F2(IP, T, λ)

If the wp from that calculation matches what was sent by sdshell, then the assumption is made that the user has the card and pin, and should be allowed in. If not, aceserver calculates F2(IP, T, λ+1) and F2(IP, T, λ-1) and compares them to the value received.

8. If a match was successful, then an authorization message is sent to the sdshell: $E_{wp[2]}$(authorization), shell to run.

If a match is not found, aceserver looks for a match with λ-10— λ+10. It can send a message requesting a second tokencode, which allows it to ensure that its estimate of the clock is correct. It can also send back a failure message, encrypted with wp[1] as the key (FAQ 4.14).

The Packets:

These packet formats are not completely known at this time, but what we have found demonstrates some of the weak, non-cryptographic authentication taking place. Each packet is shown as the body of the udp packet, followed by an analysis of the packet. We do not show the authorization packet format.

Each packet contains several magic numbers and place holders, as well as some constants which are switched around according to set rules. There are also at least two sequence numbers in each packet.

Sequence A increments 1 for each packet client sends in a session.
Sequence B decrements 1 per packet
Sequence C increments by 5 each message
Sequence D decrements by 5 each message

A B C D and F are bytes that are sent back and forth with some transposition. E changes from session to session.
The username, unencrypted is expressed in hex. (adam=61 64 61 6d)
Underlined bytes never seem to change within a packet type

```
         8         16        24       32
+----+----+----+----+----+----+----+----+
 4500 0098 1633 0000 3c11 517f cd88 41c6
 c034 4720 09a0 02f3 0084 0000 6702 0010
 0000 0000 0000 0000 0000 0000 0000 0000
(208 more 0 bytes)

         8         16        24       32
+----+----+----+----+----+----+----+----+
 Securid   |SeqA|0000|mrkr|seqB| 'A'  'B'
 'C'  'D' | 'E' 'F' |0084 0000|6702 0010
```

Figure 2 (Hello)

```
         8         16        24        32
+----+----+----+----+----+----+----+----+
 4500 0098 c9b2 0000 3111 a8ff c034 4720
 cd88 41c6 02f3 09a0 0084 0000 6c02 0010
 4a58 8000 0000 000a 3273 c5c6 003a 5bfa
 c92d 5cc2 b52d cdcd df39 0ce8 0cfa 0b54
 4b55 55dd 6bd9 e5fa 7931 7ee8 c72a 77a4
 3c16 d2f8 897b 5fc8 dcc4 0cc4 ae86 a1f4
 7279 e41d 66b3 100b a577 2c47 3762 d7d9
 ab50 36d8 56b3 55aa 0599 d30a 6be8 6191
 3349 3bff eafb 3a83 84e8 955e 4576 9071
 bc96 df03 07f4 7605

         8         16        24        32
+----+----+----+----+----+----+----+----+
 Securid   |SeqC|0000|mrkr|seqD| 'C'  'D'
  'A'  'B' | 'F' 'E' |0084 0000|6702 0010
 4a58 8000 0000 000a 32| Encoded time
 roughly decodes to October 27 1996
 15.25 EST
```

Figure 3, Time

```
        8         16        24        32
+----+----+----+----+----+----+----+----+
4500 0098 1634 0000 3c11 517e cd88 41c6
c034 4720 09a0 02f3 0084 0000 6502 0010
0000 0000 6164 616d 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0001 2721 2e54 99da b278
561a 80cb a483 c1ee 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000

+----+----+----+----+----+----+----+----+
 Securid   |SeqA|0000|mrkr|seqB| 'A'  'B'
  'C'  'D' | 'E' 'F' |0084 0000|6702 0010
 0000 0000|username |0000 0000 0000 0000
 '0' characters to fill out to byte 201
 0000 0000 0000 0001| DES encrypted wp[1]
 DES (continued)    |0000 0000 0000 0000
```

Figure 4, Passcode verification request

III. Attacking the Protocol

There appears to be essentially no private information in the key used in step 8. The key used, wp[2], is derived from three chunks of information: The IP address, the timestamp, and the Passcode. The IP address is known to anyone who can connect to the machine to break in. The timestamp is slightly more difficult, but an Ace/Server will send it to any machine that can send it UDP packets. If you can not get the timestamp, the time can be easily guessed, especially if the site is using NTP or some other time setting protocol which causes them to have accurate clocks. The Passcode can be chosen by the attacker.

Thus, the key might have up to 2^6 bits of information if time were granular to the second, but time appears to be granular to the minute. Since time packets are sent to anyone who requests them, the key has essentially no entropy.

1. Telnet to the target machine
2. Enter a sniffed username and password.
3. Send a time request to the Ace/Server
4. Compute wp[1] for F2(IP(target), T, 123456)
5. Find port sdshell will listen on
6. Send passcode of 123456
7. Package and send $DES_{wp[1]}$(Allow access) to sdshell on expected port (See Figure 5)

The actual statefulness of the protocol is an open question. If there substantial state in the protocol, and if it is checked, then the attack may only be practical if the attacker is on the local network already, because sniffing certain packets may be needed to find the appropriate values

Thus, any attacker with UDP access to the Ace/Client can send it a UDP packet that convinces sdshell that they are legitimate, and should be allowed in. (The attacker also needs the format of the authorization packet, the F2 hash, and a username and password.)

Timing the attack is quite easy. In response to a variety of sniff and race attacks, SDTI has added a delay in all authorization responses. The attacks being defended against involve watching a legitimate user logging in, and mirroring their actions. At the very end of the login sequence, the attacker either sends in the "enter" character faster, or sends in all of the possible last digits to complete the authentication code. (This has a roughly 30% chance of getting in, and are documented in (peiterz)) In response to these attacks, aceserver will delay from 1-15 seconds (administrator configurable) before responding to a request, and if it sees multiple requests in that time, will reject them all. However, this delay acts as a window of opportunity to forge the authorization packet. (It also opens an obvious denial of service attack.)

There is a practical difficulty in determining what port the sdshell will bind to. We expect that this port can be found fairly easily. If the last security offered by the protocol is in the difficulty of finding the port, then the system can fairly be termed security through obscurity. This is especially true if the attacker is local, and wants to impersonate another user. If the user is local, they need only to run the command "netstat -a" to find sdshell listening. If they are remote, an extended probe may be needed to find the port range to attack.

IV. Defending Your Site

We consider two groups of attackers, outsiders and insiders. The attacks by outsiders are easy to defeat, attacks by insiders are not.

To defend yourself against the attack being carried out from the outside, block all incoming udp packets, except those that you need, such as DNS. This is a good policy anyway, and many sites that do this already will not need to take any action, except to verify that things work as designed.

Defending against insiders is not so easy. You lose the ability to use SecurID if you block udp packets. We urge those sites who use SecurID for internal authentication to require Security Dynamics to provide protocol fixes that have been subjected to peer review. Fixing this problem will require an update for each sdshell deployed. Installing anti-sniffing hubs, or deploying IPsecurity may provide sufficient defenses. However, none of these can be done quickly or cheaply. Note that session encryption provides no defense, since this is an attack on the authorization mechanism for a users session, not one that involves stealing passcodes. Also, it is worth considering using a different authentication tool for your Ace/Server until you are confident this problem has been fixed.

Insiders who want to masquerade as a different user on their local host have a particularly easy time of it, with access to the port numbers, time stamp, and other information.

V. Fixing the protocol

The protocol should be stateful. A stateful protocol would allow this problem to be fixed with a small change to the server software, namely replacing the timestamp sent out with a nonce of the same size. That would put enough information out of the reach of the attacker to make the attack impractical. It would also allow for easy detection when one user is trying to login multiple times on the same machine. In addition, a stateful protocol would require enough network access to forge TCP sequence numbers, which is more work than is needed in the case of a UDP based protocol.

The key used in the final step should be the previously negotiated secret key. That will fix the problem, by making it difficult to send a forged packet that will be accepted as authentic by sdshell but require updates to all the software deployed. Changing key management procedures, is of course, fraught with risk. That risk is probably less than continuing with the problems shown herein, and can be mitigated by proper cryptographic design. Confidence in the new protocol can be enhanced by publishing the protocols, and subjecting them to peer review.

It may seem sensible to separate the authentication and encryption keys. The possibility of signing the final authentication message should be considered. However, using a separate key that needs to be shared with each machine puts a large burden on the server to manage keys, where one DES key will do as well as two or more. The keys should probably be changed automatically from time to time to protect against an attacker who has obtained root access to the machine continuing to be able to get in, by his knowledge of the cryptographic keys.[2]

VI. Future Directions

For Research

The random numbers used as the client DES key seem to be generated fairly quickly, and may be weak. This observation is from personal experimentation with forced key regeneration. The process is much faster than generating SKIP, PGP, or other long lived keys. There may be order of magnitude improvements available over a brute force search of the DES key space. Obtaining known plaintext for these attacks is easy; depending on the strength of F2, obtaining chosen plaintext may also be feasible.

F2, SDTI_encrypt, and the card hash are all over a decade old, and do not take advantage of the many lessons learned by cryptographers since their creation. There are may be effective attacks against each. (SDTI_encrypt is a proprietary encryption algorithm by which some data stored on disk is protected.) The card hash can be obtained from the aceserver software, which can be obtained from the computer underground. If the card hash were broken, it may be possible to predict the expected output of a card, which would make it easy to steal pins, and then pretend to have the card. Examining these hashes would be an enjoyable avenue of research. The softcard, with its wintel implementation is a particularly nice target, because of the availability of free and commercial decompilers for that environment. Between the availability of the software in the underground, and the appearance of easily disassembled PC versions of the card hash, the ability of

[2] There are few systems which do well once the attacker has compromised the keys. Thus attack[s] against the system when session keys are known to an attacker are not examined in detail.

Security Dynamics to keep the card hash a secret over time is in question. It would be better to publish it, and allow it to be subject to broad peer review.

Version 2.1 and 2.2 of the software use an X11 based GUI, and require that at least some portion of the X libraries be installed. There are probably exploitable weaknesses in most configurations that would enable an attacker to breach the security of the Ace/Server. Those worried about this should probably isolate the Ace/Server, and only administrate it locally, or via SSH, or some other encryption package that ensures the X components will not weaken the system as a whole.

For Security Dynamics

Publishing a protocol before its widely deployed may prevent this sort of problem in the future. Even the best people miss things, and putting the protocol out for public review assures that the good guys as well as the bad can be assured that its strong.

There have been persistent rumors that SecurID was trivially bypassed for a long time. This may be the result of this, or other, problems[3]. As long as the protocol is hidden, it takes a lot of effort to find problems. For a long time, only the bad guys have had that amount of time and motivation.

VII. Contacts with Security Dynamics

Security Dynamics was first notified of this bug in July 1996, when Mark Warner and Chris MacNeil told us that the bug had been found and fixed by adding the client secret key into the information hashed by F2, thus, wp=F2(IP, T, P, c). Details about when this happened were not provided. When we asked John Brainard about this in August, he suggested that the attack would work. Security Dynamics was notified about the planned publication of this paper in November.

[3] TCP hijacking is a strong possibility.

VIII. Acknowledgements

Andrew Gross, Mark Chen, Bruce Schneier, and Matt Blaze provided advice, encouragement and guidance. Marcus Ranum, and Shabbir Safdar provided useful excellent feedback on drafts.

VIII. Bibliography

(FAQ) Security Dynamics, *Securid FAQ*,
http://www.securid.com/ID51.162015188143/Resources/FAQs/sdfaq3.html
http://www.securid.com/ID51.162015188143/Resources/FAQs/sdfaq4.html

(peiterz) peiterz, *Weaknesses In SecurID*

(Schneier) Bruce Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Ed.* John Wiley and Sons, New York, 1996.