TROJAN.GTALK | Cyber Engineering Services

Published: 2011-12-15 · Archived: 2026-04-06 03:12:22 UTC

Today I am going to write about an interesting Trojan, whose concept (controlling malware via instant messaging) has been used for some time. However Christmas came early this year and during one of our recent engagements we came across the C2 portion of this Trojan (screen shots are located at the end of this article).

The Trojan itself utilizes gloox, which is a free and publicly available jabber/XMPP client. Jabber if you are unaware is an open standard for instant messaging, which is employed by the instant messaging portion of Google Talk. This sample will connect to Google Talk with hard coded credentials. The C2 portion of this Trojan family will also connect to Google Talk using credentials provided at run time via the GUI. Once the two components have successfully authenticated with Google Talk all of the communication between the components and the Google Servers will be encrypted by means of TLS and SASL. The C2 portion can then gather system information, run the pslist and pskill command, upload and download files, issue sleep commands, and obtain a reverse shell.

The Trojan and the C2 have an additional layer of encoding, which to me was the interesting part of both of these samples. The hard coded credentials, the commands and responses for this sample are all encoded/decoded in the same manner. So that this article doesn't go from technical to soul crushingly boring only a high level explanation of the encoding/decoding will be provided below. The actual credentials are not provided in this document, however similar data was used as examples.

Trojan.GTalk Analysis

```
File Name:  Trojan.GTalk.exe
File Size:  353792 bytes
MD5:        8845cb5b4e450cb10a3b6ca41a9b4319
SHA1:       bd224865730ff72d960a8ea49be315fdc615edb3
PE Time:    0x4E4A32CF [Tue Aug 16 09:05:19 2011 UTC]
PEID Sig:   Microsoft Visual C++ 8
Sections (5):
Name      Entropy  MD5
.text     6.58     bfb2e60a800996224698f5a81b80e8d1
.rdata    4.95     dbd4ac5000eda9e6e9124d72858d29b7
.data     4.46     54c204495e80764a21da3decd330cbb3
.rsrc     4.51     ffb05bcee52f5e69168029d4ffa5ccf1
.reloc    4.35     e6cfc56984a9068e2e5d3ca27cf67919
```

```
AV: 2/43 (4.7%) [VIRUS TOTAL]
```

It should be noted that the hash values above do not match the hash values listed in Virus Total. The log on credentials were removed from the sample that was submitted to Virus Total. The hash values above are the correct hashes for the

sample with the encoded credentials still in place.

- This sample does not entrench itself on the compromised system. Most likely the Trojan is entrenched on the compromised system either manually or by a dropper/installer file.

Decoding Credentials

This sample will take care of some basic housekeeping before it begins to decode the credentials that will be used to authenticate to the Google Talk servers. The credentials can be located in the file at offset **0x42d84**. An example of the log on credentials (username in blue and password in red) can be seen below and are both null terminated strings.

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00042D80   00 00 00 00 2B 34 71 4B  69 51 64 35 4D 2B 6F 4F       +4qKiQd5M+oO
00042D90   70 66 4E 62 6A 37 75 2F  75 71 6A 61 4D 78 73 57   pfNbj7u/uqjaMxsW
00042DA0   31 50 58 37 46 6D 75 39  4E 4C 6D 7A 5A 48 4E 58   1PX7Fmu9NLmzZHNX
00042DB0   62 66 63 3D 00 00 00 00  00 00 00 00 00 00 00 00   bfc=
00042DC0   00 00 00 00 2B 34 71 4B  49 56 6F 50 2B 56 54 6A       +4qKIVoP+VTj
00042DD0   71 4C 79 4B 78 44 39 41  2F 67 65 39 38 4F 6F 2F   qLyKxD9A/ge98Oo/
00042DE0   63 48 47 4B 69 67 3D 3D  00 00 00 00 00 00 00 00   cHGKig==
```

- This sample will Base64 decode the first string using a custom alphabet mapping. An example of the first string, in its decoded form, can be seen below.

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000   FB 8A 8A 8A A6 F9 33 E3  A8 A5 F3 5B 8F BB BF B9   ûŠŠŠ¦ù3ã¨¥ó[ »¿¹
00000010   A8 EA 33 1B 16 D4 F5 F7  16 6B B4 34 B9 B3 C4 73   ¨ê3  Ôõ÷ k´4¹³Äs
00000020   57 6D F7                                          Wm÷
```

- This sample will then use a hard coded table that is located at offset **0x4e208** to further decode the above string. This step is just a large substitution cipher. The table located at the referenced offset is concealed in a larger portion of code, which is not used by the Trojan. The table (0x100 bytes in length and in black) can be seen below.

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
0004E1E0   54 60 3C 50 1F 20 97 A8  37 04 21 FF 06 17 DC AF   T`<P  –¨7 !ÿ  Ü¯
0004E1F0   A9 09 42 D1 B5 8B 3B 2D  AC 47 8C 86 3D 29 B8 84   © BÑµ‹;-¬GŒ†=)¸„
0004E200   10 7B 96 A6 1B E8 33 F3  41 9C 83 34 E1 D1 E4 B0    {–¦ è3óAœƒ4áÑä°
0004E210   1C E9 3C 70 80 0E 4A 93  F8 2A 06 B4 4C 55 7C E5    é<p€ J“ø* ´LU|å
0004E220   53 2D 5B FC 49 79 67 DC  DD E2 38 44 A2 66 6F 5A   S-[üIygÜÝâ8D¢foZ
0004E230   A9 F5 A0 62 AC EF 57 73  C8 A6 BE FE CD 97 4D E0   ©õ b¬ïWsÈ¦¾þÍ–Mà
0004E240   78 14 48 EE DA F4 0D 1F  8F D6 EA AF D0 25 74 F1   x HîÚô  ÖêÐ%tñ
0004E250   28 4E 86 2E 15 9D C2 BB  DB 98 76 99 D8 27 3F CA   (N†. Â»Û˜v™Ø'?Ê
0004E260   A7 4F 47 03 8D A3 A8 46  D9 0B 58 9A D5 8A 18 22   §OG £¨FÙ XšÕŠ "
0004E270   1A CE 37 CB AA B6 6B C0  8C 95 91 8B 68 CC D2 B1    Î7Ëª¶kÀŒ•‘‹hÌÒ±
0004E280   59 2B 4B F9 87 89 BF 12  0A 7A 77 7B F7 52 F3 61   Y+Kù‡‰¿  zw{÷Róa
0004E290   B7 29 00 ED F2 96 69 13  63 45 17 5D 51 C4 FA DE   ·) íò–i cE ]QÄúÞ
0004E2A0   7D 35 88 84 56 C1 B2 82  90 6D B9 AB A5 D7 FB BC   }5ˆ„VÁ²‚m¹«¥×û¼
```

```
0004E2B0   DF E8 43 11 F0 32 9B E7  64 C7 33 AD 30 EC 24 31   ßèC ð2›çdÇ30ì$1
0004E2C0   F6 7F 6E 07 C6 36 BA 75  C3 08 23 AE 50 0C BD 81   ön Æ6ºuÃ #®P ½
0004E2D0   1B 0F 8E 3E 42 9F 5F 71  1E EB A1 21 40 2C 02 C5     Ž>BŸ_q ë¡!@, Å
0004E2E0   B8 72 3A 3D E6 19 CF 65  92 20 10 9E 6C 54 39 01   ¸r:=æ Ïe’  žlT9
0004E2F0   FD 04 85 B5 05 5C C9 94  D4 6A 09 FF B3 2F 16 60   ý …µ \É”Ôj ÿ³/ `
0004E300   3B 7E 26 1D D3 A4 5E E3                           ;~& Ó¤^ã
```

- An example of how the substitution cipher works is as follows. The Trojan will use the first byte of the string (0xFB or decimal 251) and take the value located in that position and replace the original value of the string. This will occur for each byte of the string.

Position	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
Value	**92**	**20**	**10**	**9E**	**6C**	**54**	**39**	**01**	**FD**	**04**	**85**	**B5**	**05**	**5C**	**C9**	**94**
Position	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	**FB**	FC	FD	FE	FF
Value	**D4**	**6A**	**09**	**FF**	**B3**	**2F**	**16**	**60**	**3B**	**7E**	**26**	**1D**	**D3**	**A4**	**5E**	**E3**

Original String:

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000   FB 8A 8A 8A A6 F9 33 E3  A8 A5 F3 5B 8F BB BF B9   ûŠŠŠ¦ù3ã¨¥ó[ »¿¹
00000010   A8 EA 33 1B 16 D4 F5 F7  16 6B B4 34 B9 B3 C4 73   ¨ê3  ÔõÙ k¹4¹³ds
00000020   57 6D F7                                          Wm÷
```

Decoded String:

```
00000000   1D 00 00 00 FB 5E FE 9E AF D7 FF 33 13 07 75 7F  ...û^þž¯×ÿ3  u
00000010   DF 82 FE FC 7D 40 2F 7E 7C CB 7F 32 7F AD D5 8B  ß‚þü}@/~|Ë2Õ‹
00000020   CA B6 60                                         Ê¶`
```

The next part of the decoding scheme is explained, for brevity, at a very high level. From my research I could not determine that this is a standard or well-known algorithm. Fully explaining how this algorithm works could be a blog unto itself. In the future, if time allows, I will write an article covering all the details of how this algorithm works and how to decode/encode data using the algorithm. The algorithm creates a 4,392 byte table of values. During the decoding process the position of values (specifically the ones used to decode) are exchanged with other values in the table, adding another layer of protection.

- The first Dword of the decoded string is the length of the final decoded string after the next stage of decoding.

- The final stage involves an algorithm that encodes and decodes data on the bit level. This bit stream encoding comprises a series of instructions which breaks each byte down into its binary equivalent (0xFB would be 1111 1011) . Each one of these binary values are treated as an integer and added to a hard coded starting value. The sum of the two pieces will act as an offset into the previously referenced **4,392** byte table. This table is created in memory at run time, from another set of instructions.
- The offset into the table will point to a word value, which will be added to the next integer representation of the binary data. This technique continues until the sum of the word value and the binary value exceeds the hard coded value **0×273**. Once this criteria has been met the algorithm branches into another set of instructions. These instructions will perform some simple math to determine a pointer into the table. The value at this pointer is the decoded value, which will be written into memory. The algorithm then branches into another set of instructions that scrambles and alters (by means of addition) values in the 4,392 byte table, by exchanging several word values that were used to decode the previous byte of data. The algorithm will then continue with the steps outlined above until it reaches another value above **0×273**.
- The result is the decoded string. The Trojan will then complete the sames steps to decode the password used to authenticate with the Google Talk servers.

Trojan Communication

The Trojan communication portion of this sample involves authenticating to Google Talk servers. This is accomplished with the credentials that were decoded above. Once an attacker has authenticated to the Google Talk Servers (via the GUI C2 node), the two pieces can begin communicating. The C2 node will issue commands, which are transmitted as integer values. These values are encoded in the steps above reversed (bit stream encoded, substitution cipher, and then Base64 encoded) and transmitted to the Trojan, via a secured conduit provided unwittingly by Google . The Trojan will decode the message it receives and send a response to the C2 in the same manner. If the C2 node establishes a reverse shell or uploads/downloads files, that data will also be encoded in the same manner.

Once you get past the encoding/decoding portion of this sample everything else, including the commands, are straight forward and have been seen before in previously analyzed samples. Below is a screen shot of how this sample determines the commands sent.


commands1-1024x354

I have included screen shots relating to the functionality of the GUI C2 portion of this family.


first

Initial Screen


logon2

Log On Prompt

I provided a set of credentials that I created for the analysis of this sample.


loggedon

Logged On, showing available compromised machine and embedded username

The sample that was analyzed was patched with another set of credentials that I created for the analysis of this sample. The two google accounts were paired or connected prior to the analysis.

Initial Command Screen for available compromised machine

Info Command

Pslist Command

Pskill Command

Uploading a file to the compromised machine

Downloading a file from the compromised machine

Reverse shell to the compromised machine
