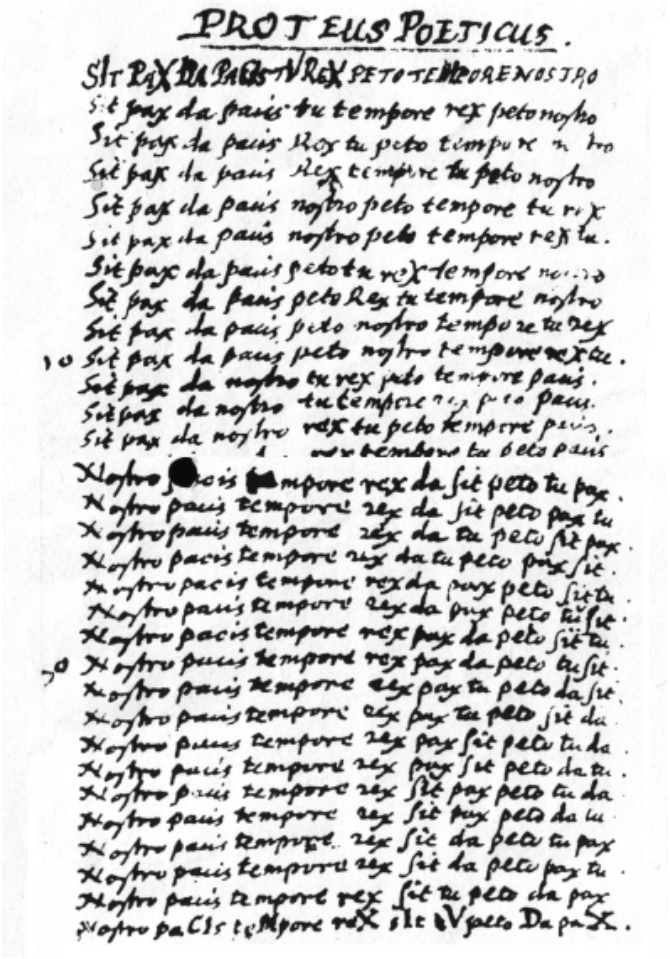
BUFFER OVERFLOWS: SUBVERTING NETWORK CODES IN COMPUTATIONAL ART

FLORIAN CRAMER

BUFFER OVERFLOW 1627 A.D.

„Sit Pax da pacis tu rex peto tempore nostro“[1]



Date: Sept. 11, 2004.

[1][Wag71], 2f.

„Let there be peace, gives us peace in this time, I ask you, my lord“.

The above line was written by an anonymous poet in Germany, in the ninth year of the 30-years-war that destroyed much of Middle Europe. The timestamp is given in the text itself, because its capitalized letters form, in the poetic form of a so-called chronostych, the letter sequence “MDCXXVII”, the Latin number 1627. There is more mathematics going on in the poem: The manuscript contains 24,480 lines each of which shuffle the words of the first line in a different way according to the mathematical-combinatory law of permutation. They write out all possible permutations which keep the meter of initial line.[2].

The text thus is three things at once: A poem, a prayer and a computational algorithm prototyping a modern computer program. The combination of poetry and algorithmics however was not new as such, since poems permuting their words in every line are known since the Latin antiquity and were canonized as “Proteus verse” in the poetics of Julius Caesar Scaliger in 1562.[3] What seems remarkable here is the combination of algorithmic poetry and peace prayer. A prayer is, by its nature, a performative or (after J. L. Austin[4]) a illocutionary speech act. As a computational program, this performative quality, or intensity, is reinforced. Another example that combines prayer, verbal art, and intensification through repetition and variation is, 350 years later, Steve Reich’s tape composition “It’s gonna rain” [example].

In both works, the 17th century poem and the 20th century tape music, the speech act becomes a physical performance. In the poem, it is not the voice, but the act of writing since the anonymous poet writes down all 24,480 lines by hand. In modern terms, one could call it cracker-style “flooding” of a communication channel, or a poetic virus spreading not a love, but a peace message, in the hope that through its multiplications and viral mutations, the prayer may be heard both by God and spread as a peace spell among mankind. The communication network flooded here is metaphysical however, not physical like the modern Internet.

NETWORK BUFFER OVERFLOWS

Just like a modern computational network virus, the poem has four essential components which it intrinsically links to each other in order to propagate:

(1) the source code, or the (here implicit) algorithmic instruction
(2) the execution of the instruction, into 24,480 instances of output
(3) a network channel; the prayer between men and god, possibly also including the church congretation (there is another permutative peace prayer from that time that was sung as choral in a church)
(4) the target; human communication and politics, via the vehicle of god. The poem tries, so-to-speak, to end attacks with weapons through an attack whose weapon is the word.

[2][?], 3
[3][Sca61]
[4][Aus55]

Of particular delicacy is the relation between word and instruction. Communication networks could be generally defined as entities which transport information by the virtue of a framework of codes that govern their functioning. The code could be legal agreements and regulations, such as in the postal network, or it could be machine instruction code such as in the Internet with its software protocols and forwarding agents. For the network to function, it is important to strictly isolate its internal instruction code from the information transported. What happens in the permutational prayer however is that the information transported tries to break out of its containment and rewrite, or overwrite, the communication architecture.

In other words, the data turns into a program, i.e. that what is transported unexpectedly reprograms the system. This is exactly the way an E-Mail virus works: The message is not just simply a message, but an agent. In computer programming, a comparable, yet more general construct of this mutation of data into programs exists in the concept of the so-called "buffer overflow", or "stack overflow". Today, buffer overflows are the most frequent reason of software security holes both in Microsoft Windows and in Unix-like operating system. The security section of the current "Linux Weekly News", for example, lists seven grave security holes caused by buffer overflows discovered only in the week between August 26 and September 2, among others in the Adobe Acrobat reader, in the Linux kernel, in the popular Python scripting language, in the Samba networking code and in Squid, the software powering most web proxy servers in the Internet. A buffer overflow creates exactly the kind of security holes described before: with their help, user data can be turned into program instructions and thus allow a user to hack (crack) a remote system, for example the computer running a web proxy server.

This is how a buffer overflow works: Computer programs store temporary data in stacks. In the most common programming languages C and C++, the size of the stack must be defined in advance by the programmer.[5] If the program puts more data into a stack than the programmer anticipated, then it writes into memory used by other programs running on the system. For example, if an address database program was designed to hold up to 32 characters for name entry and the programmer did not take precautions to verify and, if necessary, shorten the user input, a user of the database could overflow the buffer by entering more than 32 characters which thus could end up somewhere in the running operating system and be processed as administrative commands. This way, malicious instructions can be inserted into the running system, even by a user without administrative access to the machine.

In a similar way, the 24,480 lines of the permutational poem are designed as a buffer overflow through which mere data mutates into a program, namely a benevolent "white hat" hacker program to spread peace on earth. From there, it is not hard to imagine opposite-complimentary "black hat" scenarios.[6]

Another reading of the 17th century poem is that it tries to stretch the limits of poetry in its original sense of "poeisis", creation. It does something in very direct, unmetaphorical way, instead of just being, in the understanding of art shaped later

[5]Programming languages with higher hardware abstraction don't require this, but can be affected as well if their interpreters or compilers are themselves written in a low-level language like C, as the above Python example shows.

[6]And here I think of course of Al Qaida and its systematic tactical use of travel networks, telephone networks, mouth-to-mouth-propaganda, video and television, and the Internet.

by romanticism, an aesthetic object. It seems that art whose understanding it is to do something out of itself is typically at odds with the communication networks within which it operates, and tries to rewrite its codes through finding cracks in its cultural setup. This is true to for the radical avant-garde movements of Futurism, Dada, Surrealism, Fluxus, body and performance art. But as the example of the poem shows, the performance does not need to be physical, but can be created purely on the level of symbols.

BUFFER OVERFLOWS AND DIGITAL ART

A contemporary example of such a symbolic performance is the work „OSS“ by the Dutch-Belgian net artists jodi `http://oss.jodi.org`. It uses the Javascript programming language to flood the web browser with myriads of small moving windows. However, the site does no real harm to a computer and creates the impression of an Internet virus only in the imagination of the spectator. This illusionism was so successful that www.jodi.org was temporarily shut down by its provider ca. 1998, because its management believed it infected computers. After jodi, a whole genre of Internet art and Internet poetry, often referred to as ”codework“, played with this illusion and the slippery, user-frightening boundaries of data and program code in the digital network.

```
From: "][mez][" <netwurker@hotkey.net.au>
To: _arc.hive_@lm.va.com.au,7-11@mail.ljudmila.org
Subject: _cross.ova.ing ][4rm.blog.2.log][ 12/06-10-06-03_


_____________________________________________
_{ soc[d]ial engine     09:02am 12/06/2003_
_____________________________________________

# [social engine] woo[t!]l.gathering
# [re]mixed N d.filing sy.stem[N root//shard N slice]
# rod-bird strait vs corna.stoned N da[ta]nge(nt)rous

# pictory purrfect N pump_muscles//bloated S[OAP]car_[t]issues
# e[mailing].lectro.lighting.my.pores.in2.sms.destruction


--dialin.objects.with.honeyed.lines
_____________________________________________________________
```

The above example is an blog, or diary, entry of Australian net artist mez (Mary Anne Breeze). She writes in a self-invented language that hybridizes English and syntactical elements of programming languages to allow multiple readings of her text, for example in the title “social engine”/“dial engine”. Similarly, the text is a reflection of social system as technical systems with its blending of computer tech jargon and references to the human body. Like all codeworks, the writing plays with the potential confusion of its readers about whether it is data or a program, a message from a human being or a machine, and whether opening this text might be

doing something harmful to the machine. Also this is poetic in the literal sense of technical doing, and it lives entirely in the context of E-Mail and the communication network codes of the Internet. It is thus a metaphorical buffer overflow in that it acts on the reader's imagination (and fear) of the machine rather than on the machine itself.

To conclude my presentation: Buffer overflows and other mutations from data into program code, whether real or imaginary, digital or pre-digital, are potent examples of both the power and pathologies of communication networks. Their ambiguity makes them attractive as artistic material. They also show how digital art is not necessarily clean laboratory hightech, but can also be abysmal, ironical or even sarcastic art. In extreme cases, like artistic virus programming,[7] it can bring up the old issue of how art relates to ethics.

It is a typical attitude in artistic circles to first embrace network security holes as welcome contingencies of seemingly rigid systems, and then go through a learning process once the own communication networks, like servers and mailing lists, get abused and cracked.[8] Security holes like those announced last week for Python and Samba then are no longer funny. Artistic exploitation of security holes on the other hand shows that absolute network security is a totalitarian illusion, all the more, when codes can also be subverted via hacking the imagination of the user [who might wipe her/his computer's hard disk thinking that it got infected by jodi's or mez's virus].

REFERENCES

[Aus55] J.L. Austin. *How To Do Things With Words*. Harvard University Press, Cambridge, Massachusetts, 2 edition, 1975 (1955). 2

[Sca61] Julius Caesar Scaliger. *Poetices libri septem*. ?, Lyon, 1561. 2

[Wag71] Christian Wagenknecht. Proteus und permutation. *Text und Kritik*, 30:1–11, 1971. 1

E-mail address: `cantsin@zedat.fu-berlin.de`

[7] See the "I Love You" exhibition accompanying this conference

[8] Examples are the disappearance of the open audio archive orang.orang.org through a cracker attack on the server, or the multiple spamming problems that gradually turned previously open net cultural mailinglists into closed or semi-closed forums.