Hacking NIMEs

Abram Hindle
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada
abram.hindle@ualberta.ca

ABSTRACT

NIMEs typically focus on novelty but the cost of novelty is often to ignore other non-functional requirements and concerns such as usability or security. Digital security has probably not been a concern for performers due to the duration of their performances and lack of disrespectful hackers, known as crackers, in attendance carrying the appropriate equipment and software necessary to hack a performance. Yet many modern NIMEs could be hacked from smart-phones in the audience. The lack of security hardening makes NIMEs an easy target — but a question arises: if hacking can interrupt or modify a performance couldn't hacking itself also be performance? Thus would music hacking, live-hacking, be similar to live-coding? In this paper we discuss how NIMEs are in danger of being hacked, and yet how hacking can be an act of performance too.

Author Keywords

network security, software security, opensound control, hacking, live-hacking

ACM Classification

H.5.5 [Information Interfaces and Presentation] Sound and Music Computing, K.6.5 [Security and Protection] Unauthorized access (e.g., hacking, phreaking)

1. INTRODUCTION

A problem with developing an instrument for computer music performance is all of the non-functional requirements one has to address, such as usability, reliability, and run-time performance. Often a computer musician runs into run-time performance problems as their computer cannot keep up with their music. But one kind of non-functional requirement that is rarely addressed in computer music performance is security.

One threat to security that we hear about in the media day to day is hacking. The media portrays hacking as the malicious use of a computers to gain access to networks, computers, and information without permission. Richard M. Stallman [19], of the Free Software Foundation, argues that this malicious use is "security breaking" and should be called cracking, while the subversive and playful use of technology in unexpected ways should be considered hacking. This subversion is the face of hacking within the NIME community which has unquestionably welcomed the hacker ethos [19, 13]. A hacker is someone who tries to push the limits of a device, a system, or a piece of software. Many NIME instruments and experiments do just that, for instance Ferguson et al [7] sought to control guitar feedback through a method of automatically generating controlled guitar feedback — not something a guitar was intended for.

NIME instruments tend to push boundaries, but those boundaries are often not security relevant. There is a whole class of performances, NIMEs, and computer music instruments who are affected by the concern of security: wireless performances, wireless NIMEs, and internet hosted networked-computer music performances.

Currently wireless performances are popular, especially with laptop orchestras [21, 18] that use wireless networks to transmit MIDI and OSC messages. Some performances occur over the internet, on computers in data-centers, or within the cloud [5, 9]. Networking and networked computers open up avenues for malicious use and even malicious audience interactions or new potentials for performance.

Thus security is relevant to NIME community both as a concern and a non-functional requirement that should be addressed, but also as a *novel form of performance* similar to live-coding [12, 22]. Consider these three contexts: 1) performance that is interrupted maliciously by a cracker (a malicious hacker); 2) a performance that is augmented, modified, or interacted with by a hacker; 3) a performance that is the live-hacking of a performance meant for hacking. When an instrument's security is hacked, a performance can be turned on its head and manipulated via hacking, hacking is a form of improvisation akin to live-coding.

One of the most popular technologies used in wireless and wired networked performance is *OpenSound Control* [24] (OSC). OpenSound Control is used by many software synthesizers including Max/MSP, pure-data, CSound, chuck, SuperCollider, and many more. In this paper we will use OpenSound Control as the main vector of attack and talk about security issues that one faces if one uses OpenSound Control, especially over UDP rather than TCP. We subvert existing network security tools such as `scapy` and `pcap` [4, 10] to manipulate *OpenSound Control* messages on a network in order to repeat, replay, and modify messages.

In this paper we contribute the following:

- Discuss security weaknesses in current networked NIMEs.
- Propose some solutions to security issues facing NIMEs.
- Propose hacking as performance — much like glitch [17] or live-coding [12, 22].
- Provide concrete examples of automated methods of hacking OSC driven performances.

NIME'16,
.

1.1 Dangers of Performance Cracking

A performance can be non-musically interrupted by an unethical hacker, a cracker. There are 3 main avenues that a performer should be weary of: denial of service, security exploitation of networked computers, and dangerous parameters.

Denial of service (DOS) is when a service such as a networked computer is attacked such that it is overloaded. This could be done by making too many requests, making large requests, or rapidly requesting work be done at a higher rate than work can be accomplished filling a queue. Playing too many instances of an instrument concurrently can achieve a denial of service. Jamming, in the sense of radio and network availability, can be considered a denial of service if an attacker flooded a network with irrelevant requests. A network could be flooded by using up available bandwidth, perhaps by downloading a movie via bit-torrent, preventing the network from being used for performance. If more than 1 host takes part in this denial of service is it considered a *distributed denial of service* (DDOS). OSC can be exploited to DOS services simply by repeating commands.

Security exploitation is when weaknesses in software are exploited in order for an intruder to gain access to a computer. We will not cover much of these issues in this paper but be aware that once software is known to have exploitable bugs that attacks are often automated, enabling a hacker to simply scan a network and automatically attack vulnerable clients. Products such as Metasploit [15] collect such exploitable bugs and payloads to deploy on vulnerable systems. The best defense for a performer is to use up to date software and isolate their performance computers from potential curious hackers and malicious crackers.

Dangerous parameters are a concern if an attacker gains access to a network they could attack OpenSound Control enabled devices on the network. Some of these devices can damage themselves. For instance, you could imagine that a robot-drummer could be damaged by commands that cause it to strike too hard. Any device that could be harmed via OSC commands should be isolated or have its commands filtered such that it is within acceptable parameters.

2. UDP AND OSC

One of the main weaknesses of modern computer music performance is the reliance on transport protocols that are connection-less. UDP, unlike TCP, only contains port numbers and no other context. Thus if one captures a sole UDP message, one can repeat it and inject it back onto the network using tools such as `pcap` [10].

OSC usually sits on top of UDP and provides a path for routing. It has no abstraction or support for a session or a connection. Some authors use OSC over TCP. Typically one listens for OSC packets on a UDP port and then forwards the OSC message to a software synthesizer such as SuperCollider.

2.1 Weakness of UDP

UDP does not maintain a connection, thus UDP packets can be lost, arrive out of order, and also be potentially duplicated. UDP packets do not need any sort of handshaking and can be sent directly. UDP packets are also susceptible to *spoofing*, this is when one host impersonates another host and sends traffic from a different IP address. Thus UDP packets can be replayed and because there are no unique identifiers or sequence number neither side knows if a packet is in fact new.

Notice in Figure 1 that there is no session ID or anything in the headers to track a session or connection. This is because both IP and UDP are connection-less transport protocols. They are meant to allow routing and communication/transport of data. What is not depicted in this figure is the ethernet packet which often encapsulate these IPV4/UDP packets. Any kind of connection could be maintained through OpenSound Control parameters or paths but it would require everyone involved to agree to a connection protocol. Thus UDP does not strongly enforce identity or security during communication.

2.2 Weakness of OSC over UDP

The OSC message format for OSC packets has no identifiers or sequence numbers to ensure in-sequence operation — although there is a bundle abstraction for combining multiple messages into 1 packet [1]. When OSC is combined with UDP anyone can capture and replay OSC messages, spoofing their origin, or just inject any particular messages onto the network.

Many OSC configurations do not validate a client's IP address against a white-list. But OSC servers can white-list certain clients to provide a minor hurdle for hackers. OSC provides no protection against duplicate messages, or other hosts injecting messages. Furthermore typical setups for receiving UDP OSC packets do not limit the number of clients. Anyone sharing the network could send a OSC message to a host.

Thus OSC over UDP is susceptible to:

- replay attacks — where traffic is captured and replayed;
- denial of service (DOS) and distributed denial of service (DDOS) attacks [8];
- spoofing — whereby the origin of the OSC packet is obfuscated [8].

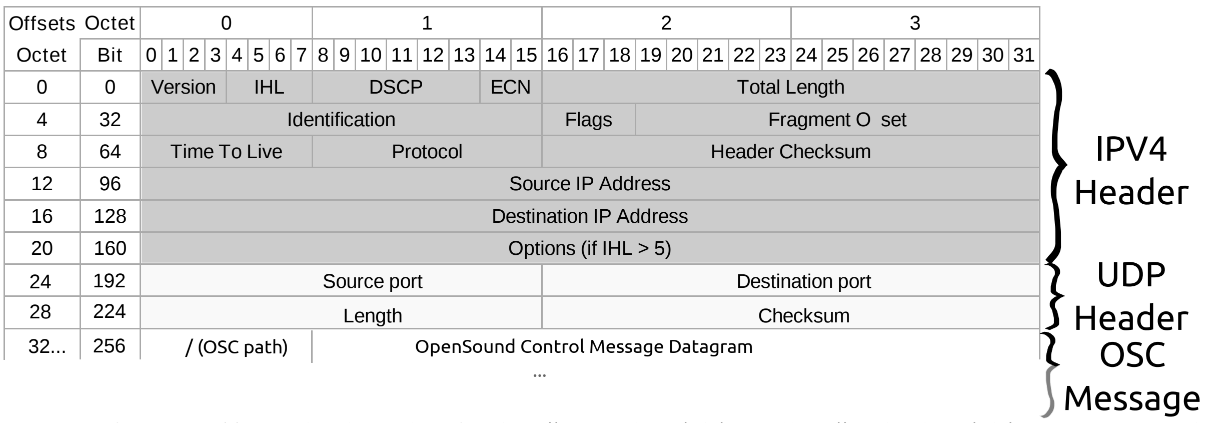
3. PREVIOUS WORK

There is much work relevant to networked computer music. Barbosa et al. [3] surveyed networked computer music since the 1970s. One reason to employ networking in a computer music instrument is to provide wireless interfaces or user interfaces to mobile devices over the web, thereby allowing audience participation. Oh et al. [16] demonstrate the use of smartphones in audience participatory music and performance. Jordà [11] describes many patterns of collective control and multi-user instruments as well as the management of musicality of instruments. Dahl et al. [5] used twitter in TweetDreams to promote audience participation by aggregating audience tweets to alter the music instrument watching that twitter account.

User interfaces are often communicated over the network via HTML. For instance Jesse Allison et al. [1, 2] present the Nexus framework which distributes a musical user-interface via the web. Weitzner et al. [23] provided a user-interface proxy for Max/MSP called *massMobile*. This tool enables interaction with Max/MSP over the web. Some of these interfaces by definition could be automated or potentially "hacked".

Laptop orchestras [21, 18] are composed of laptop synthesizer users who are connected together, often with OSC messaging. Lee et al. [12] describe many opportunities for live network coding. They argue that networked live coded music allows for interesting mixes of centralized and decentralized synthesis and control. Lee's arguments amplify our argument that open and relatively insecure networked live music allows for decentralized control and hacking based synthesis.

[1]OSC bundles will start with `#` rather than a `/`



Offsets Octet	Octet	0	1	2	3
Octet	Bit	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
0	0	Version, IHL	DSCP, ECN	Total Length	
4	32	Identification		Flags, Fragment O set	
8	64	Time To Live	Protocol	Header Checksum	
12	96	Source IP Address			
16	128	Destination IP Address			
20	160	Options (if IHL > 5)			
24	192	Source port		Destination port	
28	224	Length		Checksum	
32...	256	/ (OSC path)	OpenSound Control Message Datagram		
			...		

Derivative of CC-BY-SA 4.0 (C) 2016 Wikimedia Foundation from https://en.wikipedia.org/wiki/IPv4 and https://en.wikipedia.org/wiki/User_Datagram_Protocol

Figure 1: IPV4 + UDP + OSC Message packet structure. Notice the lack of a session ID or any identifier that could be used to track state or a connection. UDP is a connection-less protocol and contains only source port, destination port, length and check-sum. IP addresses are provided by the IPV4 header. The OpenSound Control message resides after the UDP header.

Some of the OSC relevant security issues, related to OSC over UDP, were discussed within the OpenSound Control Forums [8].

4. CONTEXTS OF NETWORKED MUSIC

To understand the impact of security issues we need to look at some of the contexts of networked computer music. We will address 2 main contexts, performing live before an audience with networked instruments/installations, and performances over the internet.

4.1 Laptop Orchestras, Live Networked Performance, and Networked Installations

Laptop orchestras [21, 18] as discussed before typically are networked portable computers in a room or on stage, operated by performers in a fashion similar to how a chamber orchestra might perform. The networking for a laptop orchestra is often wired, but wifi is so popular that many laptops do not come with ethernet ports anymore. Laptop orchestras often present a piece performed by numerous computer musicians on their laptop. Live networked performance is any live performance that uses communications networks. Networked installations are musical installations that are networked computer music instruments and are typically meant for interacting with the public.

4.1.1 *Attack Vectors*

If the laptops are totally wired into the network their attack surface is much smaller. A malicious audience member might have to physically manipulate a switch or hub connected to the network. If the wired network is connected to the internet, the attacker only needs to sniff packets and they might discover the traffic of the laptop orchestra. Regardless wired networks typically requires physical access or an internet connection for an attacker or curious hacker to connect.

Wifi (802.11abgn) is very popular and most laptops come equipped with wifi cards. Wifi networks are far easier to covertly gain access to. Many performances will try to improve latency by not using encryption [14] (WEP / WPA / WPA/2 / etc.) and leave the wifi-network open. In this case, the attacker can simply associate with the network and watch for musical traffic. Once an OSC message is sent across the network an attacker or curious hacker can copy and resend that message.

If a networked performance or installation provides physical access to a computer just about anything could happen. The computer could be rebooted and forced to boot a new operating system that enables the attacker or curious hacker to modify the installation itself. Even without physical access to the main computer, if a browser was provided there are many ways for an attacker to start an inspector and use JavaScript to hijack the installation. These issues are out of the scope of this work as we focus on network security.

4.1.2 *Solutions?*

For wired networks: try to ignore traffic from outside the local network, and make physical access difficult.

For wifi networks the accessible wifi-area can be tuned with specially designed and organized antennas [14]. Wifi networks can try to prevent intrusion via encryption via WEP, WPA, or WPA/2. Encryption requires performers share such information about how to connect before hand and keep it a secret from the audience. In the case of WEP or WPA, these are crackable within the time of a performance [6] — use WPA/2 if you are encrypting your wifi-network while keeping your key secret.

- Bring your own wifi-router and use encryption (WPA/2).
- Choose wifi channels wisely: tune your router to avoid interference with other channels.
- Consider using unpopular spectra such as 802.11a to avoid interference.
- In case of audience networked interaction, use a separate network for audience participation.
- Avoid Wireless if possible; consider wired connections.
- Harden user interfaces — ensure there is no way to quit or escape.
- If possible do not connect to the internet.

4.2 Over internet performance

Some performances take place over the internet. Tome et al. [20] present a *massively multi-player online* (MMO) drum machine that used twitter and is available as website. Ignoring the concerns of OSC, the website can be attacked in terms of denial of service, as well it can have its security breached — this is beyond the scope of this paper. Regardless the performance is hosted online and available to anyone. Alternatively we have performances like TweetDreams [5] that are clients to other internet services.

4.2.1 Attack Vectors

The main attack vector is the internet itself and then the interface surface, the webservices, and open OSC ports of the *over internet performance* itself. These services are susceptible to interruption via denial of service attacks and live streams can be affected as well. TweetDreams could be interrupted if the network that was hosting the performance was affected or interfered with.

Sometimes performers assume no one will do anything negative and leave a service open — this allows exploration but can also cause interruptions. The main issue with the internet is overload — too many requests will harm a performance. Or in the case of a client to the internet performance, any interference with network access will result in an interruption of the performance. Something as trivial as a user downloading email on the same wifi network could cause an interruption.

4.2.2 Solutions?

Thus what should we watch out for with internet enabled performances?

- Use a wired connection to the internet for internet client performances.
- Assume the worst and require authentication for control systems.
- Rate-limit input from sessions and hosts.
- Employ DDOS protection or distributed hosting — use content delivery networks or ISPs who can provide DDOS protection.

5. ARMORING INSTRUMENTS

In general if one wants to prevent interference with one's performance consider using encryption not only to lock down the wifi network, but to ensure that messages sent are from trusted sources.

Signed messages.

OSC messages could include a string which contains a cryptographic signature. This allows the host to determine who made the message and prevents tampering with the message — it does not prevent replay attacks. An attacker can replay signed messages at a later time.

White-listing MAC addresses or IP addresses.

White-listing MAC addresses prevents attackers or the curious from joining your wifi network, but they can often just sniff wifi-packets and steal the appropriate MAC address — MAC addresses can be set. IP addresses can be spoofed once the attacker is on the same network. A knee high fence will keep some people out of your yard, thus minor hurdles can befuddle attackers who are racing against the clock to disturb a performance before it is over.

Lock down Wifi or use a VPN.

If you use encryption (WPA/2) on Wifi you will prevent most attackers. Alternatively if a virtual private network is employed it is unlikely an attacker can access the virtual private network (VPN).

DOS protection via rate limiting.

Denial of Service can avoided by limiting the number of commands can execute at one time, especially on a per-client basis.

6. HACKING AS PERFORMANCE: CASE STUDY

Hacking need not be malicious, while much of this work has focused on dealing with unwanted interruptions or malicious cases of hacking, cracking, it is important to note that hacking can be a performance unfolding over time. Thus much like live-coding or playing an instrument, the hacking of a performance can be itself performative. Those who perform by hacking we will refer to as *live-hackers* much like we refer to live-coders.

There are numerous ways to turn hacking/packet manipulation into an engaging and interesting performative work:

- Display the live-hacker's performance terminals used for the hacking — show the network traffic and the manipulation of the traffic.
- Visualize network traffic with graphics, histograms, and communication graphs.
- Project the use of an interactive interpreter such as bpython or the scapy interactive shell [4] to live-code the packet routing and sniffing.
- Combine live-coders and live-hackers.
- Forward OSC packets to different addresses and hosts.
- Capture and replay allows for riffs discovered during improvisation to be recorded and played back across the network regardless of the number of hosts involved to play that riff.

In the following two subsections, examples of automatic OSC packet manipulation are provided that follow the same general structure: a network adapter is placed into promiscuous mode and a program listens to UDP OSC traffic being sent to it or sent near it. The network adapter is the network card (either wifi or ethernet). Promiscuous mode tells the network adapter to listen to all packets regardless of the MAC addresses of the packets. This enables traffic sniffing and allows programs like scapy [4] to listen to traffic sent to other hosts over the same medium (CAT5 cable or wireless). The program sniffing these packets, then responds either by re-sending the OSC messages later or by forwarding it off to a client to manipulate further. The amount of code necessary to listen to for OSC packets is minimal and a Python example is depicted in Figure 2.

```
import dpkt, pcap

pc = pcap.pcap(name="lo")
pc.setfilter("udp")

for ts, pkt in pc:
    eth = dpkt.ethernet.Ethernet(pkt)
    ip, udp = (eth.data, eth.data.data)
    if packet[0] == '/':
        print 'udp'
```

Figure 2: Minimal OSC packet listener in Python with `pcap`[10]

Download executable source code of the following examples, to apply proxy and chorus effects to existing OSC packets: `https://archive.org/details/20160201-NIME-275`[2].

[2]Source code at `https://github.com/abramhindle/`

6.1 Chorus.py

Chorus.py demonstrates the ability to listen, retrieve, and replay OSC messages with a certain delay. Chorus.py can operate automatically with no user interaction. When listening to a network, upon detecting an OSC message (bundles are currently unsupported) on any UDP port, the Chorus.py program will resend that message with some optional delay. This functionally creates a chorus effect, similar to hitting a key twice in rapid succession.

To avoid infinite loops, chorus.py has to maintain a cache of recently sent packets such that it does not resend its own packets. Chorus.py has 2 main modes of operation, a spoof mode whereby the exact ethernet frame is duplicated on the network, and a no-spoof mode where by the packet is resent. Both modes are needed because user space programs do not necessarily receive the traffic that is injected — thus if one was running Chorus.py on a performance computer one would not see the spoofed packets appear on the network from the same computer.

Chorus.py is an example of executing macros with OSC messages where by performances could be computer augmented without prior knowledge of the instruments or hosts on the network.

It can also cause chaos if deployed upon the network of an unsuspecting performer whereby their old input will be repeated causing potential confusion.

6.2 Proxy.py

Proxy.py forwards observed OSC packets to a client, such as pure-data or SuperCollider, who then can manipulate a packet and send it back to the Proxy.py for re-transmission. What proxy.py does is it listens to the network, sniffing for OSC messages, and when it sees an OSC message it adds a destination string that packet, containing the host and port, and forwards the new message to the client. Clients will receive this modified OSC packet, manipulate the values and send the packet back to Proxy.py via OSC. The clients also have the option of overriding the destination, effectively bouncing OSC messages to other unsuspecting hosts.

Figure 3 shows the architecture of Proxy.py. The OSC packet source sends a packet that proxy.py observes on the network. Proxy.py then sends the message to the client OSC proxy client. The OSC proxy client manipulates the OSC packet and sends it back to the proxy.py proxy. Then the proxy.py proxy reads the destination from the OSC packet, removes the destination information, and sends the modified OSC packet to the OSC destination. This demonstrates how one could use a relatively interactive computer music language like pure-data or max/MSP to engage in hacking NIMEs by manipulating OSC packets.

Figure 4 shows how Proxy.py can manipulate OSC packets from an OSC client on a wireless Android tablet. In that example, the proxy.py pure-data patch rewrote the path of the OSC packet to use a different instrument altogether. [3]

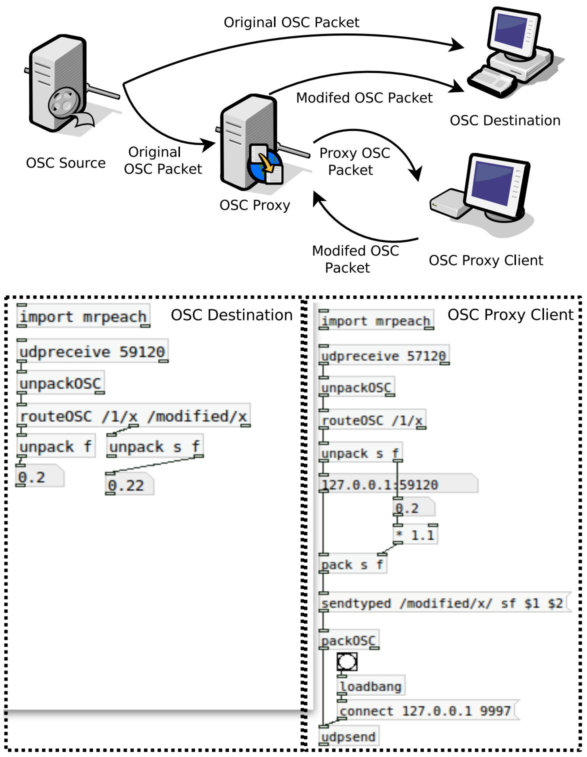
7. DISCUSSION

7.1 The Double Edged Sword

Open Sound Control is meant to be simple much like MIDI, but it is meant to address the power of modern computers with 32-bit processors and associated data-types. Wright's OSC is functional and beautiful in its simplicity. Yet OSC's simplicity unfortunately allows for unintended subversion. This subversion can frustrate and perplex clients of OSC but it also allows OSC to be used in a broadcast manner.

`hacking-nimes-src`

[3]See the video of this interaction at: `https://archive.org/details/20160201-NIME-275`



```
import mrpeach
udpreceive 59120
unpackOSC
routeOSC /1/x /modified/x
unpack f    unpack s f
0.2         0.22
```

```
import mrpeach
udpreceive 57120
unpackOSC
routeOSC /1/x
unpack s f
127.0.0.1:59120
0.2
* 1.1
pack s f
sendtyped /modified/x/ sf $1 $2
packOSC
loadbang
connect 127.0.0.1 9997
udpsend
```

Figure 3: An example of an OSC "proxy" architecture where OSC packets are intercepted by the proxy server. These packets are forwarded to a client who manipulates the OSC packets and sends them back to the proxy for re-transmission to the original OSC destination. The first PD patch displayed is the OSC destination patch. The second PD patch receives OSC packets from the proxy.py proxy and manipulates the values and the path of the packet, redirecting the path from /1/x to /modified/x and multiplying the first floating point value by 1.1. Consider how the proxy passes off an OSC packet to the client, then forwards the client traffic with the destination information removed.

Furthermore this simplicity enables us to build automated tools to manipulate transmitted OSC messages further enabling interesting performance opportunities.

The lack of security affordances of OSC cede to its improvisational affordances. One could guess that perhaps improvisation is likely many times more important to the NIME community than security, except in a few scenarios. This openness of OSC should be embraced as OSC packet manipulation lets one manipulate existing instruments rather than building totally new instruments to enable parameter manipulation. Some OSC instruments cannot be modified but packet manipulation opens an avenue of instrument modification and augmentation.

8. CONCLUSIONS AND FUTURE WORK

This work focused on co-opting network security tools, such as pcap [10] and scapy [4] into the service of computer music for mostly good purposes. We demonstrated that Open-Sound Control (OSC) messages are easy to capture and easy to manipulate. We demonstrated that these messages could be used positively for musical improvisation purposes. By integrating such a packet listening framework into an OSC service (proxy.py) we demonstrate that one could hack OSC

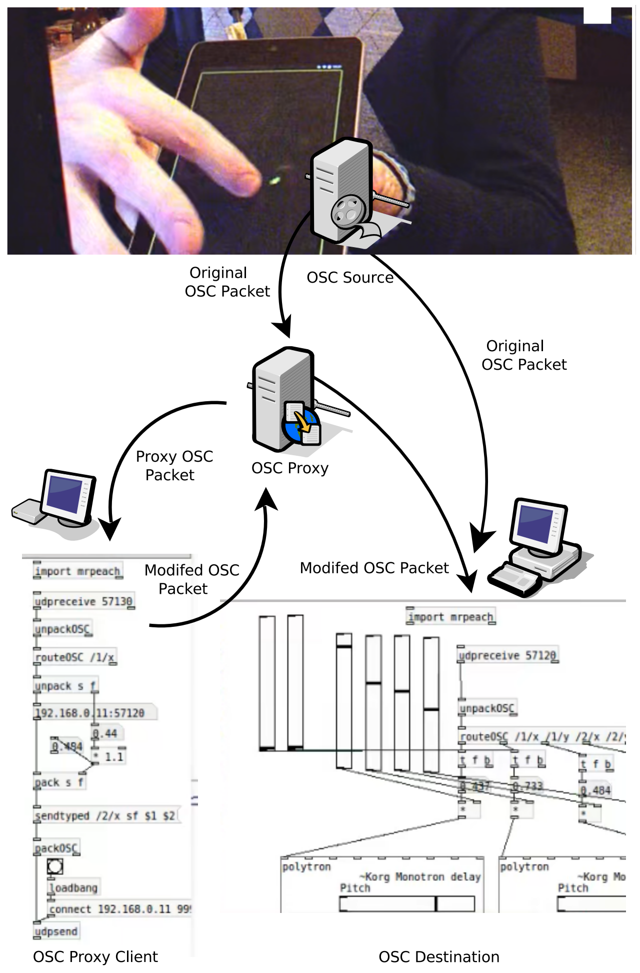


Figure 4: OSC messages from GoOSC on Android are sent to Proxy.py, the OSC Proxy. These OSC packets are sent to the OSC Proxy Client, a pure-data patch, that multiplies x-coordinate of the GoOSC pad widget by 1.1 and changes the path of the OSC message to send it to /2/x, another synthesizer depicted as the OSC destination. Essentially enabling a polyphonic response from remote OSC Messages.

packets live with tools such as pure-data. We named performance hacking as *live-hacking*, and provided suggestions for such performances.

Thus will live-hacking become a new kind of NIME performance? Future work includes incorporating "live-hacking" into NIMEs and networked computer music performances. Future work also includes extending the automated OSC packet manipulators to support Max/MSP `udpsend` format which is similar to OSC. In order to make this work more accessible to NIME practitioners a more coherent library of services should be proposed and designed. Future work should also explore the musicality of hacking tools such as Metasploit [15]. In this work we addressed network security, but hacking existing software systems to produce music via security exploits is an un-addressed area.

9. REFERENCES

[1] J. Allison. Distributed performance systems using html5 and rails. In *Proceedings of the 26th Annual Conference of the Society for Electro-Acoustic Music*, 2011.

[2] J. Allison, Y. Oh, and B. Taylor. Nexus: Collaborative performance for the masses, handling instrument interface distribution through the web. In *NIME*, 2013.

[3] Á. Barbosa. Displaced soundscapes: A survey of network systems for music and sonic art creation. *Leonardo Music Journal*, 13:53–59, 2003.

[4] P. Biondi. Scapy project, 2003. `http://www.secdev.org/projects/scapy/`.

[5] L. Dahl, J. Herrera, and C. Wilkerson. Tweetdreams: Making music with the audience and the world using real-time twitter data. In *International Conference on New Interfaces For Musical Expression*, Oslo, Norway, 2011.

[6] T. d'Otreppe. Aircrack-ng homepage. http://www.aircrack-ng.org.

[7] S. Ferguson, A. Martin, and A. Johnston. A corpus-based method for controlling guitar feedback. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 541–546, Daejeon, Republic of Korea, May 2013. Graduate School of Culture Technology, KAIST.

[8] A. Freed and AWS. Udp spoofing and dos attacks, 2005. `https://archive.org/details/UDPSpoofingWithOsc` originally from `http://opensoundcontrol.org/topic/42`.

[9] A. Hindle. Cloudorch: A portable soundcard in the cloud. *Proceedings of New Interfaces for Musical Expression (NIME), London, United Kingdom*, 2014.

[10] V. Jacobson, C. Leres, and S. McCanne. pcap-packet capture library. *UNIX man page*, 2001.

[11] S. Jordà. Multi-user Instruments: Models, Examples and Promises. In *NIME'05*, pages 23–26, 2005.

[12] S. W. Lee and G. Essl. Models and opportunities for networked live coding. *Live Coding and Collaboration Symposium 2014*, 1001:48109–2121, 2014.

[13] S. Levy. *Hackers: Heroes of the Computer Revolution - 25th Anniversary Edition*. O'Reilly Media, Inc., 1st edition, 2010.

[14] T. Mitchell, S. Madgwick, S. Rankine, G. Hilton, A. Freed, and A. Nix. Making the most of wi-fi: Optimisations for robust wireless live music performance. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 251–256, London, United Kingdom, June 2014. Goldsmiths, University of London.

[15] J. O'Gorman, D. Kearns, and M. Aharoni. *Metasploit: The penetration tester's guide*. No Starch Press, 2011.

[16] J. Oh and G. Wang. Audience-participation techniques based on social mobile computing. In *Proceedings of the International Computer Music Conference 2011 (ICMC 2011)*, Huddersfield, Kirkless, UK, 2011.

[17] R. Quintas. Glitch delighter : Lighter's flame base hyper-instrument for glitch music in burning the sound performance. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 212–216, Sydney, Australia, 2010.

[18] S. Smallwood, D. Trueman, P. R. Cook, and G. Wang. Composing for laptop orchestra. *Computer Music Journal*, 32(1):9–25, 2008.

[19] R. M. Stallman. On hacking. `https://stallman.org/articles/on-hacking.html`, 2010.

[20] B. Tome, D. Haddad, T. Machover, and J. Paradiso. Mmodm: Massively multipler online drum machine. In E. Berdahl and J. Allison, editors, *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 285–288, Baton Rouge, Louisiana, USA, May 31 – June 3 2015. Louisiana State University.

[21] D. Trueman. Why a laptop orchestra? *Organised Sound*, 12(02):171–179, 2007.

[22] G. Wakefield, C. Roberts, M. Wright, T. Wood, and K. Yerkes. Collaborative live-coding with an immersive instrument. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 505–508, London, United Kingdom, June 2014. Goldsmiths, University of London.

[23] N. Weitzner, J. Freeman, S. Garrett, and Y.-L. Chen. massMobile - an Audience Participation Framework. In *NIME'12*, Ann Arbor, Michigan, May 21-23 2012.

[24] M. Wright, A. Freed, and A. Momeni. Opensound control: State of the art 2003. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 153–159, Montreal, Canada, 2003.