CAPTCHAs fail to stop automated attacks

- CAPTCHA solvers employ human workers to solve CAPTCHAs.
- A botnet computer submits a CAPTCHA challenge using a program.
- Average latency: 11.80s; Accuracy: 90% (outsourcing to Antigate.com)

Observation: majority of exploits do not escalate to root

Compromising OS requires root (privileged) access

Observation:
- Only around 1/3 Android malware uses root exploits [1].
- Less than 50% of the botnets with top malicious values [2] may compromise trusted system programs.
- The fraction of vulnerabilities in CVE database containing root privilege escalation is decreasing.

Connecting clients behind NAT

Multiplex an existing TCP connection
1. Web server inserts special tags around CAPTCHA data and sends it.
2. A kernel packet filter on the client side strips the CAPTCHA data and passes it to a trusted CAPTCHA renderer.

Compatibility with legacy clients without kernel packet filter
1. Place CAPTCHA data as a JS string variable inside an HTML file. The string will be empty if packet filter is installed.
2. Use JS code to display the a CAPTCHA image inside the browser.

Performance
- Server side needs to escape special CAPTCHA tags.
- Client side needs to strip the CAPTCHA data

Experiments on a machine with core i7-860, 8GB RAM

Preventing screenshots of a CAPTCHA
- A Compositor (e.g., X11server) is a program that allocates screen buffer, organizes windows of applications and provides screenshot API.
- The CAPTCHA renderer notifies the compositor to disable the screenshot API when a CAPTCHA is being shown on the screen.

Preserving traditional CAPTCHA interface
- A traditional CAPTCHA is shown on the browser and submits the solution with HTML form.
- Compositor places the renderer window inside the browser window to make the CAPTCHA appeared to be rendered inside the browser.

References