Netgator: Malware Detection Using Program Interactive Challenges

Brian Schulte, Haris Andrianakis, Kun Sun, and Angelos Stavrou

GEORGE MASON UNIVERSITY

Intro

- Increase of stealthy malware in enterprises
  - Obfuscation, polymorphic techniques
- Often uses legitimate communication channels
  - HTTP
    - Volume of traffic makes it difficult to process all communications
  - HTTPS
    - Lack of inspection currently
  - Disguised as legitimate applications

Intro

- Netgator
  - Inspection of legitimate ports/protocols
    - Port 80, HTTP/S
  - Transparent proxy
  - 2 parts
    - Passive
      - Determine type of application
      - Easily catch “dumb” malware
    - Active
      - Challenge based on expected functionality (PICs)

Intro

- Focus on HTTP/S, browsers

- Study of 1026 malware samples
  - Out of samples where network activity was observed, ~80% utilized HTTP/S
  - Very high percentage of HTTP/S malware try to masquerade as browsers
  - None passed our challenges

Intro

- PIC
  - Challenge comprised of a request and expected response pair
  - Communication intercepted
  - Response it sent back to exercise known functionality of advertised program
  - If expected answer is returned, communication is allowed to pass through
    - If not, drop connection

Intro

- 2 pronged approach
  - Passive to classify traffic
  - Active to “challenge” application
- Prototype built using HTML, Javascript, and Flash challenges
- Low overhead
  - 353 ms end-to-end latency

Design and Implementation

- 2 major parts
  - Passive
  - Active

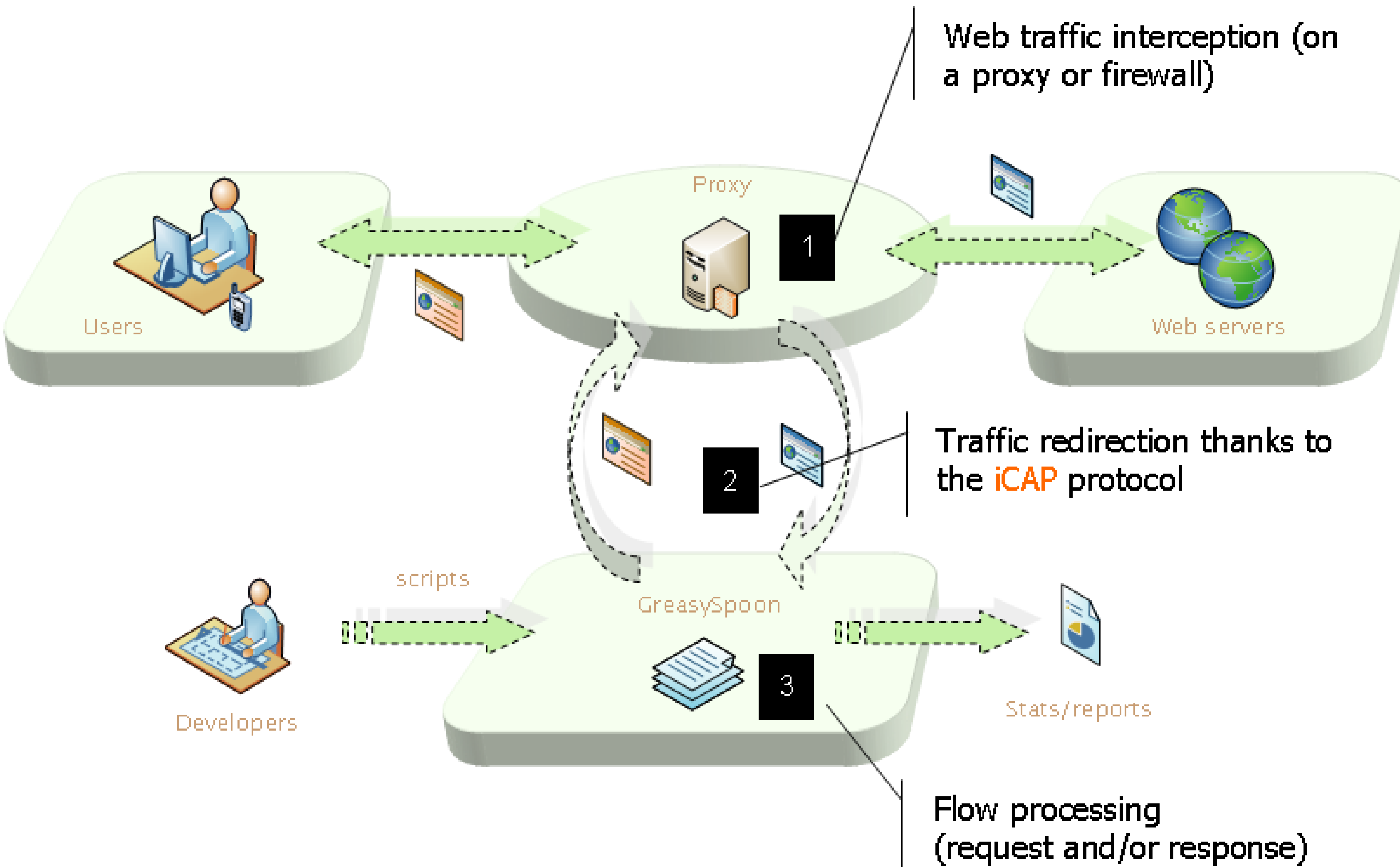
- Passive
  - Establish type of application
    - Browser, VOIP, OS updates, etc…
  - Signatures are determined by unique HTTP header orderings

Active Challenge Architecture

- Proxy & ICAP server duo
  - Squid, HTTP/S transparent proxy
  - Greasyspoon, Java based ICAP server

- What is ICAP?
  - Internet Content Adaption Protocol
  - Allows modification of all elements of HTTP request/response
    - Body, headers, URL, etc…

Active Challenge Architecture



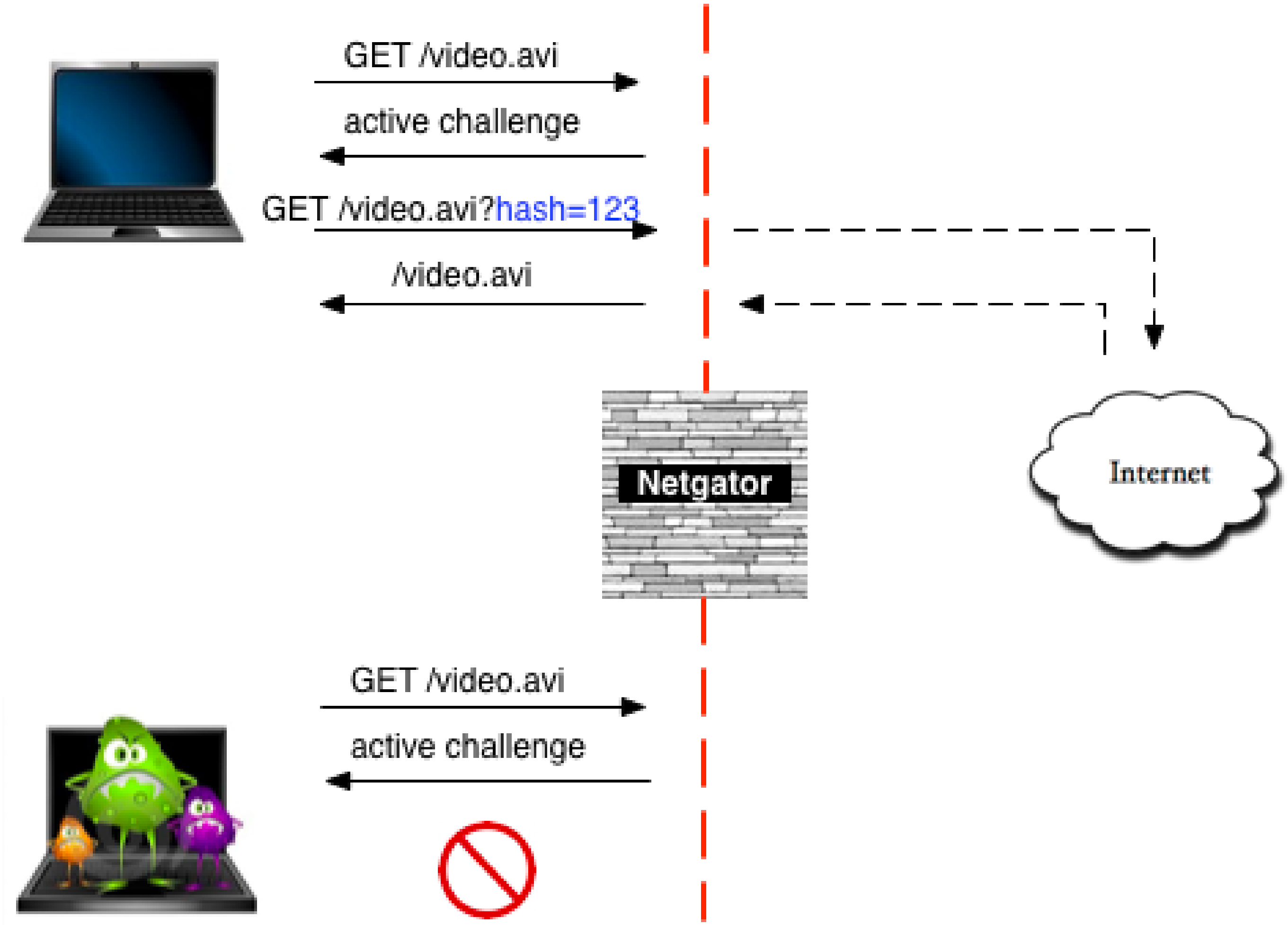
Active Challenges

- For known applications, we challenge them based on known functionality
  - For browsers, HTML/Flash/Javascript
- Challenge code comprised of a redirect to the originally requested file with a hash appended as a parameter
- To cut down on overhead, text/html data is challenged on the response

Active Challenges

- Two types
  - Request
  - Response
- Request challenging
  - Stop the initial communication
  - Send back challenge immediately
  - Higher latency, good protection
- Response challenging
  - Allow original response to come back
  - Imbed challenge in original response
  - Lower latency, possibly lower security

Active Challenges – Request Challenge



Active Challenges – Request challenging

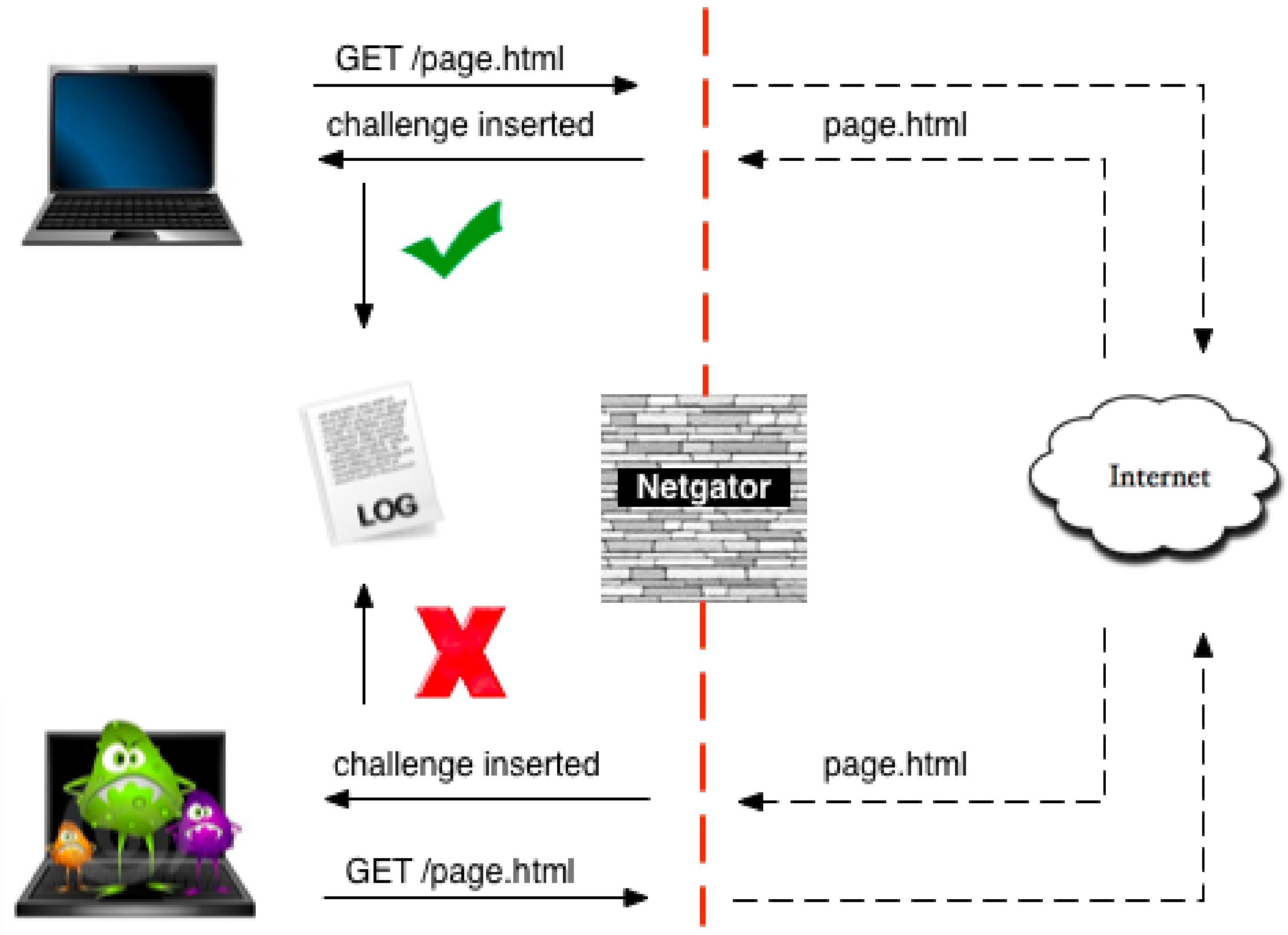
- Hash is unique each time
  - Based on time, requesting IP, requested URL, and secret key
- Headers replaced with HTTP response headers
  - Forces the new response back to the client
- Challenge code example, Javascript:

```
<html>
<head>
<script type="text/javascript">
window.location = {URL requested}?=\
                                    {hash generated}
</script>
</head>
<body></body>
</html>
```

Active Challenge – Response Challenge

- Challenging every request at the request would cause a lot of overhead
  - Challenge text/html data at the response
- Let the original request pass through
  - Insert challenge inside the original response
- Client gets response and then challenge is processed

Active Challenge – Response Challenge



Active Challenges

- The hash is what tells the proxy if the application passed the challenge
  - Attacker can just parse out hash
- Encrypt the hash with a Javascript implementation of AES
- The challenge that is sent back now contains the code (and key) to decrypt the hash
  - Forces the attacker to have a full Javascript engine to decrypt the hash

Active Challenges – Handling SSL

- Squid's SSL-bump utilized

- Traffic encrypted with Netgator's key
  - Decrypted at proxy for processing
  - Re-encrypted with external site's key when leaving proxy

Active Challenges

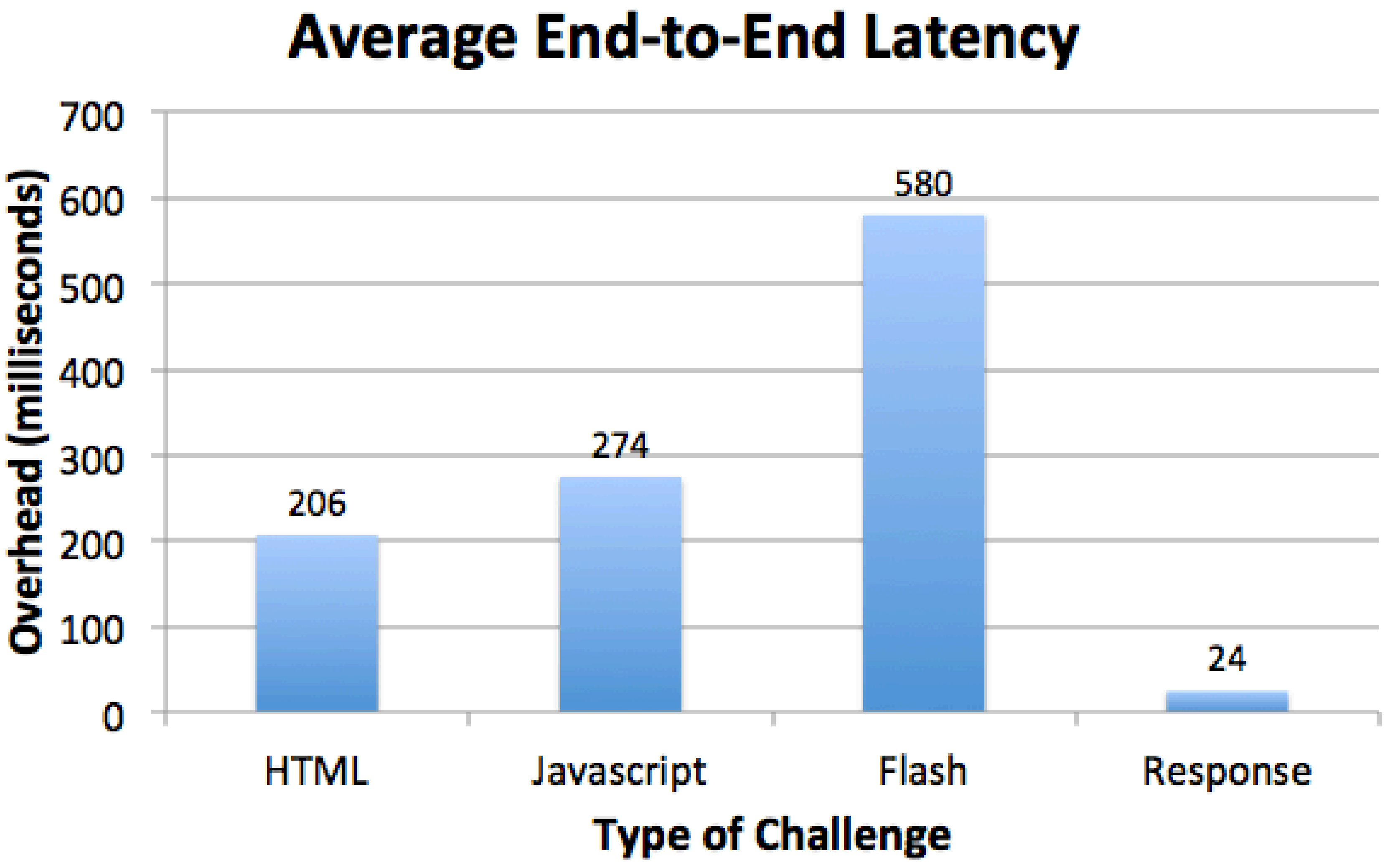
- Further cutting down on overhead
  - Automatically pass network requests if the client has passed a challenge for that site’s domain
- Client has passed challenge for www.foo.com
  - Request for www.foo.com/bar passes automatically
- Records are periodically cleaned
  - Avoid malware “piggy-backing” off legitimate client’s who passed challenges

Experimental Evaluation

- Used PlanetLab nodes for download tests
- Downloads of 3 different file sizes
  - 10KB, 100KB, 1MB
- 3 challenges types
  - HTML, Javascript, Flash
- Request and Response challenging

GEORGE MASON UNIVERSITY

Experimental Evaluation



Experimental Evaluation

- HTML lowest overhead
- Javascript results
  - Nice middle ground between difficulty to pass challenge and measured overhead
- Flash results
  - Highest overhead
  - Toughest challenge, combines Javascript and Flash
- Response challenge results
  - By far the lowest, lower security though since the original response is let through

Discussion

- Attackers will attempt evasion
  - Using a different user-agent/header signature
    - If unknown, communications are blocked
    - If known, challenge will still be sent
- Some legitimate applications might not be able to have challenges crafted
  - Whitelist can be created

Related Works

- Closest to our work is work by Gu et al.
  - Active botnet probing to identify obscure command and control channels
- Main differences
  - We do not expect nor ever rely on a human to be behind an application’s communications
  - Our work focuses on legitimate applications rather than malicious botnets

Related Works

- Our work similar to OS and application fingerprinting
  - Nmap
- CAPTCHA puzzles
  - Instead of focusing on humans, focus on the application
- Traditional botnet detection
  - BotSniffer, BotHunter, BotMiner

Conclusion

- Netgator
  - Inline malware detection system
  - 2 parts
    - Passive to classify traffic and thwart “dumb” malware
    - Active to challenge applications identity
      - Program Interactive Challenges
  - Fully transparent to the user
  - Average latency
    - 353ms for request challenges
    - 24ms for response challenges