OWASP 2021 Virtual APPSEC

PRESENTED BY:
Erik Elbieh, Security Consultant & Researcher
Palindrome Technologies

We’re not in HTTP anymore: Investigating WebSocket Server Security

Talk Summary

1. How WebSockets Work
2. Summary of WebSockets Research
3. New STEWS tool(s)

Erik Elbieh’s Brief Bio

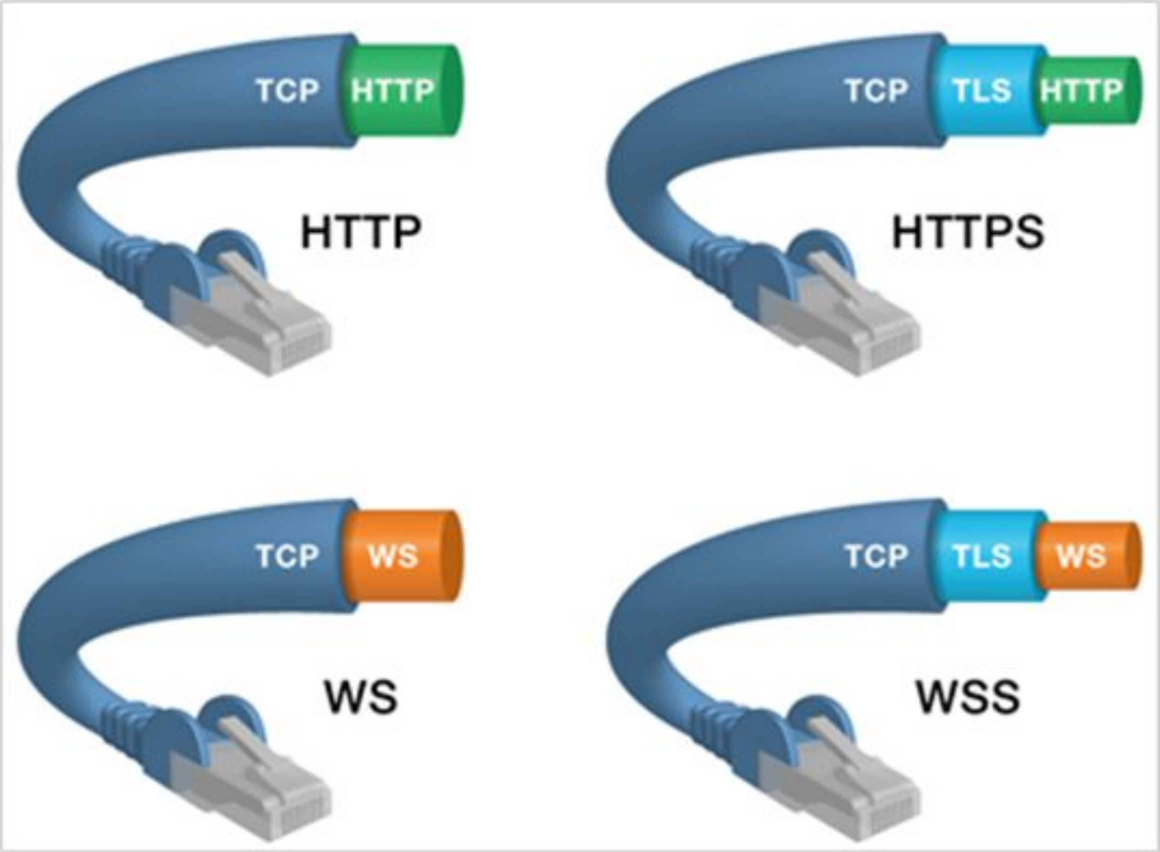
- Security Researcher and Consultant at Palindrome Technologies
  - Pen testing telecom systems, web apps, Kubernetes, and more
- Previously a Security Engineer at General Motors
  - Secured vehicle modules, Bluetooth specialist
- OSCP certified since 2019
- Graduated from Columbia University and Bard College at Simon’s Rock
- More at erikelbieh.com

Part 1: How WebSockets Work

WebSocket Protocol History

- Created in 2010-2011 (RFC6455)
- Provides a low-overhead web protocol for real-time communications
- WebSocket servers are often distinct from HTTP servers

WebSocket vs. HTTP



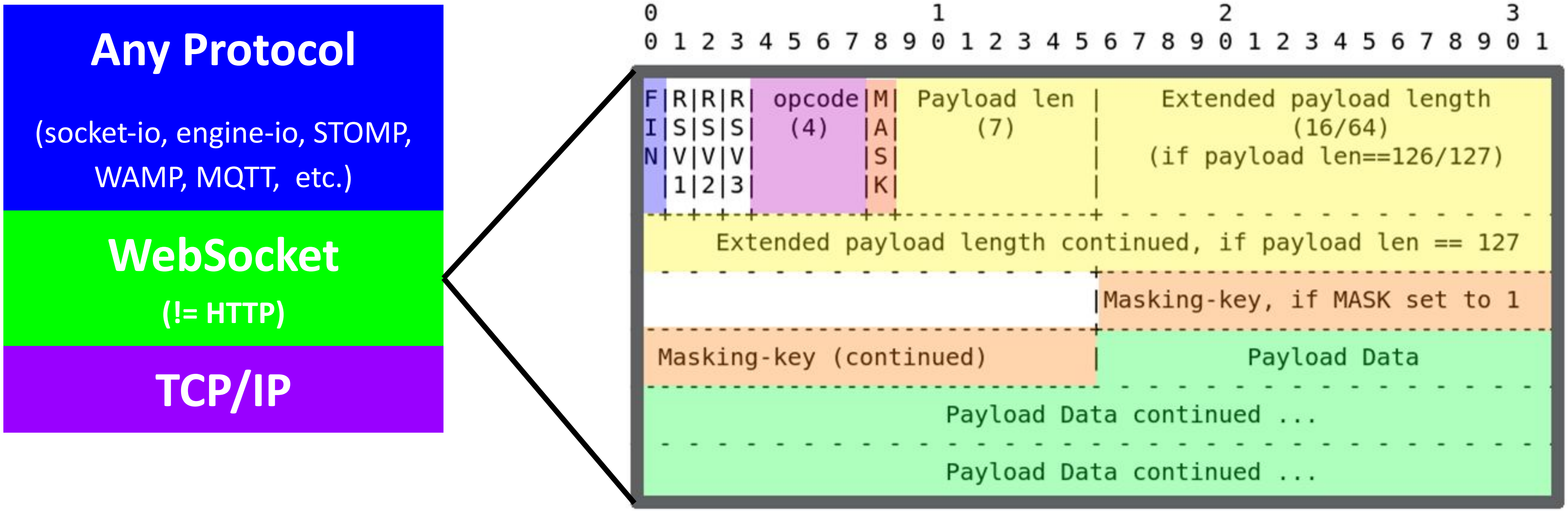
Source: https://developerinsider.co/difference-between-http-and-http-2-0-websocket/

WebSocket vs. HTTP

- WebSockets don’t use the request/response approach that HTTP does. WebSockets remain open until closed. This allows webpage updates to happen without refreshing the webpage (alternative to XHR, etc.)
  - Note: Proxies are usually built for the request/response approach HTTP uses and can have WebSockets vulnerabilities
- HTTP has headers (AKA overhead) with every request/response, but after a WebSocket is started, there is no similar header. Lower overhead is good for frequent back-and-forth real time communication.

WebSocket Stack

WebSocket Frame



Any Protocol

(socket-io, engine-io, STOMP, WAMP, MQTT, etc.)

WebSocket

(!= HTTP)

TCP/IP

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

F|R|R|R| opcode|M| Payload len |    Extended payload length
I|S|S|S|  (4)  |A|     (7)     |             (16/64)
N|V|V|V|       |S|             |   (if payload len==126/127)
 |1|2|3|       |K|             |
-+-+-+-+-------+-+-------------+ - - - - - - - - - - - - - - -
     Extended payload length continued, if payload len == 127
 - - - - - - - - - - - - - - - +-------------------------------
                               |Masking-key, if MASK set to 1
-------------------------------+-------------------------------
 Masking-key (continued)       |          Payload Data
-------------------------------- - - - - - - - - - - - - - - -
                     Payload Data continued ...
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                     Payload Data continued ...
```

WebSockets Higher-Level Protocols

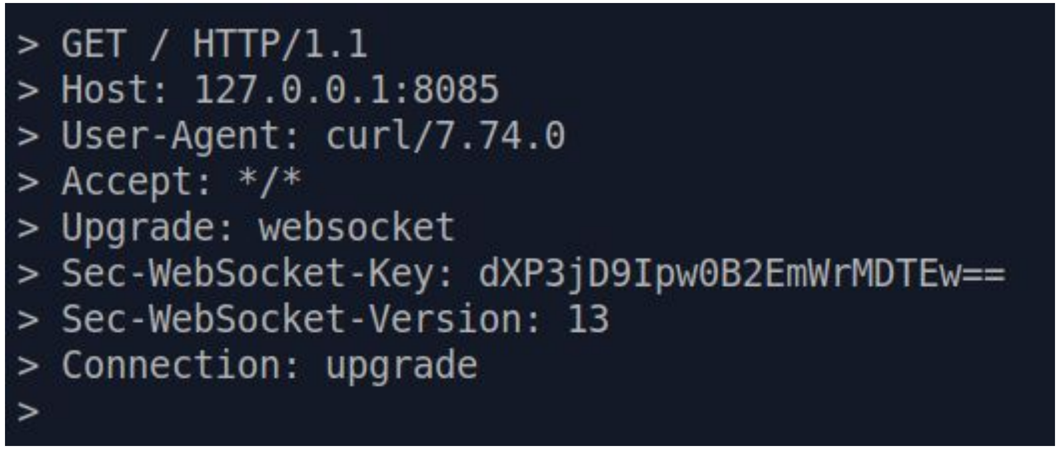
- Some protocols are (or can be) implemented on top of WebSockets:
  - Socket.io
  - Engine.io
  - STOMP
  - WAMP
  - MQTT

WebSocket Example: Phase 1

Key Point: WebSockets use HTTP to “kickstart” the WebSocket protocol

Step 1: HTTP request from browser

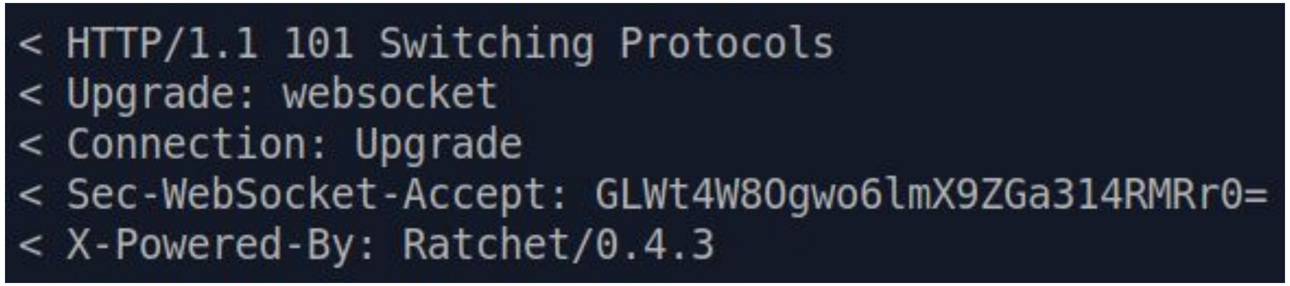
(Note the many uses of the word “WebSocket”)



```
> GET / HTTP/1.1
> Host: 127.0.0.1:8085
> User-Agent: curl/7.74.0
> Accept: */*
> Upgrade: websocket
> Sec-WebSocket-Key: dXP3jD9Ipw0B2EmWrMDTEw==
> Sec-WebSocket-Version: 13
> Connection: upgrade
>
```

Step 2: HTTP response from server

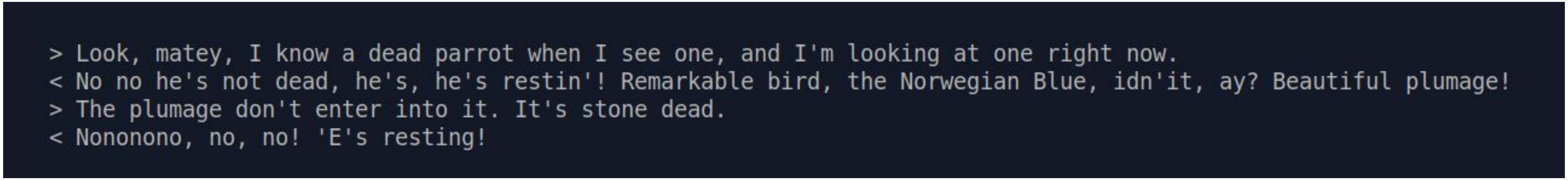
“101 Switching Protocols” is a ‘rare’ HTTP status code that often indicates a WebSocket was started



```
< HTTP/1.1 101 Switching Protocols
< Upgrade: websocket
< Connection: Upgrade
< Sec-WebSocket-Accept: GLWt4W8Ogwo6lmX9ZGa314RMRr0=
< X-Powered-By: Ratchet/0.4.3
```

WebSocket Example: Phase 2

Not much to see because the WebSocket
Protocol focuses on minimizing overhead.
Chat application example shown below



```
> Look, matey, I know a dead parrot when I see one, and I'm looking at one right now.
< No no he's not dead, he's, he's restin'! Remarkable bird, the Norwegian Blue, idn'it, ay? Beautiful plumage!
> The plumage don't enter into it. It's stone dead.
< Nononono, no, no! 'E's resting!
```

WebSockets in the Wild

Use cases include:

- Chat bots, especially customer service
- Slack, Discord, and other chat platforms
- Maps tracking real-time movement
- Live finance data websites
- Cryptocurrency websites
- Smart TV remote control!?
- Kubernetes/Docker API!?

Try This at Home Kids!

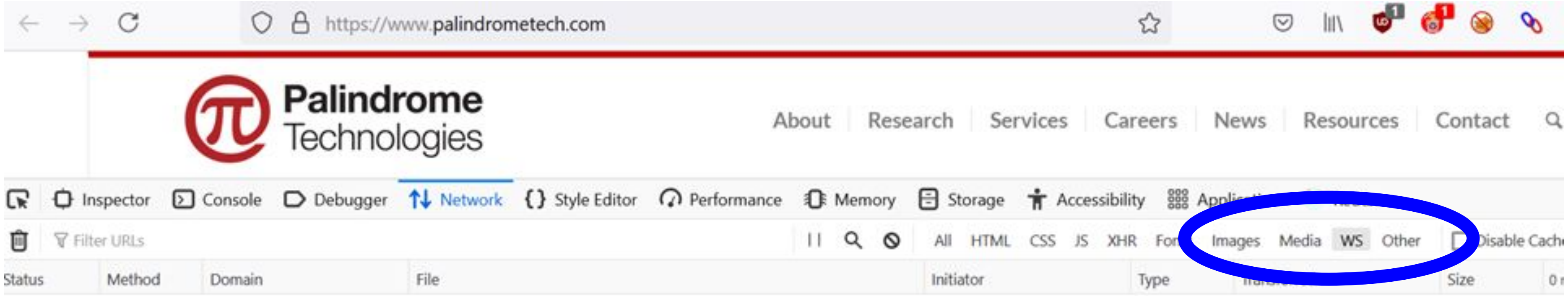
Try This at Home Kids!

1. Open web browser developer tools (Control+Shift+I in Firefox or Chrome) and visit the Network tab
2. Click "WS" to filter for only WebSockets traffic
3. Visit a webpage with WebSockets, such as:
    a. Finance: https://finance.yahoo.com/
    b. Sports: https://www.livescore.in/
    c. Chat: https://support.zoom.us
    d. Live maps: https://www.marinetraffic.com
4. Observe initial WebSocket request and response

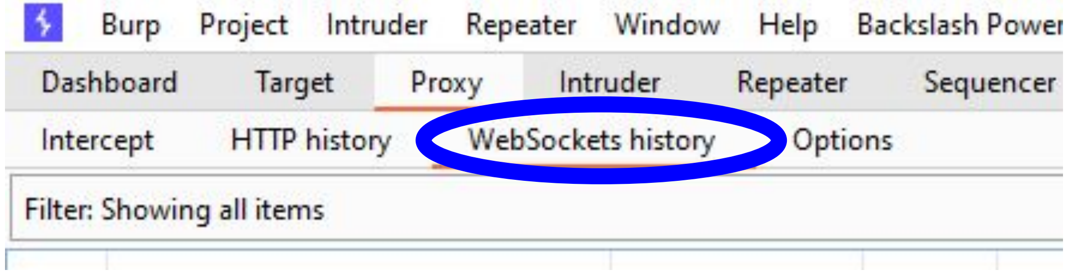
Note: Web proxy tools like Burp Suite and OWASP ZAP store WebSocket traffic in a separate tab from HTTP traffic

Finding WebSockets

Firefox



Burp Suite

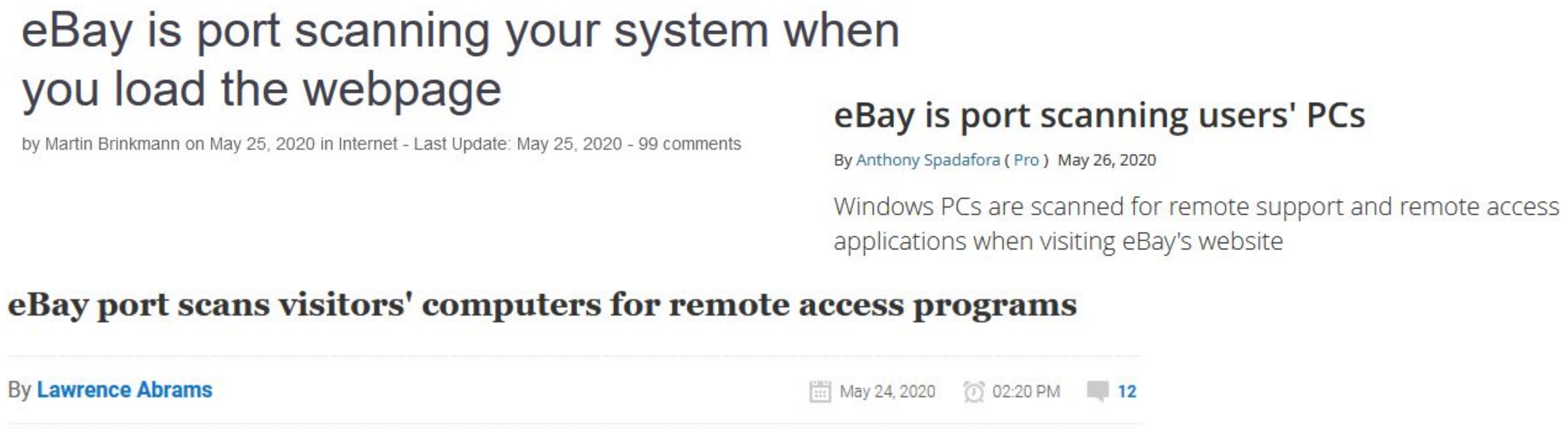


Part 2: Summary of WebSockets Research

Highlights of Prior WebSockets Security Research

- 2011: Firefox 4 temporarily removes WebSocket support due to protocol issue
- 2016: SOP, a HTTP CSRF mitigation, doesn't apply to WebSockets -> Cross Site WebSocket Hijacking (CSWSH)
- 2019: Proxies that don't properly handle WebSockets can lead to WebSocket Smuggling

Port Scanning with WebSockets



eBay is port scanning your system when you load the webpage

by Martin Brinkmann on May 25, 2020 in Internet - Last Update: May 25, 2020 - 99 comments

eBay is port scanning users' PCs

By Anthony Spadafora (Pro) May 26, 2020

Windows PCs are scanned for remote support and remote access applications when visiting eBay's website

eBay port scans visitors' computers for remote access programs

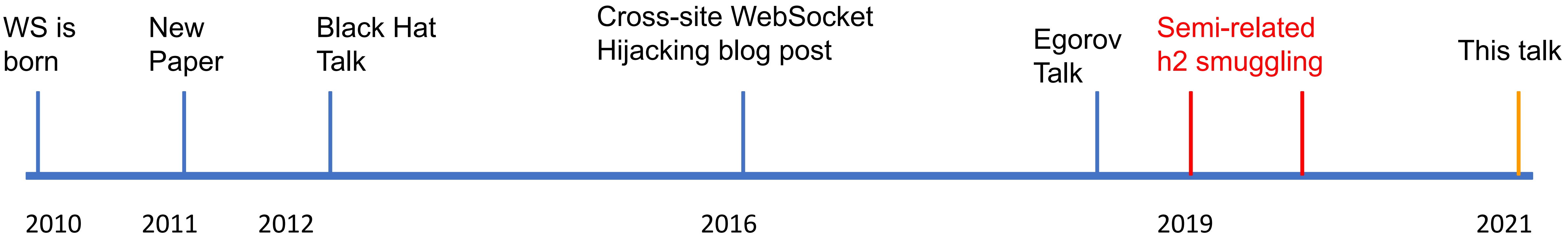
By Lawrence Abrams May 24, 2020 02:20 PM 12

Related slide deck: https://datatracker.ietf.org/meeting/96/materials/slides-96-saag-1

Port Scanning and WebSockets

Tom Gallagher
NSA Information Assurance
tmgall4@empire.eclipse.ncsc.mil

Timeline of Prior Related Research



WS is born

New Paper

Black Hat Talk

Cross-site WebSocket Hijacking blog post

Egorov Talk

Semi-related h2 smuggling

This talk

2010

2011

2012

2016

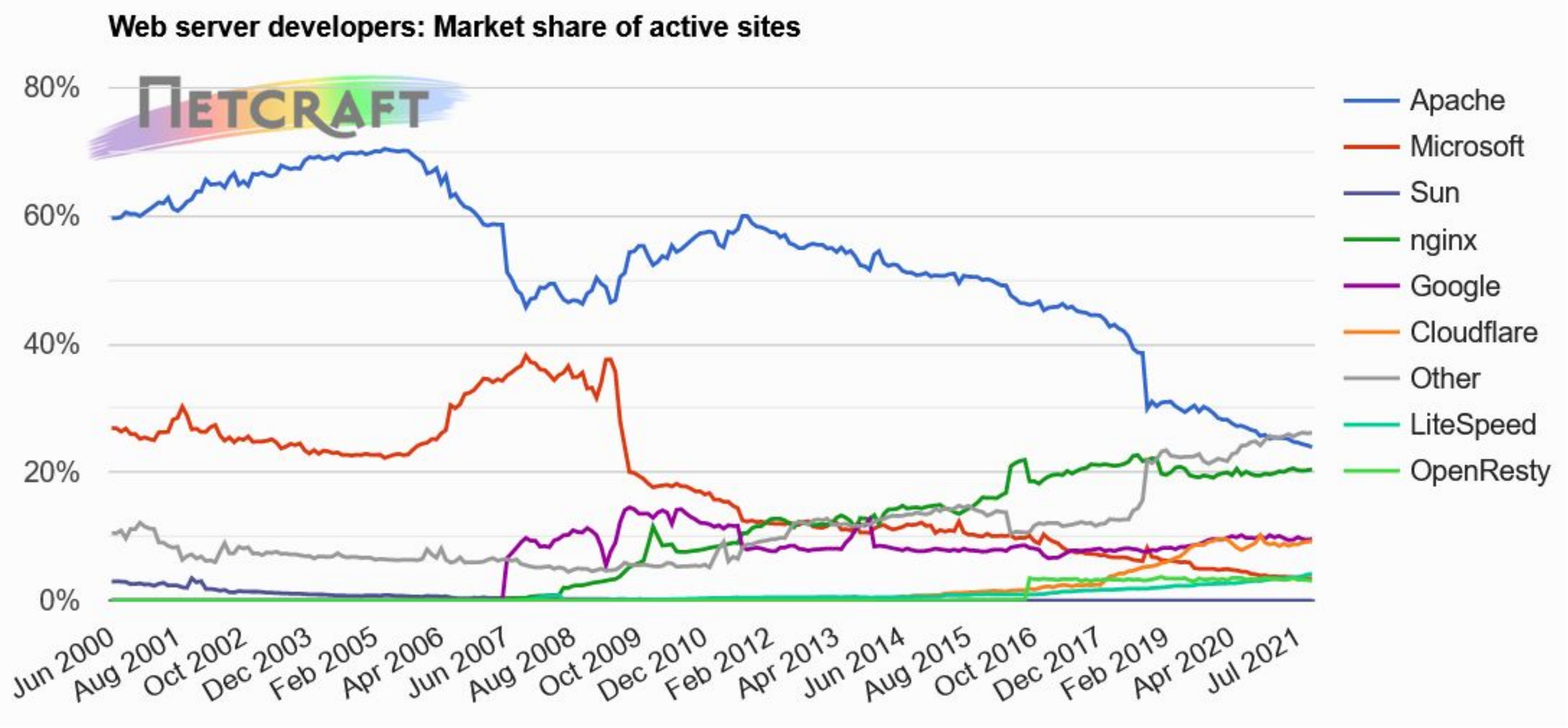
2019

2021

Takeaways from Past Research

- Large scale security testing of WebSockets “in the wild” hasn’t been publicly done before
- Research has been focused on the protocol level and proxy (mis)handling - but what about the server implementations?
- HTTP gets all the attention

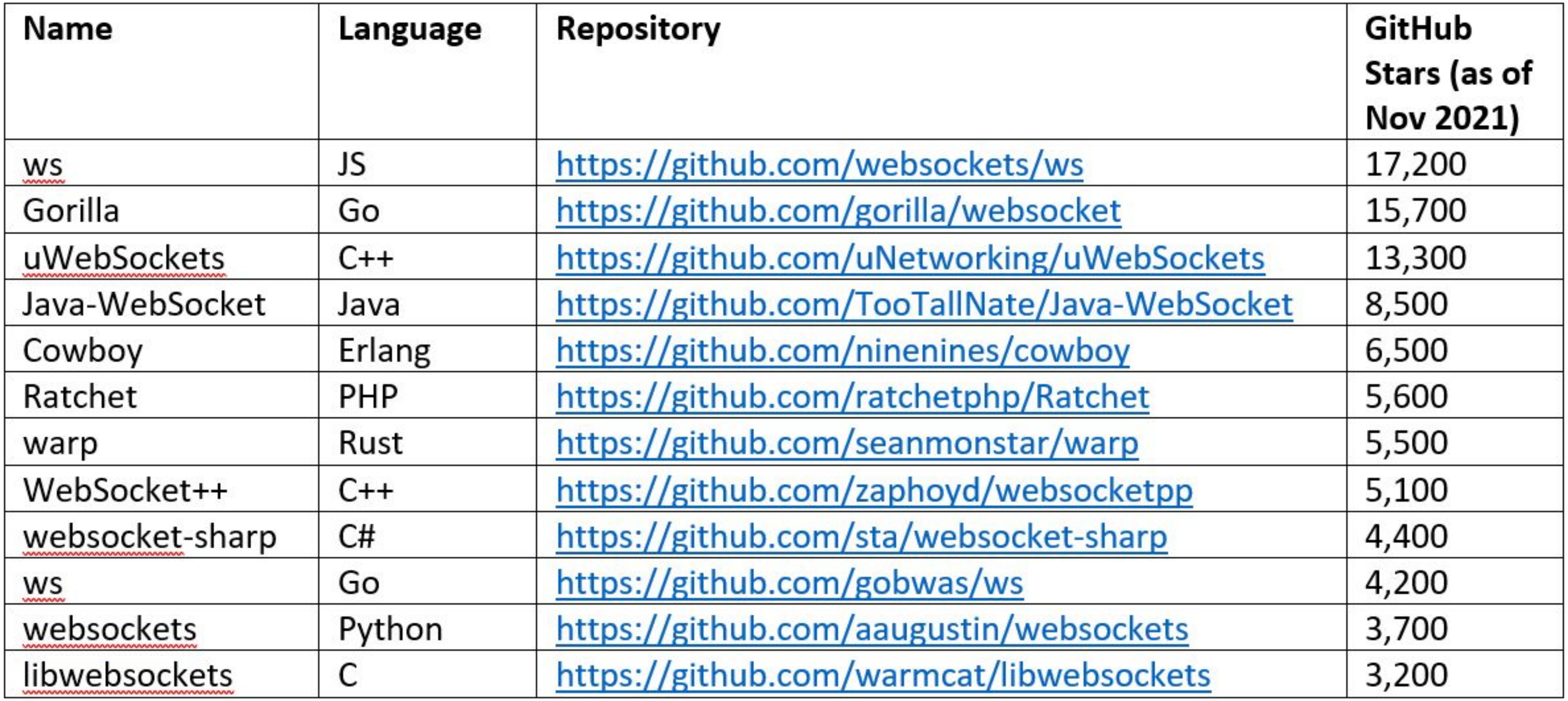
HTTP Servers Market share



Source: https://news.netcraft.com/archives/2021/10/15/october-2021-web-server-survey.html

WebSocket Servers Market share

Common WebSocket Server Implementations



Name	Language	Repository	GitHub Stars (as of Nov 2021)
ws	JS	https://github.com/websockets/ws	17,200
Gorilla	Go	https://github.com/gorilla/websocket	15,700
uWebSockets	C++	https://github.com/uNetworking/uWebSockets	13,300
Java-WebSocket	Java	https://github.com/TooTallNate/Java-WebSocket	8,500
Cowboy	Erlang	https://github.com/ninenines/cowboy	6,500
Ratchet	PHP	https://github.com/ratchetphp/Ratchet	5,600
warp	Rust	https://github.com/seanmonstar/warp	5,500
WebSocket++	C++	https://github.com/zaphoyd/websocketpp	5,100
websocket-sharp	C#	https://github.com/sta/websocket-sharp	4,400
ws	Go	https://github.com/gobwas/ws	4,200
websockets	Python	https://github.com/aaugustin/websockets	3,700
libwebsockets	C	https://github.com/warmcat/libwebsockets	3,200

Part 3: New STEWS tool(s)

Who doesn't like free stuff?

Released today, fresh out of the oven!

1. STEWS repository: https://github.com/PalindromeLabs/STEWS
   a. Includes whitepaper and this slide deck
2. WebSockets Playground: https://github.com/PalindromeLabs/WebSocket-Playground
3. WebSockets Security Awesome: https://github.com/PalindromeLabs/awesome-websockets-security

Top Tools Lack WebSocket Custom Test Support

1. nmap: https://seclists.org/nmap-dev/2015/q1/134
2. Burp Suite (supports WebSockets, but not for extensions): https://forum.portswigger.net/thread/websockets-api-support-c8e1660b9f0ab
3. nuclei: https://github.com/projectdiscovery/nuclei/issues/539

STEWS

STEWS = Security Testing and Enumeration of WebSockets

Performs 3 key steps in WebSockets security testing:

1. Discovery
2. Fingerprinting
3. Vulnerability Detection

1. WebSockets Discovery

Why WebSocket endpoint discovery is difficult:

1. WebSockets use HTTP to start a connection, but observing HTTP alone does not indicate a WebSocket
2. Websites often start WebSockets using JavaScript, so WebSocket endpoints aren't always found parsing HTML
   a. Sometimes the main website is not linked to the WebSocket because the WebSocket endpoint is a standalone API
3. WebSockets may only exist at one specific URL path and at one specific port of the endpoint

1. WebSockets Discovery

Approaches to discovering WebSockets:

1. Finding WebSockets on a specific website
   a. Spider website HTML and search for WebSocket keywords in source code (downsides: false positives)
   b. Spider website and load all JavaScript and watch for HTTP 101 responses (downsides: loading all JS is slow)
2. Finding WebSockets on any website
   a. Use wordlist of common WebSocket endpoints and brute force a large list of websites (downsides: only testing wordlist endpoints)

1. WebSockets Discovery

Approaches to discovering WebSockets:

1. Finding WebSockets on a specific website
   a. Spider website HTML and search for WebSocket keywords in source code (downsides: false positives)
   b. Spider website and load all JavaScript and watch for HTTP 101 responses (downsides: loading all JS is slow)
2. Finding WebSockets on any website
   a. Use wordlist of common WebSocket endpoints and brute force a large list of websites (downsides: only testing wordlist endpoints)

Good for finding many WebSocket endpoints quickly

1. WebSockets Discovery

Difficulties in scalable WebSocket endpoint discovery:

1. Tools like masscan and zmap are fast at endpoint detection
   a. ...However, they work at the TCP/IP layer and we need to operate at the HTTP/WebSocket layer
2. Burp Suite’s Turbo Intruder is fast at the HTTP layer
   a. ...However, Turbo Intruder documentation states “it's designed for sending lots of requests to a single host”, not testing many hosts
3. ZGrab2 is a fast application-layer scanner
   a. ...However, requires some tweaks to support WebSocket requests

1. WebSockets Discovery

Acquiring large lists of URLs

1. Googling “Top million URLs”: https://www.letmegooglethat.com/?q=top+million+urls
2. Zone Files: https://czds.icann.org/home
   a. Zone Files are what DNS servers use for lookups
   b. Downside is that many URLs in zone file aren’t active

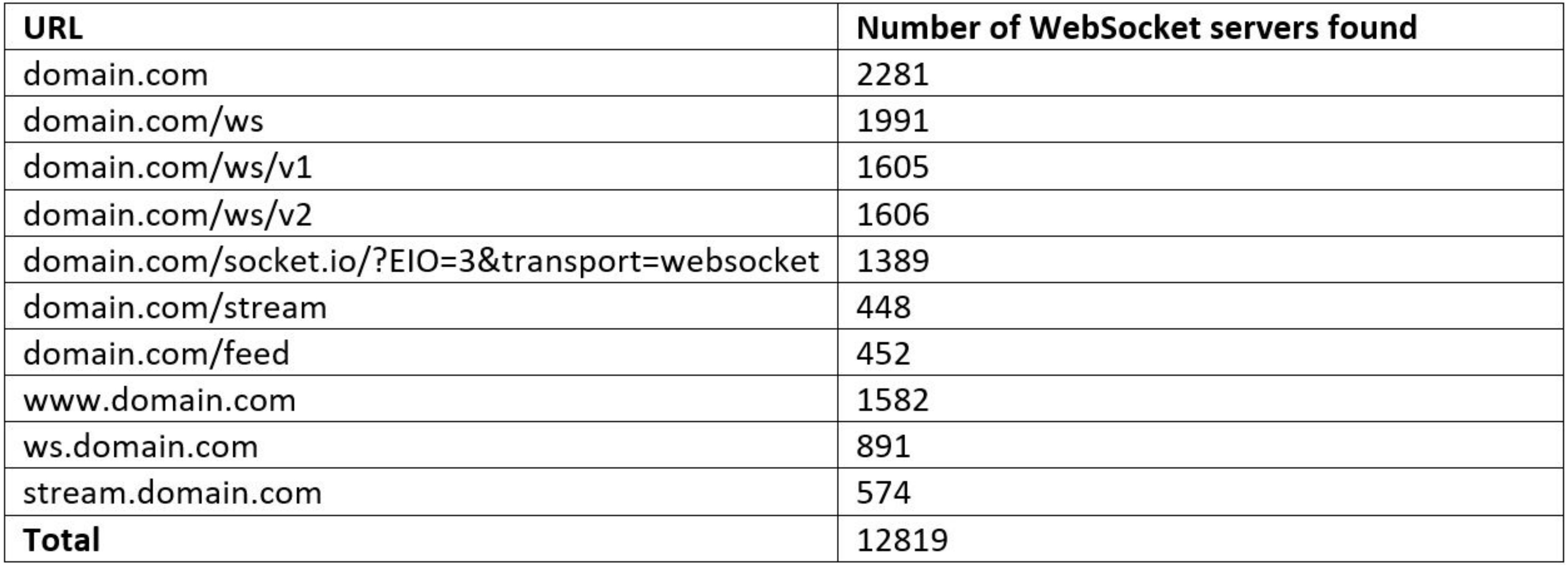
1. WebSockets Discovery

Other difficulties:

- Large number of DNS lookups can be a bottleneck
  - Many DNS servers have rate limit
  - Using multiple DNS servers can help solution
  - zgrab2 allows DNS lookup beforehand (using zdns, massdns, etc.)
- Obtaining wordlist of probable WebSocket paths to brute force requires manual effort
  - Found known WebSocket endpoints through random browsing, bug bounty reports, reading GitHub WebSocket repository issues

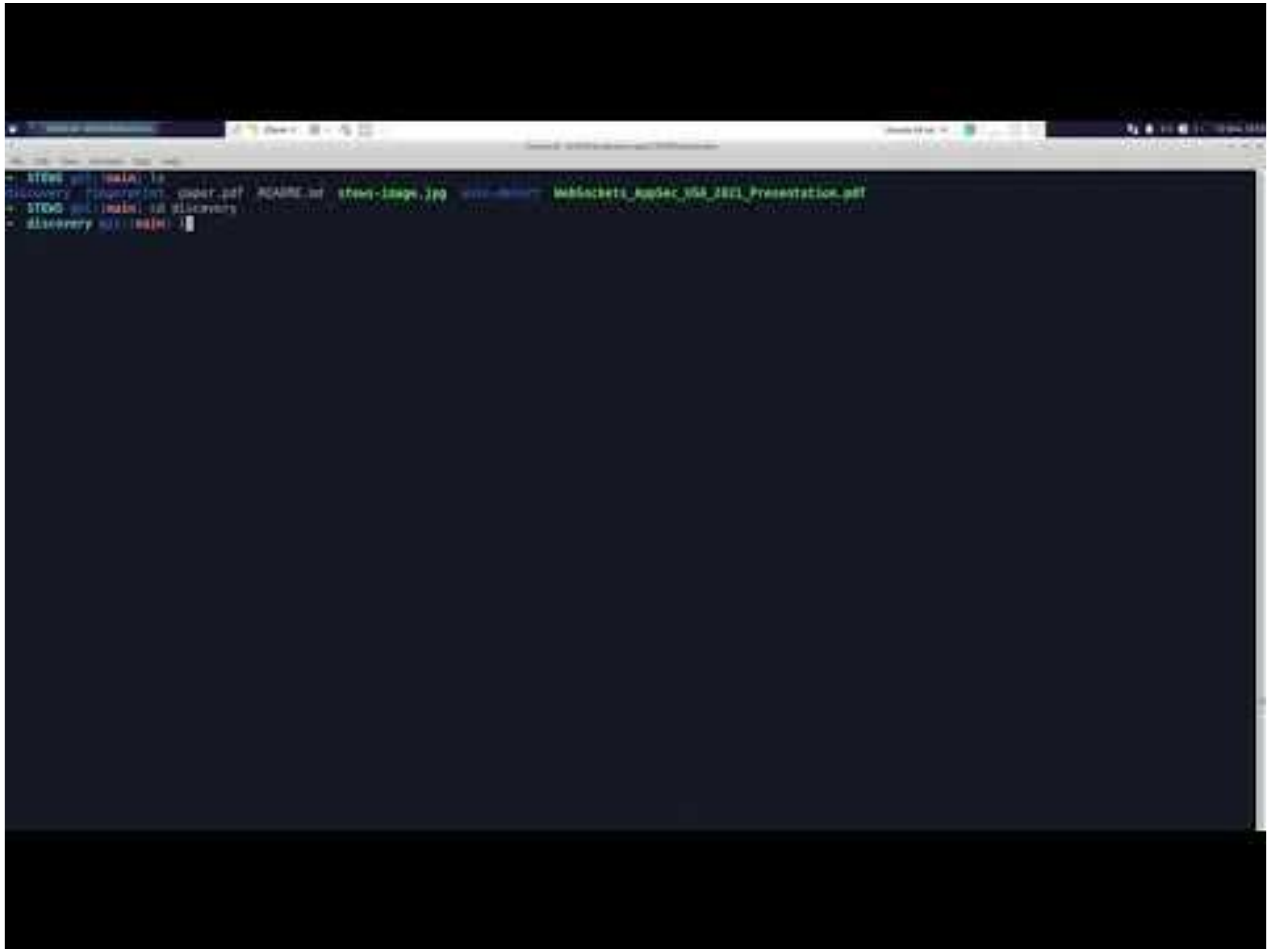
1. WebSockets Discovery

From ~3 million domains



URL	**Number of WebSocket servers found**
domain.com	2281
domain.com/ws	1991
domain.com/ws/v1	1605
domain.com/ws/v2	1606
domain.com/socket.io/?EIO=3&transport=websocket	1389
domain.com/stream	448
domain.com/feed	452
www.domain.com	1582
ws.domain.com	891
stream.domain.com	574
Total	12819

STEWS Discovery Demo



```
➜  STEWS git:(main) ls
discovery  fingerprint  paper.pdf  README.md  stews-image.jpg  vuln-detect  WebSockets_AppSec_USA_2021_Presentation.pdf
➜  STEWS git:(main) cd discovery
➜  discovery git:(main) 
```

2. WebSockets Fingerprinting

The challenge: to find implementation-level differences between WebSocket server implementations in order to identify them

"In theory there is no difference between theory and practice – in practice there is"

2. WebSockets Fingerprinting

A few of the most popular WebSocket servers include:

- uWebSockets (C++)
- Gorilla (Go)
- ws (JavaScript)
- websockets (Python)
- Spring Boot (Java)

But there’s dozens of WebSocket server implementations

2. WebSockets Fingerprinting

Differences from other fingerprinting tools:

- HTTP fingerprinters only handle 1 protocol, whereas WebSockets use HTTP to negotiate the switch to WebSockets, meaning STEWS fingerprinting handles 2 protocols
- Tools like nmap query specific URL paths to gain information, but WebSocket servers usually only listen at a specific URL path

2. WebSockets Fingerprinting

To find WebSocket server identifying features, use a simple deterministic fuzzer to test different features of the WebSocket Server, such as:

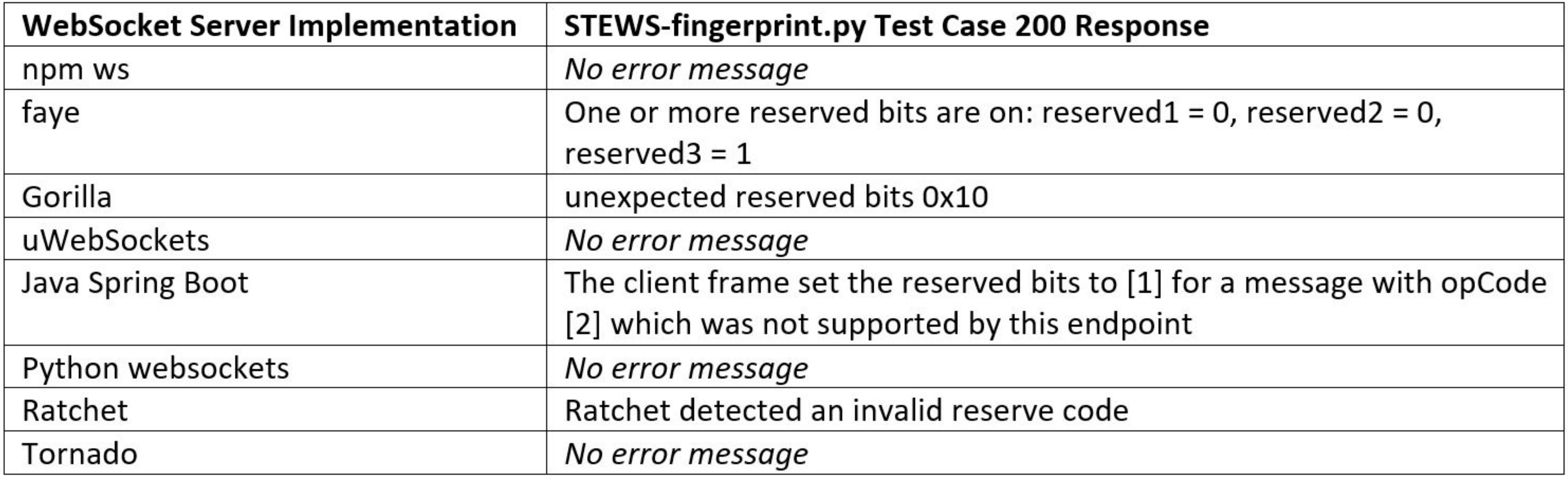
- Supported WebSocket Protocol Version Numbers
- Reserved and opcode bit support
- Verbose error messages
- Default maximum data length

2. WebSockets Fingerprinting

Over 50 different STEWS fingerprinting test cases:

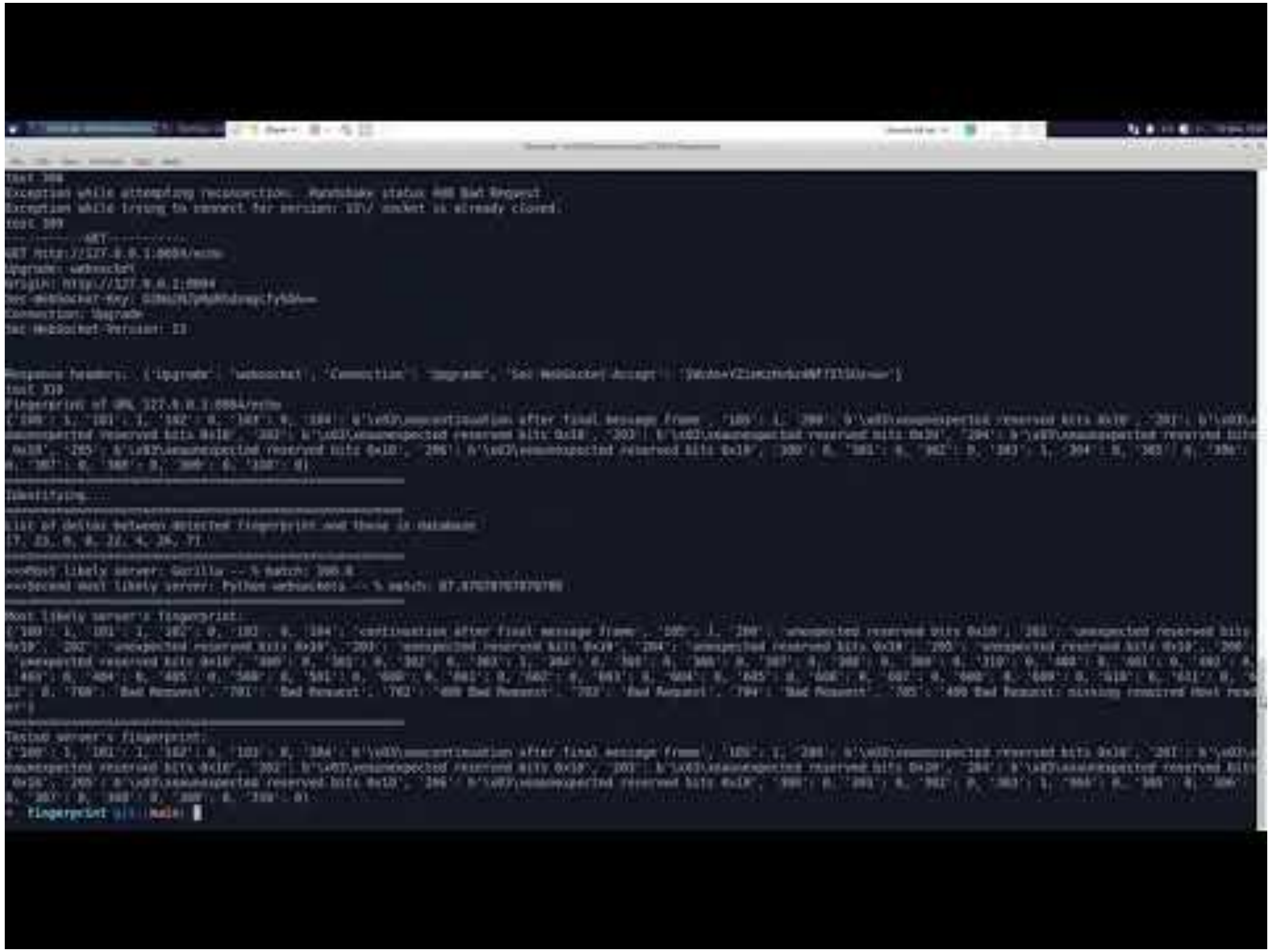
- 100-series tests: opcode tests (WebSocket protocol)
- 200-series tests: rsv bit tests (WebSocket protocol)
- 300-series tests: version tests (HTTP protocol)
- 400-series tests: extensions tests (HTTP protocol)
- 500-series tests: subprotocol tests (HTTP protocol)
- 600-series tests: long payload tests (WebSocket protocol)
- 700-series tests: hybi and similar tests (WebSocket protocol)

2. WebSockets Fingerprinting



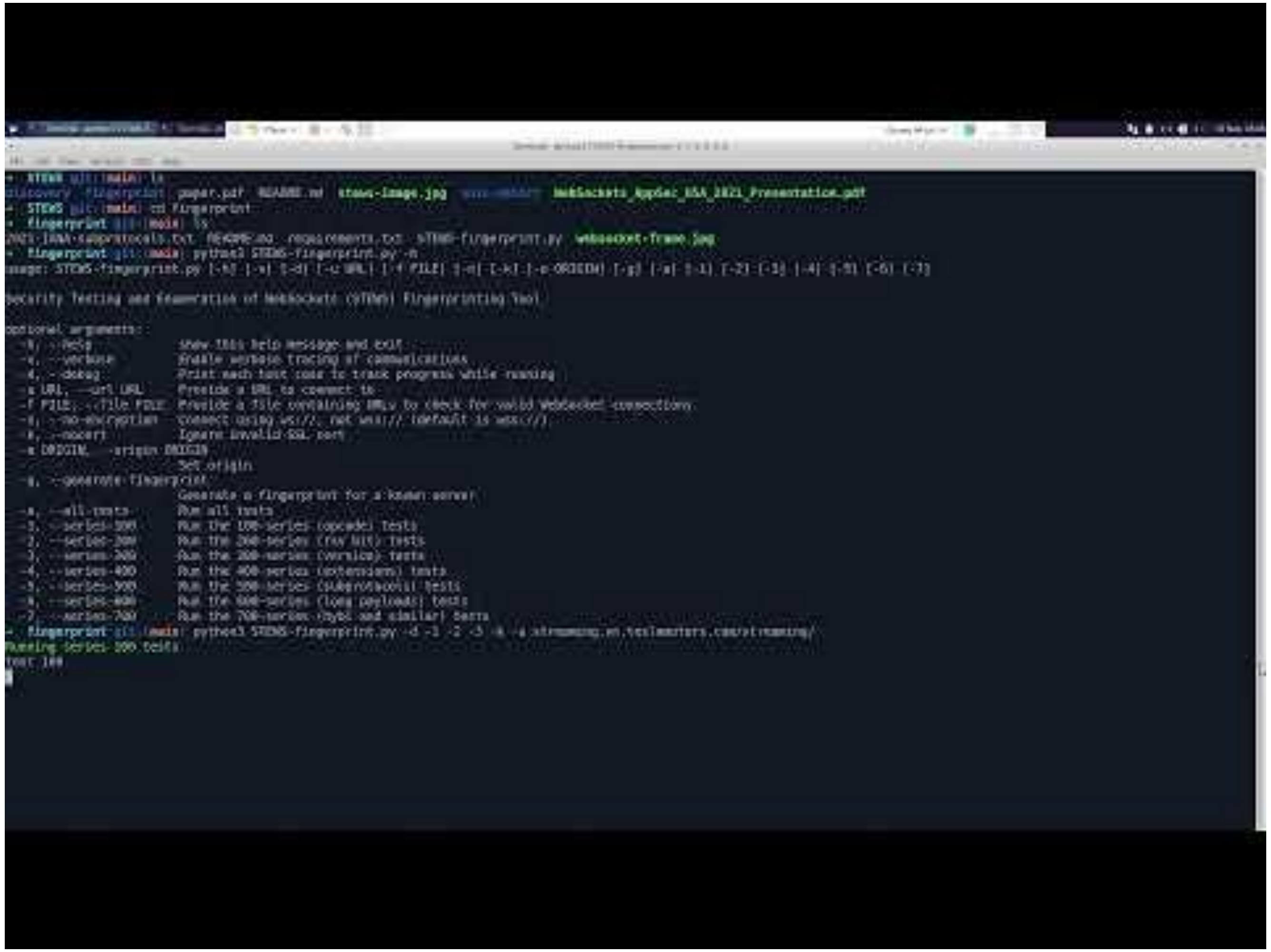
WebSocket Server Implementation	STEWS-fingerprint.py Test Case 200 Response
npm ws	*No error message*
faye	One or more reserved bits are on: reserved1 = 0, reserved2 = 0, reserved3 = 1
Gorilla	unexpected reserved bits 0x10
uWebSockets	*No error message*
Java Spring Boot	The client frame set the reserved bits to [1] for a message with opCode [2] which was not supported by this endpoint
Python websockets	*No error message*
Ratchet	Ratchet detected an invalid reserve code
Tornado	*No error message*

STEWS Fingerprint Local Server Demo



```
test 308
Exception while attempting reconnection: Handshake status 400 Bad Request
Exception while trying to connect for version: 13\/ socket is already closed.
test 309
----------- GET -----------
GET http://127.0.0.1:8084/echo
Upgrade: websocket
Origin: http://127.0.0.1:8084
Sec-WebSocket-Key: [illegible]
Connection: Upgrade
Sec-WebSocket-Version: 13


Response headers: {'Upgrade': 'websocket', 'Connection': 'Upgrade', 'Sec-WebSocket-Accept': '[illegible]'}
test 310
Fingerprint of URL 127.0.0.1:8084/echo
{'100': 1, '101': 1, '102': 0, '103': 0, '104': b'\x03\xeacontinuation after final message frame', '105': 1, '200': b'\x03\xeaunexpected reserved bits 0x10', '201': b'\x03\x
eaunexpected reserved bits 0x10', '202': b'\x03\xeaunexpected reserved bits 0x10', '203': b'\x03\xeaunexpected reserved bits 0x10', '204': b'\x03\xeaunexpected reserved bits
 0x10', '205': b'\x03\xeaunexpected reserved bits 0x10', '206': b'\x03\xeaunexpected reserved bits 0x10', '300': 0, '301': 0, '302': 0, '303': 1, '304': 0, '305': 0, '306':
0, '307': 0, '308': 0, '309': 0, '310': 0}
==================================================
Identifying...
==================================================
List of deltas between detected fingerprint and those in database
[7, 23, 0, 0, 22, 4, 26, 7]
==================================================
>>>Most likely server: Gorilla -- % match: 100.0
>>>Second most likely server: Python websockets -- % match: 87.87878787878788
==================================================
Most likely server's fingerprint:
{'100': 1, '101': 1, '102': 0, '103': 0, '104': 'continuation after final message frame', '105': 1, '200': 'unexpected reserved bits 0x10', '201': 'unexpected reserved bits
0x10', '202': 'unexpected reserved bits 0x10', '203': 'unexpected reserved bits 0x10', '204': 'unexpected reserved bits 0x10', '205': 'unexpected reserved bits 0x10', '206':
 'unexpected reserved bits 0x10', '300': 0, '301': 0, '302': 0, '303': 1, '304': 0, '305': 0, '306': 0, '307': 0, '308': 0, '309': 0, '310': 0, '400': 0, '401': 0, '402': 0,
 '403': 0, '404': 0, '405': 0, '500': 0, '501': 0, '600': 0, '601': 0, '602': 0, '603': 0, '604': 0, '605': 0, '606': 0, '607': 0, '608': 0, '609': 0, '610': 0, '611': 0, '6
12': 0, '700': 'Bad Request', '701': 'Bad Request', '702': '400 Bad Request', '703': 'Bad Request', '704': 'Bad Request', '705': '400 Bad Request: missing required Host head
er'}
==================================================
Tested server's fingerprint:
{'100': 1, '101': 1, '102': 0, '103': 0, '104': b'\x03\xeacontinuation after final message frame', '105': 1, '200': b'\x03\xeaunexpected reserved bits 0x10', '201': b'\x03\x
eaunexpected reserved bits 0x10', '202': b'\x03\xeaunexpected reserved bits 0x10', '203': b'\x03\xeaunexpected reserved bits 0x10', '204': b'\x03\xeaunexpected reserved bits
 0x10', '205': b'\x03\xeaunexpected reserved bits 0x10', '206': b'\x03\xeaunexpected reserved bits 0x10', '300': 0, '301': 0, '302': 0, '303': 1, '304': 0, '305': 0, '306':
0, '307': 0, '308': 0, '309': 0, '310': 0}
➜  fingerprint git:(main) ▌
```

STEWS Fingerprint Public Server Demo

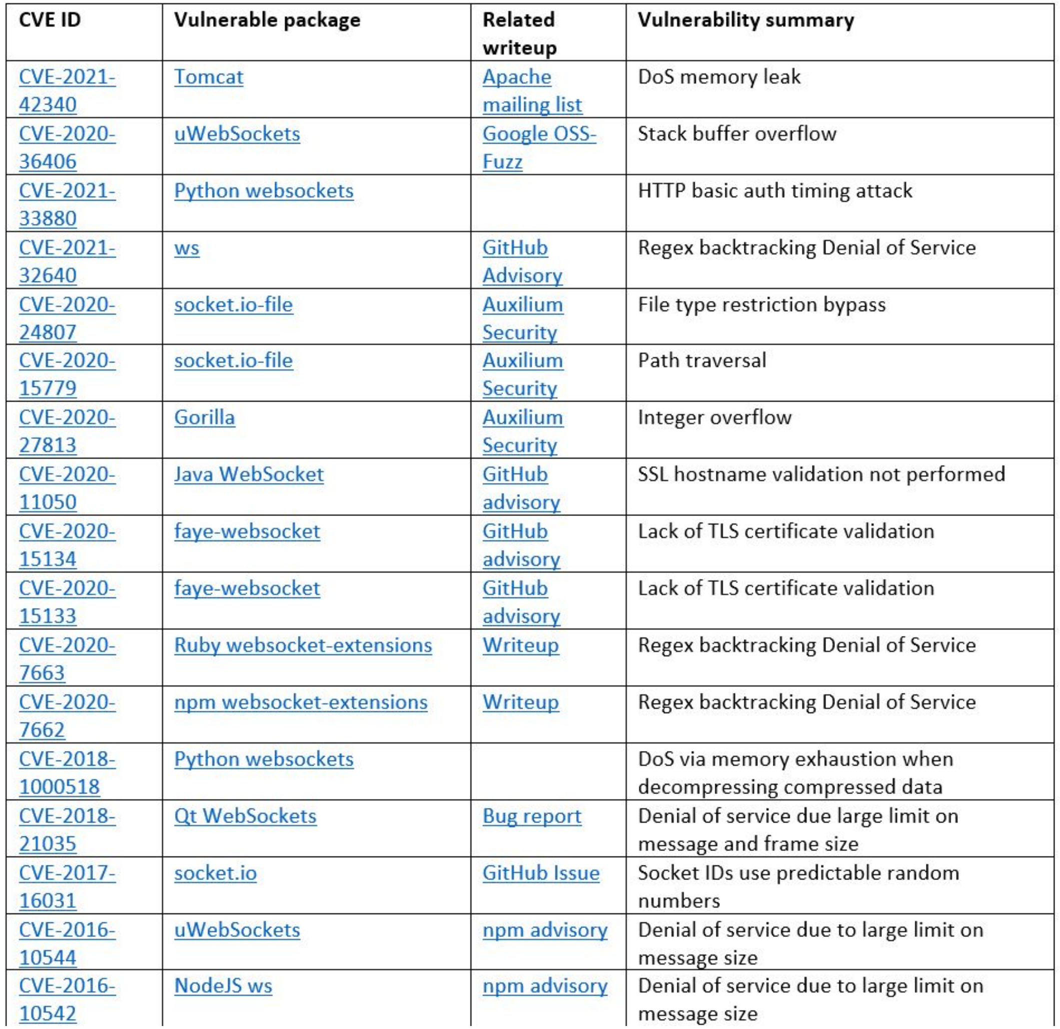


```
→ STEWS git:(main) ls
discovery  fingerprint  paper.pdf  README.md  stews-image.jpg  vuln-detect  WebSockets_AppSec_USA_2021_Presentation.pdf
→ STEWS git:(main) cd fingerprint
→ fingerprint git:(main) ls
2021-IANA-subprotocols.txt  README.md  requirements.txt  STEWS-fingerprint.py  websocket-frame.jpg
→ fingerprint git:(main) python3 STEWS-fingerprint.py -h
usage: STEWS-fingerprint.py [-h] [-v] [-d] [-u URL] [-f FILE] [-n] [-k] [-o ORIGIN] [-g] [-a] [-1] [-2] [-3] [-4] [-5] [-6] [-7]

Security Testing and Enumeration of WebSockets (STEWS) Fingerprinting Tool

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         Enable verbose tracing of communications
  -d, --debug           Print each test case to track progress while running
  -u URL, --url URL     Provide a URL to connect to
  -f FILE, --file FILE  Provide a file containing URLs to check for valid WebSocket connections
  -n, --no-encryption   Connect using ws://, not wss:// (default is wss://)
  -k, --nocert          Ignore invalid SSL cert
  -o ORIGIN, --origin ORIGIN
                        Set origin
  -g, --generate-fingerprint
                        Generate a fingerprint for a known server
  -a, --all-tests       Run all tests
  -1, --series-100      Run the 100-series (opcode) tests
  -2, --series-200      Run the 200-series (rsv bit) tests
  -3, --series-300      Run the 300-series (version) tests
  -4, --series-400      Run the 400-series (extensions) tests
  -5, --series-500      Run the 500-series (subprotocols) tests
  -6, --series-600      Run the 600-series (long payloads) tests
  -7, --series-700      Run the 700-series (hybi and similar) tests
→ fingerprint git:(main) python3 STEWS-fingerprint.py -d -1 -2 -3 -4 -u streaming.vn.teslamotors.com/streaming/
Running series 100 tests
Test 100
```

3. WebSockets Vulnerability Detection

WebSocket servers have a few CVEs…

A longer list of WebSocket server CVEs found in WebSocket Security Awesome

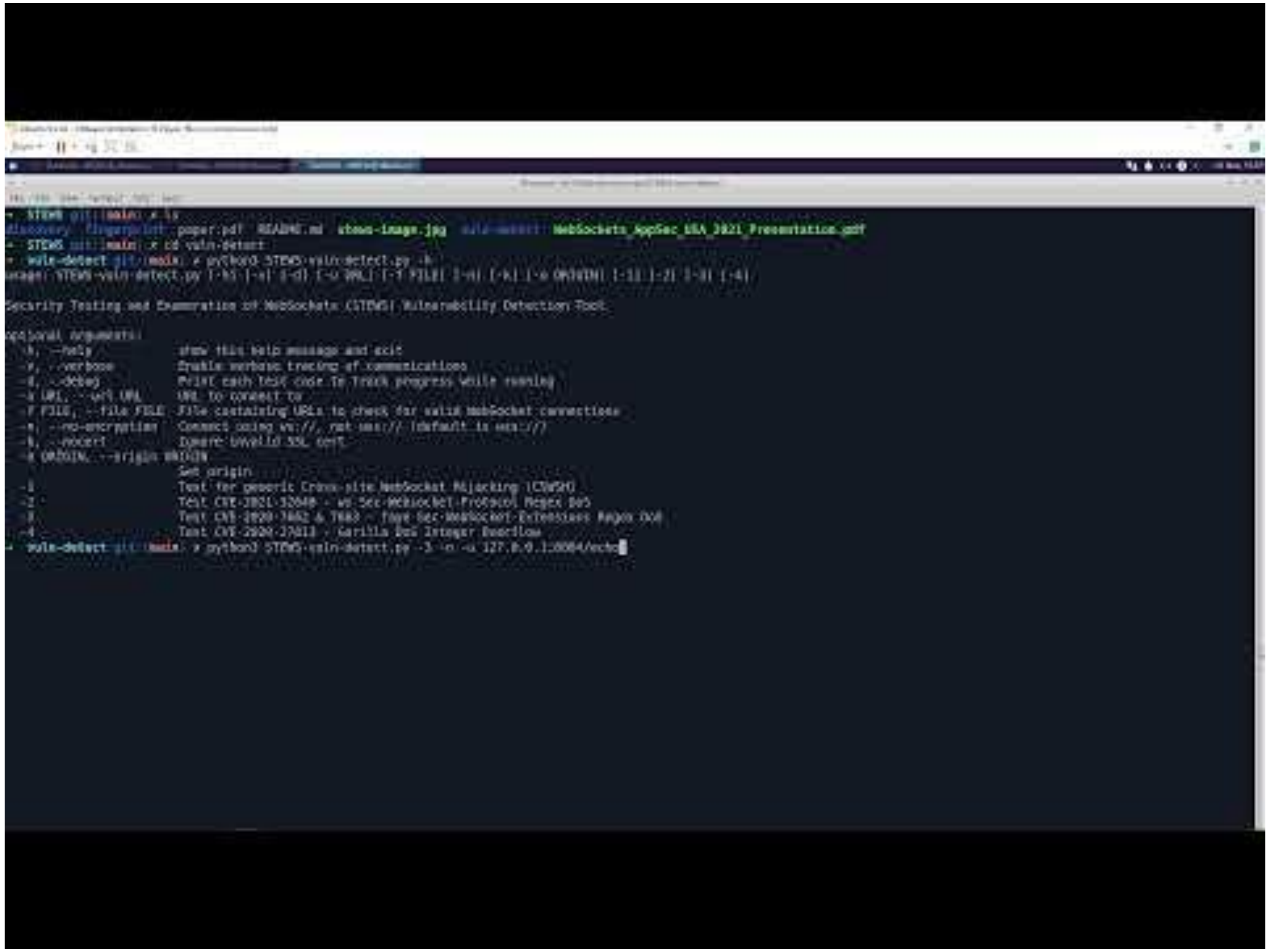


CVE ID	Vulnerable package	Related writeup	Vulnerability summary
CVE-2021-42340	Tomcat	Apache mailing list	DoS memory leak
CVE-2020-36406	uWebSockets	Google OSS-Fuzz	Stack buffer overflow
CVE-2021-33880	Python websockets		HTTP basic auth timing attack
CVE-2021-32640	ws	GitHub Advisory	Regex backtracking Denial of Service
CVE-2020-24807	socket.io-file	Auxilium Security	File type restriction bypass
CVE-2020-15779	socket.io-file	Auxilium Security	Path traversal
CVE-2020-27813	Gorilla	Auxilium Security	Integer overflow
CVE-2020-11050	Java WebSocket	GitHub advisory	SSL hostname validation not performed
CVE-2020-15134	faye-websocket	GitHub advisory	Lack of TLS certificate validation
CVE-2020-15133	faye-websocket	GitHub advisory	Lack of TLS certificate validation
CVE-2020-7663	Ruby websocket-extensions	Writeup	Regex backtracking Denial of Service
CVE-2020-7662	npm websocket-extensions	Writeup	Regex backtracking Denial of Service
CVE-2018-1000518	Python websockets		DoS via memory exhaustion when decompressing compressed data
CVE-2018-21035	Qt WebSockets	Bug report	Denial of service due large limit on message and frame size
CVE-2017-16031	socket.io	GitHub Issue	Socket IDs use predictable random numbers
CVE-2016-10544	uWebSockets	npm advisory	Denial of service due to large limit on message size
CVE-2016-10542	NodeJS ws	npm advisory	Denial of service due to large limit on message size

3. WebSockets Vulnerability Detection

- Ideally the detection process of a CVE does not involve exploiting it, but often there is no other way
- STEWS vuln-detect includes checks for a few CVEs, though more should be added in the future:
  - CVE-2020-27813 (Gorilla DoS Integer Overflow)
  - CVE-2020-7662 & CVE-2020-7663 (faye Sec-WebSocket-Extensions Regex DoS)
  - CVE-2021-32640 (ws Sec-Websocket-Protocol Regex DoS)

STEWS Vuln Detect Demo



```
➜  STEWS git:(main) ✗ ls
discovery  fingerprint  paper.pdf  README.md  stews-image.jpg  vuln-detect  WebSockets_AppSec_USA_2021_Presentation.pdf
➜  STEWS git:(main) ✗ cd vuln-detect
➜  vuln-detect git:(main) ✗ python3 STEWS-vuln-detect.py -h
usage: STEWS-vuln-detect.py [-h] [-v] [-d] [-u URL] [-f FILE] [-n] [-k] [-o ORIGIN] [-1] [-2] [-3] [-4]

Security Testing and Enumeration of WebSockets (STEWS) Vulnerability Detection Tool

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         Enable verbose tracing of communications
  -d, --debug           Print each test case to track progress while running
  -u URL, --url URL     URL to connect to
  -f FILE, --file FILE  File containing URLs to check for valid WebSocket connections
  -n, --no-encryption   Connect using ws://, not wss:// (default is wss://)
  -k, --nocert          Ignore invalid SSL cert
  -o ORIGIN, --origin ORIGIN
                        Set origin
  -1                    Test for generic Cross-site WebSocket Hijacking (CSWSH)
  -2                    Test CVE-2021-32640 - ws Sec-WebSocket-Protocol Regex DoS
  -3                    Test CVE-2020-7662 & 7663 - faye Sec-WebSocket-Extensions Regex DoS
  -4                    Test CVE-2020-27813 - Gorilla DoS Integer Overflow
➜  vuln-detect git:(main) ✗ python3 STEWS-vuln-detect.py -3 -n -u 127.0.0.1:8084/echo
```

Summary

Part 1: WebSockets work like HTTP, but less examined

Part 2: Minimal research done around WebSocket security and popular tools lack support

Part 3: STEWS toolset provides off-the-shelf tooling for discovery, fingerprinting, and vulnerability detection of WebSocket servers

Ideas for Future Research

1. Security of WebSockets subprotocols
2. Security of WebSocket Compression (RFC 7692)
3. Fast JavaScript-based spidering to discover WebSocket endpoints on single domain
4. Can other HTTP-type attacks be ported to WebSocket servers?

Over a dozen additional ideas listed in whitepaper

Recommended Additional Resources

PortSwigger WebSocket mini-CTF exercises:
https://portswigger.net/web-security/websockets

Mikhail Egorov's 2019 conference talk:
https://www.youtube.com/watch?v=gANzRo7UHt8

WebSocket Protocol RFC, RFC 6455:
https://datatracker.ietf.org/doc/html/rfc6455

WebSocket Protocol Compression RFC, RFC 7692:
https://datatracker.ietf.org/doc/html/rfc7692

Thank You!

Questions?

Email: erik.elbieh@palindrometech.com

Site: https://erikelbieh.com

OWASP 2021 >irtual APPSEC

THANK YOU!