Google

Web Application Vulnerability Trends in the Wild

aaj@google.com, W3C TPAC 2018

Background

GOOGLE VULNERABILITY REWARD PROGRAM

2017 Year in Review

1,230
INDIVIDUAL REWARDS

274
PAID RESEARCHERS

113
COUNTRIES REPRESENTED IN BUG REPORTS

60
COUNTRIES REPRESENTED IN BUG REWARDS

$112,500
BIGGEST SINGLE REWARD

$160,000+
DONATED TO CHARITY

Google Application Security

Home Learning **Reward Programs** Hall of Fame Research

Google VRP Patch Rewards AutoFuzz Patch Rewards Research Grants Chrome Rewards Android Rewards Google Play Rewards

Category	Examples	Applications that permit taking over a Google account [1]	Other highly sensitive applications [2]	Normal Google applications	Non-integrated acquisitions and other sandboxed or lower priority applications [3]
Vulnerabilities giving direct access to Google servers					
Remote code execution	*Command injection, deserialization bugs, sandbox escapes*	$31,337	$31,337	$31,337	$1,337 - $5,000
Unrestricted file system or database access	*Unsandboxed XXE, SQL injection*	$13,337	$13,337	$13,337	$1,337 - $5,000
Logic flaw bugs leaking or bypassing significant security controls	*Direct object reference, remote user impersonation*	$13,337	$7,500	$5,000	$500
Vulnerabilities giving access to client or authenticated session of the logged-in victim					
Execute code on the client	Web: *Cross-site scripting* Mobile / Hardware: *Code execution*	$7,500	$5,000	$3,133.7	$100
Other valid security vulnerabilities	Web: *CSRF, Clickjacking* Mobile / Hardware: *Information leak, privilege escalation*	$500 - $7,500	$500 - $5,000	$500 - $3,133.7	$100

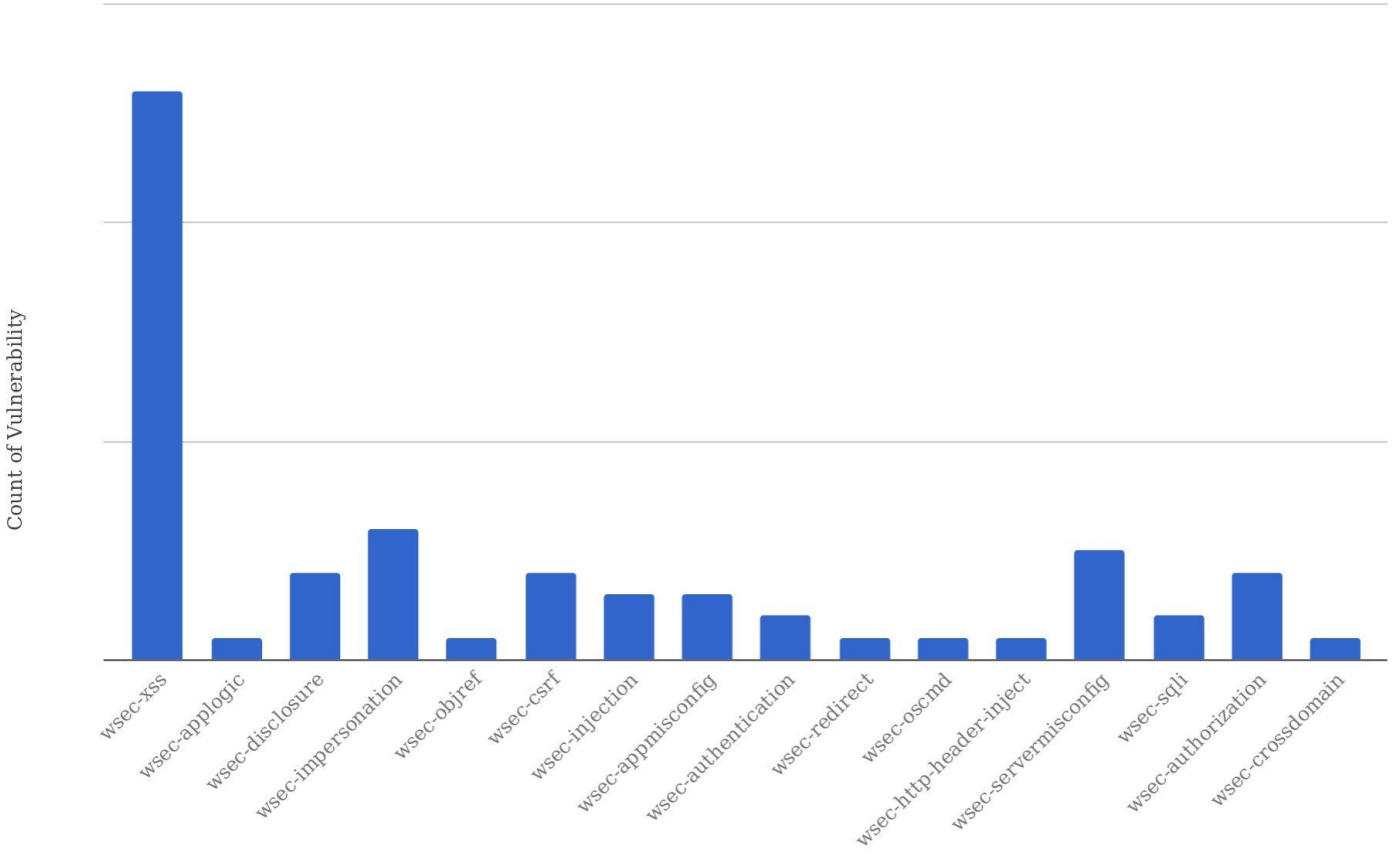
Ecosystem of web applications at Google

- Server-side code
  - 4 major languages: Java, C++, Python, Go
  - 16 HTML template system engines
  - Dozens of server-side stacks/frameworks
- Client-side code: mostly JS and TypeScript
  - A diverse set of frameworks: Angular, Polymer, GWT, Closure
- 619 distinct applications under *.google.com
  - 2 billion lines of code total
  - Large amount of third-party code, including in external repositories
- Hundreds of acquired companies, often with very different infrastructure

Traditional SDL/hardening approaches have limits => **emphasis on the platform**.

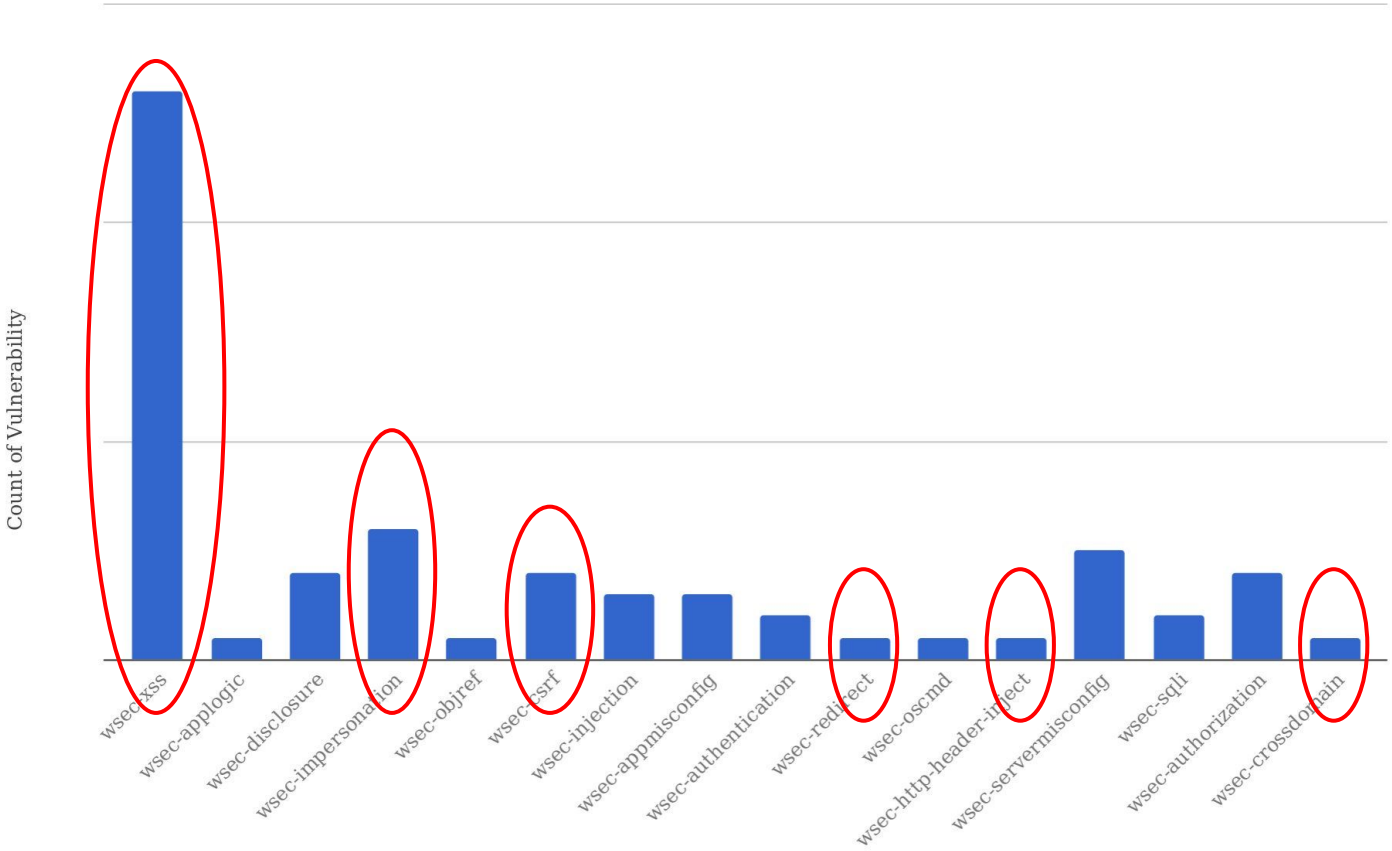
Vulnerabilities

Paid bounties by vulnerability on Mozilla websites in 2016 and 2017



Source: @jvehent, Mozilla (legend)

Paid bounties by vulnerability on Mozilla websites in 2016 and 2017



Source: @jvehent, Mozilla (legend)

VULNERABILITIES BY INDUSTRY

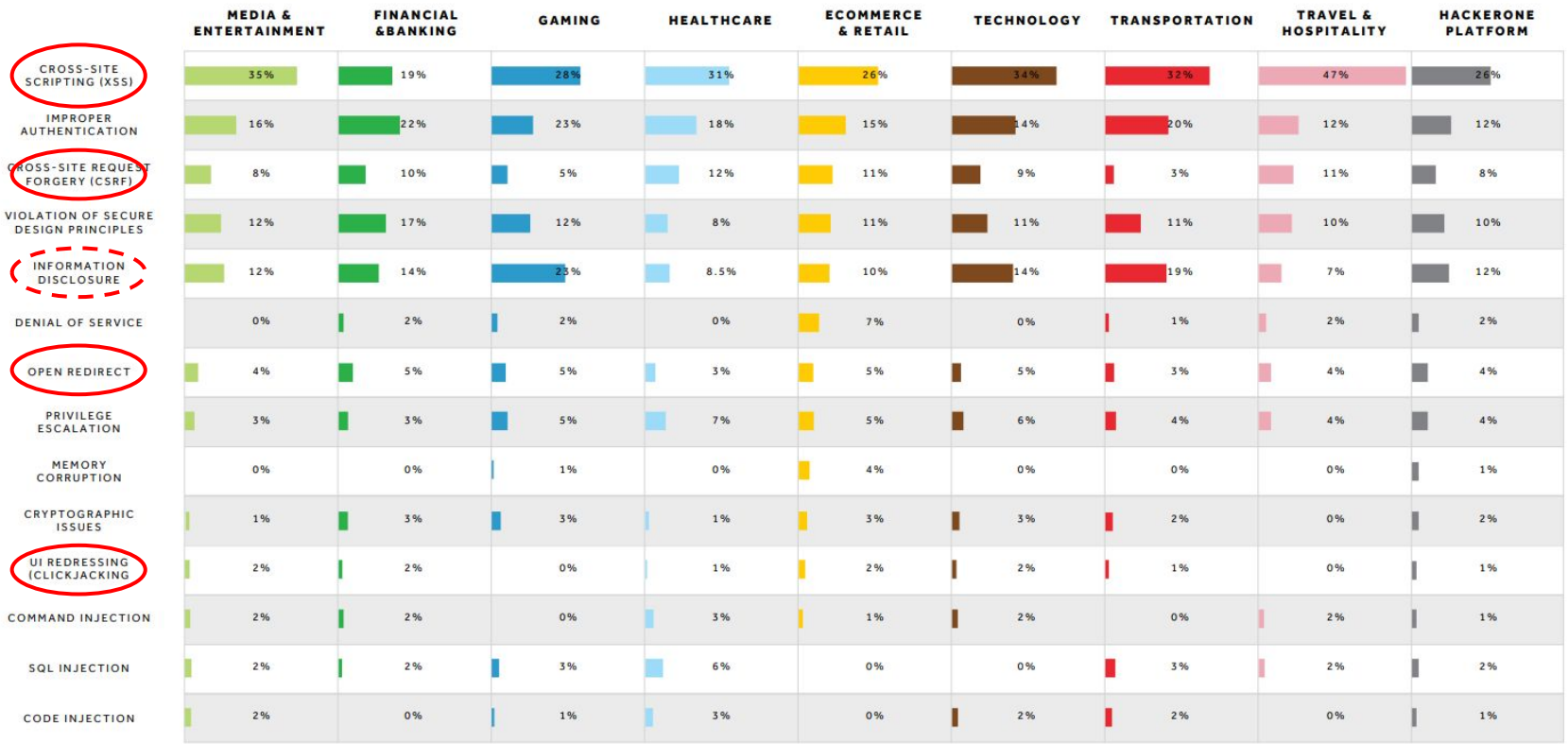
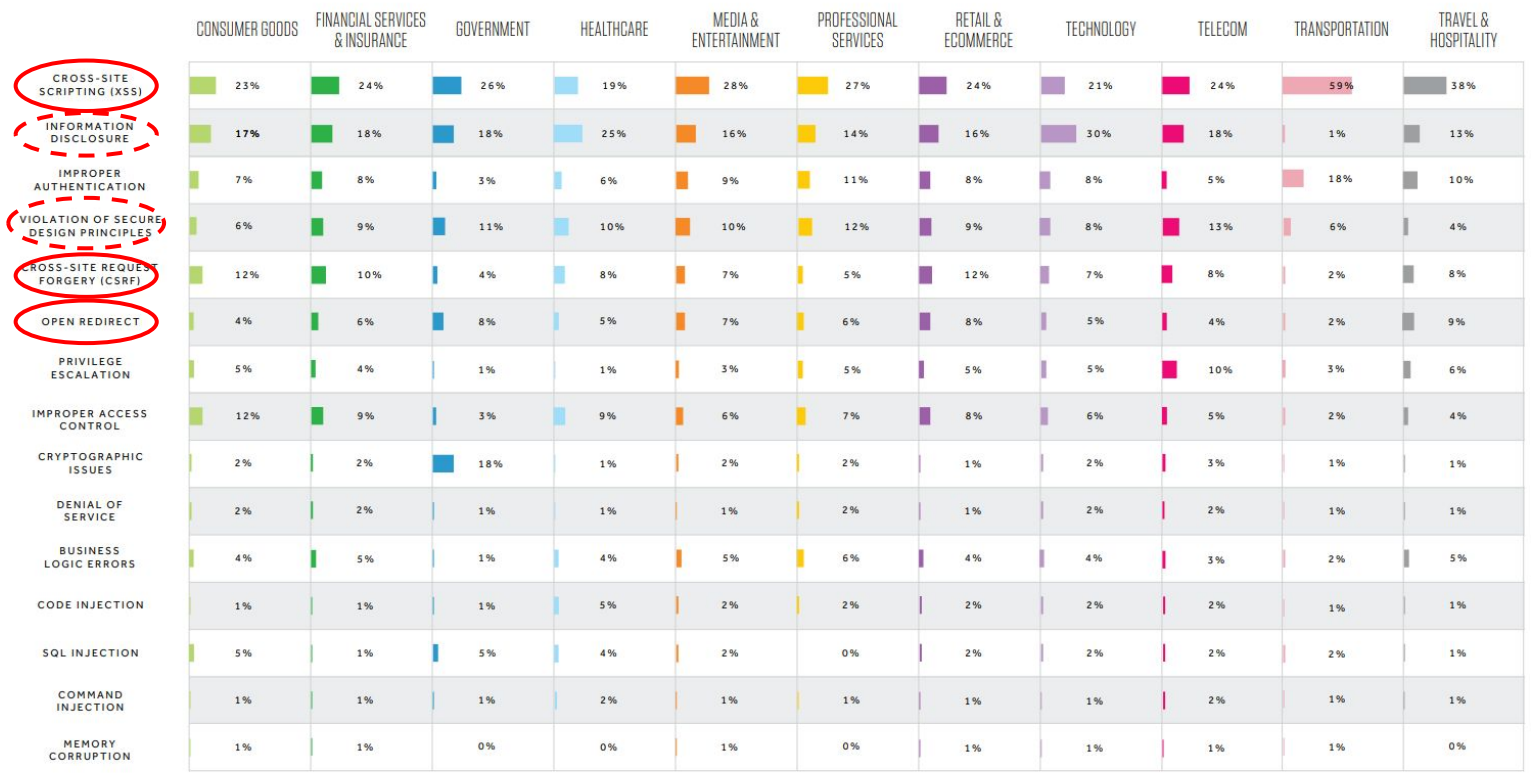


Figure 2: *Percentage of vulnerability type by industry from 2013 to May 2017.*

Source: HackerOne report, 2017

Vulnerabilities by Industry

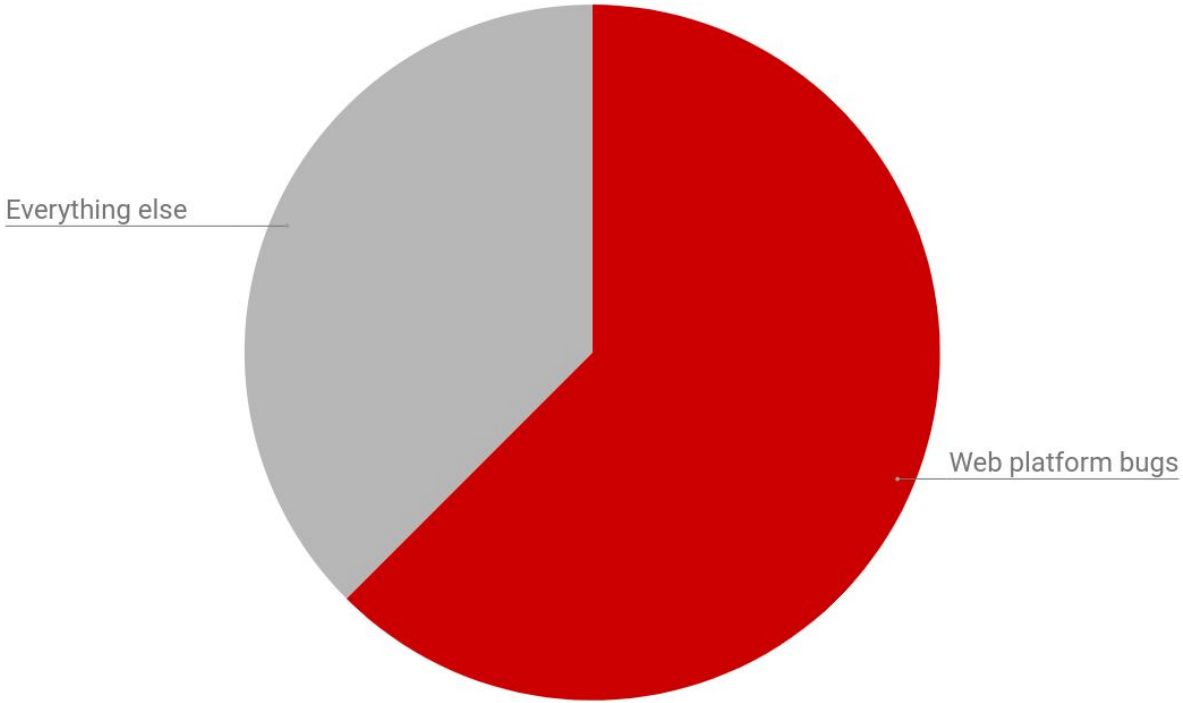


	CONSUMER GOODS	FINANCIAL SERVICES & INSURANCE	GOVERNMENT	HEALTHCARE	MEDIA & ENTERTAINMENT	PROFESSIONAL SERVICES	RETAIL & ECOMMERCE	TECHNOLOGY	TELECOM	TRANSPORTATION	TRAVEL & HOSPITALITY
CROSS-SITE SCRIPTING (XSS)	23%	24%	26%	19%	28%	27%	24%	21%	24%	59%	38%
INFORMATION DISCLOSURE	17%	18%	18%	25%	16%	14%	16%	30%	18%	1%	13%
IMPROPER AUTHENTICATION	7%	8%	3%	6%	9%	11%	8%	8%	5%	18%	10%
VIOLATION OF SECURE DESIGN PRINCIPLES	6%	9%	11%	10%	10%	12%	9%	8%	13%	6%	4%
CROSS-SITE REQUEST FORGERY (CSRF)	12%	10%	4%	8%	7%	5%	12%	7%	8%	2%	8%
OPEN REDIRECT	4%	6%	8%	5%	7%	6%	8%	5%	4%	2%	9%
PRIVILEGE ESCALATION	5%	4%	1%	1%	3%	5%	5%	5%	10%	3%	6%
IMPROPER ACCESS CONTROL	12%	9%	3%	9%	6%	7%	8%	6%	5%	2%	4%
CRYPTOGRAPHIC ISSUES	2%	2%	18%	1%	2%	2%	1%	2%	3%	1%	1%
DENIAL OF SERVICE	2%	2%	1%	1%	1%	2%	1%	2%	2%	1%	1%
BUSINESS LOGIC ERRORS	4%	5%	1%	4%	5%	6%	4%	4%	3%	2%	5%
CODE INJECTION	1%	1%	1%	5%	2%	2%	2%	2%	2%	1%	1%
SQL INJECTION	5%	1%	5%	4%	2%	0%	2%	2%	2%	2%	1%
COMMAND INJECTION	1%	1%	1%	2%	1%	1%	1%	1%	2%	1%	1%
MEMORY CORRUPTION	1%	1%	0%	0%	1%	0%	1%	1%	1%	1%	0%

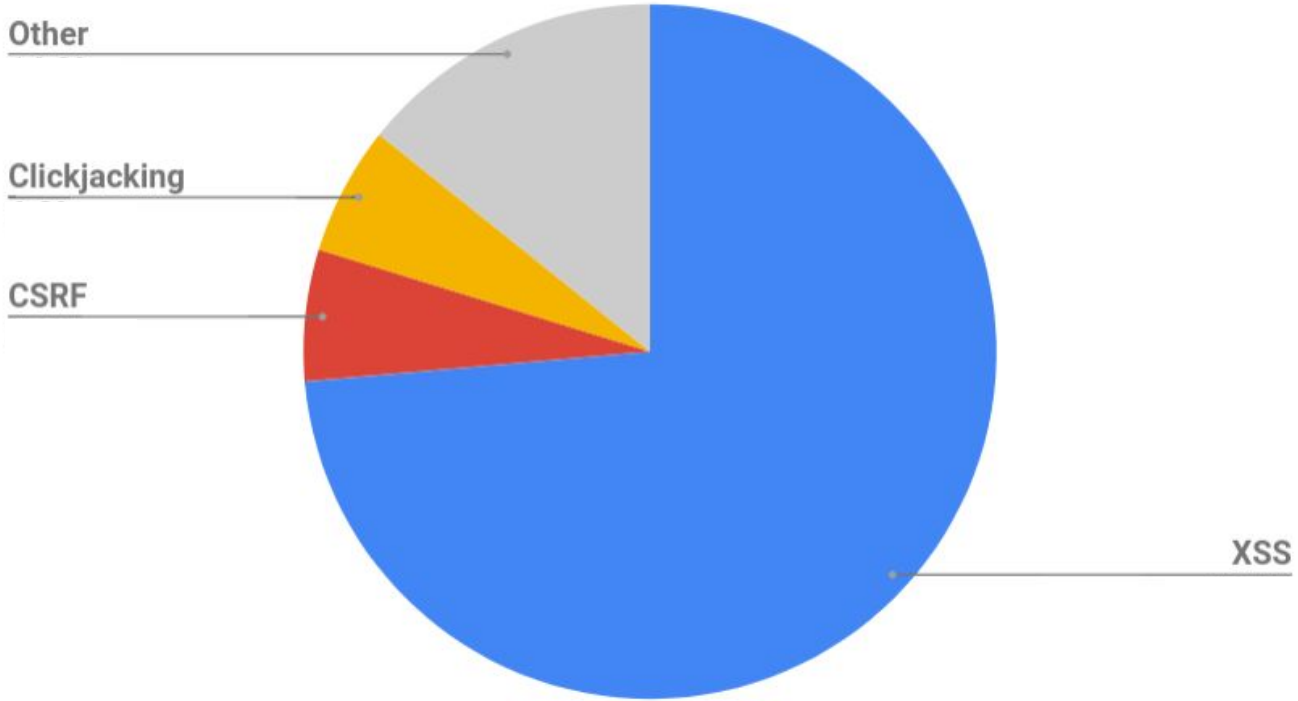
Figure 5: *Listed are the top 15 vulnerability types platform wide, and the percentage of vulnerabilities received per industry.*

Source: HackerOne report, 2018

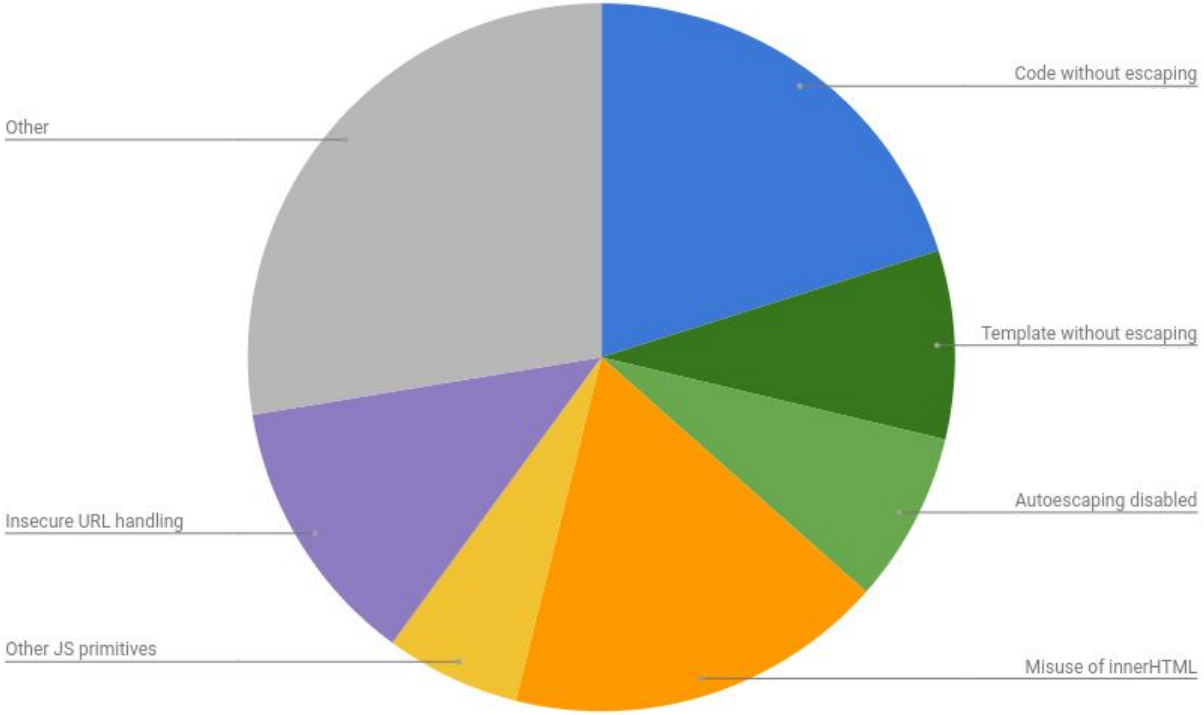
Total Google VRP Rewards (since 2014)



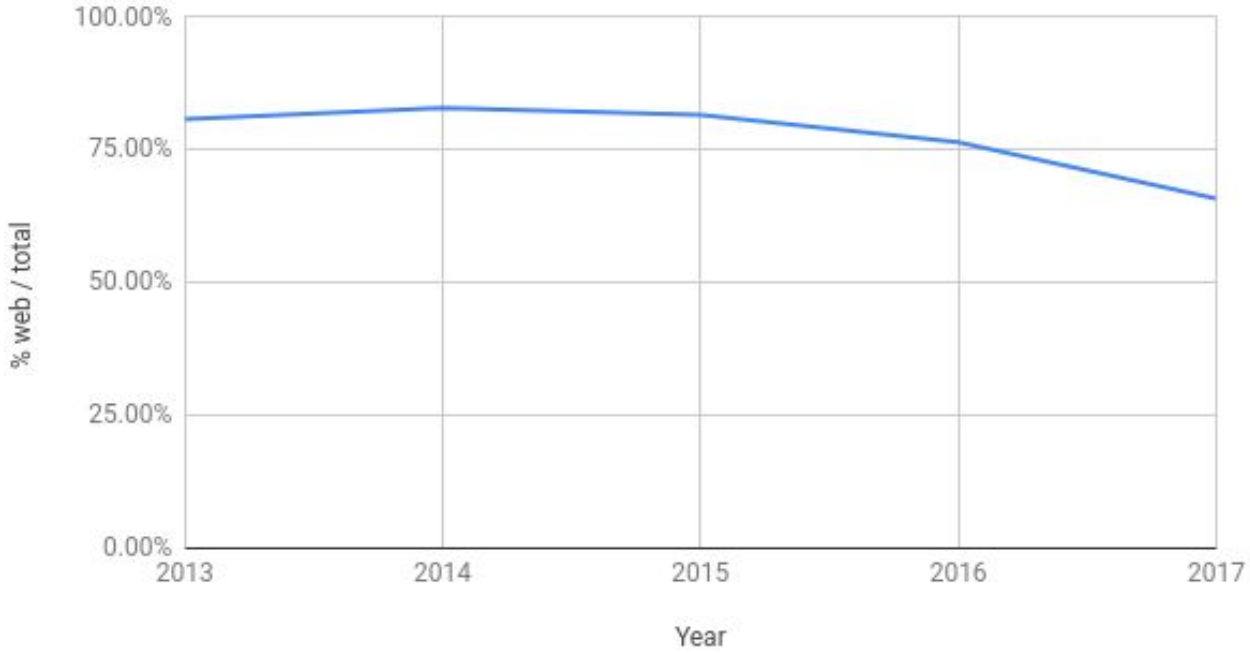
Google VRP Rewards for Web Platform Bugs



Main Causes of XSS Vulnerabilities



Web Platform Vulnerabilities as % of Total



Summary of Vulnerability Trends

- The majority of application vulnerabilities are *"web platform"* issues exploitable against logged-in application users.
- Main vulnerability classes:
  - **XSS** in its various forms
  - CSRF, XSSI / information disclosure, clickjacking / UI redress.
- Long tail of issues caused by cross-origin leakiness of the platform:
  - XS-Search, size leaks, pixel-perfect leaks, window.frame counting

A high-level view of web security

Three major classes of problems:

#1. Lack of transport safety

No confidentiality / integrity of traffic => all bets are off.

Vulnerabilities:

- The use of HTTP, use of non-Secure cookies, mixed scripting/content.

Specs:

- HSTS, Mixed Content, UIR, Secure Contexts, ...

#2. Injections

Attacker's scripts running in a vulnerable origin => all bets are off.

Vulnerabilities:

- XSS

Specs:

- CSP3, Trusted Types, [Suborigins], Sanitization

#3. Forced loading of endpoints from victim's origin

Broad class of purpose-specific attacks that violate integrity or confidentiality.

- Violating integrity by forcing the inclusion of a resource:
  - CSRF, clickjacking
- Violating confidentiality by forcing the inclusion of a resource:
  - XSSI, XS-Search & timing attacks, pixel-perfect attacks, …

Note: This is getting worse as new APIs are added to the web platform.

Specs:

- SameSite cookies, CORB/CORP, Sec-Metadata, COWP, [Isolate-Me]

Analysis

The (transport, injections, cross-origin leaks) model covers a large majority of the web platform bugs security engineers see in modern applications.

There are several areas of web platform security that it doesn't cover:

- **Containment**: HTML sandbox, COWL, script capability restrictions
- **Attacks by trusted resources**: SRI, Referrer Policy
- **Direct attacks on the browser** (e.g. history/cache sniffing) or **on the user**

These classes of issues are still worth spending time on.

Final words

To build security into the web platform we need to give developers mechanisms to solving the three big problems in their applications:

- Secure transport
- Injections
- Cross-origin leaks

Failing to address these problems will have a large cost for the platform: developers will either spend a lot of resources on compensating for the deficiencies of the platform or they'll be forced into a constant state of insecurity.

If these mechanisms work as opt-in, we might be able to turn them on by default.

[end]

Bonus: Isolation features in response to 👻Spectre👻

Three major areas of work to protect against speculative execution attacks:

- How do I **limit access to my resources?** [summary]
  - Any response loaded in `no-cors` mode can be exfiltrated by evil.com
  - Specs: CORB, CORP, Sec-Metadata, SameSite cookies,
- How do I **make sure my documents live in their own process?** [summary]
  - Two sets of converging goals: **browsers** want to allow process-based isolation; **authors** want severing of window references
  - Specs: COWP ("level 1") / the old CSP3 `disown-opener' keyword
- How do we **restrict the capabilities of documents with dangerous features?**
  - Ensure that documents with fine-grained timers can't bypass the SOP
  - Specs: COWP ("level 2"), X-Bikeshed-Force-Isolate