Web security

EECE 571B “Computer Security”

Konstantin Beznosov

cross-origin CSS attacks and countermeasures

same-origin policy

- HTML document can include images, scripts, videos, and other documents, etc., from any site.
- the document's scripts may not directly examine content loaded from other sites.
  - a script can only inspect the content of a nested document if it came from the same origin as the script itself.

adversary model

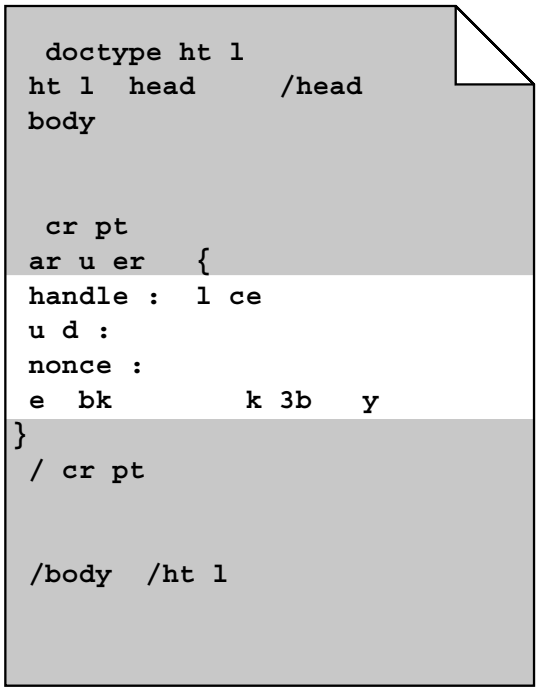
- **web attacker** -- a malicious principal who owns a domain name and operates a web server
- objectives: steal data from another web site (**target**) that should only be revealed to a particular user (**victim**)
- capabilities
  - can send and receive arbitrary network traffic, but only from its own servers
  - can inject strings into the target site, even into pages that it cannot retrieve
    - its injections must pass server-side cross-site scripting (XSS) filters
  - can entice the victim into visiting attacker's site
- excluded capabilities
  - cannot modify or eavesdrop on the victim's network traffic to other sites,
  - cannot generate "spoofed" packets that purport to be from some other site.
  - cannot install malicious software on the victim's computer
  - the victim does not disclose any sensitive information while on the attacker's site

unexpected interactions make the attack possible

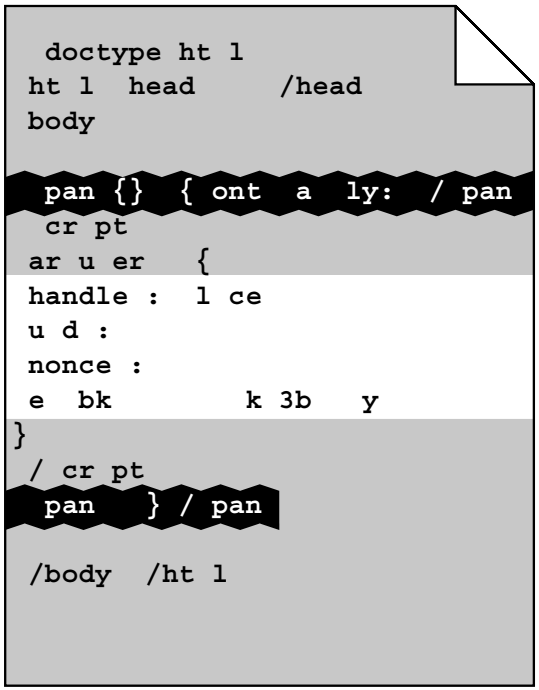
- session authentication
  - client-side state (e.g., HTTP cookies) to manage a distinct “session” for each visitor
  - sent on each request
- cross-origin content inclusion
  - browsers permit web pages to include resources (images, scripts, style sheets, etc.) from any origin
  - requests for cross-origin resources transmit any credentials (cookies or HTTP authentication tokens) associated with the site that hosts the resource, not credentials associated with the site whose page made the reference
  - a confidential resource from one site can be included into a page that could not read it directly; it will be visible to the user, but not to scripts running in the page
- error-tolerant style sheet parsing
  1. discard the current syntactic construct
  2. skip ahead until what appears to be the beginning of the next one
  3. start parsing again.

attack steps

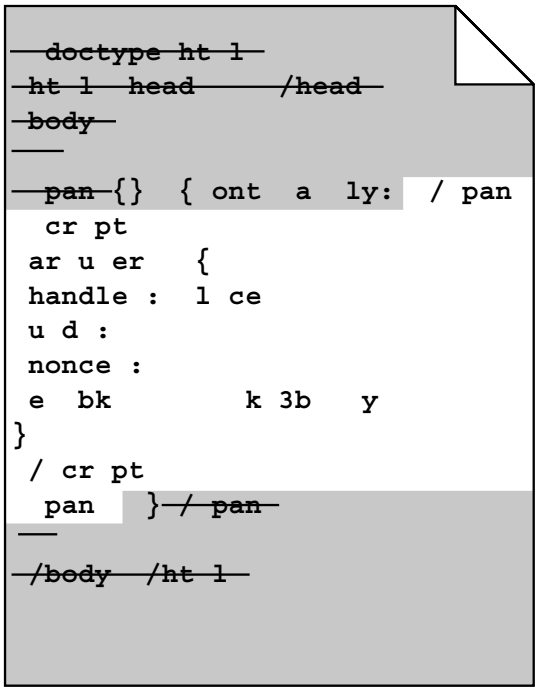
1. injects strings into the target document that bracket the data to be stolen
2. entices the victim into visiting a malicious page under attacker's control
3. the malicious page imports the target document as if it were a style sheet
   - can extract confidential information from the parsed style rules



HTML document; secret data is highlighted.



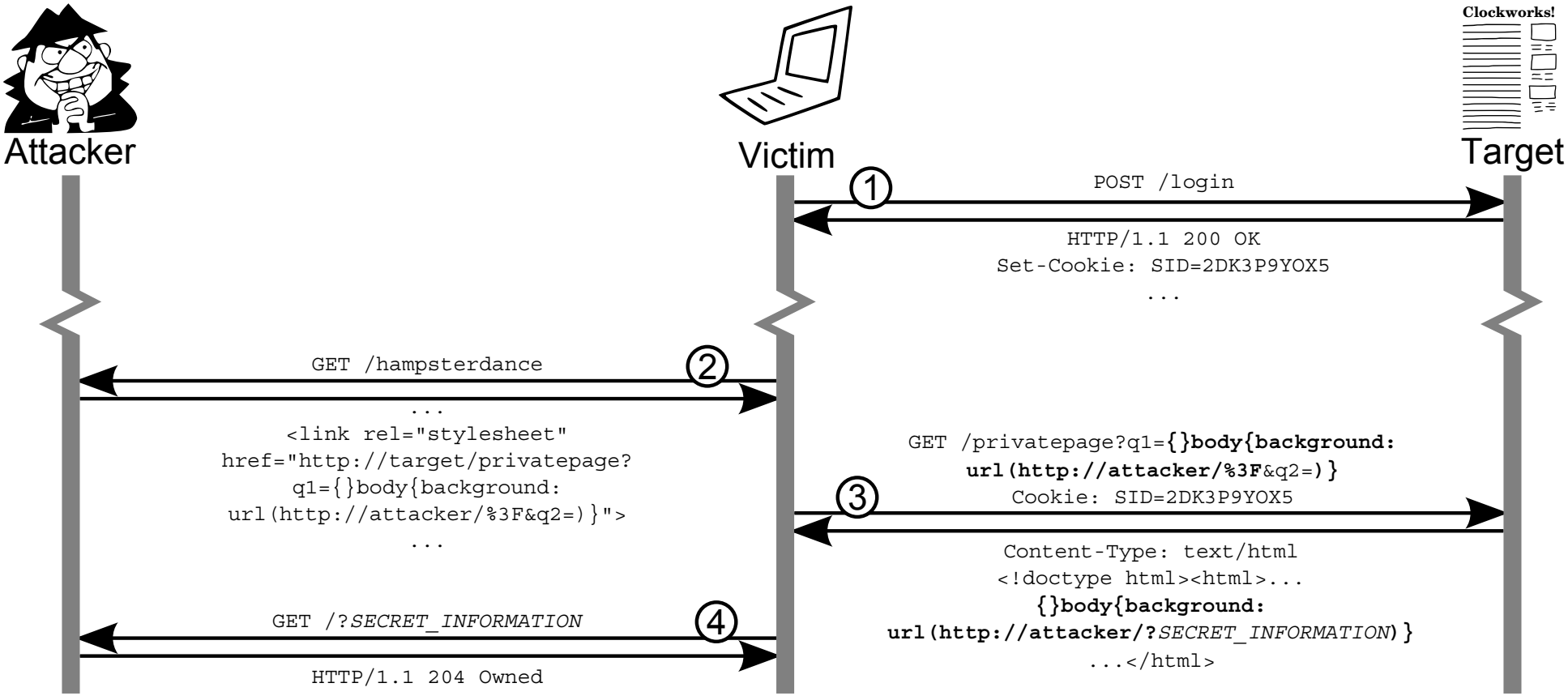
Attacker injects CSS leader and trailer around secret.



CSS parser skips most of the document, loads secret as a valid style rule.

adopted from [1]

attack example



adopted from [1]

countermeasures

- **strict enforcement**: enforce content type checking for style sheets loaded cross-origin
  - style sheets only be processed if they are labeled with the HTTP header Content-Type: text/css
  - may cause legitimate requests for cross-origin style sheets to fail, if the server providing the style sheet is misconfigured

- **minimal enforcement:** block a CSS resource iff it is
  1. loaded cross-origin
  2. has an invalid content type
  3. syntactically malformed

Alexa's top 100,000 sites

Requesting server	Rendering mode	Total	HTTP error	Correct type		Incorrect type	
				Well-formed	Malformed	Well-formed	Malformed
Same-origin	Standards	180,445	1,497	178,017	506	424	1
	Quirks	25,606	466	24,445	332	304	59
Cross-origin	Standards	47,943	347	47,345	104	147	**0**
	Quirks	6,075	53	5,891	57	74	**0**
	Total	260,069	2,363	255,698	999	949	60

adopted from [1]

- strict enforcement would break 62 (74 sheets) Alexa's sites.
- 60 style sheets were both malformed and labeled with an incorrect content type
  - none of these were served cross-origin
  - minimal enforcement policy would not break any site
- unauthenticated access only

detecting web spam with Monarch

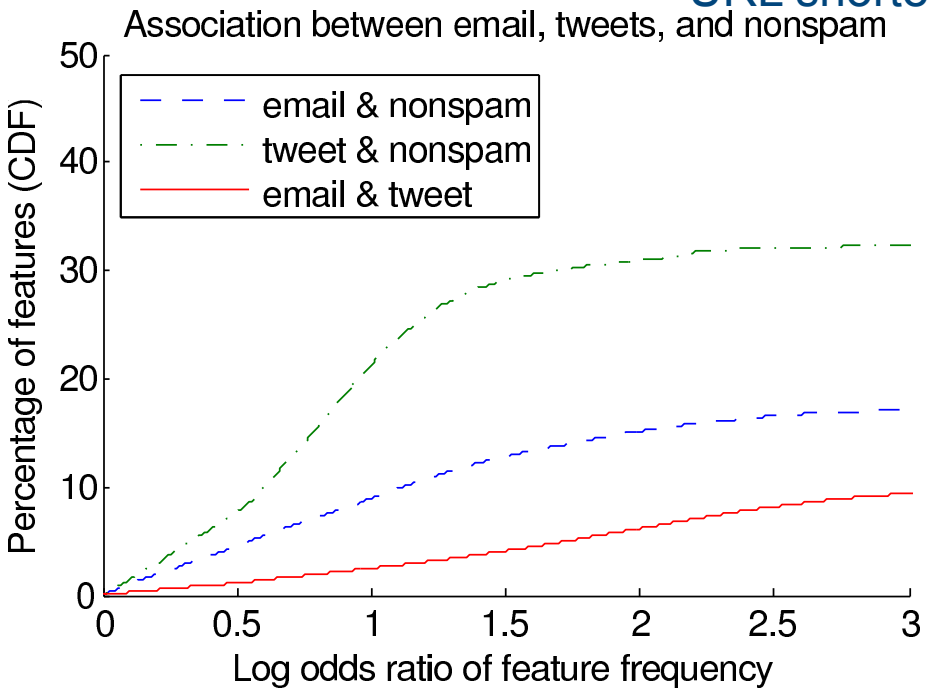
e-mail spam vs. web spam

e-mail spam

- short-lived campaigns
- quickly churn through spam domains

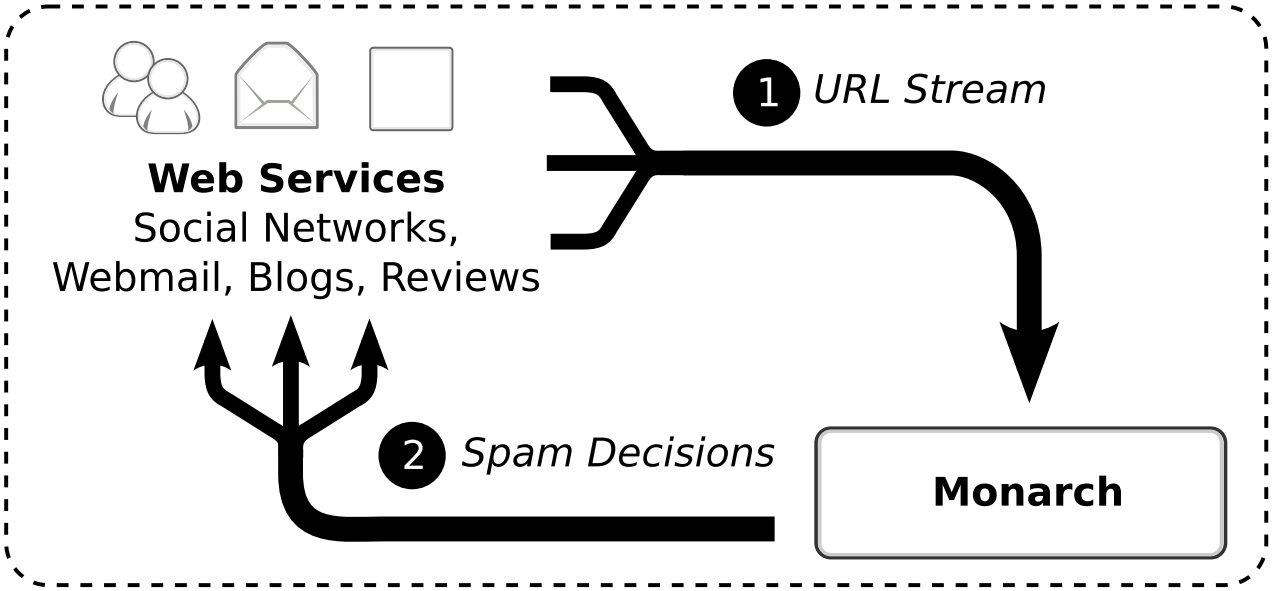
web (Twitter) spam

- long lasting campaigns
- often abuse
  - public web hosting
  - generic redirectors
  - URL shortening services



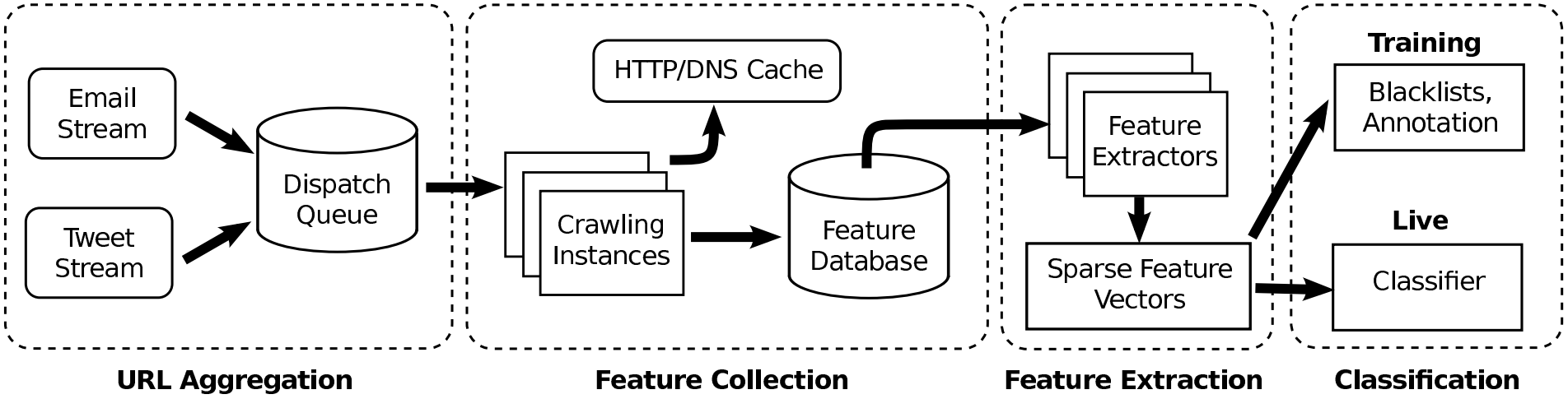
adopted from [2]

use of Monarch



adopted from [2]

data flow



adopted from [2]

collected features

Source	Features	Collected By
Initial URL, Final URL	Domain tokens, path tokens, query parameters, is obfuscated?, number of subdomains, length of domain, length of path, length of URL (From here on out, we denote this list as URL features)	Web browser
Redirects	URL features for each redirect, number of redirects, type of redirect	Web browser
Frame URLs	URL features for each embedded IFrame	Web browser
Source URLs	URL features for every outgoing network request; includes scripts, redirects, and embedded content	Web browser
HTML Content	Tokens of main HTML, frame HTML, and script content	Web browser
Page Links	URL features for each link, number of links, ratio of internal domains to external domains	Web browser
JavaScript Events	Number of user prompts, tokens of prompts, onbeforeunload event present?	Web browser
Pop-up Windows	URL features for each window URL, number of windows, behavior that caused new window	Web browser
Plugins	URL features for each plugin URL, number of plugins, application type of plugin	Web browser
HTTP Headers	Tokens of all field names and values; time-based fields are ignored	Web browser
DNS	IP of each host, mailserver domains and IPs, nameserver domains and IPs, reverse IP to host match?	DNS resolver
Geolocation	Country code, city code (if available) for each IP encountered	IP analysis
Routing Data	ASN/BGP prefix for each IP encountered	IP analysis

adopted from [2]

classification

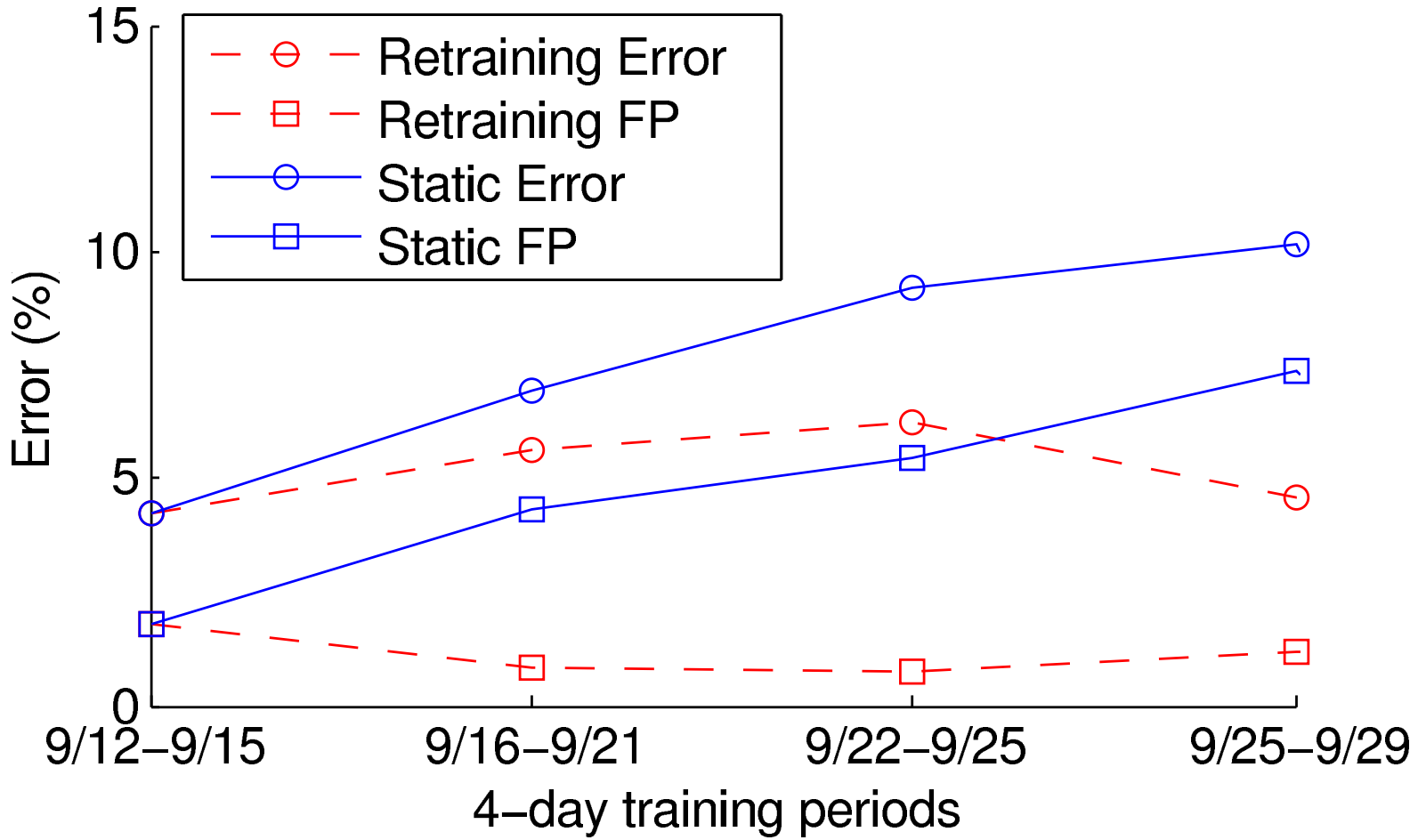
- high-dimensional (more than 10^7) space of features
- each URL is a data point representing a sparse (1,000-1,500) feature vector
- linear classifier
- data sets
  - URLs captured by spam traps operated by major email providers (1.2M)
  - blacklisted URLs appearing on Twitter (0.5M)
  - non-spam URLs appearing on Twitter (represent a non-spam data sample) (9M)

evaluation

Feature Type	Accuracy	FP	FN
Source URLs	89.74%	1.17%	19.38%
HTTP Headers	85.37%	1.23%	28.07%
HTML Content	85.32%	1.36%	28.04%
Initial URL	84.01%	1.14%	30.88%
Final URL	83.59%	2.34%	30.53%
IP (Geo/ASN)	81.52%	2.33%	34.66%
Page Links	75.72%	15.46%	37.68%
Redirects	71.93%	0.85%	55.37%
DNS	72.40%	25.77%	29.44%
Frame URLs	60.17%	0.33%	79.45%

adopted from [2]

accuracy over time



time

Component	Median Run Time (seconds)
URL aggregation	0.005
Feature collection	5.46
Feature extraction	0.074
Classification	0.002
Total	**5.54**

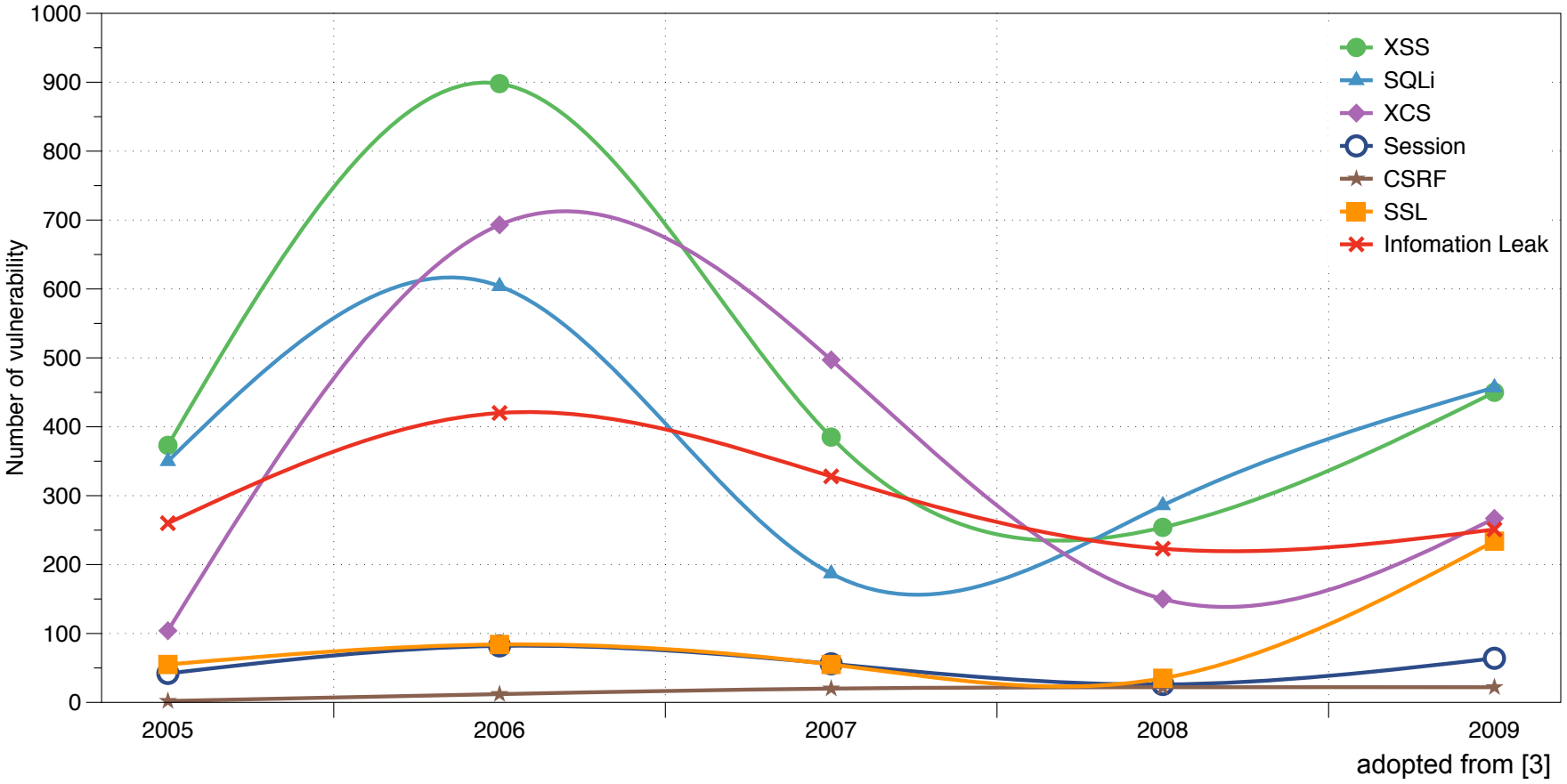
cost

Component	AWS Infrastructure	Monthly Cost
URL aggregation	1 Extra Large	$178
Feature collection	20 High-CPU Medium	$882
Feature extraction	—	$0
Classification	50 Double Extra Large	$527
Storage	700GB on EBS	$70
Total		**$1,587**

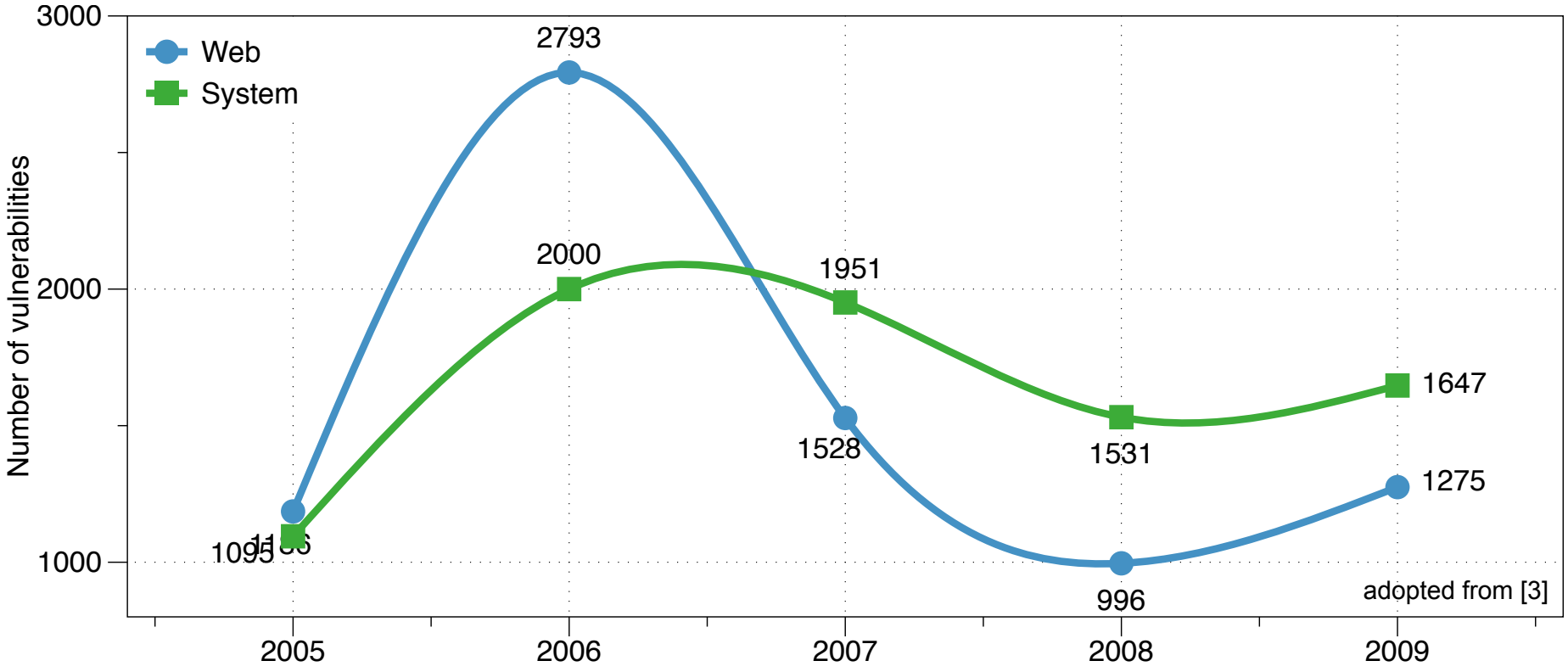
Amazon Web Services

stats on web application vulnerabilities

vulnerabilities in web applications



web application vs. system vulnerabilities

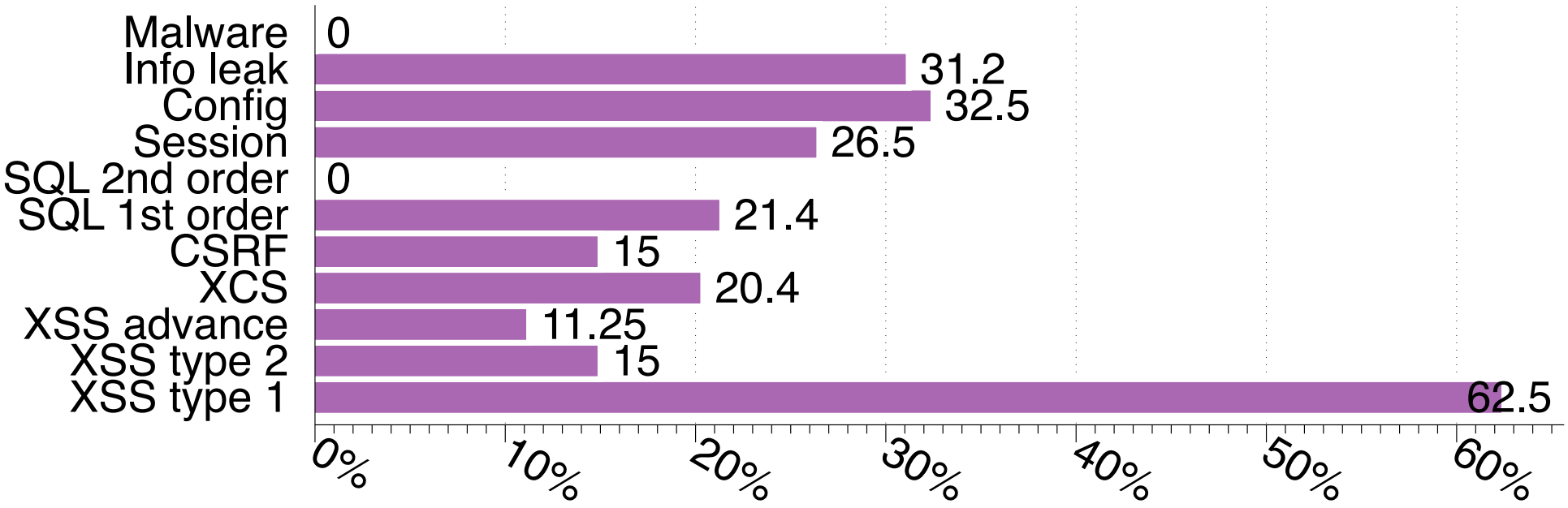


security black-box testing of web applications

categories of vulnerabilities

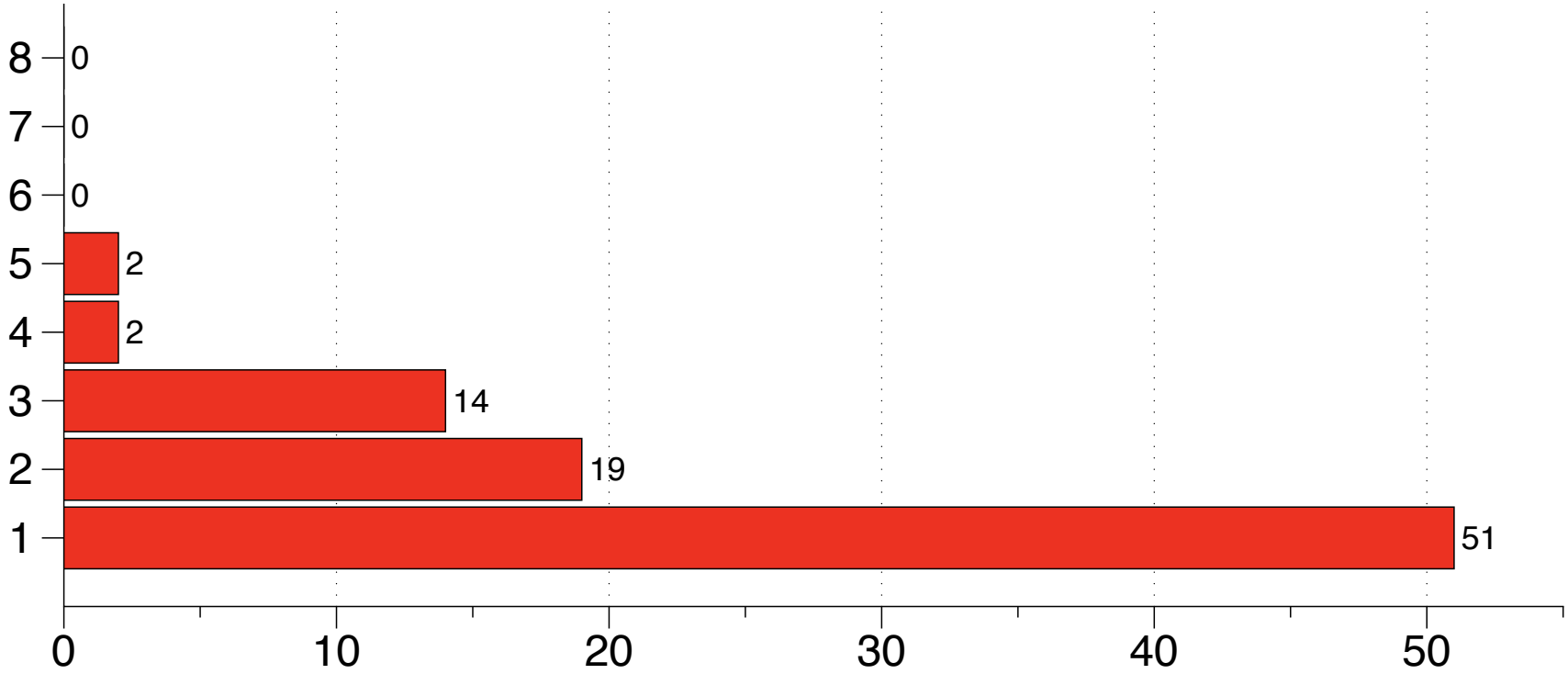
- **cross-channel scripting (XCS)**
  - all vulnerabilities allowing the attacker to inject code in the web server that manipulates the server or client browser
    - XPath injection, Malicious File Upload, Open Redirects, Cross-Frame Scripting, Server Side Includes, Path Traversal, Header Injection (HTTP Response Splitting), Flash Parameter Injection, and SMTP Injection.
  - **cross-site scripting (XSS)**
    - **XSS type 1** -- reflected XSS via <script> HTML tag
    - **XSS type 2** -- stored XSS vulnerabilities where un-sanitized user input is written to the database and later performs scripting when read from the database
    - **XSS advanced** -- novel forms of reflected and stored XSS, using non-standard tags and keywords, or using Flash and similar technologies
  - **SQL Injection (SQLI)**
    - **SQLI 1st order** -- immediate command execution upon user input submission
    - **SQLI 2nd order** -- input is loaded from the database
- **session management -- session management flaws as well as authentication and cookie flaws**
  - credentials sent over unencrypted HTTP, auto-complete enabled in the password field, submitting sensitive information over GET requests, weak password and password recovery questions, and weak registration CAPTCHAs
  - insecure session cookies, non-HttpOnly cookies, too broad cookie path restrictions, predictable session and authentication id values, session fixation, ineffective logout, mixed content pages, and caching of sensitive content.
- **cross-site request forgery**
  - forms without any authorization token and also forms which utilize tokens with very few bits of entropy, session tokens that do not reset after form submission, GET- method forms vulnerable to CSRF, and CSRF-like JSON hijacking vulnerabilities.
- **information disclosure**
  - leaking of sensitive information regarding SQL database names via the die() function and existent user names via AJAX requests. Backup source code files left accessible, and path disclosure vulnerabilities present.
- **server and cryptographic configuration**

average scanner rate of detecting vulnerabilities



adopted from [3]

false positives



adopted from [3]

credits

These slides incorporate parts of the following:

1. Lin-Shung Huang, Zack Weinberg, Chris Evans, and Collin Jackson, "**Protecting browsers from cross-origin CSS attacks**," In Proceedings of the 17th ACM conference on Computer and communications security (CCS '10), pp. 619-629.
2. Thomas, K.; Grier, C.; Ma, J.; Paxson, V.; Song, D.; , "**Design and Evaluation of a Real-Time URL Spam Filtering Service**," In Proceedings of IEEE Symposium on Security and Privacy, pp.447-462, 22-25 May 2011.
3. Bau, Jason; Bursztein, Elie; Gupta, Divij; Mitchell, John, "**State of the Art: Automated Black-Box Web Application Vulnerability Testing**," In Proceedings of IEEE Symposium on Security and Privacy, pp.332-345, 16-19 May 2010.