IERG4210 Web Programming and Security

Course Website: https://course.ie.cuhk.edu.hk/~ierg4210/
Live FB Feedback Group: https://fb.com/groups/ierg4210.2015spring/

Vulnerability Scanning of WebApps & Course Reviews

Lecture 12

Dr. Adonis Fung
phfung@ie.cuhk.edu.hk

Information Engineering, CUHK
Product Security Engineering, Yahoo!

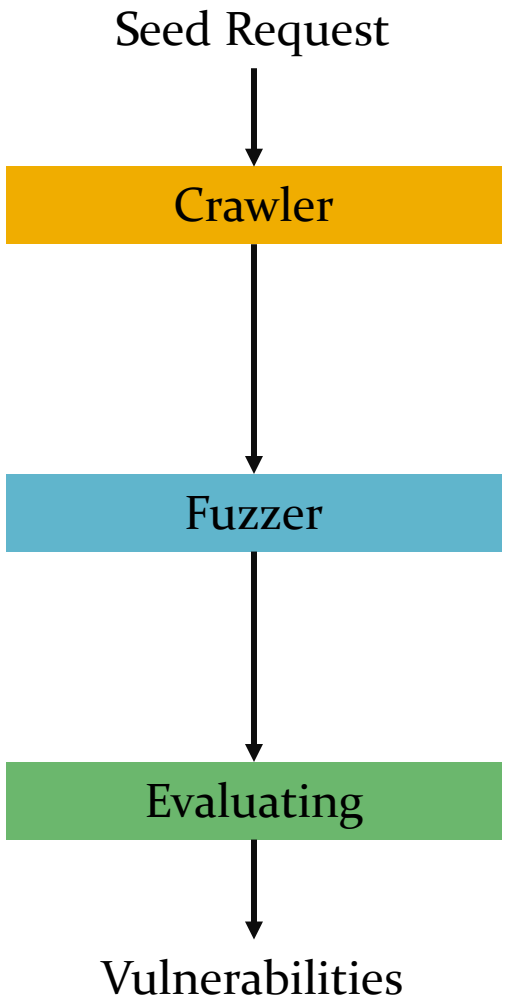
Agenda

- Automated WebApp Vulnerability Scanning
  - On the high-level architecture/concept
    - Blackbox: Crawling, Fuzzing, Output Evaluating
    - Whitebox: Source Code available for test/analysis
    - Blackbox v.s. Whitebox + Greybox
  - On actual usage
    - How to operate an automated scanner
    - How to combine it with a human knowledge
    - To be covered by our invited speaker, Scottie
      - a practitioner in penetration testing from NTT

Blackbox Vulnerability Scanning

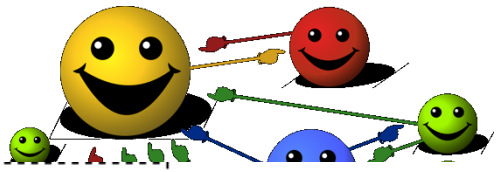
- Blackbox Scanning
  - NIST Definition: Explores a web application by crawling through its web pages and examines it for security vulnerabilities, which involves generation of malicious inputs and evaluation of application's responses.
  - Assumption: No knowledge on internal (i.e., server-side) logics
- Core Components
  - *Crawling*: Discover, record, and follow new links/requests
  - *Fuzzing*: Repeat recorded requests with mutated request parameters
  - *Evaluating*: Mostly pattern matching, vulnerability dependent
- Web Vulnerabilities covered
  - Reflected XSS, stored XSS, DOM XSS, SQL injection, Path disclosure, Server Profiling, SSL/TLS settings, LFI/RFI, etc...
  - Others: Missing Authentication Check, CSRF, Parameter Tampering

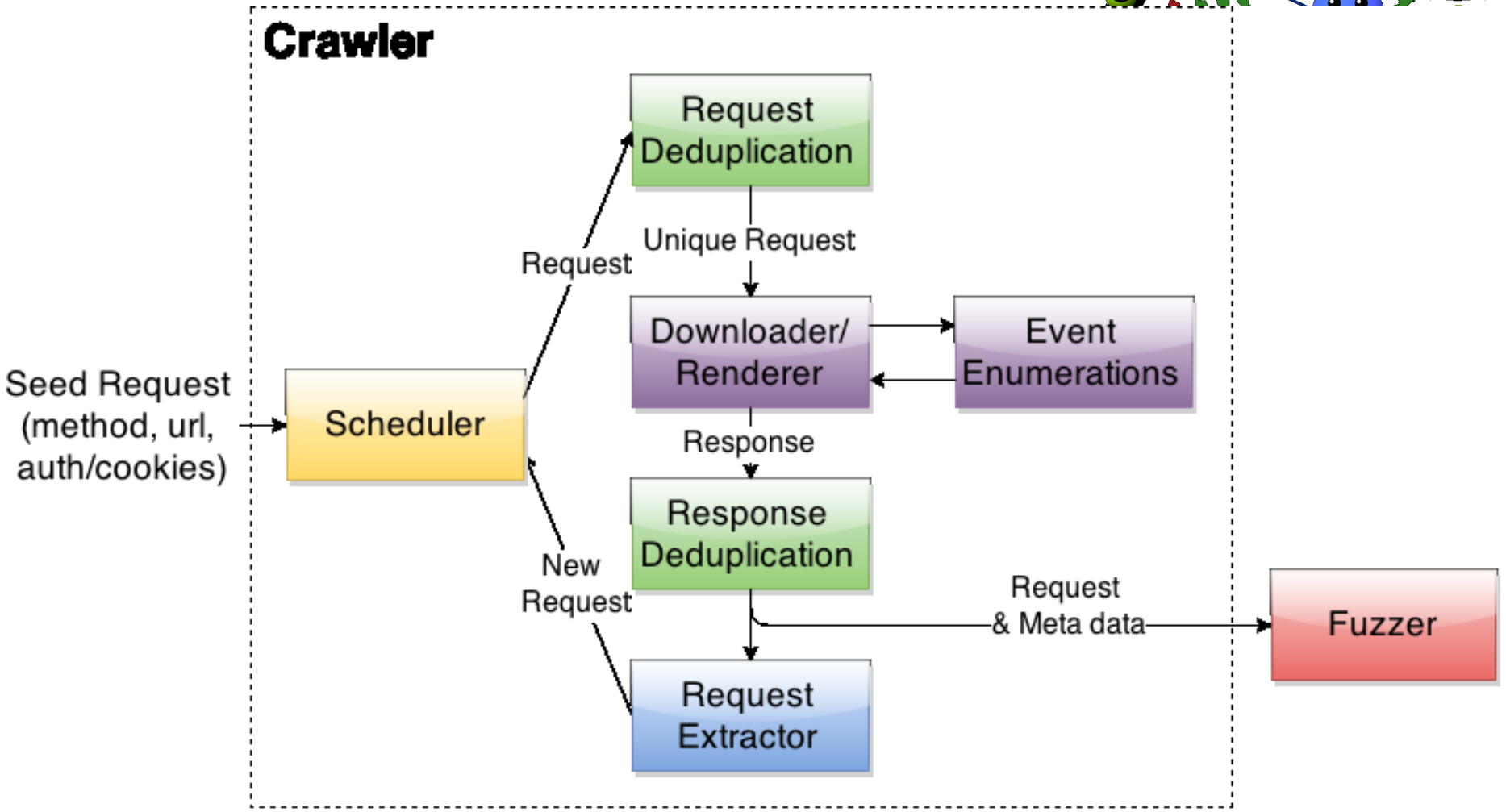
Architecture of Blackbox Scanner



- Crawler
  - Make request to http://example.com
  - Extract Link/Form/XHR/Redirections
  - Store the captured requests to queue
- Fuzzer
  - Malicious input generator replacing request parameters (e.g., query, body, headers)
  - Reproduce the request
- Output Evaluating
  - Mostly pattern matching to check if malicious inputs being reflected or caused an error

Crawling





A.k.a. spider. Works exactly like a search engine, but no need to do rankings

Request Extractor - Crawling

- Hyperlinks by document.getElementsByTagName()
  - <a href="url">, <area href="url">
- Forms by enumerating all key/value possibilities
  - <form method="post" action="url”>
    <input name=“” value=“”>
    <select name=“” multiple></select>
    <textarea name=“”></textarea>
- Redirections by detecting page navigations
  - HTTP 3xx Location Header
  - JavaScript location object
  - HTML meta header
- XHRs by wrapping XmlHTTPRequest objects
  - XMLHttpRequest.prototype.open(method, url, async)
  - XMLHttpRequest.prototype.send(reqBody)

Request/Response Deduplication

- Request Deduplication
  - Observation: Multiple pages might have hyperlinks to the same page
  - Purpose: avoid making duplicated requests
    - At most one req. per signature := (method, URL, sorted req. parameters)
  - Pros: save resources, bandwidth. faster
  - Cons: if state changed at server, might miss newly introduced links
- Response Deduplication
  - Observation: Multiple pages are derived from the same template
  - Purpose: avoid fuzzing similar pages
    - Response features: hyperlinks, tags hierarchy, similarity indexes…
- Historic Response Deduplication
  - Purpose: fuzz only if a page changed (no longer similar)

Downloader/Renderer & Event Enumerations

- Downloader/Renderer
  - Using a Headless Browser (i.e., full-featured, no screen)
  - To save bandwidth, no need to download images and CSS
- Event Enumerations
  - Event capturing and synthesizing amid catching newly generated links
  - Example Events: onload, onclick, onkey, onmouse, setTimeout, etc…
  - Advanced: Workflow event explorations
    - Say, a page is found to have 5 buttons
    - Click button 1, a dialog is shown with more buttons
    - What to click next? Depth-first (New buttons first) v.s. Breath-first
    - Still an open and good research topic!

WIVET Crawling Coverage

Rank	#	Logo	Vulnerability Scanner	Version	Vendor	WIVET Score	Chart
1			WebInspect	10.1.177.0	HP Application Security Center	96.00% Detection Rate	
2			Acunetix WVS	9.0	Acunetix	94.00% Detection Rate	
2			N-Stalker	X	N-Stalker	94.00% Detection Rate	
2			NTOSpider	6.0	NT OBJECTives	94.00% Detection Rate	
2			Syhunt Dynamic	5.0.0.7	Syhunt	94.00% Detection Rate	
2			Tinfoil Security	X	Tinfoil Security	94.00% Detection Rate	
3			Acunetix WVS Free Edition	8.0	Acunetix	92.00% Detection Rate	
3			IBM AppScan	9.0.0.999 / 8.8.0.0	IBM Security Systems Division	92.00% Detection Rate	
3			Netsparker	3.1.7.0	Netsparker Ltd	92.00% Detection Rate	
3			QualysGuard WAS	2014-01-21	Qualys, Inc.	92.00% Detection Rate	
4			Netsparker Community Edition	3.1.6.0	Netsparker Ltd	91.00% Detection Rate	
5			ZAP	2.2.2	OWASP	73.00% Detection Rate	
6			Vega	1.0	Subgraph	50.00% Detection Rate	
7			SkipFish	2.10	Michal Zalewski - Google	48.00% Detection Rate	
8			JSky (Commercial Edition)	3.5.1	NoSec	44.00% Detection Rate	
8			Wapiti	2.3.0	OWASP	44.00% Detection Rate	
9			ProxyStrike	2.2	Edge Security	39.00% Detection Rate	
10			WebSecurify (Opensource Version)	0.9	GNU Citizen	28.00% Detection Rate	
11			arachni	0.4.6	Tasos Laskos	19.00% Detection Rate	
11			ParosPro	1.9.12	MileSCAN Technologies	19.00% Detection Rate	
11			W3AF	1.6	W3AF developers	19.00% Detection Rate	
12			Burp Suite Professional	1.5.20	PortSwigger	16.00% Detection Rate	
12			N-Stalker 2012 Free Edition	10.13.11.31	N-Stalker	16.00% Detection Rate	

Ref: http://sectoolmarket.com/wivet-score-unified-list.html

Fuzzing

- General Approach
  - Reproduce a request/event from crawler
  - With some request parameters mutated
    - Replaced with some (hardcoded) attack vectors for XSS, SQL, etc…
    - a.k.a. malicious input generation
- Adaptive Approach
  - Attack vector list is long, and thus time/bandwidth consuming
  - First round: try if some sensitive chars are reflected or result in error
    - E.g., for XSS, using ><' " to test if any of those sensitive chars are escaped
  - Second round: only if yes, fuzz it with further attack vectors

Evaluating

- General Approach (mostly pattern matching)
  - For every fuzzing attempt, examine the HTTP response
  - Reflected XSS: if the attack vector is reflected (not escaped/altered)
  - Stored XSS: if the attack vector is found in any previous HTTP response
  - SQLi: if the attack vector results in any error message or more/less data
  - etc…
- Other Approach
  - DOM XSS: untrusted inputs rendered on client-side (not from HTML)
  - Tainting: track if a variable propagate from source to critical sinks
    - Instrument a headless browser
    - Special marker on all variables from request parameters
    - Inherit special marker for all string operations (like copy)
    - Marked vars reaching critical sinks (e.g., innerHTML, eval(), etc)
- Coverage: Yahoo's Webseclab, Google's FiringRange, SecToolMarket, etc

Whitebox Vulnerability Scanning

- Testing with knowledge to server-side source code/settings
  - Pattern Matching
    - Check if there're any safe calls (e.g., sanitizer functions)
    - Check if there're any dangerous calls (e.g., dynamic execution like eval())
  - Tainting
    - Given an attack vector coming through req.query / req.body
    - Check if it reaches non-sanitized html contexts (XSS) or SQL calls (SQLi)
  - Control Flow Test
    - Explore all branches that might be traversed
    - Each branch instrumented to check test coverage
  - Symbolic Execution (most computationally expensive)
    - Again, explore all branches that might be traversed
    - Execute a program without a concrete value like tainting
    - Determine what constraints can reach a particular branch (if (s==1) fail())
    - Use a constraint solver to determine the actual value

Whitebox v.s. Blackbox

- Advantages over blackbox approach
  - Enhance/ensure exhaustiveness
    - Statements instrumented to check percentage covered by tests
    - Can literally go through every branch/possibility (e.g., if-then-else)
    - Can help with getting a precise input for exploit generation
- Disadvantages over blackbox approach
  - Approach is language and framework specific
  - May require domain knowledge on the application logic/functionality
  - Does not scale well with complex and large apps
- Greybox: whitebox + blackbox
  - For instance, it's much easier to get an exhaustive sitemap by traversing routes in express/node.js
  - With blackbox alone, it depends on providing all entry points as seeding requests for subsequent crawling

The Invited talk

- Mr. Scottie Tse
  - IE Alumnus
  - Penetration Tester, now with NTT
- Topics
  - Security tools
  - Penetration Testing
  - Certifications
  - Career Information

Topics to be Covered (1/2)

- Web Architecture
  - HTTP, URL, etc

- Web Dev. Languages
  - HTML, CSS, PHP, (No)SQL
  - JavaScript heavy
- Web Dev. Components
  - User Interface Design
  - Forms Handling
  - Database Management
  - Session Management & Auth

- Web App. Security:
  - 8 Security Principles
  - Security Goals: Confidentiality, Integrity, Availability, Auth, Non-repudiation
  - Browser Security Model: SOPs
  - Mashup Devel and Security / Cross-origin Communications
  - Top Application Security Risks

Topics to be Covered (2/2)

- Transport Layer and Browser Security
  - TLS/SSL, PKI, Certificates, Digital Signatures, SSH
  - Cert Pinning, 2FA, XSS Audits, Content Security Policy, Extensions, etc
- Security Testing
  - Penetration Testing
  - Web App Crawling and Scanning

- Building Fast and Scalable WebApp, plus Optimizations
  - Scalability Concerns/Solutions
  - Using Cloud Resources
  - Settings and Code Tweaks
  - Search Engine Optimizations

Covered over 95% of what was promised in lecture 1

What promised on Day 1 and finally achieved

- This course studies the programming and security of web applications.
  - The programming languages for both client- and server-side will be introduced, with security design principles and common vulnerabilities highlighted early on.
  - Open protocols, standards and real-world case studies regarding banking and e-commerce security will be used for illustrations.
  - Optimization and performance issues will also be covered.
- This course also extends to the security threats confronting web browsers, transport protocols and web servers, as well as optionally the mobile and cloud computing.
- Being security-conscious throughout the development cycle, students will have the opportunity to practice with web and mobile programming projects.

Appreciations and Reflections

- **Your Effort is Greatly Appreciated!**
  - Could be a course more demanding than your FYP (as promised, sorry :)
  - You have learned some hottest skills and security thinking
- Painful Yet Rewarding Experience!
  - Every one has a different learning curve, some faster, some a bit slower
  - Majority build a secure e-commerce site in such a short period of time
- Reflections and Wishes
  - Build and Maintain secure web applications
  - Pioneers in the field of security
  - Open start-ups?
  - Stay humble
- Thanks for tolerating my glitches and limitations
  - Your time to do final course evaluations

Final Examination

- Date: Monday 4 May, 2015
- Time: 9:30am-11:30an
- Venue: Sir Run Run Shaw Hall, Central Campus

- Syllabus: Everything excluding this week of lecture

- Open-XXX: Infinite number of non-digital notes and books
- Format: Section I – 24 MCs (38marks)
  Section II – 8 Short Questions (45marks)
  total: 83marks

  (format similar to Final Exam of previous offering,
  download 2012s papers in https://library.cuhk.edu.hk/)

Question Types of Section II (45marks)

1. Compare and Contrast: sth you know ☺ (6)
2. Same-origin policy: legal and illegal cross-origin (6)
3. SOP, XSS and Security Principles (6)
4. Compare and Contrast: again, sth you know ☺ (8)
5. Coding: JavaScript language (3)
6. Coding + Case Study: Clickjacking (4)
7. Case Study: CSRF (7)
8. Coding: Find vulnerability in non-secure code (5)

Some are straightforward, while more requires an analytical mind