OWASP
Open Web Application Security Project

HUNT - Data Driven Web Hacking & Manual Testing

JP Villanueva
Trust & Security Engineer @bugcrowd
@swagnetow

Contributions

Motley crew @bugcrowd
SecEng and SecOps teams
Bug Hunters, Pentesters, Code Analysis
Burp Suite fans
Github contributors

The Problems

1. Increasingly large and complicated web applications that need manual testing
2. Applications assessment training lacks “tribal knowledge” of vulnerability location
3. No in-tool workflow for web hacking methodologies

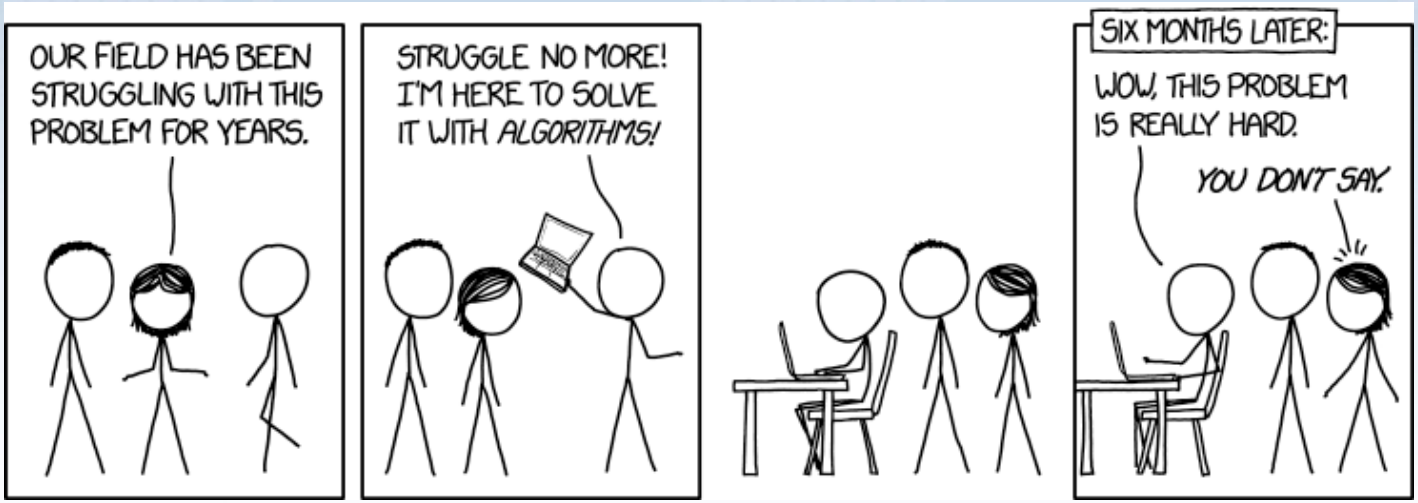
Current Solutions

1. Hacker who can eyeball and effectively find security bugs
    a. May or may not have a methodology
    b. Definitely has accrued “tribal knowledge”
    c. Bug hunts and/or does consultant work
2. Dynamic Scanner
    a. Limited test cases (fuzzing)
    b. Cost prohibitive
    c. Limited in detection cases (dynamic pages, errors, etc)
    d. Complex sites are hard (auth)

Introducing HUNT

1. Tribal knowledge passive alerts
2. Methodology in Burp
3. Manual testing references in Burp

Level 1 - HUNT Scanner



Source: xkcd

Bug Location (Tribal Knowledge)

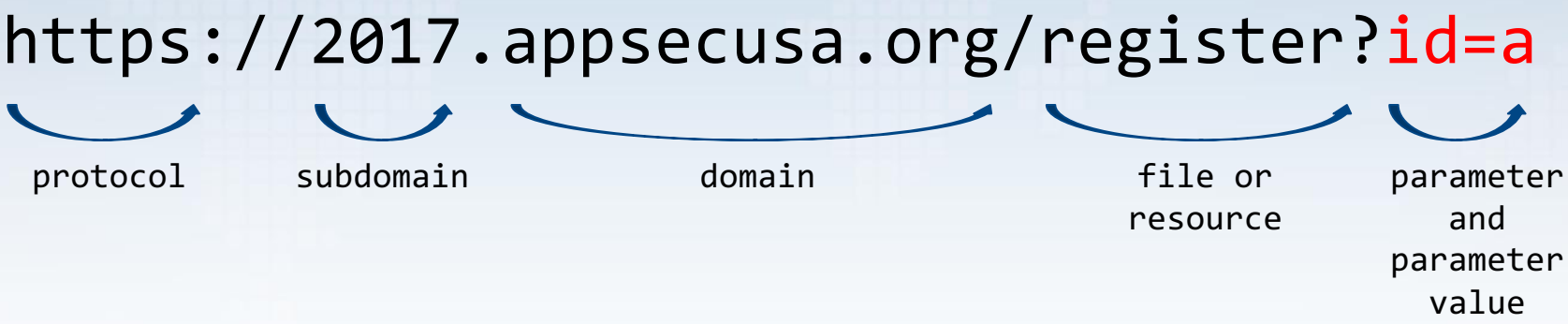
- Data from over 600+ bug bounty programs
  - ~2 web targets per program on average
    - 2017.appsecusa.org, 2016.appsecusa.org
  - ~15 parameters per target on average

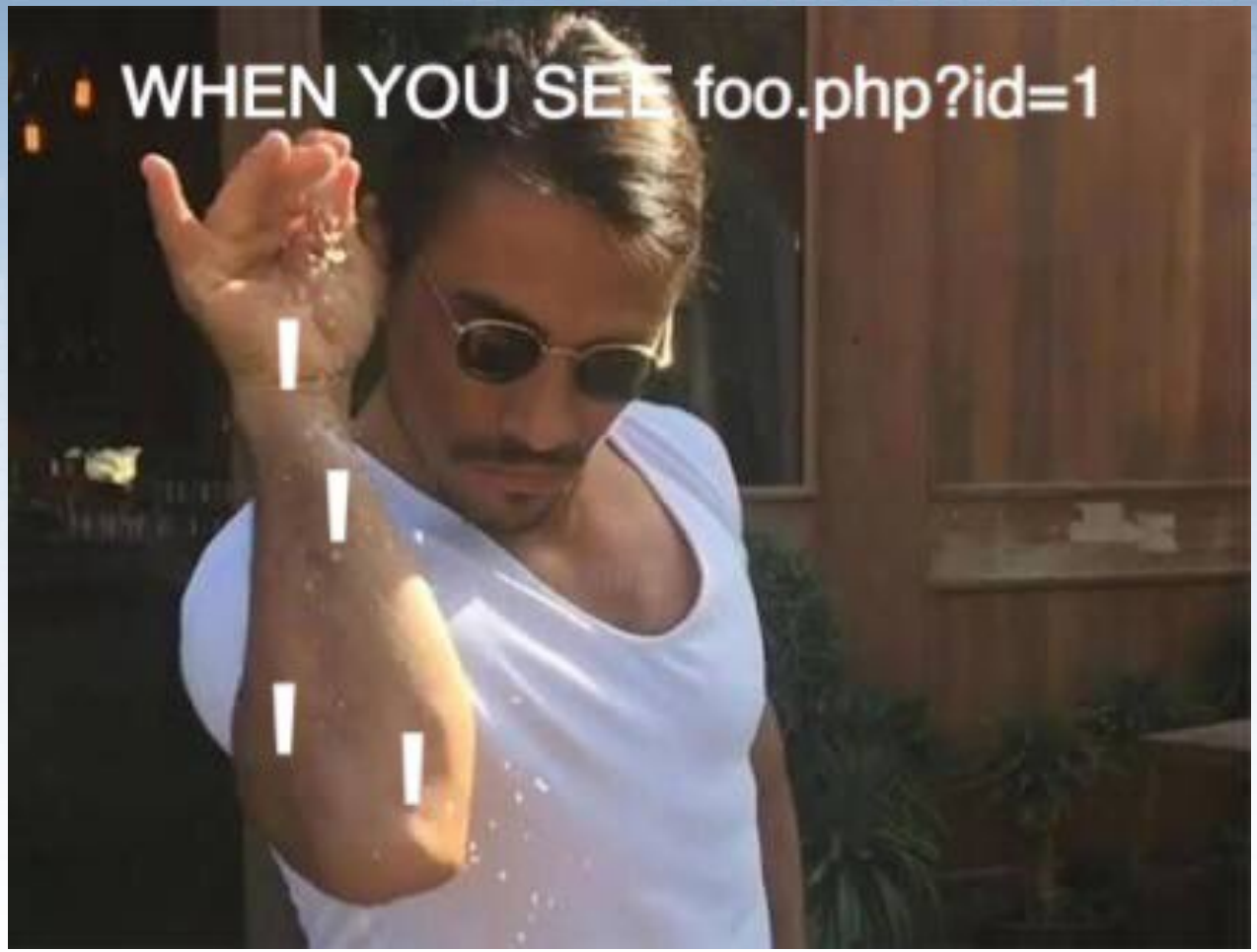
$\Rightarrow$ 600 * 2 * 15 ~= 18,000 parameters seen

Vulnerability Locations

- ⇒ 18,000 parameters
  - Reduce to params with vulns on them
  - Reduce to only Critical and High severity bugs/vulns
  - Sort by recurring instances
  - Include top 5-10 reoccurring instances per vuln/bug category
  - Review top 100 for possible permutations manually and/or with regex
  - Manually add ancillary data (pentest/fuzzdb/seclists/etc)

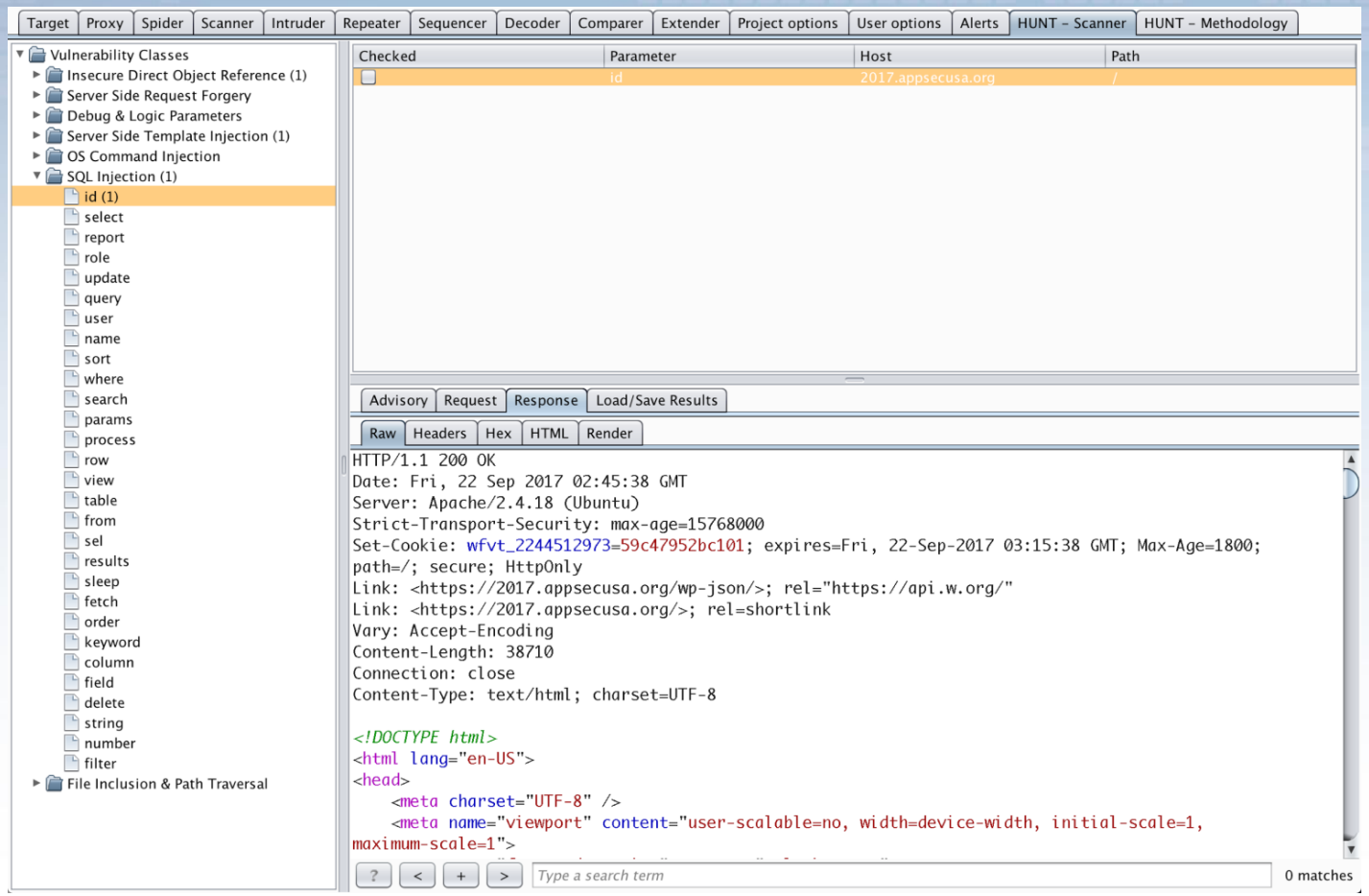
Exhibit A



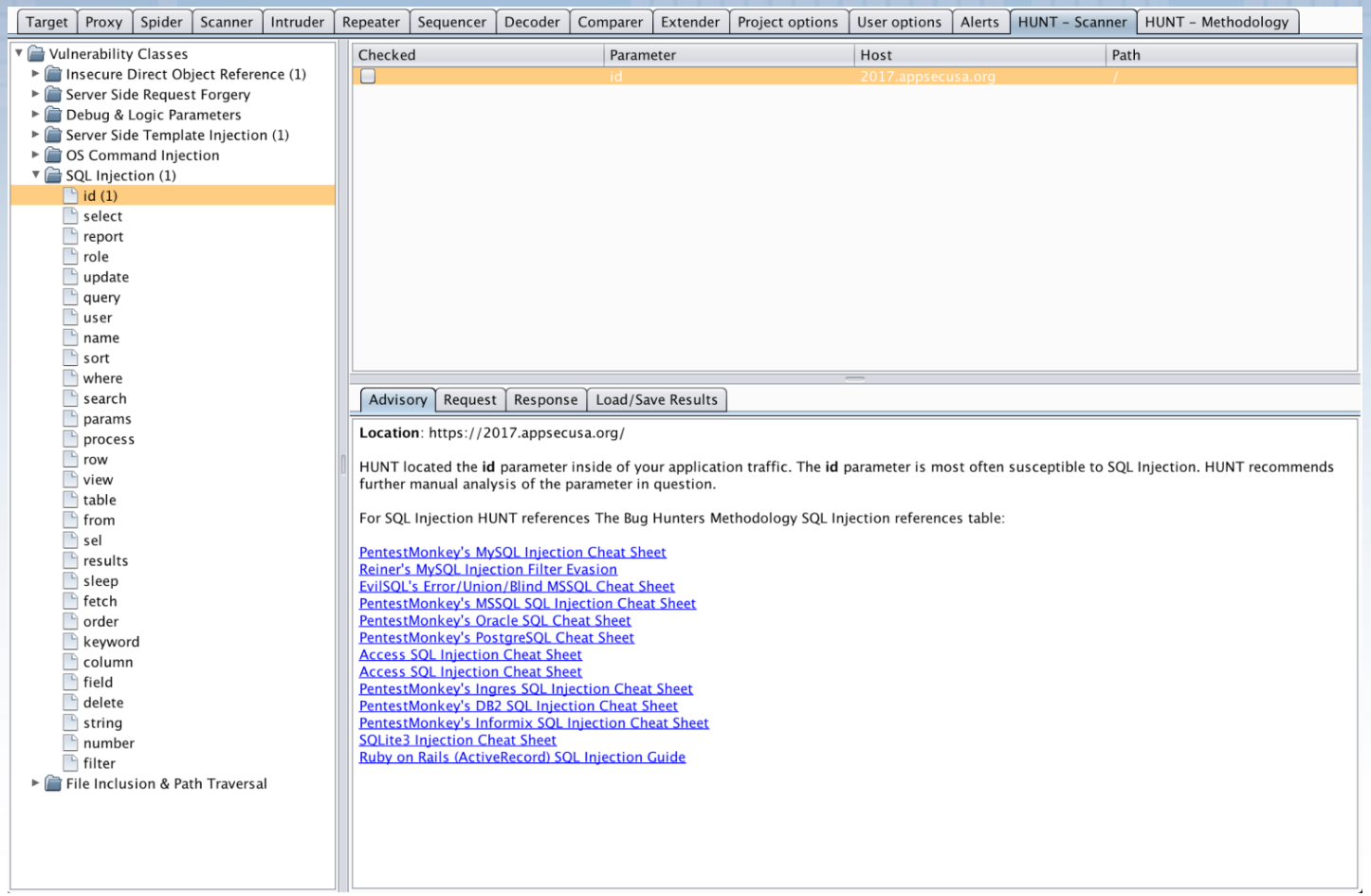


Source: Twitter

Alerts



Advisory



Bug Location by Bug/Vuln Class

Source: Shirtoid.com

SQL Injection

{regex + perm} id	{regex} select	{regex} report	{regex} role
{regex} update	{regex} query	{regex + perm} user	{regex + perm} name
{regex} sort	{regex} where	{regex + perm} search	{regex} params
{regex} process	{regex + perm} row	{regex + perm} view	{regex} table
{regex + perm} from	{regex + perm} sel	{regex} results	{regex} sleep
{regex} fetch	{regex + perm} order	{regex} keyword	{regex} count
{regex + perm} column	{regex} input	{regex + perm} key	
{regex + perm} code	{regex + perm} field	{regex} delete	{*type*} Custom headers
{regex} string	{regex} number	{regex + perm} filter	{*type*} JSON and XML services

File Includes/Directory Indexing

{regex + perm} file	{regex} location	{regex} locale	{regex + perm} path
{regex} display	{regex} load	{regex + perm} read	{regex} retrieve
{regex + perm} folder	{regex} style	{regex + perm} doc	{regex} document
{regex} root	{regex} pdf	{regex} pg	{regex} include
{regex} list	{regex} view	{regex} img	{regex} image

Server Side Request Forgery 🔥🔥🔥

{regex + perm} dest	{regex} redirect	{regex + perm} uri	{regex} path
{regex} continue	{regex + perm} url	{regex} window	{regex} next
{regex} data	{regex} reference	{regex + perm} site	{regex} html
{regex + perm} val	{regex} validate	{regex} domain	{regex} callback
{regex} return	{regex + perm} page	{regex} feed	{regex} host
{regex} port			

OS Command Injection

{regex} daemon	{regex + perm} upload	{regex + perm} dir
{regex} execute	{regex + perm} download	{regex + perm} log
{*type*} .cgi	{regex} ip	
{regex} cli		

Insecure Direct Object Reference

{regex + perm} id	{regex + perm} user	
{regex + perm} account	{regex + perm} number	
{regex + perm} order	{regex + perm} no	
{regex + perm} doc	{regex + perm} key	
{regex + perm} email	{regex + perm} group	
{regex + perm} profile	{regex + perm} edit	REST numeric paths

Server Side Template Injection

{regex + perm} template	content	id
preview	redirect	view
activity	name	

Debug & Logic Parameters

access	admin	dbg
debug	edit	grant
test	alter	clone
create	delete	disable
enable	exec	execute
load	make	modify
rename	reset	shell
toggle	adm	root
cfg	config	

HUNT Scanner Implementation

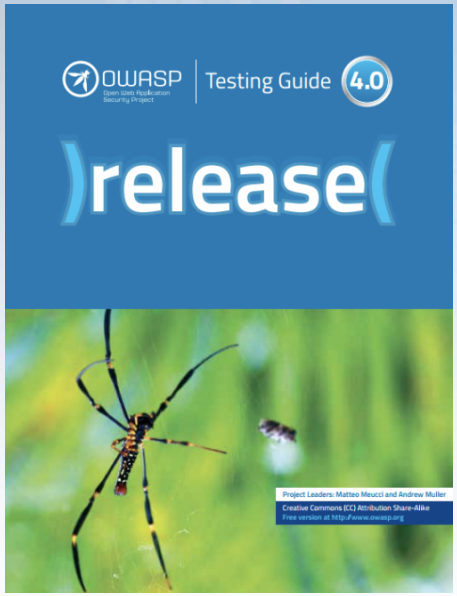
```
IBurpExtender, IExtensionStateListener, IScannerCheck, ITab, ITextEditor

def doPassiveScan(self, request_response):
        raw_request = request_response.getRequest()
        raw_response = request_response.getResponse()
        request = self.helpers.analyzeRequest(raw_request)
        response = self.helpers.analyzeResponse(raw_response)

        parameters = request.getParameters()
        url = self.helpers.analyzeRequest(request_response).getUrl()
        vuln_parameters = self.issues.check_parameters(self.helpers, parameters)

        is_not_empty = len(vuln_parameters) > 0

        if is_not_empty:
            self.issues.create_scanner_issues(self.view, self.callbacks, self.helpers, vuln_parameters, request_response)

        # Do not show any Bugcrowd found issues in the Scanner window
        return []
```

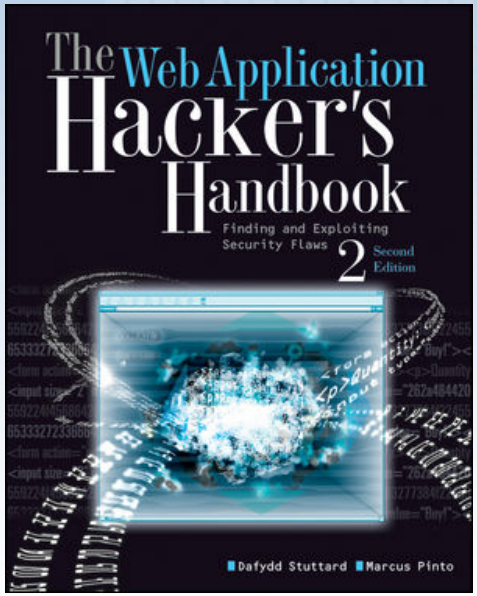
Live Demo: PLEASE WORK PLEASE

Level 2 - HUNT Methodology

Source: Capcom

Methodologies



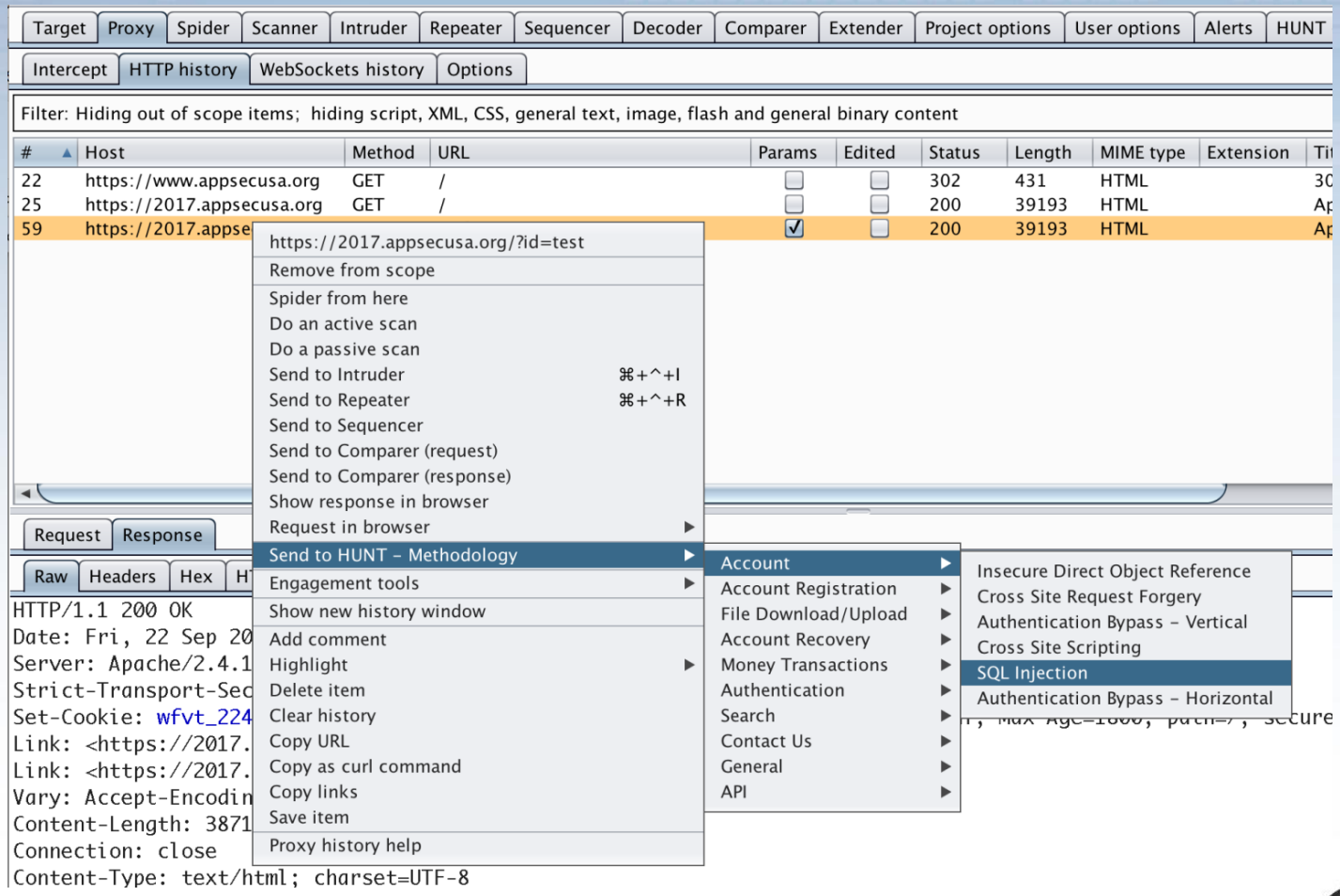


HIPAA

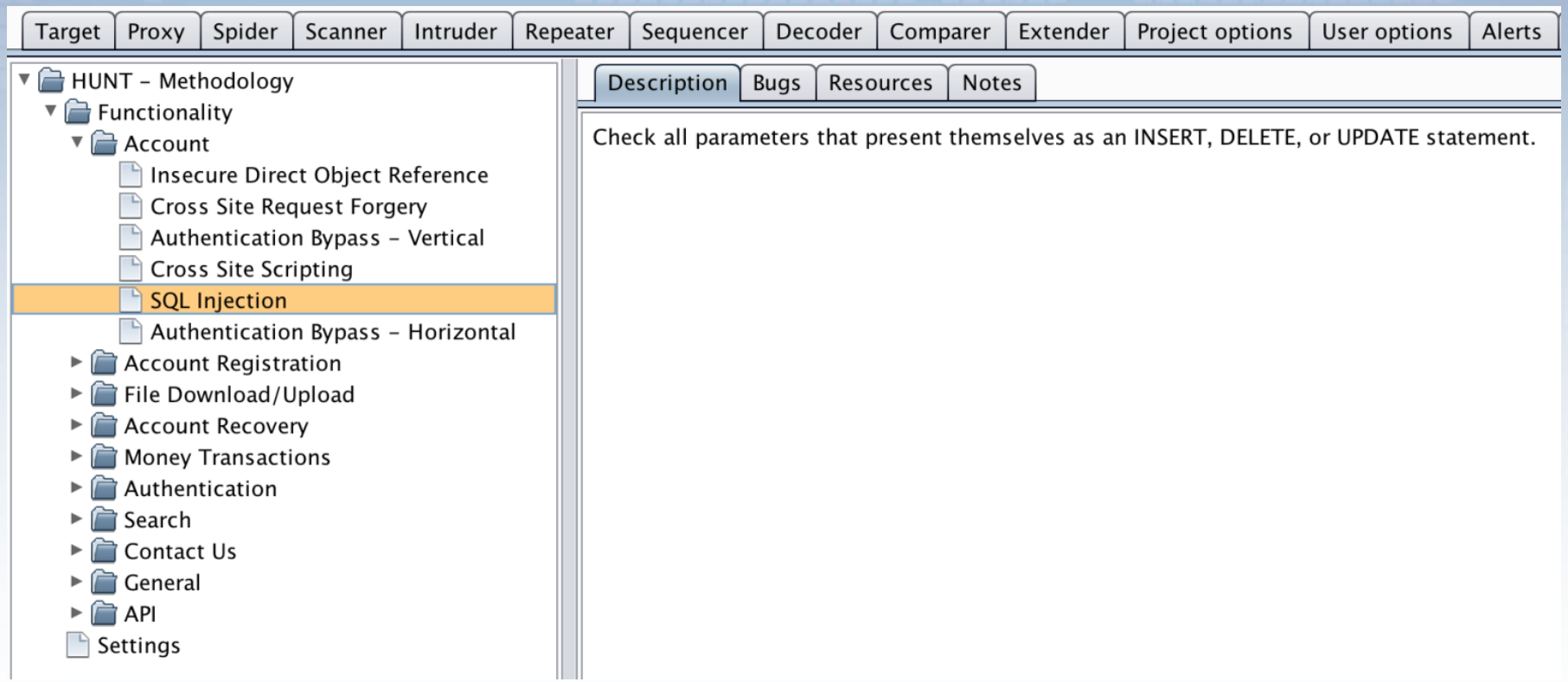
PCI

OWASP
Open Web Application
Security Project

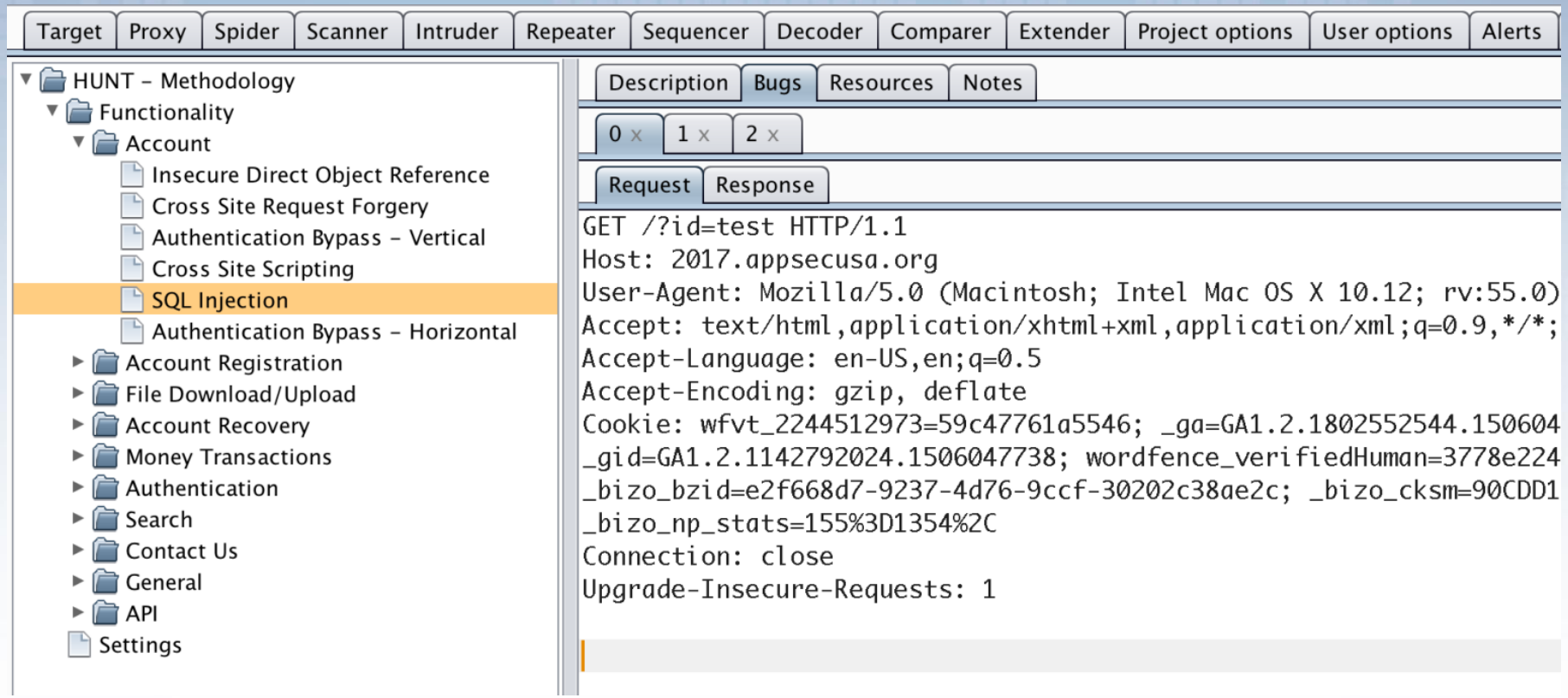
Right Click -> Send-To Methodology



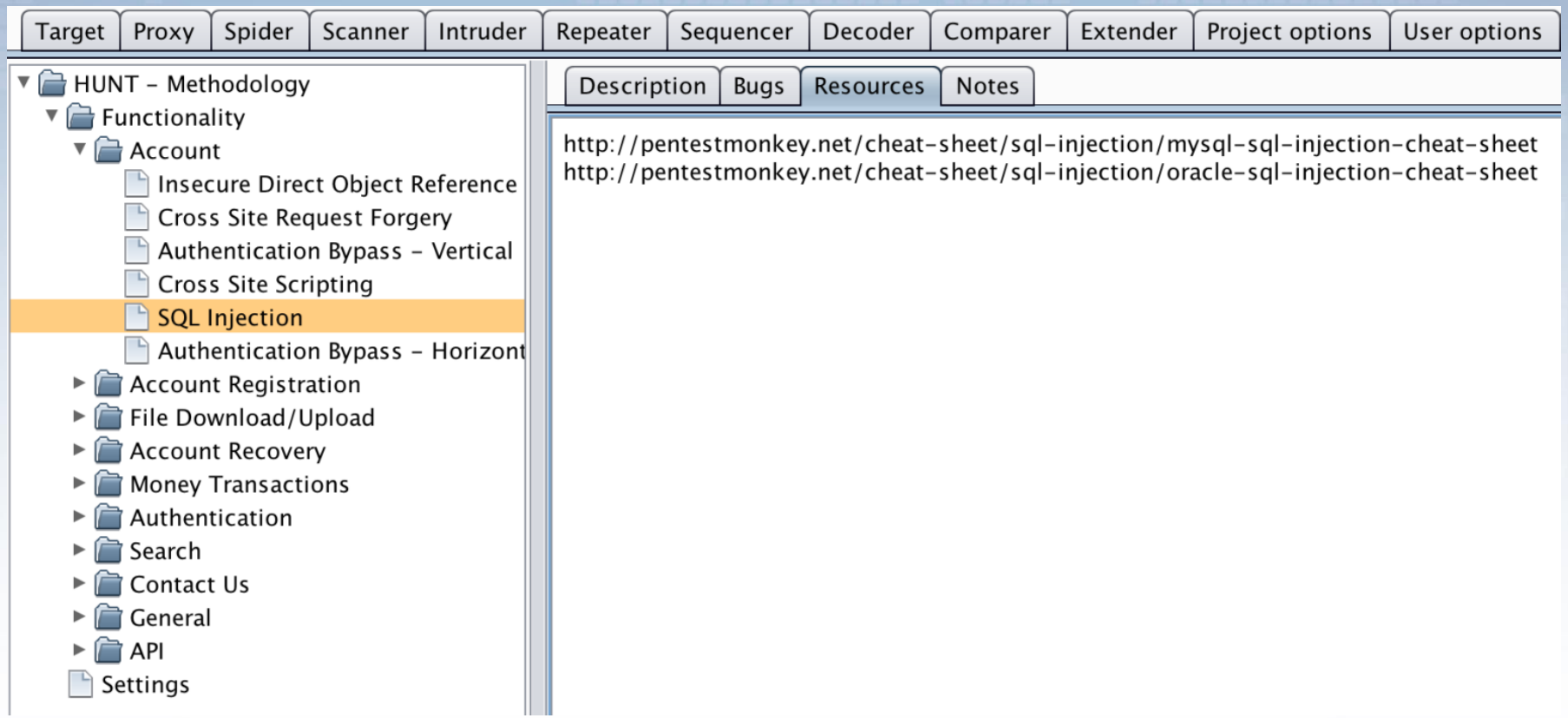
Description



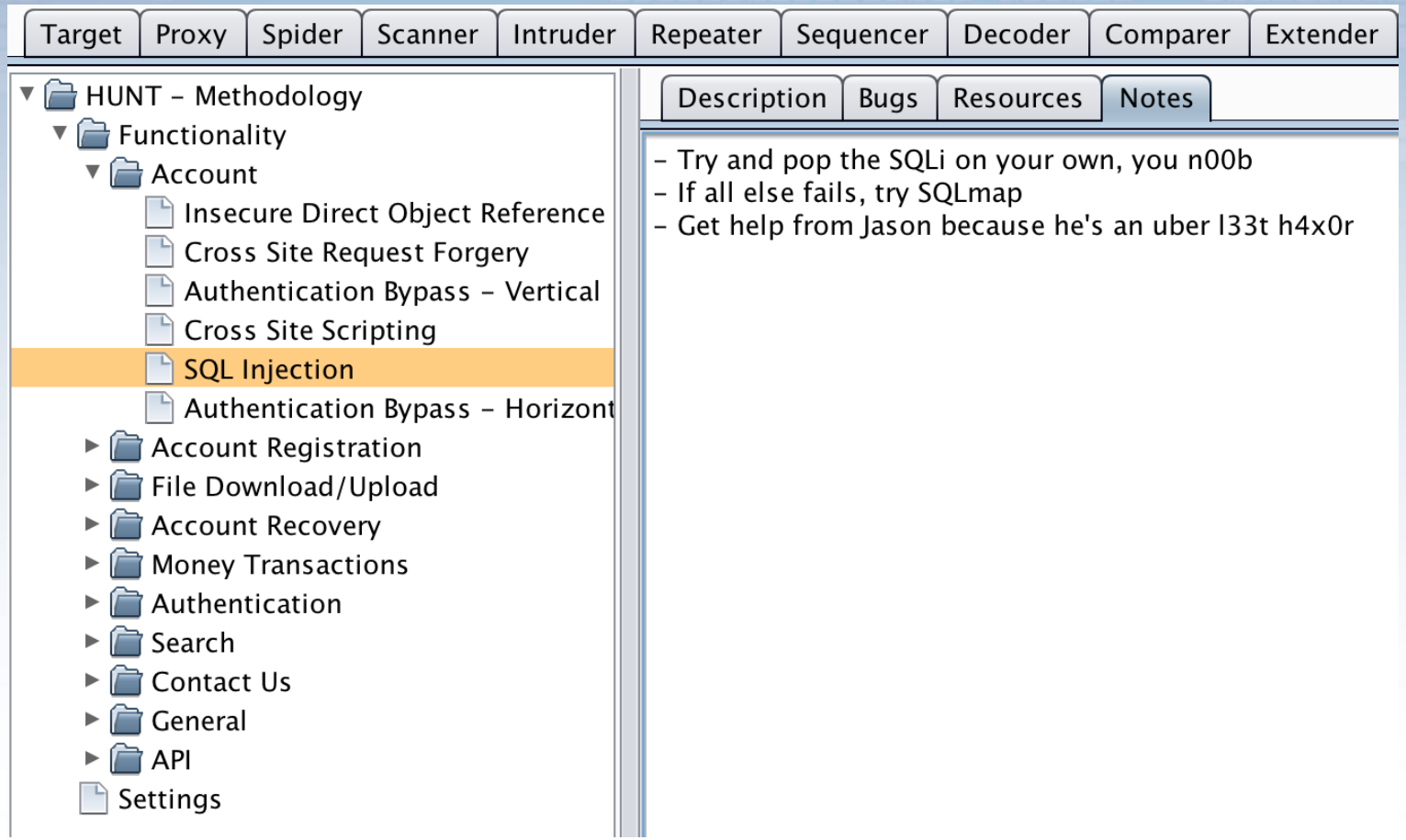
Multiple Request/Response



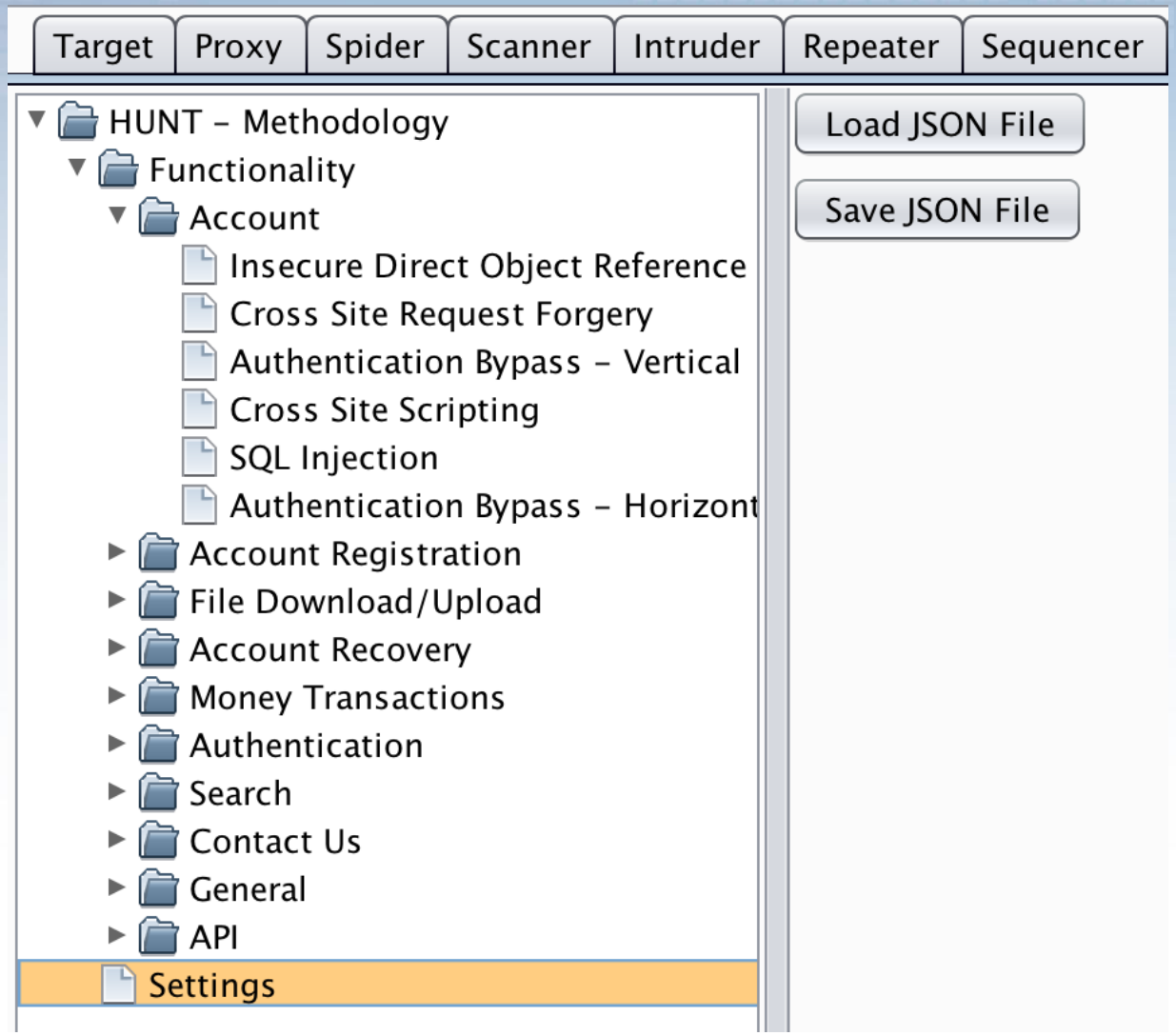
Resources


Target
Proxy
Spider
Scanner
Intruder
Repeater
Sequencer
Decoder
Comparer
Extender
Project options
User options
HUNT - Methodology
Functionality
Account
Insecure Direct Object Reference
Cross Site Request Forgery
Authentication Bypass - Vertical
Cross Site Scripting
SQL Injection
Authentication Bypass - Horizont
Account Registration
File Download/Upload
Account Recovery
Money Transactions
Authentication
Search
Contact Us
General
API
Settings
Description
Bugs
Resources
Notes
http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
http://pentestmonkey.net/cheat-sheet/sql-injection/oracle-sql-injection-cheat-sheet

Notes


Target
Proxy
Spider
Scanner
Intruder
Repeater
Sequencer
Decoder
Comparer
Extender
HUNT - Methodology
Functionality
Account
Insecure Direct Object Reference
Cross Site Request Forgery
Authentication Bypass - Vertical
Cross Site Scripting
SQL Injection
Authentication Bypass - Horizont
Account Registration
File Download/Upload
Account Recovery
Money Transactions
Authentication
Search
Contact Us
General
API
Settings
Description
Bugs
Resources
Notes
- Try and pop the SQLi on your own, you n00b
- If all else fails, try SQLmap
- Get help from Jason because he's an uber l33t h4x0r

Save/Load JSON File



OWASP
Open Web Application Security Project

Methodology Implementation

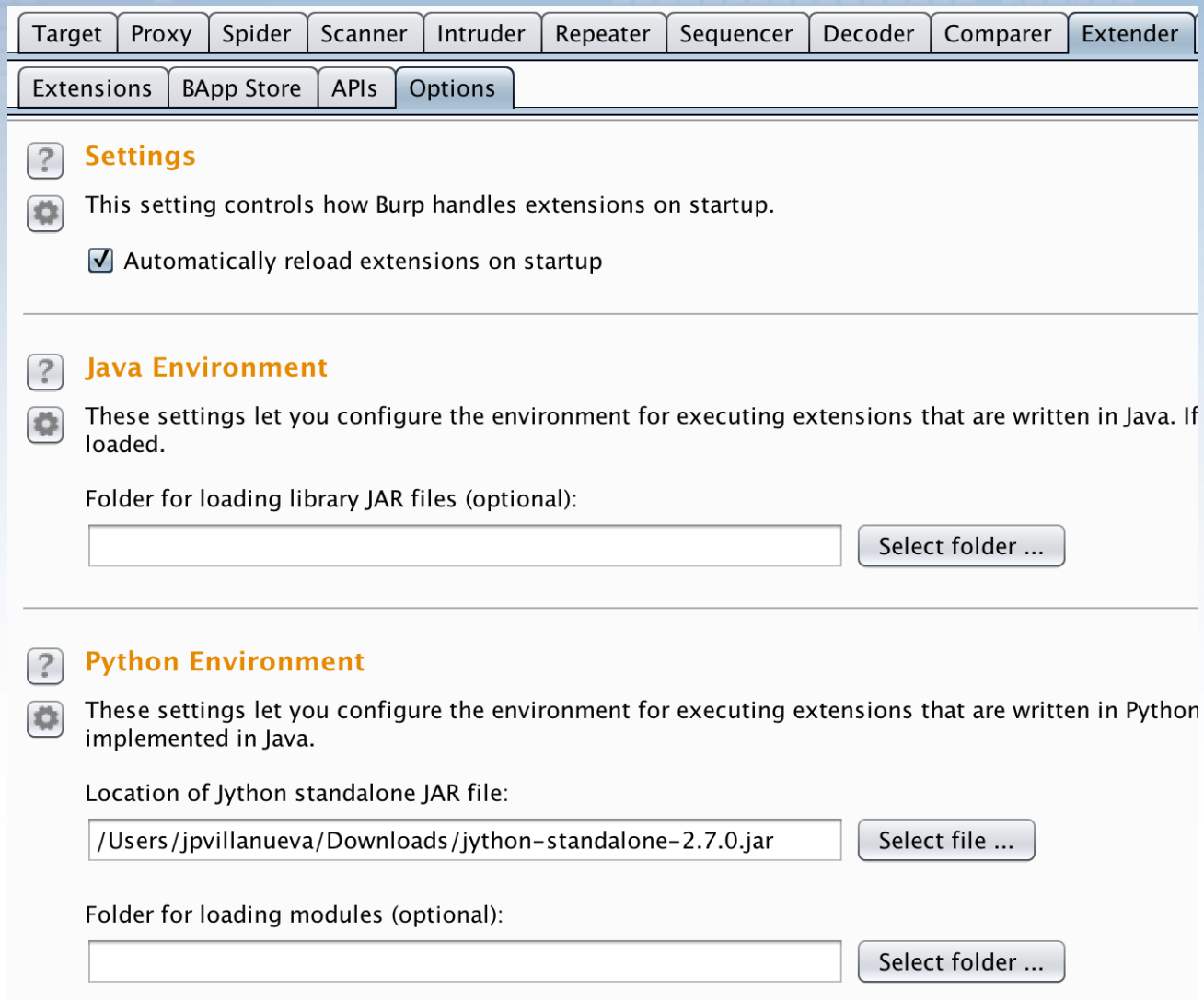
```
IExtensionStateListener, IContextMenuFactory, ITab

def createMenuItems(self, invocation):
functionality = self.checklist["Functionality"]

        # Create the menu item for the Burp context menu
        bugcatcher_menu = JMenu("Send to HUNT - Methodology")

        for functionality_name in functionality:
            vulns = functionality[functionality_name]["vulns"]
            menu_vuln = JMenu(functionality_name)

            # Create a menu item and an action listener per vulnerability
            # class on each functionality
            for vuln_name in vulns:
                item_vuln = JMenuItem(vuln_name)
                menu_action_listener = MenuActionListener(self.view, self.callbacks, request_response,
functionality_name, vuln_name)
                item_vuln.addActionListener(menu_action_listener)
                menu_vuln.add(item_vuln)

            bugcatcher_menu.add(menu_vuln)

        burp_menu = []
        burp_menu.append(bugcatcher_menu)

        return burp_menu
```

Live Demo: Part Deux

Plugin Installation

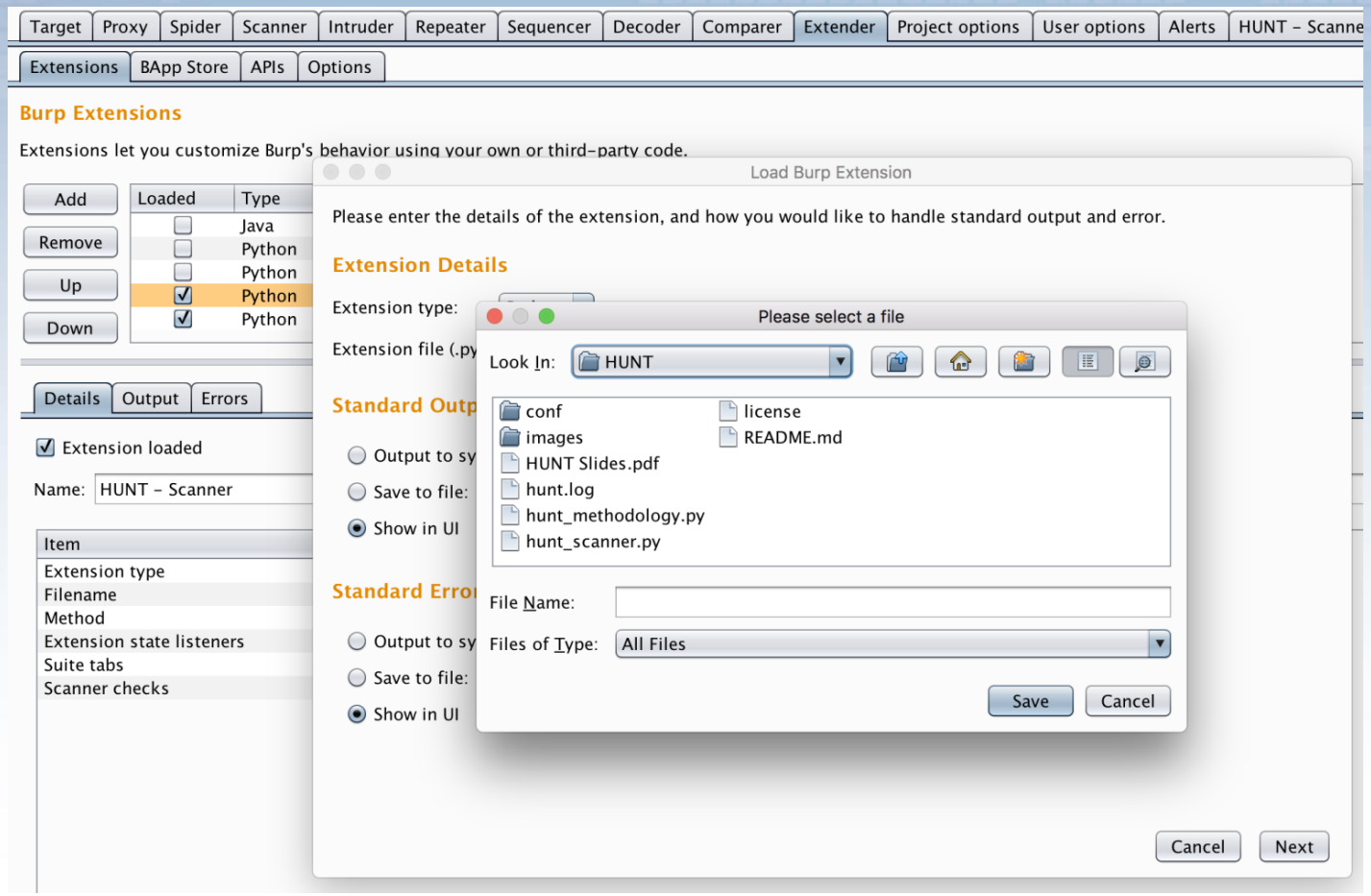
Source: github

Installation - Jython

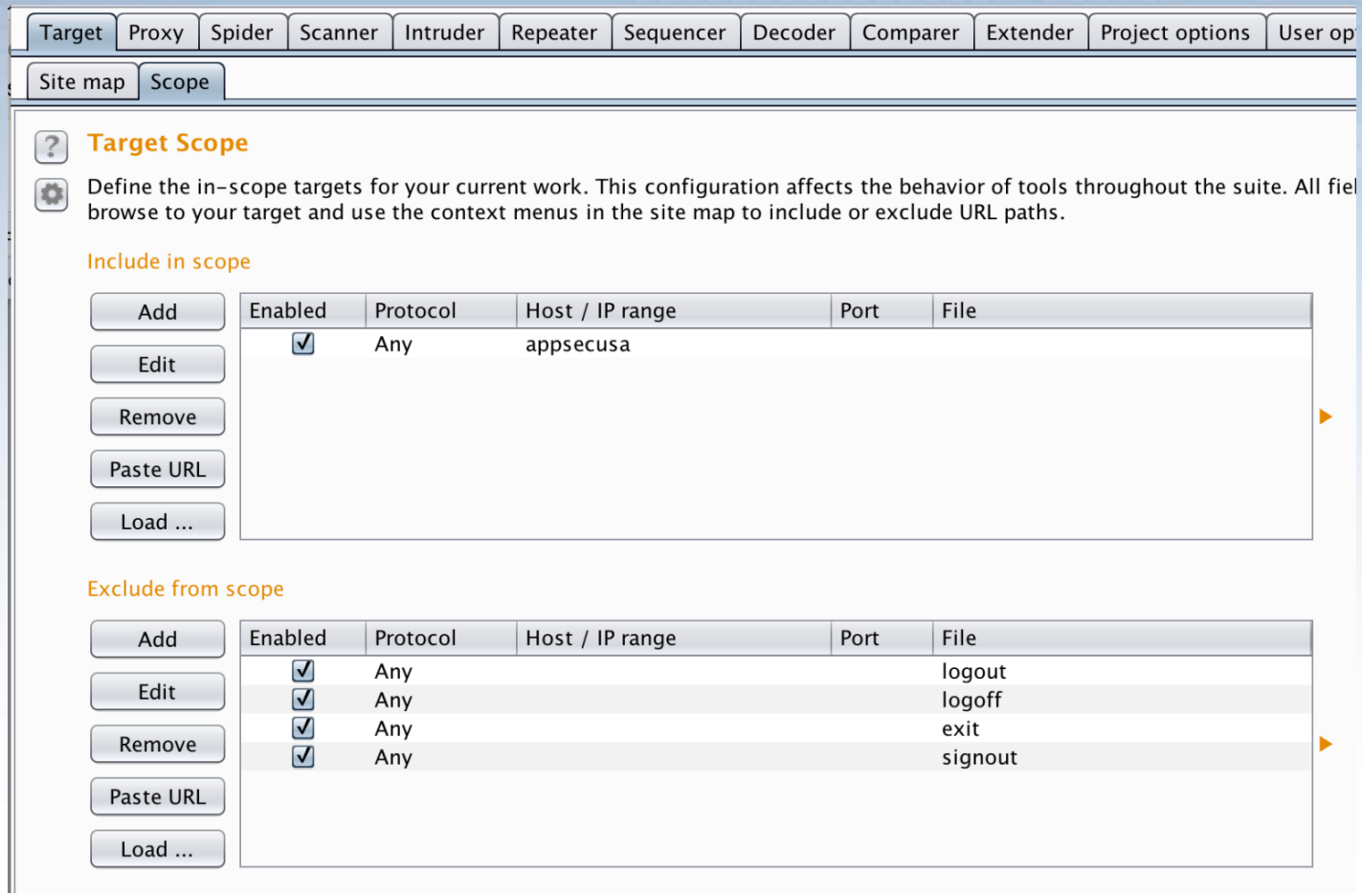


OWASP
Open Web Application Security Project

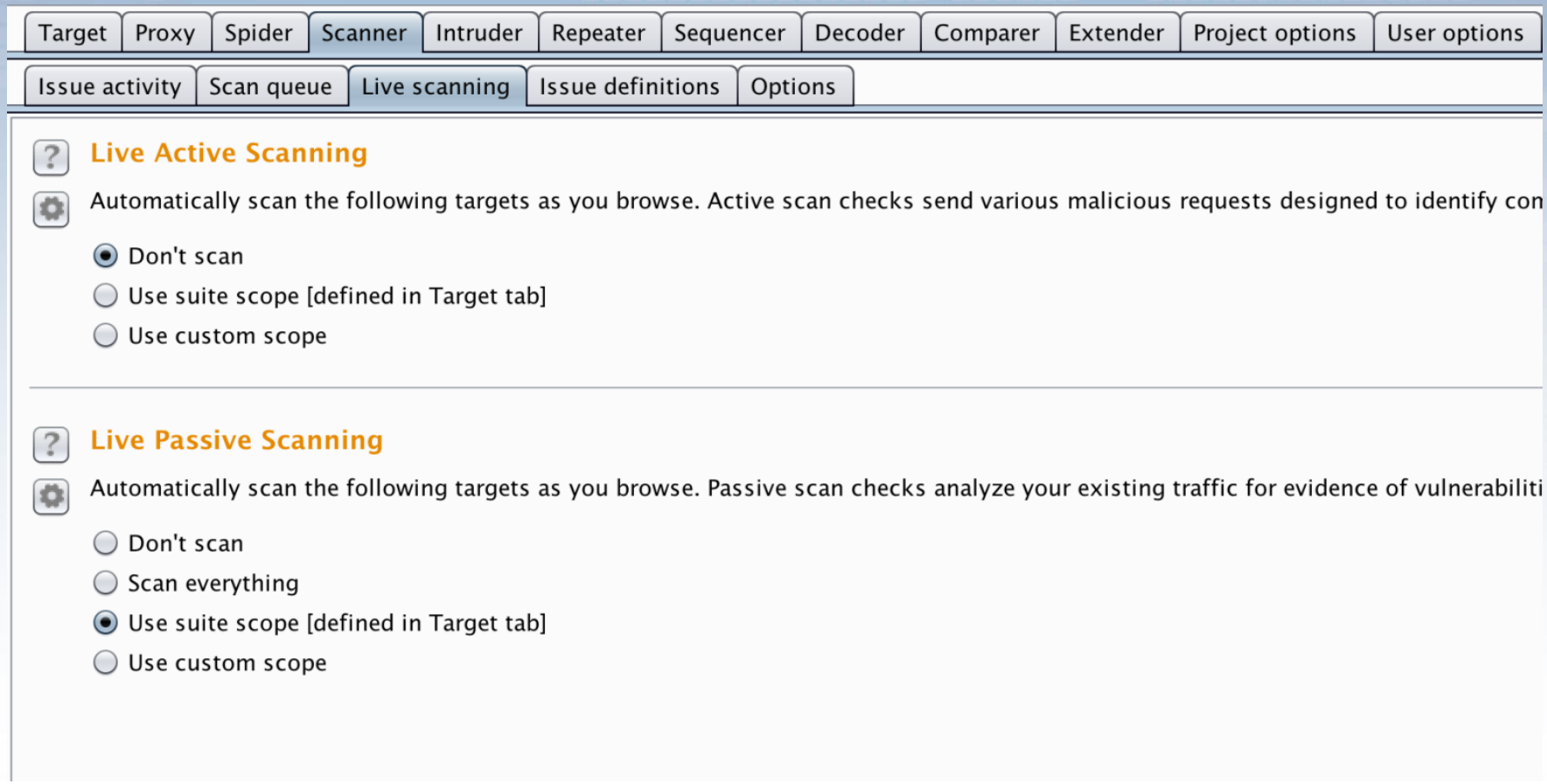
Installation - Plugin



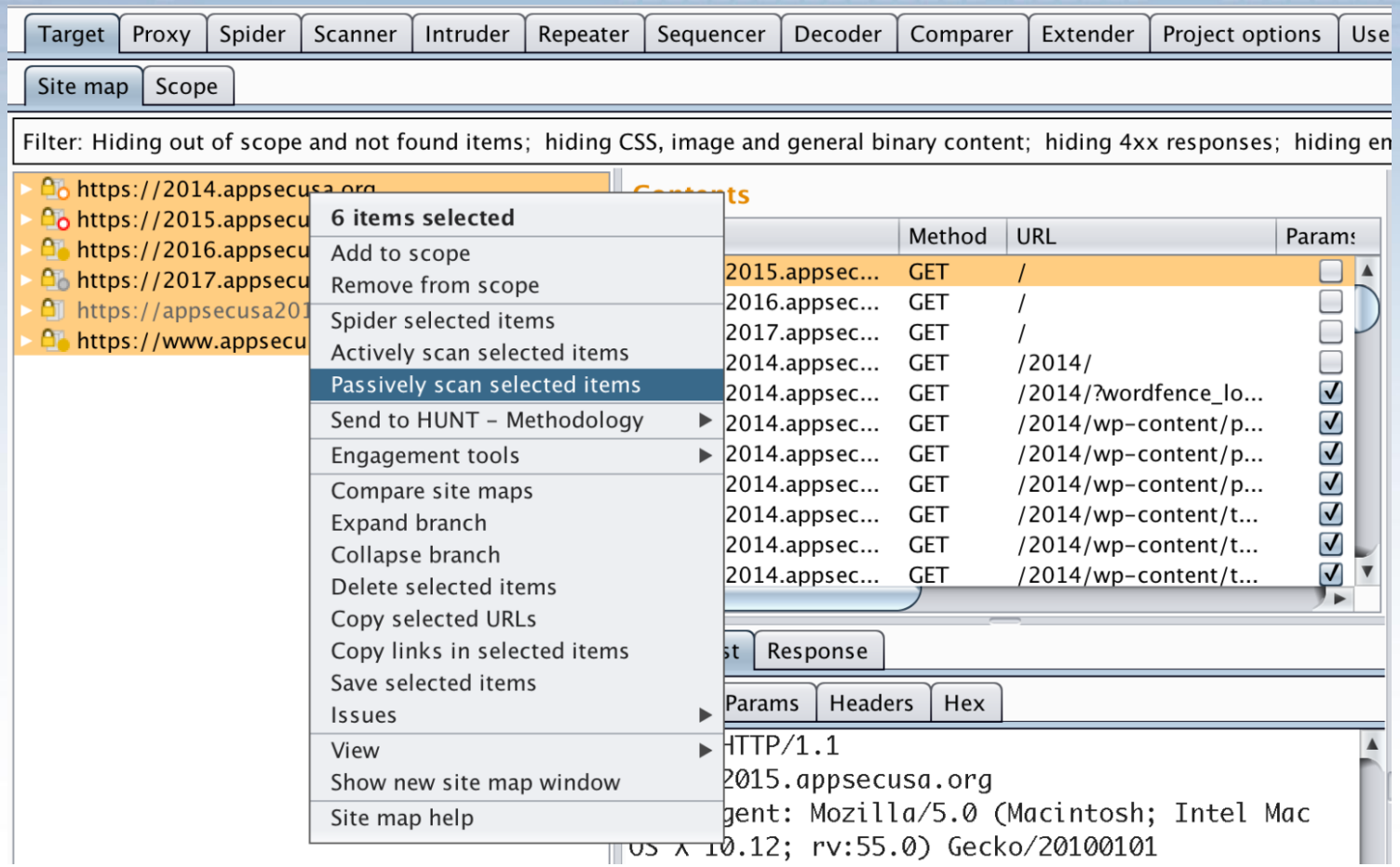
Setting Target Scope



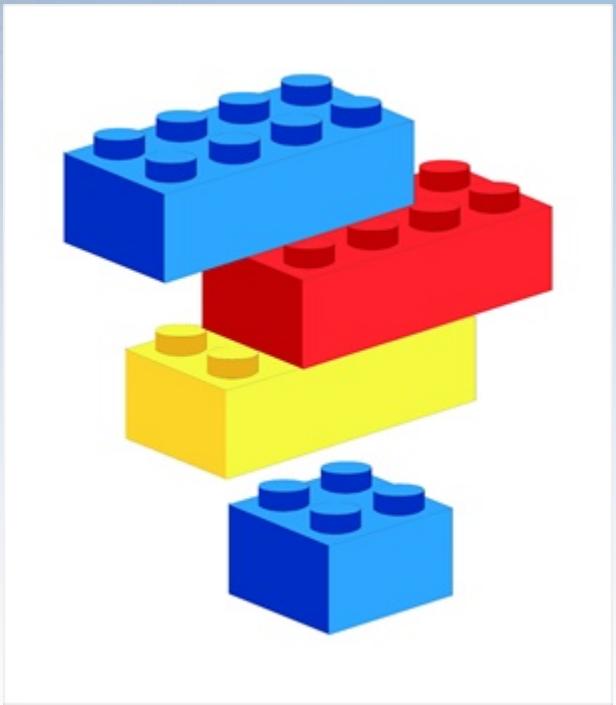
Setting Passive Scanner Scope



Running the Passive Scanner



Extensibility



Source: logic-canvas.com

Scanner Extensibility

Creating new issue checks are as simple as adding to the JSON file.

```
{
  "issues": {
    "OS Command Injection": {
      "check_location": {
        "request": true,
        "response": false
      },
      "detail": "HUNT located the <b>$param$</b> parameter
inside of your application traffic. The <b>$param$</b>
parameter is most often susceptible to OS Command Injection.
HUNT recommends further manual analysis of the parameter in
question.<br><br>For OS Command Injection HUNT recommends the
following resources to aid in manual testing:",
      "level": "Information",
      "name": "Possible OS Command Injection",
      "params": [
        "daemon",
        "upload",
        "dir",
        "execute",
        "download",
        “vulnerable_parameter”
      ]
    }
}
```

OWASP
Open Web Application Security Project

Methodology Extensibility

Creating new methodologies are as simple as adding to the JSON file.

```
{
  "checklist": {
    "Settings": "",
    "Functionality": {
      "NEW METHODOLOGY SECTION": {
        "description": "",
        "tests": {
          "Authentication Bypass - Vertical": {
            "description": "Check to see if the login sequence
can be bypassed in any way to get higher level permissions.",
            "resources": [],
            "bugs": [],
            "notes": ""
          }
        }
      }
    }
  }
}
```

OWASP
Open Web Application Security Project

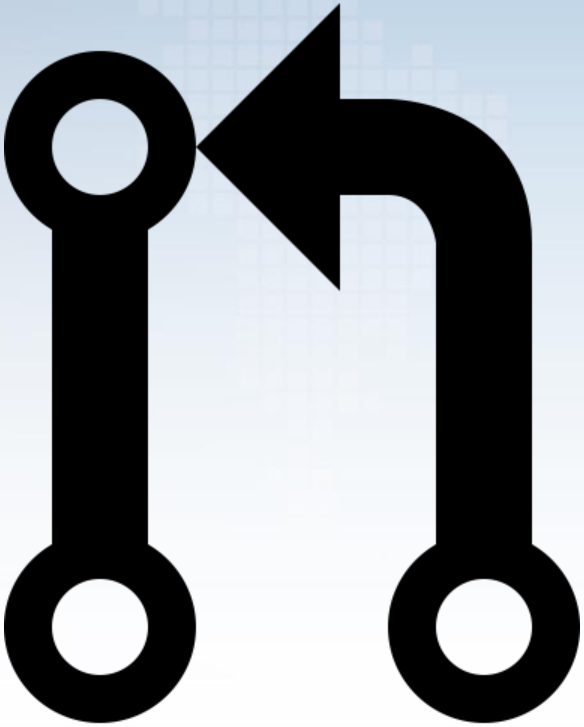
Live Demo: Electric Boogaloo

The Future

- More built-in methodologies
  - OWASP, PCI, HIPAA, CREST, PTES
- Port to ZAP?
- More scanner checks/vulnerability classes
- More resources
- Dynamic JSON structure support
- Perfect GUI lol
- REST Support
- ~~Full Burp helpers (right click, search, highlight, etc)~~
- Resource/File name analysis (Instead of params)
- Alerts on content types (XML, JSON, Multipart-form)
- Response analysis alerts (errors)

Questions?

https://www.github.com/bugcrowd/hunt



OWASP
Open Web Application Security Project