

Security

Vulnerability Testing

Vulnerabilities

- Vulnerabilities appear everywhere in the stack
 - Modern systems are very large and complex
 - Impossible to test all possible use cases in advance
- Long history of
 - Network protocol vulnerabilities
 - OS vulnerabilities
 - Application vulnerabilities
 - Browsers, web servers, database mgmt systems, mail programs
 - Web apps (see OWASP Top 10)
 - Mobile apps
- Also non-technical vulnerabilities”
 - Social engineering
 - Illness, loss of personnel
 - Power failure, comms problems, fire, flood, earthquake, ...

Tracking vulnerabilities

- **Repositories**
 - **CVE: Common Vulnerabilities and Exposures**
 - Unique ID assigned to each vulnerability identified, e.g. CVE-2017-7269
 - <https://cve.mitre.org/>
 - **CWE: Common Weakness Enumeration** (cwe.mitre.org)
 - **CVSS: Common Vulnerability Scoring System**
 - For assessing severity of a problem
 - **National Vulnerability Database (NVD)**
 - **SecurityFocus**
 - **SANS Internet Storm Center**
 - **CERT (Computer Emergency Response Team)**
 - **Anti-malware vendors (Symantec, Kaspersky, AVG, etc)**

Software Vulnerability Testing

- Find flaws in the code early
- Many different techniques
 - Static (against source or compiled code)
 - Security focused static analysis tools
 - Peer review process
 - Formal security code review
 - Dynamic (against running code)
 - Scanning
 - Penetration testing
- Goal
 - Ensure completeness (across all vulnerability areas)
 - Ensure accuracy (minimize false alarms)

Software Vulnerability Testing

DAST:
Dynamic Application
Security Testing
(focus on running app)

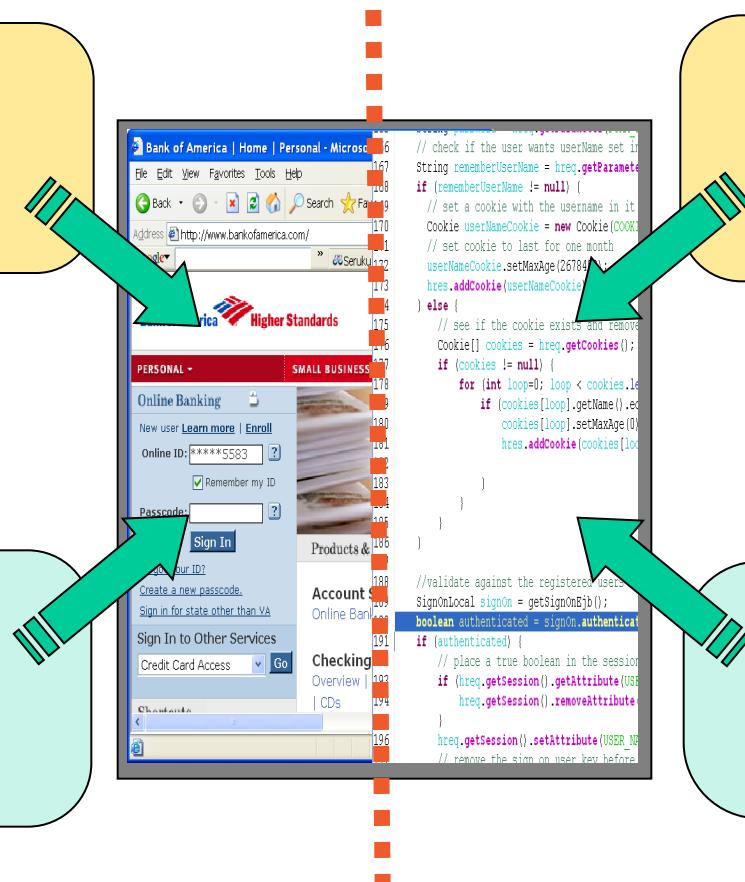
SAST:
Static Application
Security Testing
(focus on source code)

Manual
Penetration
Testing

Manual
Code
Review

Automated
Vulnerability
Scanning

Automated
Static Code
Analysis



Secure Code Review

- Static / dynamic analysis tools available
- Requires manual inspection too
- Benefits:
 - Improves code quality
 - Prevents security bugs
 - Increased developer awareness and understanding

Vulnerability Patterns

```
public class DamagedStrutsForm extends ActionForm
{
    public void doForm( HttpServletRequest request) {
        UserBean u = session.getUserBean();
        u.setName(request.getParameter("name"));
        u.setFavoriteColor(request.getParameter("color"));
    }

    public boolean validate( HttpServletRequest request) {
        try {
            if ( request.getParameter("name").indexOf("<scri" ) != -1 ) {
                logger.log("Script detected" );
                return false;
            }
        } catch( Exception e ) {}
        return true;
    }
}
```

The diagram illustrates four specific security vulnerabilities found in the provided Java code:

- Failure to Validate**: Two annotations point to the assignments of `name` and `color` from the `HttpServletRequest`. These assignments directly set the values without any prior validation.
- Blacklist Validation**: An annotation points to the `indexOf` check for the string "`<scri`". This is a form of blacklist validation where specific known malicious patterns are checked for.
- Fail Open**: An annotation points to the `return true;` statement at the end of the `validate` method. This indicates that any exception caught during validation will result in the method returning `true`, effectively bypassing validation.

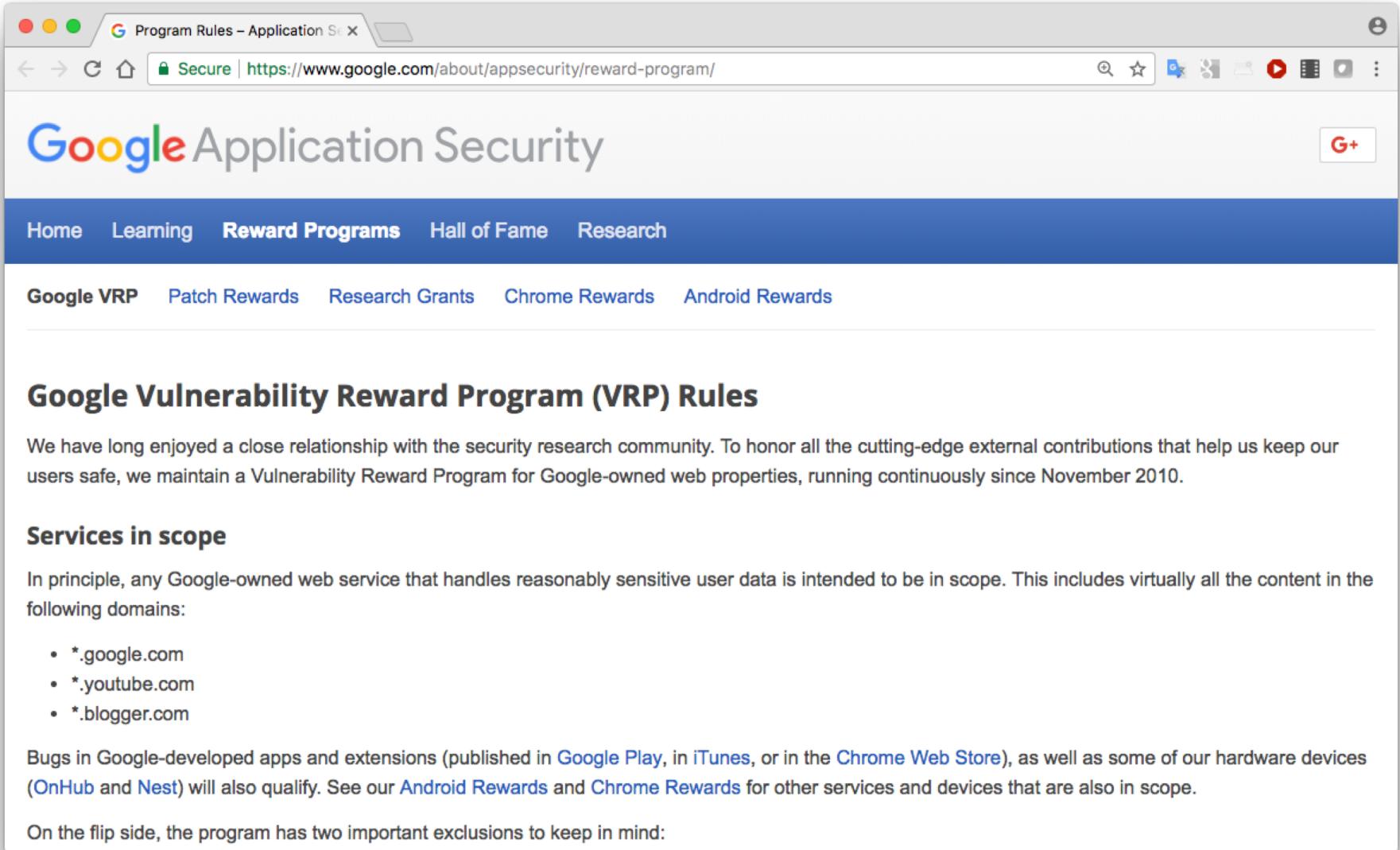
Ethical Hacking

Ethical Hacking...

- Also known as **Penetration Testing**
- Searching for weaknesses and vulnerabilities
- Trying out known attacks
- Authorised breaking into systems
 - You MUST have permission from the system's owner!



Some companies offer bounties...



A screenshot of a web browser window displaying the Google Application Security Reward Program rules. The browser interface includes standard controls like back, forward, and search, along with social sharing icons for Google+ and others.

The main content area shows the Google logo, the title "Google Application Security", and a navigation bar with links to Home, Learning, Reward Programs (which is the active tab), Hall of Fame, and Research. Below this, a secondary navigation bar lists "Google VRP", "Patch Rewards", "Research Grants", "Chrome Rewards", and "Android Rewards".

Google Vulnerability Reward Program (VRP) Rules

We have long enjoyed a close relationship with the security research community. To honor all the cutting-edge external contributions that help us keep our users safe, we maintain a Vulnerability Reward Program for Google-owned web properties, running continuously since November 2010.

Services in scope

In principle, any Google-owned web service that handles reasonably sensitive user data is intended to be in scope. This includes virtually all the content in the following domains:

- *.google.com
- *.youtube.com
- *.blogger.com

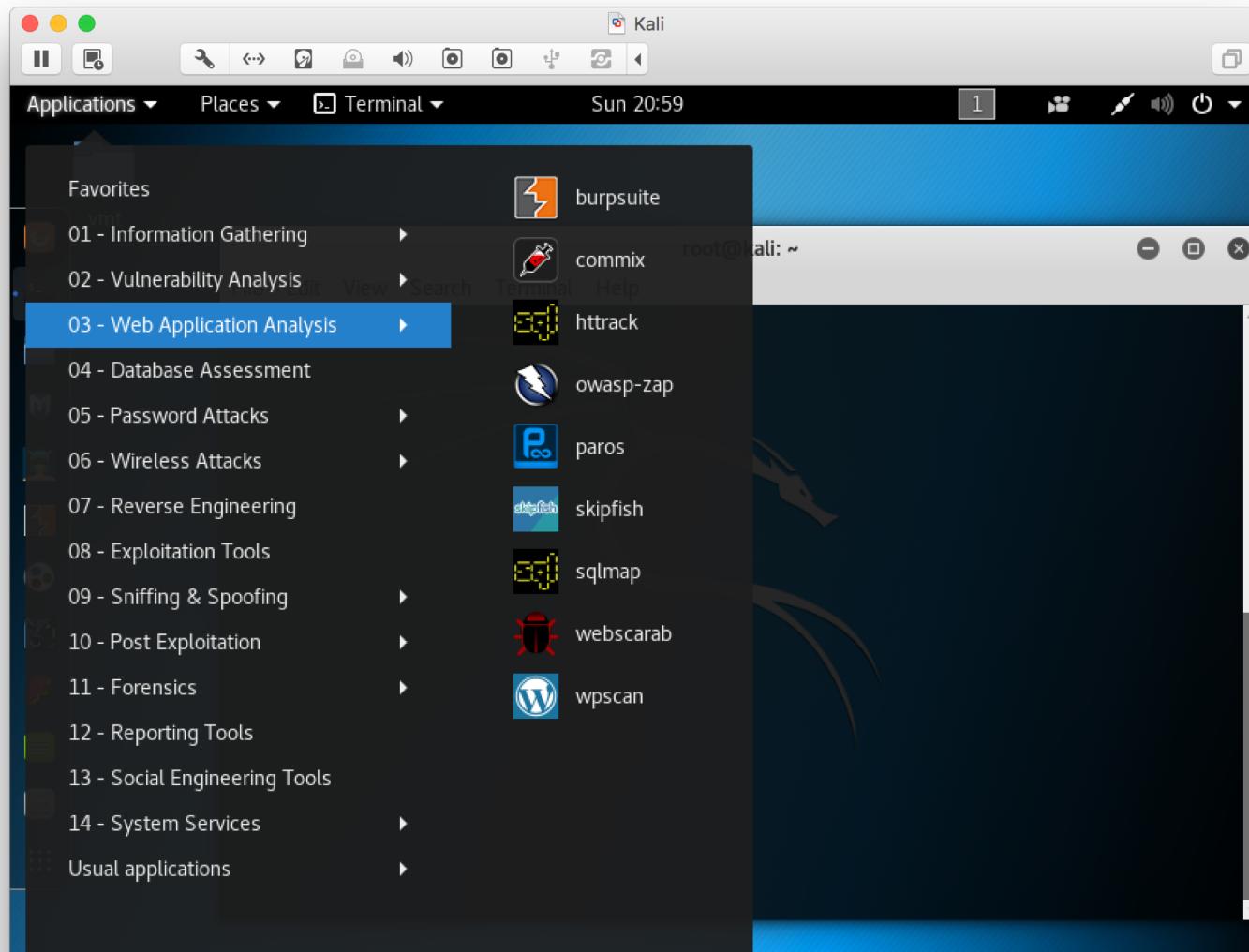
Bugs in Google-developed apps and extensions (published in [Google Play](#), in [iTunes](#), or in the [Chrome Web Store](#)), as well as some of our hardware devices ([OnHub](#) and [Nest](#)) will also qualify. See our [Android Rewards](#) and [Chrome Rewards](#) for other services and devices that are also in scope.

On the flip side, the program has two important exclusions to keep in mind:

KALI LINUX

Advanced penetration testing and security auditing Linux distribution

- 600+ built-in penetration testing tools
- Including web app attack tools
- Free & open source
- Secure environment



Stages of an attack

- Reconnaissance
 - Accessing public information (whois, DNS, web searches, social media posts), “Google hacking”, Maltego, social engineering, ...
- Scanning
 - Port scanning (nmap), software version-mapping, automated vulnerability scanning tools, specialist search engines (shodan.io), ...
- Exploit systems
 - Authentication grinding (password cracking), passive and active sniffing, buffer overflows, session hijacking, DNS cache poisoning, denial of service, web application attacks, ...

Stages of an attack (continued)

- Keeping access
 - Having gone to the trouble of breaking in, the attacker wants to get back in easily, and facilitates this by installing back doors and/or remote control software
 - Trojan horses, netcat listeners, rootkits
- Covering tracks
 - File hiding, log editing, use of covert channels (steganography)

General multi-purpose web app attack tools

- Typical features
 - Proxy for traffic interception/modification
 - Vulnerability scanning
 - Site crawling
 - Fuzzing
- Popular tools
 - Burp suite
 - Commerical and free/community edition
 - OWASP Zed Attack Proxy (ZAP)
 - Free and open source
 - W3AF
 - Free and open source

Tools for specific purposes

- Sniffing
 - Wireshark
- Port scanning
 - Nmap, netcat
- Proxies for intercepting/modifying traffic
 - WebScarab, Paros proxy, ...
- Tools for specific attack types
 - Commix – command injection
 - sqlmap – SQL injection
 - Skipfish – maps site by crawling links and dictionary-based guessing
 - setoolkit – includes site cloning for phishing attacks

Deliberately vulnerable web apps

- Good for practicing ethical hacking
- Examples
 - OWASP WebGoat
 - DVWA
(Damn Vulnerable Web App)
 - + many others...

The screenshot shows a Mozilla Firefox browser window with the title "Stored XSS Attacks - Mozilla Firefox". The address bar shows the URL <http://192.168.0.5/WebGoat/attack?Screen=508&menu=900&Restart=50>. The page itself is titled "Stored XSS Attacks" and features a red background with a cracked texture. At the top right, there's a "Logout" link. Below the title, there's a navigation bar with links like "Hints", "Show Params", "Show Cookies", "Lesson Plan", "Show Java", and "Solution". On the left side, there's a sidebar with a menu including "Introduction", "General", "Access Control Flaws", "AJAX Security", "Authentication Flaws", "Buffer Overflows", "Code Quality", "Concurrency", "Cross-Site Scripting (XSS)", "Phishing with XSS", "LAB: Cross Site Scripting", and several "Stage" links from 1 to 6. The main content area contains a text input field labeled "Title:" with the value "`art('Ha Ha Ha');`</script>" and a larger "Message:" text area below it. A "Submit" button is located at the bottom of the message area. At the very bottom, there's a "Message List" section containing three identical script tags: `<script language="javascript" type="text/javascript">alert('Ha Ha Ha');</script>`.

Web Application Firewall

- Protects web applications
- Applies a set of rules to incoming HTTP requests and outgoing responses & logs/monitors/filters accordingly
 - Typically looks for SQLi, XSS,, known vulnerabilities, ...

AWS
WAF
config

The screenshot shows the AWS WAF & Shield console with the URL <https://console.aws.amazon.com/waf/home?region=eu-west-1#/wizard/>. The page title is "Set up a web access control list (web ACL)". On the left, a sidebar lists steps: Concepts overview, Step 1: Name web ACL, **Step 2: Create conditions**, Step 3: Create rules, and Step 4: Review and create. The main content area is titled "Create conditions". It explains that conditions specify filters for requests forwarded to AWS resources like CloudFront. Under "Cross-site scripting match conditions", it says there are no conditions and provides a link to "Create XSS match condition". Below this, under "Geo match conditions", it says there are no conditions and provides a link to "Create condition". To the right, a "Concepts overview" sidebar shows examples of rules: Rule 1 (Bad User-Agents) using IP and String match conditions, and Rule 2 (Detect SQLi) using SQL injection match conditions.

Set up a web access control list (web ACL)

Concepts overview

Step 1: Name web ACL

Step 2: Create conditions

Step 3: Create rules

Step 4: Review and create

Create conditions

Conditions specify the filters that you want to use to allow or block requests that are forwarded to AWS resources such as Amazon CloudFront distributions.

Cross-site scripting match conditions

Name [Create condition](#)

You don't have any cross-site scripting match conditions. Choose [Create XSS match condition](#) to get started.

A cross-site scripting match condition specifies the parts of a web request (such as a User-Agent header) that you want AWS WAF to inspect for cross-site scripting threats. [Learn more](#)

Geo match conditions

Name [Create condition](#)

You don't have any geo match conditions. Choose [Create condition](#) to get started.

A geo match condition lets you allow, block, or count web requests based on the geographic origin of the request. [Learn more](#)

Concepts overview

Web ACL example if requests match

Rule 1, Bad User-Agents, then block

IP match condition Suspicious IPs

and

String match condition Bad bots

or if requests match

Rule 2, Detect SQLi, then block

SQL injection match condition SQL i checks

Common attack vector: Malformed input

- Inputting data (e.g. in a web form) to cause a program to behave unusually.
- Often takes advantage of known vulnerability

The screenshot shows a web browser window titled "Student results". The address bar contains the URL "http://sqlil.witdemo.net/results.html". The main content area displays the heading "Insecure Institute of Technology" and "Student Results System". Below this, a message reads "Please enter your student number and press \"Submit\"". A red oval highlights the input field, which contains the value "junk' or '1'='1". To the right of the input field is a "submit" button.

Insecure Institute of Technology

Student Results System

Please enter your student number and press "Submit"

junk' or '1'='1

submit

Common attack vector: Malformed input

- Common types of malformed input attack:
 - SQL injection
 - Buffer overflow
 - Cross site scripting (XSS)
 - XML External Entity (XXE) attack

Background: HTTP request types

HTTP Request Methods

- HTTP is a fairly simple protocol with a small number of methods that define actions to be performed on a specified resource (such as a web page), identified by a URL.

HTTP Request Methods

Method	Purpose
GET	Requests data from a specified resource
POST	Submits data to be processed to a specified resource
HEAD	Same as GET but returns only HTTP headers and no document body
PUT	Uploads a representation of the specified URI
DELETE	Deletes the specified resource
OPTIONS	Returns the HTTP methods that the server supports
CONNECT	Converts the request connection to a transparent TCP/IP tunnel

HTTP GET v POST

- **GET**
 - Designed to retrieve resources (often files) from a server
 - However URI syntax allows a lot of flexibility, so it's easy to use GET to send data as part of URI ("URL encoding")
- **POST**
 - Designed to send data to a web server.
 - Data provided in the body of the message rather than in the URI
 - More flexible and **more secure**
 - URLs usually cached by browsers, and often bookmarked, shared etc
 - URLs usually logged by proxies and web servers

HTTP GET v POST

- **GET example**

```
GET /path/login?username=jbloggs&password=topsecret  
HTTP/1.1  
Host: www.site.com  
User-Agent: Mozilla/5.0 ...
```

- **POST example**

```
POST /path/login HTTP/1.1  
Host: www.site.com  
User-Agent: Mozilla/5.0 ...
```

Header

```
username=jbloggs&password=topsecret
```

Body