CS 578: CYBER-SECURITY

PART IV: WEB SECURITY

Sanghyun Hong
sanghyun.hong@oregonstate.edu





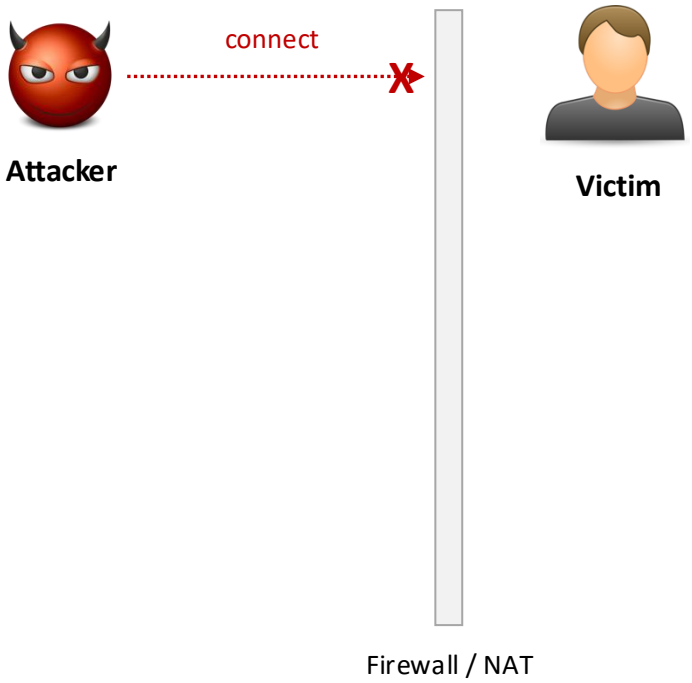
RECAP – WHERE WE ARE

- Part I – Network/Internet security
  - The attacker is outside your machine
- Part II – Computer systems security
  - The attacker is inside your machine
  - Memory safety: how the attacker gets in
  - OS security: decide what the attacker may do (access control)
- Part III – Isolation and (software-induced) breaks
  - The attacker is inside your program or is the OS itself
  - The attacker exploits your hardware and breaks integrity and confidentiality
- Part IV – Software/Web security
  - The attacker turns bugs in your software into exploits
  - The attacker uses the web to *deliver* exploits and malware (today!)

MALWARE DELIVERY – HOW EXPLOITS REACH YOU

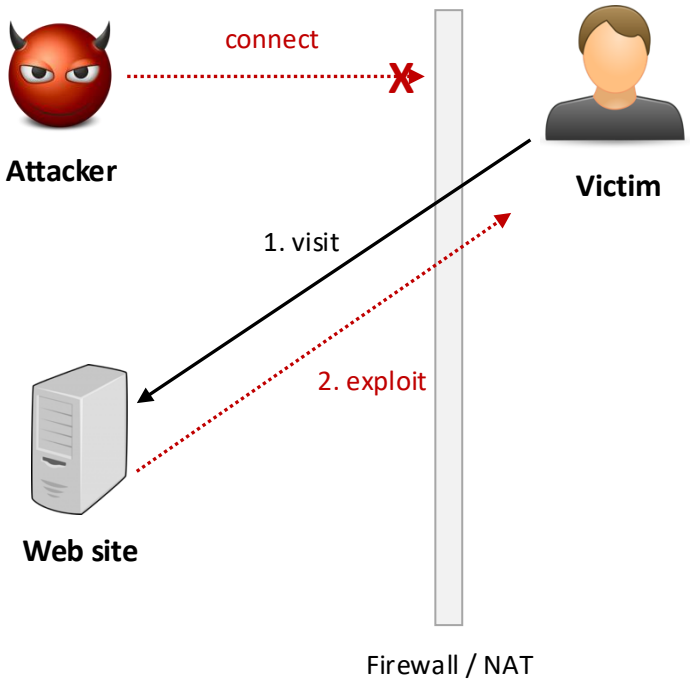
How does an exploit reach the victim?

- Last time: an exploit for one program
  - How does it reach many victims?
- Push-based: the attacker connects to you
  - Scan the Internet for vulnerable services
  - ex) worms
  - Firewalls and NATs block incoming connections



How does an exploit reach the victim? – cont'd

- Last time: an exploit for one program
  - How does it reach many victims?
- Push-based: the attacker connects to you
  - Scan the Internet for vulnerable services
  - ex) worms
  - Firewalls and NATs block incoming connections
- Pull-based: you connect to the attacker
  - You visit a web page
  - The page sends the exploit to your browser
  - Outgoing web traffic is allowed



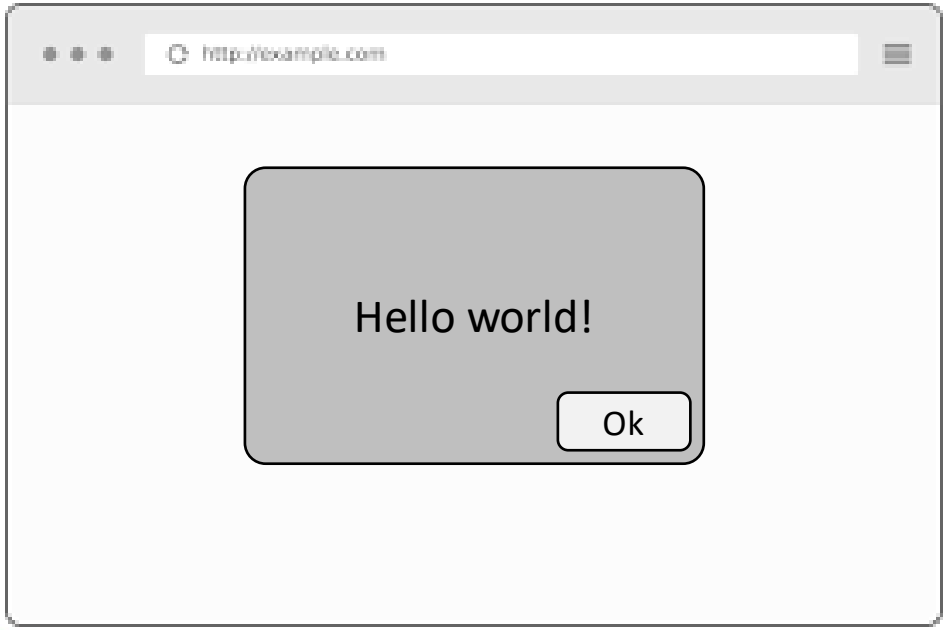
WEB BASICS – WHAT RUNS IN YOUR BROWSER

JAVASCRIPT

- JavaScript
  - A programming language for running code in the web browser
  - Runs on the client-side
    - The server sends code as part of the HTTP response
    - The code runs in the browser, not in the web-server
  - Used to manipulate web pages (HTML and CSS)
    - Makes modern websites interactive
    - JavaScript can be directly embedded in HTML with <script> tags
  - Supported by all modern web browsers
    - Most modern webpages involve JavaScript

JAVASCRIPT – CONT'D

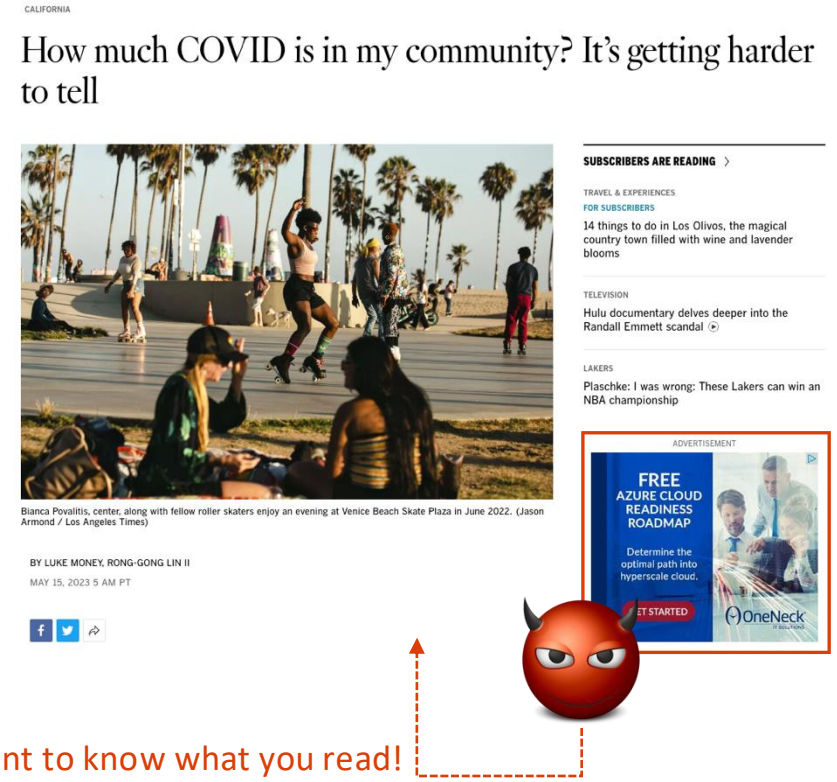
- JavaScript example
  - Create a pop-up message
  - HTML: <script>alert(”Hello world!")</script>



If the browser loads the HTML, it will run the embedded JavaScript and create a pop-up window.

SECURITY RISKS ON THE WEB

- Risk I:
  - Malicious websites should not be able to tamper with our information or interactions on other websites
  - Example:
    - We visit “latimes.com”
    - Malicious folks do “ad” on this site
    - The “ad” runs some JavaScripts and extracts our information from “latimes” (e.g., which type of articles we read)



SAME-ORIGIN POLICY

- Same-origin policy
  - A rule that prevents one website from tampering with other *unrelated* websites
    - Enforced by the web browser
    - Prevents a malicious website from running scripts on other websites
    - Pages from the same site don't need to be isolated to each other



SAME-ORIGIN POLICY – CONT'D

- Same-origin policy
  - Two websites have the same origin *if and only if*
  - The protocol, domain, and port of the URL all match exactly

Domain I	Domain II	Same-origin?
http://cs.org	http://www.cs.org	No, domain mismatch
http://cs.org	https://cs.org	No, protocol mismatch
http://cs.org:80	http://cs.org:8080	No, port mismatch
https://cs.org/photo.png	https://cs.org/data/my.htm	Yes

SAME-ORIGIN POLICY – CONT'D

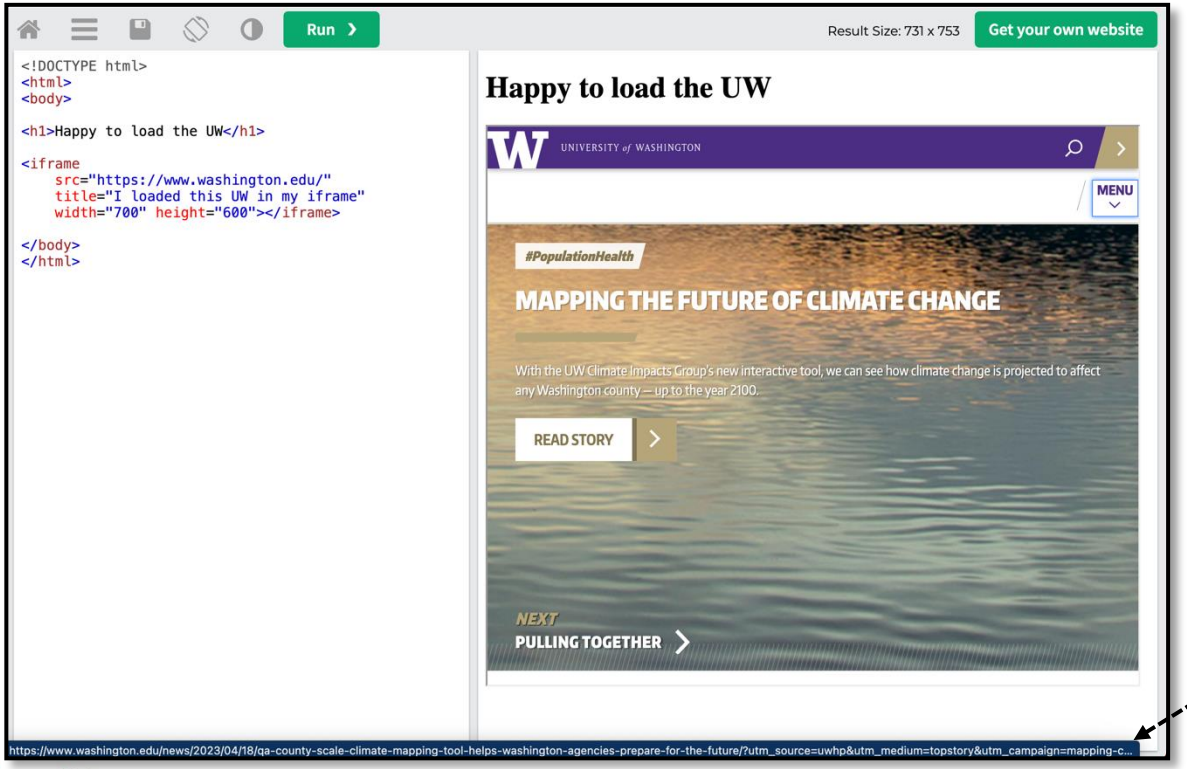
- Same-origin policy
  - Two websites have the same origin *if and only if*
  - The protocol, domain, and port of the URL all match exactly
  - Example scenario:
    - **cs.org** embeds **google.com**
    - The inner frame cannot interact with the outer frame
    - The outer frame cannot interact with the inner one

SAME-ORIGIN POLICY – EXCEPTION

- Same-origin policy
  - Two websites have the same origin *if and only if*
  - The protocol, domain, and port of the URL all match exactly
  - Example scenario:
    - **cs.org** embeds **google.com**
    - The inner frame cannot interact with the outer frame
    - The outer frame cannot interact with the inner one
  - Exception I:
    - JavaScript runs with the origin of the page that loads it
    - ex. **cs.org** fetches JavaScript from **google.com**:
      - The JavaScript has the origin of **cs.org**
      - **cs.org** has “copy-pasted” JavaScript onto its webpage

IFRAME – A PAGE INSIDE A PAGE

- An iframe loads another website inside your page

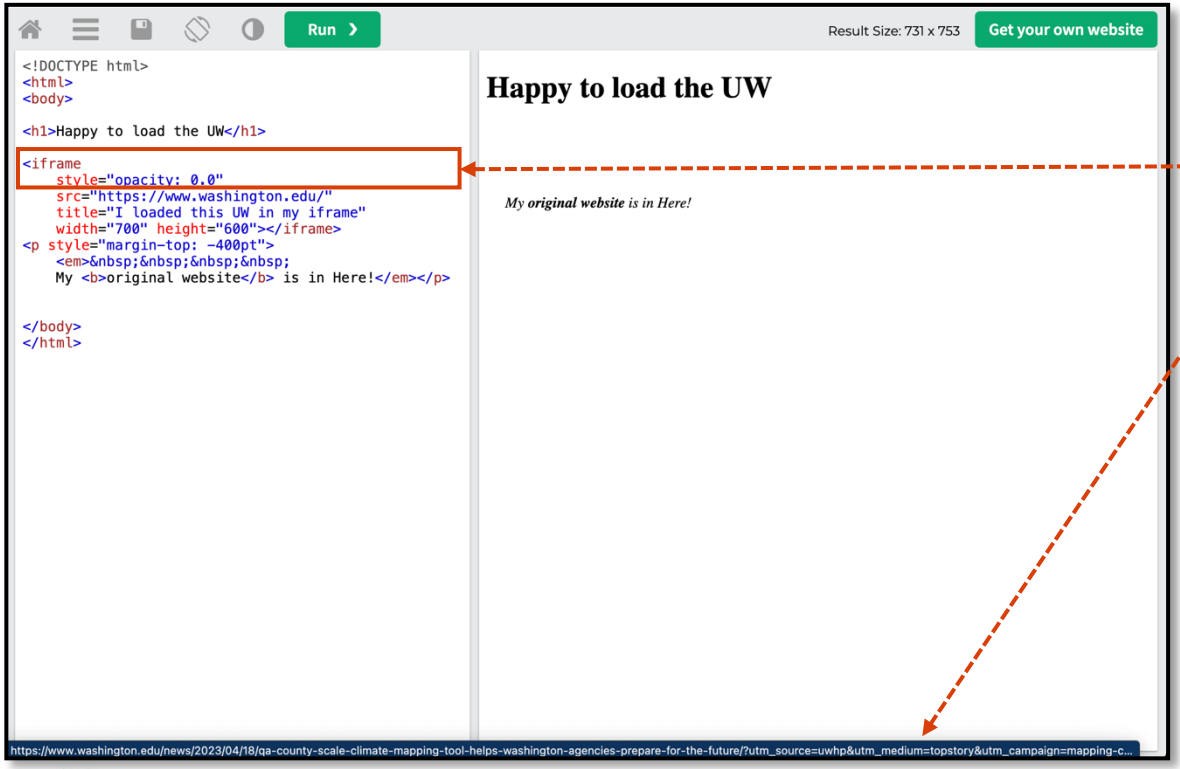


Note: any links on the website in the iframe are "washington.edu"

Users can click it, but we cannot make the website automatically click this link due to the same origin policy

IFRAME – CONT'D

- Make the iframe invisible



Now the iframe is invisible

But the browser still loads and runs everything inside it

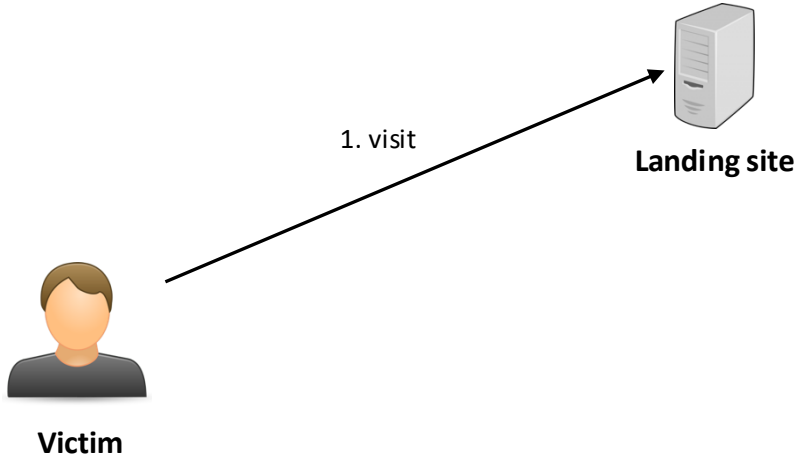
IFRAME AND THE SAME-ORIGIN POLICY

- Same-origin policy
  - The iframe cannot read your page
  - Your page cannot read the iframe
- But the iframe still runs code in your browser
  - The browser is a program with bugs
  - Code in the iframe can exploit them
- Drive-by download
  - You visit a page, and malware is installed and run
  - No click, no download dialog

ALL YOUR IFRAMES – DRIVE-BY DOWNLOADS ON THE WEB

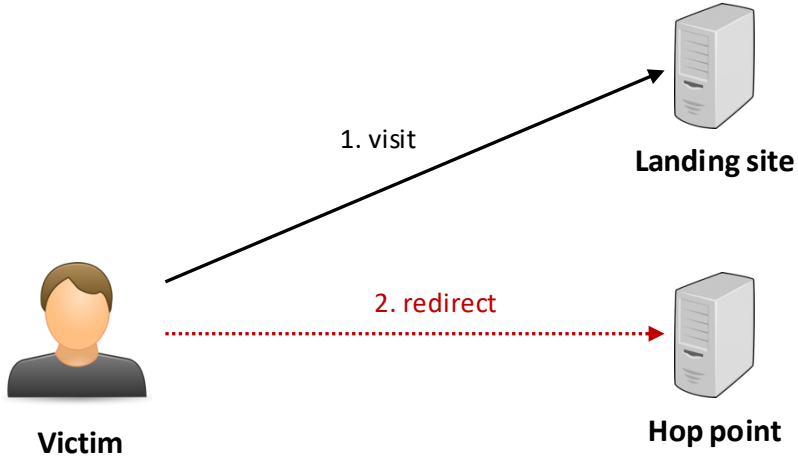
ALL YOUR IFRAMES – DRIVE-BY DOWNLOAD

- Landing site
  - A normal site with injected content
  - ex) a hidden iframe
  - 1. The victim visits it



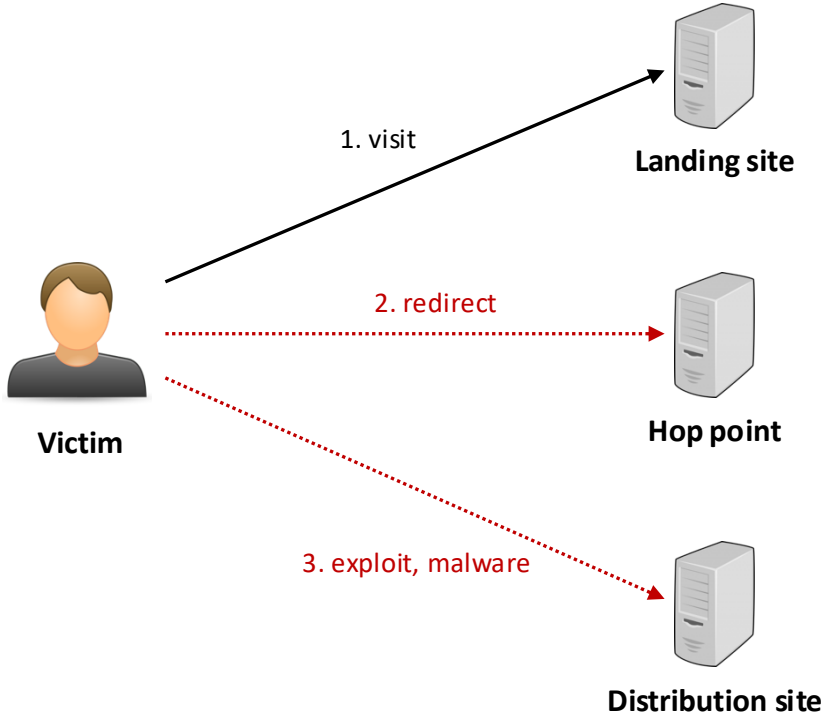
ALL YOUR IFRAMES – DRIVE-BY DOWNLOAD – CONT'D

- Landing site
  - A normal site with injected content
  - ex) a hidden iframe
  - 1. The victim visits it
- Hop points
  - 2. The browser is redirected, often many times
  - Harder to trace back



ALL YOUR IFRAMES – DRIVE-BY DOWNLOAD – CONT'D

- Landing site
  - A normal site with injected content
  - ex) a hidden iframe
  - 1. The victim visits it
- Hop points
  - 2. The browser is redirected, often many times
  - Harder to trace back
- Distribution site
  - 3. Sends the exploit, then the malware
  - Installed and run, no click



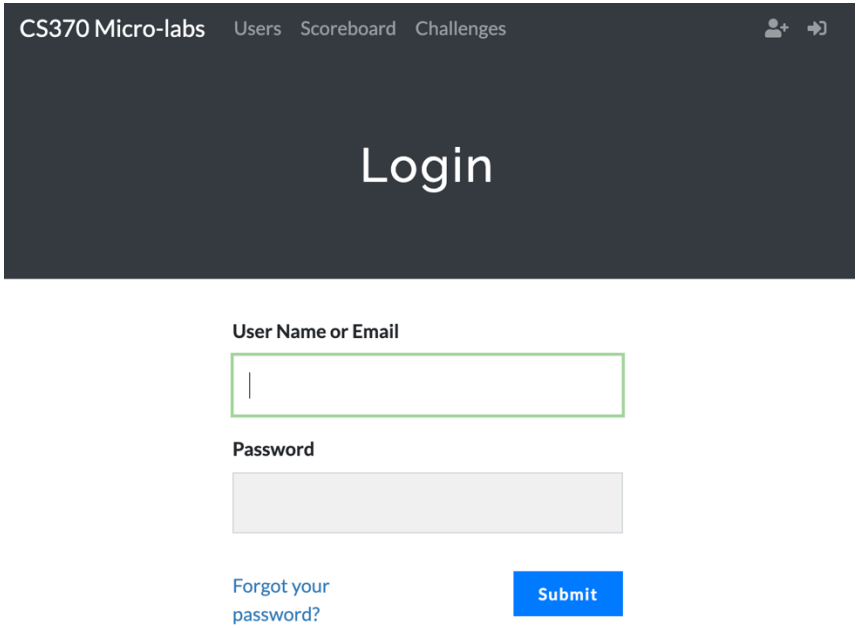
ALL YOUR IFRAMES – CONTROLLING A BENIGN SITE

- How does the attacker put content into a normal site?
  - Compromise the web server
    - ex) old PHP forum software, SQL injection
  - Post user content that allows HTML
    - ex) forums, blogs, comments
  - Show an ad
    - The site loads ads from other companies
- No need to own a popular site
  - Borrow its visitors

PRELIM – SQL INJECTION

- Exploit the system's weakness
  - SELECT (username, password) FROM users WHERE username = 'neuronoverflow' and password = SHA256(secret + "my-super-secure-password!@#$11")

- SQL injection
  - We supply 'or 'a'='a as a password
  - SELECT (username, password) FROM users WHERE username = 'neuronoverflow' and password = '' or 'a' ='a'
  - THIS IS ALWAYS TRUE!!!

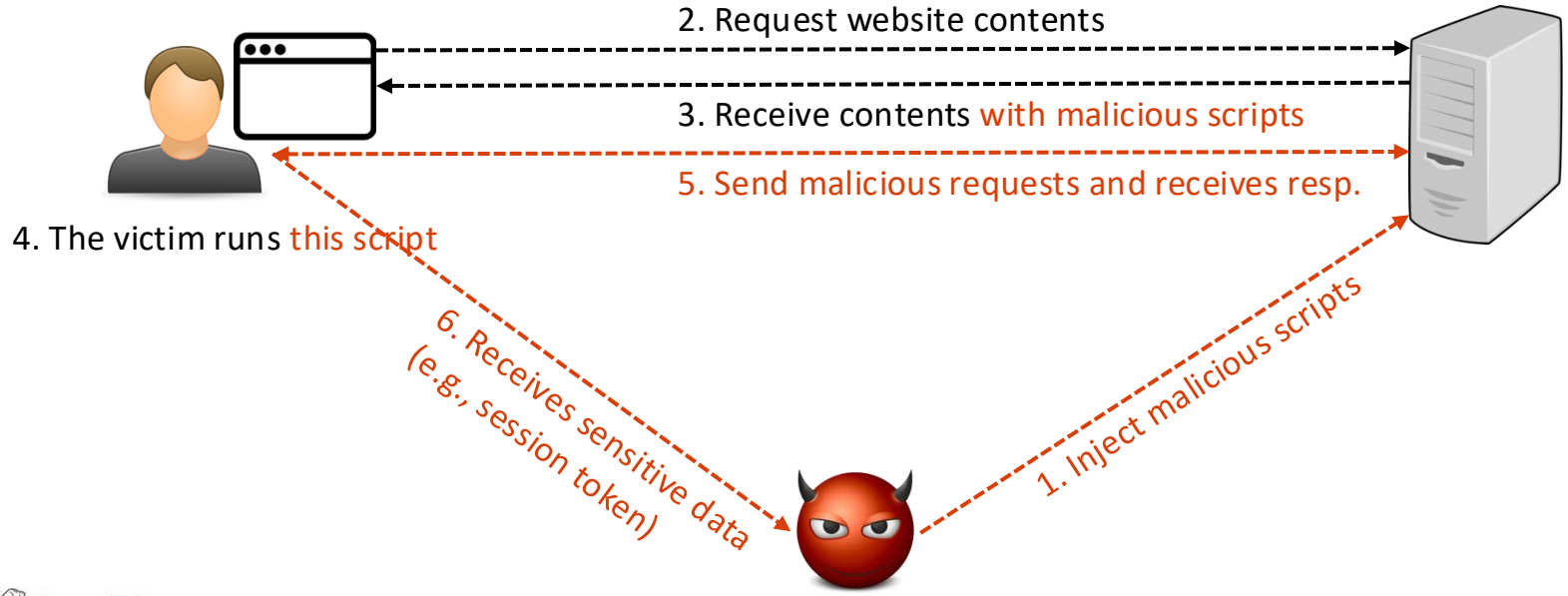


PRELIM – STORED XSS

- Stored XSS (persistent XSS)
  - The attacker's JavaScript is stored on the legitimate server
  - Example: Facebook pages
    - Anyone can load a Facebook page with content provided by users
    - An adversary puts some JavaScript on their Facebook page
    - Anyone who loads the attacker's page will run JavaScript (with the origin of Facebook)
  - **Note:** stored XSS requires the victim to load the page with injected JavaScript

PRELIM – STORED XSS

- Stored XSS illustration
  - The attacker's JavaScript is stored on the legitimate server
  - **Note:** stored XSS requires the victim to load the page with injected JavaScript

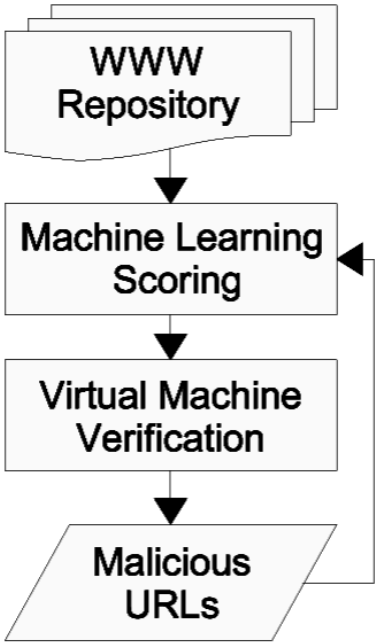


ALL YOUR IFRAMES – RESEARCH QUESTIONS

- Goal
  - Measure drive-by downloads on the whole web
- Questions
  - How common are they?
  - Does avoiding shady sites help?
  - How does the malicious content get in?
  - What does the attacker's infrastructure look like?
  - What happens after infection?
- Challenge
  - Billions of URLs; each check takes minutes in a VM

ALL YOUR IFRAMES – FINDING MALICIOUS PAGES

- Stage 1: machine-learning score
  - About one billion pages a day
  - ex) out-of-place iframes, obfuscated JavaScript, iframes to known distribution sites
  - Keep about one million URLs a day
- Stage 2: verify in a virtual machine
  - Slow, but tells for sure
- Found URLs go back to Stage 1

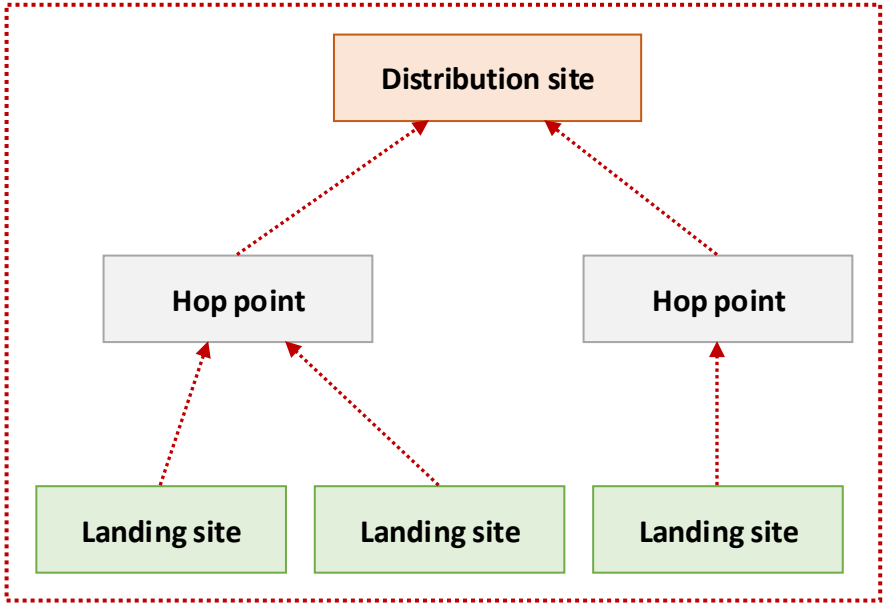


ALL YOUR IFRAMES – VERIFICATION

- A honeypot that browses
  - Windows VM with an unpatched Internet Explorer
  - Open the URL, wait about two minutes
- What it watches
  - New processes
  - File system and registry changes
  - Anti-virus scans of the HTTP responses
- Decision
  - Malicious: state changes and at least one anti-virus hit
  - Suspicious: state changes, no anti-virus hit

ALL YOUR IFRAMES – DELIVERY TREES

- Which site sent the browser where?
  - Follow the Referer header between requests
  - If missing: find the URL in the fetched HTML and JavaScript
- Malware delivery tree
  - Leaves: landing sites
  - Root: the distribution site
- Distribution network
  - All trees that end at one distribution site



One distribution network

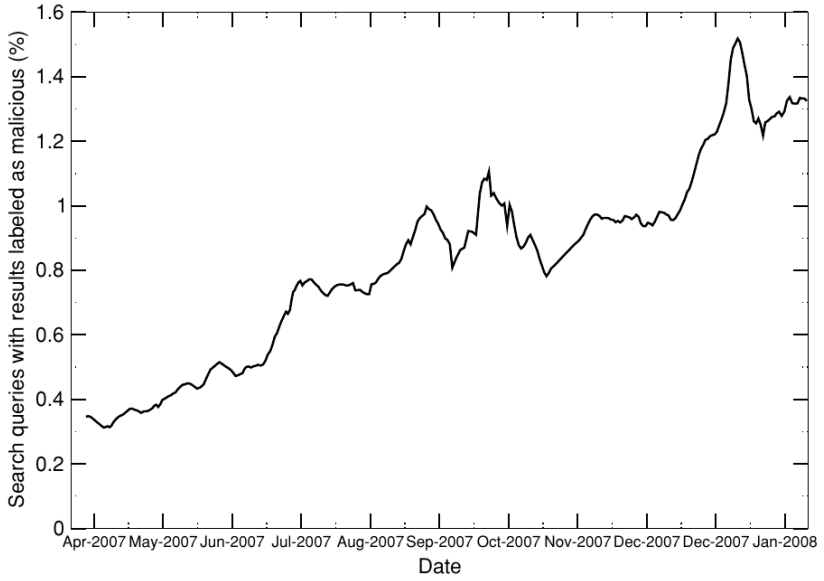
ALL YOUR IFRAMES – HOW COMMON?

- Ten months of data (Jan – Oct 2007)

	Count
URLs checked in a VM	66,534,330
Malicious landing URLs	3,417,590
Suspicious landing URLs	3,385,889
Malicious landing sites	181,699
Distribution sites	9,340

ALL YOUR IFRAMES – HOW OFTEN DO USERS SEE THEM?

- Google search queries
  - 1.3% return at least one malicious URL
  - Rising over 2007
- Top 1M URLs in search results
  - About 6,000 belong to malicious sites
  - Spread across all ranks
  - The most popular one: rank 1,588

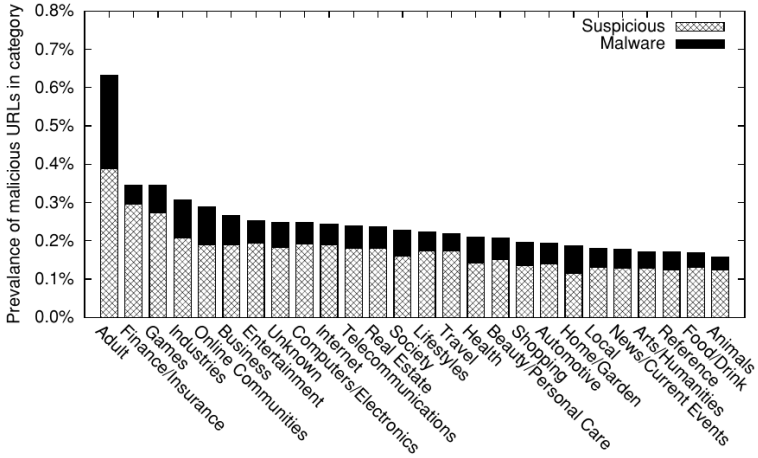


ALL YOUR IFRAMES – WHERE ARE THEY HOSTED?

- Distribution sites
  - China 67%, US 15%, Russia 4%
- Landing sites
  - China 64.6%, US 15.6%, Russia 5.6%
- Mostly local
  - 96% of landing sites in China point to distribution sites in China
- Why?
  - Old, unpatched web servers

ALL YOUR IFRAMES – DOES AVOIDING SHADY SITES HELP?

- Sites grouped by category
  - 7.2M URLs sampled
- Adult sites: the highest
  - About 0.6% of URLs
- But every category has them
  - News, travel, health, …
  - Safe browsing alone does not protect yc



ALL YOUR IFRAMES – HOW DOES THE CONTENT GET IN?

- Web server software of landing sites
  - Read from the Server and X-Powered-By headers

Software	Count	Unknown
Apache	55,088	26.5%
PHP	27,873	8.5%
Microsoft IIS	113,905	n/a

Old: older than the latest version in Jan 2007, with known vulnerabilities

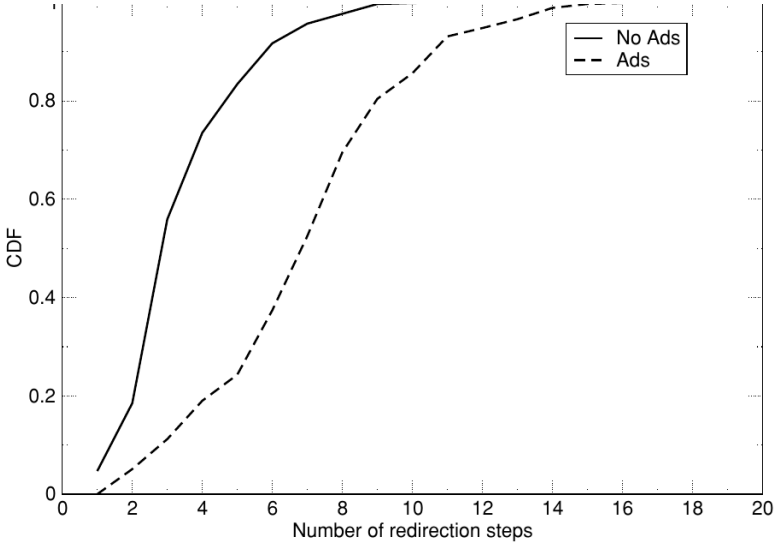
ALL YOUR IFRAMES – DRIVE-BY DOWNLOADS VIA ADS

- Ads are third-party content
  - A page is only as secure as its weakest part
- Ad syndication
  - An ad network resells its space to another network
  - That network can resell it again



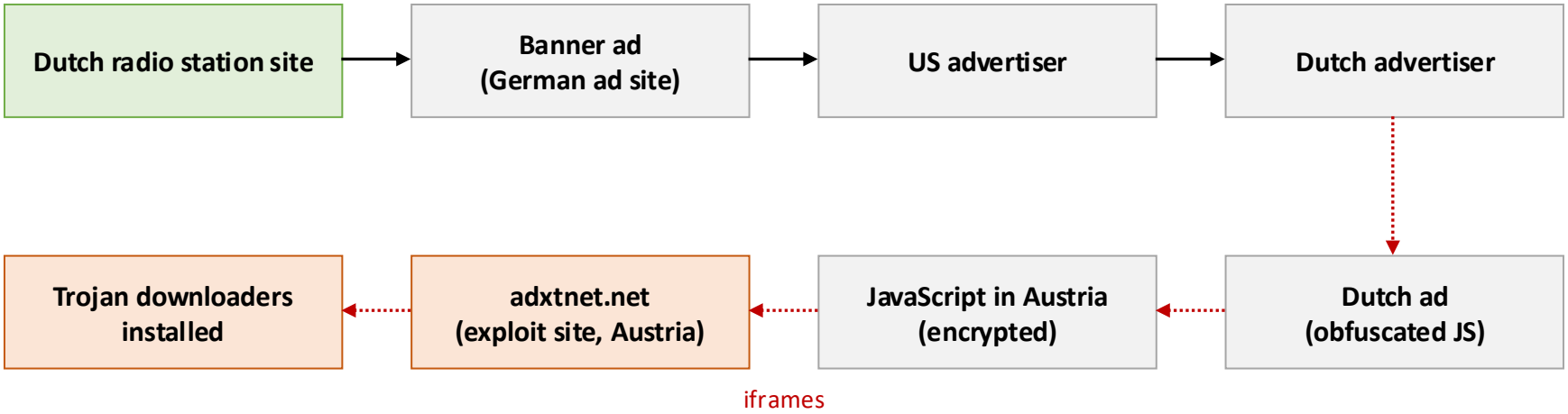
ALL YOUR IFRAMES – DRIVE-BY DOWNLOADS VIA ADS – CONT'D

- Landing sites with malicious ads
  - 2% of landing sites
  - 12% when weighted by search results
- Long chains
  - 75% used multiple levels of syndication
  - Half needed more than 6 redirects
- Short spikes
  - Appear on thousands of sites at once
  - Removed at once



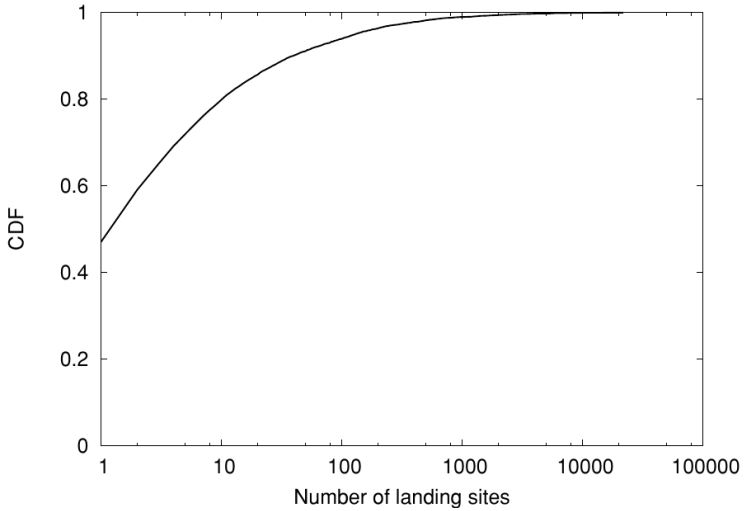
ALL YOUR IFRAMES – AN AD CHAIN

- Example from the paper
  - Each redirect gives another party control of the page



ALL YOUR IFRAMES – DISTRIBUTION INFRASTRUCTURE

- Landing sites per distribution site
  - About 45% use only one
  - Some have over 21,000
- Few networks
  - 70% in a few IP ranges
  - 95% in 210 ASes
- Shared
  - 80% of networks share a landing site



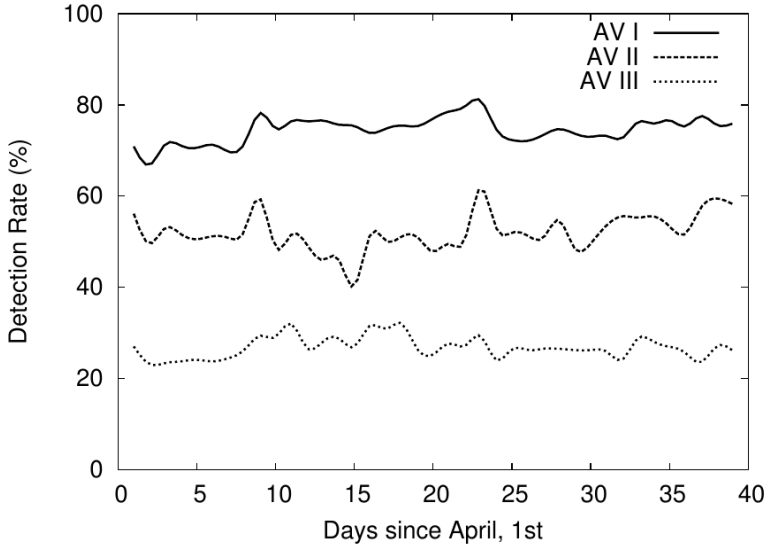
ALL YOUR IFRAMES – AFTER INFECTION

- Downloads
  - 8 executables per URL on average, up to 60
- Registry changes on 57.5% of landing pages

Change	What the malware does	Landing pages
Startup	Runs again after reboot	51.27%
Security	Turns off the firewall or updates	36.18%
Preferences	Changes the home page or search engine	23.5%
BHO	Adds its own code to the browser	6.99%

ALL YOUR IFRAMES – CAN ANTI-VIRUS CATCH IT?

- Setup
  - Suspicious binaries from April 2007
  - Scanned daily with the latest signatures
- Best engine: about 70% on average
- Why so low?
  - Pull-based: AV sensors rarely visit these pages
  - Binaries change often



ALL YOUR IFRAMES – COUNTERMEASURES

- What we can do?

Attacker needs	Countermeasure	Problem
Unpatched browser	Automatic updates	Users delay; zero-days
Control of a server	Patch server software	38% of Apache was old
HTML in user content	Filter HTML (as for XSS)	Filters miss cases
Space in ads	Ad networks check ads	Long resale chains
Visitors to the page	Warn before visiting	Must find the page first
Malware on disk	Anti-virus	Best engine about 70%

DARK PATTERNS AT SCALE IN-CLASS PRESENTATION BY HAO-RONG AND LIYUAN

RECAP – WHERE WE ARE

- Part I – Network/Internet security
  - The attacker is outside your machine
- Part II – Computer systems security
  - The attacker is inside your machine
  - Memory safety: how the attacker gets in
  - OS security: decide what the attacker may do (access control)
- Part III – Isolation and (software-induced) breaks
  - The attacker is inside your program or is the OS itself
  - The attacker exploits your hardware and breaks integrity and confidentiality
- Part IV – Software/Web security
  - The attacker turns bugs in your software into exploits
  - The attacker uses the web to deliver exploits and malware

Thank You!

Sanghyun Hong

https://true-lab.ai/courses/Sec-Grad/current



