CSci 4271W
Development of Secure Software Systems
Day 15: Fuzzing and web security part 1

Stephen McCamant
University of Minnesota, Computer Science & Engineering

Outline

(System) virtual machines

- Presents hardware-like interface to an untrusted kernel
- Strong isolation, full administrative complexity
- I/O interface looks like a network, etc.

Virtual machine designs

- (Type 1) hypervisor: 'superkernel' underneath VMs
- Hosted: regular OS underneath VMs
- Paravirtualizaion: modify kernels in VMs for ease of virtualization

Virtual machine technologies

- Hardware based: fastest, now common
- Partial translation: e.g., original VMware
- Full emulation: e.g. QEMU proper
  - Slowest, but can be a different CPU architecture

Modern example: Chrom(ium)

- Separates "browser kernel" from less-trusted "rendering engine"
  - Pragmatic, keeps high-risk components together
- Experimented with various Windows and Linux sandboxing techniques
- Blocked 70% of historic vulnerabilities, not all new ones
- `http://seclab.stanford.edu/websec/chromium/`

Outline

Testing and security

- "Testing shows the presence, not the absence of bugs" – Dijkstra
- Easy versions of some bugs can be found by targeted tests:
  - Buffer overflows: long strings
  - Integer overflows: large numbers
  - Format string vulnerabilities: `%x`

Random or fuzz testing

- Random testing can also sometimes reveal bugs
- Original 'fuzz' (Miller): `program </dev/urandom`
- Even this was surprisingly effective

Mutational fuzzing

- Instead of totally random inputs, make small random changes to normal inputs
- Changes are called *mutations*
- Benign starting inputs are called *seeds*
- Good seeds help in exercising interesting/deep behavior

Grammar-based fuzzing

- Observation: it helps to know what correct inputs look like
- Grammar specifies legal patterns, run backwards with random choices to generate
- Generated inputs can again be basis for mutation
- Most commonly used for standard input formats
  - Network protocols, JavaScript, etc.

What if you don't have a grammar?

- Input format may be unknown, or buggy and limited
- Writing a grammar may be too much manual work
- Can the structure of interesting inputs be figured out automatically?

Coverage-driven fuzzing

- Instrument code to record what code is executed
- An input is interesting if it executes code that was not executed before
- Only interesting inputs are used as basis for future mutation

AFL

- Best known open-source tool, pioneered coverage-driven fuzzing
- American Fuzzy Lop, a breed of rabbits
- Stores coverage information in a compact hash table
- Compiler-based or binary-level instrumentation
- Has a number of other optimizations

Outline

More choices for isolation, cont'd

Testing and fuzzing

Announcements intermission

The web from a security perspective

Cross-site scripting

More cross-site risks

Wheeler reading questions

- Due (on Canvas) Thursday night
- Note no late submissions, so do them on time

Midterm 1 grade statistics

```
<=5 | *
  6 | 6799
  7 | 677778
  8 | 00111223444555888
  9 | 2222224566666666
```

- Mean: 82.9, Median: 84

Outline

More choices for isolation, cont'd

Testing and fuzzing

Announcements intermission

The web from a security perspective

Cross-site scripting

More cross-site risks

Once upon a time: the static web

- HTTP: stateless file download protocol
  - TCP, usually using port 80
- HTML: markup language for text with formatting and links
- All pages public, so no need for authentication or encryption

Web applications

- The modern web depends heavily on active software
- Static pages have ads, paywalls, or "Edit" buttons
- Many web sites are primarily forms or storefronts
- Web hosted versions of desktop apps like word processing

Server programs

- Could be anything that outputs HTML
- In practice, heavy use of databases and frameworks
- Wide variety of commercial, open-source, and custom-written
- Flexible scripting languages for ease of development
  - PHP, Ruby, Perl, etc.

Client-side programming

- Java: nice language, mostly moved to other uses
- ActiveX: Windows-only binaries, no sandboxing
  - Glad to see it on the way out
- Flash and Silverlight: last important use was DRM-ed video
- Core language: JavaScript

JavaScript and the DOM

- JavaScript (JS) is a dynamically-typed prototype-OO language
  - No real similarity with Java
- Document Object Model (DOM): lets JS interact with pages and the browser
- Extensive security checks for untrusted-code model

Same-origin policy

- *Origin* is a tuple (scheme, host, port)
  - E.g., (http, www.umn.edu, 80)
- Basic JS rule: interaction is allowed only with the same origin
- Different sites are (mostly) isolated applications

GET, POST, and cookies

- `GET` request loads a URL, may have parameters delimited with `?`, `&`, `=`
  - Standard: should not have side-effects
- `POST` request originally for forms
  - Can be larger, more hidden, have side-effects
- *Cookie*: small token chosen by server, sent back on subsequent requests to same domain

User and attack models

- "Web attacker" owns their own site (`www.attacker.com`)
  - And users sometimes visit it
  - Realistic reasons: ads, SEO
- "Network attacker" can view and sniff unencrypted data
  - Unprotected coffee shop WiFi

Outline

More choices for isolation, cont'd

Testing and fuzzing

Announcements intermission

The web from a security perspective

Cross-site scripting

More cross-site risks

XSS: HTML/JS injection

- Note: CSS is "Cascading Style Sheets"
- Another use of injection template
- Attacker supplies HTML containing JavaScript (or occasionally CSS)
- OWASP's most prevalent weakness
  - A category unto itself
  - Easy to commit in any dynamic page construction

Why XSS is bad (and named that)

- `attacker.com` can send you evil JS directly
- But XSS allows access to `bank.com` data
- Violates same-origin policy
- Not all attacks actually involve multiple sites

Reflected XSS

- Injected data used immediately in producing a page
- Commonly supplied as query/form parameters
- Classic attack is link from evil site to victim site

Persistent XSS

- Injected data used to produce page later
- For instance, might be stored in database
- Can be used by one site user to attack another user
  - E.g., to gain administrator privilege

DOM-based XSS

- Injection occurs in client-side page construction
- Flaw at least partially in code running on client
- Many attacks involve mashups and inter-site communication

No string-free solution

- For server-side XSS, no way to avoid string concatenation
- Web page will be sent as text in the end
  - Research topic: ways to change this?
- XSS especially hard kind of injection

Danger: complex language embedding

- JS and CSS are complex languages in their own right
- Can appear in various places with HTML
  - But totally different parsing rules
- Example: `"..."` used for HTML attributes and JS strings
  - What happens when attribute contains JS?

Danger: forgiving parsers

- History: handwritten HTML, browser competition
- Many syntax mistakes given "likely" interpretations
- Handling of incorrect syntax was not standardized

Sanitization: plain text only

- Easiest case: no tags intended, insert at document text level
- Escape HTML special characters with *entities* like `<` for <
- OWASP recommendation: `& < > " ' /`

Sanitization: context matters

- An OWASP document lists 5 places in a web page you might insert text
  - For the rest, "don't do that"
- Each one needs a very different kind of escaping

Sanitization: tag allow-listing

- In some applications, want to allow benign markup like `<b>`
- But, even benign tags can have JS attributes
- Handling well essentially requires an HTML parser
  - But with an adversarial-oriented design

Don't deny-list

- Browser capabilities continue to evolve
- Attempts to list all bad constructs inevitably incomplete
- Even worse for XSS than other injection attacks

Filter failure: one-pass delete

- Simple idea: remove all occurrences of `<script>`
- What happens to `<scr<script>ipt>`?

Filter failure: UTF-7

- You may have heard of UTF-8
  - Encode Unicode as 8-bit bytes
- UTF-7 is similar but uses only ASCII
- Encoding can be specified in a `<meta>` tag, or some browsers will guess
- `+ADw-script+AD4-`

Filter failure: event handlers

```
<IMG onmouseover="alert('xss')">
```

- Put this on something the user will be tempted to click on
- There are more than 100 handlers like this recognized by various browsers

Use good libraries

- Coding your own defenses will never work
- Take advantage of known good implementations
- Best case: already built into your framework
  - Disappointingly rare

Content Security Policy

- Added HTTP header, W3C recommendation
- Lets site opt-in to stricter treatment of embedded content, such as:
  - No inline JS, only loaded from separate URLs
  - Disable JS `eval` et al.
- Has an interesting violation-reporting mode

Outline

HTTP header injection

- Untrusted data included in response headers
- Can include CRLF and new headers, or premature end to headers
- AKA "response splitting"

Content sniffing

- Browsers determine file type from headers, extension, and content-based guessing
  - Latter two for ~ 1% server errors
- Many sites host "untrusted" images and media
- Inconsistencies in guessing lead to a kind of XSS
  - E.g., "chimera" PNG-HTML document

Cross-site request forgery

- Certain web form on `bank.com` used to wire money
- Link or script on `evil.com` loads it with certain parameters
  - Linking is exception to same-origin
- If I'm logged in, money sent automatically
- Confused deputy, cookies are ambient authority

CSRF prevention

- Give site's forms random-nonce tokens
  - E.g., in POST hidden fields
  - Not in a cookie, that's the whole point
- Reject requests without proper token
  - Or, ask user to re-authenticate
- XSS can be used to steal CSRF tokens

Open redirects

- Common for one page to redirect clients to another
- Target should be validated
  - With authentication check if appropriate
- *Open redirect*: target supplied in parameter with no checks
  - Doesn't directly hurt the hosting site
  - But reputation risk, say if used in phishing
  - We teach users to trust by site