



Hacking Web Applications

Edmund Dengler

e•sentire
Critical Security Solutions

Why the concern?

- ◆ Web applications increasing in volume
- ◆ “Web services” new buzz
- ◆ Need to make sure everything working

“Injection” Attacks

- ◆ Pass in specially crafted inputs that will get inserted into commands or output
 - Cross-site scripting
 - SQL/LDAP injection

Cross-site Scripting (XSS)

- ◆ Inject text to be rewritten into web page
- ◆ Example: Field “FirstName”, use value:

- Foo”>

```
<script language=“javascript”>
```

```
alert(‘hello’)
```

```
</script>
```

```
<br
```

... Cross-site Scripting (XSS)

- ◆ Type 0: Access page in local environment
 - Allows running of commands on target computer
- ◆ Type 1: Inject code into another site
 - Known as “non-persistent”, “reflected”
 - Allows stealing of credentials, etc
- ◆ Type 2: Inject code into a news site
 - Known as “stored”, “persistent”, “second-order”
 - Allows stealing of credentials, etc

SQL Injection

- ◆ Insert SQL commands
- ◆ Example: pass in username/password
 - SQL executes to retrieve person
 - Suppose SQL looks like:
select * from accounts
where username = '\$username'
and password = '\$password'

... SQL Injection

- ◆ Pass in password as:
' or '1' = '1
- ◆ Now SQL looks like:
select * from accounts
where username = '\$username'
and password = " or '1' = '1'

... SQL Injection

- ◆ Information leaking
 - Error codes/messages in web page
 - Useful to help me debug my attacks!
 - For SQL, useful information such as tables not existing
 - Error messages may even tell me what software you are running!

LDAP Injection

- ◆ LDAP = “Lightweight Directory Access Protocol”
- ◆ Used by many sights to manage accounts and access
- ◆ Same idea as SQL, but instead inject LDAP commands

Backend systems

- ◆ LDAP/SQL are examples of back-end system attacks
- ◆ Any other “back-end service” may be similarly exploitable
 - Example: perl scripts
- ◆ Remediation: awareness of “escape” code and prevention of injection

Injection Remediation

- ◆ Trust nothing from the users!
- ◆ Be aware of escape codes for backend systems and output, and either prevent input or escape on output
 - Note: Unicode and other encoding schemes make simple regular expression matching difficult!

Cross Site Request Forgery

- ◆ Many sites use cookies to maintain session
 - Lot easier than session variables!
- ◆ BUT, could allow malicious URLs to be passed to target
- ◆ Example: image tag in an HTML email that would turn off a firewall
 - Image URL executes, session cookie automatically passed, command executes

...Cross Site Request Forgery

- ◆ Guess common apps: Amazon, banking
- ◆ Does require user to be logged on
 - BUT, this is prevalent

...Cross Site Request Forgery

◆ Remediation:

- Using POST may help, BUT Javascript can easily forge POST requests
- Use a local session var only stored in page, NOT cookies
 - Does require Javascript to be on though!

Application Logic Attacks

- ◆ Are you doing the thing correctly?
- ◆ Example: Banking records access
 - Just change the ID, voila, new statement

Brute Force Attacks

- ◆ Try different usernames/passwords
- ◆ If application doesn't limit how many tries I can do, I can eventually find some account if this is a large site
 - Googling email addresses is your friend! (There are even convenient scripts for this)

... Brute Force Attacks

◆ Remediation:

- Limit number of tries for an account in a time period
 - Slow them down!
- Limit number of different unsuccessful username/passwords from a single IP (careful of NAT!)
- Monitor for “low and slow” attacks

Variables

- ◆ Are they a reference or the object?
- ◆ Examples found of use of filenames as parameters (with full paths)
 - What if I just change the filename?
 - Access password files, or anything
 - “Sand-boxing”

Hidden fields

- ◆ The previous were “direct” attacks
- ◆ Can exploit application
- ◆ “Stateless” model causes issues
 - Hard to program!
- ◆ Example: Trading account
 - Hidden field of which account to use
 - Just change the account

XML attack

- ◆ XML data passing for Web services
- ◆ Typically use local library for processing
 - Much more powerful than simple XML checking!
- ◆ If allow included DTD (or inline schemas), then can use commands to access files and such

... XML attack

◆ Remediation:

- Turn off DTD processing (if possible)
- Use a “dumber” parser

Test code “attack”

- ◆ Developers leave old pages on website
 - Fixes done to live pages, but NOT to backups
 - If I can guess the name, I can use the old pages!

Insecure Communications

- ◆ Not encrypting communications allows for sniffing
- ◆ Example: GMail encrypts logon but NOT “mail” pages unless you explicitly ask for it
 - I can read all the mail you are reading over a wireless connection
 - In addition, I can steal your session key and access GMail as you!

Infrastructure attack

- ◆ Maybe your app is safe, but did whoever setup your webserver secure it?
 - Default pages left behind
 - Default services left on
 - Microsoft IIS is worst example of this!

Remediation

- ◆ Don't trust the user!
- ◆ Check, validate, escape everything!
- ◆ Make sure business logic properly implemented!
- ◆ Test for exploits!
- ◆ Don't just think "How do I make this work?", also think: "How can someone break this?"

Automatic Remediation

- ◆ Some more tools are coming along to do code flow analysis to look for standard security errors
 - Note the word “standard”, this will not catch everything!



Thank you!



- ◆ Any questions?