



Application Bug Chaining

Mark Piper
OWASP User
Catalyst IT Ltd.
markp@catalyst.net.nz

OWASP

July 2009

Copyright © The OWASP Foundation
Permission is granted to copy, distribute and/or modify this
document under the terms of the OWASP License.

The OWASP
<http://www.owasp.org>
Foundation

Welcome!

- My name is Mark :)
- Today's Goals:
 - Propagate the basic idea of bug “chaining”
 - Demonstrate that rating web vulnerabilities by severity can be difficult
 - Discuss how we may better classify bug severity
 - Have a little fun
- The Agenda:
 - A look at bug severity
 - Rating bugs
 - Chaining bugs
 - Examine a real world case study

How severe is a bug?

- How to rate a bug?
- Where do we begin?
- The basics:
 - What is the impact?
 - Server compromise?
 - Client compromise?
 - Is authentication required?
 - Other prerequisites?

How severe is a bug? (Cont...)

- How is access to the application obtained?
- Where does the application reside?
- What is the underlying database / OS?
 - Stacked queries?
 - File-system write permissions?
 - File-system read permissions?
- What information is compromised?
- Application availability?
- Can the vulnerability be exploited en masse?

How severe is a bug? (Cont...)

- Classic classification “rules”:
 - Server-Side: Higher severity
 - Client-Side: Lesser severity
 - Un-authenticated: Higher severity
 - Authenticated: Lesser severity
 - Internet facing: Higher severity
 - Internal network: Lower severity
 - Mass exploitability: Higher severity
 - Targeted exploitability: Lower severity

Additional Considerations

- Are there additional mitigations in place?
 - Web application firewalls?
 - Is there timing issues in exploiting the bug?

The Severity Game



Rate The Following Bugs

- Have a crack at rating the severity as:
 - Low → Medium
 - Medium → High
 - High → **Critical**
 - **Critical!**

Round #1

- Issue: SQL Injection
- Underlying DB: MySQL (non-stackable)
- Requires: User-Authentication, GET
- Notes: results in 'non-standard' error page
- URL Example:
*http://site/index.php?
file=TagCloud&module=Leads&action=LeadsAjax&recordid=14&ajx
action=GETTAGCLOUD&recordid=1SQL*
- Result:
*SELECT tag,tag_id,COUNT(object_id) AS quantity FROM site_freetags
INNER JOIN site_freetagged_objects ON (site_freetags.id = tag_id)
WHERE 1=1 AND tagger_id = 2 AND module = 'Leads' AND object_id
= 1SQL GROUP BY tag,tag_id ORDER BY quantity DESC*

Round #2:

- Issue: Arbitrary File Upload
- Requires: User-Authentication, POST
- Notes: resulting file location partially known

- Example:

*“.php” = **BAD**. “.PHP”, “.phtml” = **GOOD**.*

Rate the following (Round #3):

- Issue: Local File Disclosure
- Requires: User-Authentication, GET
- Notes: None

- URL Example:

*http://site/index.php?
action=PortalAjax&mode=ajax&module=Portal&file=../../../../../../../../
../**proc/self/environ%00**&datamode=data*

Rate the following (Round #4):

- Issue: Cross-Site Scripting
- Requires: User-Authentication, GET
- Notes: Reflective

- URL Example:

*http://site/index.php?
module=Calendar&action=index&parenttab=%22%3E%3Cscript
%3Ealert(document.cookie);%3C/script%3E*

Severity?

- Authenticated SQL Injection?
 - Medium → High
- Authenticated File Upload?
 - Medium → High
- Authenticated Local File Include?
 - Low → Medium
- Cross-Site Scripting?
 - Low → Medium

Bonus Round!



Question #1

If the victim of a Cross-Site scripting attack is authenticated to the target application, is the attacker then considered authenticated for any subsequent attacks against the same application?

Question #2

Consider the previous 4 bugs. What happens to the severity of the bugs if we combine them?

1+2...2+3...**2+4?**

Severity?

- Authenticated SQL Injection?
 - Medium → High
- Authenticated File Upload?
 - **Critical!**
- Authenticated Local File Include?
 - Medium → High
- Cross-Site Scripting?
 - **Critical!**
- New finding: “Un-authenticated” Script Execution
 - **Critical!**

Bug Chaining

- Exactly what the name implies!
- Is a mind set more than a “bug class”
- The art of chaining multiple bugs to create exploitable vulnerabilities
- Avoiding pointillistic thinking
- “Glue code”
- Often considered more complex to develop and deliver

Bug Chaining (Cont...)

- Many potential exploit conditions exist
- Client bugs to target server
 - XSS / CSRF / Web Service Clients → server
- Server bugs to target the client
 - SQL injection → client malware
- Server bugs to target other server bugs
 - Shared application resources
 - RPC attacks
- Client bugs to target multiple servers:
 - Client → Application 1 → SSO → Application 2

Bug Chaining (Cont...)

- It is 2009!
- Generally, external is tighter than internal
- That “gooey marshmallow centre” is now the target
- In order to reach the target some creativity is now required by attackers
- A number of frameworks to create complex exploits

Chaining Examples

- PHPMyAdmin \leq 3.1.3:
 - Bug #1: Insecure permissions
 - Bug #2: Script injection
 - Exploit: PHP script execution
- SugarCRM \leq 5.2.0e:
 - Bug #1: Flawed extension validation
 - Bug #2: Predictable file name
 - Bug #3: Direct file request (?)
 - Exploit: PHP script execution

A better way?

- How may we better determine the severity of a bug?
- CVSSv2?
 - **C**ommon **V**ulnerability **S**coring **S**ystem v2.0
 - Adopted by many organisations
 - Considers exploit complexity, application location, authentication, target likelihood etc.
 - Can get very complex
 - Can often be time consuming
 - Can be difficult to follow

The VtigerCRM Example

"You can explain this stuff all day, but when network admins actually see you do it, that's when they learn" - Brett Moore

The VtigerCRM Example

- Large Open-Source CRM system
- Reported issues in 2008
- Fixed in 5.0.4 “Security Update 1”
- Patched version is **not** the default download
- Combine bugs #2 and #4 to create & execute a remote command execution exploit (connect-back)
- This is a very common condition
- We wont cover XSS delivery

Chaining #2 & #4

- Use XSS to control the users browser
- Generate a file to upload
 - Connect-back shellcode
- Have the user upload on our behalf
 - HTTP POST via AJAX
- Have the user discover & request the file
 - Only have a partial location
 - We may not be able to directly request
 - Brute force

Chaining #2 & #4 (Cont...)

- Introducing BeEF:

- By Wade of NGS / bindshell.net
- Browser Exploitation Framework
- Modular exploits
- Autorun modules
- Control multiple victims
- Originally written to demonstrate Inter-Protocol Exploitation (IPE)

Chaining #2 & #4 (Cont...)

- VtigerCRM Beef Module:
 - Javascript (client payload)
 - PHP (attack assistance)
 - No requirement for the user browser to remain open
 - Maybe be executed as an auto-run module
 - Written for this demo in < 2 hours

http://freedomisnothingtofear.com/xplt_vtiger.tar.gz

DEMO!

References / Links

- <http://www.first.org/cvss/>
- <http://www.owasp.org/>
- <http://vtiger.com/>
- <http://bindshell.net/>
- <http://nostarch.com/js2.html>
- <http://secunia.com/>

Questions?

markp@catalyst.net.nz