

Hands-on Security Tools

SecAppDev 2009

Caveats and Warnings

- This is **not** a sales pitch for any product(s)
 - If you want to talk to a sales person, tell me
 - Otherwise, you will NOT get calls or spam
- You are **not** authorized to “test” any systems other than your own
 - If you do, then don’t call me from prison
 - I don’t know you

Prerequisites

- Computer (shared or solo)
 - Windows, OS X, Linux
 - Local admin access
- Virtual machine environment (Vmware, Parallels, Virtual Box)
- JDK (newer is better)
- Development environment (for source analysis tool)
 - C or Java
 - Make, Ant, Eclipse (3 or 2), Visual Studio, Rational, Websphere

Objectives and Intros

- We'll look at several tools described in my *security tools* class
- Idea is to give everyone a glimpse of several tools
 - NOT to turn anyone into an expert on any tool
- Safe, sales-free env
- Flow
 - Describe each tool
 - Demo (where applicable)
 - Class tries tool with specific objectives
 - Discuss results and applicability

Secondary Goals

- Learn
- Experiment with the tools
- Judge for yourself
- Have fun

Setup environment

- We'll use a combination of stuff
 - Live CDs
 - OWASP, Network Security Toolkit (NST) 1.8
 - Desktop installations
- For live CDs, virtual machine is highly recommended
 - Copy CD image ISO into your VM folder
 - Set up separate Linux VMs for each
 - Recommend “no hard drive” options

Infosec tools

Categories include

- Network port scanners
- Vulnerability scanners
- Application scanners
- Web application proxies
- Network sniffers

(For a great list, see <http://sectools.org/>)

Software security tools

Categories include

- Static code analysis tools
- Testing tools
 - Fuzzers
 - Interposition tools
 - System monitors
 - Process analyzers
 - Etc.

Network and vul scanners

Usage: determine open and potentially vulnerable network services

- Mainstay of “penetration testers”
- Useful for verifying deployment environment
- Validating on-going maintenance
- Rarely directly valuable to developers

Examples

- Nmap, nessus, Metasploit, ISS, Core Impact, Retina

NMAP

- [Http://nmap.org](http://nmap.org)
- Open source and free
- Available on numerous OSes
- Command line and GUI
- Unix command-line folks will love this...
 - *nmap -h* lists options
 - Numerous !

Nessus

- <http://nessus.org>
- Free, but not open source
 - Parent company is <http://www.tenablesecurity.com>
 - Commercial
- Supports several OSes
 - Linux (RH, Suse, Debian, but not Ubuntu)
 - Windows, OS X, Solaris, FreeBSD
- Client/server model (but 3.0 can now run without server)

Metasploit

- <http://metasploit.org>
- WARNING!!!
- Open source exploit/payload tool
- Extensible with exploits written in Ruby
- Runs on most OSes
- CLI, menu, GUI, and WUI front-ends

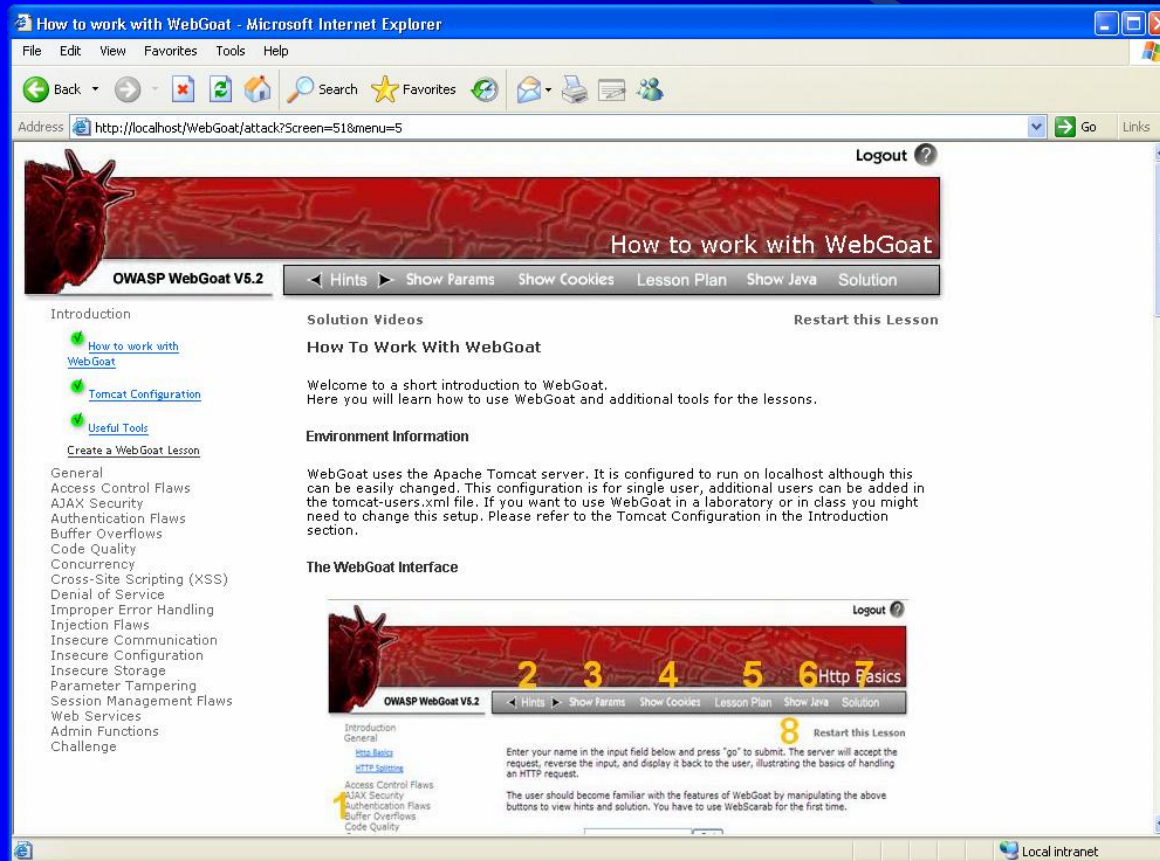
Web application testing

- First, the manual approach
 - A lot of times, there's no substitute for this
 - Kind of like a single-stepping debugger
- Testing proxies are useful
 - Man-in-the-middle between browser and app
- Examples
 - **WebScarab**, Paros Proxy

The tools we'll use

- OWASP tools (freely available)
 - Your web browser (IE or Firefox preferred)
 - **WebGoat** -- a simple web application containing numerous flaws and exercises to exploit them
 - Runs on (included) Apache Tomcat J2EE server
 - **WebScarab** -- a web application testing proxy
- Instructor demo
- Class installation of both tools

WebGoat

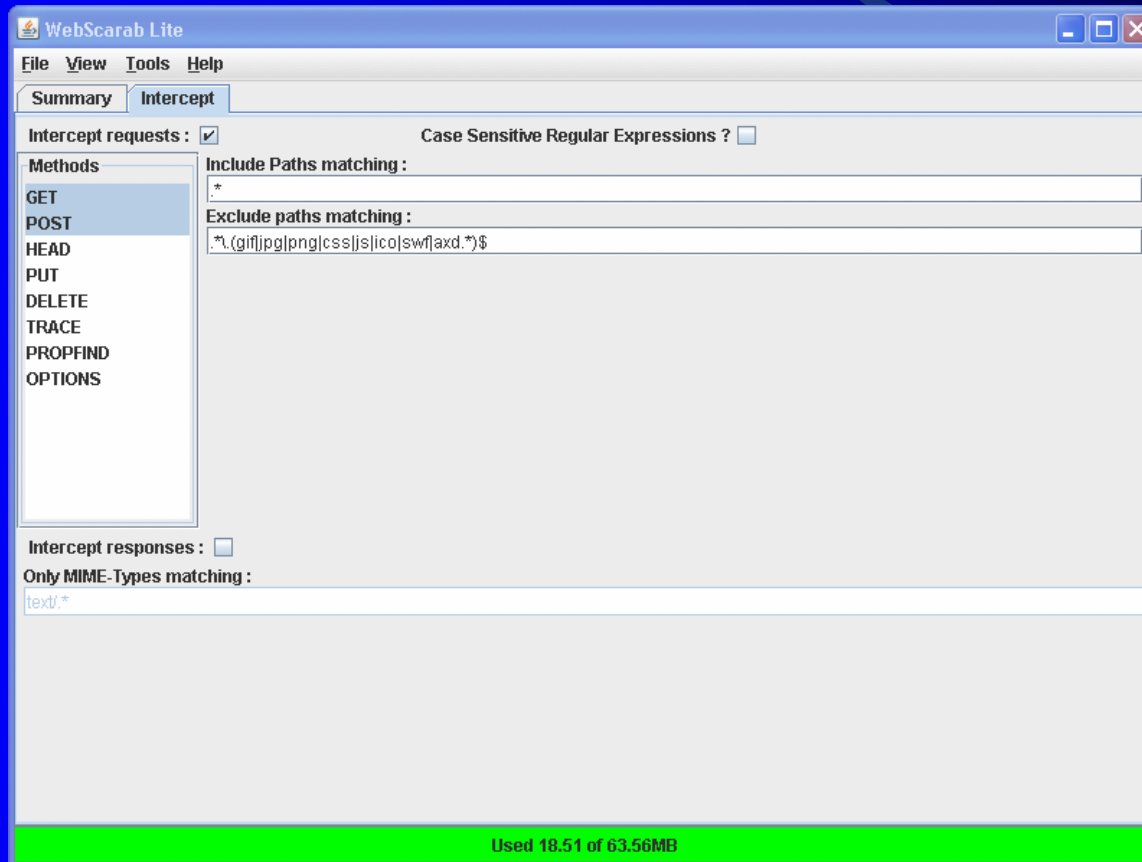


Setting up WebGoat

- Run WebGoat on TCP port 8080
 - From WebGoat folder (GUI or command line)
 - Windows: *webgoat_8080.bat*
 - OS X or Linux: *./webgoat.sh start8080*
 - (Will need to *chmod +x webgoat.sh* first)
 - Verify in browser *http://localhost:8080/WebGoat/attack*

At this point, WebGoat is running, but you'll still need a testing proxy to perform some attacks

WebScarab



Next, set up WebScarab

- Run WebScarab
 - Default listener runs on TCP port 8008
 - Ensure listener is running within WebScarab
- Configure proxy
 - Set *web browser proxy* point to TCP port 8008 on 127.0.0.1 (localhost)
 - Include proxy for localhost
 - Connect once again to
<http://localhost:8080/WebGoat/attack>

Troubleshooting

- Scarab not running
 - Listener turned off or on wrong port
- Browser proxy not configured or misconfigured
 - IE behaves differently than Firefox
 - IE 7 often “misbehaves”
 - Make sure proxy is set for localhost and 127.0.0.1
 - Try using 127.0.0.1. (note the “.” at end)
 - Turn off anti-phishing or “safe browsing” features
 - Ensure JavaScript is running
 - Try Firefox if you are an IE user, and vice versa

WebGoat tips

- Report card shows overall progress
- Don't be afraid to use the “hints” button
 - Show cookies and parameters can also help
 - Show java also helpful
 - None of these are typical on real apps...
- Learn how to use it
- Fabulous learning tool

Familiarizing Goat and Scarab

- WebGoat

- Do “Web Basics” exercise
- Try Hints and other buttons
- Look at report card

- WebScarab

- Turn on intercepts
 - Requests
 - Responses
- Explore and experiment
 - Parsed vs. raw view
- Try editing a request
 - Modify parameters
 - Add/omit parameters

#1 Cross site scripting (“XSS”)

- Can occur whenever a user can enter data into a web app
 - Consider all the ways a user can get data to the app
- When data is represented in browser, “data” can be dangerous
 - Consider this user input

```
<script>
alert(document.cookie)
</script>
```
- Where can it happen?
 - ANY data input
- Two forms of XSS
 - Stored XSS
 - Reflected XSS
- Two WebGoat exercises to see for yourself

Stored XSS

- Attacker inputs script data on web app
 - Forums, “Contact Us” pages are prime examples
 - All data input must be considered
- Victim accidentally views data
 - Forum message, user profile, database field
- Can be years later
 - Malicious payload lies patiently in wait
 - Victim can be anywhere

Stored XSS exercise

Stored XSS Attacks - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address <http://localhost/WebGoat/attack?Screen=50&menu=900>

Logout ?

Stored XSS Attacks

OWASP WebGoat V6.2

◀ Hints ▶ Show Params Show Cookies Lesson Plan Show Java Solution

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)

Phishing with XSS

LAB: Cross Site Scripting

Stage 1: Stored XSS
Stage 2: Block stored XSS using Input Validation
Stage 3: Stored XSS Revisited
Stage 4: Block stored XSS using Output Encoding
Stage 5: Reflected XSS
Stage 6: Block Reflected XSS

Stored XSS Attacks

Cross Site Request Forgery (CSRF)

Reflected XSS Attacks

HTTPOnly Test

Cross Site Tracing (XST) Attacks

Denial of Service
Improper Error Handling
Injection Flaws

Solution Videos Restart this Lesson

It is always a good practice to scrub all input, especially those inputs that will later be used as parameters to OS commands, scripts, and database queries. It is particularly important for content that will be permanently stored somewhere in the application. Users should not be able to create message content that could cause another user to load an undesirable page or undesirable content when the user's message is retrieved.

Title:

Message:

Submit

Message List

ASPECT SECURITY
Application Security Specialists

OWASP Foundation | Project WebGoat | Report Bug

Local intranet

Reflected XSS

- Attacker inserts script data into web app
- App immediately “reflects” data back
 - Search engines prime example
 - “*String* not found”
- Generally combined with other delivery mechanisms
 - HTML formatted spam most likely
 - Image tags containing search string as HTML parameter
 - Consider width=0 height=0 IMG SRC

Reflected XSS exercise

Reflected XSS Attacks - Microsoft Internet Explorer

Address: http://localhost/WebGoat/attack?Screen=49&menu=900

Logout ?

Reflected XSS Attacks

OWASP WebGoat V6.2

Hints Show Params Show Cookies Lesson Plan Show Java Solution

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)

Phishing with XSS

LAB: Cross Site Scripting

- Stage 1: Stored XSS
- Stage 2: Block stored XSS using Input Validation
- Stage 3: Stored XSS Revisited
- Stage 4: Block stored XSS using Output Encoding
- Stage 5: Reflected XSS
- Stage 6: Block Reflected XSS

Stored XSS Attacks

Cross Site Request Forgery (CSRF)

Reflected XSS Attacks

HTTPOnly Test

Cross Site Tracing (XST) Attacks

Denial of Service
Improper Error Handling
Injection Flaws

Solution Videos [Restart this Lesson](#)

For this exercise, your mission is to come up with some input containing a script. You have to try to get this page to reflect that input back to your browser, which will execute the script and do something bad.

Shopping Cart

Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	69.99	<input type="text" value="1"/>	\$0.0
Dynex - Traditional Notebook Case	27.99	<input type="text" value="1"/>	\$0.0
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	<input type="text" value="1"/>	\$0.0
3 - Year Performance Service Plan \$1000 and Over	299.99	<input type="text" value="1"/>	\$0.0

The total charged to your credit card: \$0.0

Enter your credit card number:

Enter your three digit access code:

Done Local intranet

XSS issues

- Why is this #1?
 - Input validation problems are pervasive
 - Focus on functional spec
 - Why is it such a big deal?
 - *Highly* powerful attack
 - Anything the user can do, the attacker can do
 - Take over session
 - Install malware
 - Copy/steal sensitive data
- Reflected (via spam email) attacks most common technique by phishers*

How bad is XSS?

- Perhaps the most (in)famous example is the MySpace Samy virus
 - XSS content in author's page that added any viewer as a friend whenever viewed
 - In less than 24 hours, Samy had > 1 million "friends"
 - MySpace crashed and was down for 3+ days

JavaScript Obfuscation

- Used to hide the real intent of a JS
- Many (!) examples exist
- Increasingly difficult to detect

- **Example**

```
var a="Hello World!";  
function MsgBox(msg)  
{  
  alert(msg+"\n"+a);  
}  
MsgBox("OK");
```

- **Becomes**

```
//language=jscript.encode  
#@~^1wAAAA==~mD~|!X FF  
XT']Jw6W%wa+*-X□Z'6v;wavw-X T-  
aXF-avww6F wa+Z-a□W-a  
8EBJwX!zJ~r-X*s'6*ArTI-  
mDP|T6yFGyaq'!!X  
qG+aZ$T6ZDi6EU^DkWU~|!a 8{y6+v{Z6  
8Gya&*CV□DOc|!6yqGy6&3mT6yFF  
a!,TXFD_|T6yF{+XF#IN,im!X+8G+X  
v{!X 8{ X!,!X Dbp5j4AAA==^#~@
```

- (Source javascriptobfuscator.com)

Application vul scanners –1

Category of black box test tools that attempts additional “application level” vul probes

- E.g., SQL injection, buffer overflows, cookie manipulation, Javascript tampering
- Increasing in popularity among pen testers
- Useful at verifying (web) app is not vulnerable to the most common attacks
- Moderately useful to developers

Application vul scanners –3

Examples

- IBM/Watchfire's Appscan, HP/SPI Dynamics' WebInspect, Nikto

Nikto

- <http://nikto.org>
- Written in Perl
- Simple and low-level app scans

AppScan

- <http://www.watchfire.com> (acquired by IBM)
- Windows only
- Commercial application scanner
- We'll look at eval copy
 - *Only* able to scan <http://demo.testfire.net>

Fuzzers –1

Growing field of app testing that involves sending malformed data to/from app

- Tools, frameworks, and APIs are popping up
- “One size fits all” approach is highly problematic
 - Informed fuzzing vs. uninformed fuzzing
- Still early adoption among pen testers (arguably)
- Dev knowledge is necessary to get most of it

Fuzzers –2

- Fuzzing can and should be done from unit to entire app tests
- QA test team needs to acquire and learn

Examples

- OWASP's **JBroFuzz**, PEACH, SPI Fuzzer, GPF, Codenomicon, Mu Security, SPIKE, Sulley

“At Microsoft, about 20 to 25 percent of security bugs are found through fuzzing a product before it is shipped”

JBroFuzz

- http://www.owasp.org/index.php/Category:OWASP_JBroFuzz
- Open source from OWASP
- Simplistic, but can fuzz
 - Fields in any web app form
 - URL guessing
- Project is still alpha-stage

Static code analyzers –1

Review source code for common coding bugs

- A bit of history

- 1999: First examples appear from research projects
 - E.g., ITS4, RATS, Flawfinder
 - Tokenize input streams and perform rudimentary signature analysis
 - Accurate at finding *strcpy()* and the like, but lacking context to really be useful

Static code analyzers –2

- 2001: “2nd generation” tools arrive
 - E.g., Fortify, Ounce Labs, Coverity
 - Parse and build abstract syntax tree for analysis
 - Enables execution flow, data flow, etc., traces
 - Significant leap forward, but much work remains
 - Hundreds of common bugs in several languages
 - Management tools for overseeing, measuring, and policy enforcement
 - Integration into popular IDEs
- Still, many are shelfware

Static code analyzers –4

- *Then* do large scale analysis at project completion
- Possibly using more than one tool set

Examples

- Fortify *SCA*, Ounce Labs *Ounce 5*, Coverity *Prevent*, Klocwork

Fortify SCA

- <http://fortifysoftware.com>
- Commercial source code analyzer
- Supports numerous platforms, languages, compilers, and IDEs
- *License caveats for this class*
- Other features: security manager, rule builder

The Challenge!

- Rules of the game

1. You may use WebScarab
2. All access to the Challenge app must be via browser
3. You may NOT use command-line or other OS interface
4. Questions are ok, but I will answer to everyone

Kenneth R. van Wyk
KRvW Associates, LLC

Ken@KRvW.com

<http://www.KRvW.com>

