Detection and Evasion of Web Application Attacks

K. K. Mookhey

Founder & CTO

Network Intelligence (I) Pvt. Ltd.

www.nii.co.in

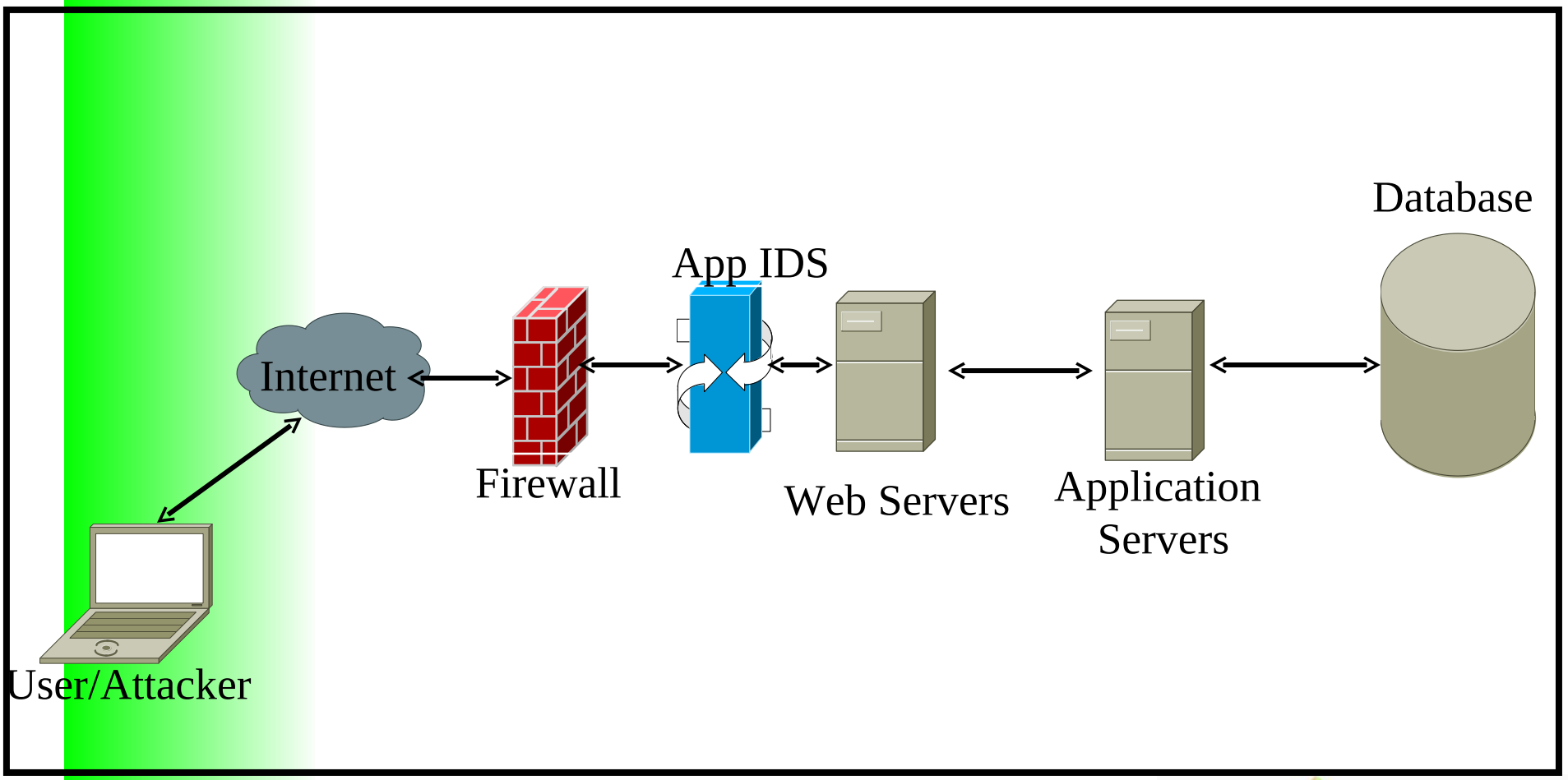
Agenda

- Static Detection Techniques
- Dynamic Detection Techniques
  - Signature-based
  - Anomaly-based
- Typical Web Application Attacks
- Signature-based detection
- Anomaly-based detection
- Conclusion

Current state

- Web applications represent highly vulnerable attack avenues
- Most discussions on web application security, center on attacking it and secure coding to protect it
- Methods for ***detecting*** such attacks are coming into their own
- Existing detection methods are being tested before customers accept these solutions as standard

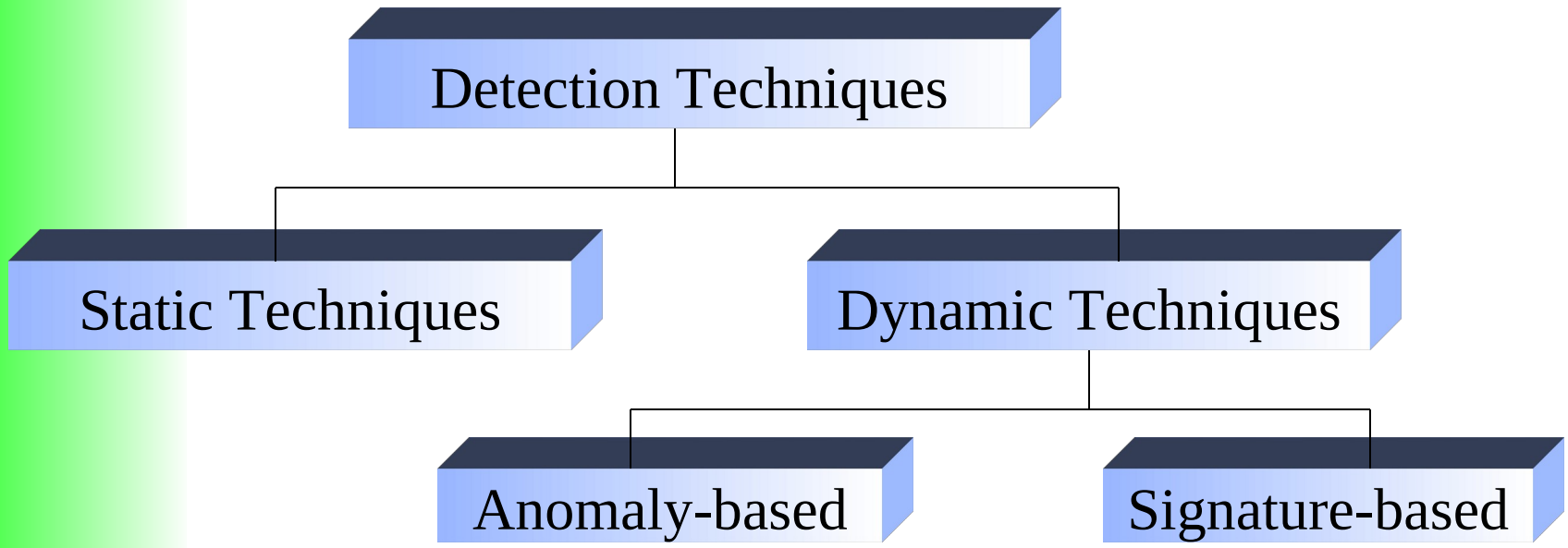
Solution Positioning



NETWORK INTELLIGENCE
Information Security Consulting

Detecting Web Application Attacks

My classification



What information is needed

- We need the following fields for an effective investigation:
  - Source IP
  - Timestamp
  - HTTP Method
  - URI requested
  - Full HTTP data sent
- Attack data could be in:
  - URI
  - HTTP headers from client
  - Cookie
  - Basically anywhere

Detection Techniques

- Using static techniques
  - Happens post-occurrence of event
  - Parse log files using standard tools/techniques
  - Aim is forensics investigation
- Using dynamic techniques
  - Detect the attack as it happens
  - Trigger alarms when attack is happening
  - Aim is detect/prevent in real-time

Static Detection

Static detection techniques

Data sources to look at:

- Web Server Logs
- Application Server Logs
- Web Application's custom audit trail
- Operating system logs

What's missing:

- POST data (only GET data available)
- HTTP Headers only partially represented
- Cookie or Referer data depends on web server

Web Server Logs

IIS Web Server Log entry (with almost all options selected)

2004-06-23 11:44:53 192.168.0.70 POST /sqlinject2.pl - 200 797 640 43082 HTTP/1.1 Mozilla/5.0+(Windows;+U;+Windows+NT+5.0;+en-US;+rv:1.4)+Gecko/20030624+Netscape/7.1+(ax) - http://192.168.0.25/sqlinject2.html

This is an SQL injection attack – surely doesn't look like one!

POST Request data is missing

HTTP Headers are missing

Static Detection Fails to detect:

- HTTP Header attacks can't be detected:
  - The Template: F attack can't be detected
  - Attacks that overflow various HTTP header fields
- Web application attacks in a POST form
  - SQL injection
  - Cross-site scripting
- Forceful browsing – user tries to access page without going through prior pages that would ensure proper authentication and authorization

Static Detection does detect:

- Automated attacks using tools such as Nikto or Whisker or Nessus
- Attacks that check for server misconfiguration (../../winnt/system32/cmd.exe)
- HTML hidden field attacks (only if GET data – rare)
- Authentication brute-forcing attacks
- Order ID brute-forcing attacks (possibly) – but if it is POST data, then order IDs cannot be seen

Dynamic Detection

Dynamic detection techniques

Methods:

- Application Firewall
- In-line Application IDS
- Network-based IDS (possibly) adapted for applications

Advantages:

- Complete packet headers and payload available
- Including HTTP headers
- POST request data
- URI request data

Dynamic Detection Techniques

- The web application intrusion detection space is divided into two possibilities:
  - Signature-based
  - Anomaly-based
- Each has its own implementation and effectiveness issues

Comparison Table

Signature-based	Anomaly-based
Easier to implement	More complicated
Cheaper – DIY	Mostly commercial solutions
False positives	False positives are fewer, but…
As well as false negatives	False negatives as well
Popular for detecting known web server attacks. Can be tweaked to do decent web application detection.	Used for both web server, as well as web application attacks

Signature-based

- Snort IDS has 868+ signatures out of 1940+ for web layer attacks
- Most are for known vulnerabilities in web servers, such as:
  - IIS directory traversal
  - IIS .ida, .idq, etc. attempts
  - Chunked Transfer-encoding attacks
- Only a few are generic signatures for web application attacks, such as for:
  - cross-site scripting
  - /usr/bin/perl or other Unix command attempts

mod_security

- Works specifically with Apache
- Can scan in-depth and fine-grained checks
- Can scan cookies as well
- Also supports PCRE
- Can be configured as IPS – ‘exec’
- Can’t detect:
  - Session id brute forcing
  - Forced browsing
  - Authentication brute-forcing
  - HTML hidden field manipulation
- Comes with a Perl script to convert all Snort rules to its own ruleset

The Attacks

Web Server Attacks (A10)

- These are usually with tools such as Nikto or Nessus
- Default run of these tools is easily detected by Snort or any other IDS: rules will fire all over the place
- Tools have IDS evasion techniques
- Effective only to some extent, eventually you will get flagged
- More flags will be ‘DOUBLE DECODING ATTACK’ on Snort
- Demo {}

Downloading entire website

- Often an attacker will crawl the entire website and download it locally
- Objective is to study the process flow, structure, and overall programming logic used by developers
- Also to find out any client-side javascripting encryption or obfuscation used
- Also to search for HTML comments or any other pieces of critical information

Detection

- Similar to a portscan – but its happening at the application layer
- Web site logs will show almost entire website being accessed in a very short time interval
- Almost impossible to write an signature for this
- Perfectly suited for anomaly detection
- How about a Snort preprocessor for this? – issues?

Cross-site scripting (A4)

XSS

- Attacks the end-user
- Works due to failure in input as well as output validation by the web application
- User input is produced without parsing as output
- Works by inserting HTML meta-tags, which contain java script or other malicious code

Cross-site scripting

- Existing snort signatures:

For typical <script>alert(document.cookie)</script>

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS
$HTTP_PORTS (msg:"WEB-MISC cross site scripting attempt";
flow:to_server,established; content:"<SCRIPT>"; nocase;
classtype:web-application-attack; sid:1497; rev:6;)
```

For typical <img src=javascript> attack:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS
$HTTP_PORTS (msg:"WEB-MISC cross site scripting HTML
Image tag set to javascript attempt"; flow:to_server,established;
content:"img src=javascript"; nocase; classtype:web-
application-attack; sid:1667; rev:5;)
```

Evasion of these

- Can be trivially evaded:
  - <a href="javascript#[code]">
  -
  - <img src="javascript:[code]">
  - <xml src="javascript:[code]">
- Demo {}

Better signatures

- Enter PCRE – Perl Compatible Regular Expressions
- Greater flexibility
- One signature can catch multiple attacks
- Lower learning curve for Unix admins – regex is part of daily life
- Regular expressions work with:
  - Snort IDS
  - Eeye's SecureIIS
  - Apache's mod_security (best bet)

Signatures for XSS

- **/((\%3D)|(=))[^\n]*((\%3C)|<)[^\n]+((\%3E)|>)**
- Checks for occurrence of:
  - =
  - Followed by one or more non-newline characters
  - Followed by < or hex-equivalent
  - Zero or more /
  - And then > or hex-equivalent
- This will catch almost any remote attempt to attack XSS
- Very few false positives

Demo of XSS Signatures

Malicious redirection

- Some sites have code which will redirect user to another part of the website or a partner website:

http://www.nii.co.in/redirect.php?target=www.partnersite.com

This can be manipulated to

http://www.nii.co.in/redirect.php?target=www.evilsite.com

Can be obfuscated using hex or Unicode encoding or even URL mangling:

1. Redirection to IP address in Octal or Hex (URL Munge)

2. Conversion to URL encoded values

http://www.nii.co.in/redirect.php?target=%68%74%74%70%3A%2F%2F%37%35%32%37%32%30%32%38%31%37

Attack outcome is similar to an XSS attack

Detection of this

- If the set of sites to which redirection is to be allowed is know
- Then signature can be written in PCRE to detect any input that is not belonging to that set:

target=[^(www.partner.com)]

- Mod_security can be used to refer specifically to the particular argument type as well
- Anomaly-based detection is ideal, since the bank of clean data would include only references to partner.com
- And by definition, any variation would be flagged

Forceful browsing (A2)

- User tries to directly access a web page that requires previous authentication
- If web application is badly coded, attempt may be successful
- For instance, access to http://www.nii.co.in/orders.php, requires successful authentication at: http://www.nii.co.in/login.php
- Very difficult to write signature, unless there is a stateful application engine that records whether authentication was first successful or not
- Anomaly-based detection is best bet

SQL Injection (A6)

SQL injection

- Demo of standard SQL injection attack {}
- Typical attackers will try the following:
  - Just a single-quote
  - A boolean True expression: 1’or’1’=‘1
  - A commented input admin;--
- At an intermediate stage:
  - SELECT, INSERT, UPDATE, DELETE, etc.
- At an advanced stage:
  - UNION
  - EXEC XP_CMDSHELL

SQL injection – key inputs

The key input types for this are:

- SQL meta-characters:
  - Single-quote
  - Comment characters
  - Query separators, such as semi-colon (;)
- Some boolean logic sooner or later
- Possibly the word ‘union’ or ‘select’ or ‘insert’ or ‘delete’ at an advanced stage
- Possibly even ‘exec xp_cmdshell’, if attacker determines database as Microsoft SQL Server

Regex for SQL injection detection

- **[^\n]+(\%3D)|(=)) (\%27)|(\')|(\-\-)|(\%23)**
- Typical POST data would look like:
  - username=test&password=1’or’1’=‘1
- Watch out for:
  - One or more non-newline characters
  - Followed by the = sign, which denotes the occurrence of an input field
  - Then the single-quote or hex-equivalent
  - Or double-dash (as comment character)
  - Or semi-colon
  - Or /**/ if used for evasion

Problems

- Leads to false positives
- Some of the characters could occur as genuine non-malicious input:
  - O'Conner??
- Need further tweaking
- But could be kept for later forensics
- Important: With mod_security, this signature can be added at a more fine-grained level – specific parameter within a specific script to be checked
- Also, mod_security can scan cookie values as well

Boolean SQL injection

- Intention is to manipulate the SQL query into a true value always:

Select username, password from user_table where username=‘user_supplied_input1’ and password=‘user_supplied_input2’

If user supplied password as

1’or’1’=‘1

Query becomes

Select username, password from user_table where username=‘user_supplied_input1’ and password=‘1’or’1’=‘1’

Regex for this

- Attack signature could be **1’or’1’=‘1**
- Could also be **1’or’B’>’A**
- Could be any Boolean expression, as long as it is OR’ed and results in a TRUE value

/\w*((\%27)|(\’))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix

- Explanation
- **\w*** - zero or more alphanumeric or underscore characters
- **(\%27)|\’** – the ubiquitous single-quote or its hex equivalent
- **(\%6F)|o|(\%4F))((\%72)|r|(\%52)** – the word ‘or’ with various combinations of its upper and lower case hex equivalents.

Caveat: be careful if your application uses forms such as process.php?id=OR123

Other keywords to detect

- EXEC XP_
- EXEC SP_
- OPENROWSET
- EXECUTE IMMEDIATE
- UNION SELECT
- INSERT
- UPDATE

Evasion of these

Some common evasion techniques, which need to be taken care of:

- Different encodings, such as URL encoding, or UTF-8 encoding.

  Counter: Snort preprocessors decode encoded URL strings before applying signature check

- White spaces used intermittently by attacker

Counter: Use [\s]+ to check for one or more whitespaces

- Use of SQL comments -- or /**/

Counter: write signature for detecting:

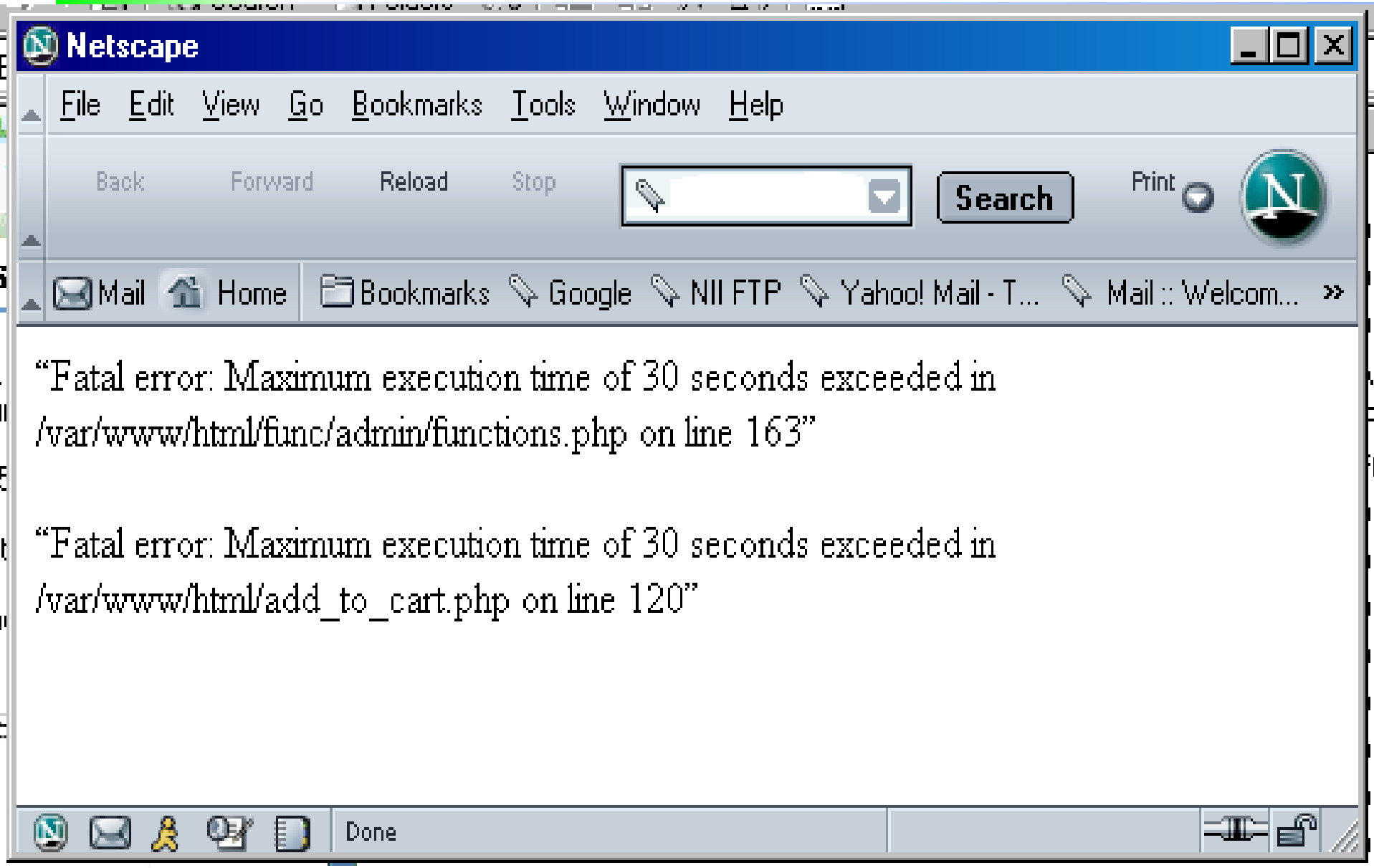
--

/*

Other attacks

Buffer overflows (A5)

- Buffer overflows against web applications do not always yield significant results
- Buffer overflows are typically used to exploit known vulnerabilities
- However, sometimes interesting information can be revealed
- For instance, a large input value is entered into an input field, and gets fed into a PHP function producing the following error
- Note this is also A7 ‘Improper Error Handling’ in OWASP Top Ten



Signature for this?

- So the regex might look like this:

/[\w]+\=[^\=]{500,+}\&/

- Other alternatives are to use the SecFilterByteRange in mod_security

```
SecFilterByteRange 32 126
```

- Or use the the following directives within Apache
  - *LimitRequestBody*
  - *LimitRequestFieldsize*

Command Execution (A7)

- Used if input may be going into a Perl or PHP system() call or C execve() call
- Say a URI like "lame.cgi?page=ls%20-al"
- All the characters could easily occur as part of a genuine URI
- Snort has multiple signatures for various OS commands
- Snort signatures can misfire
- mod_security comes with Perl script to convert most of the Snort rules to its own directives
- SecureIIS and URLScan can do the same job for IIS

Null byte poison attack (A1)

- Used to end an input string, as the null byte is the end-of-string character in C
- "%00"
- This is definitely malicious traffic
- Null byte has no business in genuine URIs
- Trivial to detect
- Just watch out for ‘%00’

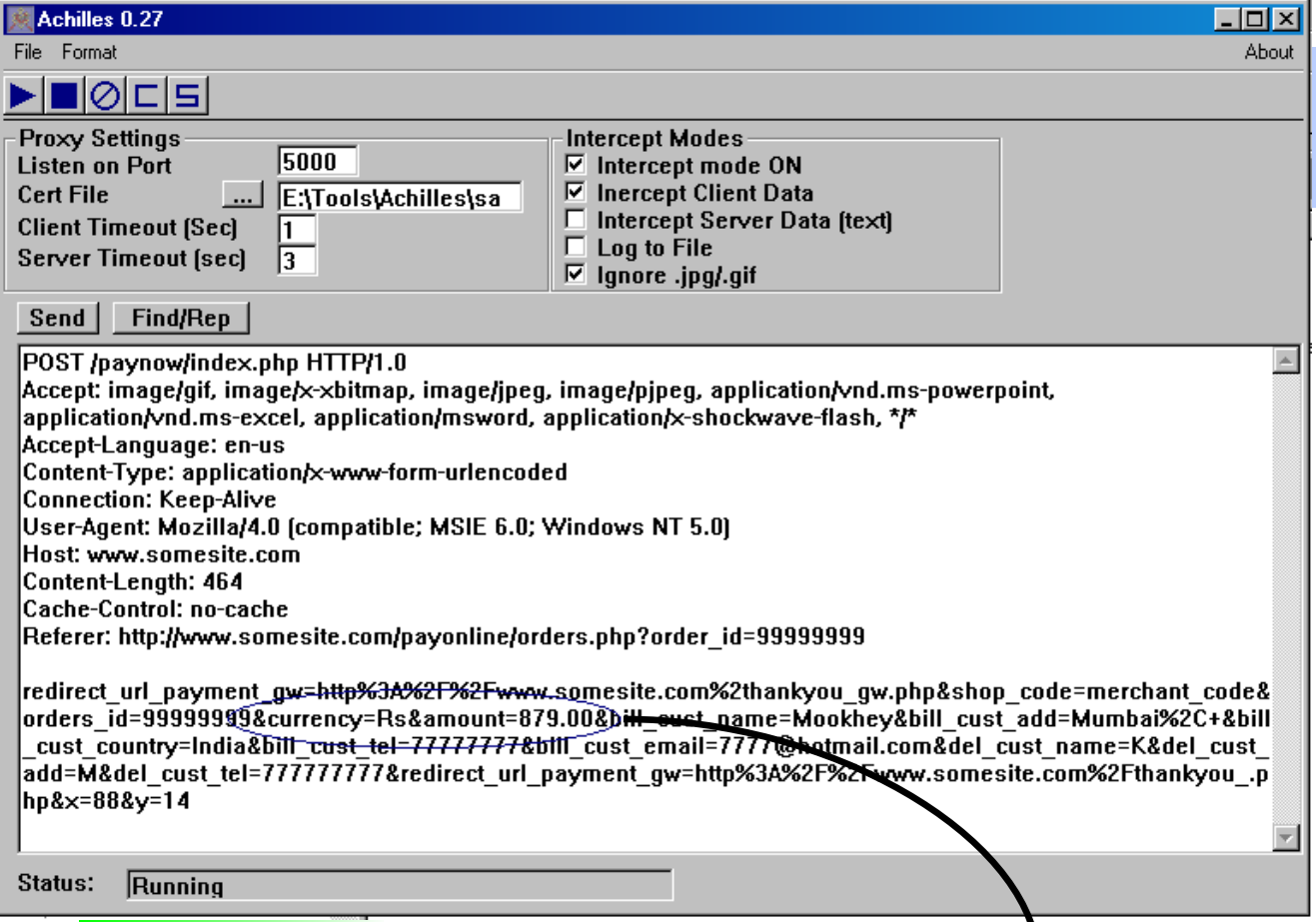
The pipe ‘|’ (A1)

- Used for piping the output of one command into the input of another
- Used if input may be going into a Perl or PHP system() call or C execve() call
- This is also definitely malicious
- Trivially, identified, and signature can be written

Where signature-based detection fails

Hidden Field Manipulation

- Developers assume HTML hidden fields will be input unchanged
- Parameter manipulation:
  - Attacker manipulates price from $200 to $2
  - Almost impossible to write a working signature for this
  - Anomaly-based detection would (possibly) work

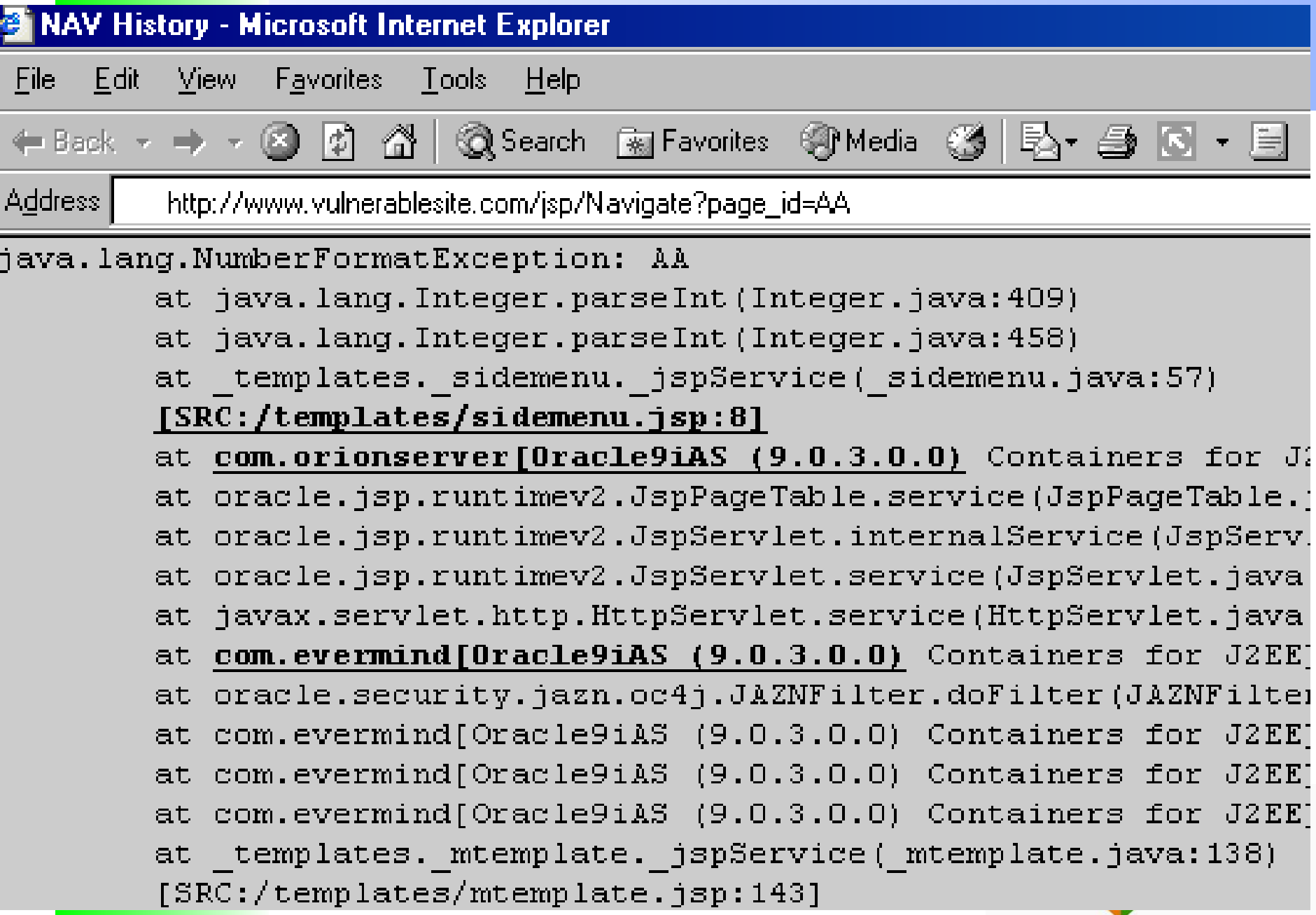


currency=Rs&amount=879.00&

NETWORK INTELLIGENCE
Information Security Consulting

Invalid input (A1)

- Entering numeric values in web applications, where alphabets are expected
- Entering alphabets where numeric values are expected
- Modifying the case of the file being requested
- Attempts such as these typically yield information or path disclosure results
- Not possible to write specific signatures for all the input fields in the web application



Authentication/Authorization Attacks (A3)

- These attacks typically brute-force the authentication mechanism
- For example, use of Brutus for dictionary-attack against Basic Authentication or Form-based Authentication
- Or custom Perl script for brute-forcing session IDs or order IDs, or any such similar attack if these IDs are not truly random enough:

```
http://www.nii.co.in/getorder.php?order_id=200406271
http://www.nii.co.in/getorder.php?order_id=200406272
http://www.nii.co.in/getorder.php?order_id=200406273
http://www.nii.co.in/getorder.php?order_id=200406274
```

Detection of these

- Not possible to write a signature to detect these
- As individually, each request is perfectly legitimate traffic
- Static detection techniques using log analysis, might detect it
- But if POST request is used, then all that will be seen in the logs is repeated requests for:

http://www.nii.co.in/getorder.php?order_id

And not the actual Order ID being requested

Possible detection

- Possible if some sort of rule-based correlation (RBC) can be used
- An RBC rule could say,
  - if Snort flags this 10 times within 60 seconds from the same source IP
  - then raise a stink
- A Snort rule could be created, if there is an outgoing HTTP 401 Authentication Failed message
- But, genuine mistakes during authentication would raise too many alarms to investigate

Another possibility?

- These attacks are similar in nature to portscans at the network layer
- Rapid HTTP requests for URLs that change at specific locations:
  - Either the form fields (for authentication attacks)
  - Or session IDs (for authorization attacks)
- Could a Snort preprocessor be possibly written for this?

Anomaly-based Detection

Anomaly-based

- Based on assumption that normal traffic can be defined
- Attack patterns will differ from such ‘normal’ traffic
- Anomaly-based detection system will go through a learning phase to register such ‘normal’ traffic
- Analysis will be done for individual field attributes as well as for entire query string
- This difference should be able to be expressed quantitatively

Anomaly-based

- Some of the attributes that could be analyzed are:
  - Input length
  - Character distribution
  - Parameter string structure
  - Parameter absence or presence
  - Order of parameters
- Important: Learning must be on actual web traffic, not old web server logs. Logs do not contain all critical data where attack traffic could occur, such as cookies or HTTP headers, POST data, etc.
- Commercial products dominate this field
- Choice is influenced by cost-benefit analysis

A quick overview

- AppShield from Sanctum Inc.
- Imperva's SecureSphere
- Teros Secure Application Gateway
- NetContinuum's Application IDS

Conclusion

Key points

- Signature-based IDS is good enough to detect a large majority of initial web app attacks
- It fails in detecting certain unique attacks, such as price manipulation or forceful browsing or malicious redirection
- Some signatures may yield large number of false positives
- Anomaly-based detection is based on training the IDS to learn normal web traffic
- Products are still maturing
- Maybe best solution is a combination of signature-based to detect majority of simpler attacks, and anomaly-based to detect sophisticated application-specific attacks
- Cost-benefit will be the deciding factor

References

- Christopher Kruguel and Giovanni Vigna. *Anomaly Detection of Web-based Attacks*, October 2003
- Detection of Web Application Attacks, http://www.securityfocus.com/infocus/1768
- SQL Signatures Evasion http://www.imperva.com/application_defense_center/white_papers/sql_injection_signatures_evasion.html
- Mod_security www.modsecurity.org

Questions?

www.nii.co.in