Modeling and Mitigation of XPath Injection Attacks for Web Services using Modular Neural Networks

Gajendra Deshpande, KLS Gogte Institute of Technology, Belagavi
Dr. Shrirang Kulkarni, National Institute of Engineering, Mysuru

Introduction

- Cyber Space is a national asset
- XML is a heart of many mainstream technologies, Web Services, Service Oriented Architecture(SOA), Cloud Computing etc.
- Web Services vulnerabilities can be present in Operating System, Network, Database, Web Server, Application Server, Application code, XML parsers and XML appliances
- New technologies – New Challenges ➔ (Old challenges + New Challenges)

Problem Definition

- To secure web resources from XPath injection attack using modular recurrent neural networks.

Proposed Solution

- The proposed solution uses modular recurrent neural network architecture to identify and classify atypical behavior in user input. Once the atypical user input is identified, the attacker is redirected to sham resources to protect the critical data.
- Count based validation technique

An attacker can craft special user-controllable input consisting of XPath expressions to inject the XML database and bypass authentication or glean information that he normally would not be able to.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
    <user>
        <username>gandalf</username>
        <password>!c3</password>
         <account>admin</account>
   </user>
</users>
```

```
string(//user[username/text()='gandalf' and
password/text()='!c3']/account/text())
```

```
string(//user[username/text()='' or '1' = '1' and
password/text()='' or '1' = '1']/account/text())
```

System Design

Some valid inputs:
Email-id
Mobile number
Alphanumeric word

Some malicious inputs:
'1 or 1=1
user' or 'a'='a
%00

Some invalid inputs:
Very large input string
String with special characters
String formed from different character set

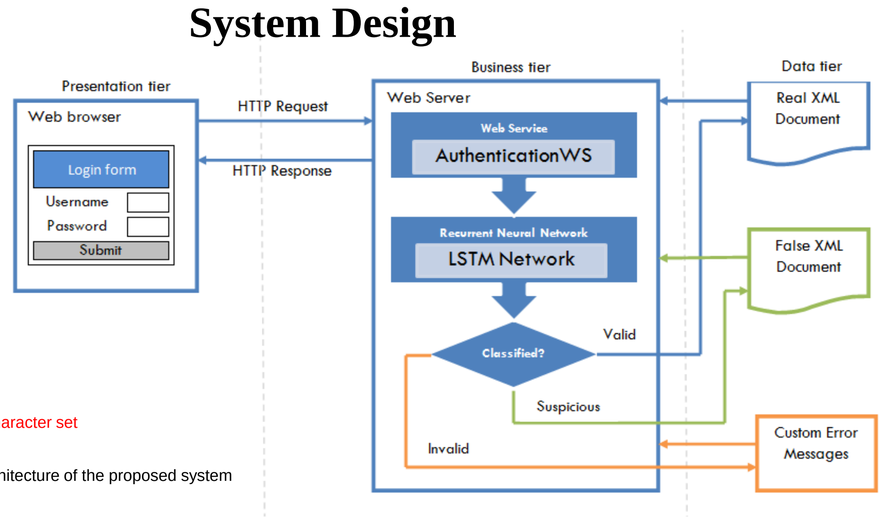


Fig. 1: Three tier architecture of the proposed system

Algorithm

1. Scan the user input.
2. Determine the length of user input.
3. Count the frequency of every character in the user input [a-z, A-Z, 0-9, ' " . @ # % + = ? :].
4. If the frequency of character is below the threshold value set for that particular character in Table 4 then set the error code to 40.
5. Else if the frequency of characters [. @ # % + = ' "] is above the threshold value set for that particular character in Table 4 then set the error code to 4000.
6. Else set the error code to 400.
7. Build a recurrent neural network 1 consisting of 50 neurons with hidden layer as LSTM network and output layer as SoftMax.
8. Use Rprop- trainer to train the network using the training dataset created using error codes in Table 2.
9. Use the test dataset created in real time to validate against the training dataset.
10. Build a recurrent neural network 2 consisting of 50 neurons with hidden layer as LSTM network and output layer as SoftMax.
11. Use Rprop- trainer to train the network using the training dataset created using number of login attempts in Table 1.
12. Use the test dataset created in real time to validate against the training dataset.
13. If train error and test error of both the networks are 0.0% then
    1. Finally classify the input vector based on the outputs of both the neural networks in Table 3.
    2. If the user input is successfully classified as 'valid' and found in the real XML file then Return the message "login successful".
    3. Else if the user input is classified as 'malicious' then Return the contents of the fake XML file.
    4. Else if the user input is classified as 'invalid' then Return the 'error' message.
14. Else repeat the steps 8 through 13.

Table 3. Final classification of input vector

Output of Neural Network 1	Output of Neural Network 2	Final Classification
Valid	Valid	Valid
Valid	Malicious	Malicious
Malicious	Valid	Malicious
Invalid	Valid	Invalid
Valid	Invalid	Invalid
Invalid	Malicious	Malicious
Malicious	Invalid	Malicious
Malicious	Malicious	Malicious

Table 1. Training dataset for classification of login attempts (Neural network 1)

Number of login attempts	Class
1	Valid
2	Valid
3	Valid
4 or more	Malicious

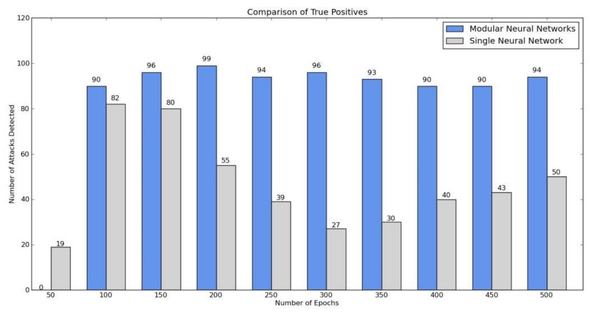
Table 2. Training dataset for classification of error codes (Neural network 2)

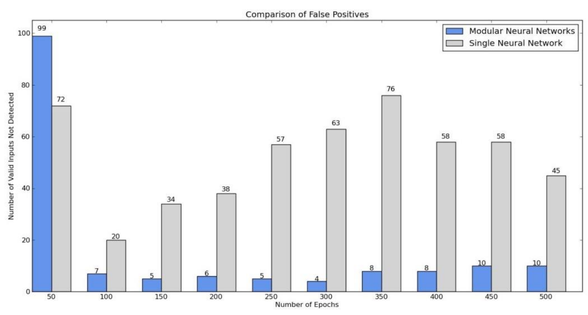
Error code	Class
40	Valid
400	Invalid
4000	Malicious

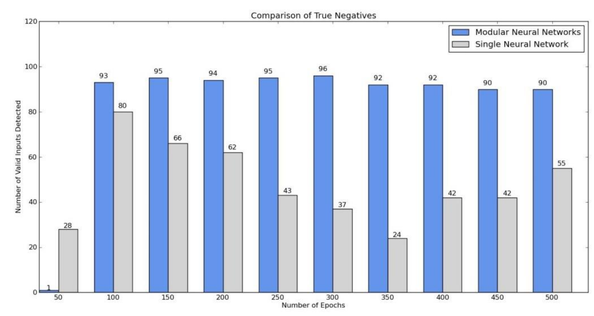
Table 4. Characters with threshold value

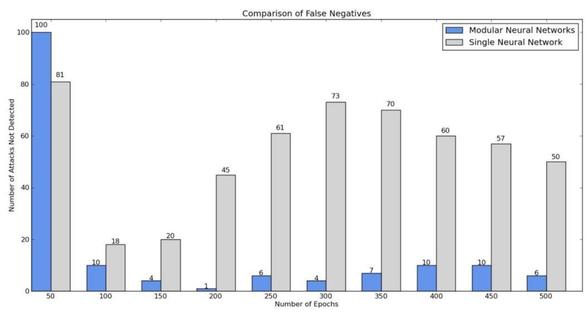
Special Character	Threshold	Error Code
Single quotes (')	1	40
Double quote (")	0	4000
Dot (.)	2	40
Alphabets ([a-zA-Z])	Any	40
Digits ([0-9])	Any	40
At the rate (@)	1	40
Equal to (=)	0	400
Square Brackets ([,])	0	400
Round Brackets ((,))	0	400
Curly Brackets ({,})	0	400
Slashes (\, /)	0	400
Asterisk (*)	0	400
Pipe (|)	0	400
Any other character	0	400

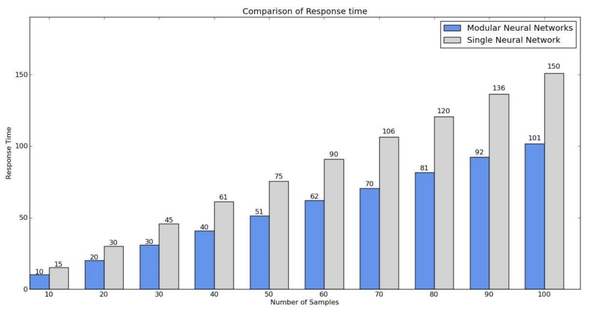
Results











	Average detection rate including an outlier		Average detection rate excluding an outlier	
	MNN %	SNN %	MNN %	SNN %
True Positives	84.2	46.5	93.55	51.66
False Negatives	15.8	53.5	6.45	48.33
True Negatives	83.8	47.9	93.11	53.22
False Positives	16.2	52.1	6.88	46.77

Conclusions

- Our solution offers improved security over existing methods by misleading the attackers to false resources and custom error pages
- Our results also show that the system accepts legitimate input although the user input may contain some special characters and rejects only truly malicious inputs.
- Our solution combines modular neural networks and count based validation approach to filter the malicious input
- Our solution has resulted in increased average detection rate of true positives and true negatives and decreased average detection rate of false positives and false negatives
- The security systems have to be successful every time. But attacker has to be successful only once.

References

[1] **Thiago Mattos Rosa, Altair Olivo Santin, Andreia Malucelli, "Mitigating XML Injection Attack through Strategy based Detection System", IEEE Security and Privacy, 2011**
[2] Nuno Antunes, Nuno Laranjeiro, Marco Vieira, Henrique Madeira, "Effective Detection of SQL/XPath Injection Vulnerabilities in Web Services", IEEE International Conference on Services Computing, 2009
[3]Nuno Laranjeiro, Marco Vieira, Henrique Madeira, "A Learning Based Approach to Secure Web Services from SQL/XPath InjectionAttacks", Pacific Rim International Symposium on Dependable Computing, 2010
[4] V. Shanmughaneethi, R. Ravichandran, S. Swamynathan, "PXpathV: Preventing XPath Injection Vulnerabilities in Web Applications", International Journal on Web Service Computing, Vol.2, No.3, September 2011
[5] CAPEC-83: XPath Injection, http://capec.mitre.org/data/definitions/83.html
[6] Mike W. Shields, Matthew C. Casey, "A theoretical framework for multiple neural network systems", 2008
[7] Hanh H. NguyenÆ Christine W. Chan, "Multiple neural networks for a long term time series forecast", Springer, Neural Comput & Applic (2004) 13: 90–98
[8] Anand, R., Mehrotra, K., Mohan C.K., Ranka S., "Efficient classification for multiclass problems using modular neural networks", IEEE Transactions on Neural Networks, Volume 6, Issue 1, 1995
[9] S. Hochreiter and J. Schmidhuber. "Long short-term memory. Neural Computation", 9 (8): 1735–1780, 1997.
[10] Derek D. Monner, James A. Reggia, "A generalized LSTM-like training algorithm for second-order recurrent neural networks"
[11] Anders Jacobsson, Christian Gustavsson, "Prediction of the Number of Residue Contacts in Proteins Using LSTM Neural Networks", Technical report, IDE0301, January 2003
[12] P.A. Mastorocostas, "Resilient back propagation learning algorithm for recurrent fuzzy neural networks", ELECTRONICS LETTERS, Vol. 40 No. 1, 2004
[13] Martin Riedmiller, Rprop – Description and Implementation Details, Technical report, 1994
[14] Tom Schaul, Justin Bayer, Daan Wierstra, Sun Yi, Martin Felder, Frank Sehnke, Thomas Rückstieß, Jürgen Schmidhuber. "PyBrain", Journal of Machine Learning Research, 2010
[15] Bottle: Python Web Framework, http://bottlepy.org/docs/dev/
[16] matplotlib, http://matplotlib.org/contents.html