NAVEX: Precise and Scalable Exploit Generation for Dynamic Web Applications

Abeer Alhuzali, Rigel Gjomemo, Birhanu Eshete, and V.N. Venkatakrishnan

University of Illinois at Chicago

Web Applications

- **Common Characteristics**
  - Content generated on the fly to improve usability and responsiveness
  - Tasks require a series of steps to accomplish
    - e.g., online shopping: view → select → add to cart → checkout
  - Dependencies among them
- **However**
    - **Increase** application **complexity**
    - **Increase analysis difficulty**

Web Application Example

view.php → addToCart.php → checkout.php

- **How to Exploit?**
  - Find a vulnerability
  - Craft an exploit string for that vulnerability
  - Find a navigation path to the vulnerability
    - e.g.: http…view.php → http…cart.php → http..checkout.php
- **Exploit is:**

1. http://localhost../view.php?item_quant=3&item_name=book
2. http://localhost../addToCart.php?type=order
3. http://localhost.../checkout.php?delivery_desc=nothing'; DROP table TB- -&submit=yes

Problem & Challenges

- **Problem: How to automatically construct exploits for large and complex web application?**
- **Challenge #1: Scalability:**
    - Large code base consisting of hundreds of modules with large number of intra-module execution paths
- **Challenge #2: Sinks reachability:**
    - Have to derive inputs that reach ‘deep sinks’
    - Exploit input has to
        - navigate through the complex dependencies among modules
        - satisfy module and path constraints

Challenges

- **Challenge #3: Dynamic features of web applications**
  - dynamically generated content may drive the navigation of the application to vulnerable sinks
    - Forms, links, JavaScript content
- **Challenge #4: handling multiple vulnerability classes**
  - e.g., injection vulnerabilities (SQLI, XSS, etc.) and logic vulnerabilities (e.g., EAR)
  - minimal changes to the analysis

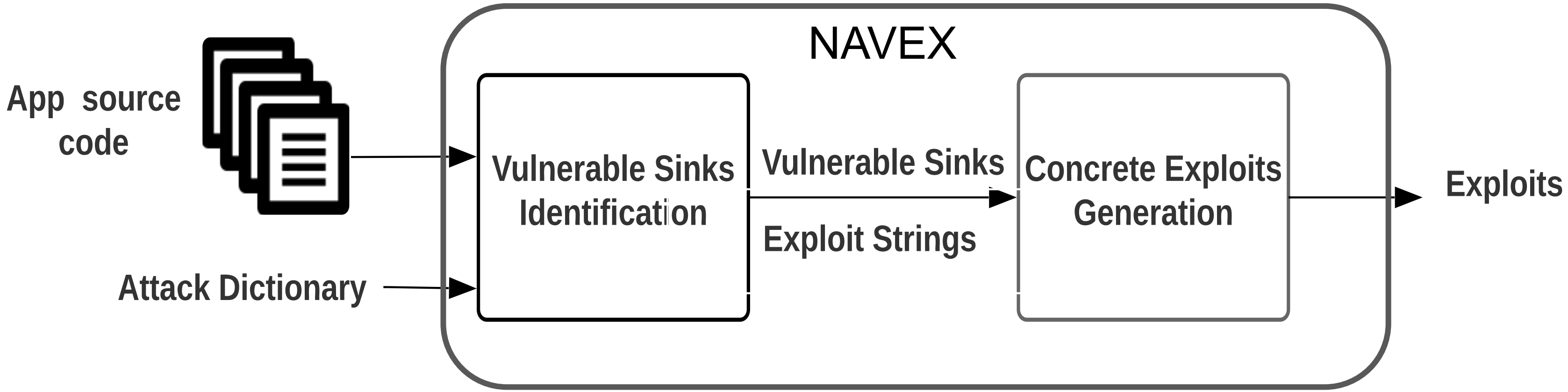
➡ **Goal: Automatic exploit generation approach that addresses these challenges**

Our Main Contribution: NAVEX, a system that has identified over two hundred exploits in modern PHP web applications

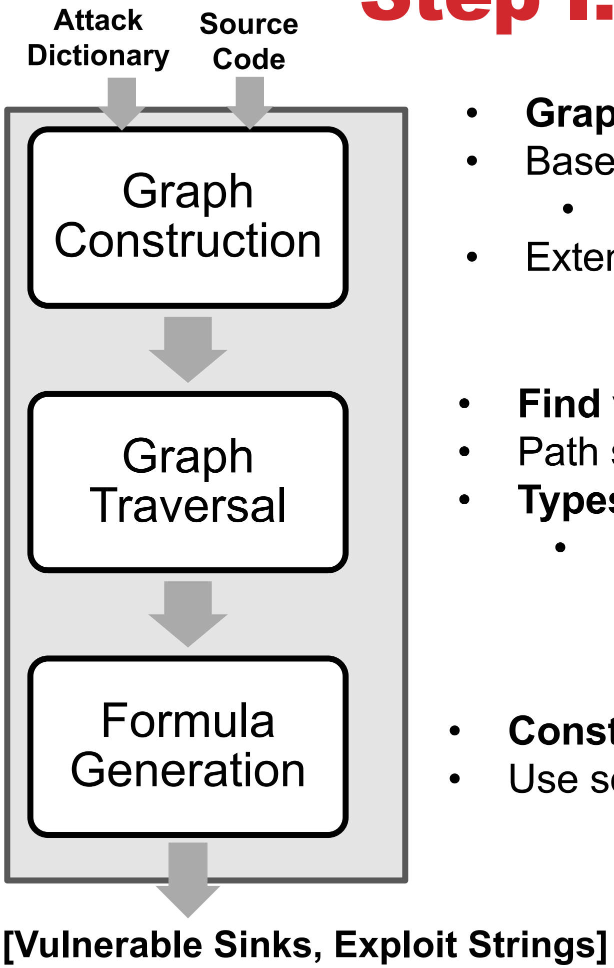
Approach Overview

- Find vulnerable sinks using static analysis methods
- Build a graph representation of navigation structure of applications dynamically
- Find navigation paths to the identified vulnerabilities
- Final exploit construction

NAVEX Architecture



Step I: Vulnerable Sinks Identification

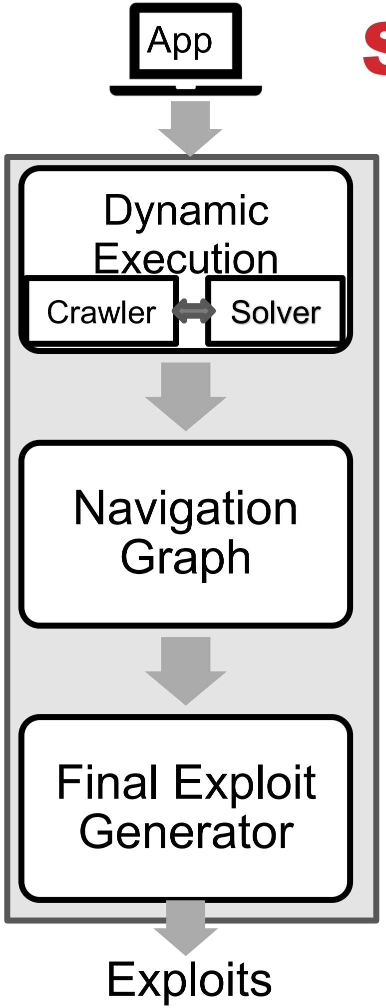


- **Graph model of source code**
- Based on Code Property Graphs (CPGs)
  - CPG = AST+CFG+ call graph+DDG
- Extend CPGs with ***sanitization* and *database constraints* tags**

- **Find vulnerable paths to sensitive sinks**
- Path sensitive analysis
- **Types: Forward** and **backward** traversals based on vulnerability type
  - E.g., **backward** search for **injection vulnerabilities**

- **Construct formulas** from **vulnerable path statements**
- Use solver to generate **exploit strings**

Step II: Concrete Exploits Generation

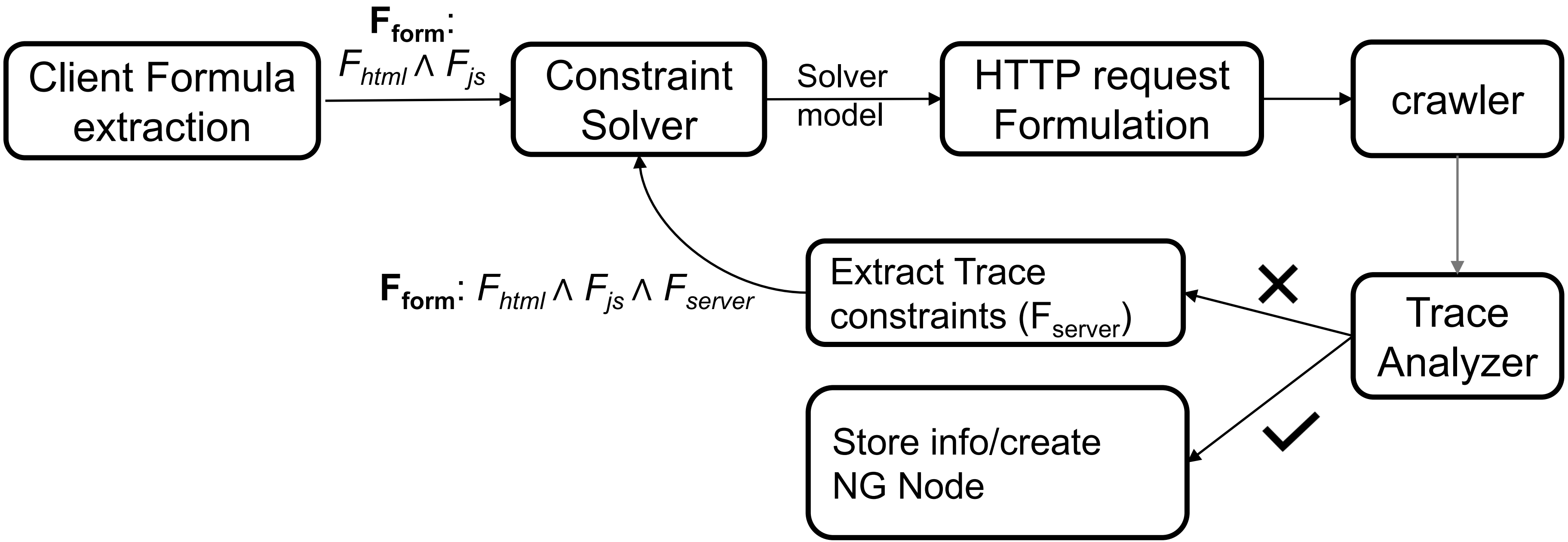


- **Links:** stored and used as new URLs to crawl
- **Forms:** Generate **form inputs automatically**
  - Extract constraints from forms
- **JavaScript :** concolic execution based on NoTamper (Bisht et al.,CCS'10)

- An **application-wide navigation graph**
  - represents possible sequences of module executions
- Directed graph
  - ***node***: HTTP request
  - **edge:** navigation between nodes (type is *link* or *form)*

- Search the NG to **find navigation paths to vulnerable sinks**

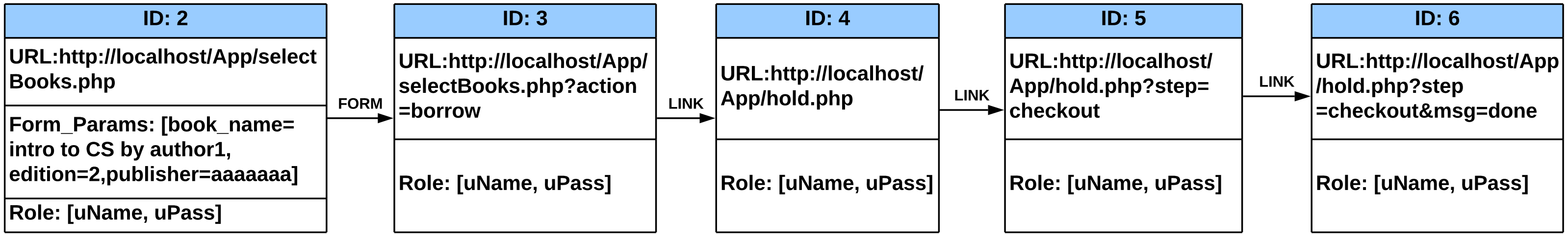
Input Generation



Combining Static & Dynamic Results

- **Example:**
  - vulnerability in **PathToApp/App/checkout.php, checkout.php is included** by **hold.php** (no direct access)
  - Navigation Graph: **no node of a URL = "….checkout.php"**
- **Problem:** combining the results produced by the step of vulnerable sink identification (static analysis) with the Navigation Graph (dynamically generated).
- **Solution: Inclusion Map**
  - Constructed statically, [Parent file -> included files]

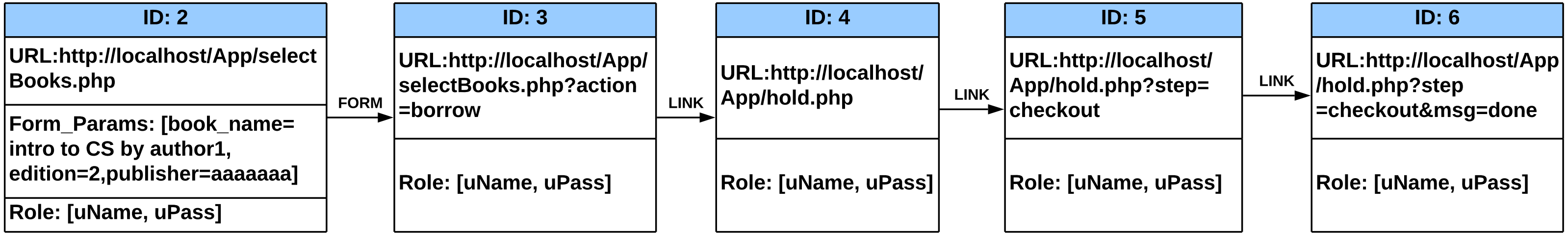
Searching Navigation Graph



- **Input :**
  - **vulnerable sink (destination URL)** = http://localhost/App/hold.php
  - **exploit string** is msg =<script>alert(”XSS”);</script> (GET)
  - **Public URL (source URL)** = http://localhost/App/selectBooks.php

- **Search Results:**
  - nodes of [id=2, id=3, id=4, id=5, id=6]
    - http://localhost/App/hold.php?step=checkout&**msg=done**
    - http://localhost/App/hold.php?step=checkout&**msg=<script>alert(”XSS”);</script>**

Final Exploit



1. **http://localhost/App/index.php**
2. **http://localhost/App/selectBooks.php**

   POST params:[book name=intro to CS by author1, edition=2,publisher=aaaaaaa]
3. **http://localhost/App/selectBooks.php?action=borrow**
4. **http://localhost/App/hold.php**
5. **http://localhost/App/hold.php?step=checkout**
6. **http://localhost/App/hold.php?step=checkout&msg=<script>alert(”XSS”);</script>**

EVALUATION

Dataset

- **26 real-world** open-source **PHP** web applications
- Total of **3.2M SLOC** and **22K PHP files**
- Applications selection criteria
  - **Popular and large** PHP apps
    - Such as **WordPress**, **OpenConf, HotCRP, Drupal, Gallery, Joomla, LimeSurvey, Collabtive**, and **MediaWiki**
  - **Comparison** with **state-of-the-art work** in exploit generation (e.g., Chainsaw (Alhuzali et al., CCS'16)) and vulnerability analysis (e.g., RIPS (Dahse and Holz, NDSS'14))

Results Summary

- NAVEX constructed a total of **204 exploits**
  - **195** are on **injection vulnerabilities (SQLI and XSS)**.
  - **9** are on **logic vulnerabilities (EAR).**
- The **enhanced CPG reduced FPs by 87%** on average.
- **Client-side code analysis** for building the navigation graph **enhanced the precision of exploit generation by 54%** on average.
- Drill down **as deep as 6 HTTP requests** to stitch together **exploits.**

SQLI Exploits

- Reported **155 SQLI exploitable sinks**
- **No false positives**
- Constructed **105 concrete SQLI exploits**
- **Vulnerable** web apps
    - **osCommerce (2.3.3)**
    - **phpBB (2.0.23)**
    - **myBloggie, Scarf, Dnscript, WeBid, Eve, SchoolMate, geccbblite, FAQforge, and WebChess**

XSS Exploits

- Found **133 XSS exploitable sinks**
- 5 false positives
- Generated **90 XSS exploits**
- **Vulnerable** web apps
  - **HotCRP (2.60)**
  - **osCommerce (2.3.4)**
  - **osCommerce (2.3.3)**
  - **CPG**
  - **MediaWiki**
  - **phpBB (2.0.23)**
  - **myBloggie, Scarf, Dnscript, WeBid, Eve, SchoolMate, FAQforge, and WebChess**

EAR EXPLOITS (LOGIC EXPLOITS)

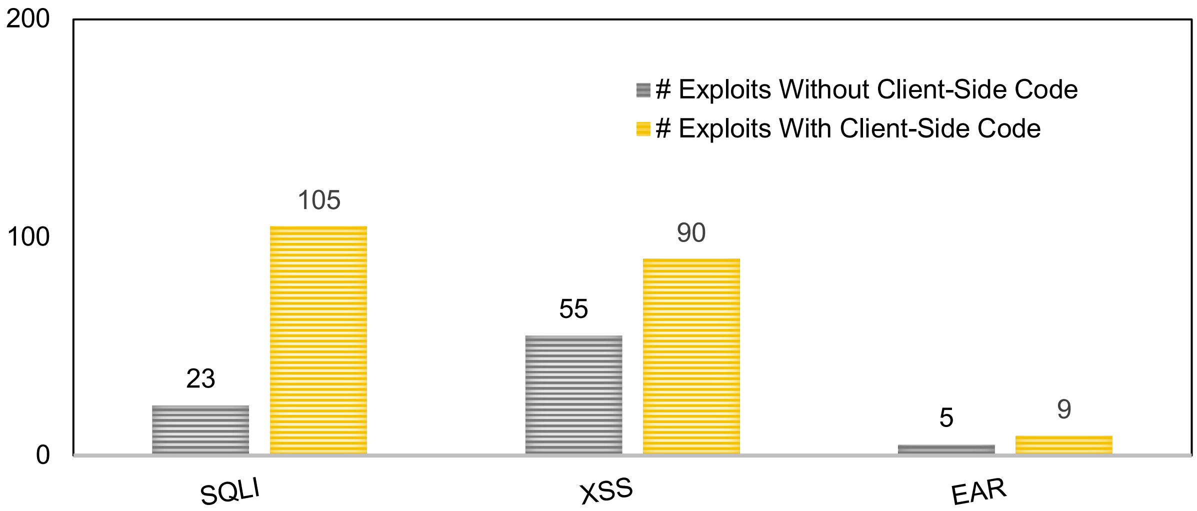
- Found **22 EAR** vulnerabilities
- 3 false positives
- Generated **9 EAR exploits**
- **Vulnerable** web apps
  - **HotCRP (2.100)**
  - **HotCRP (2.60)**
  - **OpenConf**
  - **osCommerce (2.3.4)**
  - **osCommerce (2.3.3)**
  - **LimeSurvey**
  - **Collabtive**
  - **MediaWiki**
  - **myBloggie, WeBid**, and **Eve**

Performance and Scalability

- **Performance**: **total time** to **find exploitable sinks** and to **generate exploits** per vulnerability type.
  - **Vulnerability identification** from **17.28** to **109.27** minutes.
  - **Exploit generation** from **1.38** to **40.20** minutes.

Effect of Client-side Code Analysis

- **Forms are common**
  - **Number of unique forms** ranges from **3 to 186** (average of 45 form/application).
- **Input generation and constraints extraction from client-side code → improve the crawling coverage.**
- **NAVEX constructed more exploits.**



Conclusion

- NAVEX is an automatic exploit generation system that considers
  - **dynamic features** and **the navigational complexities** of modern web applications
- NAVEX constructed **204 exploits**
  - **195** are on **injection vulnerabilities**
  - **9** are on **logic vulnerabilities**
- **Outperform prior work** on the precision, efficiency, and scalability of exploit generation.

THANK YOU FOR YOUR ATTENTION
QUESTIONS?

NAVEX is available at
https://github.com/aalhuz/navex

GitHub

REFERENCES

1. G. Wassermann, D. Yu, A. Chander, D. Dhurjati, H. Inamura, and Z. Su. Dynamic test input generation for web applications. In Proceedings of the 2008 international symposium on Software testing and analysis, pages 249–260, 2008.
2. M. Martin and M. S. Lam. Automatic generation of xss and sql injection attacks with goal-directed model checking. In Proceedings of the 17th conference on Security symposium, pages 31–43, 2008.
3. J. Dahse and T. Holz. Simulation of Built-in PHP Features for Precise Static Code Analysis. In Symposium on Network and Distributed System Security (NDSS), 2014.
4. J. Dahse and T. Holz. Static Detection of Second-Order Vulnerabilities in Web Applications. In 23rd USENIX Security Symposium (USENIX Security 14), pages 989–1003, 2014.
5. B. Eshete, A. Alhuzali, M. Monshizadeh, P. A. Porras, V. N. Venkatakrishnan, and V. Yegneswaran. EKHunter: A Counter-Offensive Toolkit for Exploit Kit Infiltration. In 22nd Annual Network and Distributed System Security Symposium, NDSS, 2015.
6. S. Huang, H. Lu, W. Leong, and H. Liu. CRAXweb: Automatic Web Application Testing and Attack Generation. In IEEE 7th International Conference on Software Security and Reliability, SERE, pages 208–217, 2013.
7. A. Kieyzun, P. J. Guo, K. Jayaraman, and M. D. Ernst. Automatic Creation of SQL Injection and Cross-Site Scripting Attacks. In IEEE 31st International Conference on Software Engineering (ICSE), pages 199–209, 2009.
8. G. Wassermann and Z. Su. Sound and precise analysis of web applications for injection vulnerabilities. In ACM Sigplan Notices, volume 42, pages 32–41. ACM, 2007.
9. D. Balzarotti, M. Cova, V. V. Felmetsger, and G. Vigna. Multi-module Vulnerability Analysis of Web-based Applications. In the 14th ACM Conference on Computer and Communications Security (CCS), pages 25–35, 2007.
10. D. Brumley, P. Poosankam, D. Song, and J. Zheng. Automatic Patch-Based Exploit Generation is Possible: Techniques and Implications. In Proceedings of the 2008 IEEE Symposium on Security and Privacy, 2008.
11. T. Avgerinos, S. K. Cha, B. L. T. Hao, and D. Brumley. AEG: Automatic Exploit Generation. In NDSS, volume 11, pages 59–66, 2011.
12. H. Hu, Z. L. Chua, S. Adrian, P. Saxena, and Z. Liang. Automatic Generation of Data-Oriented Exploits. In 24th USENIX Security Symposium (USENIX Security 15), pages 177–192. USENIX Association, 2015.
13. Yamaguchi, F., Golde, N., Arp, D., & Rieck, K. (2014, May). Modeling and discovering vulnerabilities with code property graphs. In *Security and Privacy (SP), 2014 IEEE Symposium on* (pp. 590-604). IEEE.
14. Backes, M., Rieck, K., Skoruppa, M., Stock, B., & Yamaguchi, F. (2017, April). Efficient and Flexible Discovery of PHP Application Vulnerabilities. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on* (pp. 334-349). IEEE.
15. Alhuzali, A., Eshete, B., Gjomemo, R., & Venkatakrishnan, V. N. (2016, October). Chainsaw: Chained automated workflow-based exploit generation. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (pp. 641-652). ACM.
16. Bisht, P., Hinrichs, T., Skrupsky, N., Bobrowicz, R., & Venkatakrishnan, V. N. (2010, October). NoTamper: automatic blackbox detection of parameter tampering opportunities in web applications. In *Proceedings of the 17th ACM conference on Computer and communications security* (pp. 607-618). ACM.