EDITOR **JEFFREY VOAS**
NIST; j.voas@ieee.org

Web Application Security: Threats, Countermeasures, and Pitfalls

Hsiu-Chuan Huang, National Chiao Tung University and Chunghwa Telecom Laboratories

Zhi-Kai Zhang, Hao-Wen Cheng, and Shiuhpyng Winston Shieh, National Chiao Tung University

Penetration testing is a crucial defense against common web application security threats such as SQL injection and cross-site scripting attacks. A proposed web vulnerability scanner automatically generates test data with combinative evasion techniques, significantly expanding test coverage and revealing more vulnerabilities.

The prevalence of the World Wide Web makes websites and their visitors attractive targets for various types of cybercrime including data breaches, spearphishing campaigns, ransomware, and fake technical support scams. According to Symantec's most recent *Internet Security Threat Report*, more than 229,000 attacks against websites occur each day, and more than 76 percent of websites contain unpatched vulnerabilities.[1] Web attack toolkits such as RIG and Sundown make it easier than ever for cybercriminals to exploit such vulnerabilities.

Among the most critical security threats to today's websites, which are dynamic, interactive, and collaborative, are injection and cross-site scripting (XSS) attacks.[2] An *injection attack* occurs when an adversary sends data to an improperly coded application that tricks it into executing unintended commands or queries specified by the adversary. A root cause of unauthorized data access is SQL injection, which involves sending malicious data to alter SQL queries executed by web application databases. Other common injection attacks modify Lightweight Directory Access Protocol statements and OS commands. An *XSS attack* exploits improperly coded web applications to send malicious scripts to users' browsers for execution. Adversaries can leverage XSS attacks to hijack user sessions, deface websites, or redirect victims

to malicious sites. In the third quarter of 2016, SQL injection and XSS attacks accounted for 55 percent of all web application attacks.[3] Therefore, understanding these attacks and developing possible countermeasures are vital.

In this article, we discuss several proposed countermeasures and their pitfalls. We then introduce VulScan, a new web vulnerability scanner that uses penetration testing and combinative evasion techniques to evade firewalls and filters and to discover injection as well as XSS vulnerabilities in target systems, thereby improving web application security.

SQL INJECTION AND XSS COUNTERMEASURES

SQL injection and XSS countermeasures fall into three main categories: secure implementation, defense mechanism deployment, and penetration testing. Most of the top 10 flaws identified by the Open Web Application Security Project (OWASP; www.owasp.org) can be either eliminated or mitigated in practice using techniques such as adopting proper access control, avoiding unsafe APIs, input validation or sanitization, output encoding or escaping, applying strong cryptography and security protocols, and following the principle of least privilege.

Secure implementation

Parameterized queries and input validation and sanitization can help prevent SQL injection. Parameterized queries—prepared statements with variable binding—enable a database management system to correctly differentiate queries and data and prevent it from executing unintended operations. Input validation or sanitization validate user-supplied data to filter or escape special characters that might alter the queries' intent. On the other hand, data encryption and following the least privilege principle can reduce the impact of successful SQL injection attacks.

Techniques for preventing XSS attacks include input validation or sanitization, output encoding or escaping, and use of a content security policy.[4,5] Although output encoding or escaping built-in visitors' browsers can partially prevent user-supplied data from being executed, web developers should still apply input validation or sanitization and output encoding or escaping to prevent XSS content from being stored and reflected.

Defense mechanism deployment

Another countermeasure is to deploy defense mechanisms, such as web application firewalls (WAFs), outside the web application servers. WAFs analyze all inbound traffic, detect attacks using policy rules or expressions, and block detected attacks. Deploying WAFs is cost-effective because it avoids the need for web developers, most of whom aren't security experts, to perform exhaustive code review or rewrite all vulnerable code. As with all rule-based defense mechanisms, the rule-set coverage dictates the strength of security protection.

Penetration testing

Penetration testing, also called white-hat evaluation, can help web developers discover and locate system vulnerabilities. Manual penetration tests, which require the participation of security experts and domain experts, are time-consuming and usually only applied to critical applications. Automated penetration testing tools are available but provide less extensive test-data coverage, which determines both false-positive and false-negative rates.

COUNTERMEASURE PITFALLS

Despite these countermeasures, SQL injection and XSS attacks remain major threats to web application security.

With respect to secure implementation, it's difficult if not impossible to write vulnerability-free code. For example, using parameterized queries that separate query instructions from user input can help prevent SQL injection attacks but can't address developer carelessness or the need in certain cases to concatenate user-supplied data to queries.

Defense mechanisms are also problematic because most filters and WAFs are signature or rule based; consequently, improperly coded filters and misconfigured WAFs might fail to block or sanitize malicious user input. Creating new defensive rule sets to counter new attack payloads is a never-ending race for security experts, who continually evaluate various evasion techniques to bypass filters and WAFs.[6–8] Because filters and WAF rules must be manually written, establishing a rule set with sufficient coverage remains a major challenge.

Penetration testing faces a similar dilemma, as most test tools use manually predefined data.[9] Automatic generation of effective test data is also a challenge.

VULSCAN

We developed a web vulnerability scanner, VulScan, that automatically generates test data using a combination of evasion techniques that effectively bypass filters and WAFs to reveal SQL injection and XSS vulnerabilities.

Combinative evasion techniques

Conventional web vulnerability scanners, such as OWASP's Zed Attack Proxy (ZAP),[9] rely on manually predefined evasion techniques and thus have inherently limited test-data coverage. In contrast, VulScan automatically generates combinative evasion techniques, thereby significantly expanding test-data coverage.

Consider `<script>alert('XSS')</script>`, which is often used in XSS penetration tests. Most filters and WAFs deal with the `<script>` tag well, but evasion techniques can mutate this kind of basic test data to bypass defense mechanisms. For example, changing `<script>` to `<ScRiPt>` could evade those filters and WAFs that only check the lowercase word `<script>`. Similarly, changing `<script>` to `<scr<script>ipt>`

can evade the filters and WAFs that only remove <script> once. Combining the capitalization and nonrecursive evasion techniques could mutate <script> to <ScR<ScRiPt>iPt>. Manually writing rules against such combinative evasion techniques is obviously much harder. Therefore, automatically generating the combinations is desirable.

System components and operation

As Figure 1 shows, VulScan has four main components: the *web crawler* analyzes the target website, gathers system information, and extracts the entry points for additional tests; the *scanner knowledge base* stores the unmutated basic test data; the *evasion techniques knowledge base* stores various evasion techniques that will be used to mutate the test data; and the *scanner engine* connects all modules and generates reports.

Besides test coverage, test efficiency is a key goal. Automatically generating all combinations of evasion techniques expands test coverage but also exhausts computational power and network bandwidth. Thus, pruning unneeded combinations is important.

To determine such combinations, VulScan first gathers system information from the target. Each evasion technique has its own property file indicating the environment it can apply. Because some evasion techniques can be applied only to certain target types, a combination of these techniques can be pruned if it mismatches a target type.

To further prune unneeded combinations, two relations between evasion techniques are defined as follows:

- *Conflict relation*. Evasion techniques can't be combined in sequences—for example, a technique that replaces a whitespace with an inline comment along with a technique that replaces a whitespace with a tab. Also, evasion techniques only for MySQL can't be combined with those only for MSSQL.
- *Order relation*. An order relation exists between evasion techniques if they can be combined only in a particular sequence. For example, if there's a technique that transforms an attack pattern into a form that another technique can't obfuscate, obfuscation must be applied before transformation.

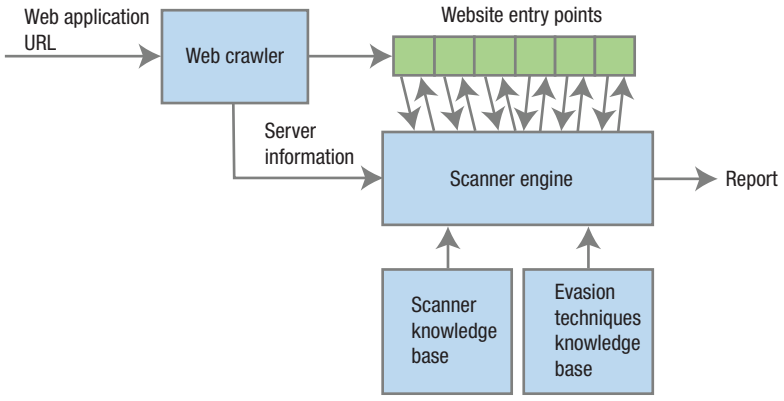

Figure 1. VulScan architecture. The system relies on four main components (blue boxes) to increase both test coverage and test efficiency.

```
01: Algorithm combineEvasionTechniques is
02:   INPUT:            info       - system information of the target website gathered by Web Crawler
03:                     max_et     - maximum number of evasion techniques for each test
04:                     type       - "SQL Injection" or "XSS"
05:                     ETKB       - Evasion Techniques Knowledge Base contains basic evasion techniques
06:   OUTPUT:           CET        - set of Combinative Evasion Techniques for mutating the test data
07:
08:   CET ← {}                                          // initialization
09:   CET_CURRENT← {}                                   // initialization
10:   level ← 1                                         // initialization
11:   for each t in ETKB do                             // t - an evasion technique
12:     if (t matches info and type)                    // if it's applicable to target website or not
13:       CET ← CET ∪ {t}                               // add the applicable evasion technique
14:   CET_LAST_LVL ← CET                                // keep current CET for the next level
15:   while (level < max_et) do                         // loops for combining evasion techniques
16:     for each tc in CET_LAST_LVL do                  // tc - a combinative evasion technique
17:       for each t in ETKB do                         // t - an evasion technique
18:         if(t matches info and type)&& \             // whether current t is compatible with tc or not
19:           ((tc,t) does not conflict)&& \            // check for Conflict Relation and Order Relation
20:           ((tc,t) obeys order Relation)
21:           CET_CURRENT ← CET_CURRENT ∪ {(tc,t)}      // add (tc,t) to applicable list
22:     CET ← CET ∪ CET_CURRENT                         // add combinative evasion techniques to output
23:     CET_LAST_LVL ← CET_CURRENT                      // prepare for the next level
24:     CET_CURRENT ← {}                                // prepare for the next level
25:     level ← level+1                                 // prepare for the next level
26:   return CET                                        // output combinative evasion techniques
```

Figure 2. VulScan's algorithm for generating combinative evasion techniques.

Algorithm generation

Figure 2 shows the algorithm VulScan uses to automatically generate combinative evasion techniques for mutating test data. The algorithm only selects techniques that match the target system and scan type, don't violate the conflict relation, and obey the order relation. The input are `info` (target system information), `max_et` (maximum number of evasion techniques), `type` (scan type), and `ETKB` (evasion techniques knowledge base). The algorithm will fetch at most `max_et` evasion techniques from `ETKB`, check if they meet the aforementioned requirements, combine those techniques meeting all requirements, and output a combinative evasion technique.

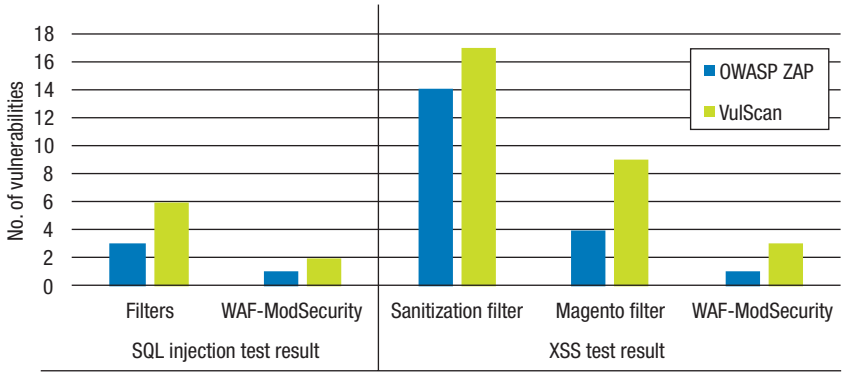

Figure 3. System test results. VulScan detected more SQL injection and cross-site scripting (XSS) vulnerabilities than the Open Web Application Security Project's (OWASP's) Zed Attack Proxy (ZAP), a conventional web vulnerability scanner. WAF: web application firewall.

SYSTEM EVALUATION

To evaluate VulScan's accuracy, we chose OWASP's WebGoat[10] and a set of real web applications as the subjects under test. We compared the total number of vulnerabilities detected by VulScan with the number detected by ZAP.

To test for SQL injection vulnerabilities, we used VulScan against two defense mechanisms. The first implements three common filters—whitespace, case-sensitive, and nonrecursive—to encode, remove, or block malicious payloads. The second mechanism implements OWASP's ModSecurity, a popular open source WAF that enforces the OWASP 2.2 Core Rule. As the left side of Figure 3 shows, VulScan bypassed the filter and WAF, detecting more flaws than ZAP. In one case, VulScan generated a test payload not covered by ZAP by combining the capitalization and SQL comment evasion techniques, illustrating VulScan's expanded test coverage.

To test for XSS vulnerabilities, we implemented a built-in PHP sanitization filter and a Magento filter to protect the target websites; again we used ModSecurity as the WAF. As the right side of Figure 3 shows, VulScan also found more XSS flaws than ZAP did.

As for VulScan's efficiency, the pruning mechanism is very effective. In this instance, it reduced the number of combinations of two evasion techniques used for SQL injection and XSS testing by 68.7 and 71.8 percent, respectively.

There's no silver bullet for web application security; threats will continue to grow and evolve. Along with secure implementation and defense mechanism deployment, penetration testing remains a crucial, yet imperfect, countermeasure. Automatically generating test data with combinative evasion techniques is a promising means to significantly expand test coverage.

In a small field test, our proposed VulScan system successfully discovered privacy-leaking SQL injection and XSS vulnerabilities in some real websites protected by filters and WAFs that could not be detected by a popular web vulnerability scanner. After receiving our report, the sites' system administrators corrected these flaws.

In future work, we hope to limit the burden on computational power and network bandwidth caused by expanded test coverage by prioritizing test combinations with the best chances to discover vulnerabilities. This could be as hard as making predictions, but a feedback- or statistics-based mechanism is a feasible starting point.

ACKNOWLEDGMENTS

This work is supported in part by Taiwan's Ministry of Science and Technology, the National Security Council of Taiwan, the Telecom Technology Center, Taiwan's Ministry of Justice Investigation Bureau, Chunghwa Telecom, the Taiwan Information Security Center at National Chiao Tung University, and Taiwan's Ministry of Education.

REFERENCES

1. Symantec, *Internet Security Threat Report*, vol. 22, April 2017; www.symantec.com/security-center/threat-report.
2. Open Web Application Security Project, "OWASP Top 10—2013: The Ten Most Critical Web Application Security Risks," 2013; www.owasp.org/index.php/Top10#OWASP_Top_10_for_2013.
3. Akamai, *Akamai's [State of the Internet]/Security: Q3 2016 Report*, vol. 3, no. 3, 2016; www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q3-2016-state-of-the-internet-security-report.pdf.
4. M. West, A. Barth, and D. Veditz, eds., *Content Security Policy Level 2*, W3C recommendation, 15 Dec. 2016; www.w3.org/TR/CSP2.
5. I. Yusof and A.K. Pathan, "Mitigating Cross-Site Scripting Attacks with a Content Security Policy," *Computer*, vol. 49, no. 3, 2016, pp. 56–63.
6. Open Web Application Security Project, "SQL Injection Bypassing WAF," 2016; www.owasp.org/index.php/SQL_Injection_Bypassing_WAF.
7. Open Web Application Security Project, "XSS Filter Evasion Cheat Sheet," 2016; www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet.
8. A. Sadeghian, M. Zamani, and S. Ibrahim, "SQL Injection Is Still Alive: A Study on SQL Injection Signature Evasion Techniques," *Proc. Int'l Conf. Informatics and Creative Multimedia* (ICICM 13), 2013, pp. 265–268.
9. Open Web Application Security Project, "OWASP Zed Attack Proxy

Project," 2016; www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.
10. Open Web Application Security Project, "OWASP WebGoat Project," 2016; www.owasp.org/index.php/WebGoat.

HSIU-CHUAN HUANG is a PhD candidate in the Department of Computer Science at National Chiao Tung University (NCTU) and a security researcher at Chunghwa Telecom Laboratories. Contact her at pattyhuang.cs99g@g2.nctu.edu.tw.

ZHI-KAI ZHANG is a PhD candidate in the Department of Computer Science at NCTU. Contact him at skyzhang.cs99g@g2.nctu.edu.tw.

HAO-WEN CHENG received an MS in computer science from NCTU. Contact him at chris38c28@gmail.com.

SHIUHPYNG WINSTON SHIEH is a Distinguished Professor and past chair of the Department of Computer Science, as well as director of the Taiwan Information Security Center, at NCTU. Contact him at ssp@cs.nctu.edu.tw.