Combinatorial Methods in Security Testing

Dimitris E. Simos, SBA Research
Rick Kuhn, National Institute of Standards and Technology
Artemios G. Voyiatzis, SBA Research
Raghu Kacker, National Institute of Standards and Technology

Combinatorial methods can make software security testing much more efficient and effective than conventional approaches.

Many software security vulnerabilities result from the exploitation of ordinary coding flaws, rather than design or configuration errors. One study found that 64 percent of vulnerabilities are the result of such common bugs as missing or incorrect parameter checking, which leaves applications open to common vulnerabilities including buffer overflows or SQL injection.[1] Although this statistic might be discouraging, it also means that better functionality testing can also significantly improve security.

SECURITY TESTING

Testing that can reveal complex faults that occur only under rare conditions could be especially effective. Empirical data show that most failures are triggered by a single parameter value, or interactions between a small number of parameters (generally two to six)—a relationship known as the *interaction rule*.[2] An example of a single-value fault might be a buffer overflow that occurs when the length of an input string exceeds a particular limit. Only a single condition must be true to trigger the fault: input length > buffer size. A two-way fault is more complex, because two particular input values are needed to trigger the fault. One example is a search/replace function that only fails if both the search string *and* the replacement string are single characters. If one of the strings is longer than one character, the code doesn't fail; thus we refer to this as a two-way fault. More generally, a t-way fault involves t such conditions.

Figure 1 plots the cumulative percentage of faults at $t = 1–6$ for various software applications studied by the National Institute of Standards and Technology (NIST) and others.[3–6] As shown, the fault detection rate increases rapidly with interaction strength, up to $t = 4$, reaching 100 percent detection with four- to six-way interactions. Thus, the impossibility of exhaustive testing of all possible inputs isn't a barrier to high-assurance testing. That is, although we can't test all possible combinations of input values, failures involving more than six variables are extremely unlikely because they haven't been seen in practice, so testing all possible combinations provides very

EDITOR JEFFREY VOAS
National Institute of Standards and Technology;
j.voas@ieee.org

little benefit beyond testing four- to six-way combinations.

The effectiveness of any software testing technique depends on whether test settings corresponding to the actual faults are included in the test sets. When such settings aren't included, the faults won't be detected. Conversely, we can be confident that the software works correctly for *t*-way combinations contained in passing tests. For security evaluations, it isn't enough that failures are unlikely to occur in ordinary usage, because attackers seek out even complex flaws. Testing only to verify requirements coverage is insufficient for security, or even for assuring critical functionality.

Matrices known as *covering arrays* (CAs) can be computed to cover all *t*-way combinations of variable values, up to a specified level of *t* (typically $t \leq 6$), making it possible to efficiently test all such *t*-way interactions.[7] But any test set, whether constructed as a covering array or not, contains a large number of combinations. We can measure this combinatorial coverage—the coverage of *t*-way combinations in a test set—to better understand test set quality. These measurements contribute quantitative input for risk analysis, helping to answer questions such as: How many different scenarios have been checked? Are the untested scenarios important? How significant is the risk if we don't increase the test coverage? Does market share or other external forces (for example, conformance with standards) justify increasing the test coverage?

COMBINATORIAL SECURITY TESTING

SBA Research and NIST have developed a research program that aims to bridge the gap between combinatorial testing (CT) and security testing and, in the process, establish a new research field: *combinatorial security testing*. Several case studies illustrate our experiences thus far.

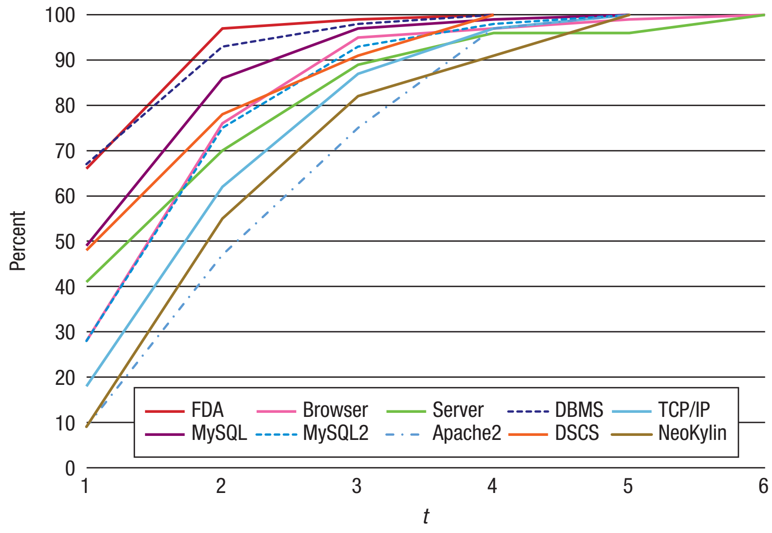

Figure 1. Cumulative fault distribution at *t* = 1–6 for various software applications. The fault detection rate increases rapidly with interaction strength, up to *t* = 4, reaching 100 percent detection with four- to six-way interactions.

Parsing untrusted Web content

The World Wide Web Consortium (W3C) tidy service (http://services.w3.org/tidy/tidy) is designed to detect and correct HTML code. It accepts a URL through a Web form, then parses its HTML source code and reports any fixes that should be made. The service has been online for many years now, being exposed to multiple instances of malformed code.

In coordination with the W3C, SBA Research performed an external Web application penetration test.[8] We first created an input parameter model (IPM) of the Document Object Model (DOM) and generated an attack grammar. The resulting IPM was an enhancement of our previous CT-based approaches for Web security testing.[9–11] We then used the NIST Automated Combinatorial Testing for Software (ACTS) tool to produce test cases that ensured 100 percent coverage of the DOM's two- and three-way parameter interactions, as recent studies demonstrated its effectiveness for Web security testing.[12] We tested the online service against the generated test suite using a prototype cross-site scripting (XSS) injection tool and succeeded in discovering a previously unknown remote XSS vulnerability of this popular service. We note that in this case the source code of the W3C tidy service wasn't available (black-box testing); only a Web interface was available to interact with it.

The sophistication of the attack vector produced by our CT technique might explain why the vulnerability had gone unnoticed for so long. The W3C promptly fixed the offending code and acknowledged SBA Research's effort.

Web application security

SBA Research demonstrated a second example use of automated Web penetration testing with the Koha integrated library management system (https://koha-community.org). Koha is used by various organizations, including

UNESCO, the Spanish Ministry of Culture, and the Vienna Cultural Museum. The source code of Koha is available under an open source license. SBA Research developed an IPM to test the API of this Web-based application. The IPM modeled the parameters passed back and forth in the HTTP requests encoded as URL parameters. We differentiated two groups of tests: one using a normal (nonprivileged user) account of the system, and one using an administrator (privileged user) account.

Combinatorial methods are ideally suited for the Internet of Things environment, where testing can involve a very large number of nodes and combinations.

We successfully modeled 43 URLs accepting between 5 and 15 parameters, each of various values. We used the NIST ACTS tool to produce test suites that fully covered all possible two-, three-, four-, and five-way parameter combinations. We carried out the testing experiments with the XSS injection tool we used to also test the W3C online tidy service.

We discovered more than 50 cases of XSS vulnerabilities (www.exploit-db.com/exploits/37389) in Koha and reported these to the developers (https://koha-community.org/security-release-koha-3-20-1; https://koha-community.org/security-release-koha-3-16-12). Two related Koha 3.20.1 security problems have since been assigned identifiers on MITRE's Common Vulnerabilities and Exposures (CVE) list: CVE-2015-4630 and CVE-2015-4631.

System-call testing

The kernel of an operating system is its central authority to enforce security features. The Linux user base is extremely large—for example, in 2013 more than 1.5 million Android devices were activated per day.[13] Some manual tests exist for the Linux kernel, such as those created by the Linux Test Project (https://github.com/linux-test-project/ltp). There are also fuzzing techniques for system-call testing.[14] One such Linux system-call fuzz tester is Trinity (http://codemonkey.org.uk/projects/trinity).

We generated IPMs for the Linux system-call API by introducing novel combinatorial modeling methodologies.[15] Furthermore, we developed a highly configurable combinatorial kernel testing framework, namely ERIS, which encompasses automated test execution and logging capabilities. The testing framework allows any test generator to be plugged in for generating automated tests for the Linux system calls. We used the NIST ACTS tool to produce test suites covering numerous *t*-way combinations depending on the number of system-call arguments. Our testing experiments revealed various erroneous cases that we flagged for further analysis.

Hardware Trojan detection

Contemporary hardware design shares many similarities with software development practice. The insertion of malicious functionality in hardware is a realistic threat. Hardware Trojan activation can be controlled using a short input pattern out of billions of possibilities, and the effect of the Trojan's payload can be observed as erroneous circuit output. The attack can be as subtle as introducing a faulty operation on a cryptographic core and deriving the cryptographic key afterward.[16]

Established functional testing techniques don't cope well with hardware Trojans due to the enormous space of possible input signals used as activation patterns. Modeling the attack as a functional black-box testing problem, we used CT to reduce the test suites' size and, consequently, the testing time by three orders of magnitude compared to alternative approaches, thereby guaranteeing multiple Trojan activations. This research also resulted in new, optimized CAs with interaction strengths beyond 6 and introduced CT as an efficient means for hardware as well as software security testing.[17,18]

Protocol interaction testing

Software implementations of the Transport Layer Security (TLS) protocol are critical in the security of Internet communications and beyond. Software bugs and attacks still surface and can be attributed to the complexity of the protocol and its large number of interactions. System designers and integrators face a challenging task: they must ensure that their system's TLS implementation can correctly handle all *cipher suites*—the named combination of cryptographic algorithms to subsequently use that are negotiated between a client and a server in the TLS connection establishment phase—and, at the same time, conform to a desired level of a security.

We presented a coverage measurement for recommendations on available TLS cipher suites.[19] After deriving the appropriate IPMs, we measured and analyzed the cipher suites using the NIST Combinatorial Coverage Measurement (CCM) tool. None of the proposed recommendations covered all two-way combinations of algorithms appearing in a cipher suite; this might be due to incompatibilities or security considerations. The analysis results also had implications for aspects of test quality. For example, increasing the number of potential interactions between configuration settings could also increase the risk of bugs or vulnerabilities arising from feature interactions among two or more components. Thus, measuring the level of two-way, three-way, and higher-strength interactions might be informative for testing.

The success of our CT-based approaches for security testing in a wide variety of use cases motivates further intensive research in this area. In particular, combinatorial security testing might prove particularly useful for the Internet of Things. IoT systems send and receive data from a large, often continually changing set of interacting devices, and the number of potential communicating pairs increases with the square of the number of devices. Combinatorial methods are ideally suited for this environment, where testing can involve a very large number of nodes and combinations. C

REFERENCES

1. J. Heffley and P. Meunier, "Can Source Code Auditing Software Identify Common Vulnerabilities and Be Used to Evaluate Software Security?," *Proc. 37th Ann. Hawaii Int'l Conf. System Sciences* (HICSS 04), 2004; www.computer.org/csdl/proceedings/hicss/2004/2056/09/205690277.pdf.
2. D.R. Kuhn, R.N. Kacker, and Y. Lei, *Introduction to Combinatorial Testing*, CRC Press, 2013.
3. D.R. Kuhn, D.R. Wallace, and A.M. Gallo Jr., "Software Fault Interactions and Implications for Software Testing," *IEEE Trans. Software Eng.*, vol. 30, no. 6, 2004, pp. 418–421.
4. K.Z. Bell, "Optimizing Effectiveness and Efficiency of Software Testing: A Hybrid Approach," PhD dissertation, North Carolina State Univ., 2006.
5. D. Cotroneo et al., "How Do Bugs Surface? A Comprehensive Study on the Characteristics of Software Bugs Manifestation," *J. Systems and Software*, vol. 113, 2016, pp. 27–43.
6. Z. Ratliff et al., "The Relationship between Software Bug Type and Number of Factors Involved in Failures," to be published in *Proc. 27th Int'l Symp. Software Reliability Eng.* (ISSRE 16), 2016.
7. Y. Lei et al., "IPOG: A General Strategy for T-Way Software Testing," *Proc. 14th Ann. IEEE Int'l Conf. and Workshops Eng. of Computer-Based Systems* (ECBS 07), 2007, pp. 549–556.
8. T. Guild, "RXSS Security Audit Results," blog, 11 Dec. 2014; www.w3.org/blog/2014/12/rxss-security-audit-results.
9. J. Bozic, D.E. Simos, and F. Wotawa, "Attack Pattern–Based Combinatorial Testing," *Proc. 9th Int'l Workshop Automation of Software Test* (AST 14), 2014; doi: 10.1145/2593501.2533502.
10. J. Bozic et al., "Attack Pattern–Based Combinatorial Testing with Constraints for Web Security Testing," *Proc. IEEE Int'l Conf. Software Quality, Reliability, and Security* (QRS 15), 2015, pp. 207–212.
11. B. Garn et al., "On the Applicability of Combinatorial Testing to Web Application Security Testing: A Case Study," *Proc. Workshop Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing* (JAMAICA 14), 2014, pp. 16–21.
12. J. Bozic et al., "Evaluation of the IPO-Family Algorithms for Test Case Generation in Web Security Testing," *Proc. IEEE 8th Int'l Conf. Software Testing, Verification, and Validation Workshops* (ICSTW 15), 2015; doi:10.1109/ICSTW.2015.7107436.
13. A. Dobie, "Android Reaches 900 Million Activations," Android Central, 15 May 2013; www.androidcentral.com/android-reaches-900-million-activations.
14. A. Gauthier et al., "Enhancing Fuzzing Technique for OKL4 Syscalls Testing," *Proc. 6th Int'l Conf. Availability, Reliability, and Security* (ARES 11), 2011, pp. 728–733.
15. B. Garn and D.E. Simos, "Eris: A Tool for Combinatorial Testing of the Linux System Call Interface," *Proc. IEEE 7th Int'l Conf. Software Testing, Verification, and Validation Workshops* (ICSTW 14), 2014, pp. 58–67.
16. S. Bhasin et al., "Hardware Trojan Horses in Cryptographic IP Cores," *Proc. Workshop Fault Diagnosis and Tolerance in Cryptography* (FDTC 13), 2013, pp. 15–29.
17. P. Kitsos et al., "Exciting FPGA Cryptographic Trojans Using Combinatorial Testing," *Proc. IEEE 26th Int'l Symp. Software Reliability Eng.* (ISSRE 15), 2015, pp. 69–76.
18. A.G. Voyiatzis, K.G. Stefanidis, and P. Kitsos, "Efficient Triggering of Trojan Hardware Logic," *Proc. IEEE 19th Int'l Symp. Design and Diagnostics of Electronic Circuits and Systems* (DDECS 16), 2016, pp. 200–205.
19. D.E. Simos et al., "TLS Cipher Suites Recommendations: A Combinatorial Coverage Measurement Approach," to be published in *Proc. IEEE Int'l Conf. Software Quality, Reliability, and Security* (QRS 16), 2016.

DISCLAIMER

Products may be identified in this document, but identification doesn't imply recommendation or endorsement by NIST, nor that the products identified are necessarily the best available for the purpose.

DIMITRIS E. SIMOS is a key researcher in applied discrete mathematics for information security and leads the combinatorial security testing team at SBA Research. Contact him at dsimos@sba-research.org.

RICK KUHN is a project leader and computer scientist in the Computer Security Division of the Information Technology Laboratory, National Institute of Standards and Technology (NIST). Contact him at kuhn@nist.gov.

ARTEMIOS G. VOYIATZIS is a key researcher in networked systems security at SBA Research. Contact him at avoyiatzis@sba-research.org.

RAGHU KACKER is a project leader and mathematical statistician in the Mathematical and Computational Sciences Division of the Information Technology Laboratory, NIST. Contact him at raghu.kacker@nist.gov.