Combinatorial Security Testing - 10 Years Later

Dimitris E. Simos
Paris Lodron University of Salzburg
Salzburg University of Applied Sciences

Manuel Leithner
Salzburg University of Applied Sciences

Rick Kuhn
National Security Institute
Virginia Tech

Bernhard Garn
Paris Lodron University of Salzburg

Raghu Kacker
Information Technology Laboratory
US National Institute of Standards and Technology

Jeff Lei
Computer Science
University of Texas Arlington

2026/01/16

Abstract Ten years ago we introduced a research program applying high strength combinatorial test methods to vulnerability and fault detection for cybersecurity. In this article, we discuss advances in the field over the past 10 years, and where these methods may be effective for today's complex security problems.

Introduction

Combinatorial testing has developed over the past 25 years from simple pairwise testing techniques, to high assurance test methods.

The need to test interactions between components and inputs has been recognized since the early days of computing. Pairwise, or all-pairs, testing developed from the statistical field of design of experiments, with the objective of testing such interactions. For example, a text editing tool might be observed to fail with a single character search string and a single character replacement string, but work correctly if only one of these conditions is true. So testing all 2-way combinations of these parameter values could detect this problem.

But what if a failure involves more than two factors interacting? There is no guarantee that covering all pairs of parameter values would detect this. Some errors are only revealed when more than two conditions are true. Surprisingly, no one had investigated the distribution of interactions involving more than two factors prior to a series of studies by NIST from 1999 to 2004 [35]. Looking at many types of software applications revealed similar patterns of failure-triggering interactions, as shown in Fig. 1. Combinatorial test tools typically provide up to 6-way combination coverage, and have been demonstrated to provide highly effective fault detection using far fewer resources than other test methods.

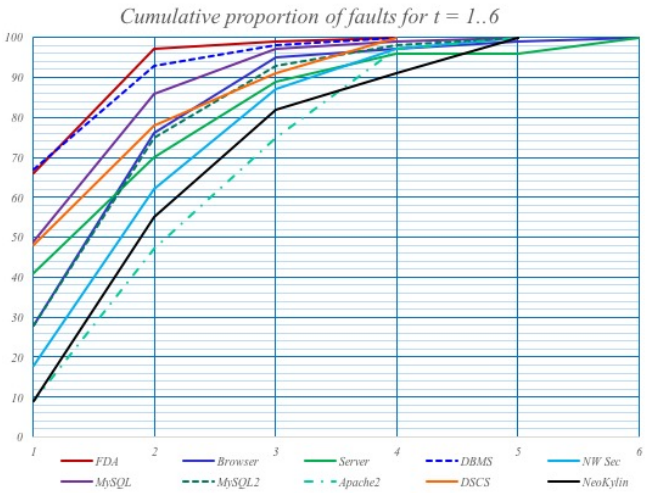
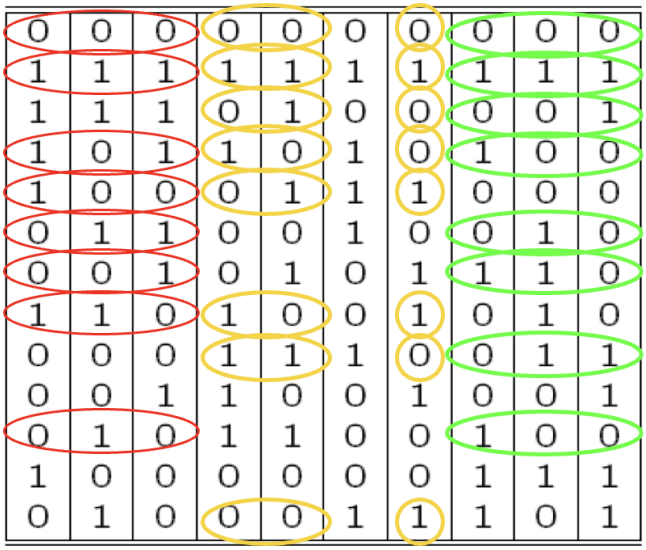


Figure 1: Distribution of Interaction Faults

Now consider what this means for software testing. If nearly all failures can be triggered by t or fewer factors interacting, then testing all t-way combinations can create the conditions required for exposing nearly all failures. It will still be necessary to use category partitioning or other methods to bin continuous-value parameter values, but fault detection can be equal, or nearly equal, to exhaustive testing with a 20X to 700X reduction in test set size. Tools capable of compressing combinations into a small number of tests make this method practical for industrial use, providing better testing at lower cost.



0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1
1	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0	1

Figure 2: Covering array of 10 binary parameters

Combinatorial Methods in Testing

The fundamental tool for this approach is a **covering array**, a matrix which includes all t-way combinations of values, for some specified level of t. An example is shown in Fig. 2, which shows a 3-way covering array of 10 binary parameters. It can be seen that any three columns include all eight possible binary values.

More recently, combinatorial methods have been extended beyond static covering arrays to testing sequences of events or input value combinations, extending the efficiency advantages to dynamic testing. Three types of mathematical structures are important for combinatorial security testing (CST).

- Covering array – all t-way combinations of variable values. Each row represents a test and each column gives values for variables. For example, Fig. 2 shows an array with 13 rows that cover all 3-way combinations of 10 binary variables. If we take any three columns, all eight possible values of three binary variables will appear somewhere in the column. For exhaustive coverage of n variables with v values each, v^n tests would be required, but with a covering array, the number of tests is proportional to v^t log n. Thus, even with a large number of variables, the number of tests can be kept to a practical number.

- Sequence covering array – includes all s-length sequences. Most systems, hardware or software, produce different responses depending on the order of events encountered in test or operation. A sequence covering array includes all s-length permutations of events, possibly interleaved with other events, in some row of the array. The number of rows is proportional to log n for n events, rather than $n!$ for all possible permutations.

- Ordered combinatorial cover – includes all s-length sequences of t-way combinations of variable values. The concepts of combination covers and sequence covers are combined in the third structure considered here – the ordered combination cover. This array includes all s-length sequences of t-way combinations of variable values. That is, every t-way combination of values is eventually followed by every other t-way combination of values. Ordered combination covers can also be constructed, using deBruijn sequences, to contain every t-way combination of values is immediately followed by every other t-way combinations of values in some s-length sequence of tests.

Practical covering array generation tools are freely available, with a well established user base including some of the world's largest organizations [35], [18]. One widely used tool is ACTS [54], [35], which produces efficient covering arrays. ACTS also includes support for constraints, which specify parameter values that must be included or excluded based on user-defined logic expressions. CAgen [48], can be used for efficient construction of covering arrays and optimized generation of large instances. CAgen supports improved constraint handling as well as multithreaded techniques [49]. A comprehensive review of the eight most widely used covering arrays tool has been published [28], analyzing the strengths and properties of the tools, enabling users to select those that most closely match the needs of their organizations.

Combinatorial Methods for Security

The range of cybersecurity problems today is vast, and often viewed as evolving faster than practical solutions. Let's consider a few of the most enduring and challenging issues, before illustrating how combinatorial methods have been advanced to provide solutions.

Combinatorial Security Testing (CST) is the application of combinatorial testing to the detection of vulnerabilities [45]. In contrast to classic conformance-focused testing, wherein the input domain of a system is modeled through its input parameters and a discretized form of their respective permitted values, CST is not principally concerned with producing valid inputs or pseudo-exhaustively covering the input space of the system under test (SUT). Instead, it models the terminals and nonterminals of an *attack grammar*. Recall that a formal grammar gives rise to a language made up of individual words; in CST, these words are called *attack vectors*. An attack grammar is designed so that the resulting attack vectors have a high probability of exploit-

ing a potential vulnerability in the SUT. It is commonly crafted in collaboration with security experts proficient in identifying the type of vulnerability in question. In contrast to conformance testing, CST attack grammars tend to be reusable across a wide variety of SUTs, as long as the target vulnerability remains constant. One more difference lies in the task of the test oracle. Traditionally, the task of this component is to evaluate whether the response or behavior of the SUT when confronted with a particular test case is correct according to its requirements and business rules. In CST, the oracle is scarcely concerned with this aspect; instead, it aims to reliably and accurately identify the symptoms of successful exploitation. While a well-crafted CST oracle is reusable across SUTs, it tends to be more invasive than conformance-focused oracles, often requiring binary instrumentation, recording of intermediate outputs such as generated database queries, or modified runtimes with extended analysis capabilities.

In addition to vulnerability detection, combinatorial methods can be especially valuable in analyzing complex access control rules [17], with a technique that instantiates rules with values from a covering array [26]. A model checker is then used to produce counterexamples that can be converted to input-expected result pairs, thus solving the test oracle problem.

Internet and WWW Security

The web is presently the most popular and ubiquitous means of exchanging information. No longer confined to dedicated browsers, myriads of mobile apps, store displays, and even point-of-sale terminals are ultimately based on web applications. At the same time, security flaws continue to plague the landscape, with high-impact injection vulnerabilities such as cross-site scripting (XSS) or SQL injections remaining prevalent despite the availability of effective mitigations. The impact ranges from denial of service to data breaches and compromised user accounts, often leading to significant financial damage and loss of reputation. Developers and testers alike require the means to conduct high-coverage security testing in a short time frame, necessitating efficient testing methods.

The applicability and usability of CST approaches in this domain was initially confirmed in a 2014 work [5] that introduced the use of an attack grammar created in collaboration with professional penetration testers to describe the structure of inputs capable of exploiting XSS vulnerabilities. The results of this work also led to the discovery of a previously unknown vulnerability in the World Wide Web Consortium's (W3C) *tidy* service used to validate and optimize HTML code. Additionally, a refined version of the attack grammars provided by this work were later utilized in the construction of an attack-pattern based automated security testing process [2].

Subsequent works have increasingly enhanced the fault detection capabilties of CST methods in this area by a) selecting attack grammars according to the *output context* of an injection [42], i.e. the logical location of the tester-provided input in the output generated by the SUT; for the purposes of XSS, this might be an attribute of a HTML element or text contained within a piece of JavaScript code; and b) identifying characters that are being filtered by the SUT for the purposes of truncating test sets and applying additional escape mechanisms that aim to obfuscate these inputs so that filters are no longer capable of detecting and removing them [30]. Enhanced with these innovations, modern CST methods are capable of identifying more injection vulnerabilities than state-of-the-art open source security testing tools including sqlmap, w3af and wapiti [15].

In order to increase the efficiency of developers who seek to mitigate injection vulnerabilities discovered through CST methods, researchers developed a process that is capable of identifying the components of an attack vector that were required for successful exploitation [43]. A similar approach was later used to automatically generate and refine a knowledge base of XSS attack vectors [8]. In this work, combinatorial fault localization [16] is used to iteratively learn the constraints required to construct successful exploits.

Some of the authors of the aforementioned works have also investigated methods to identify web application endpoints despite the increasing reliance of modern sites on client-side techniques. As the effects of JavaScript and other modern web standards can not be observed using traditional static analysis techniques, they utilized dynamic analysis based on actual browser engines [31]. While much more thorough, they also tend to incur some performance penalties, a factor that led to the evolution into a concurrent hybrid method [32] that combines a high-performance static analysis-based crawler with the previously mentioned work.

Software cannot be proven free from vulnerabilities. Common approaches to mitigate the impact of unknown injection flaws include web application firewalls [7], which aim to detect and neutralize malicious input before it reaches the application, and HTML output filters [55] that seek to modify user-generated content so that potentially injected code will not be executed. Recently introduced CST-based techniques provide the means to evaluate the efficacy of such approaches at scale.

Securing Protocols and Networks

Protocols are the cornerstones of network communication between devices. They encapsulate the format, content and logical order of messages transmitted between endpoints, enabling interoperability regardless of manufacturer and implementation characteristics. While

customers rely on the correctness, safety and security of their devices and applications, design or implementation errors do occur and often result in widespread issues such as the OpenSSL Heartbleed vulnerability (CVE-2014-0160), attacks on the DNS infrastructure, or downgrade and eavesdropping exploits against encryption protocols. They may lead to the disclosure or manipulation of secret information, denial of service, and compromised accounts, servers or websites.

The last decade has witnessed the use of combinatorial testing techniques for identifying inconsistencies and faults in TLS implementations, covering all aspects of the protocol, from certificate parsing (presented below in the section on Applied Cryptography) to malformed messages [38] to altered communication sequences, where advanced concepts such as weighted CST [11] and a combination of CST with AI-based planning methods [3, 39] have proven useful. Combinatorial coverage techniques have additionally been utilized to analyze ciphersuite recommendations for use in the TLS protocol [44]. One of the most advanced works in this area greatly enhances the techniques of these earlier works to automatically constrain input parameter models (IPM), generate SUT-specific test sets, and partially automate the construction of oracles used to evaluate 13 TLS implementations [33]. In addition to two vulnerabilities in MatrixSSL, this work identified more than 200 faults across all SUTs.

Other works have focused on internet of things (IoT) devices, especially in connection with smart cities and home automation [9]. The ubiquitous nature of these devices and their strong privacy impact make them particularly valuable as systems under test.

Similar to IoT devices, industrial control systems tend to feature limited computing resources. At the same time, the potential impact of vulnerabilities in this class of devices is nothing short of disastrous; many standards thus require fully exhaustive testing of components used in these settings. The increasing complexity due to the introduction of remote management features may, however, render such approaches infeasible. A work investigating the use of CT for cloud-connected industrial control systems [47] concludes that it offers a reasonable trade-off between verification depth and model complexity for formal verification of complex systems in the absence of requirements mandating exhaustive testing.

The concept of immutability is essential to the operation of blockchains and digital ledger technologies as a whole. Obtaining a high degree of assurance that smart contracts, on-chain programs that can automate transactions and cannot be updated once deployed, behave correctly according to their specification, is vital to the secure operation of decentralized finance. A 2022 work [4] combines constraint solving, selective state exploration, and combinatorial testing to improve code coverage for these applications.

Similar to digital ledgers, anonymity protocols have strong security requirements and aim to provide a high degree of assurance that even well-funded threat actors, including nation states, are unlikely to succeed in attacking their users. Tor is a low-latency anonymity network and likely the most popular of its kind; as such, it is uniquely exposed to the efforts of authoritarian regimes to unmask its users. A recent work uses CST methods to target the client configuration interface of this tool [41] and identifies several flaws in the configuration checker.

5G represents the most recent widely deployed generation of mobile networks, designed to offer scalability, security and significantly improved bandwidth compared to its predecessor LTE. The Non-Access Stratum (NAS) protocol enables the establishment and maintenance of communication sessions between user equipment and core network services. Using CST methods, Wang et al. [50] discovered a total of five vulnerabilities in two distinct 5G simulation environments, srsUE and UERANSIM, leading to leakage of user identity information, integrity protection bypass, or desynchronization of the state of the user device with the 5G core network, amongst other issues.

One of the most recent applications of CST is in evaluating the security of devices featuring Bluetooth Low Energy, a personal area network commonly utilized for streaming data between mobile phones or laptops and audio, healthcare or fitness devices. Schreiber et al. [37] model the individual layers of the protocol and evaluate 11 devices, ultimately discovering 19 distinct vulnerabilities that range from short-term disruptions to denial of service requiring a reboot to potential memory corruption issues.

Denial of service attacks can sometimes be achieved through a packet sequence that results in a network deadlock. Thus, it is important to ensure that configurations cannot lead to deadlock, and simulations can be highly effective for this analysis. A 2020 work showed that combinatorial methods can identify deadlock configurations faster and more consistently than conventional random Monte Carlo simulation approaches. [25].

Buffer overflows have always been one of the most significant sources of vulnerabilities, not least in the context of protocol security. In one application of combinatorial methods, Wang et al. [51] showed how the strong code coverage generated by combinatorial testing can improve detection of these vulnerabilities.

More recently, some members of the combinatorial testing research community have turned their attention towards reverse engineering of specifications for unknown or proprietary protocols with the aim of inferring input parameter models for CST [29]. The pragmatic nature of such endeavors tends to reveal challenges concerning well-established methods in our field: In this instance, it identified several open questions regarding

the definition of input space coverage for complex real-world protocols.

Hardware and Kernel Security

While the vast majority of works in this field target user-space software, the most insidious attackers undermine the very basis of security assumptions in computing: The operating system (OS) kernel and the hardware layer are singularly high-impact targets for resourceful adversaries.

Seeking to identify vulnerabilities in the Linux implementation of system calls, KERIS [13] models the parameters of these OS functions and utilizes the Kernel Address Sanitizer (KASAN) to detect invalid memory accesses. The evolution from its predecessor work ERIS [10], which utilized similar models, illustrates the singularities of CST mentioned above: While the earlier approach relied on kernel messages, system call return values and side effects, only the addition of an advanced oracle with greater instrumentation capabilities allowed it to evolve from a regression testing method to one with a focus on security vulnerabilities.

Virtually every nation relies on imports to meet their demand for chips. The potential impact of maliciously inserted hardware trojans, minute modifications between the original chip design and the final physical product, was recognized almost a decade ago by members of the CST community. A 2015 work [22] utilizes the coverage properties offered in our field to excite hardware trojans – that is, it seeks to detect triggers that alter the functionality of a processor compared to its normal mode of operation. A later work [19] extends these capabilities by utilizing locating arrays to identify the concrete combination of bits to enable such modified behaviour.

Physical unclonable functions (PUFs) are security features that rely on minute differences that arise unavoidably due to imperfections in the manufacturing process, such as the path delay between electronic components, the resistance of metals, or the breakdown of oxides over time. While these factors are significant enough to be measured, they are generally assumed to be uncontrollable to the degree required to clone them and are thus used in fingerprinting and cryptographic applications. A recent CST work [27] analyzes the use of *combinatorial frequency differences* (CFD) for analyzing the susceptibility of physical unclonable functions (PUFs) to machine learning attacks. CFD compares the frequency of occurrence of t-way combinations between two data sets. The method may be useful for identifying bit combinations that have a disproportionately strong influence on PUF response bit values.

Applied Cryptography

CST finds ample applications in fields that require a high degree of assurance and bear a particular criticality for the security of systems.

The primary example for such an area is cryptographic implementations. Vulnerabilities in such libraries may be used to break security properties such as confidentiality (preventing unauthorized access to information) or integrity (ensuring that information can not be covertly modified). The correctness of algorithm implementations is commonly validated using test vectors, specific inputs for which the correct output is known. A 2019 work [40] evaluates the combinatorial coverage of these test sets and identifies avenues towards the quantification of residual risk incurred when these sets are used. Researchers have additionally developed novel methods that go beyond the state of the art; in particular, CST methods have been used to identify weaknesses in cryptographic hash functions [34], having been shown to reduce the number of tests required to detect obscure bugs in cryptographic code by orders of magnitude. As the practical feasibility of quantum computing-based attacks on cryptography continues to improve, efforts to implement and deploy post-quantum secure alternatives have been ongoing. In order to validate the implementations of these novel algorithms, researchers at Loyola University Maryland and the US National Institute for Standards and Technology have investigated the use of metamorphic and combinatorial testing techniques [36]. This work found a high rate of faults in their SUT and established the applicability of metamorphic testing to cryptographic applications.

TLS has been mentioned above in the context of protocol security. It is one of the most popular, although not the only, application of x.509 certificates, which are used to cryptographically verify the identity of entities. A 2017 work [23] aims to identify vulnerabilities in certificate parsers through the use of inputs created using CST methods. Flaws in these software components might be used to impersonate other websites or certificate authorities and thus perform man-in-the-middle attacks.

Conclusion and Future Outlook

Over the past decade, combinatorial security testing has evolved from a promising application of combinatorial testing into a mature, diverse, and empirically validated methodology giving rise to a broader research field encompassing high assurance security testing across software, protocols, hardware, and applied cryptography.

Moreover, there has been preliminary work in other domains inspired by CST, e.g. for testing autonomous and highly configurable systems, browser fingerprinting

and developing crisis scenarios for production facilities. The aim of the rest of the section is to outline future directions of research in CST going beyond classical software security testing.

Testing Autonomous Systems Autonomous vehicles promise to greatly enhance the safety and efficiency of automotives. At the same time, they require rigorous testing to ensure that road users will under no circumstances be endangered. The vast parameter space to be considered has been identified as one of the main reasons behind the complexity of testing autonomous vehicles. A proposed testing methodology for such systems combines a test case generation approach based on combinatorial testing with an automated test oracle based on runtime verification [53]. A common approach to ensuring the correct response of autonomous vehicles under a wide range of circumstances is virtual validation using simulation. However, it is not trivial to generate scenarios for critical scenario identification automatically. A work by Tao et al. [46] describes test generation targeting an Autonomous Emergency Breaking (AEB) system using an ontology-based representation of the domain and combinatorial testing. Kampel et al. [20] later proposed a combination with fault locating arrays that similarly targets the AEB function and is capable of identifying parameter-value combinations leading to crash scenarios. But how do combinatorial testing techniques compare to other approaches? A 2023 article [24] analyzes its performance while identifying critical scenarios for AEB functions. In contrast to random testing and search-based methods, CT at strength 3 (i.e. covering all interactions of three parameters) detects all different AEB function faults. At the same time, the work finds that search-based testing exhibits a higher likelihood of detection for specific crash scenarios, ultimately recommending a composite approach in this domain.

All aforementioned techniques assume that a specification of the SUT is available and the manufacturer of the autonomous driving vehicle component cooperates with the testing process. In practice, this is not always the case: Researchers may opt to buy electronic control units (ECUs) on their own in order to evaluate their safety and security. Under these circumstances, documentation of available messages – commonly exchanged using the CAN bus protocol [1] – tends to be incomplete or unavailable. Demonstrating the versatility of combinatorial testing, a recent work [52] utilizes combinatorial test sets to identify unknown or undocumented functionality in ECUs, paving the way for subsequent analysis without relying on vendor cooperation.

Combinatorial Fingerprinting Despite the best efforts of IT security staff, adversaries continue to compromise user devices, often enabling them to take over login sessions and act on behalf of their victims. One way to identify this situation is the use of fingerprinting techniques to identify the client software or device as uniquely as possible. A 2019 work [12] utilizes combinatorial methods to generate mutated TLS handshakes with the goal of identifying individual browser versions. An enhanced version [14] additionally incorporates properties of the client's original handshake message and features a larger case study, demonstrating a significant increase in the entropy of the achieved splittings. This allows for a more precise identification of the TLS client and version than permitted by either of the underlying approaches in isolation.

Disaster Management Training exercises are an important tool in disaster management, as they can assist in a multitude of tasks, such as planning pre-crisis resource requirements and allocation, response planning and help train emergency personnel for actual crises. In [6], combinatorial sequences for the automated generation of scenarios for disaster exercises have been utilized. The derived scenarios fulfill different notions of combinatorial sequence coverage, a property first shown for combinatorial event based testing. Recently, this approach has been used to increase the resilience of production facilities via quantifying diversity in combinatorial disaster scenarios [21].

Disclaimer. The opinions, recommendations, findings, and conclusions contained in this article do not necessarily reflect the views or policies of the National Institute of Standards and Technology (NIST) or the United States Government. Any commercial or non-commercial entity identified here does not imply recommendation or endorsement by NIST or that they are best available.

References

[1] Mehmet Bozdal, Mohammad Samie, Sohaib Aslam, and Ian Jennions. Evaluation of CAN bus security challenges. *Sensors*, 20(8):2364, 2020.

[2] Josip Bozic, Bernhard Garn, Ioannis Kapsalis, Dimitris Simos, Severin Winkler, and Franz Wotawa. Attack pattern-based combinatorial testing with constraints for web security testing. In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pages 207–212. IEEE, 2015.

[3] Josip Bozic, Kristoffer Kleine, Dimitris E Simos, and Franz Wotawa. Planning-based security testing of the SSL/TLS protocol. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 347–355. IEEE, 2017.

[4] Huadong Feng, Xiaolei Ren, Qiping Wei, Yu Lei, Raghu Kacker, D. Richard Kuhn, and Dimitris E. Simos. Magicmirror: Towards high-coverage fuzzing of smart contracts. In *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 141–152, 2023.

[5] Bernhard Garn, Ioannis Kapsalis, Dimitris E Simos, and Severin Winkler. On the applicability of combinatorial testing to web application security testing: a case study. In *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*, pages 16–21, 2014.

[6] Bernhard Garn, Klaus Kieseberg, Dominik Schreiber, and Dimitris E. Simos. Combinatorial sequences for disaster scenario generation. *Operations Research Forum*, 4(50), 2023.

[7] Bernhard Garn, Daniel Sebastian Lang, Manuel Leithner, D Richard Kuhn, Raghu Kacker, and Dimitris E Simos. Combinatorially XSSing web application firewalls. In *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 85–94. IEEE, 2021.

[8] Bernhard Garn, Marco Radavelli, Angelo Gargantini, Manuel Leithner, and Dimitris E Simos. A fault-driven combinatorial process for model evolution in XSS vulnerability detection. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 207–215. Springer, 2019.

[9] Bernhard Garn, Dominik-Philip Schreiber, Dimitris E Simos, Rick Kuhn, Jeff Voas, and Raghu Kacker. Combinatorial methods for testing internet of things smart home systems. *Software Testing, Verification and Reliability*, 32(2):e1805, 2022.

[10] Bernhard Garn and Dimitris E Simos. Eris: A tool for combinatorial testing of the linux system call interface. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*, pages 58–67. IEEE, 2014.

[11] Bernhard Garn, Dimitris E Simos, Feng Duan, Yu Lei, Josip Bozic, and Franz Wotawa. Weighted combinatorial sequence testing for the TLS protocol. In *2019 IEEE international conference on software testing, verification and validation workshops (ICSTW)*, pages 46–51. IEEE, 2019.

[12] Bernhard Garn, Dimitris E Simos, Stefan Zauner, Rick Kuhn, and Raghu Kacker. Browser fingerprinting using combinatorial sequence testing. In *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, pages 1–9, 2019.

[13] Bernhard Garn, Fabian Würfl, and Dimitris E Simos. KERIS: a CT tool of the linux kernel with dynamic memory analysis capabilities. In *Haifa Verification Conference*, pages 225–228. Springer, 2017.

[14] Bernhard Garn, Stefan Zauner, Dimitris E Simos, Manuel Leithner, Richard Kuhn, and Raghu Kacker. A two-step TLS-based browser fingerprinting approach using combinatorial sequences. *Computers & Security*, 114:102575, 2022.

[15] Bernhard Garn, Jovan Zivanovic, Manuel Leithner, and Dimitris E Simos. Combinatorial methods for dynamic gray-box SQL injection testing. *Software Testing, Verification and Reliability*, 32(6):e1826, 2022.

[16] Laleh Sh Ghandehari, Jaganmohan Chandrasekaran, Yu Lei, Raghu Kacker, and D Richard Kuhn. BEN: A combinatorial testing-based fault localization tool. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 1–4. IEEE, 2015.

[17] Vincent C Hu, Rick Kuhn, Dylan Yaga, et al. Verification and test methods for access control policies/models. *NIST Special Publication*, 800:192, 2017.

[18] Raghu N Kacker, D Richard Kuhn, Yu Lei, and Dimitris E Simos. Factorials experiments, covering arrays, and combinatorial testing. *Mathematics in Computer Science*, 15(4):715–739, 2021.

[19] Ludwig Kampel, Paris Kitsos, and Dimitris E Simos. Locating hardware trojans using combinatorial testing for cryptographic circuits. *IEEE Access*, 10:18787–18806, 2022.

[20] Ludwig Kampel, Michael Wagner, Dimitris E Simos, Mihai Nica, Dino Dodig, David Kaufmann, and Franz Wotawa. Applying CT-FLA for AEB function testing: A virtual driving case study. In *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 237–245. IEEE, 2023.

[21] Klaus Kieseberg, Konstantin Gerner, Bernhard Garn, Wolfgang Czerni, Dimitris E. Simos, D. Richard Kuhn, and Raghu Kacker. Combinatorial Methods for Enhancing the Resilience of Production Facilities. In *2025 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 310–313, 2025.

[22] Paris Kitsos, Dimitris E Simos, Jose Torres-Jimenez, and Artemios G Voyiatzis. Exciting fpga cryptographic trojans using combinatorial testing. In *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, pages 69–76. IEEE, 2015.

[23] Kristoffer Kleine and Dimitris E Simos. CoveringCerts: Combinatorial methods for X.509 certificate testing. In *2017 IEEE International conference on software testing, verification and validation (ICST)*, pages 69–79. IEEE, 2017.

[24] Florian Klück, Yihao Li, Jianbo Tao, and Franz Wotawa. An empirical comparison of combinatorial testing and search-based testing in the context of automated and autonomous driving systems. *Information and Software Technology*, 160:107225, 2023.

[25] D Richard Kuhn, Raghu Kacker, and Yu Lei. Random vs. combinatorial methods for discrete event simulation of a grid computer network. In *Selected Papers Presented at MODSIM World 2009 Conference and Expo*, 2010.

[26] D Richard Kuhn and Vadim Okun. Pseudo-exhaustive testing for software. In *2006 30th Annual IEEE/NASA Software Engineering Workshop*, pages 153–158. IEEE, 2006.

[27] D Richard Kuhn, Mohammad S Raunak, Charles B Prado, Vinay C Patil, and Raghu N Kacker. Combination frequency differencing for identifying design weaknesses in physical unclonable functions. In *ICST Workshops*, pages 110–117, 2022.

[28] Manuel Leithner, Andrea Bombarda, Michael Wagner, Angelo Gargantini, and Dimitris E Simos. State of the CArt: evaluating covering array generators at scale. *International Journal on Software Tools for Technology Transfer*, 26(3):301–326, 2024.

[29] Manuel Leithner and Dimitris E. Simos. Reverse engineering for input modeling: Input parameter model inference from network traces. In *IFIP International Conference on Testing Software and Systems*, pages 178–194. Springer, 2025.

[30] Manuel Leithner, Bernhard Garn, and Dimitris E Simos. HYDRA: Feedback-driven black-box exploitation of injection vulnerabilities. *Information and Software Technology*, 140:106703, 2021.

[31] Manuel Leithner and Dimitris E. Simos. XIEv: dynamic analysis for crawling and modeling of web applications. In *SAC '20: Proceedings of the 35th Annual ACM Symposium on Applied Computing*, SAC '20, page 2201–2210, New York, NY, USA, 2020. Association for Computing Machinery.

[32] Manuel Leithner and Dimitris E. Simos. CHIEv: concurrent hybrid analysis for crawling and modeling of web applications. *SIGAPP Applied Computing Review*, 21(1):5–23, July 2021.

[33] Marcel Maehren, Philipp Nieting, Sven Hebrok, Robert Merget, Juraj Somorovsky, and Jörg Schwenk. TLS-Anvil: Adapting combinatorial testing for TLS libraries. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 215–232, 2022.

[34] Nicky Mouha, Mohammad S Raunak, D Richard Kuhn, and Raghu Kacker. Finding bugs in cryptographic hash function implementations. *IEEE transactions on reliability*, 67(3):870–884, 2018.

[35] NIST. Combinatorial methods for trust and assurance. `https://csrc.nist.gov/acts/`, 2025.

[36] Sydney Pugh, Mohammad S Raunak, D Richard Kuhn, and Raghu Kacker. Systematic testing of post-quantum cryptographic implementations using metamorphic testing. In *2019 IEEE/ACM 4th International Workshop on Metamorphic Testing (MET)*, pages 2–8. IEEE, 2019.

[37] Dominik-Philip Schreiber, Manuel Leithner, Jovan Zivanovic, and Dimitris E. Simos. Bluetooth low energy security testing with combinatorial methods. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, 2025.

[38] Dimitris E Simos, Josip Bozic, Feng Duan, Bernhard Garn, Kristoffer Kleine, Yu Lei, and Franz

Wotawa. Testing TLS using combinatorial methods and execution framework. In *IFIP International Conference on Testing Software and Systems*, pages 162–177. Springer, 2017.

[39] Dimitris E Simos, Josip Bozic, Bernhard Garn, Manuel Leithner, Feng Duan, Kristoffer Kleine, Yu Lei, and Franz Wotawa. Testing TLS using planning-based combinatorial methods and execution framework. *Software quality journal*, 27(2):703–729, 2019.

[40] Dimitris E Simos, Bernhard Garn, Ludwig Kampel, D Richard Kuhn, and Raghu N Kacker. Knowledge extraction for cryptographic algorithm validation test vectors by means of combinatorial coverage measurement. In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, pages 195–208. Springer, 2019.

[41] Dimitris E Simos, Bernhard Garn, Dominik Schreiber, Manuel Leithner, D Richard Kuhn, and Raghu Kacker. On combinatorial security testing for the tor anonymity network client. In *2024 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 207–210. IEEE, 2024.

[42] Dimitris E Simos, Bernhard Garn, Jovan Zivanovic, and Manuel Leithner. Practical combinatorial testing for XSS detection using locally optimized attack models. In *2019 IEEE International conference on software testing, verification and validation workshops (ICSTW)*, pages 122–130. IEEE, 2019.

[43] Dimitris E Simos, Kristoffer Kleine, Laleh Shikh Gholamhossein Ghandehari, Bernhard Garn, and Yu Lei. A combinatorial approach to analyzing cross-site scripting (XSS) vulnerabilities in web application security testing. In *IFIP International Conference on Testing Software and Systems*, pages 70–85. Springer, 2016.

[44] Dimitris E Simos, Kristoffer Kleine, Artemios G Voyiatzis, Rick Kuhn, and Raghu Kacker. TLS cipher suites recommendations: A combinatorial coverage measurement approach. In *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 69–73. IEEE, 2016.

[45] Dimitris E Simos, Rick Kuhn, Artemios G Voyiatzis, and Raghu Kacker. Combinatorial methods in security testing. *Computer*, 49(10):10–1109, 2016.

[46] Jianbo Tao, Yihao Li, Franz Wotawa, Hermann Felbinger, and Mihai Nica. On the industrial application of combinatorial testing for autonomous driving functions. In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 234–240. IEEE, 2019.

[47] Peter WV Tran-Jørgensen, Tomas Kulik, Jalil Boudjadar, and Peter Gorm Larsen. Security analysis of cloud-connected industrial control systems using combinatorial testing. In *Proceedings of the 17th ACM-IEEE International Conference on Formal Methods and Models for System Design*, pages 1–11, 2019.

[48] Michael Wagner, Kristoffer Kleine, Dimitris E Simos, Rick Kuhn, and Raghu Kacker. Cagen: A fast combinatorial test generation tool with support for constraints and higher-index arrays. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 191–200. IEEE, 2020.

[49] Michael Wagner, Manuel Leithner, Dimitris E. Simos, Rick Kuhn, and Raghu Kacker. Developing multithreaded techniques and improved constraint handling for the tool CAgen. In *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 87–93, 2022.

[50] Siyuan Wang, Zhiwei Cui, Jie Xu, and Baojiang Cui. An efficient vulnerability detection method for 5g nas protocol based on combinatorial testing. In *International Conference on Emerging Internet, Data & Web Technologies*, pages 64–74. Springer, 2024.

[51] Wenhua Wang, Yu Lei, Donggang Liu, David Kung, Christoph Csallner, Dazhi Zhang, Raghu Kacker, and Rick Kuhn. A combinatorial approach to detecting buffer overflow vulnerabilities. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pages 269–278. IEEE, 2011.

[52] Christoph Wech, Reinhard Kugler, Manuel Leithner, and Dimitris E. Simos. Combinatorial testing methods for reverse engineering undocumented CAN bus functionality. In *Proceedings of the 19th International Conference on Availability, Reliability and Security*, ARES '24, New York, NY, USA, 2024. Association for Computing Machinery.

[53] Franz Wotawa. Testing autonomous and highly configurable systems: Challenges and feasible solutions. In *Automated Driving: Safer and More Efficient Future Driving*, pages 519–532. Springer, 2016.

[54] Linbin Yu, Yu Lei, Raghu N Kacker, and D Richard Kuhn. ACTS: A combinatorial test generation

tool. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 370–375. IEEE, 2013.

[55] Jovan Zivanovic, Manuel Leithner, Dimitris E Simos, Michael Pitzer, and Peter J Slanina. Combinatorial methods for HTML sanitizer security testing. In *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 255–259. IEEE, 2023.