SECURITY INNOVATION

Software Security Testing
BLENDING ART & SCIENCE

Since 2002, Security Innovation

has been testing software applications that the world runs on — from breathalyzers and mobile applications to enterprise ERP systems and cloud infrastructure. Our experts augment the use of commercial and open source automated security tools with specifically crafted methodologies and tools to deliver the most efficient assessments and maximize risk reduction in our customer's technologies. Tools, like humans, excel at certain tasks. Over the years, we've formulated assumptions around when to use tools versus manual techniques, and often conduct experiments comparing the two so that we can optimize our own resources and advise our clients to do the same.

While it is possible to dig a ditch with a teaspoon, a backhoe is far more efficient. However, a backhoe is overkill when pouring sugar into a cup of coffee. The same is true with application security testing. Both automated tools and manual effort play a critical role in an organization's application security program because they find different types of vulnerabilities. Tools find common problems quickly and humans hunt down vulnerabilities that elude tools. The challenge is finding the optimal balance of both to achieve breadth and depth of coverage in the most efficient manner - and ensure test efforts are commensurate with each application's level of risk and complexity.

For this study, our team used a variety of scanners, utilities and manual techniques to test twenty large-scale enterprise applications. Elements, such as time, were recorded to find vulnerabilities using manual and automated techniques; vulnerability type found by each; and the specific tool type used for each vulnerability found. The goal was to understand which techniques were most effective:

- **in which Testing Conditions**
- **by Type of Application**
- **by Vulnerability Class**

AUTOMATION

The goal of automated tools, especially scanners, is to attain breadth of coverage quickly. Automation can streamline pattern matching and identify recognizable vulnerabilities with speed and accuracy humans can't match. Where tools fall short are with false positives, false negatives, and missed coverage due to constraints such as authentication for role-sensitive applications or the use of technology that eludes "spidering" or "crawling" in web applications. The result is often an incomplete and enumerated attack surface. Tools also can't provide business or risk context and only offer high level best practices versus prescriptive and contextual remediation guidance that an expert can provide. Additionally, knowing how to use a tool effectively and what its strengths and limitations are will maximize its utility.

PROS	CONS
Find common problems quickly	Often miss business logic, complex, and design-level vulnerabilities
Can integrate into software development build cycles for automated scanning	Complacency – people trust the reports and coverage
Tests are reusable and can be run simultaneously on different machines	False positives are time consuming and need expertise to validate
Once tests begin running, expertise is not required	Provide limited remediation guidance
Coverage for source code ("white box") testing covers most common development languages	Typically limited to Web applications for dynamic ("black box") testing
Less expensive and faster than manual efforts	Configuration and setup time can be significant

MANUAL TESTING

The goal of manual security testing is to achieve depth of testing with a focus on finding serious and complex security defects that automation cannot. The tester interacts with an application in all of the different user roles (end user, administrator, etc.) to force the application to behave in ways it was never intended.

Manual security testing is an important part of a security testing regimen to exercise applications in ways automation cannot. For example, interacting with Flash elements that scanners are blind to. Manual testing is slower than automated testing, but necessary to fill code coverage gaps left by tools.

PROS	CONS
Good at finding business logic and compound vulnerabilities and focusing on hot spots	Requires highly trained and skilled testers
Tester can build upon knowledge acquired during testing for future tests (more efficient over time)	Tests are not reusable, though test patterns are
Can test any application type	Knowledge leaves organization with tester
Edge case testing that can't be easily automated	More expensive and slower than automation
Can find systemic issues, e.g., design flaws	Inconsistency from tester to tester
Provides context-aware, detailed remediation help	
Very low to zero false-positive rates	

STUDY SET-UP AND OBJECTIVES

For this internal study, the Security Innovation team selected twenty large-scale, enterprise applications with varying complexities that represent a spectrum of operational applications commonly deployed: ecommerce Web sites, knowledge and support portals, social media platforms, CRM and other business applications. The methodologies, techniques, and specific tools used during testing were documented along with every defect found. The goal was to understand which technique is most effective at uncovering vulnerabilities based on application and vulnerability type.

Several variables were identified that could affect outcomes and require future data collection and correlation:

- **Tool/support for the application under test (AUT)**
  A tool that is unaware of a rich user interface, for example, will probably miss entire classes of security defects. This is why it's important to select the best tool for the job at hand
- **Consistency of tool operator for each test**
  Changing operators can impact even simple tasks when using different automation tools for the same AUT. Whenever possible, this control was considered a variable.
- **Time**
  Automated tools are commonly used early on in testing to "crawl" an application, whereas manual testing in those areas is limited to save time. In order to measure the independent capabilities of both approaches, each application was put through a consistent battery of tests, acknowledging the superior coverage capabilities of automation in certain areas. Mindful of these variables, tests were conducted in a way that would not provide an unfair advantage to one tool or approach over another.

Tools used in automated scans:

- Burp Suite Professional
- XSS Me (Firefox add-on)
- Arachni web scanner
- Nessus
- Nmap

Tools used in manual scans:

- Publicly available add-ons
- Custom scripts
- Basic Web-browser proxy plugins

Having tested more than a thousand applications using both manual and automated techniques, our team was anticipating the following results:

- Given the absence of time constraints, manual testing would discover almost (if not) all Web application security defect types, and automated testing would uncover approximately half.
- Manual testing would be much more time and resource intensive than automation when finding the same type of defect.
- It would be difficult for automated tools to uncover compound vulnerabilities and nearly impossible to uncover any defects that are related to the business functionality of the software application.
- Automation would find more information disclosure defects (e.g. software version information) and overly informative error messages.

RESULTS SUMMARY

The study validated most of the team's assumptions regarding the effectiveness of tools vs. manual testing. Greater detail, as well as additional data points, are available in the Results Detail section.

1. There were **60 unique** (one that is found by one technique and not the other) vulnerabilities discovered. Manual testing found **59 unique vulnerabilities,** and **automation found one**.
2. Given ample time, manual testing provided approximately twice the vulnerability coverage of automation: **manual testing, 58 found 116 of 117 total vulnerabilities by tools**.
3. **Manual testing found 30 different types of vulnerabilities; automation found 14.**
4. **Certain types of vulnerabilities eluded tools,** sometimes completely. Brute forcing passwords and identifying session management flaws were commonly missed by automation.
5. **Automation was weak at finding complex cross-site scripting (XSS)** when compared to manual methods. In particular, many stored XSS and reflective XSS defects were missed. We credit this discrepancy to the complexity required in test payloads.

RESULTS DETAIL

Unique Defects

Unique defects are those that are uncovered by either automation or manual testing, but not by both.

Across the 20 applications, a total of 117 defects were uncovered. Out of these, manual testing uncovered 59 unique defects whereas automation uncovered one.

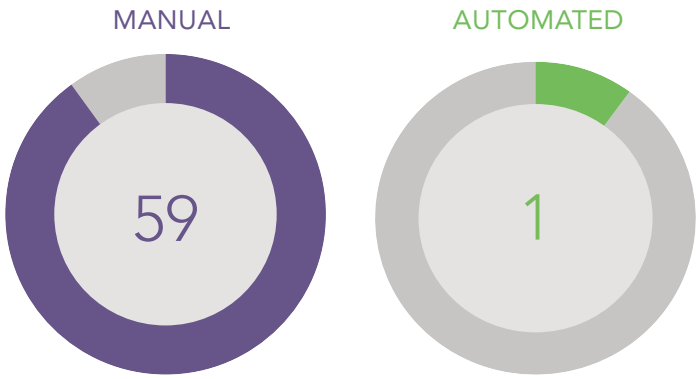

Figure 1: unique defects found by method

Total Defects

The total number of defects uncovered by each approach is related to the number of unique defects found. Manual testing uncovered 116 of 117 defects, whereas automation uncovered 58.

The effectiveness of automated testing depends on the type of applications being tested. Given the wide range of application tested, the team expected automation to uncover approximately half of the total defects. Results were in-line with expectations: automation detected 49.57% of the total number of vulnerabilities. Given absence of time constraints, the team expected manual testing to uncover almost all of the defects. As expected, this technique uncovered 99.15% of them.

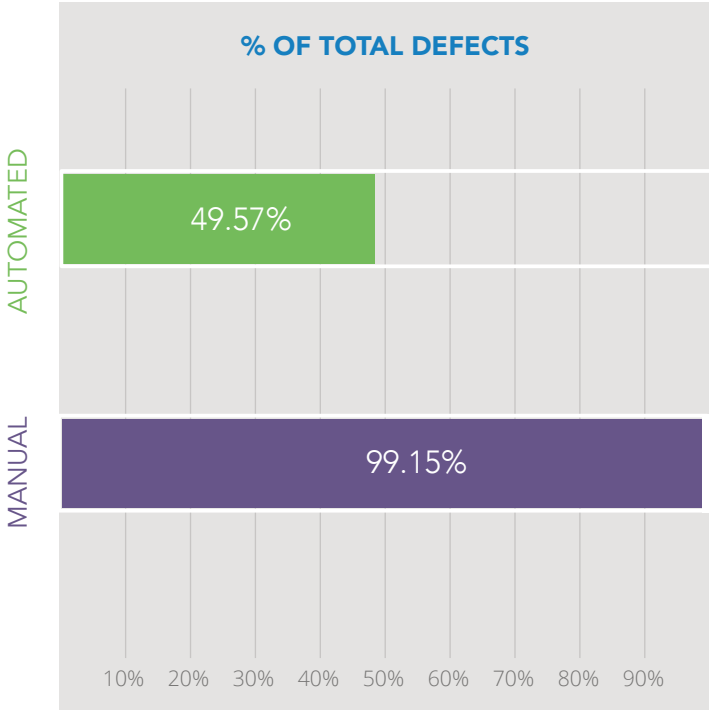

Figure 2: total defects found by method

Types of Defects

30 types of defects were uncovered between the sample applications. Manual testing found 100% of the vulnerabilities, whereas automated testing found about half. As anticipated, manual testing uncovered uncommon vulnerabilities that tools cannot like client side validation.

Only one Business Logic defect was found during manual testing and as expected, automation was unable to detect this. Automation fell short of our expectations by finding fewer information disclosure defects - manual testing uncovered 13 instances as compared to 8 found by automation.

The following chart shows the number of defect types detected by each approach in descending order and highlights three key findings:

1. All of the 30 defect types were uncovered during Manual testing.
2. Automated testing never yielded a higher number of defects for any defect type.
3. There are a large number of defect types that were not found using automation.

All of the 30 defect types were uncovered during Manual testing. Automated testing never yielded a higher number of defects for any defect type.

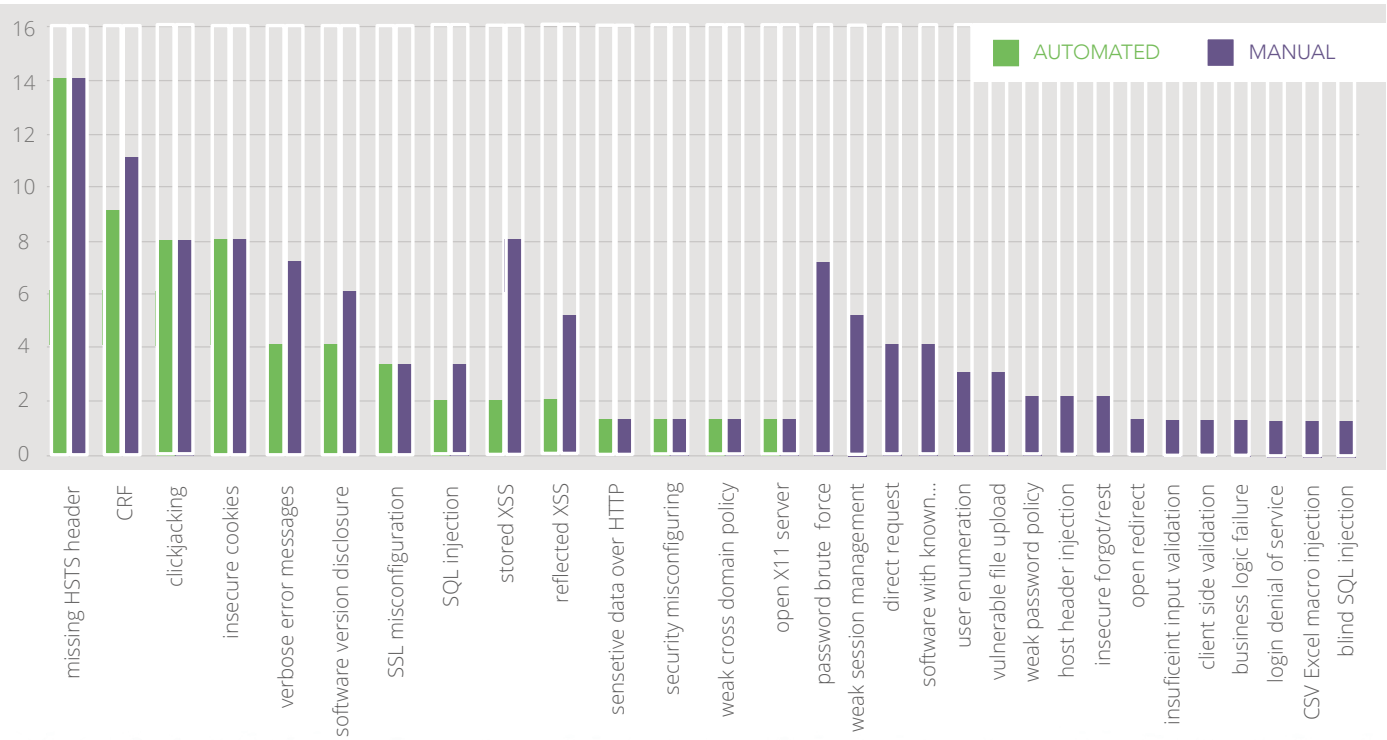

Figure 3: number of defects detected by each approach

Common Defects

The table below lists 14 defects types uncovered by both automation and manual testing. It also shows the number of defects each approach found and the average time multiplier manual testing took.

TYPE OF DEFECT	AUTOMATION	MANUAL	TIME
Missing HSTS Header	14	14	1x
Cross-Site Request Forgery (CSRF)	9	11	1.1x
Clickjacking	8	8	1x
Insecure Cookies	8	8	1x
Verbose Error Messages	4	7	2x
Software Version Disclosure	4	6	1.2x
SSL Misconfiguration	3	3	1x
SQL Injection	2	3	2x
Stored XSS	1	8	1.5x
Reflected XSS	1	5	1.2x
Sensitive Data over HTTP	1	1	1.75x
Security Misconfiguration	1	1	2x
Weak Cross Domain Policy	1	1	1x
Open X11 Server	1	1	1x

Figure 3: automation excels at getting breadth of coverage quickly

Conducting each attack type manually, e.g., testing every input to an application, is tedious and time-consuming. The strength of automation is its ability to identify general weak spots in the application that an experienced tester can focus on and conduct deeper and more specialized attacks. Some tools are great at recognizing a pattern and applying similar attacks. For example, if a scanner recognizes a SQL database error message it may try variants of SQL injection attacks in order to improve coverage breadth.

Other tests where automation excels include:

- Comparing strings to quickly cover version checks, error messages, configuration issues, TLS cipher suites and options, HTTP server verbs, and the like
- CSRF that require sending similar requests with a few headers changed based on authorization rules

This confirmed the team's assumptions that tools are a great way to begin an assessment so testers can understand what coverage gaps manual attacks need to fill.

Defects Delta

Since manual testing found all types of defects covered under this study, it is useful to see which types of defects were more likely to go undetected by automation. The graph below shows the defect types in decreasing order regarding the difference in number of defects found via manual testing vs. automation.

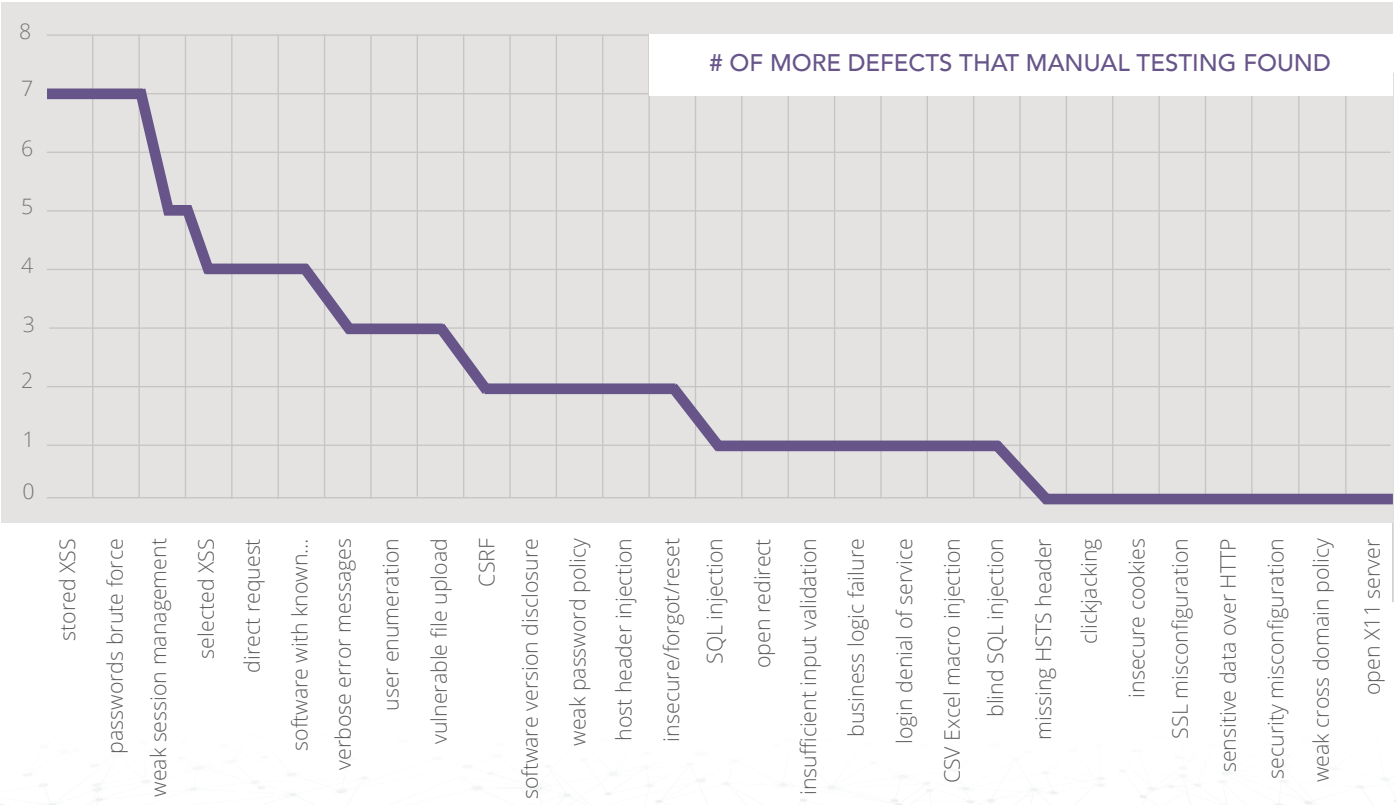

Figure 5: efficacy of manual testing vs. automated by vulnerability type

STUDY USE CASES

Since our team has often found automation to provide credible leads towards uncovering session mismanagement, we were surprised to find that it did not uncover any during our study. Below is a detailed use case demonstrating the inability of automation to uncover session mismanagement vulnerabilities in this specific instance.

Security Innovation discovered several key areas where manual testing showed advantages over automation:

- **Weak Session Management**
- **Vulnerable File Upload**
- **XSS (Cross-site scripting)**

Weak Session Management

During automated testing of a CRM application, the results did not find Session Management Weaknesses. Manual testing was then performed, discovering a Direct Object Access (DOA) vulnerability in a POST parameter not discovered in the prior automated test. When this previously missing parameter was added to a request, the user was given access to resources previously unavailable to them.

This indicated that the application:

1. Was not performing proper authorization checks.
2. May not be handing its other POST parameters in a secure manner either.

Determined to find other vulnerable parameters, further manual testing was performed on the sensitive parameter types, such as those maintaining user identity and environment. Any tampering of those parameters resulted in the server session being terminated as well as the browser being escorted back to the login page. After tampering, deleting and injecting forbidden payloads into seven other parameters, the team found a post authentication parameter named “department” to be the most resistant to change - there was something special about it and the server killed the session if you tried to change/edit it. That post authentication parameter was saved and added to an authentication request. Under a special test case where the department parameter was the lone parameter in the authentication request (no username, no password) it was discovered that the application server let the user in.

Automation can be programmed to perform very complicated checks, but not intuition, like the example described above. Such testing requires analysis and the use of variables across multiple requests and responses, potentially using them in context where they are not supposed to exist, and therefore often with unanticipated consequences. Such defects are almost always critical in severity and usually require manual testing to discover.

Vulnerable File Upload & XSS

Many applications allow users to upload files to the application server. In our study, 3 of the 20 applications were found to have vulnerabilities in their upload functionality. Simple file upload features can usually be well evaluated via automation. Automation is also very good at identifying XSS defects based on a dataset of exploit strings targeting known weaknesses in popular defense mechanisms as well as finding the same XSS defect in multiple locations of the application. This is an area where a manual endeavor would be too time consuming to be effective. Below is a use-case on file upload and XSS:

This example demonstrates a combo-bug (one that exposes two types of defects and where one would be difficult to find without the other), that is much more common than one may think. The team tested a file upload feature that implements special functionality to prevent malicious file uploads. However, this special security functionality has a weakness of its own, which not only allows malicious files to be uploaded, but also can exploit XSS defects that may have otherwise gone unnoticed.

The application also allows the user to upload PNG or JPEG files and instead of relying on file name extensions, the application checks the uploaded le for valid PNG or JPEG headers. We observed the application accepting a PNG image even when the file was named test.html.

Upon further investigation, the team discovered that the uploaded file was not being assigned system generated names and was actually saved as test.html on the server. After trying to upload an HTML file, the application rejected the upload. This told us a few things:

1. The application did not care what the extension of the uploaded file was
2. The application checked the header of the uploaded file.

The objective was to upload HTML content on the server, which could then be previewed by other users of the system. Based on observations, the team initiated the upload of a valid PNG file, intercepted the request in a proxy, and modified the image content by adding HTML content with a Javascript payload in the body of the PNG file. We left the PNG headers untouched and changed the file name from pic.png to pic. html (having the correct file extension was necessary in order for the client browser to execute it.) The application accepted the modified image content because the image header was valid and the file name was not being checked by the server. When navigating to the location of image, it caused the application to execute the Javascript payload embedded within the image — game over!

The endless possibilities of what an application can do with uploaded files, coupled with the negative actions that a malicious file may cause on the server or to other users of the application, likely ensures that it will be some time before automation can be programed to detect such complex file upload vulnerability defects.

TYPE OF DEFECT	MANUAL	AUTOMATED	ANALYSIS
User Enumeration	3	0	This defect is predominantly found in the login mechanism of applications - automation is very good at detecting them. However, when user enumeration is possible via other functionality, usually after a session has been established, it requires tweaking request parameters that automation may not recognize.
Vulnerable File Upload	3	0	Due to business rules that dictate what type of file uploads should be allowed and the large number of possibilities of what an application can do with user uploaded files, it is difficult for automation to differentiate between allowed types and malicious/non-malicious uploads. Even if automation is used, manually testing the outcome of an uploaded file is important.
Weak Session Management	5	0	Automation can detect weak session management for popular session management techniques and frameworks, but struggle to detect defects in highly customized or lesser-used session management implementations.
Reflected XSS & Stored XSS	13	2	The large discrepancy in the number of unique XSS defects was due to the complexity required in the test payloads. Automation was able to find simple XSS defects whereas manual testing was able to uncover more complicated defects.

Figure 6: defect types where the manual method of testing had a clear advantage over automation

CONCLUSION

This study examines real-world test scenarios to reaffirm the strengths and limitations of automated and manual testing, which include the following observations:

- When used and interpreted properly, automation provides a fast and efficient method for flagging common vulnerabilities. However, it yields only partial code coverage, cannot detect certain vulnerability classes, and can lose efficiency gains with an abundance of false positives.
- Humans can leverage expertise and creativity to hunt down complex, business logic, and other stealth-like vulnerabilities; however, this can be time consuming so they need to focus on critical areas.
- Different tools will find different types of vulnerabilities in different types of applications – ensure your teams know what their tools are capable of.
- Leveraging automated tools early on to identify low hanging fruit and hot spots provides a useful jumpstart for testers who can then determine where they need to dig deeper.

In the absence of unlimited time and resources, organizations must find the right mix of breadth (automation) and depth (manual attacks). This is driven by application complexity and risk. Harnessing the power of both automated and manual testing will optimize results, reduce false positives, and shorten the time necessary for remediation.