Paper 4 Summary

[Rui Zhao](https://unomaha.instructure.com/courses/43633/users/40845)

All Sections

No unread replies.1212 replies.

Search entries or author Filter replies by unreadUnread     Collapse replies Expand replies

[Subscribe](https://unomaha.instructure.com/courses/43633/discussion_topics/427921)

[Frank Tursi](https://unomaha.instructure.com/courses/43633/users/46026)

[**Frank Tursi**](https://unomaha.instructure.com/courses/43633/users/46026)

Apr 28, 2021Apr 28 at 8:05pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427921)

Frank Tursi  
CYBR 8410  
Paper Summary 4

This paper introduces a new analysis tool called QED that can automatically analyze and review Java web applications that are written to standard servlet specification, determine whether the application is vulnerable to XSS and/or SQL injections, and then provide examples of those attacks. A benefit of this tool is the lack of false positives as it tests each vulnerability before making an assertion and what makes it different from existing tools is that it is based on white-box analysis, requiring the source code to make determinations based on the logic of the program. The use of an intermediary language, PQL, allows this tool set to also be expandable in the future to include other application-level taint-based vulnerabilities. The authors suggest that this type of framework should become standard for other languages, to include testing, model checking, static analysis, and dynamic monitoring.

Edited by [Frank Tursi](https://unomaha.instructure.com/courses/43633/users/46026) on Apr 28 at 8:05pm

[Jensen Miller](https://unomaha.instructure.com/courses/43633/users/2767)

[**Jensen Miller**](https://unomaha.instructure.com/courses/43633/users/2767)

Apr 28, 2021Apr 28 at 8:06pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427921)

In this paper the authors investigate taint-type vulnerabilities in a few large scale web applications using a tool called QED (I still haven't found what that stands for yet). QED uses a context free PQL language that allows users to set what exactly what the tool should be looking for while scanning. I think that QED is a static analysis tool since they refer to it "simulating" execution of the application instead of something more dynamically worded. The most interesting thing about the tool is it's ability to generate or discard attacks depending on if they have a chance of succeeding. This is mostly based around where untrusted data can make it's way into the application as well as where it can affect SQL queries. QED does a rather good job of cutting down on purposeless tests since it is possible for it to make an infinite number of tests for each function that takes an external parameter and the results were good as well. I'm curious what the performance metrics were like.  Is QED something that could be run each time that code is checked into a repository on a larger project or is it something that is time and resource intensive. I would also like to know why QED is named like it is.

[Khedir Qassim](https://unomaha.instructure.com/courses/43633/users/102151)

[**Khedir Qassim**](https://unomaha.instructure.com/courses/43633/users/102151)

Apr 28, 2021Apr 28 at 11:49pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427921)

**Automatic Generation of XSS and SQL Injection Attacks with**

**Goal-Directed Model Checking**

The article presents a practical programmable technique that can automatically generate attacks for web-based application. The system called QED which automatically generates attack exploiting taint-based vulnerabilities such as cross-site scripting (XSS) and SQL injection in large class java web applications. This can provide better analyses results for specific vulnerability of interest code and make application developers fix errors and bugs in their programs. According to the article this is the first-time model checking has been used successfully on real life java to create attack sequences that consists more of than one HTTP request.

The QED system is build based on three design principles first, QED’s ability to find different vulnerabilities other than taint based. Second application framework used in this research Java servlets, JPS (java server pages) and Apache Struts. Third QED uses JPF or Java PathFinder model for system checking which automatically compiles user-supplied queries into static analyses for web application which later generates a set of input vectors. The article consists of seven parts that discuss more details on how the QED model enumerate attacks as explained Fig 4, two components need to be provided, PQL (queries specifying the vulnerability) and set of input values for any form parameter. The input application instrumented based on the PQL query which then combined with custom generated harness that will explore URL space requests. The harness application is fed to the model checker along with stub implementations of the application server’s environment. The results of this model checker will depend on URL sequences that shows attack path.

The article also shows result of practical example application of QED algorithm on three different application benchmarks PersonalBlog, JOrganizer and JGossip to detect XSS and SQL injection vulnerabilities. The example study showed total of 10 SQL injections and 13 XSS vulnerabilities in 130,000 line of code in these three programs. These three web applications illustrate various affects that researchers can get by using QED on each one for them and revealed that model checking for larger programs remains a challenge as showed in JGossip. In addition, it suggests that there are plenty of security risks using web applications. The article concluded how to combine techniques for the first time from three different approaches to generate powerful system by using sound, dynamic monitoring, and model checking.

Finally, QED can provide better result that traditional analyses because it does not generate any false positive warning, however the QED which a goal-directed model-checking is not feasible for large, real-life programs and it seems impossible to accept all possible input paths.

[Chris Schmitt](https://unomaha.instructure.com/courses/43633/users/36306)

[**Chris Schmitt**](https://unomaha.instructure.com/courses/43633/users/36306)

May 1, 2021May 1 at 9:57am

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427921)

Chris Schmitt

CSCI 8410

Paper Summary 4

              The article discusses the system called QED, which is a program that automatically finds attack vectors for numerous web vulnerabilities like XSS and SQL injection in web applications. As web applications are starting to become more prevalent XSS and SQL injection attacks have been on the rise. The QED system proposed by the authors is based on the concrete model of checking approach and is a verification technique based on the systematic exploration of a program's state space. The algorithm was designed to take a web application and a vulnerability specification and generate a set of attacks with corresponding execution traces to identify vulnerabilities.

              During their research, they tested their system against three different Strut-based web applications. Each application varied in the number of lines of code, classes, event handlers, and dependency pairs. They used QED to locate XSS and SQL injection vulnerabilities in each application. Each web application was different and provided a different set of challenges for their QED. The first one, PersonalBlog, QED was able to prove that there were no vulnerabilities other than the ones found. QED was also able to check URLs one at a time and prove that the events have no dependencies. The second program, JOrganizer, showed that for programs with a high number of dependencies, QED can greatly improve the effectiveness of model checking and provided ample coverage. QED checked all the sequences without repeated URLs. The last program they ran their code against was JGossip, which showed that QED struggled on large programs. Overall, their static analyzer proved useful for small to medium programs and they were able to combine techniques from the following three approaches: Sound and sophisticated program analysis, Dynamic Monitoring, and Model checking.

              Overall, the authors proposed and demonstrated the usefulness of their tool. They tested it against 3 other program that ranged in size and was able to identify potential issues when their tool goes live. For a static tool, it does seem useful in identifying XSS and SQL injections, but the authors did not address how their tool compared to tools like BurpSuite or OWASP. OWASP and BurpSuite are industry-leading tools that are designed to detect web vulnerabilities similar to QED. While the author's tool seemed useful, the paper did leave me wondering why they were proposing yet another vulnerability finding tool and they did not provide sufficient information on why.

[Jesse Hays](https://unomaha.instructure.com/courses/43633/users/11487)

[**Jesse Hays**](https://unomaha.instructure.com/courses/43633/users/11487)

May 1, 2021May 1 at 11:14am

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427921)

Jesse Hays

Rui Zhao

Distributed System Security

May 1, 2021

**Reading Summary**

Automatic Generation of XSS and SQL Injection Attacks with Goal-Directed Model Checking

                When it comes to XSS & SQL injections I can attest for this still being an issue. When I have SAST reviews with development teams this is still a common vulnerability found with our tools. When it comes to developers it feels like they just want to complete their required task and be done. They do not seem to ever have security in mind.

                QED claims to find all issues and have zero false positive. If this was true that would be a huge gain for everyone (except hackers and bug bounty hunters). We spend a huge amount of time tuning our queries and documenting why this finding or that findings is false positive. If audit were to see us marking false positives without proper documentation it would not be pretty. So to have a tool that would never find false positives and to have piece of mind knowing it has found all vulnerabilities would be an amazing tool to have.

[Jeff Smolinski](https://unomaha.instructure.com/courses/43633/users/79659)

[**Jeff Smolinski**](https://unomaha.instructure.com/courses/43633/users/79659)

May 3, 2021May 3 at 12:11pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427921)

In this work the authors propose a framework detecting SQL injection and XSS vulnerabilities within Java applications. Their methods utilize approaches which are still relevant today. First, the authors consider the effects of a session as a whole on an application rather than an individual request. Second, they analyze the data flow of these ‘taint’ vulnerabilities from untrusted input, to a sanitizer, and finally to a data sink. This process allows them to catch numerous vulnerabilities without producing false positives.

The authors claim that, “by abstracting away the guts of the (web) framework, we can concentrate on our model checker’s effort on the application code itself” (pg. 32). They also specify that no untrusted data from the taint source should pass critical security boundaries unless it has passed through a sanitizer. And yet they admit that all combinations of URL parameters could not be checked and that sanitizers are checked with short and simple strings. In other words, this tool, essentially, checks for implementation errors.

It seems web application frameworks have evolved since the time of this work. For example [Django (Links to an external site.)](https://django-book.readthedocs.io/en/latest/chapter20.html#the-solution), an extremely popular Python application framework, escapes SQL injection attacks automatically unless one overrides the default behavior. Similarly, React JS does the same thing regarding XSS attacks unless one uses the [dangerouslySetInnerHTML (Links to an external site.)](https://reactjs.org/docs/dom-elements.html" \l "dangerouslysetinnerhtml" \t "_blank) attribute. In other words, **sanitizing input and output** is now largely abstracted from the user.

Web application penetration techniques have also evolved to address these changes by introducing new XSS and SQLI payloads called polyglots. These are lengthy, uncouth strings that are designed to bypass both web application firewall detection and break out of a built-in sanitizer’s context. See this [link (Links to an external site.)](https://github.com/swisskyrepo/PayloadsAllTheThings) for examples of these payloads.

There will always be a need to detect vulnerabilities caused by implementation errors. However, modern web application frameworks remove numerous opportunities for the very errors this tool hopes to catch by default unless the behavior is specifically overridden. Furthermore, this tool is not designed to introduce polyglot payloads mentioned above which are necessary, in most cases today, to introduce XSS or SQLI vulnerabilities today.

In conclusion, this tool was probably extremely useful when it was first written, but its usefulness may be largely superseded by the sanitization strategies found in today’s web application frameworks which the authors specifically presume to be effective *see*page 34 rule 4: “The previous rule does not apply if the object has been passed through one of several sanitizers that quote or escape the content of the object.”

[Xavier McCaig](https://unomaha.instructure.com/courses/43633/users/17725)

[**Xavier McCaig**](https://unomaha.instructure.com/courses/43633/users/17725)

May 3, 2021May 3 at 1:04pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427921)

Xavier McCaig

CYBR8410

Paper 4

3 May 2021

This work present a tool, QED, which seeks to automatically generate SQL and XSS injection attacks for large-scale java applications. The authors allege that QED provides improved results over traditional analysis systems, as the tool presents no false positive.

QED uses an approach known as "concrete model checking," whereby a systematic exploration of a program's state space is  undergone to provide a foundation for verifications. Further, the authors present a technique known as "goal-directed model checking," which essentially allows for optimizations which reduce the overall search space for finding potential attack vectors.

Overall, the authors present a tool which appears to be efficient, but did not test against some more well-known modern tools of similar utility. Further, the tool is significantly hampered in utility by modern frameworks which check against most of what the tool is checking against, leading a modern reader to believe QED may no longer be a tool of choice.

[Andrew Storms](https://unomaha.instructure.com/courses/43633/users/105646)

[**Andrew Storms**](https://unomaha.instructure.com/courses/43633/users/105646)

May 3, 2021May 3 at 4:27pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427921)

Andrew Storms

CYBR 8410

Paper 4

The article I found details the idea of global norms in regards to cybersecurity. Essentially, it breaks down the idea that teach individual country beyond the United States has its own standards and procedures when it comes to cybersecurity, some better than others. The idea is that all the nations of the world need to come together to set global standards when it comes to how cybersecurity is addressed. If a consensus is made on this, the nations could simply develop and maintain one set of practices for dealing with cyber threats, instead of having a sporadic number of them with varying levels of use across the globe.

The failed reality of the situation is that since the types of security for nations across the board varies so much, it often we will see gaps in the levels of care, especially those impoverished nations that may not have the resources to adequately protect infrastructure of certain attacks. When this fails, we continue to see the recent events of data breaches and ransomware attacks that continue to plague the nation and the world. This has continued to be a growing concern within the healthcare scope, as facilities continue to grapple with the severity of Covid-19. Now, the fear is that future threat of breaches will discontinue facilities from providing adequate care to patients and that patient information will be used to demoralize or financially run important organizations.

Standards and practices for these types of situations need to be set ahead of time, not after a situation or breach has already occurred. I have mentioned before and seen in many articles. It comes down to the difference between being proactive to a situation versus being reactive. More often than not, businesses are reactive and only implement change after an error has occurred. Being proactive will substantially minimize risk.

[Zidong Liu](https://unomaha.instructure.com/courses/43633/users/86488)

[**Zidong Liu**](https://unomaha.instructure.com/courses/43633/users/86488)

May 4, 2021May 4 at 1:50pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427921)

Zidong Liu

CYBR 8410  
Paper Summary 4

This article introduces a practical programmable technology that can automatically generate attacks for Web-based applications. The system called QED uses taint-based vulnerabilities to automatically generate attacks. These vulnerabilities include cross-site scripting (XSS) and SQL injection in large Java web applications. This can provide better analysis results for specific vulnerabilities in the code of interest and enable application developers to fix bugs and errors in the program. According to the introduction of this article, this is the first time that model checking has been successfully used in real-life Java to create an attack sequence containing multiple HTTP requests.

The QED system is first constructed based on three design principles, that is, QED can find other vulnerabilities besides peculiar smell. The second application framework used in this study is Java servlet, JPS (Java Server Pages) and Apache Struts. Third QED uses the JPF or Java PathFinder model to perform system checks. The model automatically compiles user-provided queries into static analysis for web applications, which then generates a set of input vectors. QED is a static analysis tool because they call it "simulating" application execution, rather than using more dynamic language. The most interesting thing about this tool is that it can generate or discard attacks based on whether the attack has a chance of success. This is mainly based on where untrusted data can enter the application and where it may affect SQL queries.

This article also shows the actual sample application results of the QED algorithm on three different application benchmarks PersonalBlog, JOrganizer and JGossip to detect XSS and SQL injection vulnerabilities. The sample research shows that in the 130,000 lines of code in these three programs, there are a total of 10 SQL injections and 13 XSS vulnerabilities. Another point is that QED does not produce any false positives, so it can provide better results than traditional analysis, but QED's goal-oriented model checking is not suitable for large, real programs, and seems to be unable to accept all possible input paths .

[John Kieran](https://unomaha.instructure.com/courses/43633/users/6118)

[**John Kieran**](https://unomaha.instructure.com/courses/43633/users/6118)

May 5, 2021May 5 at 3:56pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427921)

This paper describes an automated XSS vulnerability scanning framework known as QED.  Written primarily for Java applications, QED claims to be highly accurate with few/no false positives even on large projects of roughly 130,000 lines of code.  Without limited web programming knowledge, it was difficult to understand the usefulness of the entire project.

The authors did a suitable job trying to articulate all the requirements for QED.  However, I noted repeated instances where care had be taken when dealing with QED.  For example, in section 2.3, it seems the program can only scan for vulnerabilities of a certain type and pattern.  The authors even make a specific note saying a mis-configuration may lead to erroneous results: "Care must be taken when developing the specification—missing a propagator may lead to false negatives in the final result,  while missing sanitizers is likely to lead to many false positives."

QED lists other limitations with regard to dependency cycles which could further limit its effectiveness: "If the dependency relation is cyclic, there will be a countably infinite number of possible candidates to test. To keep the test sequence finite, we restrict our sequences to only call any given entry point once."

Its usefulness can't be completely understated, of course.  Automated scanning techniques are crucial when dealing with large amounts of code.  However, improvements in code libraries might make such techniques redundant over time.  Maybe in the future, AI-fueled techniques will perform the same action as QED with fewer limitations while still maintaining the high accuracy.

[Stevie Siy](https://unomaha.instructure.com/courses/43633/users/29318)

[**Stevie Siy**](https://unomaha.instructure.com/courses/43633/users/29318)

May 5, 2021May 5 at 4:08pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427921)

Paper 4 Summary

This paper details the program called QED, which was designed to analyze Java web applications and find possible attack vectors for cross-site scripting and SQL injection through goal-directed model checking. My first reaction after reading was that this is not any technology that is unprecedented, but it seems this paper was written earlier this century. However, it still presents a relevant technology as it includes the system that double-checks and can prove to lack false-positive results. QED is based on three main approaches: program analysis, dynamic modeling, and model checking. These steps allow QED to find as many possible attack vectors as possible and then filter them down to guarantee no false positives remain.  It is based on taint-based vulnerabilities due to use of servlets, JSPs, or Struts. All in all, the creators seem to have accomplished their goal of being able to persuade developers that there are flaws within the web applications that are being produced, and it is worth using a tool like QED to take a second look.

[Alexander Bladow](https://unomaha.instructure.com/courses/43633/users/37067)

[**Alexander Bladow**](https://unomaha.instructure.com/courses/43633/users/37067)

May 5, 2021May 5 at 7:29pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427921)

Alexander Bladow

CSCI 8410

Paper summary 4

This paper is over QED, which is a goal-directed model-checking system that automatically generates attacks exploiting taint-based vulnerabilities in large java web applications. They give a short explanation of XSS and SQL injection attacks then move onto an overview of what QED is, how it works, and the contributions that this paper puts forward. It keeps its mission statement and organization short and then goes on to define its methodology for how QED operates along with how they tested it and expanded on the concept of taint vulnerabilities. I have personally never heard of these types of attacks being referred to as taint-based vulnerabilities thou I can see why they decided to do so. The vulnerabilities that it exploits fit within four main patterns which specify the source of the tainted data, what the data accesses, and how the data is processed. It subsequently goes on the explain how its input is generated. How it does PQL instrumentation and Matching, generates page requests, and how Parameters are sent to event handlers. Once all the input has been entered and processed QED then filters the final events which include eliminating redundant URL sequences, removing repetitive cycles, and statically eliminating sequences. This filtering is quite helpful due to the fact that it is cutting down on the amount of data a person has to process to receive the desired data set. After they finished explaining how QED works they show three examples from their experimental results, which are the program PersonalBlog, JOrganizer, and JGossip which vary in popularity, complexity, and usages. Their results showed that the less complex a given program is then the fewer vulnerabilities that QED detects. In conclusion, QED is a tool that can combine sound, sophisticated analysis, dynamic monitoring, and model checking to create a system that can locate a significant number of Taint-based vulnerabilities.