Paper 1 Summary

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

All Sections

No unread replies.1111 replies.

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

[Subscribe](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)

[Collapse Subdiscussion](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)[Jesse Hays](https://unomaha.instructure.com/courses/43633/users/11487)

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

Feb 15, 2021Feb 15 at 3:54pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)

Jesse Hays

Rui Zhao

Distributed System Security

15 Feb 2021

**Reading Summary**

CyptoGuard: High Precision Detection of Cryptographic Vulnerabilities in Massive-sized Java Projects

                In the abstract this paper mentions that it had 98.61% true positives. While this number is impressive, it could be very misleading. What this says is if they find something, it is with a high likelihood it is an actual finding. For all we know as the reader, this project could have had 50,000 findings. This tool only found 1,295 findings, thus rendering this tool not a viable option.

The google play automatic app may not be as in depth as one would like but it is a start. If you look at most software companies, the product they had at the beginning is no where near the level that product is at in current time. If this google play app got enough backing and funding, I do believe it could amount to something. It is also healthy for products such as CryptoGaurd to have competition. This helps keep cost down, and ingenuity up.

According to this paper, 95% of the total vulnerabilities come from libraries that are packaged with application code. This is partly what I do for my job. I work with SAST, DAST, and SCA scans. These findings would be known as SCA, software composition analysis. I do believe this number as the number of vulnerabilities in public libraries is astounding, even if you keep your versions up to date.

A huge comparison for CG is the runtime summary on table 6. CG completed all it’s tasks while CrySL exited prematurely 18 times. As someone who uses Checkmarx and deals with its issues with SCA/OSA, that is very attractive in a tool.

Towards the end of the paper, it does discuss the number of false negatives (blind findings) which came out to be 11. How important this is depending on the severity of those vulnerabilities. I believe 11 blind vulnerabilities should always be a concern, but it could be even more concerning if those were all high vulnerabilities. Every single tool I have used has its pros and cons. To say any tool can find everything would be a naïve statement. However, in my own opinion I would rather a tool create a lot of noise (false positives) than to not create a lot of noise and potentially miss any true vulnerabilities (false negatives). You can always look through a lot of data, however you cannot look through all this data to find a vulnerability that your tool did not (if you aren’t manually pen testing.)

CryptoGuard seems like a very feasible tool that one could add to their tech stack depending on the price. I think a big lesson from this paper for students who do not work in the Application Security field is that these public libraries that so many people use are a lot less safe than one may think. You need to worry about security not only in the code you write but anything you import as well.

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

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

Feb 20, 2021Feb 20 at 3:54pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)

Regarding the automated code analysis, it is a big challenge to achieve both sound and complete (if you are interested in these two terms, please google "program analysis + soundness + completeness").

Regarding Table 6, the authors did not provide too much information about how they handle/avoid or even address the exceptions that occurred in CrlSL. Being able to analyze a particular type of code could be quite different from safely handling the corresponding exception thrown out when analyzing it.

Regarding the false negatives, no one dares to say his/her tool can handle everything in the code; therefore, there must be the FNs. This is related to the soundness. When we use such a tool, sounds to me, I would not expect it will produce really a lot of FPs; think about that in the anti-virus software. But of course we sure don't want to miss any severe vulnerability.

I like your comment "these public libraries that so many people use are a lot less safe than one may think."

[Collapse Subdiscussion](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)[Chris Schmitt](https://unomaha.instructure.com/courses/43633/users/36306)

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

Feb 15, 2021Feb 15 at 4:10pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)

Chris Schmitt

CSCI 8410

**Reading Summary**

CyptoGuard: High Precision Detection of Cryptographic Vulnerabilities in Massive-sized Java Projects

The paper discusses the misuses of cryptographic APIs and how common they are amongst programs. Cryptographic APIs are used in a variety of applications and often prove difficult to test due to them being used in code that can contain over a million lines of code. The authors designed a tool, CryptoGuard, which aimed at reducing the number of false positives without reducing the quality of the analysis. The authors focused primarily on Java projects and Android Applications from the Google Play Store in their analysis of their tool. They set out and built their threat model so the tool can address these issues in the following categories: Vulnerabilities due to predictable secretes Man in the Middle (MitM) attacks on SSL/TLS, predictable pseudorandom number generators (PRNGs), chosen-plaintext attacks (CAP), and brute force attacks. They prioritized the vulnerabilities discovered into 3 categories (high, medium, and low) based on the data available to an attacker and ease of access.

CryptoGuard set out to solve several technical challenges that other Cryptographic API scanners struggled with the first being false positives and the second being the precision vs runtime tradeoff. False positives pose an issue due to typically having to be manually reviewed. CryptoGuard set out to eliminate false positives due to phantom methods, methods with nobody present in the source code, and false positives due to data structures. Their solution was to implement a set of crypto-specific methods and refine the slicing output which helps them analyze the “phantom method.” False positives due to “bookkeeping”, hard-coded key constants, are tracked and discarded when needed. Standard cryptographic tools typically will trade precision for runtime performance, especially when it comes to code with millions of lines. The author's tool was designed to set the depth of orthogonal exploration to 1, focus on demand-driven analysis, exclude unnecessary subprojects, and analyze independent ones concurrently.

The remainder of the paper went over their “proof-of-concept” of their Application and what they tested their product on. They proposed several experimental questions they wanted to answer and began testing on 46 Apache projects and 6,181 popular Android applications that use Crypto APIs. They developed a benchmark using CryptoAPI-Bench, which covered their 16 cryptographic rules and compared CryptoGuard to CrySL, Coverity, and SpotBugs (tools currently in use by the community).  They found that their tool performed better than the others and reported more API misuse cases than Google’s built-in screening program.

On top of their analysis, the authors also listed CryptoGuards' shortcomings and discussed their goals for the future.  They provided a detailed analysis of what they found using CryptoGuard and the results of their issue request that they submitted for their tools findings. Overall, they provided a solid non-biased analysis of their tool and compared it to other industry-leading tools. They identified their tools' shortcomings and how they intend to address them and provided a brief overview of the other cryptographic tools used in their paper.

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

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

Feb 20, 2021Feb 20 at 4:03pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)

You brought this up, "set the depth of orthogonal exploration to 1"! This is a big limitation of this paper. In other words, CryptoGuard's analysis scope is very restricted - it is not able to track the value flow in the whole program, thus not being able to identify how a single value can affect the security of the entire program.

Of course, if we really want to eliminate such a restriction, we surely can! But depending on the methodology, it can also be very expensive. So I guess it is the tradeoff the authors had made in this paper.

[Collapse Subdiscussion](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)[Jeff Smolinski](https://unomaha.instructure.com/courses/43633/users/79659)

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

Feb 15, 2021Feb 15 at 5:58pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)

The CryptoGuard project is static software analysis tools similar to CrySL, Coverity, and SpotBugs. The authors’ primary motivation for CryptoGuard was to create a static analysis tool that could identify common cryptographic implementation errors in large Java projects quickly with both a low rate of false positives and high accuracy.

In their analysis of the problem space the authors submit that the code related to cryptography makes up a very small percentage of the overall project. They also cite numerous problems faced by the state of the art causing numerous false positives. There are three causes that stood out which the authors addressed. First, phantom methods which are methods whose method bodies are not available during analysis. Second, identifying false positives in data structures like referencing an element of an array. Third, identifying orthogonal methods (getters) and treating them differently than hardcoded keys. Addressing these and other pervasive issues yielded positive results.

Through intelligent program slicing rules and tuning CryptoGuard offers impressive performance. With an average runtime of 3.2 minutes, a reduction in false positives by 76% - 80%, and a precision rate of 98.21%. This is a massive improvement when compared to other tools in this space.

I would, however, have liked the authors address two more things concerning their reported eighteen false negatives. First, they did not mention the severity rating of the false negative. Second, they did not mention if the other tools they tested alerted on those issues.

This work discovered numerous vulnerabilities with 1,295 vulnerabilities reported in 46 Apache projects and 130,845 vulnerabilities reported in 6,181 Android apps. I submit this is, unfortunately, not surprising due to both the stark configuration differences between a typical development and production environment and the complications associated with implementing cryptography itself.

Some of the most common errors like using dummy hostname verifiers and truststores; SSL/TLS misuses; hardcoded secrets, salts, and IVS; and improper secret management could be attributed, in part, to artifacts from a development environment that made their way into production without the proper checks.

Using a tool like cryptoguard would be a great addition to a CI/CD pipeline so issues like this do not make it into production in the first place. Additionally, developers could receive almost immediate feedback regarding the correctness of their cryptographic implementation. It also seems like this tool could be extended to search for other common software vulnerability patterns, making it even more comprehensive.

Lastly, a security researcher or threat actor with a fast, high precision tool like this could hone in on legitimate cryptographic weaknesses quickly; leaving them more time to write an exploit.

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

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

Feb 20, 2021Feb 20 at 4:14pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)

Alright, here would bring up two points.

Program slicing: as this paper had heavily used this technique, its performance actually astonished me! Why? Because program slicing is a very naive approach to analyze the source code. It is completely not comparable to the data flow analysis which is based on the control flow of the program and the program slicing. There are really lots of limitations on the program slicing; for example, it cannot handle the reuse of variables, it is just based on the naming presented on the AST of the code. So, I guess if the authors run the CryptoGuard on much much more applications, there will be extremely lots of false negatives. Meanwhile, in the paper, they did not actually prove anything regarding the false negatives -  it is hard sometimes except we manually go through all the analyzed code, right?

The detection rules: in the program analysis there is not a universal solution to define and formulate everything. Usually, the formulation of one particular problem is very unique from others. However, I don't believe that can be the case for the crypto-api misuse detection. And still there are lots of other misuse patterns can be formulated and those patterns can actually be universally formulated to certain extent.

[Collapse Subdiscussion](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)[Robert Ernewein](https://unomaha.instructure.com/courses/43633/users/31037)

[**Robert Ernewein**](https://unomaha.instructure.com/courses/43633/users/31037)

Feb 16, 2021Feb 16 at 12:58am

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)

I began reading this paper and taking notes to the extent of creating a presentation then back off to look at the big picture. There are innumerable vulnerabilities in production code, but concentrating on the low hanging fruit should always be the starting point. Just running the numbers 1277/1295 is 98.61%, or 1277/46 = 27.75 true positives per Apache project. With CryptoGuard being a static analysis tool with a narrow scope of security misuse cases, there are far too many true positives even if some are interrelated. There is a reason why Apache Web Server resides on a Kali Linux image. It's hard not to find a vulnerability. I'm glad they focused on the Apache side of the house for most of the paper. What I have seen from Android apps makes me happy I don't own a smart phone.

Threat models given in section 2.1 are hacker 101. Easy to find, easy to exploit.

1. Vulnerabilities due to predictable secrets: Hard-coding defaults passwords/keys and/or deriving keys, salts, and iterations from them.

2. Vulnerabilities from MitM (SSL/TLS): dummy host and certificate verification.

3. Vulnerabilities from PRNGs: Simple not implementing secure random number generation.

4. Vulnerabilities from CPA: Default Passwords, storing plaintext and ciphertext together.

5. Vulnerabilities from Feasible brute force attacks: Known default (hard-coded) Username/Password combination & Poor/No strong password enforcement.

Common causes: Code reuse, failure to purge test code from production releases, failure to maintain secure encryption algorithms (OWASP) & lack of training.

Implementation of Refinement Insights (RI)s helps alleviate false positives and processing overhead, but I'm fuzzy on

RI-I: The Removal of state indicators: Variable status/boolean value? and

RI-V: Removal of constants in infeasible paths: What constitutes an infeasible path?

The vulnerabilities in the production code snippets were glaring. Ranger was worse than I expected, but I think the security gaps in the "Financial" applications, would catch most people by surprise. There is an expectation of security with such apps, but most people are unaware that CNN Money was still an HTTP site until a couple of years ago.

I've recently read some related papers, so I was not surprised by libraries being the source of 95% of Android vulnerabilities, mostly 3rd party libraries that they have no control over.

I like the criteria for inter process back-slicing, although I'm not sure if the recursive algorithm is limited by the same depth counter, or if modified to look at least as deep as the initial trigger variable.

The authors head-to-head comparisons with other application were based on common capabilities/rules, they acknowledge the limitations inherent in Static Analysis Tools, and have direction for further development.

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

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

Feb 20, 2021Feb 20 at 4:22pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)

"RI-I: The Removal of state indicators: Variable status/boolean value? and

RI-V: Removal of constants in infeasible paths: What constitutes an infeasible path?"

-- I guess it may because the authors wanted to trade the accuracy for the runtime performance. But same as you: unfortunately, they didn't prove how they can tell if this/that is a infeasible path and even evaluate the performance of this part. So it could incur false negatives.

At last, I would say writing code might be easy, but writing high-quality code is never an easy job!

[Collapse Subdiscussion](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)[Rui Zhao](https://unomaha.instructure.com/courses/43633/users/40845)

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

Feb 20, 2021Feb 20 at 5:04pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)

Here are the comments that were brought up by other students in the class. The selected comments may have the overlap with yours so I removed the duplication of the comments.

"Context analysis must also be used to determine how methods and variables will access data based on scope."

"Including language-specific context optimization for FP reduction, on-demand flow-sensitive, context-sensitive and field-sensitive program segmentation, ..."

-- In program analysis, the data flow analysis can be context-sensitive or context-insensitive. And context info is always resolved during the control/data flow analysis but not the program slicing. So this is the part in the paper also confused me: if they had considered the program context, why not performed the data flow analysis which is much more beyond the slicing?

"This can be massively helpful for security researchers to find and patch vulnerabilities before an application is widely distributed."

-- The automated vulnerability analysis is in the big trend in the next few years maybe decades? But so far there is no paper can be called state-of-art on this research direction.

"CryptoGuard detects the existence of API abuse in the code base, but does not verify whether the vulnerable code will be triggered at runtime. This problem is a general limitation of static program analysis. "

-- That is true. The code execution paths that have been discovered by the static analysis may not exist actually or may never be triggered. In order to answer this particular question, we may want to leverage the symbolic execution.

"Their findings against Apache code alone is impressive, not to mention findings against companies such as Google which were otherwise mentioned in the testing portions of the document."

-- So far those big-name companies also use either static or dynamic analysis or their combination to check the source code of their hosted applications. However, their approach is not available to the public. The program analysis is not a kind of magic pill that can cure everything.

"The authors additionally provide an open research problem suggestion to design a compiler that automatically transforms a cryptographic vulnerability or rule into a static analysis based code screening algorithm with greater functionality and reliability than is found in preexisting tools".

-- It is related to one research problem: how we formulate the security questions so that the program analysis can directly work on, right? We of course know what is right what is wrong, but how to enable the computer think in the same/similar way as us is a big challenge.

"Furthermore, during further refinements on the analysis tool the authors discovered that they could cause false negatives since the tool would not scan deep enough to locate some vulnerabilities."

-- You are correct! All because of the limit on the depth.

"This experiment has the maximum LoC of 2,571,000 on Hadoop and the average LoC is 402,000 so to address this issue, large codes bases were modularized and divided into various sub projects."

-- So they need to think about those data flowing among those sub-projects, right? Actually I don't buy this point of the paper - by doing this, false negatives would be introduced; but to avoid false negatives, splitting the large code does not make any difference.

"In my opinion, CryptoGuard can persuade industry teams to start scanning the improving the security of their code now, plus inspiring researchers to employ its technique to begin developing other precise and scalable analysis tools."

-- Also there is another big and important task: how to educate people on writing code correctly, right? That's what we are doing here~

"Mainly the CryptoGuard program tackles implementation of cryptographic libraries and APIs, but the list of rules it checks limits the program from checking against the cryptographic libraries themselves."

-- Exactly! The detection capability of CryptoGuard really depends on the rules it relies on!

"Additionally the authors of the paper call out some of companies they disclosed the vulnerabilities to that they ignored their disclosures ..."

-- That happens, even to me in the past! Well so many factors can be there but I don't know as researchers what else they can do for it.

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

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

Feb 20, 2021Feb 20 at 5:11pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)

I'm a bit confused on what we are suppose to do because the Syllabus mentioned a "Blog" post of sorts for the Paper Summaries. Are we suppose to upload the paper to canvas and post it in this discussion board? Or just post it to canvas?

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

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

Feb 20, 2021Feb 20 at 5:14pm

[Manage Discussion Entry](https://unomaha.instructure.com/courses/43633/discussion_topics/427918)

Yes we are supposed to post summaries here. For the next paper summary, I will emphasize that please post it here, not necessary to upload it to the corresponding assignment.