APPENDIX D
ADDITIONAL DETAILS FOR THE ONLINE APPENDIX

A. Evaluation Data

TABLE IV
LIST OF APPLICATIONS MUTATED USING MASC, CLOC = COUNT LINES OF CODE FROM JAVA SOURCE FILES ONLY [96], SOURCE = SOURCE CODE COLLECTED FROM/ORIGINATED FROM

ID	Name	Type	Source	CLOC
A1	2048	Android	GitHub	136
A2	BMI Calculator	Android	GitHub	145
A3	Calendar Trigger	Android	GitHub	8,553
A4	LocationShare	Android	GitHub	215
A5	NasaApodCL	Android	GitHub	706
A6	AFH Downloader	Android	GitHub	1,657
A7	A Time Tracker	Android	GitHub	2,928
A8	Kaltura Device Info	Android	GitHub	1,049
A9	Protect Baby Monitor	Android	GitHub	625
A10	Activity Monitor	Android	GitHub	1,168
A11	personalDNSfilter	Android	GitHub	8,446
A12	aTalk	Android	GitHub	254,364
A13	Car Report	Android	BitBucket	16,966
	Apache Qpid™ Broker-J			
J14.1	Broker-J - AMQP/JDBC	Java	Apache	597
J14.2	Broker-J - Tools	Java	Apache	1,725
J14.3	Broker-J - HTTP	Java	Apache	24,141
J14.4	Broker-J - Core	Java	Apache	127,280

B. Taxonomy Data

Name of Collector: <Anonymized>
Source Type: Industry
Source URL: multiple
Title: OWASP
Types of Misuses:

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
- Using weak password or weakly hashed password

https://www.owasp.org/index.php/OWASP_Testing_Project 2014
- Bad cryptography - password using MD5 hash
- Using SSLV2 (page 160 in PDF)
- RSA/DSA key length < 1024 bits for X.509
- Using MD5 for signing X.509
- Keys not generated without proper entropy

https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet
- Using MD5
- Using SHA1
- Using ECB
- Using java.util.Random()
- Not using any version of TLS older than TLSv1.2
- Using SSLv2, SSLv3 (https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices)
- Using PRNG instead of CSPRNG (cryptographically secure pseudo random number generators)

Fig. 3. An example data collection form for extracting misuse cases from artifacts

APPENDIX E
ELABORATED IMPACT OF CRYPTO-DETECTOR FAILURES

The nine crypto-detectors that we evaluate are important in practice as they are potentially used by a considerable population of developers for analyzing their Java and Android apps. As a result, any flaws they exhibit have a serious impact on the security of software they analyze. We performed a study of publicly available applications to gauge this impact ($\mathbf{RQ}_3$) *after* investigating $\mathbf{RQ}_1$ and $\mathbf{RQ}_2$.

TABLE V
RELEVANCE OF CRYPTO-DETECTORS EVALUATED USING MASC

Tool	Practical Relevance	Deployment Use Case
CryptoGuard (Academia)	Used in Oracle's internal testing [19]	*...test the security of applications in Software Assurance Marketplace* [3]
CogniCrypt (Academia)	Available as an Eclipse Plugin [18]	*...ensure that all usages of cryptographic APIs remain secure* [33]
*Xanitizer** (Industry)	Integrated into Github Code Scan [21]	*...is the essential tool for security auditors of web applications* [12]
*Coverity** (Industry)	Used by federal government agencies [13]	*...ensure compliance with security and coding standards* [13]
SpotBugs/FindSec-Bugs (Open Source)	Used by tools such as SonarQube, Xanitizer, ShiftLeft [11]	*The SpotBugs plugin for security audits of Java web applications and Android applications* [32]
QARK by LinkedIn (Industry)	Developed by LinkedIn [22] and promoted in security books [23]–[25]	*An Auditing and Attack Framework* [97], *...to recognize potential security vulnerabilities and points of concern...* [22]
LGTM (Industry)	Used in industry (*e.g.,* by Microsoft, Google) [17]	*...variant analysis platform that automatically checks your code for real CVEs and vulnerabilities* [98]
GCS (Industry)	Default offering of Github's Code-Scan Suite [99]	*...analyze the code in a GitHub repository to find security vulnerabilities and coding errors* [99]
ShiftLeft Scan (Industry)	Available in Github Code Scan integration [16]	*...performing static analysis based security testing of your applications and its dependencies* [100]

*We obtained full licenses for these proprietary tools.

TABLE VI
SELECTED SOURCES FOR EXTRACTING CRYPTO MISUSE FROM INDUSTRY

ID	Tool/Guideline
A1	Mobile Security Framework [101]
A2	KryptoWire [102]
A3	Open Web Application Security Project (OWASP) [57]
A4	SpotBugs with FindSecBugs [11]
A5	Xanitizer [12]

A. Presence of Misuse in Real-World Applications

Our first goal is to determine if the misuse instances generated by MASC that led flaws in crypto-detectors are also found in real-world apps. For this purpose, we used the GitHub Code Search [66] service to search within public repositories for crypto-APIs that are commonly misused, and further manually explored the search results to identify misuse instances similar to the ones MASC generates. Additionally, we manually searched Stack Overflow [67] and Cryptography Stack Exchange [68] for keywords such as "unsafe hostnameverifier"

TABLE VII
KEYWORDS USED FOR SEARCHING ARTIFACTS

Classes	Words Used
General Cryptography	cryptographic, cryptography, crypto, cipher, security, encrypt, encryption, vulnerabilities, attack, digest
SSL / TLS	SSL, TLS, certificate, key(s), signing
Crypto Misuse	Android, API, library, implementations, software, misuse

TABLE VIII
SELECTED SOURCES FOR EXTRACTING CRYPTOGRAPHY MISUSE, FROM ACADEMIA

ID	Title	Year	Venue
1	Why Eve and Mallory love Android: an analysis of Android SSL (in)security [2]	2012	CCS
2	The Most Dangerous Code in the World: Validating SSL Certificates in Non-browser Software [103]	2012	CCS
3	An empirical study of cryptographic misuse in Android applications [10]	2013	CCS
4	Using Frankencerts for Automated Adversarial Testing of Certificate Validation in SSL/TLS Implementations [104]	2014	S&P
5	Modeling Analysis and Auto-detection of Cryptographic Misuse in Android Applications [105]	2014	ICDAS
6	Smv-hunter: Large scale, automated detection of SSL/TLS man-in-the-middle vulnerabilities in Android apps [4]	2014	NDSS
7	Securing Android: A Survey, Taxonomy, and Challenges [106]	2015	CSUR
8	Automatically Detecting SSL Error-Handling Vulnerabilities in Hybrid Mobile Web Apps [6]	2015	SICC
9	Evaluation of Cryptography Usage in Android Applications [107]	2016	ICBICT
10	Jumping Through Hoops: Why Do Java Developers Struggle with Cryptography APIs? [108]	2016	ICSE
11	Taxonomy of SSL/TLS Attacks [54]	2016	IJCNIS
12	CogniCrypt: Supporting Developers in Using Cryptography [33]	2017	ASE
13	CrySL: An Extensible Approach to Validating the Correct Usage of Cryptographic APIs [9]	2018	ECOOP
14	A Surfeit of SSH Cipher Suites [109]	2016	CCS
15	iCryptoTracer: Dynamic Analysis on Misuse of Cryptography Functions in iOS Applications [110]	2014	NSS
16	IV = 0 Security: Cryptographic Misuse of Libraries [111]	2014	MIT
17	Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0 [112]	1999	USENIX
18	All your Droid are belong to us: A survey of current Android attacks [113]	2015	USENIX
19	A Study of Android Application Security [114]	2011	USENIX
20	SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements [115]	2013	S&P
21	Mining your Ps and Qs: Detection of widespread weak keys in network devices [116]	2012	USENIX
22	MUBench: A Benchmark for API-Misuse Detectors [117]	2016	MSR
23	Lessons Learned in Implementing and Deploying Crypto Software [118]	2013	USENIX
25	CDRep: Automatic Repair of Cryptographic Misuses in Android Applications [119]	2016	ACCS
26	Mining Cryptography Misuse in Online Forums [53]	2016	QRS-C
27	Rethinking SSL Development in an Appified World [120]	2013	CCS
28	Towards secure integration of cryptographic software [121]	2015	Onward!
29	Are Code Examples on an Online QA Forum Reliable?: A Study of API Misuse on Stack Overflow [122]	2018	ICSE
30	Program Analysis of Cryptographic Implementations for Security [123]	2017	SecDev
31	Exposing Library API Misuses via Mutation Analysis [49]	2019	ICSE
32	How Reliable is the Crowdsourced Knowledge of Security Implementation? [124]	2019	ICSE
33	Stack Overflow Considered Harmful? The Impact of Copy&Paste on Android Application Security [88]	2017	S&P
34	CRYPTOGUARD: High Precision Detection of Cryptographic Vulnerabilities in Massive-sized Java Projects [3]	2019	CCS
35	Understanding How to Use Static Analysis Tools for Detecting Cryptography Misuse in Software [52]	2019	ToR

and "unsafe x509trustmanager" to find attempts by developers to intentionally create vulnerable code.

We found several instances where `MASC's` misuse instances were either exactly represented in public repositories, or with minor variations that could easily be realized using `MASC's` existing operators. For example, consider Apache Druid [125], an application with $10.3K$ stars and 400 contributors on GitHub. Apache Druid uses `AES` in `CBC` mode with `PKCS5Padding`, a configuration that is known to be a misuse [28], [88], as shown in Listing 18:

```
this.name = name == null ? "AES" : name;
this.mode = mode == null ? "CBC" : mode;
this.pad = pad == null ? "PKCS5Padding" : pad;
this.string = StringUtils.format(
    "%s/%s/%s", this.name, this.mode, this.pad);
```

Listing 18. Transformation String formation in Apache Druid

This instance uses ternary operators to initialize the variable values which it then passes on to the `Cipher.getInstance(<parameter>)` API, which is a misuse instance similar to the one that led to Flaw **F2** (instantiated using $\mathbf{OP}_2$). Similarly, we found real apps that convert the case of algorithm values before using them in a restrictive crypto API [71] (**F5**, instantiated using $\mathbf{OP}_3$), or process values to replace "noise" characters [72] (**F6**, instantiated using $\mathbf{OP}_4$).

Further, we found an exact instance of the misuse representing **F10** in the appropriately named utility class `TrustAllSSLSocketFactory` in the repository of Apache JMeter [80] (4.7K stars in GitHub). **F11** is the generic version of **F10**, which is fairly common in repositories and libraries (*e.g.,* BountyCastle [81], [82]).

We also observed that ExoPlayer, an open source media player from Google with over 16.8K stars on GitHub, used the predictable *Random* API for creating `IvParameterSpec` objects [73], until 2019, which is an instance of the "Bad Derivation of IV" misuse from Fig. 2, instantiated using $\mathbf{OP}_6$, and similar in nature to **F9**. Similarly, developers often use constants for IV in a manner similar to **F8**, as seen in UltimateAndroid [74] (2.1K stars), and JeeSuite (570 stars) on GitHub. Thus, even though developers of open source repositories are likely to be benign (*i.e.,*$\mathbf{T}1/\mathbf{T}2$), we still found them misusing crypto APIs in ways similar to the mutants generated by `MASC`, which not only attests to the practicality of our approach, but also means that *flawed crypto-detectors may be unable to detect vulnerabilities in real applications*.

Finally, we found questions on StackOverflow where developers were purposefully attempting to *evade* detection/warnings (*i.e.,* **T**3), generally to get their app accepted to Google Play. Particularly, in this post [36] from StackOverflow, the developer

describes several ways in which they tried to get Google Play to accept their faulty `TrustManager` implementation, one of which is exactly the same as the misuse instance that led to **F17** (generated using a combination of $\mathbf{OP}_7$ and $\mathbf{OP}_{12}$, as shown in Listing 10 in the Appendix. We observed similar evasive attempts aimed at using vulnerable hostname verification [83], in a manner that can be instantiated using $\mathbf{OP}_8$ and $\mathbf{OP}_{10}$) (and similar in nature to **F15** and **F16**). We also found developers attempting to evade detection by applying context-specific conditions in the hostname verifier [84], which is partially similar to **F18** and **F19**.

B. Presence of Misuse in Apps Scanned by Crypto-Detectors

Our second goal is to gauge the impact of flaws in crypto-detectors through real-world apps that were analyzed with them. Particularly, of the nine crypto-detectors we analyze, only LGTM showcases the open-source repositories that it scans (*i.e.,* the developers/repos that use LGTM). We manually explored the top 11 Java repositories from this set (prioritized by the number of contributors), and discovered several misuse instances that LGTM tool may have failed to detect.

For example, we found that in Apache Ignite [76] (360 contributors, 3.5K stars) contains a misuse instance similar to one that led to **F2**. In this instance, only the name of the cipher is passed to the `Cipher.getInstance(<parameter>)` API [77] which causes it to always default to "ECB" mode, which is not secure. This, however, is not reported by LGTM. We found similar instances of the `ECB` misuse in Apache-Hive [78] (250 contributors, 3.4K stars). Based on our exploratory study of default rules used by LGTM, `ECB` is not considered insecure *at all* for Java, although it is considered insecure for JavaScript and CSharp [126], [127]. As a result, LGTM does not report similar misuse we found in popular community contributed repositories such as Azure SDK for Java [79] (250 contributors) at around 15 code locations in 12 different classes(*e.g.,* [128], [129]),

C. Impact of FC0: CryptoGuard and CogniCrypt

We now describe the impact of the two flaws that form our preliminary flaw class **FC0** (*i.e.,* one not detected through `MASC's` mutants), described at the beginning of Section X, *i.e.,* (1) exempting classes containing the string `android.` (exhibited by CryptoGuard), and (2) not supporting multidex (exhibited by CryptoGuard, QARK, and CogniCrypt). For this purpose, we obtained the list of $6,181$ Android apps used to evaluate CryptoGuard [3] (*i.e.,* as the authors were unable to provide us with the actual APKs), and downloaded $4,353$ out of the $6,181$ that were available on Google Play, to analyze for both characteristics, *i.e.,* `android.` and multidex.

We found that 673 (*i.e.,* 10%) apps have `android.` in at least one of the fully qualified class names, which CryptoGuard would incorrectly exempt from analysis. Similarly, we found that $2,709$, or 62.23% of the apps use multidex, which neither of CryptoGuard, QARK, and CogniCrypt would be able to analyze, although multidex is supported by all other crypto-detectors in our set. That is, CryptoGuard be unable to completely analyze at least 62.23% of the set of apps used in its original evaluation, which indicates the severity of even trivial implementation flaws on the security guarantees provided by a crypto-detector.