ESORICS

SaTC-1929701

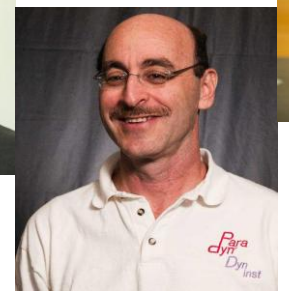
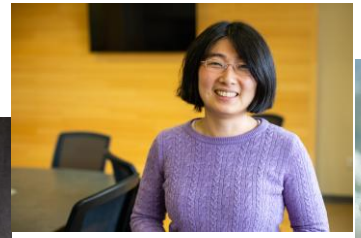
Tutorial: Principles and Practices of Secure Cryptographic Coding in Java

Ya Xiao, Miles Frantz, Sharmin Afrose
Ph.D. students
Virginia Tech

Daphne Yao
Professor
Virginia Tech






Software is everywhere

Ford GT has over 10 million lines of code

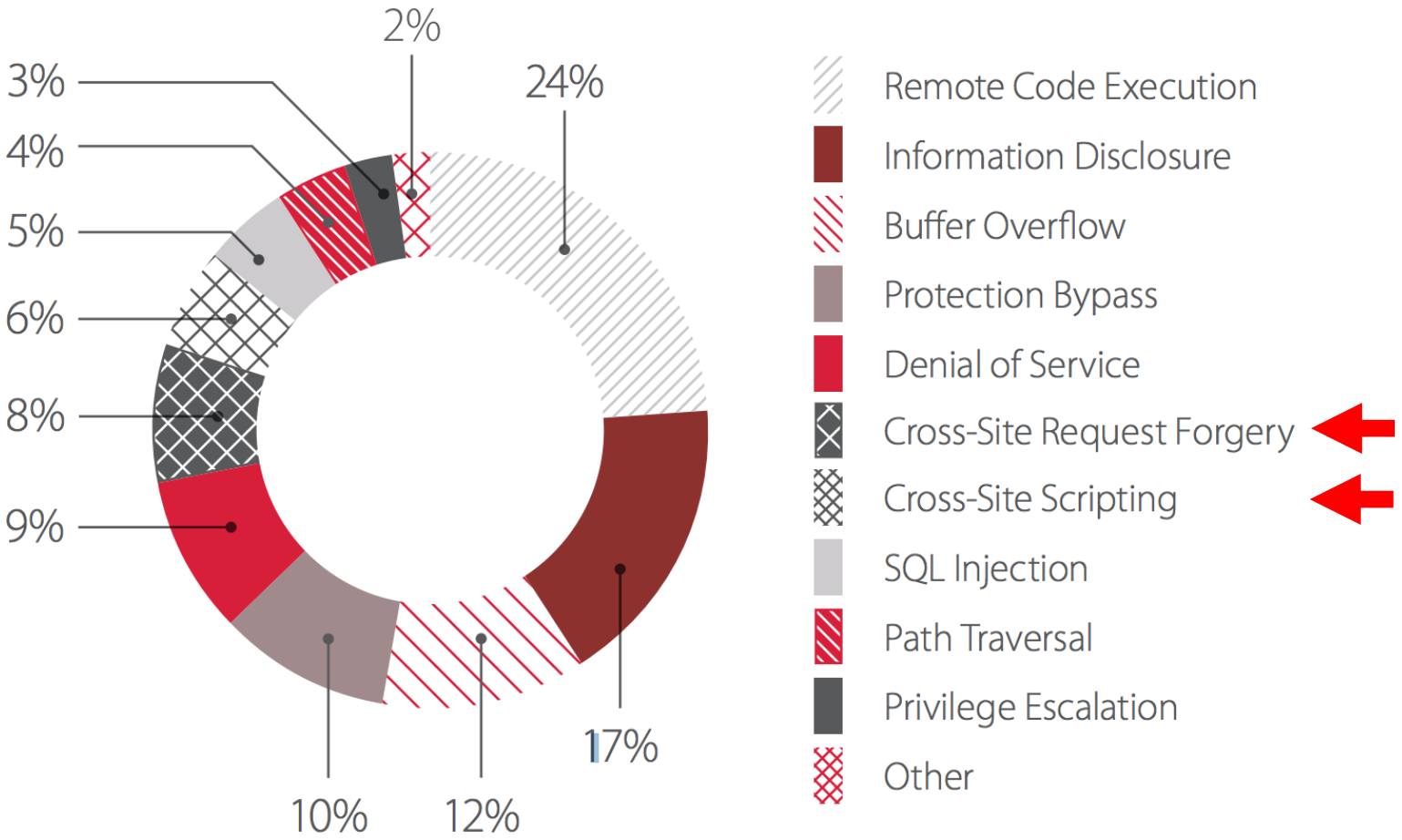
F-22 Raptor has 2 million lines of code

Boeing 787 Dreamliner has 7 million lines of code

Ford pickup truck F-150 has 150 million lines of code

https://www.eitdigital.eu/news-events/blog/article/guess-what-requires-150-million-lines-of-code/

Security of Critical Infrastructure & Cyber-physical systems (CPS)



Industrial control systems (ICS)

Types of vulnerabilities in ICS components

https://www.ptsecurity.com/upload/corporate/ww-en/analytics/ICS-Security-2017-eng.pdf
https://www.infosecurity-magazine.com/news/critical-infrastructure-more/

Hacking

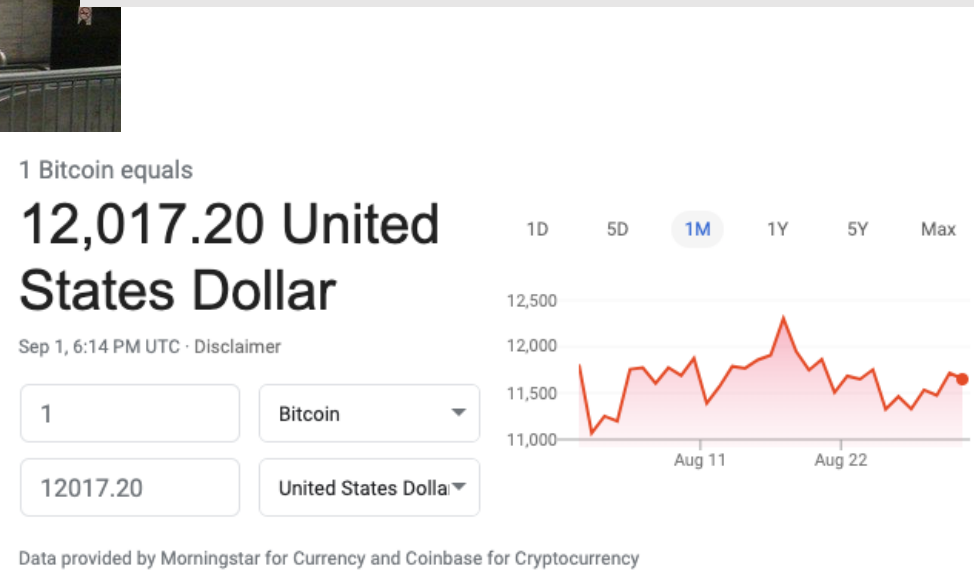
Ransomware attack on San Francisco public transit gives everyone a free ride

Nov 2016

San Francisco Municipal Transport Agency attacked by hackers who locked up computers and data with 100 bitcoin demand

MUNI stations displayed:

"You Hacked, ALL Data Encrypted. Contact For Key(cryptom27@yandex.com)ID:681 ,Enter."



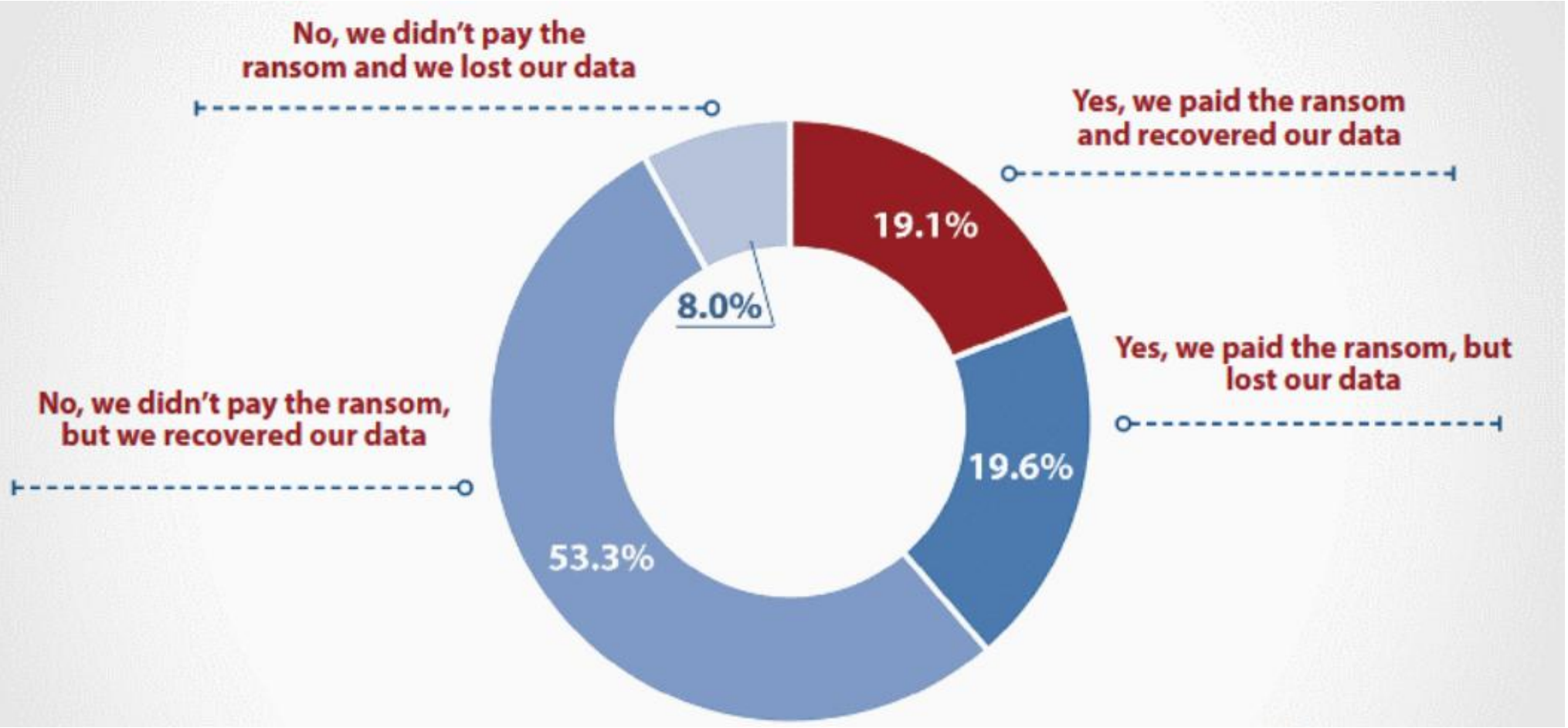
https://www.theguardian.com/technology/2016/nov/28/passengers-free-ride-san-francisco-muni-ransomeware

Colonial Pipeline confirms it paid $4.4m ransom to hacker gang after attack (2021)

https://www.theguardian.com/technology/2021/may/19/colonial-pipeline-cyber-attack-ransom

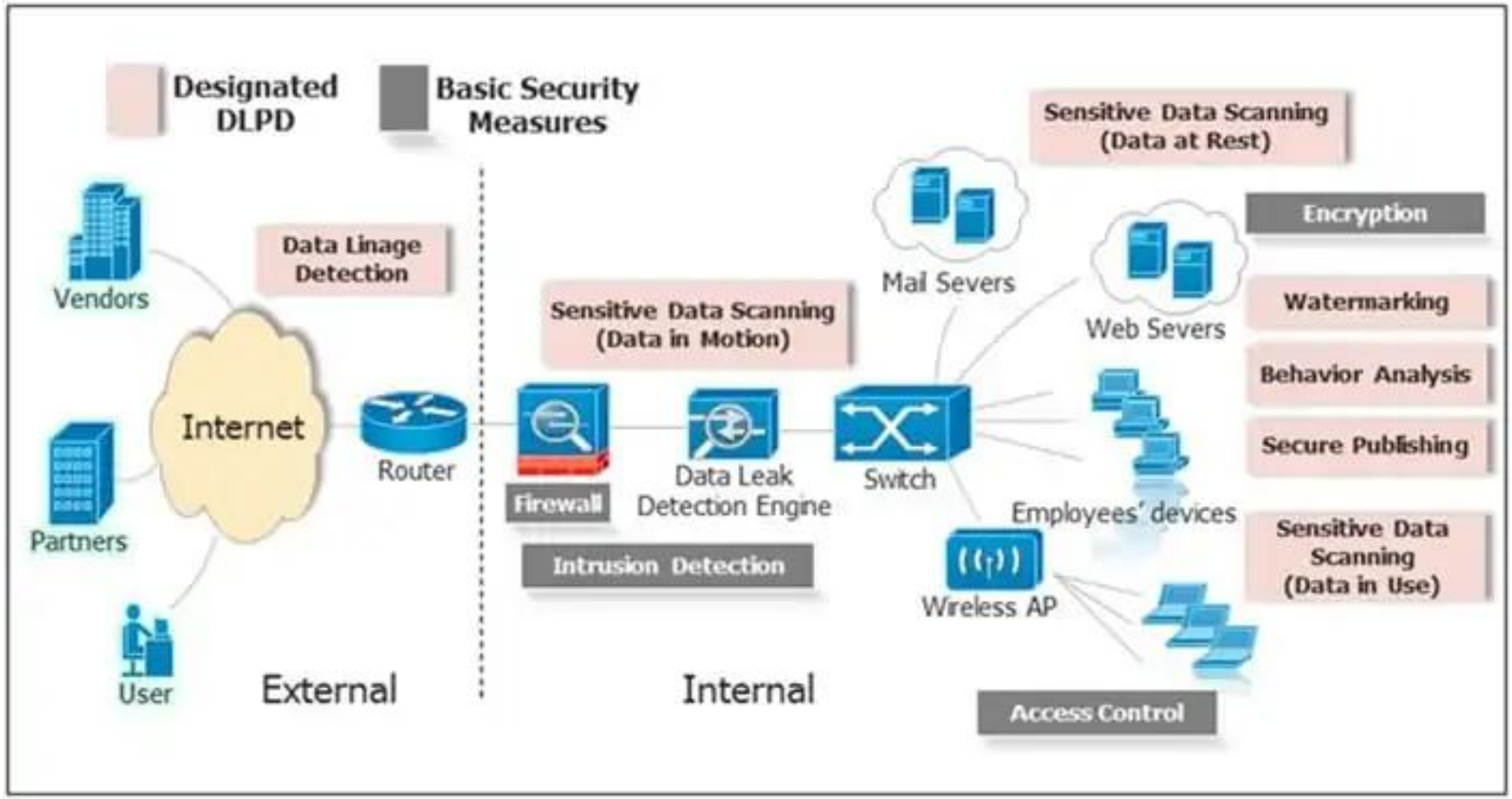
To pay or not to pay? That's the question

Survey of nearly 1,200 IT security practitioners and decision makers across 17 countries



https://www.bleepingcomputer.com/news/security/only-half-of-those-who-paid-a-ransomware-were-able-to-recover-their-data/

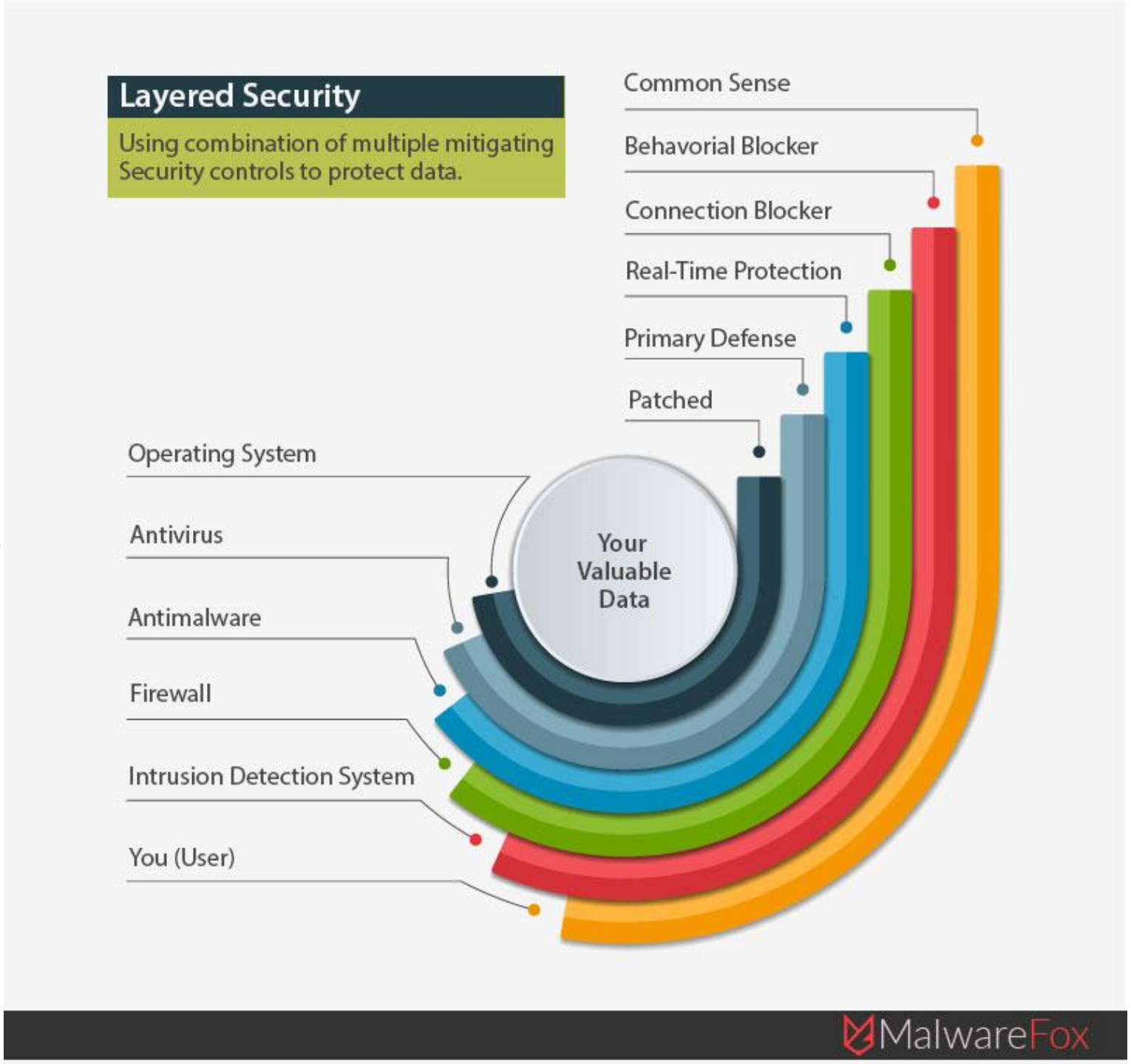
Security breaches are scary, but there are many points to prevent/detect them



Enterprise Data Breach: Causes, Challenges, Prevention, and Future Directions. Long Cheng, Fang Liu, and Danfeng Yao. **WIREs Data Mining and Knowledge Discovery.** Wiley. 2017.

Defense in depth strategy

(aka layered security)



https://www.malwarefox.com/layered-security/

This tutorial will focus on software scanning, especially for detecting crypto API misuses

We need both -- developer training & using tools

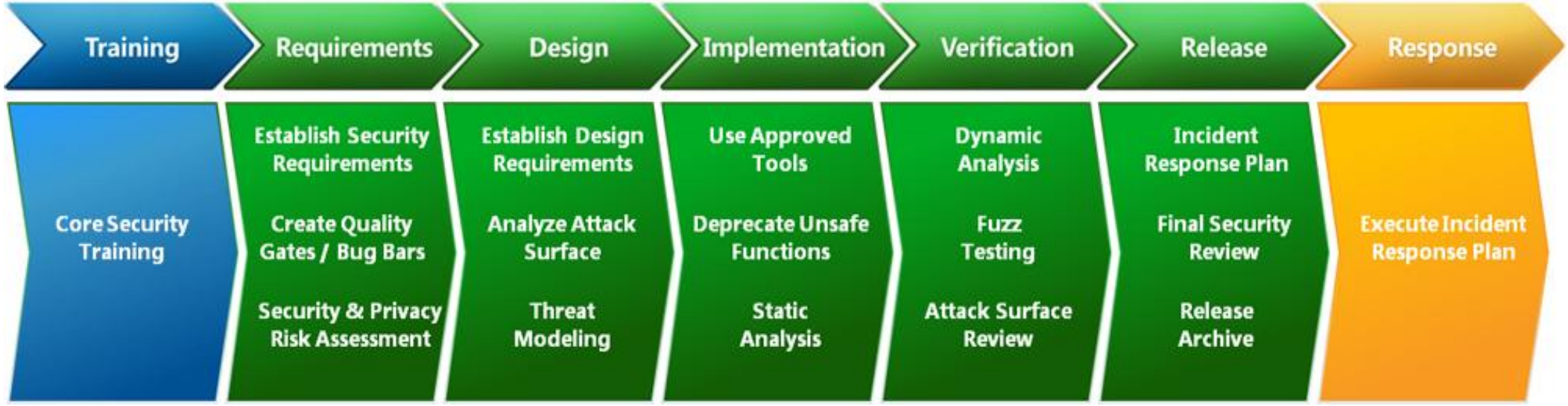
Top 10 secure coding rules

1. Validate input. Validate input from all untrusted data sources.
2. Heed compiler warnings [and other warnings].
3. Architect and design for security policies.
4. Keep it simple.
5. Default deny.
6. Adhere to the principle of least privilege.
7. Sanitize data sent to other systems.
8. Practice defense in depth.
9. Use effective quality assurance techniques.
10. Adopt a secure coding standard.

https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices

Microsoft secure development lifecycle (SDL)

Developers need TOOLS and more TOOLS



https://social.technet.microsoft.com/wiki/contents/articles/7100.the-security-development-lifecycle.aspx

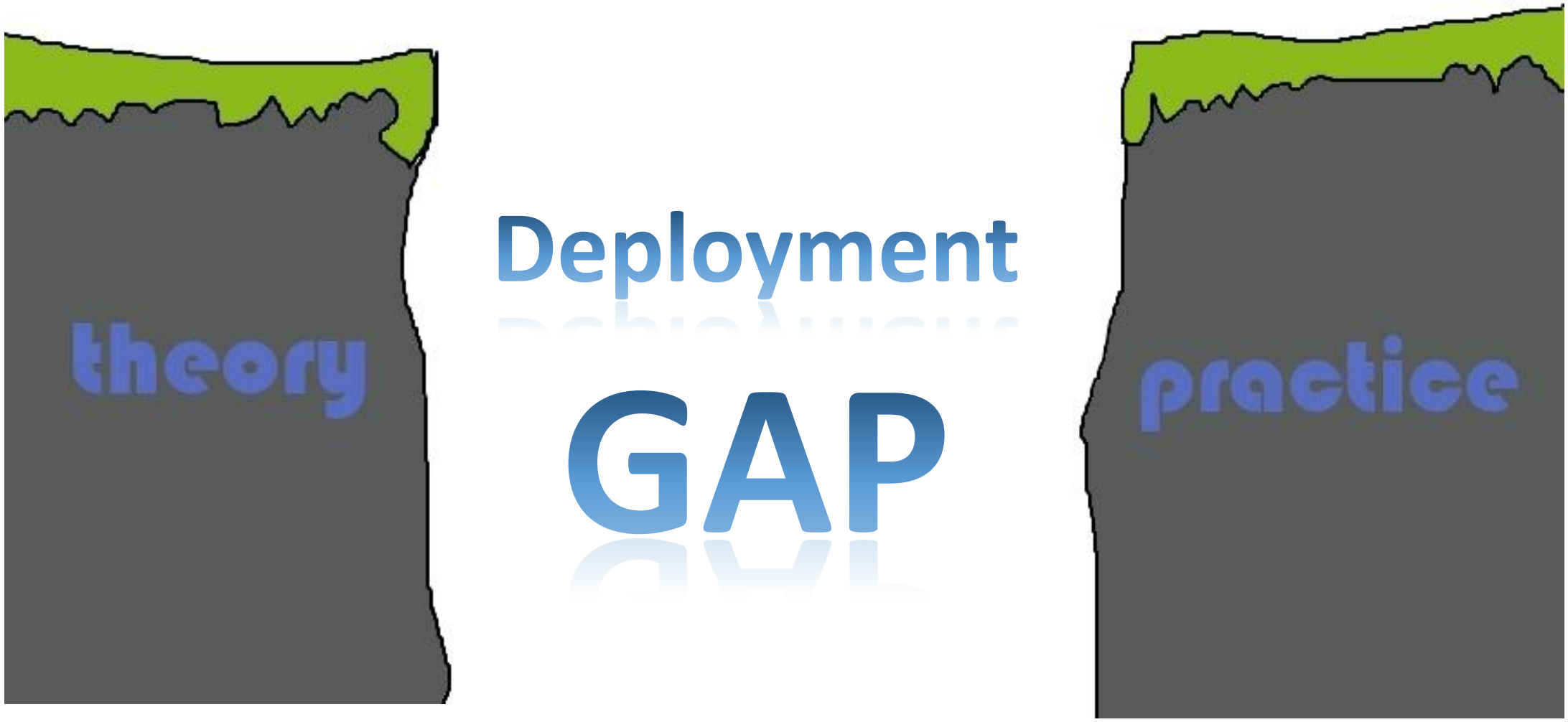
Who wouldn’t want to write secure code?

Budget

False positives

Time

Resources


theory
Deployment
GAP
practice

CSRF token in Java -- an example of the gap

Cross-Site Request Forgeries: Exploitation and Prevention

William Zeller* and Edward W. Felten*†
*Department of Computer Science
*Center for Information Technology Policy
†Woodrow Wilson School of Public and International Affairs
Princeton University
{wzeller,felten}@cs.princeton.edu

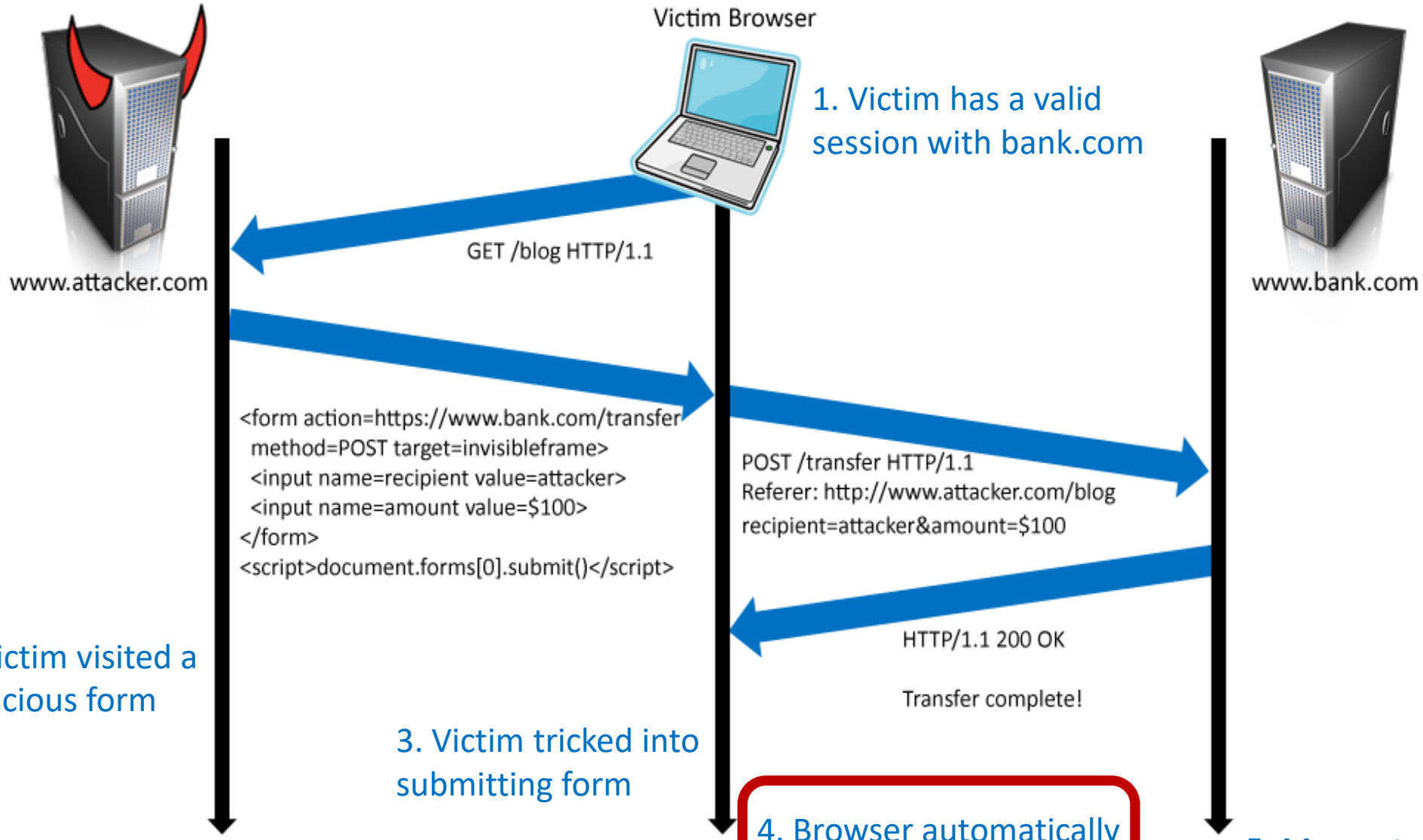
Revision 10/15/2008: Noted that the New York Times has fixed the vulnerability described below. Also clarified that our server-side CSRF protection recommendations *do*

1 Introduction

Cross-Site Request Forgery[1] (CSRF) attacks occur when a

[PDF] Robust Defenses for Cross-Site Request Forgery - Stanford Security Lab
https://seclab.stanford.edu/websec/csrf/csrf.pdf ▾
by A Barth - 2008 - Cited by 456 - Related articles
Collin Jackson. **Stanford** ... Cross-Site Request Forgery (**CSRF**) is a widely exploited web site ... the header can be used today as a reliable **CSRF** defense.

What is cross-site request forgery (CSRF) attack?



From C. Jackson

The most dangerous software vulnerabilities

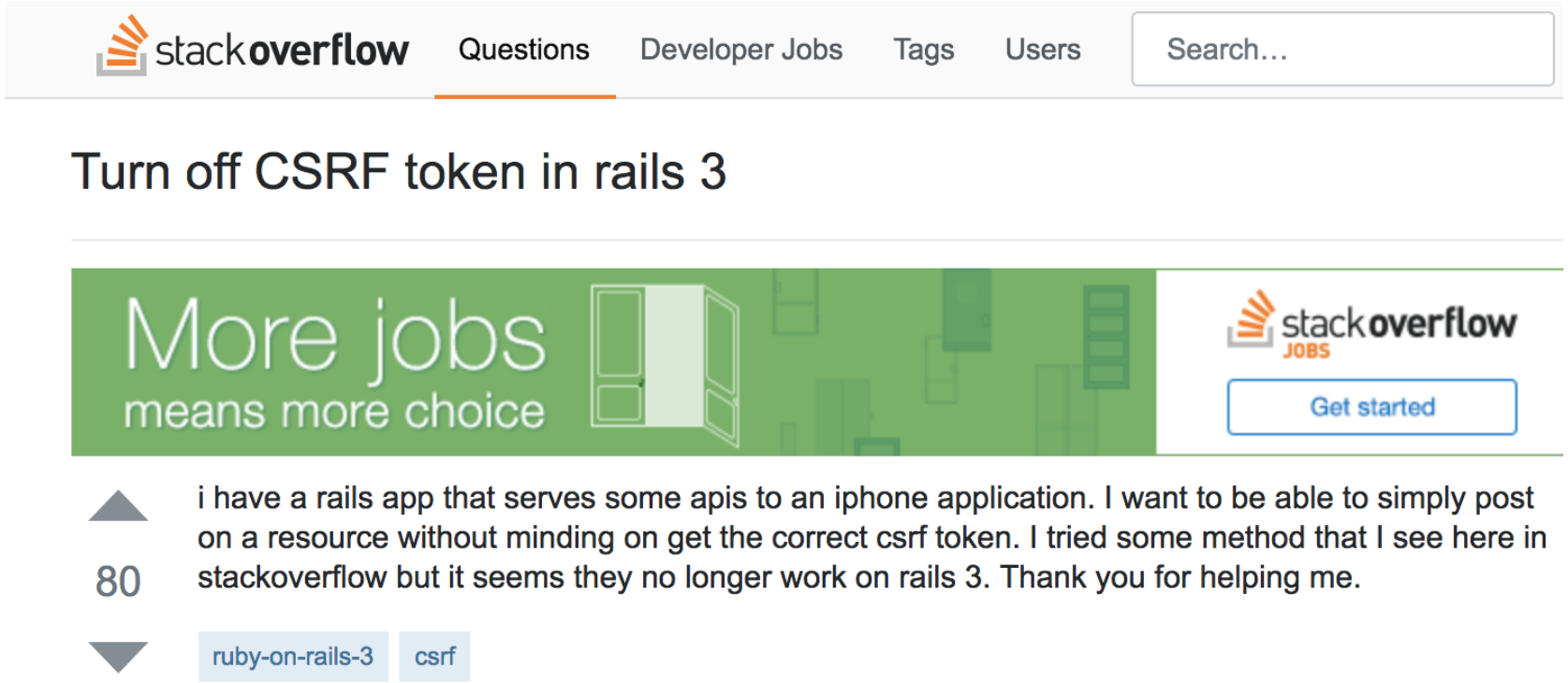
1. CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer
2. CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
3. CWE-20 Improper Input Validation
4. CWE-200 Information Exposure
5. CWE-125 Out-of-bounds Read
6. CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
7. CWE-416 Use After Free
8. CWE-190 Integer Overflow or Wraparound
9. **CWE-352 Cross-Site Request Forgery (CSRF)**
10. CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

1. CWE-89 Improper Neutralization of Special Elements used ... ('SQL Injection')
2. CWE-502 Deserialization of Untrusted Data
3. CWE-787 Out-of-bounds Write
4. CWE-78 Improper Neutralization of Special ... ('OS Command Injection')
5. CWE-120 Buffer Copy without Checking Size of ... ('Classic Buffer Overflow')
6. CWE-94 Improper Control of Generation of Code ('Code Injection')
7. CWE-798 Use of Hard-coded Credentials
8. CWE-434 Unrestricted Upload of File with Dangerous Type
9. CWE-416 Use After Free
10. **CWE-352 Cross-Site Request Forgery (CSRF)**

[Galhardo ACSAC 2020]

Developers need help

"Addingcsrf().disable() solved the issue!!! I have no idea why it was enabled by default" – a StackOverflow post



[Meng ICSE 2018] Available at: https://arxiv.org/pdf/1709.09970.pdf

Developers definitely need help

"Addingcsrf().disable() solved the issue!!! I have no idea why it was enabled by default"

"adding -Dtrust_all_cert=true to VM arguments"

"I want my client to accept any certificate (because I'm only ever pointing to one server)"

```
1  // Create a trust manager that does not validate certificate chains
2  TrustManager[] trustAllCerts = new TrustManager[]{
3    new X509TrustManager() {
4      public java.security.cert.X509Certificate[]
           getAcceptedIssuers() {return null;}
5      public void checkClientTrusted(...) {}
6      public void checkServerTrusted(...) {} };
7  // Install the all-trusting trust manager
8  try {
9    SSLContext sc = SSLContext.getInstance("SSL");
10   sc.init(null, trustAllCerts, new java.security.
         SecureRandom());
11   HttpsURLConnection.setDefaultSSLSocketFactory(sc
         .getSocketFactory());
12 } catch (Exception e) {}
```

N. Meng, S. Nagy, D. Yao, W. Zhuang, and G. Argoty. *ICSE* 2018

Influencers -- how much influence does StackOverflow have?

Insecure Posts	Total Views	No. of Posts	Min Views	Max Views	Average
Disabling CSRF Protection*	39,863	5	261	28,183	7,258
Trust All Certs	491,567	9	95	391,464	58,594
Obsolete Hash	91,492	3	1,897	86,070	30,497
Total Views	**622,922**	**17**	-	-	-

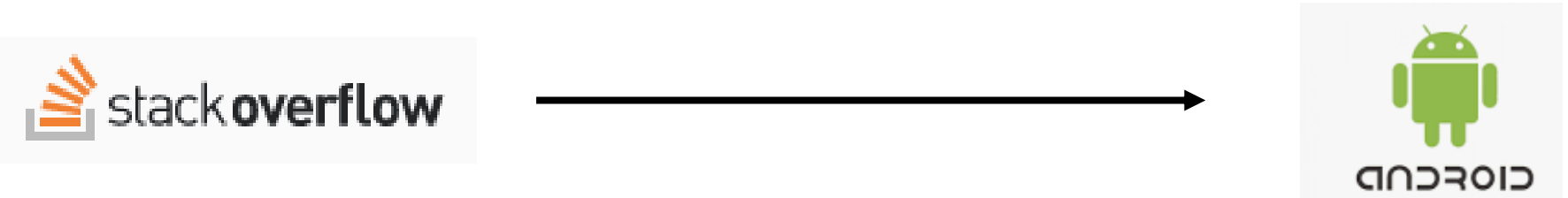
As of August 2017

Insecure StackOverflow posts seem to have a large influence on developers ☹

N. Meng, S. Nagy, D. Yao, W. Zhuang, and G. Argoty. *ICSE* 2018

* In Java Spring Security for web applications

Some StackOverflow code made its way into mobile devices



15.4% of apps contain code snippets copied from StackOverflow

Most of them contain at least 1 insecure code snippet

[Fischer 2017]

Social Dynamics on Stackoverflow

User: skanga [0]

"Do NOT EVER trust all certificates. That is very dangerous."

"the "accepted answer" is wrong and INDEED it is DANGEROUS. Others who blindly copy that code should know this."

User: MarsAtomic [6,287]

"once you have sufficient reputation you will be able to comment"

"If you don't have enough rep to comment, … then participate … until you have enough rep."

https://stackoverflow.com/questions/10594000/when-i-try-to-convert-a-string-with-certificate-exception-is-raised

CryptoGuard

CryptoGuard – Java Crypto Code Scanning with Deployment-quality Accuracy and Scalability

98.6% Precision

Out of 1,295 Apache alerts, only 18 are false alarms

Apache Ranger

hadoop

Apache Tomcat

APACHE OFBiz

MEECROWAVE
A light JAX-RS+CDI+JSON server!

Apache Ambari

APACHE Spark

Max, min and avg LoC: 2,571K (Hadoop), 1.1K (Commons Crypto), and 402K

CRYPTOGUARD DEPLOYMENT & IMPACT

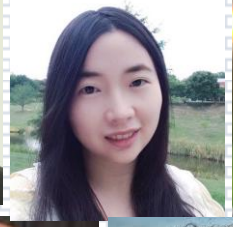
ORACLE

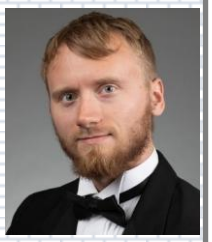
Parfait (an internal Oracle product) uses our approach to scan production code

Maven

Gradle

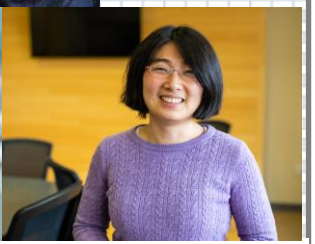
IJ



Nominated for NSA Science of Security Paper Competition





Office of Naval Research Science & Technology

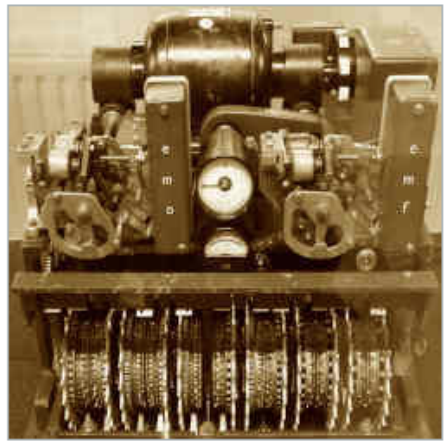
NSF

[Rahaman et al. ACM CCS 2019]
CryptoGuard and Benchmark on GitHub

ACM NEWS

A Tool for Hardening Java Crypto

By R. Colin Johnson
July 23, 2020
Comments



Researchers at the Virginia Polytechnic Institute and State University (Virginia Tech) say the vulnerability checking software they developed is mature, and nearing deployment.

Credit: Wikimedia Commons

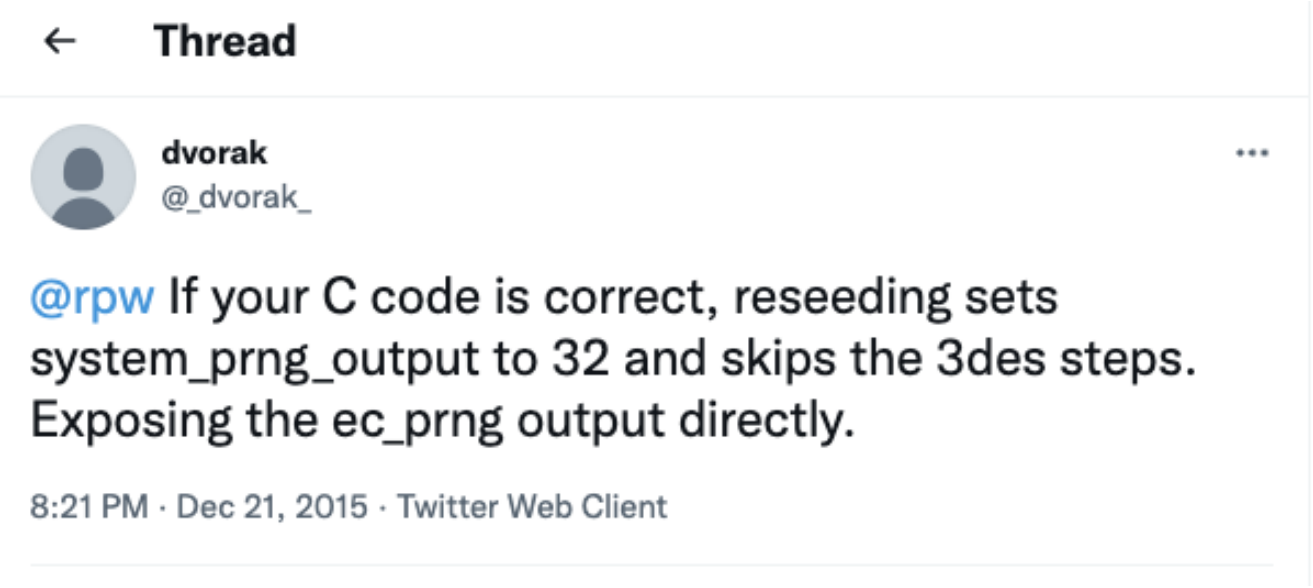
Identifying cryptographic vulnerabilities in today's million-line programs has become a critical endeavor. Because of the increasing sophistication of cybercriminals, programmers can no longer afford to test for vulnerabilities using only traditional debugging techniques, followed by releasing software, collecting bug reports and patching.

The new frontier being pursued by government, industry, and academia are automated tools that are capable of culling vulnerabilities before releasing source code into the wild. When run on existing software, such as the open-source Apache programs managing the world's servers, these tools also are finding a surprising number of vulnerabilities in software that is decades old.

Most open-source automated vulnerability checkers are still finding their way, but a team of researchers at the Virginia Polytechnic Institute and State University (Virginia Tech) claim to have vulnerability-checking software that is mature, and approaching deployment. Called CryptoGuard, the software automatically identifies cryptographic vulnerabilities in Java (and soon Python) source code. Funded by the U.S. Navy's Office of Naval Research (ONR) and the National Science Foundation (NSF), CryptoGuard is

Comm. Of ACM article on CryptoGuard: https://cacm.acm.org/news/246385-a-tool-for-hardening-java-crypto/fulltext

Juniper Dual EC Incident (2015)




```
unsigned int index;

void prng_reseed(void) {.
    ….
    // obtain a 32B secret w/ Dual EC
    index = 32;
}
```

```
void prng_generate(void) {
    ….
    if {
        … prng_reseed();
    }
    for (; index <=31; index +=8) {
        …. // generate a PR output
        memcpy(&output[index, block, 8);
    }
}
```

[Checkoway CACM 2018]
https://youtu.be/M5LMFQDN2vY

Open research problems in secure coding

- [Extensibility] Generating scanning algorithms automatically? Easily enforce new security rules?
- [AI] Auto code repair, API completion
- [Science of security] Benchmarking, measurement, comparison
- [Languages] Java, Python, others libraries?
- [Crypto libs] To ensure the security of library code

Take-home message:

know there’re tools/strategies/resources to help developers secure code

Need more research addressing practical deployment challenges

IACR

Real World Crypto Symposium

Real World Crypto Symposium aims to bring together cryptography researchers with developers implementing cryptography in real-world systems. The conference goal is to strengthen the dialogue between these two communities. Topics covered focus on uses of cryptography in real-world environments such as the Internet, the cloud, and embedded devices.

ACSAC 2020
December 7–11, 2020 • Online

Hard Topic Theme: *Deployable and Impactful Security*

Background

Since 2013, ACSAC has had a hard topic theme that focuses the conference on tackling a hard, cutting-edge, cybersecurity pr
requiring cooperation from government, industry, and academia. This year, ACSAC especially encourages contributions in th
Deployable and Impactful Security.

Related references

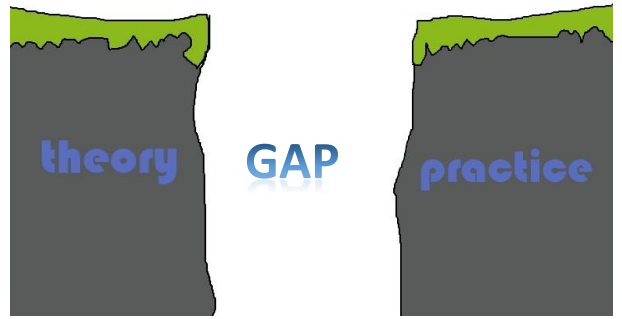
Papers:

- Sazzadur Rahaman, Ya Xiao, Sharmin Afrose, Fahad Shaon, Ke Tian, Miles Frantz, Murat Kantarcioglu, and Danfeng Yao. "Cryptoguard: High precision detection of cryptographic vulnerabilities in massive-sized Java projects." In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2455-2472. 2019.
- Sharmin Afrose, Sazzadur Rahaman, and Danfeng Yao. "CryptoAPI-Bench: A Comprehensive Benchmark on Java Cryptographic API Misuses." In *2019 IEEE Cybersecurity Development (SecDev)*, pp. 49-61. IEEE, 2019.
- Ya Xiao, Yang Zhao, Nicholas Allen, Nathan Keynes, and Cristina Cifuentes. "Industrial Experience of Finding Cryptographic Vulnerabilities in Large-scale Codebases." *arXiv preprint arXiv:2007.06122* (2020).
- Mazharul Islam, Sazzadur Rahaman, Na Meng, Behnaz Hassanshahi, Padmanabhan Krishnan, Danfeng (Daphne) Yao. Coding Practices and Recommendations of Spring Security for Enterprise Applications*. IEEE Secure Development Conference (SecDev).* 2020.
- Sharmin Afrose, Ya Xiao, Sazzadur Rahaman, Barton P. Miller, Danfeng (Daphne) Yao. Development of Benchmarks for Java Cryptographic APIs and Evaluation of Static Vulnerability Detection Tools. Under Revision. 2021.

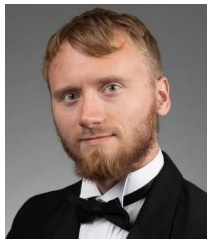
Online Resources:

- CryptoGuard. https://github.com/CryptoGuardOSS/cryptoguard
- CryptoAPI-Bench. https://github.com/CryptoGuardOSS/cryptoapi-bench
- Secure TLS/SSL code examples. https://github.com/AthenaXiao/SecureTLSCodeExample
- https://mybinder.org/v2/gh/franceme/cryptoguard/2020_SecDev_Tutorial

Our tutorial today



1. Jupyter Notebook setup

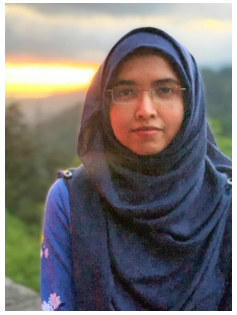


3. CryptoGuard intro

2. Complex crypto coding examples

4. Tool eval benchmarks

5. Python code security



6. Industry adoption

7. Demo time

ESORICS

Demo Sandbox

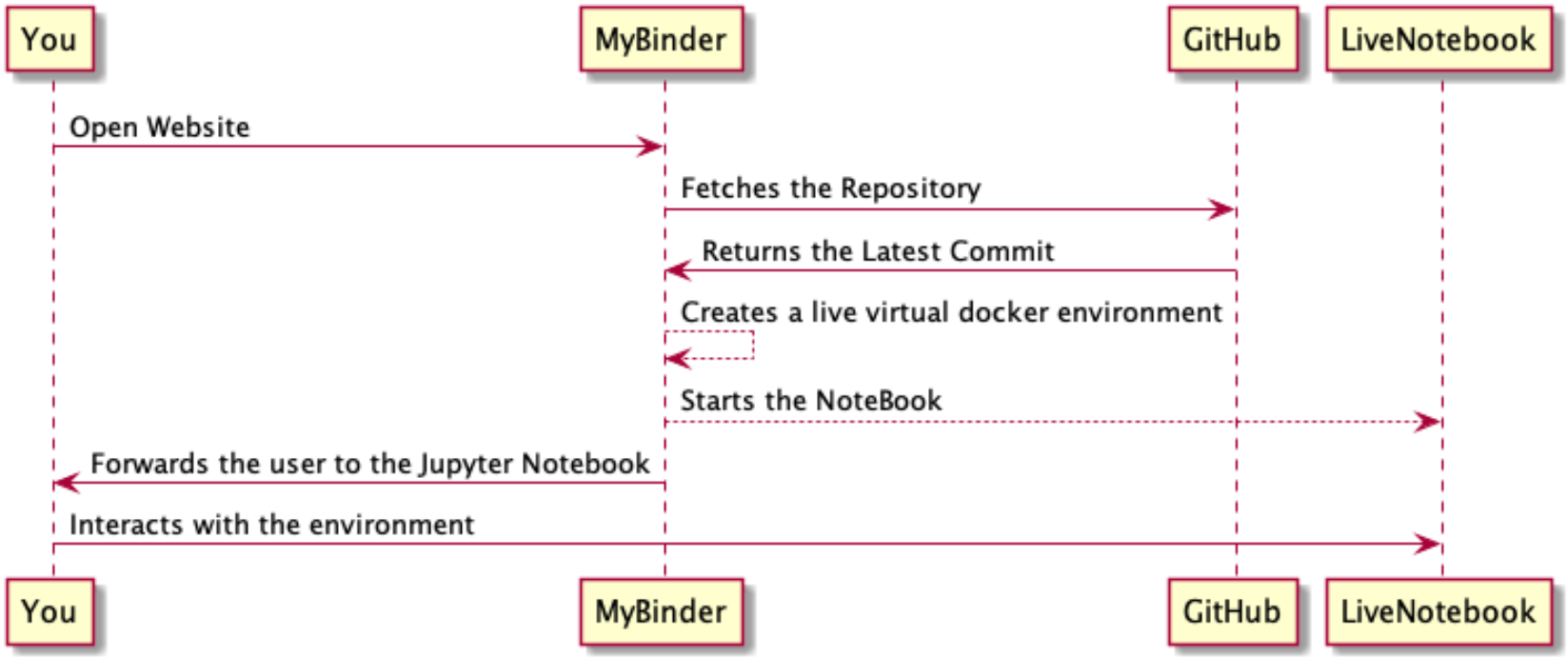
By: Miles Frantz

ESORICS

Live Environment

How is the demo being run?

- We will be using a public GitHub repository
- MyBinder is a public and free JupyterHub service
- We customize the Jupyter instance with Docker



ESORICS

GitHub Link

Where is the code located?

- The GitHub repository is located at github.com/franceme/Esorics_Conference

README.md

Principles and Practices of Secure Cryptographic Coding in Java Esorics 2021 Tutorial

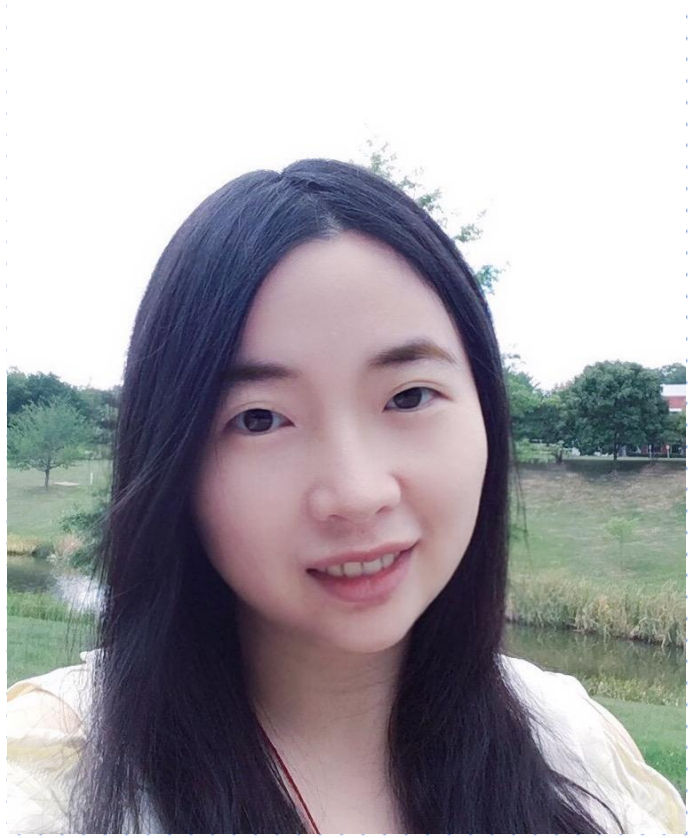
- Please visit the Notebook link launch binder to try running through our sample demo.

MyBinder

This is a website hosting a Docker image that actively runs either Java or Python3 samples. The Java Notebook is only possible by utilitzing IJava. This is still under progress.

TLS/SSL Authentication Code in JSSE

Presenter:
Ya Xiao



ESORICS

Mis-configuration of TLS/SSL can cause man-in-the-middle attacks.

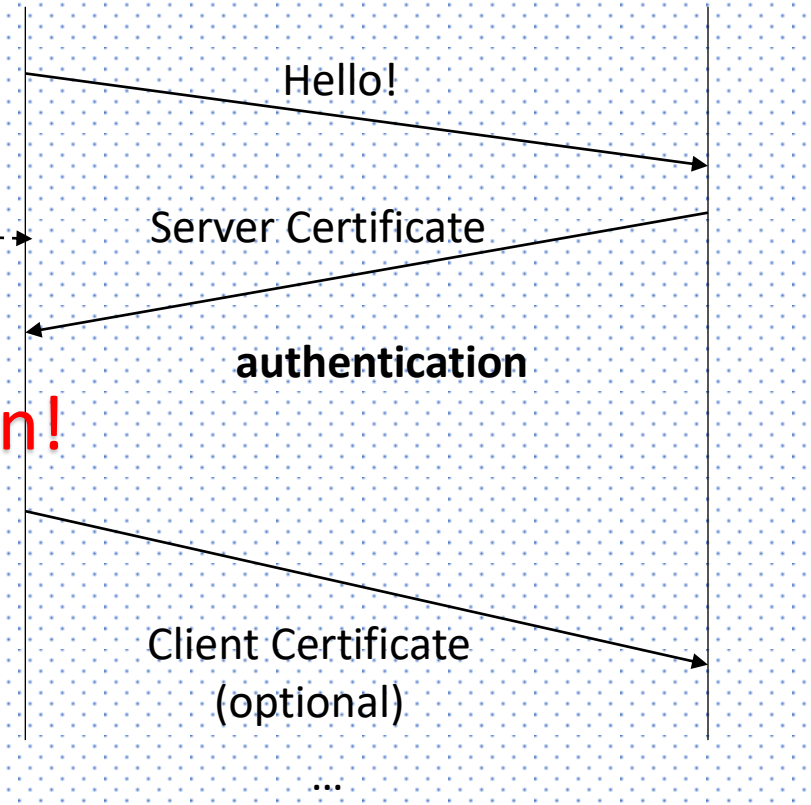
References:
[1] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. "The most dangerous code in the world: validating SSL certificates in non-browser software." In *Proceedings of the 2012 ACM conference on Computer and communications security (CCS)*, pp. 38-49. 2012.
[2] Na Meng, Stefan Nagy, Danfeng Yao, Wenjie Zhuang, and Gustavo Arango Argoty. "Secure coding practices in java: Challenges and vulnerabilities." In *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, pp. 372-383. 2018.
[3] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. "Why Eve and Mallory love Android: An analysis of Android SSL (in) security." In *Proceedings of the 2012 ACM conference on Computer and communications security (CCS)*, pp. 50-61. 2012.

ESORICS

TLS/SSL Authentication happens implicitly in a code snippet

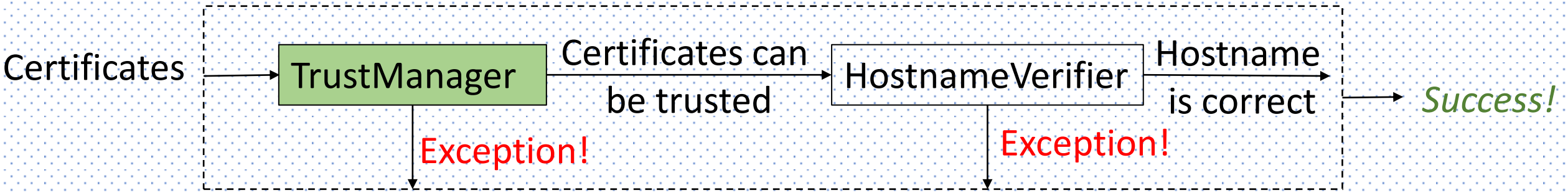
HTTPS = HTTP + TLS

```
1  URL url = new URL("https://our.example.com");
2  HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
3  InputStream in = conn.getInputStream();
```



ESORICS

Several examples of customized TrustManager



ESORICS

Customization 1: Secure or insecure?

```
1  public class SecDevTM implements X509TrustManager {
2      @Override
3      public void checkClientTrusted(X509Certificate[] chain, String authType)
4          throws CertificateException {
5          //validate certificate chain from the client
6      }
7      @Override
8      public void checkServerTrusted(X509Certificate[] chain, String authType)
9          throws CertificateException {
10         //validate certificate chain from the server
11     }
12     @Override
13     public X509Certificate[] getAcceptedIssuers() {
14         //obtain trust anchor
15         return null;
16     }
17 }
```

ESORICS

Customization 1: insecure!

```
1  public class SecDevTM implements X509TrustManager {
2      @Override
3      public void checkClientTrusted(X509Certificate[] chain, String authType)
4          throws CertificateException {
5          //validate certificate chain from the client
6      }
7      @Override
8      public void checkServerTrusted(X509Certificate[] chain, String authType)
9          throws CertificateException {
10         //validate certificate chain from the server
11     }
12     @Override
13     public X509Certificate[] getAcceptedIssuers() {
14         //obtain trust anchor
15         return null;
16     }
17 }
```

no verification happens!

It is insecure for doing nothing in the certificate validation methods (i.e. checkClientTrusted, checkServerTrusted).

ESORICS

Customization 2: Secure or insecure?

```
1  public class SecDevTM implements X509TrustManager {
2      private X509TrustManager defaultTM;
3      ...
4      @Override
5      public void checkServerTrusted(X509Certificate[] chain, String authType)
6      throws CertificateException {
7          try{
8              defaultTM.checkServerTrusted(chain, authType);
9          }
10         catch(CertificateException e){
11             Log.w("checkServerTrusted",e.toString());
12         }
13     }
14 }
```

Customization 2: insecure!

```
1  public class SecDevTM implements X509TrustManager {
2      private X509TrustManager defaultTM;
3      ...
4      @Override
5      public void checkServerTrusted(X509Certificate[] chain, String authType)
6      throws CertificateException {
7          try{
8              defaultTM.checkServerTrusted(chain, authType);
9          }
10         catch(CertificateException e){
11             Log.w("checkServerTrusted",e.toString());
12         }
13     }
14 }
```

no exception will be threw out!

Catching the exception without re-throw it is insecure!

Customization 3: Secure or insecure?

```
1  public class SecDevTM implements X509TrustManager {
2      private X509TrustManager defaultTM;
3      ...
4      @Override
5      public void checkServerTrusted(X509Certificate[] chain, String authType)
6      throws CertificateException {
7          if ((chain != null) && (chain.length == 1)) {
8              chain[0].checkValidity();
9          } else {
10             defaultTM.checkServerTrusted(chain, authType);
11         }
12     }
13 }
```

Customization 3: insecure!

```
1  public class SecDevTM implements X509TrustManager {
2      private X509TrustManager defaultTM;
3      ...
4      @Override
5      public void checkServerTrusted(X509Certificate[] chain, String authType)
6      throws CertificateException {
7          if ((chain != null) && (chain.length == 1)) {
8              chain[0].checkValidity();
9          } else {
10             defaultTM.checkServerTrusted(chain, authType);
11         }
12     }
13 }
```

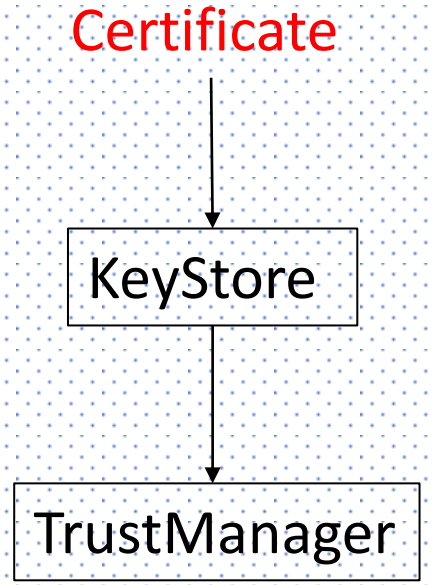
Bypassing certificate validation

checkValidity only checks whether the certificate is expired

Bypassing the certificate validation under certain condition is insecure!

Secure Customization Example
How to handle a self-signed certificate?

Secure Customization: using KeyStore



A keystore is primarily a database for storing application secrets. Keystores can also be used for storing “trust certificates” and CA chains.

A certificate can be specified as trusted by putting it in KeyStore.

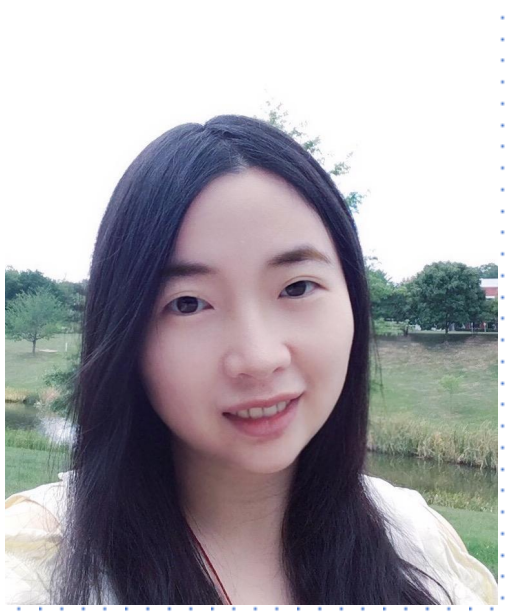
```
11  // create a new TrustManager that trusts our KeyStore
12  String tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
13  TrustManagerFactory tmf = TrustManagerFactory.getInstance(tmfAlgorithm);
14  tmf.init(keyStore);
15  TrustManager tms [] = tmf.getTrustManagers()
```

TLS/SSL Related Vulnerabilities

Vulnerability Description	Recommended Practices
Custom TrustManager to trust all certificates	Configure KeyStore
Custom Hostname verifiers to accept all hosts	Specify accepted hostnames
Custom SSLSocketFactory w/o manual Hostname verification	Manually call HostnameVerifier.verify(.)
Occasional use of HTTP	Use HTTPS

See more vulnerability types and the recommended practices for them in https://github.com/AthenaXiao/SecureTLSCodeExample

CryptoGuard Design/Results



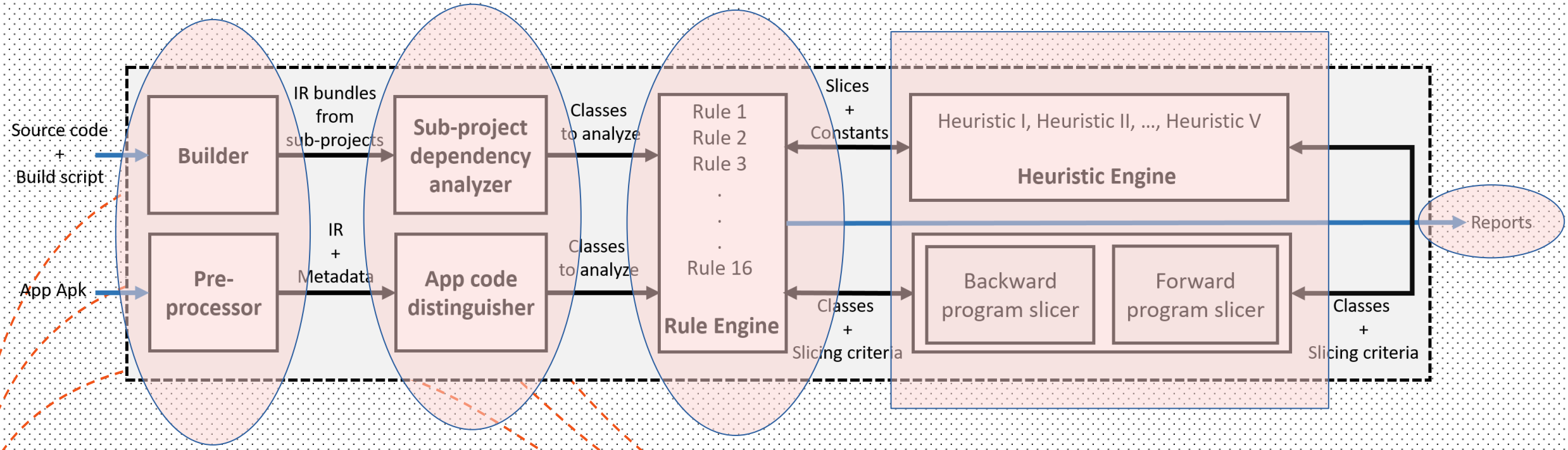
Presenter:
Ya Xiao

Slides credits: Sazzadur Rahaman

ESORICS

Cryptographic Misuse Detection with CryptoGuard

- CryptoGuard is a static analysis tool
- Dataflow analysis is implemented on Soot



UNIVERSITY LIBRARIES
VIRGINIA TECH

Precise cryptographic misuse detection is hard …

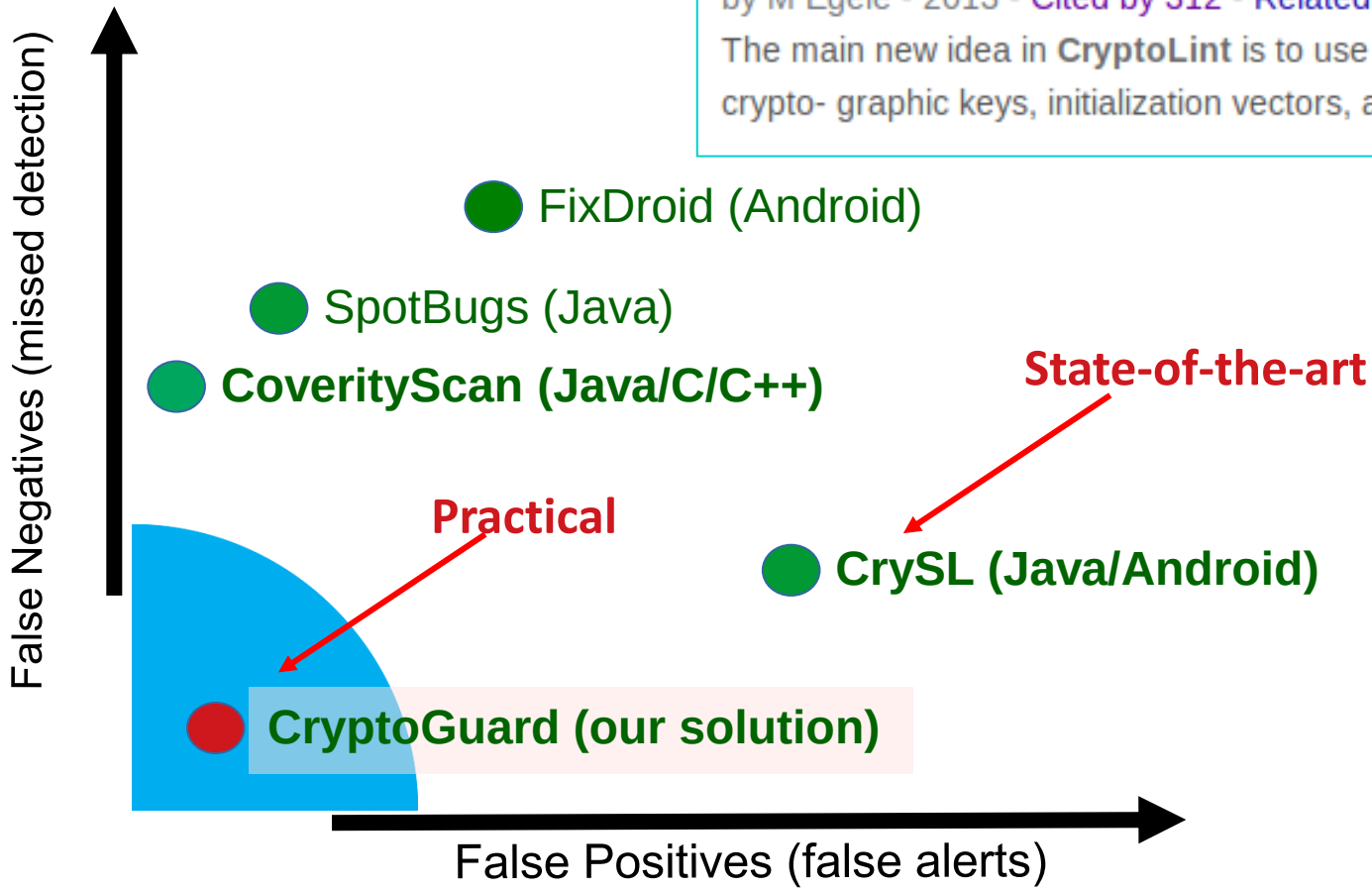
[PDF] An Empirical Study of Cryptographic Misuse in Android …

https://sites.cs.ucsb.edu › ~chris › research › doc › ccs13_cryptolint ▾

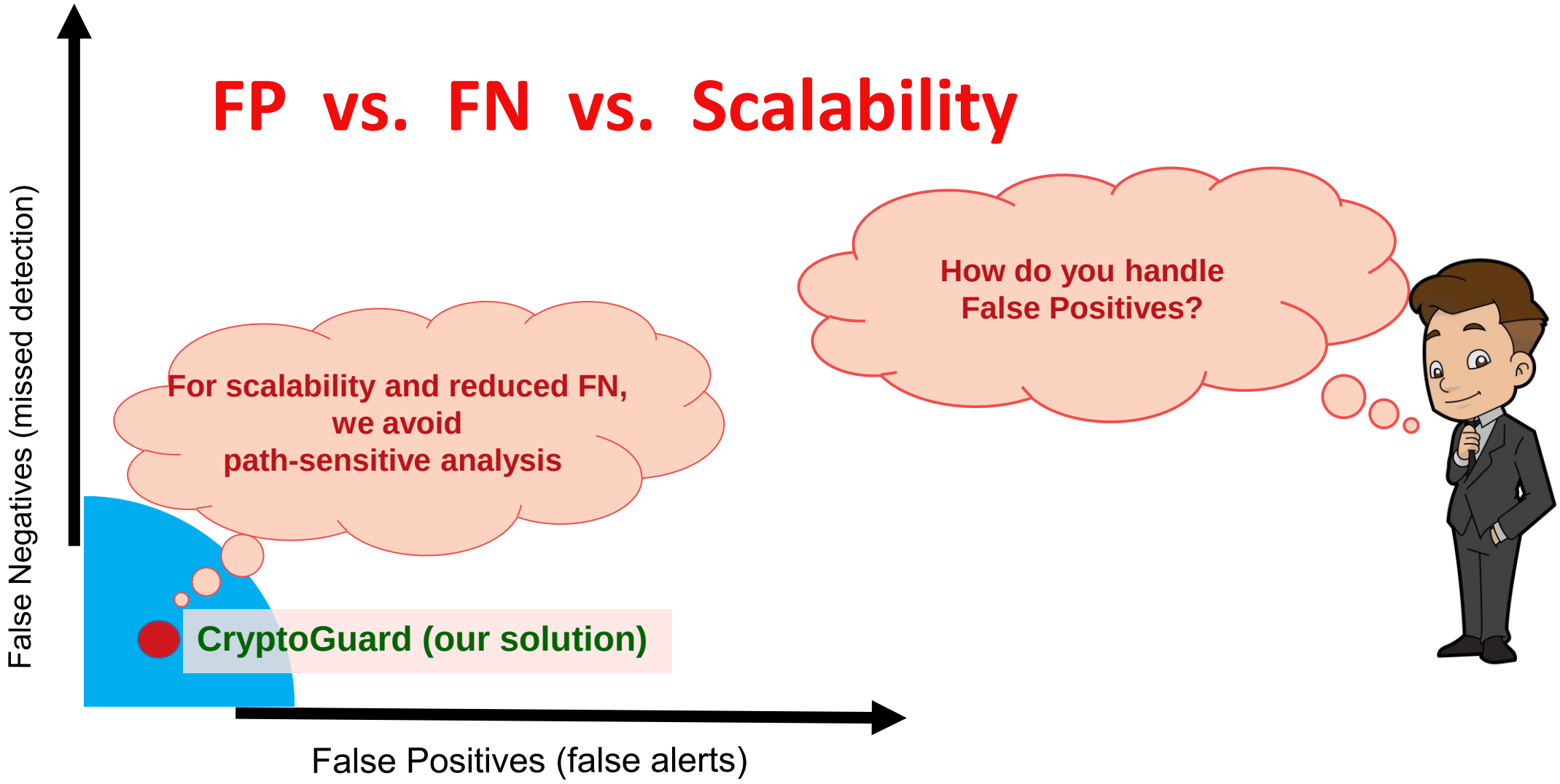
by M Egele - 2013 - Cited by 312 - Related articles

The main new idea in **CryptoLint** is to use static program slicing to identify flows between crypto- graphic keys, initialization vectors, and simila… cryptographic.

CryptoLint



Goal and Challenges



Sources of false positives ...

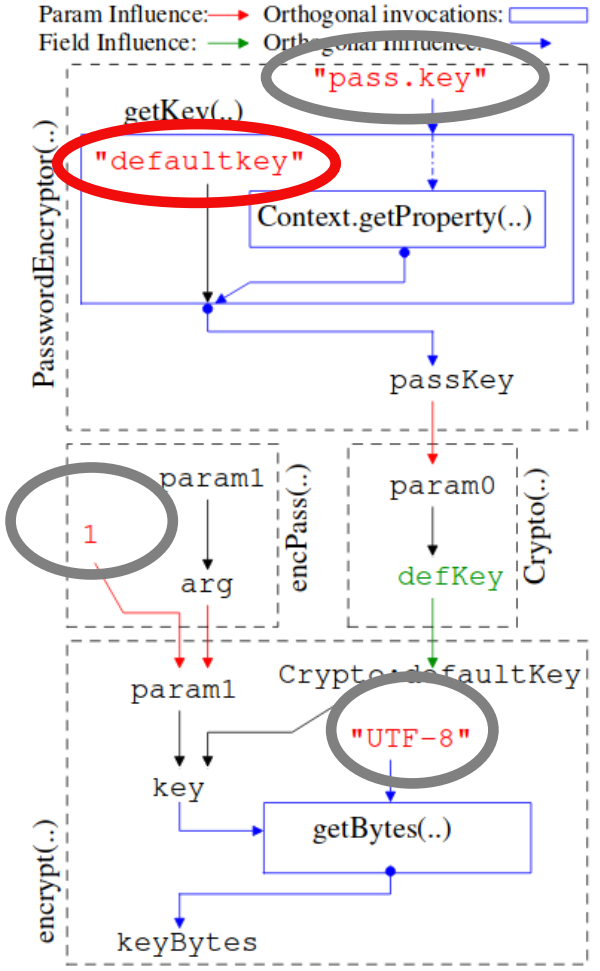
```
1 class PasswordEncryptor {
2
3  Crypto crypto;
4
5  public PasswordEncryptor(){
6   String passKey = PasswordEncryptor
                        .getKey("pass.key")
7    crypto = new Crypto(passKey); (p)
8  }
9
10 byte[] encPass(String [] arg){
11  return crypto.encrypt(arg[0], arg[1]); (p)
12 }
13
14 static String getKey(String src){
15  String key = Context.getProperty(src);
16  if (key == null) {
17    key = "defaultkey";
18  }
19  return key;
20  }
21 }
```

```
22 class Crypto {
23
24  String ALGO = "AES";
25  String ALGO_SPEC = "AES/CBC/NoPadding";
26  String defaultKey;
27  Cipher cipher;
28
29  public Crypto(String defKey){
30   cipher = Cipher.getInstance(ALGO_SPEC);
31   defaultKey = defKey; // assigning field
32  }
33
34  byte[] encrypt(String txt,String key){
35   if (key == null){
36    key = defaultKey; (f)
37   }
38   byte[] keyBytes = key.getBytes("UTF-8");
39   byte[] txtBytes = txt.getBytes();
40   SecretKeySpec keySpc =

   new SecretKeySpec(keyBytes, ALGO);

41   cipher.init(Cipher.ENCRYPT_MODE,keySpc);

                                    es);}}
```



Implementations of some methods are not available!

Reduce false positives: **Programming idioms** and **language restrictions** to the rescue!

Observation I: A vast majority of them are caused by phantom methods!

```
bytes = virtualinvoke key.<String: byte[] getBytes(String)>("UTF-8")
```

State indicator

```
key = staticinvoke <PassEncryptor: String getKey(String)>("pass.key")
```

Resource identifier

```
key = interfaceinvoke map.<HashMap: String get(String)>("key_id")
```

Resource identifier

Reduction of False Alerts by Our Refinement Insights

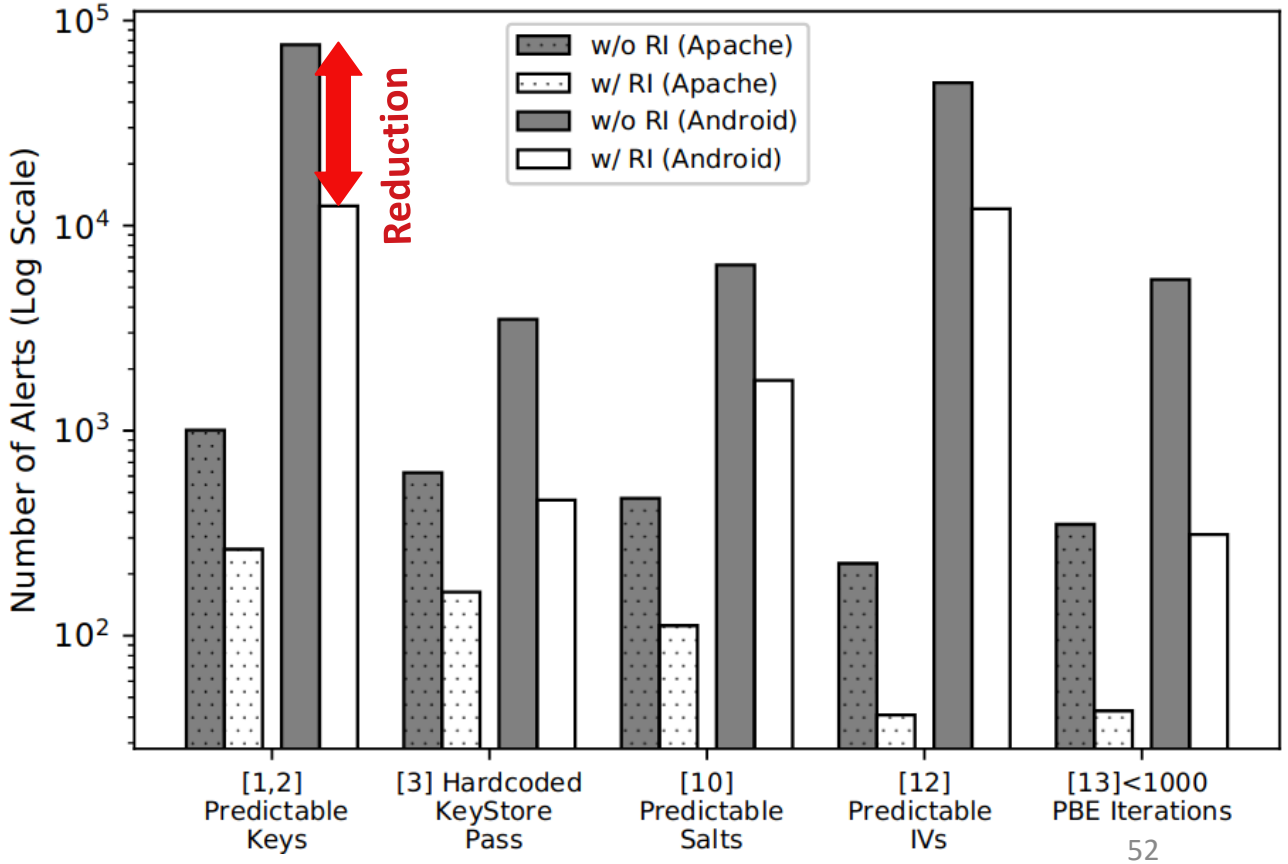
RI I: Removal of state indicators
RI II: Removal of resource identifiers
RI III: Removal of bookkeeping indices
RI IV: Removal of contextually incompatible constants
RI V: Removal of constants in infeasible paths

We customized the Data flow analysis algorithms to incorporate these insights ...

We evaluated the performance on
- 46 Apache projects
- 6,181 Android apps

Apache: 76% reduction

Android: 80% reduction



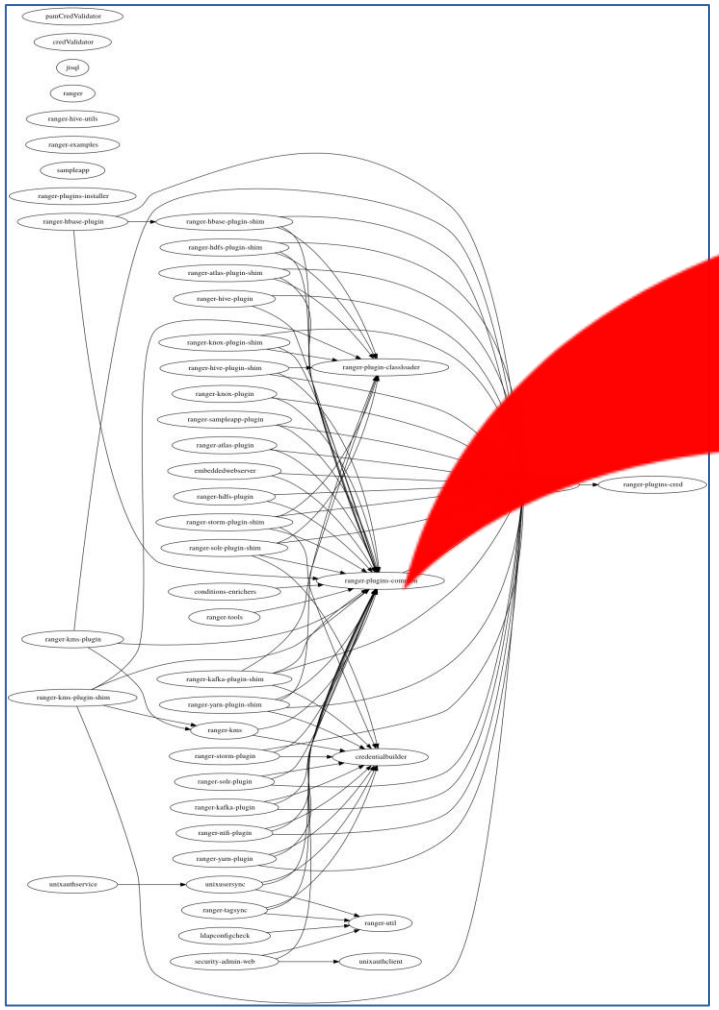
Deployment-grade accuracy

Rules	Total Alerts	# True Positives	Precision
(1,2) Predictable Keys	264	248	94.14 %
(3) Hardcoded Store Pass	148	148	100 %
(4) Dummy Hostname Verifier	12	12	100 %
(5) Dummy Cert. Validation	30	30	100 %
(6) Used Improper Socket	4	4	100 %
(7) Used HTTP	222	222	100 %
(8) Predictable Seeds	0	0	0%
(9) Untrusted PRNG	142	142	100 %
(10) Static Salts			100 %
(11) ECB mode for Symm. Crypto	41	41	100 %
(12) Static IV	41	40	97.56 %
(13) <1000 PBE iterations			97.67 %
(14) Broken Symm. Crypto Algorithm	86	86	100 %
(15) Insecure Asymm. Crypto	12	12	100 %
(16) Broken Hash	138	138	100 %
Total	**1,295**	**1,277**	**98.61 %**

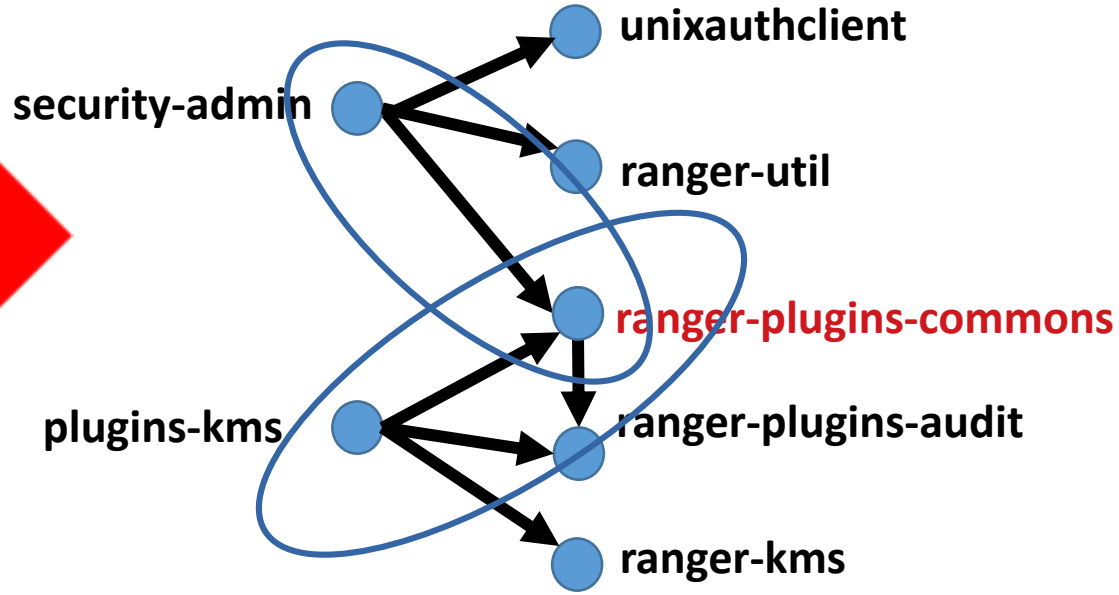
Manual analysis confirmed 18 false alerts ...

Only 1.39% false positives!

Performance Optimization With Subproject Dependency Analysis



Subproject Dependency Graph (Apache Ranger)



Root-subprojects can be analyzed in parallel!

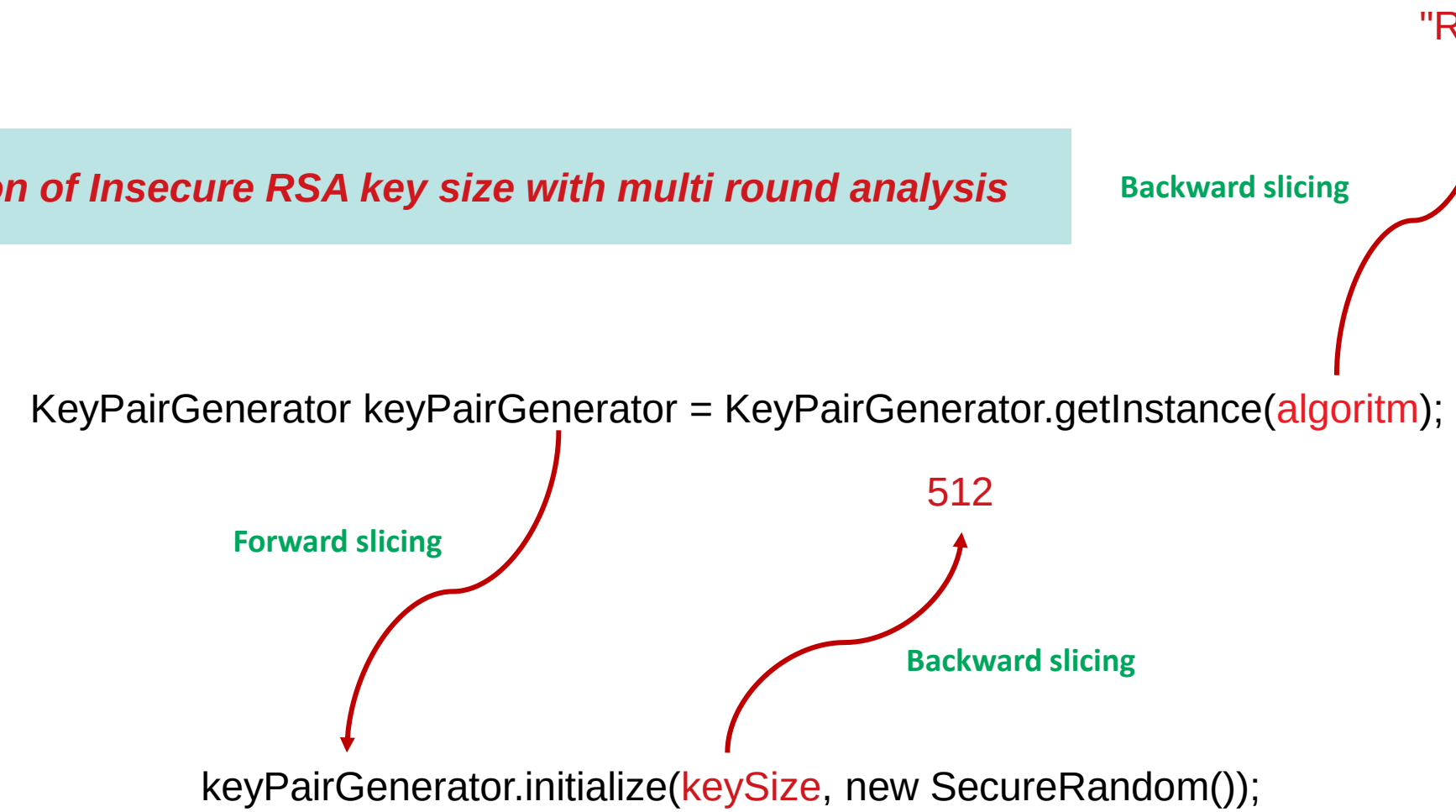
Other Features: CryptoGuard uses forward slicing for some rules (Insecure SSLSocket)

SSLSocket requires manual hostname verification

```
SocketFactory sf = SSLSocketFactory.getDefault();
SSLSocket socket = (SSLSocket) sf.createSocket(""mail.google.com", 443);
HostnameVerifier hv = HttpsURLConnection.getDefaultHostnameVerifier();
SSLSession s = socket.getSession();
if (!hv.verify("mail.google.com", s)) {
  throw new SSLHandshakeException("Expected mail.google.com, not found ");
}
// Use SSLSession
socket.close();
```

Single round of analysis is not sufficient (Insecure asymmetric crypto)

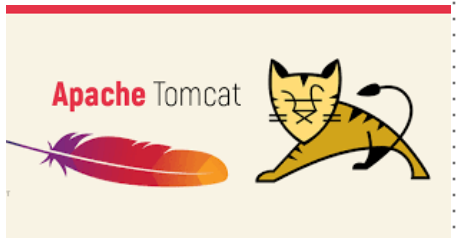
Detection of Insecure RSA key size with multi round analysis



Deployment-grade scalability -- 46 open-source Apache projects evaluated

We discovered misuses in Apache top-tier projects!

Apache Ambari



Security finding (deterministic salt)

Generates salt from the password itself!

Weak message digest

```
1 PBEKeySpec getPBEParameterSpec(String password) throws Throwable {
2     MessageDigest md = MessageDigest.getInstance(MD_ALGO); // MD5
3     byte[] saltGen = md.digest(password.getBytes());
4     byte[] salt = new byte[SALT_SIZE];
5     System.arraycopy(saltGen, 0, salt, 0, SALT_SIZE);
6     int iteration = password.toCharArray().length + 1;
7     return new PBEKeySpec(password.toCharArray(), salt, iteration); }
```

#number of Iterations is the length of the password

Android app libraries have issues

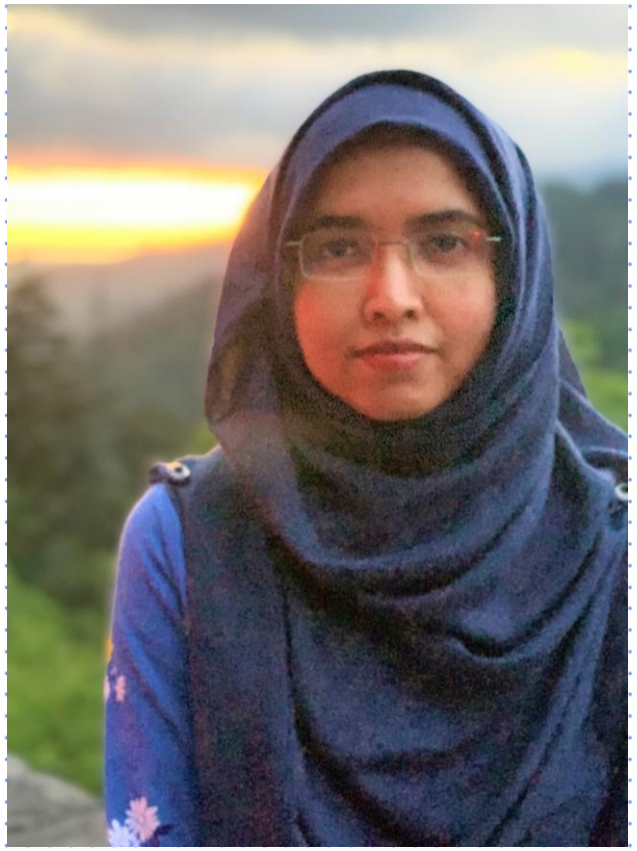
Package name	Violated Rules
com.google.api	3, **4, 5**, 7
com.umeng.anlytics	7, 9, 12, 16
com.facebook.ads	**5**, 9, 16
org.apache.commons	**5**, 9 , 16
com.tencent.open	2, 7, 9

	Rules Desc.
2	Predictable pwds for PBE
3	Predictable pwds for keystores
4	**Dummy hostname verifier**
5	**Dummy cert. verifier**
7	Use of HTTP
9	Weak PRNG
12	Static IV
16	Broken hash

96% of detected issues come from mid-level libraries

Benchmarks:
Test Cases & Evaluation

Presenter:
Sharmin Afrose



ESORICS

Java Cryptographic Benchmarks

- Two benchmarks based on Java cryptographic API misuses
- **CryptoAPI-Bench:** Includes 181 unit test cases of 18 Rules
- **ApacheCryptoAPI-Bench:** Includes 122 test cases from 10 Apache projects

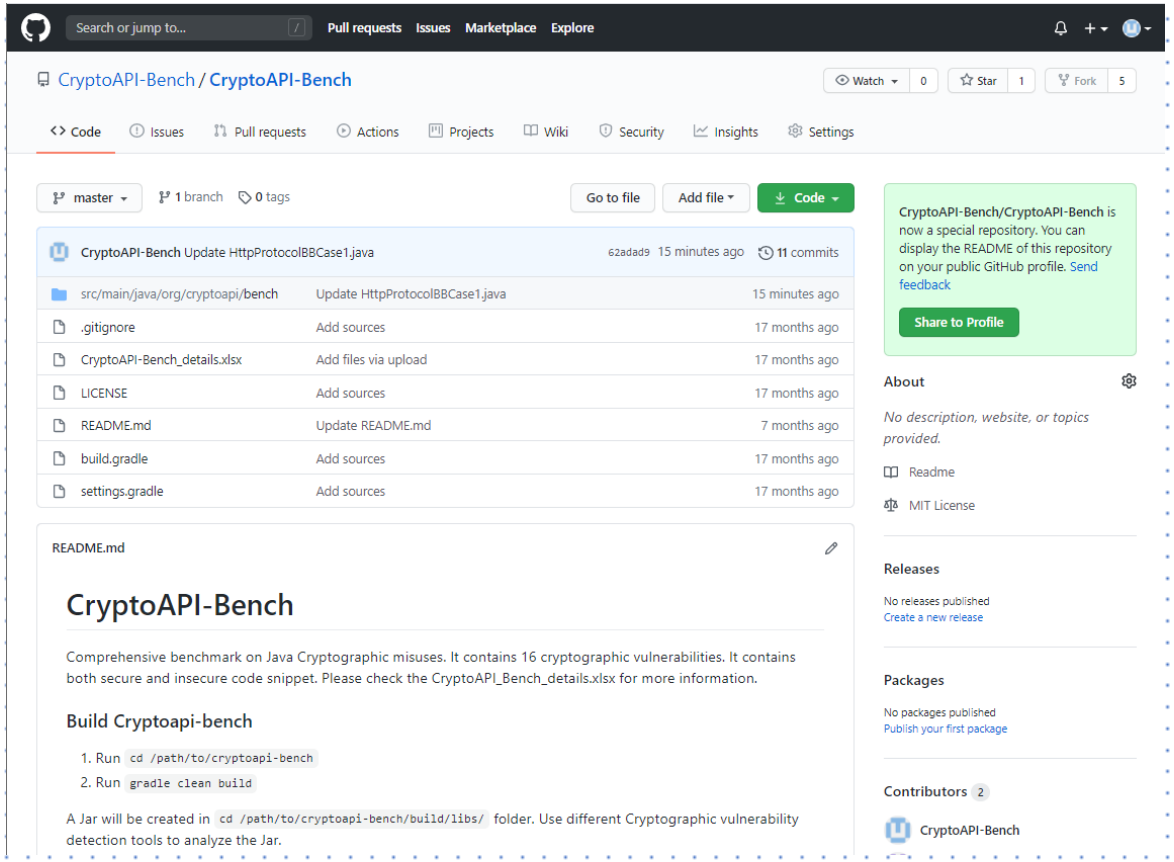
Improve tool's performance

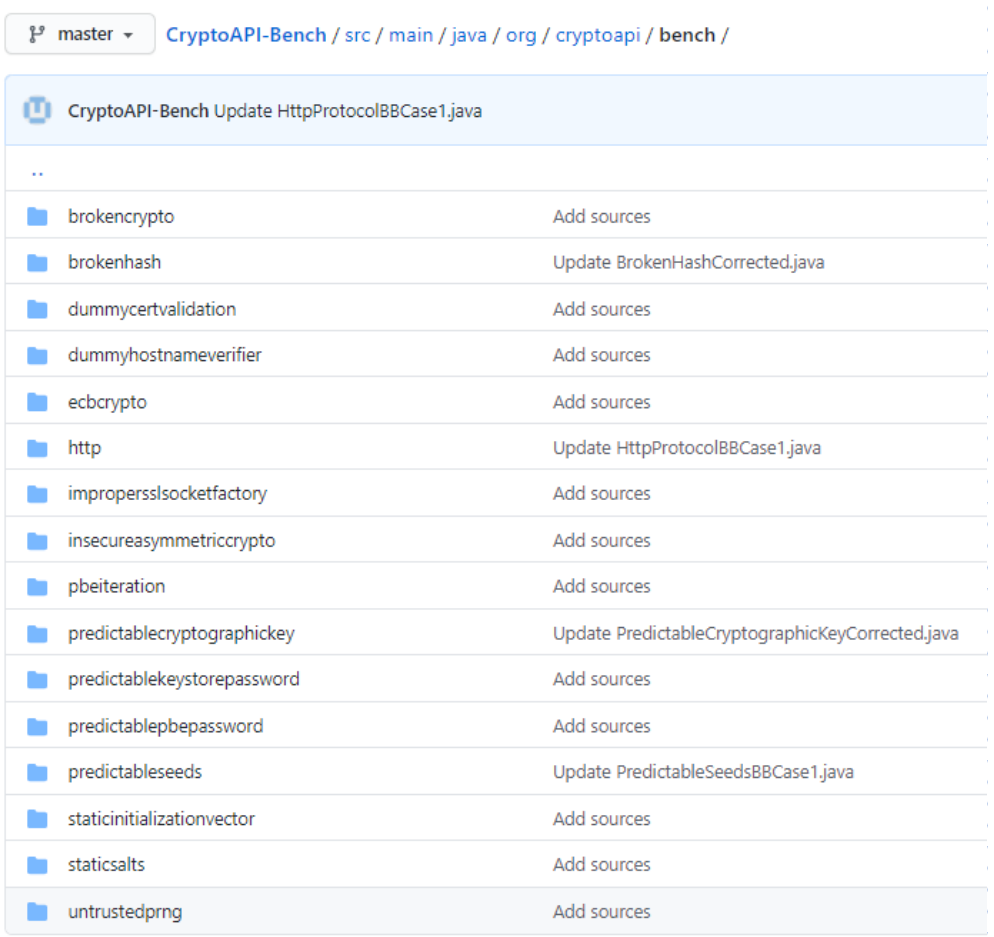
Compare different tools relative performance

Educate secure code VS insecure code

ESORICS

Benchmarks: Open-sourced





https://github.com/CryptoAPI-Bench/CryptoAPI-Bench
https://github.com/CryptoAPI-Bench/ApacheCryptoAPI-Bench

ESORICS

Test Cases: Detailed Information

master ▾ CryptoAPI-Bench / CryptoAPI-Bench_details.xlsx Go to file ···

CryptoAPI-Bench Add files via upload Latest commit 2760be4 on Apr 23, 2019 History

Files	Code Number	Vulnerability Exists?	Type of Vulnerability	Method name	Line number
PredictableCryptographicKeyBBCase1.java	1.1.1	TRUE	Static/Contant Key	main()	9, 12
PredictablePBEPasswordBBCase1.java	2.1.1	TRUE	Static/Contant password	key()	16, 22
PredictablePBEPasswordBBCase2.java	2.1.2	TRUE	Static/Constant Password	key()	16,22
PredictableKeyStorePasswordBBCase1.java	3.1.1	TRUE	Static/Constant Password	go()	23,24
DummyHostNameVerifierCase1.java	4.1.1	TRUE	Dummy Verifier	verify()	8
DummyCertValidationCase1.java	5.1.1	TRUE	Dummy Certificate	checkServerTrusted()	17
DummyCertValidationCase2.java	5.1.2	TRUE	Dummy Certificate	checkClientTrusted(), checkServerTrusted()	11,16
DummyCertValidationCase3.java	5.1.3	TRUE	Dummy Certificate	checkClientTrusted(), checkServerTrusted(), getAcceptedIssuers()	10, 15, 20
ImproperSocketManualHostBBCase1.java	6.1.1	TRUE	Socket Hostname w/o verification	main()	10
HttpProtocolBBCase1.java	7.1.1	TRUE	HTTP	main()	7

ESORICS

Test Cases: URL

- Cryptographic API: URL
- Vulnerability: Insecure website

```
String url = "http://insects.myspecies.info/";
System.out.println(new URL(url));
```

```
String url = "https://www.google.com";
System.out.println(new URL(url));
```

ESORICS

Test Cases: URL

- Cryptographic API: URL
- Vulnerability: Insecure website

Insecure

```
String url = "http://insects.myspecies.info/";
System.out.println(new URL(url));
```

```
String url = "https://www.google.com";
System.out.println(new URL(url));
```

ESORICS

Test Cases: URL

- Cryptographic API: URL
- Vulnerability: Insecure website

Insecure

```
String url = "http://insects.myspecies.info/";
System.out.println(new URL(url));
```

Secure

```
String url = "https://www.google.com";
System.out.println(new URL(url));
```

ESORICS

Test Cases: Random Number

- Cryptographic API: Random, SecureRandom
- Vulnerability: Predictable number generation

```
Random randomGenerator = new Random();
int x = randomGenerator.nextInt();
```

```
SecureRandom random = new SecureRandom();
int x = random.nextInt();
```

ESORICS

Test Cases: Random Number

- Cryptographic API: Random, SecureRandom
- Vulnerability: Predictable number generation

Insecure

```
Random randomGenerator = new Random();
int x = randomGenerator.nextInt();
```

```
SecureRandom random = new SecureRandom();
int x = random.nextInt();
```

- Follow definite mathematical algorithm
- Required attempt: 2^48

Predictable!

Break in practical time!

ESORICS

Test Cases: Random Number

- Cryptographic API: Random, SecureRandom
- Vulnerability: Predictable number generation

Insecure

```
Random randomGenerator = new Random();
int x = randomGenerator.nextInt();
```

- Follow definite mathematical algorithm
- Required attempt: 2^48

Predictable!

Break in practical time!

Secure

```
SecureRandom random = new SecureRandom();
int x = random.nextInt();
```

- Produce In-deterministic output
- Required attempt: 2^128

Unpredictable!

Need several years to break in!

ESORICS

Test Cases: Cryptographic Key

- Cryptographic API: SecretKeySpec
- Vulnerability: Constant cryptographic key

```
byte keyBytes[] = {20,10,30,5,5,6,8,7};
keyBytes = Arrays.copyOf(keyBytes,16);
SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
```

```
SecureRandom random = new SecureRandom();
String defaultKey = String.valueOf(random.ints());
byte[] keyBytes = defaultKey.getBytes();
keyBytes = Arrays.copyOf(keyBytes,16);
SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
```

ESORICS

Test Cases: Cryptographic Key

- Cryptographic API: SecretKeySpec
- Vulnerability: Constant cryptographic key

Insecure

```
byte keyBytes[] = {20,10,30,5,5,6,8,7};
keyBytes = Arrays.copyOf(keyBytes,16);
SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
```

```
SecureRandom random = new SecureRandom();
String defaultKey = String.valueOf(random.ints());
byte[] keyBytes = defaultKey.getBytes();
keyBytes = Arrays.copyOf(keyBytes,16);
SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
```

Cryptographic Key derived from

- Constant byte array
- Device ID
- Timestamp

Predictable

Insecure!

ESORICS

Test Cases: Cryptographic Key

- Cryptographic API: SecretKeySpec
- Vulnerability: Constant cryptographic key

Insecure

```
byte keyBytes[] = {20,10,30,5,5,6,8,7};
keyBytes = Arrays.copyOf(keyBytes,16);
SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
```

Cryptographic Key derived from

- Constant byte array
- Device ID
- Timestamp

Predictable

Insecure!

Secure

```
SecureRandom random = new SecureRandom();
String defaultKey = String.valueOf(random.ints());
byte[] keyBytes = defaultKey.getBytes();
keyBytes = Arrays.copyOf(keyBytes,16);
SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
```

Cryptographic Key derived from SecureRandom API

Unpredictable

Secure!

ESORICS

Test Cases: Message Digest

- Cryptographic API: MessageDigest(…)
- Vulnerability: Insecure cryptographic Hash

```
MessageDigest md = MessageDigest.getInstance("MD5");
md.update(name.getBytes());
System.out.println(md.digest());
```

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
md.update(name.getBytes());
System.out.println(md.digest());
```

ESORICS

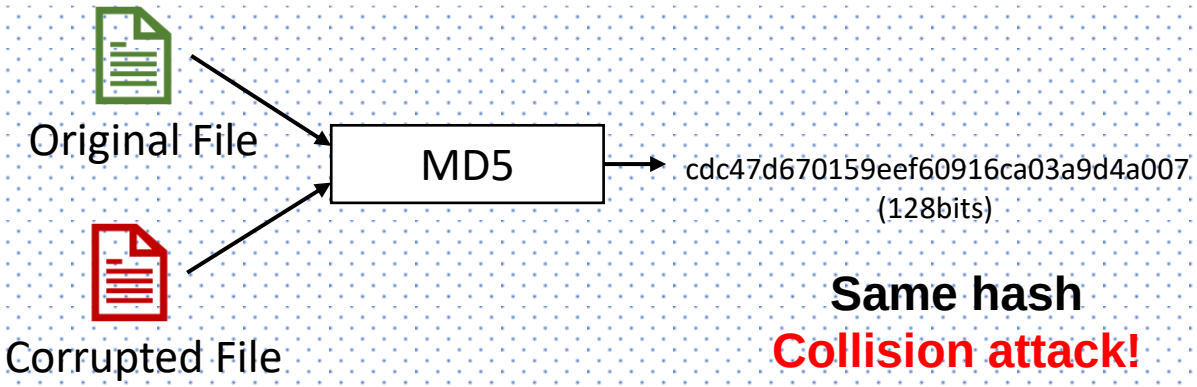
Test Cases: Message Digest

- Cryptographic API: MessageDigest(…)
- Vulnerability: Insecure cryptographic Hash

Insecure

```
MessageDigest md = MessageDigest.getInstance("MD5");
md.update(name.getBytes());
System.out.println(md.digest());
```

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
md.update(name.getBytes());
System.out.println(md.digest());
```



Same hash

Collision attack!

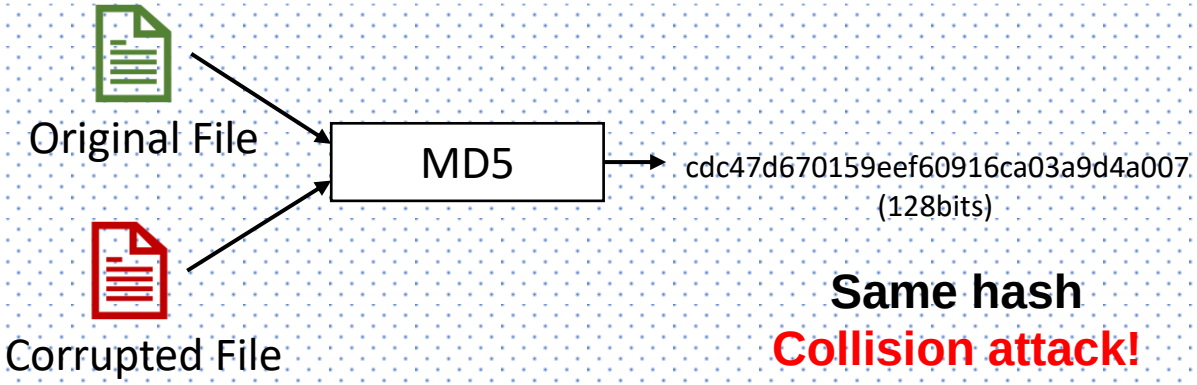
ESORICS

Test Cases: Message Digest

- Cryptographic API: MessageDigest(…)
- Vulnerability: Insecure cryptographic Hash

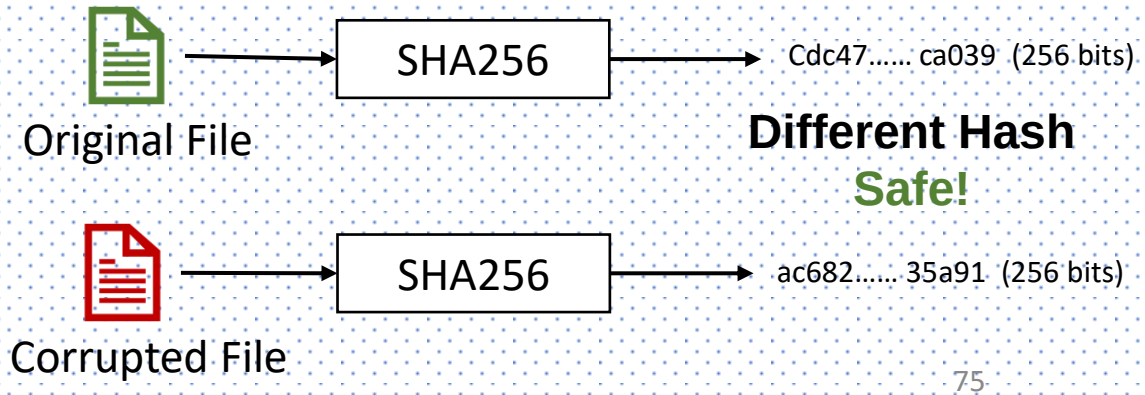
Insecure

```
MessageDigest md = MessageDigest.getInstance("MD5");
md.update(name.getBytes());
System.out.println(md.digest());
```



Secure

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
md.update(name.getBytes());
System.out.println(md.digest());
```



ESORICS

Test Cases: Cipher

- Cryptographic API: Cipher
- Vulnerability: Insecure cryptographic cipher algorithm

```
Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, key);
```

```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, key);
```

ESORICS

Test Cases: Cipher

- Cryptographic API: Cipher
- Vulnerability: Insecure cryptographic cipher algorithm

Insecure

```
Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, key);
```

Secure

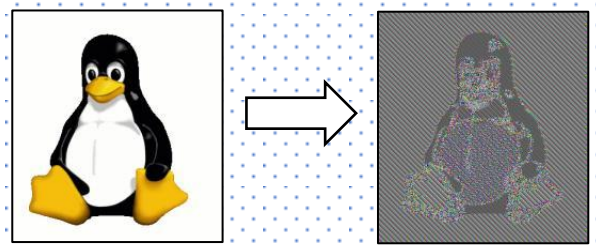
```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, key);
```

- **DES Encryption:**
  - Key size: 56

Bruteforce attack!

- **ECB Mode of Operation:**

Leak plaintext information!



Original

Using ECB

ESORICS

Test Cases: Cipher

- Cryptographic API: Cipher
- Vulnerability: Insecure cryptographic cipher algorithm

Insecure

```
Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, key);
```

Secure

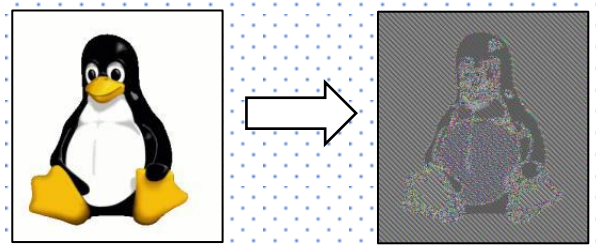
```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, key);
```

- **DES Encryption:**
  - Key size: 56

Bruteforce attack!

- **ECB Mode of Operation:**

Leak plaintext information!



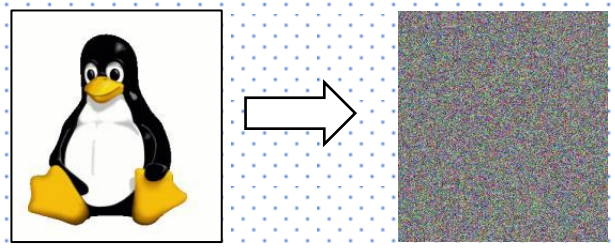
Original Using ECB

- **AES Encryption:**
  - Key size: 128, 192, 256

More Secure!

- **CBC Mode of Operation:**

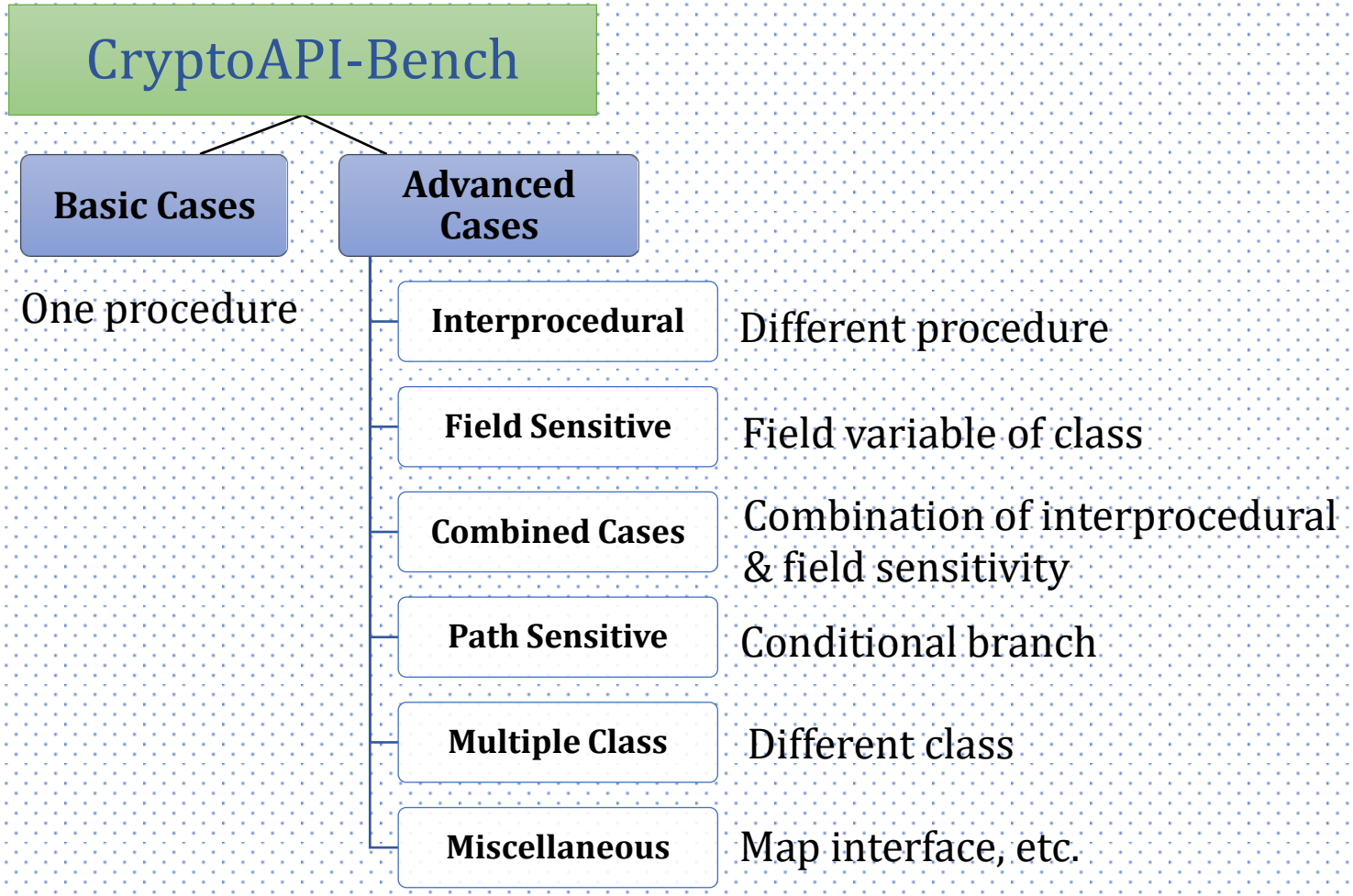
Random Appearance!



Original Using CBC

ESORICS

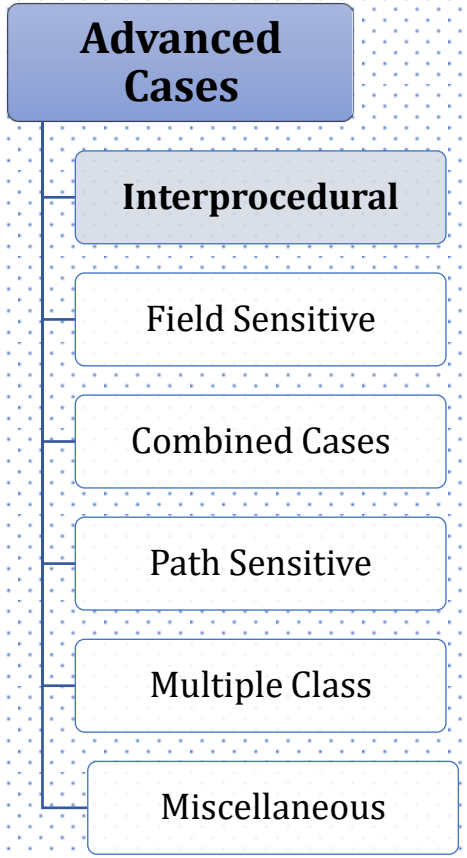
CryptoAPI-Bench: Structure



ESORICS

CryptoAPI-Bench: Interprocedural

```
1  public static void main(){
2      LessThan1000IterationPBEABICase1 lt = new LessThan1000IterationPBEABICase1();
3      int count = 20;
4      lt.go(count);
5  }
6  public void go(int count){
7      SecureRandom random = new SecureRandom();
8      PBEParameterSpec pbeParamSpec = null;
9      byte[] salt = new byte[32];
10     random.nextBytes(salt);
11
12     pbeParamSpec = new PBEParameterSpec(salt, count);
13 }
```



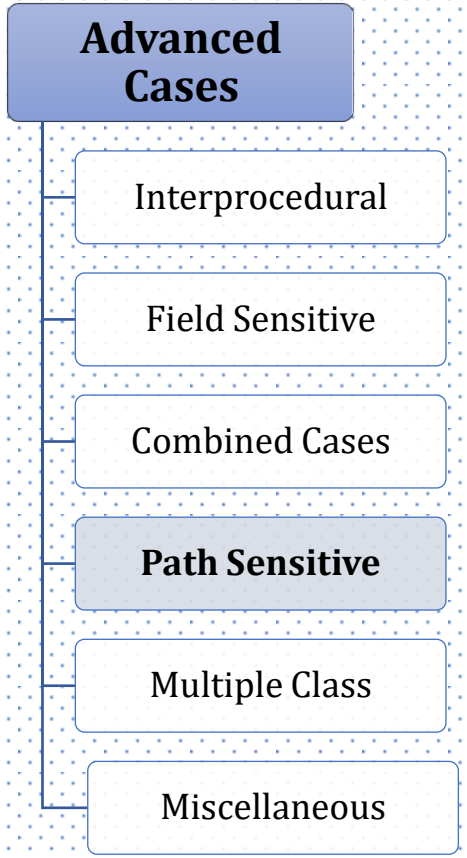
Iteration count value passed to another procedure

ESORICS

CryptoAPI-Bench: Path Sensitive

```
1  int count = 5;
2  SecureRandom random = new SecureRandom();
3  random.nextBytes(salt);
4  if(choice > 1)
5      count = 1050;
6
7  PBEParameterSpec pbeParamSpec = null;
8  pbeParamSpec = new PBEParameterSpec(salt, count);
```

Iteration count value is determined from conditional statement



ESORICS

Evaluation

CryptoAPI-Bench: Basic cases in (6 common rules):

Tools	SpotBugs	CryptoGuard	CrySL	Coverity
Recall (%)	92.86	92.86	71.43	92.86
Precision (%)	100.00	100.00	62.50	100.00

CryptoAPI-Bench: Advanced cases in (6 common rules):

Tools	SpotBugs	CryptoGuard	CrySL	Coverity
Recall (%)	0.00	95.59	58.82	19.12
Precision (%)	0.00	83.33	56.34	52.00

None designed to handle path sensitive cases

ApacheCryptoAPI-Bench: (6 common rules):

Tools	SpotBugs	CryptoGuard	CrySL	Coverity
Recall (%)	87.50	93.75	93.75	81.25
Precision (%)	100.00	100.00	50.00	81.25

Majority cases are Basic Cases

S. Afrose, S. Rahaman, and D. Yao. "CryptoAPI-Bench: A Comprehensive Benchmark on Java Cryptographic API Misuses." 2019 IEEE Cybersecurity Development (SecDev). IEEE, 2019
S. Afrose, Y. Xiao, S. Rahaman, and D. Yao. "Development of Benchmarks for Java Cryptographic APIs and Evaluation of Static Vulnerability Detection Tools" (Under Review)
Version: Cryptoduard: Commit id 97b220 ; CrySL: Commit id 004cd2 ; SpotBugs: Version 3.1.12 ; Coverity: September 2020

ESORICS

Python Static Analysis via Cryptolation

By: Miles Frantz

ESORICS

Analyzing Python

- Java is strongly typed
- Python is a weakly typed

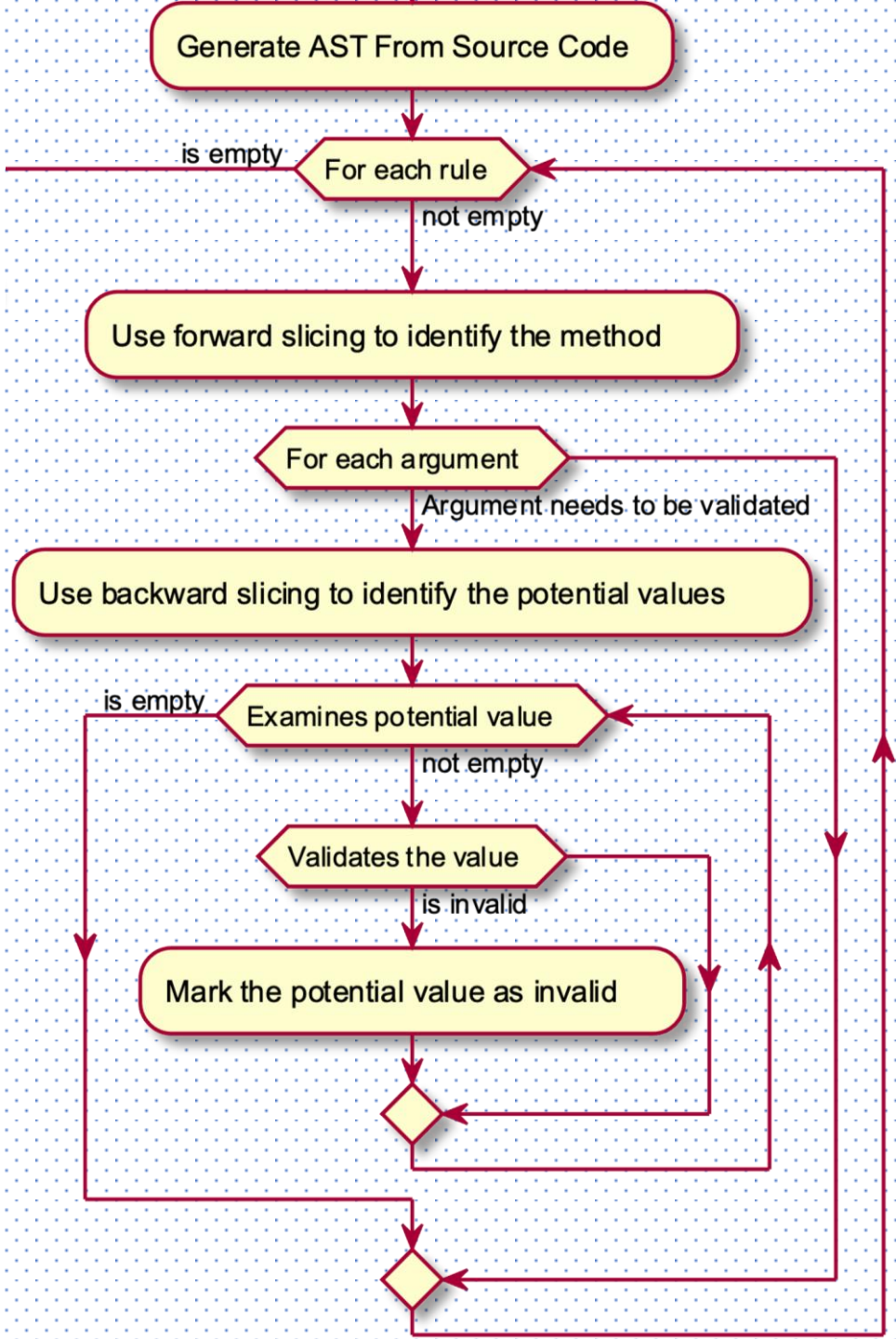
- Functions are treated as first class objects within Python

- Java is compiled
- Python is Interpreted

ESORICS

Cryptolation Structure

- Scans the source code of files
- Cycles through the different types of arguments
  - Keyword
  - Optional
  - Non-Optional
- Validates the specific argument based on
  - Type
  - Rule



ESORICS

Import Difference

How are Java and Python import statements different?

```
3 import java.math.BigInteger;
4 import java.security.MessageDigest;
5 import java.security.NoSuchAlgorithmException;
6 import java.nio.charset.StandardCharsets;
```

- Java can import using wild card statements
- Java can only import at the top of the file
- Java has one basic formula for imports

```
2 from hashlib import sha512
8 from hashlib import sha1 as sha512
```

- Python has multiple formulas for imports
- Python can rename imports
- Python can import at a local scope

ESORICS

Ambiguity in imports

Can imports be malicious (or accidentally misused)?

```
2  from hashlib import sha512
3
4  message = sha512()
5  message.update(b"Hello World")
6  print(f"Hashed by SHA512: {message.digest()}")
7
8  from hashlib import sha1 as sha512
9
10 message = sha512()
11 message.update(b"Hello World")
12 print(f"Hashed by SHA512: {message.digest()}")
```

- Line 2 and 8 import hash libraries
- Line 8 imports sha1 as sha512

- The message at line 5 is hashed using sha512
- The message at line 11 is hashed using sha1

ESORICS

Path Sensitivity

Will only happy paths be analyzed?

```
2  import urllib.request
3
4  if False:
5      url = 'https://www.google.com'
6  else:
7      url = 'http://www.google.com'
8
9  req = urllib.request.Request(url)
```

- The url is slightly changed based on the conditional
- The static analyzer has to determine the correct path flow or evaluate both conditions

ESORICS

Malicious Path Sensitivity

Can this be misused or mis-analyzed?

```
2  import urllib.request
3
4  if False:
5      url = 'https://www.google.com'
6  else:
7      url = 'http://www.google.com'
8
9  req = urllib.request.Request(url)
```

- The code path listed to the left is simple to understand
- Looking at the code we understand the requested url is not secure
- Standard security guidelines tell us to use https instead of http

ESORICS

Locality Issues

Can local based imports be an issue?

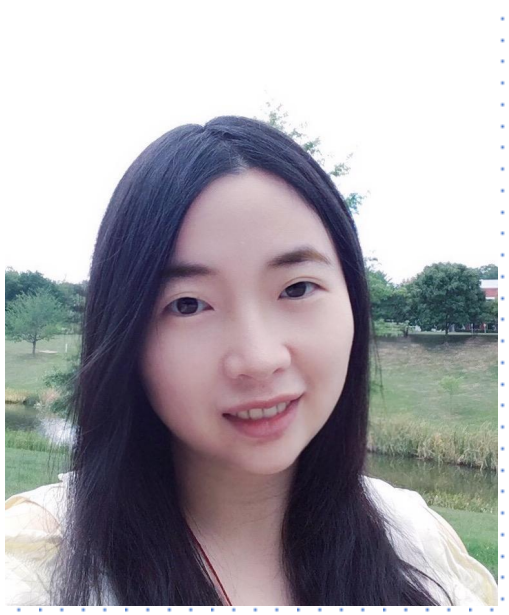
- The code listed below uses a local import to create a hmac using MD5
- The MD5 import is renamed as SHA512

```
3  def self_hash(string):
4      import hmac
5      from hashlib import MD5 as SHA512
6
7      return hmac.new(bytearray(string, 'utf-8'), msg=None, digestmod=SHA512)
8
9  if __name__ == "__main__":
10     print(self_hash('Hello'))
```

ESORICS

Parfait-CryptoScanner Design/Results

Presenter:
Ya Xiao



What does industrial strength code scanner look like?

Oracle's Parfait – an industrial strength static analysis tool for software security (started in 2007)

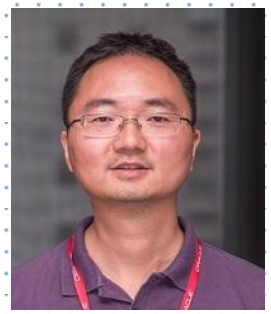
Parfait is fast -- analyzing 10.6 million of lines of code in 80 mins on a 2.9GHz AMD computer

Parfait is precise -- average false positive rate < 10%

Cristina Cifuentes and her team

https://labs.oracle.com/pls/apex/f?p=labs:49:::::P49_PROJECT_ID:13

Oracle Lab Australia implemented CryptoGuard's approach (2019) to scan production code


arXiv.org > cs > arXiv:2007.06122

Search...

Help | Advan

Computer Science > Software Engineering

[Submitted on 12 Jul 2020]

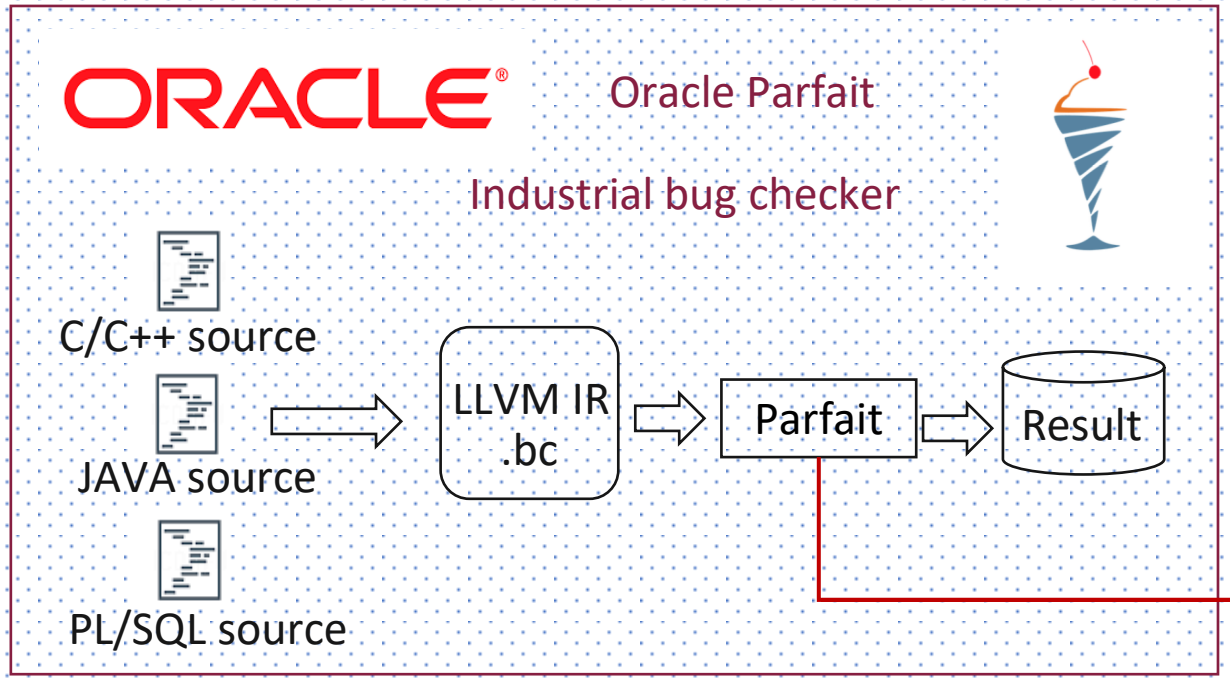
Industrial Experience of Finding Cryptographic Vulnerabilities in Large-scale Codebases

Ya Xiao, Yang Zhao, Nicholas Allen, Nathan Keynes, Danfeng (Daphne)Yao, Cristina Cifuentes

Enterprise environments need to screen large-scale (millions of lines of code) codebases for vulnerability detection, resulting in high requirements for precision and scalability of a static analysis tool. At Oracle, Parfait is one such bug checker, providing precision and scalability of results, including interprocedural analyses. CryptoGuard is a precise static analyzer for detecting cryptographic vulnerabilities in JavaTM1 code built on Soot. In this paper, we describe how to integrate CryptoGuard into Parfait, with changing intermediate representation and relying on a

https://arxiv.org/abs/2007.06122

We strengthen the Crypto Vulnerability Detection Module in Parfait



Bug Pass 1

Crypto Vulnerability Detection

...

Bug Pass n

Our work

An *interprocedural, flow-, context-, field-sensitive dataflow analysis.*

IFDS Analysis

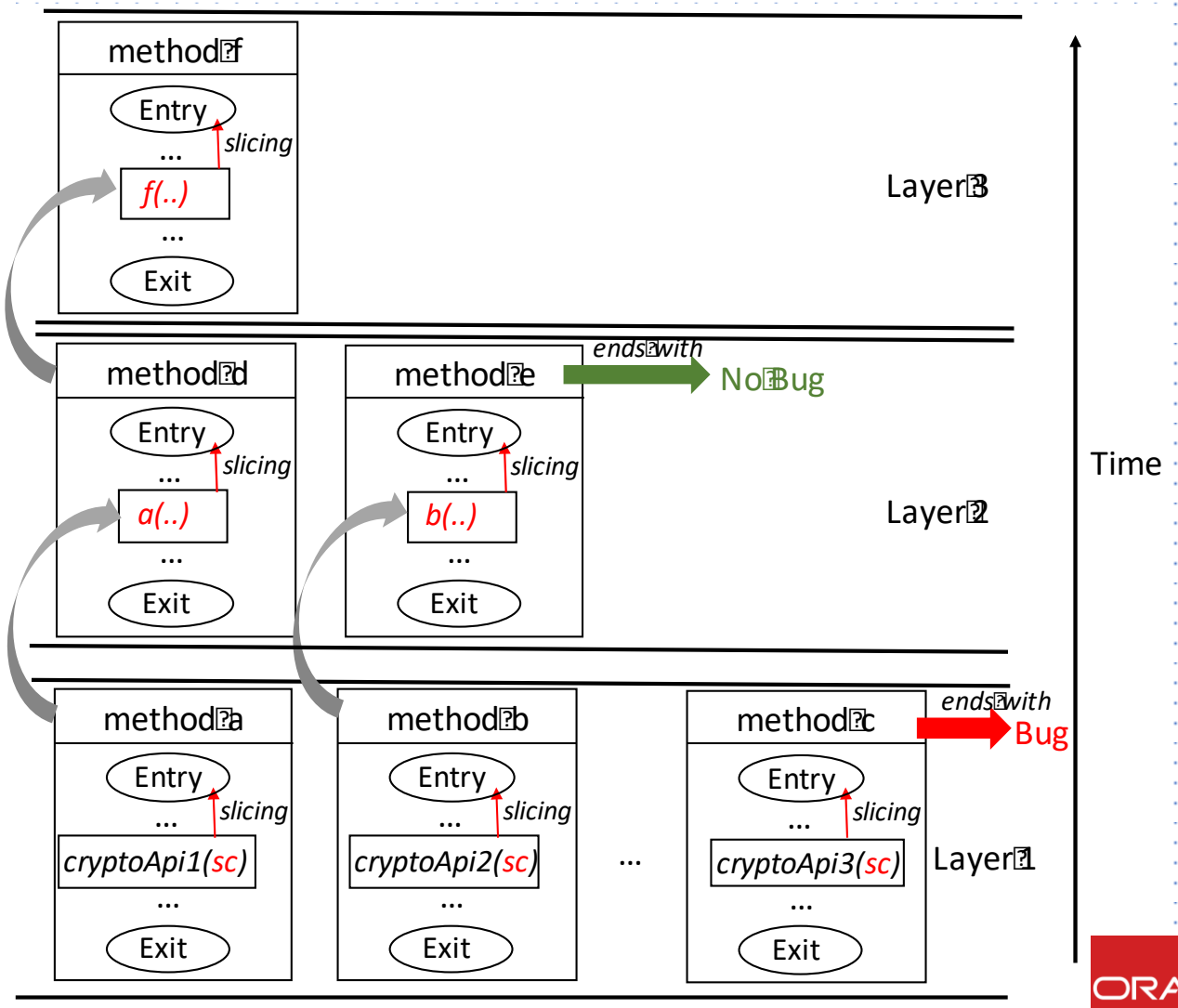
Refinement Insights

The refinement insights to remove pseudo-influences that cause false positives

Industrial strength scalability enabled by Parfait's Layered Framework Design

Schedule scanning tasks from the quickest to the slowest

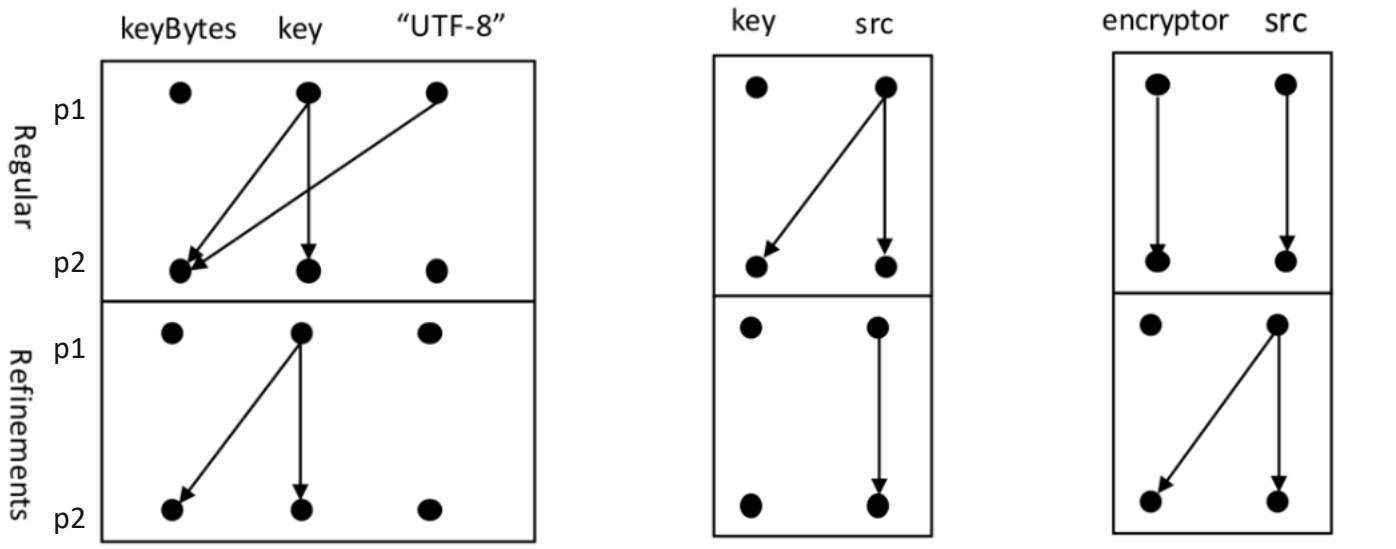
"I'll use your tool only if scanning completes overnight." – from an Oracle developer



Refined IFDS Analysis

- *Design for precisio*n: We specialize the IFDS analysis propagation through refinement insights to remove these pseudo-influences.

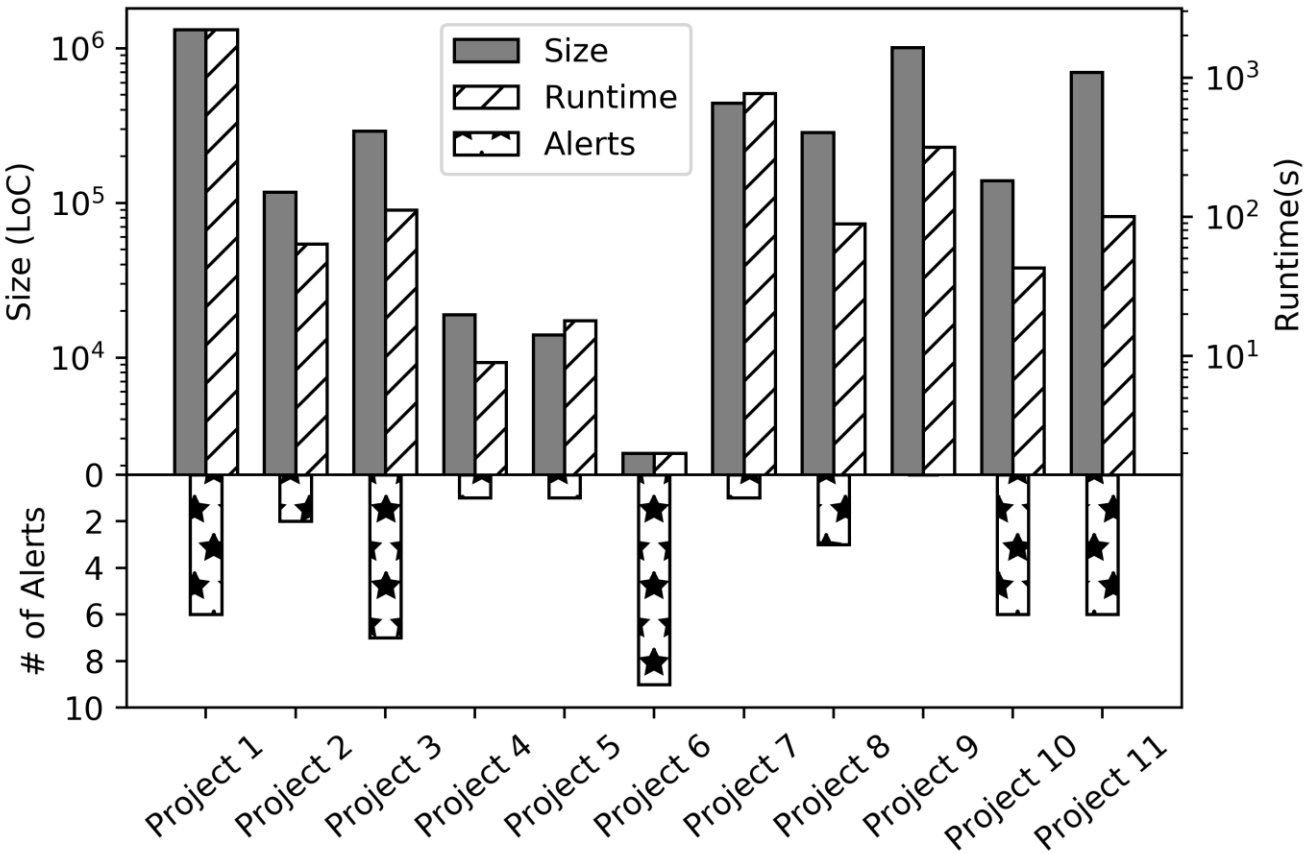
Five types of pseudo-influences in the work [CryptoGuard 2019]
- State indicators
- Resource identifiers
- Bookkeeping indices
- Contextually incompatible constants
- Constants in infeasible paths



(a) byte[] keyBytes = key.getBytes("UTF-8") **(b)** String key = Context.getProperty (src)**(c)** encryptor.<init>(src)

Results of Parfait's crypto scanning 11 internal Oracle projects (Java) -- detection approach based on CryptoGuard

- Scanned 11 projects; reported 42 vulnerabilities with 0 false positive (100% precision)
- Average runtime 338.8s for 11 projects with average 395.4k LoC



Scanning on Oracle internal projects

Parfait's benchmark evaluation (on CryptoAPI-Bench)

How many actual vulnerabilities are reported? Higher the better ☺

98.4% Recall

86.6% Precision -- **100%** precision if excluding path sensitive cases

How many reported alerts are real vulnerabilities? Higher the better ☺

Type	Total Cases	Insecure Cases	Secure Cases	Reported Cases	False Positives	False Negatives	Precision	Recall
Basic Cases	27	24	3	24	0	0	100%	100%
Multiple methods	57	56	1	54	0	2	100%	96.43%
Multiple Classes	23	18	5	18	0	0	100%	100%
Field Sensitivity	19	18	1	18	0	0	100%	100%
Path Sensitivity	19	0	19	19	19	0	0 %	0 %
Heuristics	13	9	4	9	0	0	100%	100%
Total	**158**	**125**	**33**	**142**	**19**	**2**	**86.62%**	**98.40%**

ESORICS

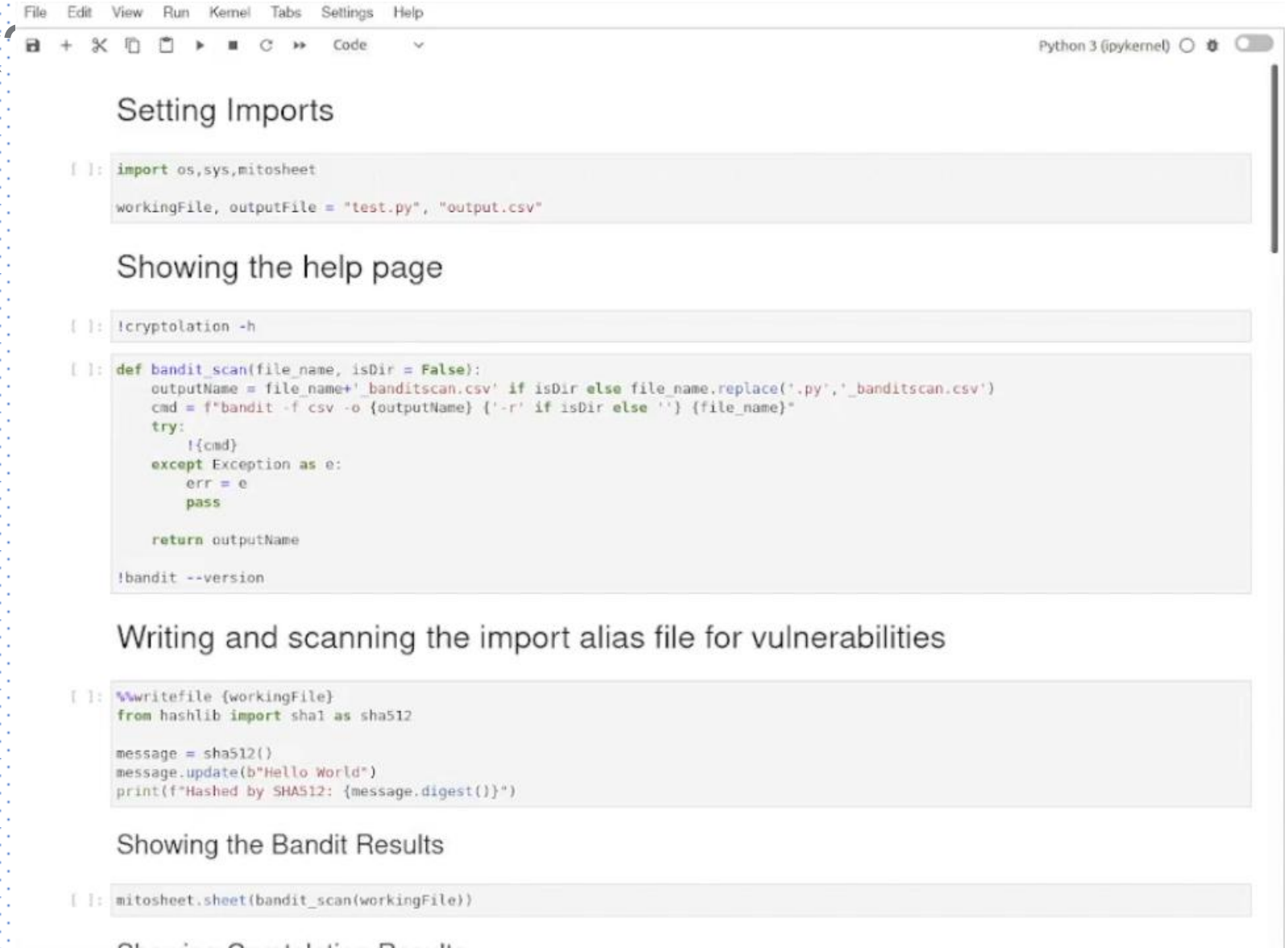
Live Demo

By: Miles Frantz

ESORICS

Running the code

How does this look‘



ESORICS

Questions?