IEEE SecDev 2020

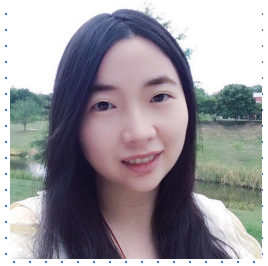
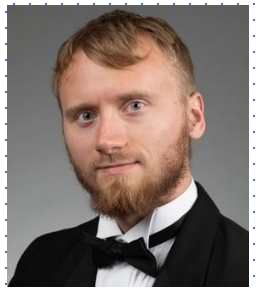
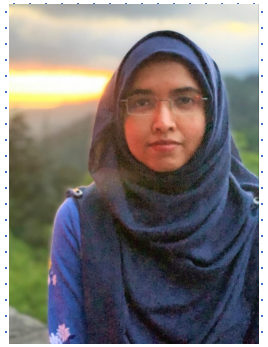
SaTC-1929701

Tutorial: Principles and Practices of Cryptographic Coding in Java

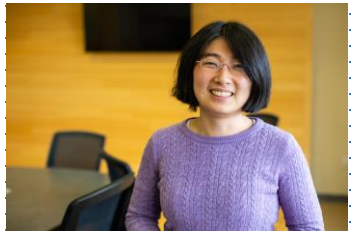
Ya Xiao, Miles Frantz, Sharmin Afrose
Ph.D. students
Virginia Tech

Daphne Yao
Professor
Virginia Tech

Sazzadur Rahaman
Assistant Professor
University of Arizona






Software is everywhere

Ford GT has over 10 million lines of code

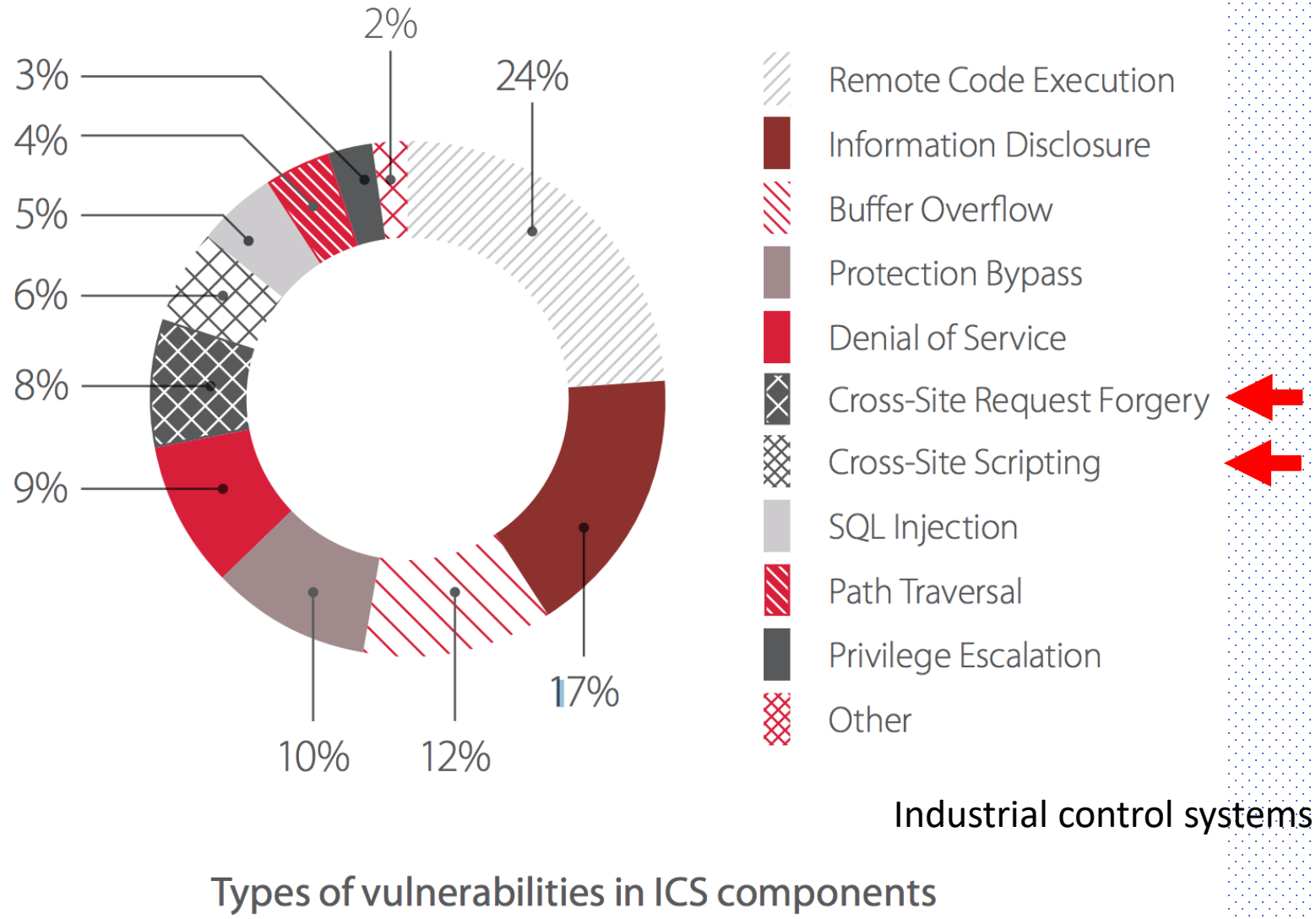
F-22 Raptor has 2 million lines of code

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)

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

Hacking

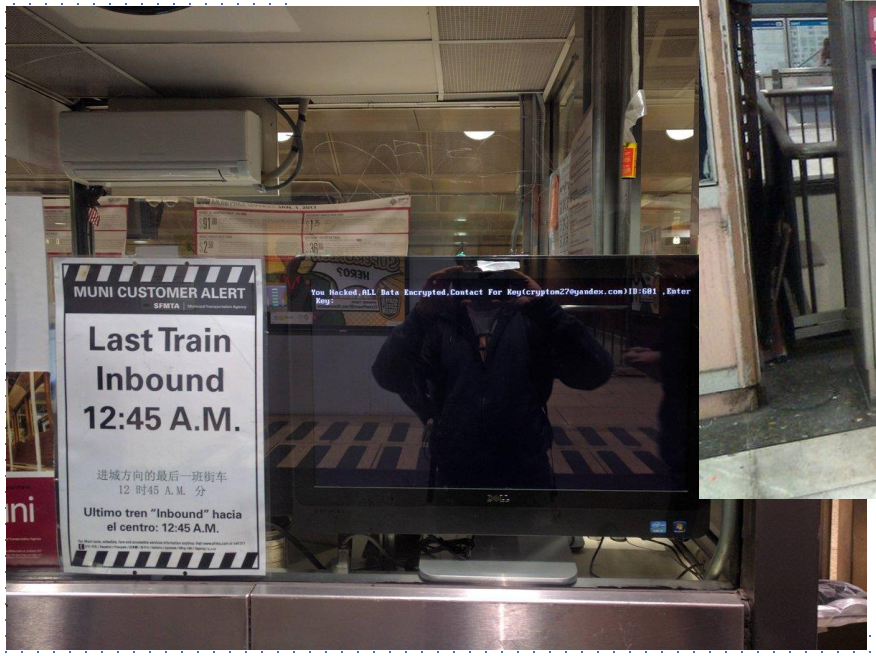
Ransomware attack on San Francisco public transit gives everyone a free ride

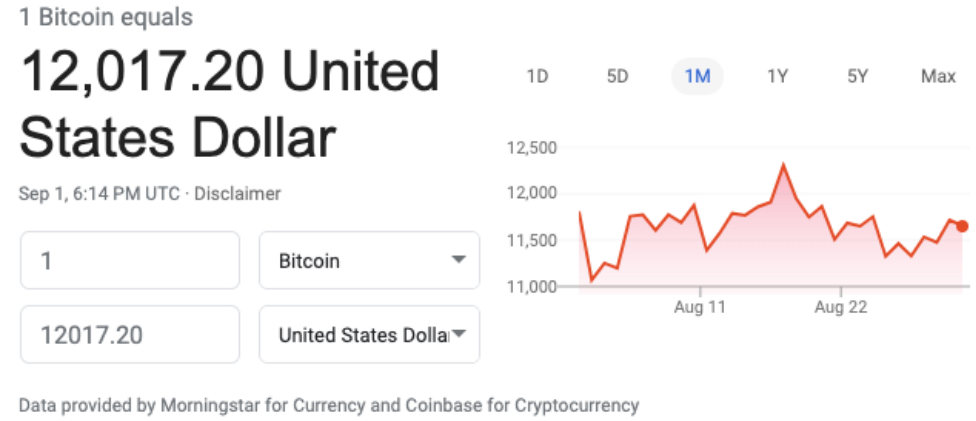
Nov 2016

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."



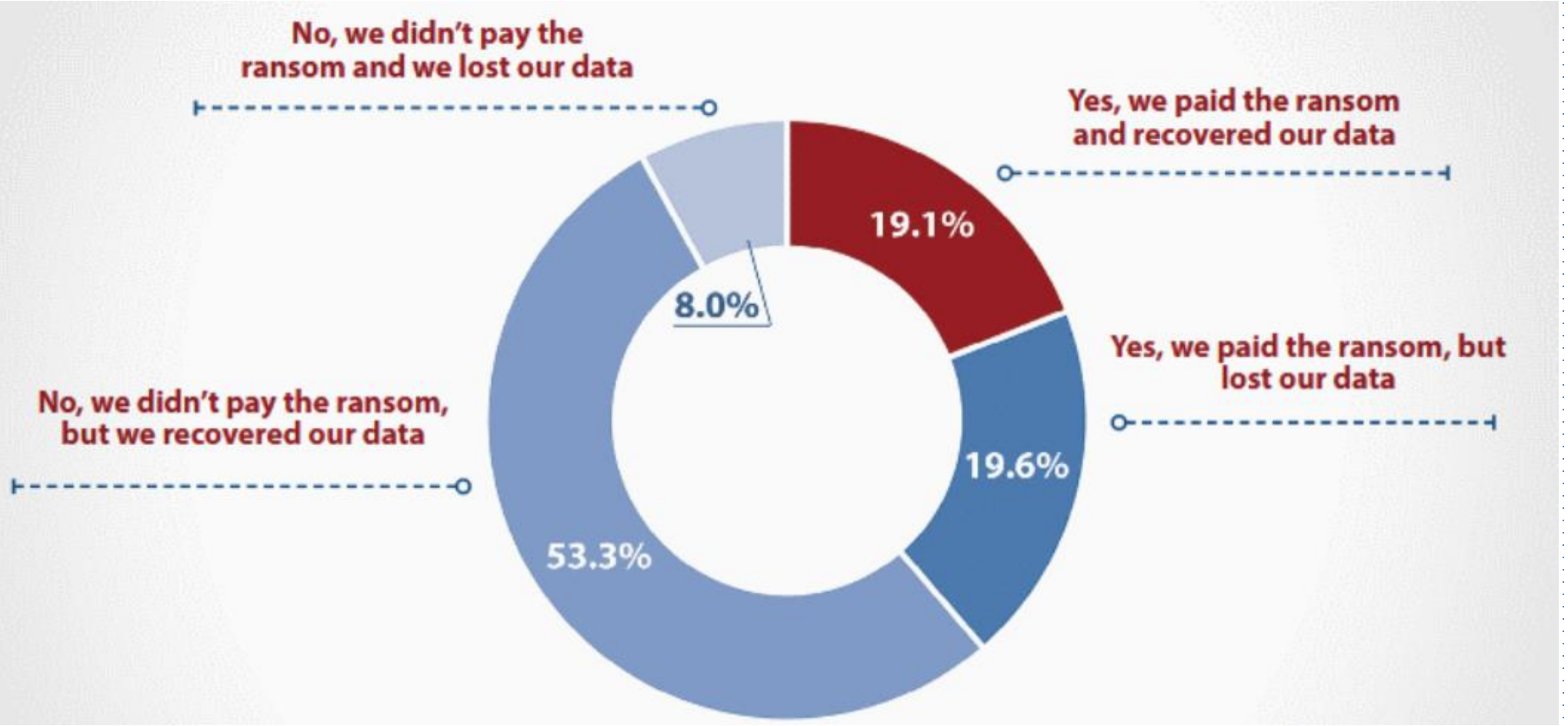


https://www.theguardian.com/technology/2016/nov/28/passengers-free-ride-san-francisco-muni-ransomeware

IEEE SecDev|2020

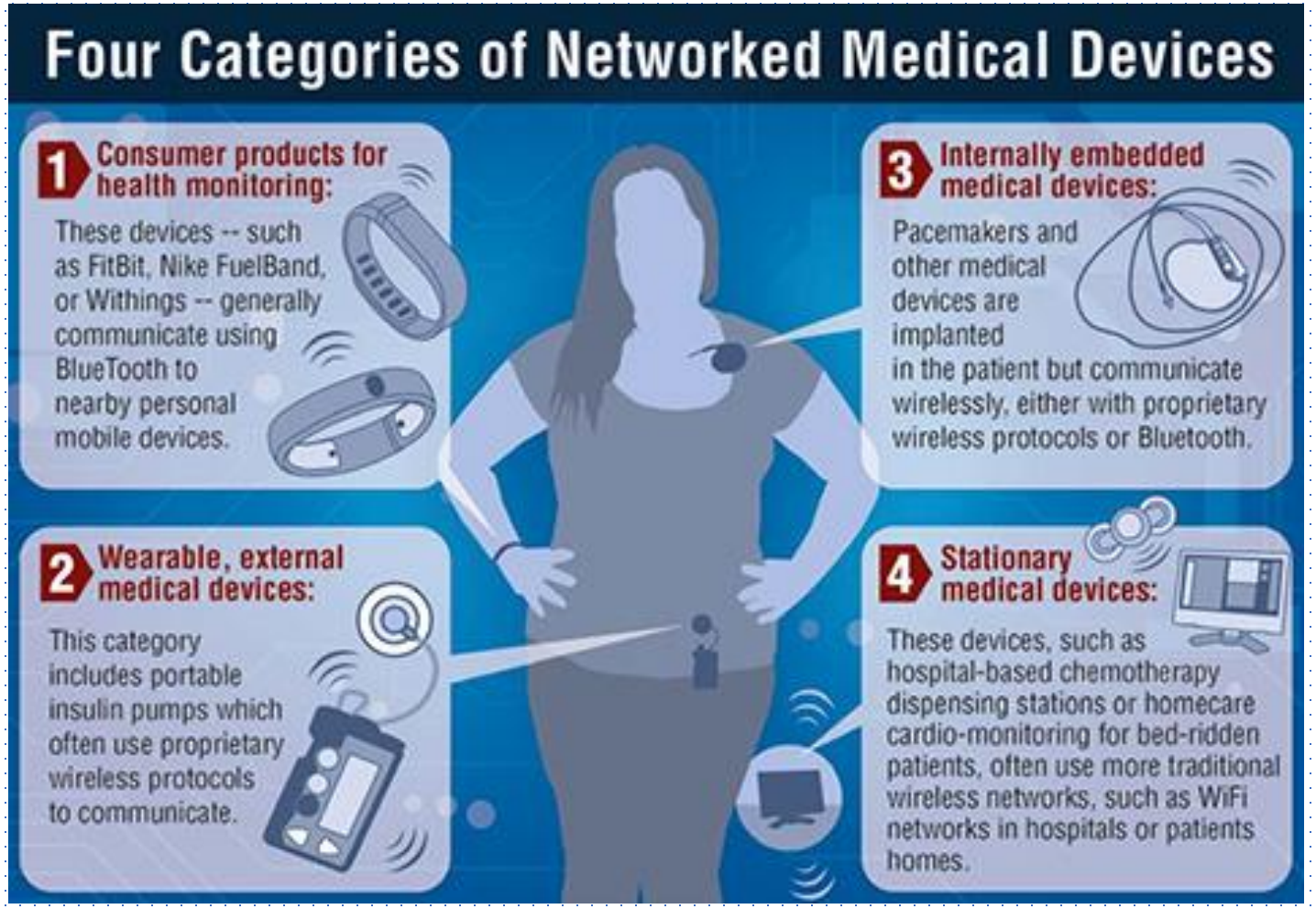
To pay or not to pay? That's the question

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/

Developers' code is getting closer and closer to your body



We need both -- developer training & using tools

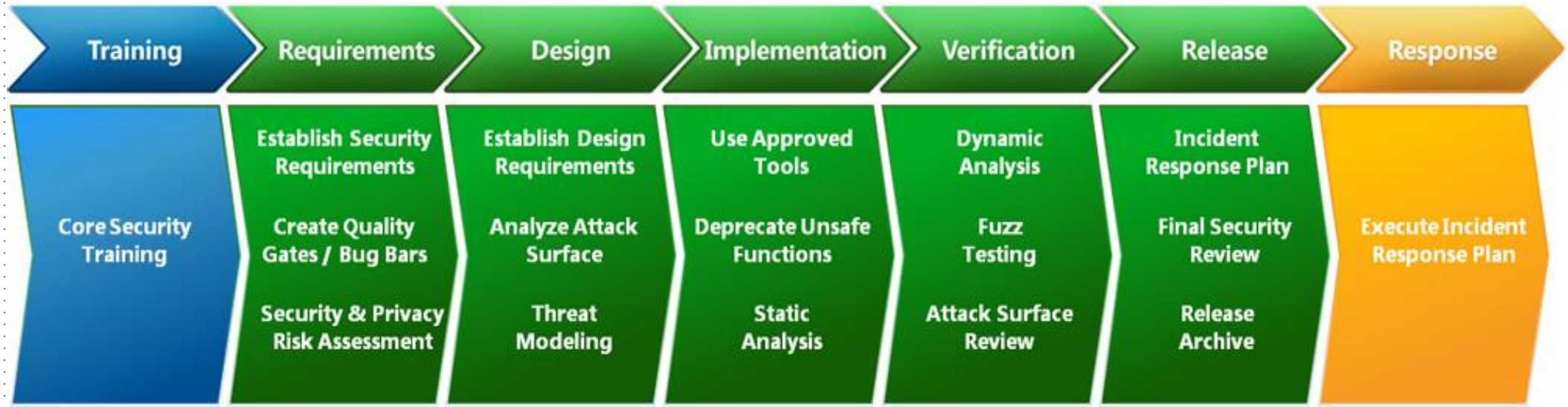
Top 10 secure coding rules

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

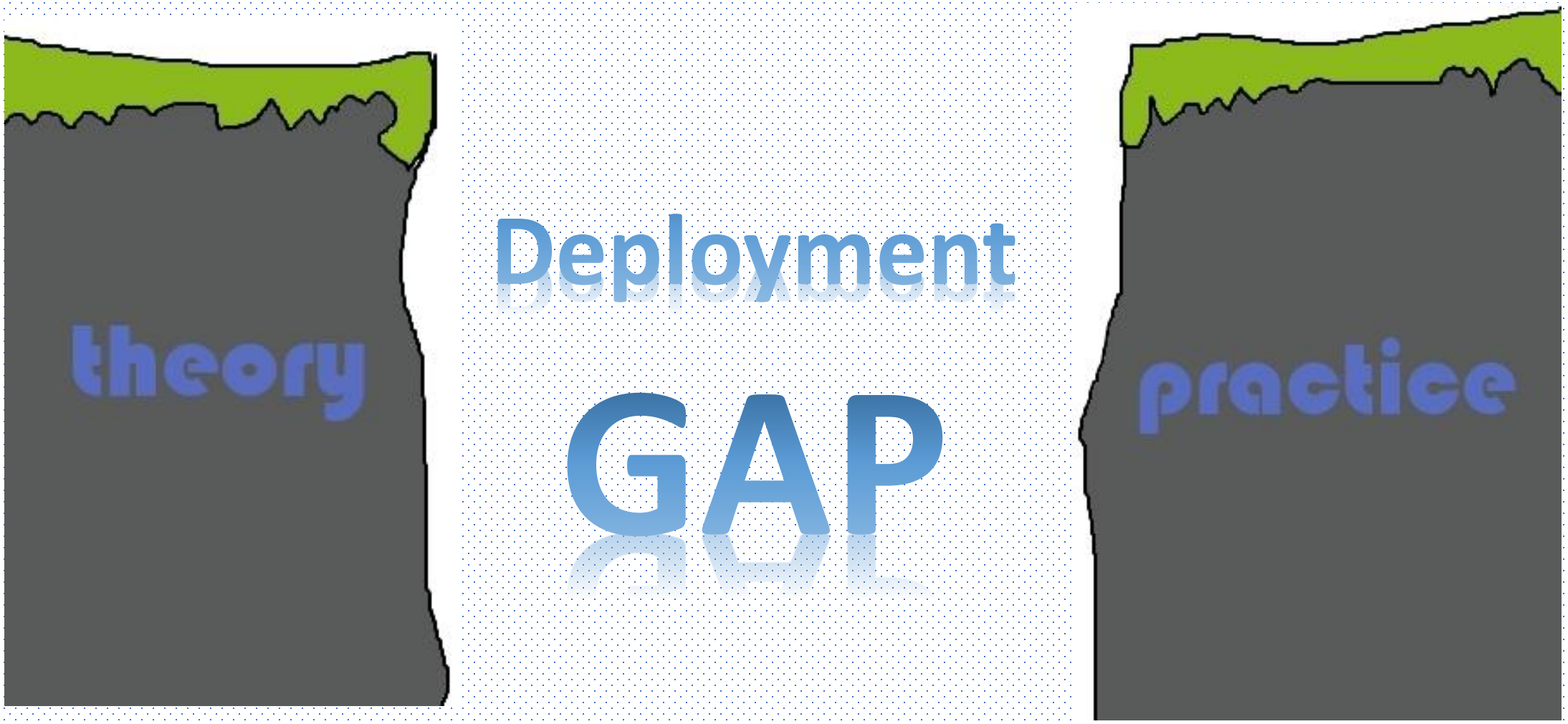
Who wouldn't want to write secure code?

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?

www.attacker.com

Victim Browser

1. Victim has a valid session with bank.com

www.bank.com

GET /blog HTTP/1.1

<form action=https://www.bank.com/transfer
method=POST target=invisibleframe>
<input name=recipient value=attacker>
<input name=amount value=$100>
</form>
<script>document.forms[0].submit()</script>

POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
recipient=attacker&amount=$100

HTTP/1.1 200 OK

Transfer complete!

2. Victim visited a malicious form

3. Victim tricked into submitting form

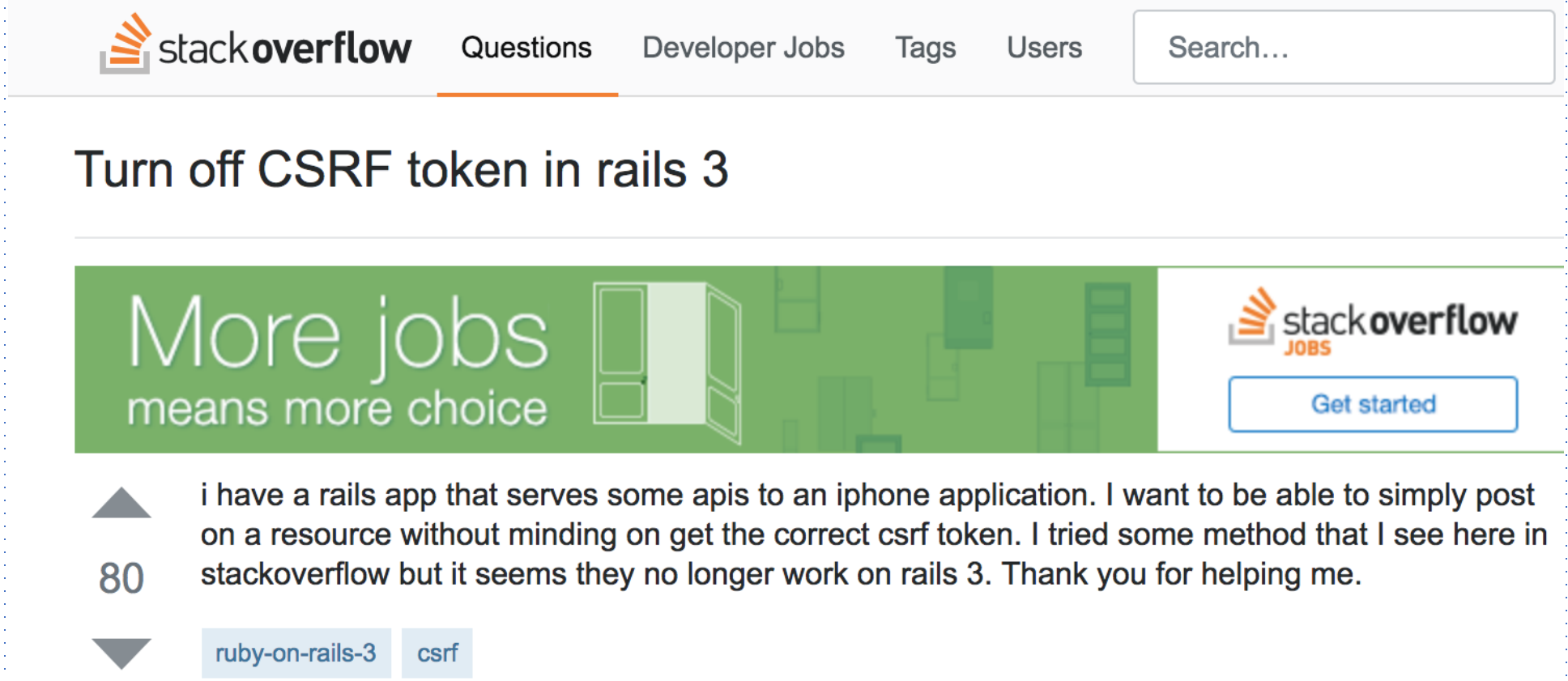
4. Browser automatically attaches session-id

5. Money transferred to attacker ☹

From C. Jackson

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

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

The paparazzi doesn't help

The Register®
Biting the hand that feeds IT

TER SOFTWARE **SECURITY** TRANSFORMATION DEVOPS BUSINESS PERSONAL TECH

Security

Java security plagued by crappy docs, complex APIs, bad advice

Boffins bash stale Stack Overflow fixes and lazy developers

By Thomas Claburn in San Francisco 29 Sep 2017 at 21:14 51 SHARE ▼

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

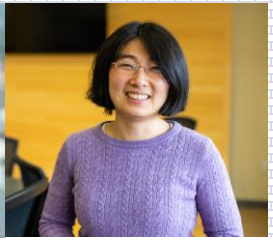
Gradle

IJ



Nominated for NSA Science of Security Paper Competition





Office of Naval Research Science & Technology

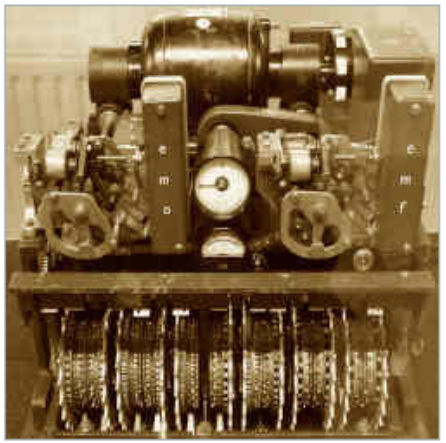
NSF

[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

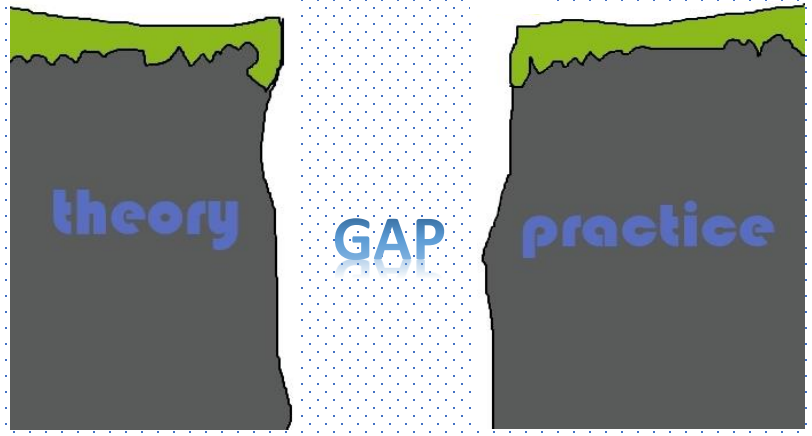
Identifying cryptographic vulnerabilities in today's million-line programs has become a critical endeavor. Because of the increasing sophistication of cybercriminals, programmers can no longer afford to test for vulnerabilities using only traditional debugging techniques, followed by releasing software,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

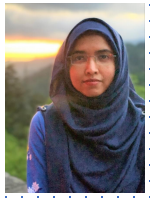
Most open-source automated vulnerability checkers are still finding their way, but a team of researchers at the Virginia Polytechnic Institute and State University (Virginia Tech) claim to have vulnerability-checking software that is mature, and approaching deployment. Called CryptoGuard, the software 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

Our tutorial today



(In)secure crypto coding examples



Secure TLS coding strategies

CryptoGuard intro/demo

Tool eval benchmark

Take-home message:

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

Related references

Papers:

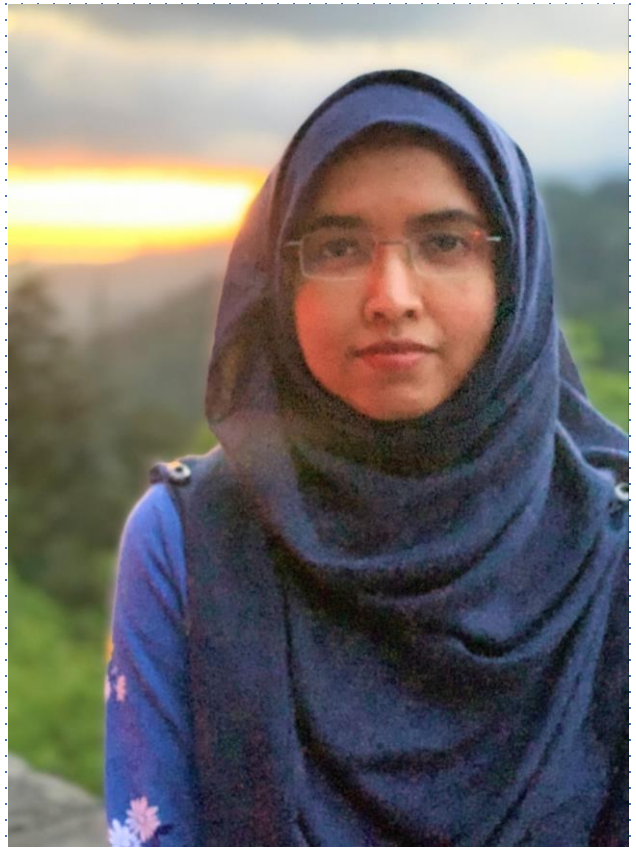
- Sazzadur Rahaman, Ya Xiao, Sharmin Afrose, Fahad Shaon, Ke Tian, Miles Frantz, Murat Kantarcioglu, and Danfeng Yao. "Cryptoguard: High precision detection of cryptographic vulnerabilities in massive-sized Java projects." In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications 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).

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

Simple Secure
vs.
Insecure Example

Presenter:
Sharmin Afrose



Code Examples for URL

Simple Secure vs. Insecure Example

Secure or 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));
```

Simple Secure vs. Insecure Example

- 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));
```

Code Examples for Random Numbers

Simple Secure vs. Insecure Example

Secure or insecure?

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

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

Simple Secure vs. Insecure Example

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

Insecure

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

Secure

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

Code Examples for Message Digests

Simple Secure vs. Insecure Example

Secure or 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());
```

Simple Secure vs. Insecure Example

- 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());
```

Code Examples for Ciphers

Simple Secure vs. Insecure Example

Secure or insecure?

```
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);
```

Simple Secure vs. Insecure Example

- 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);
```

Code Examples for Cryptographic Keys

Simple Secure vs. Insecure Example

Secure or insecure?

```
String defaultKey = "SecDev2020";
byte[] keyBytes = defaultKey.getBytes();
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");
```

Simple Secure vs. Insecure Example

- Cryptographic API: SecretKeySpec
- Vulnerability: Constant cryptographic key

Insecure

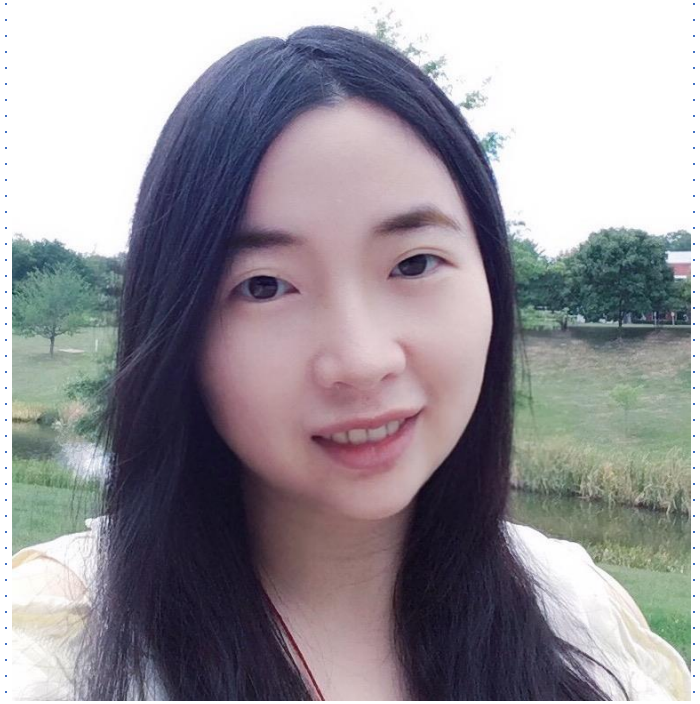
```
String defaultKey = "SecDev2020";
byte[] keyBytes = defaultKey.getBytes();
keyBytes = Arrays.copyOf(keyBytes,16);
SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
```

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");
```

TLS/SSL Authentication Code in JSSE

Presenter:
Ya Xiao



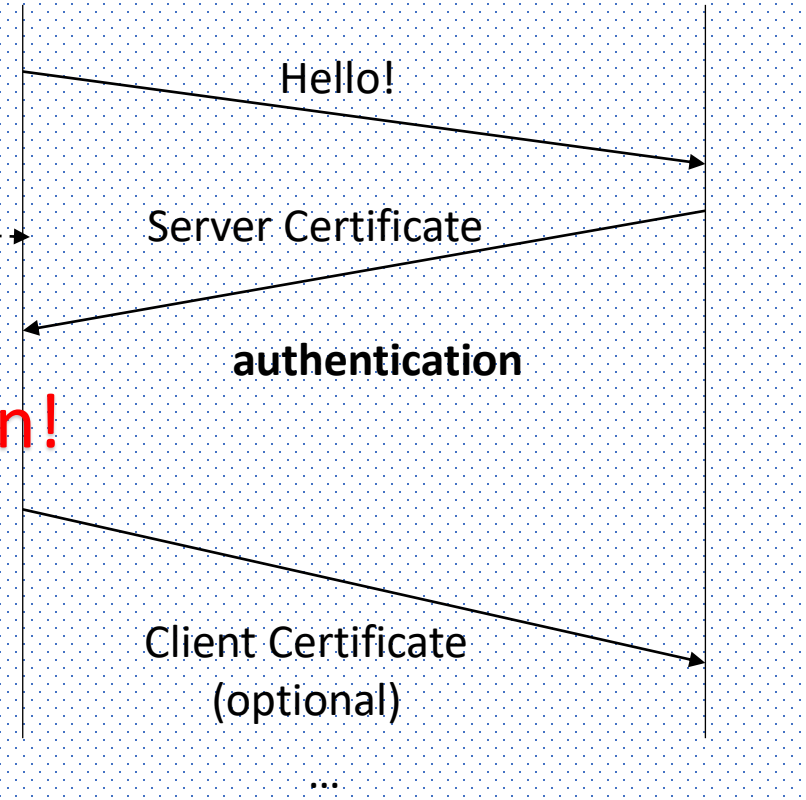
Mis-configuration of TLS/SSL can cause man-in-the-middle attacks.

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.

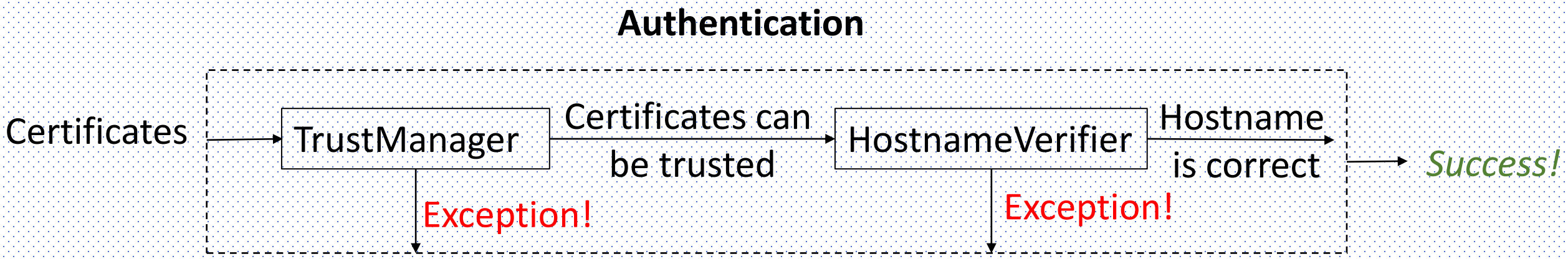
TLS/SSL happens implicitly in a code snippet

HTTPS = HTTP + TLS

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

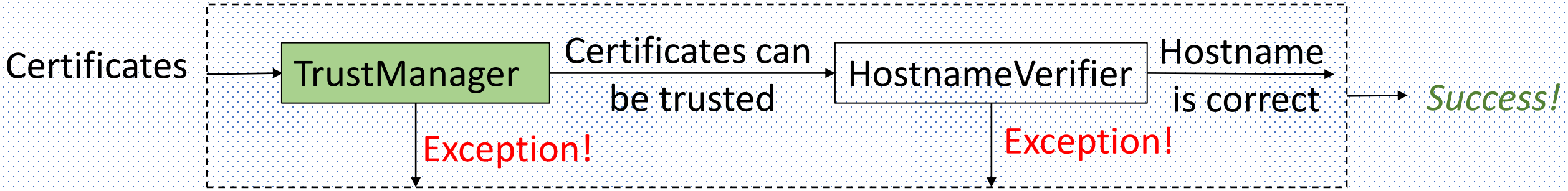


Some Exceptions can be fixed by securely customizing **TrustManager** and **HostnameVerifier**



Caution: Customization needs to be done carefully!

Several examples of customized TrustManager



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 }
```

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).

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 }
```

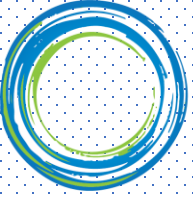
Customization 3: insecure!

```
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

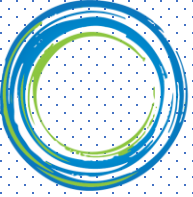
checkValidity only checks whether the certificate is expired

Bypassing the certificate validation under certain condition is insecure!



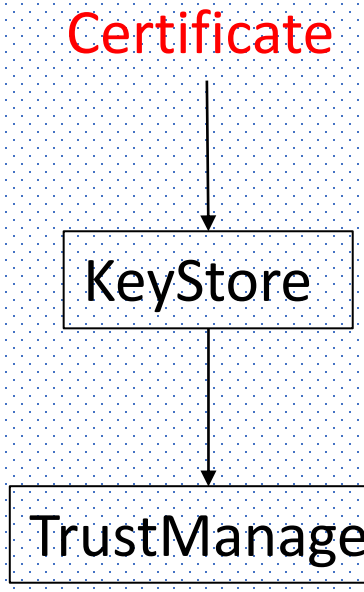
Next, we show several SECURE customized TrustManagers.

We only show the important parts of the code.
Full examples can be found in
(https://github.com/AthenaXiao/SecureTLSCodeExample)



Scenario 1: The client wants to visit the internal server (www.our.example.com) with the self-signed certificate (or certificate signed by a unknown CA (certificate authority)).

Secure Customization 1: specified trust manager



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.

Secure Customization 1: specified trust manager

```
1  // load the new certificate from an InputStream
2  CertificateFactory cf = CertificateFactory.getInstance("X.509");
3  InputStream caInput =
4  new BufferedInputStream(new FileInputStream("special_trust.crt"));
5  Certificate ca = cf.generateCertificate(caInput);
6  // create a KeyStore containing the trusted Certificates
7  KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
8  FileInputStream myKeyStore = new FileInputStream("mykeystore.jks");
9  keyStore.load(myKeyStore, null);
10 keyStore.setCertificateEntry("ca", ca);
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()
```

Scenario 2: The client wants to visit both the internal server (www.our.example.com) and external servers as normal.

The client has two KeyStores:

1. The system default keyStore in
   `${java.home}/lib/security/cacerts`
   This keystore is pre-populated with many well-known root CAs.
2. `A specified one as scenario 1.`

Secure Customization 2: backup trust manager

```
1  public class SecDevTM implements X509TrustManager {
2      // a default trust manager
3      private X509TrustManager defaultTM;
4      // a trust manager for special requirements
5      private X509TrustManager backupTM;
6      ...
7      @Override
8      public void checkServerTrusted(X509Certificate[] chain, String authType)
9      throws CertificateException {
10         try{
11             defaultTM.checkServerTrusted(chain, authType);
12         }catch (CertificateException e){
13             backupTM.checkServerTrusted(chain,authType)
14         }
15     }
16 }
```

First, delegate to the default trust manager. If it cannot handle it, try the backup trust manager.

Scenario 3: Sometimes, the system may manage multiple key stores

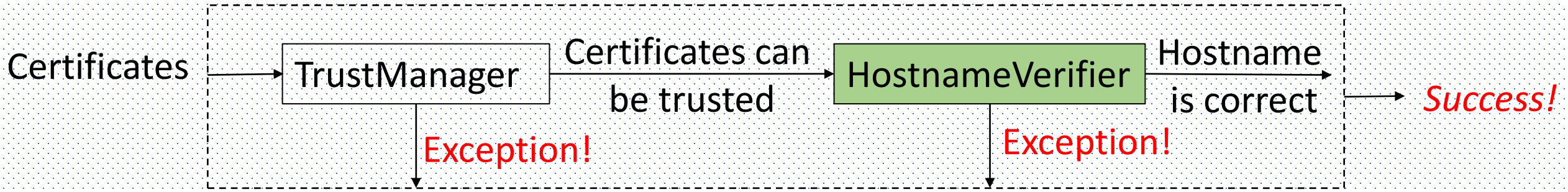
Secure Customization 3: composite trust manager

```
1  public class SecDevTM implements X509TrustManager {
2      // a list of trust managers supporting multiple key stores
3      private final List<X509TrustManager> trustManagers;
4      ...
5      @Override
6      public void checkServerTrusted(X509Certificate[] chain, String authType)
7          throws CertificateException {
8          for (X509TrustManager trustManager : trustManagers) {
9              try {
10                 trustManager.checkServerTrusted(chain, authType);
11                 return; // someone trusts them. success!
12             } catch (CertificateException e) { }
13         }
14         throw new CertificateException(
15             "None of the TrustManagers trust this certificate chain");
16     }
17 }
```

Composite trust manager from multiple trust sources (KeyStore)

Pass the validation if any trust manager trust it.

Several examples of customized HostnameVerifiers



Customization 1: Secure or insecure?

```
1  //custom a hostname verifier
2  HostnameVerifier hostnameVerifier = new HostnameVerifier() {
3      @Override
4      public boolean verify(String hostname, SSLSession session) {
5          return True;
6      }
7  };
```

Customization 1: Insecure!

```
1  //custom a hostname verifier
2  HostnameVerifier hostnameVerifier = new HostnameVerifier() {
3      @Override
4      public boolean verify(String hostname, SSLSession session) {
5          return True;
6      }
7  };
```

Allowing all hostnames is insecure!

Customization 2: Secure or insecure?

```
1  //custom a hostname verifier
2  HostnameVerifier hostnameVerifier = new HostnameVerifier() {
3      @Override
4      public boolean verify(String hostname, SSLSession session) {
5          HostnameVerifier hv =
6              HttpsURLConnection.getDefaultHostnameVerifier();
7          return hv.verify("our.example.com", session);
8      }
9  };
10
11 // tell the URLConnection to use our HostnameVerifier
12 URL url = new URL("https://our.example.org/");
13 HttpsURLConnection conn =
14     (HttpsURLConnection)url.openConnection();
15 conn.setHostnameVerifier(hostnameVerifier);
16 InputStream in = conn.getInputStream();
17 copyInputStreamToOutputStream(in, System.out);
```

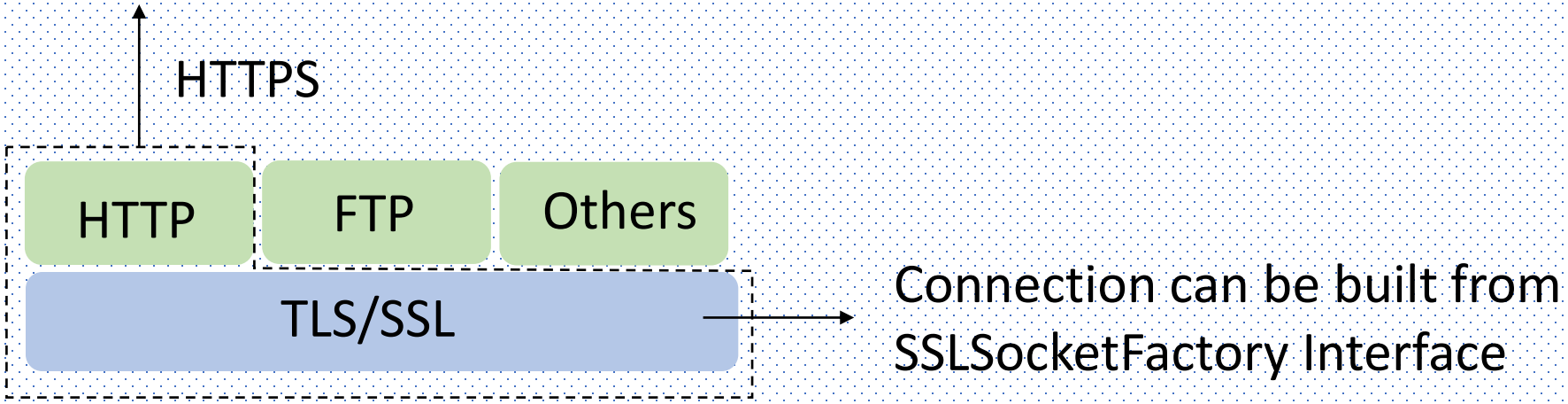
Customization 2: Secure!

```
1  //custom a hostname verifier
2  HostnameVerifier hostnameVerifier = new HostnameVerifier() {
3      @Override
4      public boolean verify(String hostname, SSLSession session) {
5          HostnameVerifier hv =
6              HttpsURLConnection.getDefaultHostnameVerifier();
7          return hv.verify("our.example.com", session);
8      }
9  };
10
11 // tell the URLConnection to use our HostnameVerifier
12 URL url = new URL("https://our.example.org/");
13 HttpsURLConnection conn =
14     (HttpsURLConnection)url.openConnection();
15 conn.setHostnameVerifier(hostnameVerifier);
16 InputStream in = conn.getInputStream();
17 copyInputStreamToOutputStream(in, System.out);
```

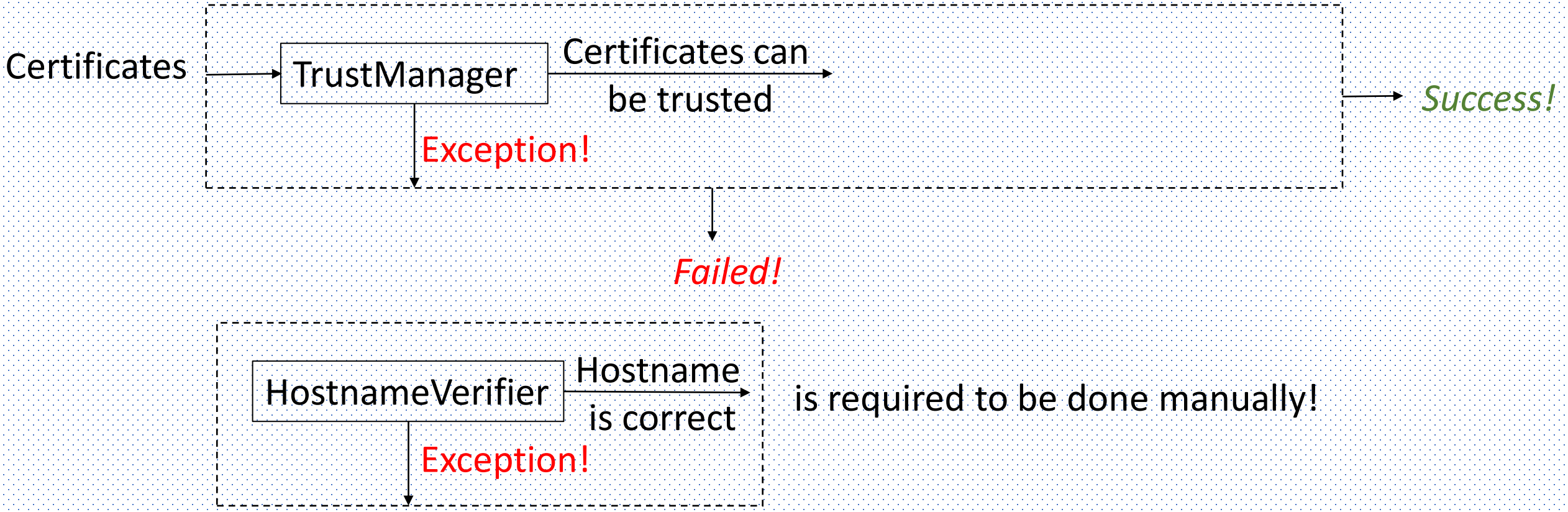
Specify the expected hostname or define specific verification logic is secure!

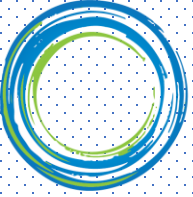
TLS/SSL connection built by SSLSocketFactory

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



The implicit authentication does not include Hostname Verification!





Several examples about usage of SSLSocketFactory

Example 1: Secure or insecure?

```
1  // create a SSLSocket
2  SSLSocketFactory sf = (SSLSocketFactory) SSLSocketFactory.getDefault();
3  SSLSocket socket = (SSLSocket) sf.createSocket("our.example.com", 443);
4
5  // ... use socket ...
6
7  // communication ends
8  socket.close();
```

Example 1: Insecure!

Hostname verification is required to perform manually!

```
1  // create a SSLSocket
2  SSLSocketFactory sf = (SSLSocketFactory) SSLSocketFactory.getDefault();
3  SSLSocket socket = (SSLSocket) sf.createSocket("our.example.com", 443);
4
5  // ... use socket ...
6
7  // communication ends
8  socket.close();
```

Handshaking implicitly happens when data is flushed. However, no hostname verification happens!

Connection with raw SSLSocketFactory Secure!

```
1  // create a SSLSocket
2  SSLSocketFactory sf = (SSLSocketFactory) SSLSocketFactory.getDefault();
3  SSLSocket socket = (SSLSocket) sf.createSocket("our.example.com", 443);
4  //verify the hostname manually
5  HostnameVerifier hv = HttpsURLConnection.getDefaultHostnameVerifier();
6  if (!hv.verify(socket.getSession().getPeerHost(), socket.getSession())) {
7      throw new SSLHandshakeException("Hostname does not match!");
8  }
9
10 // ... use socket ...
11
12 // communication ends
13 socket.close();
```

Manually calling the HostnameVerifier.verity() ensures the secure communication.

Connection with raw SSLSocketFactory Secure!

```
1  // create a SSLSocket
2  SSLSocketFactory sf = (SSLSocketFactory) SSLSocketFactory.getDefault();
3  SSLSocket socket = (SSLSocket) sf.createSocket("our.example.com", 443);
4
5  SSLParameters sslParams = new SSLParameters();
6  sslParams.setEndpointIdentificationAlgorithm("HTTPS");
7  socket.setSSLParameters(sslParams);
8  // ... use socket ...
9
10 // communication ends
11 socket.close();
```

When the algorithm field is "HTTPS", the handshaking automatically performs hostname verification.

Setting the algorithm filed as "HTTPS" is another way to secure the communication.

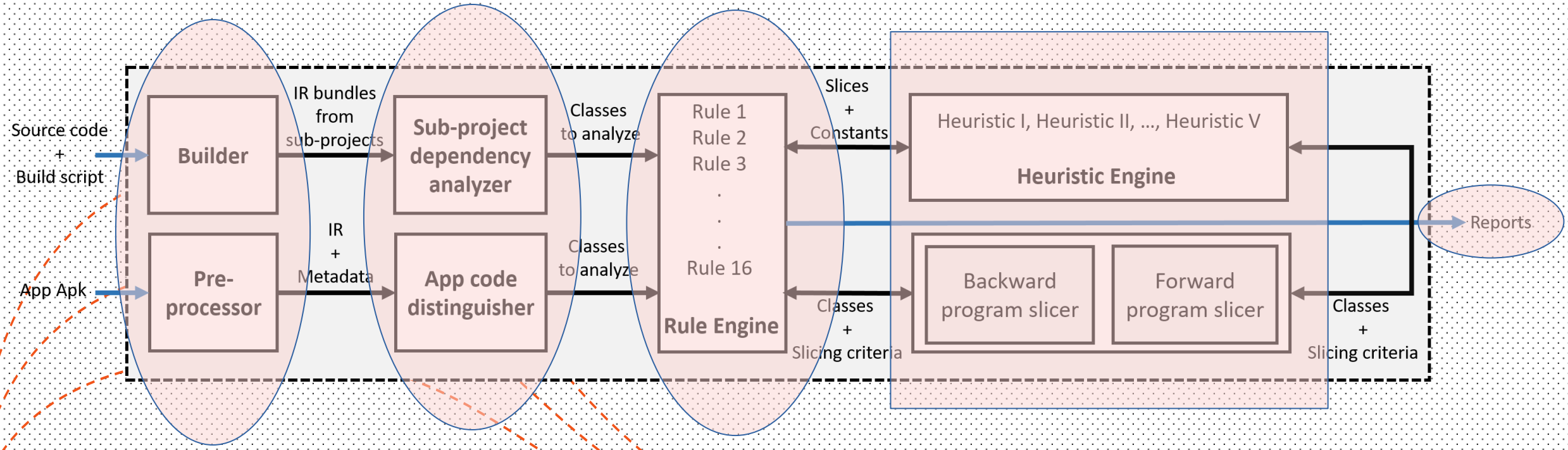
CryptoGuard Design

Presenter:
Sazzadur Rahaman

IEEE SecDev 2020

Cryptographic Misuse Detection with CryptoGuard

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



UNIVERSITY LIBRARIES
VIRGINIA TECH

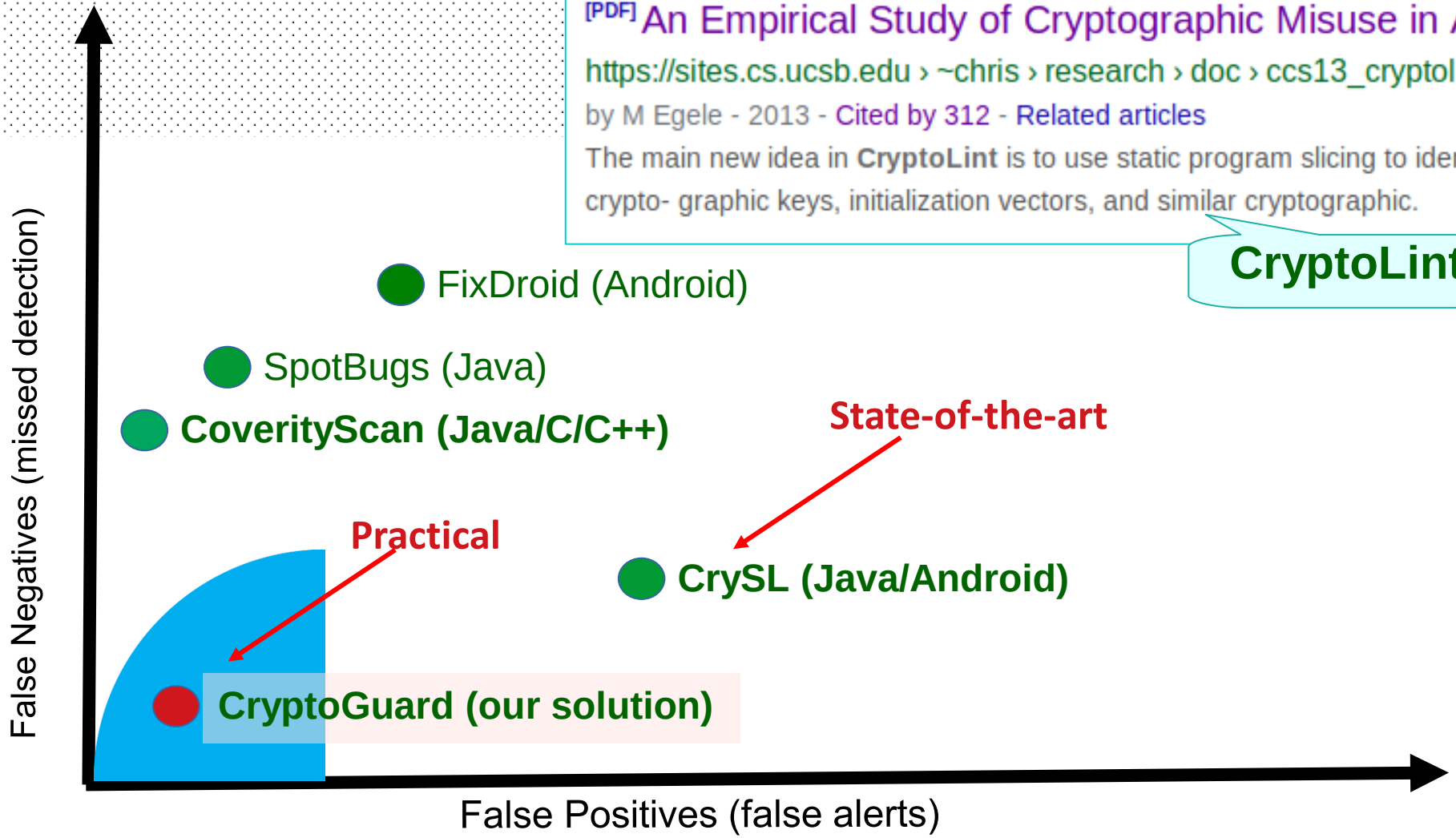
Precise cryptographic misuse detection is hard …

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

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

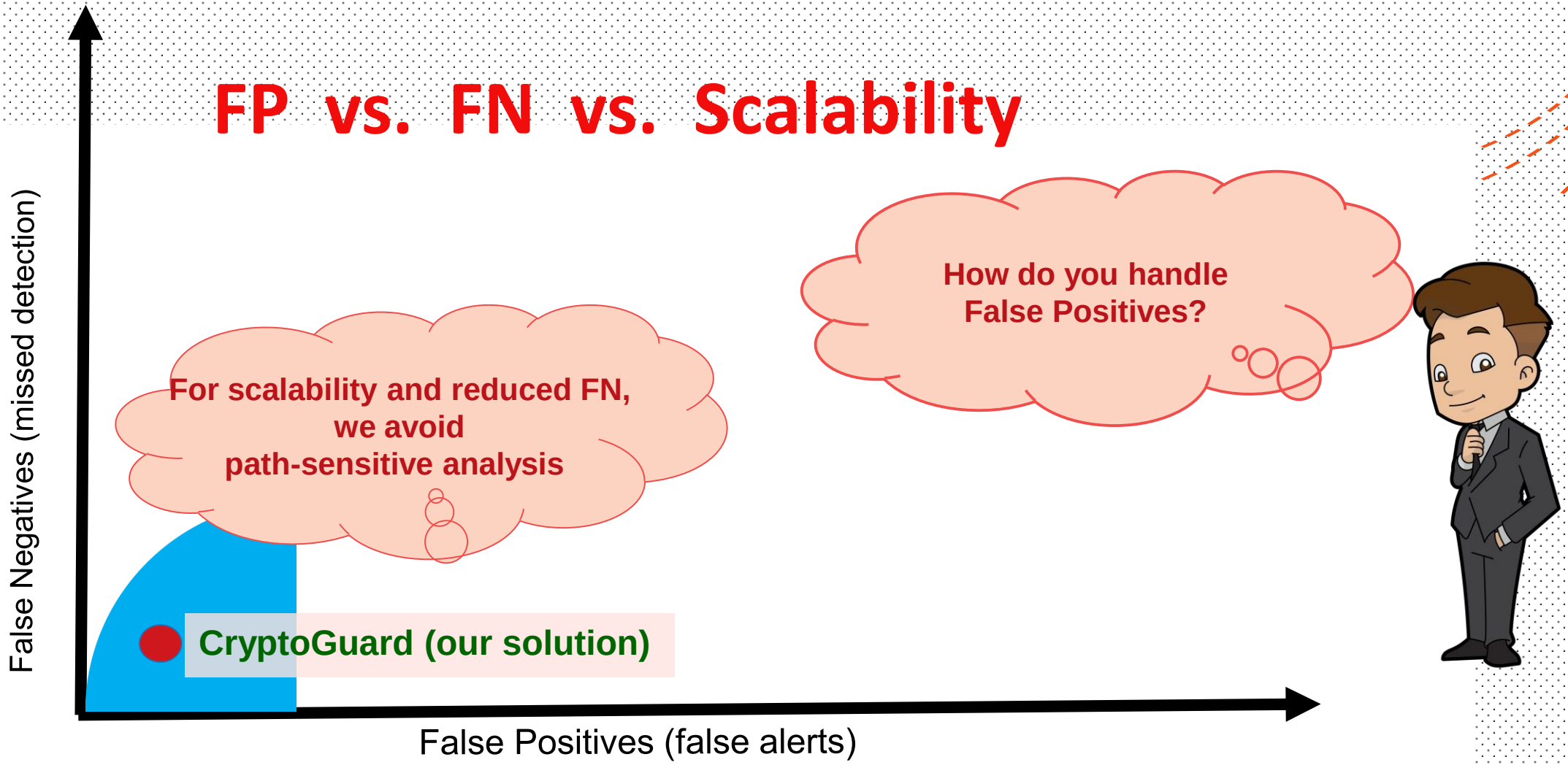
by M Egele - 2013 - Cited by 312 - Related articles

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



Goal and Challenges

FP vs. FN vs. Scalability



Sources of false positives ...

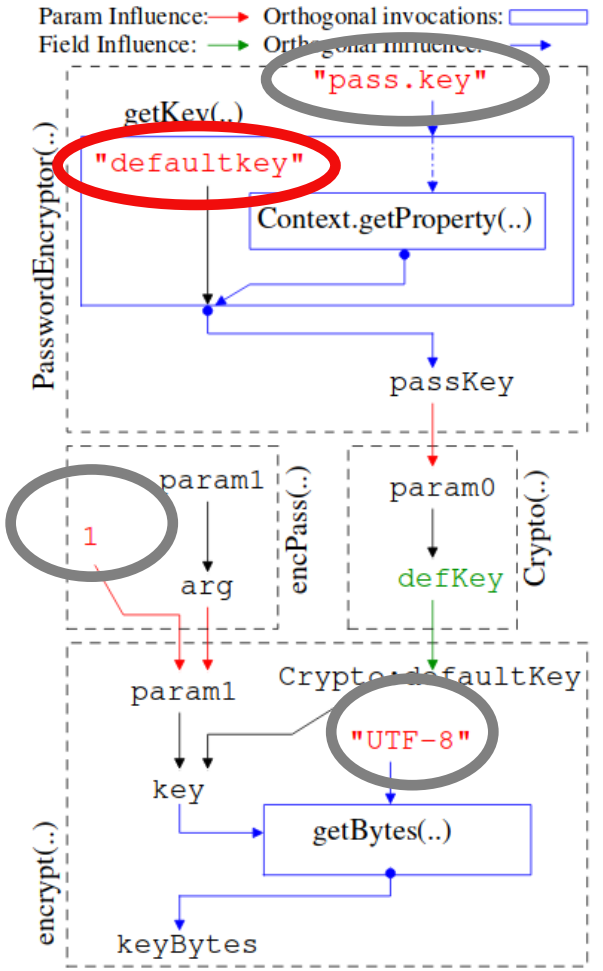
```
1 class PasswordEncryptor {
2
3  Crypto crypto;
4
5  public PasswordEncryptor(){
6   String passKey = PasswordEncryptor
                       .getKey("pass.key")
7    crypto = new Crypto(passKey); (p)
8  }
9
10 byte[] encPass(String [] arg){
11  return crypto.encrypt(arg[0], arg[1]); (p)
12 }
13
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

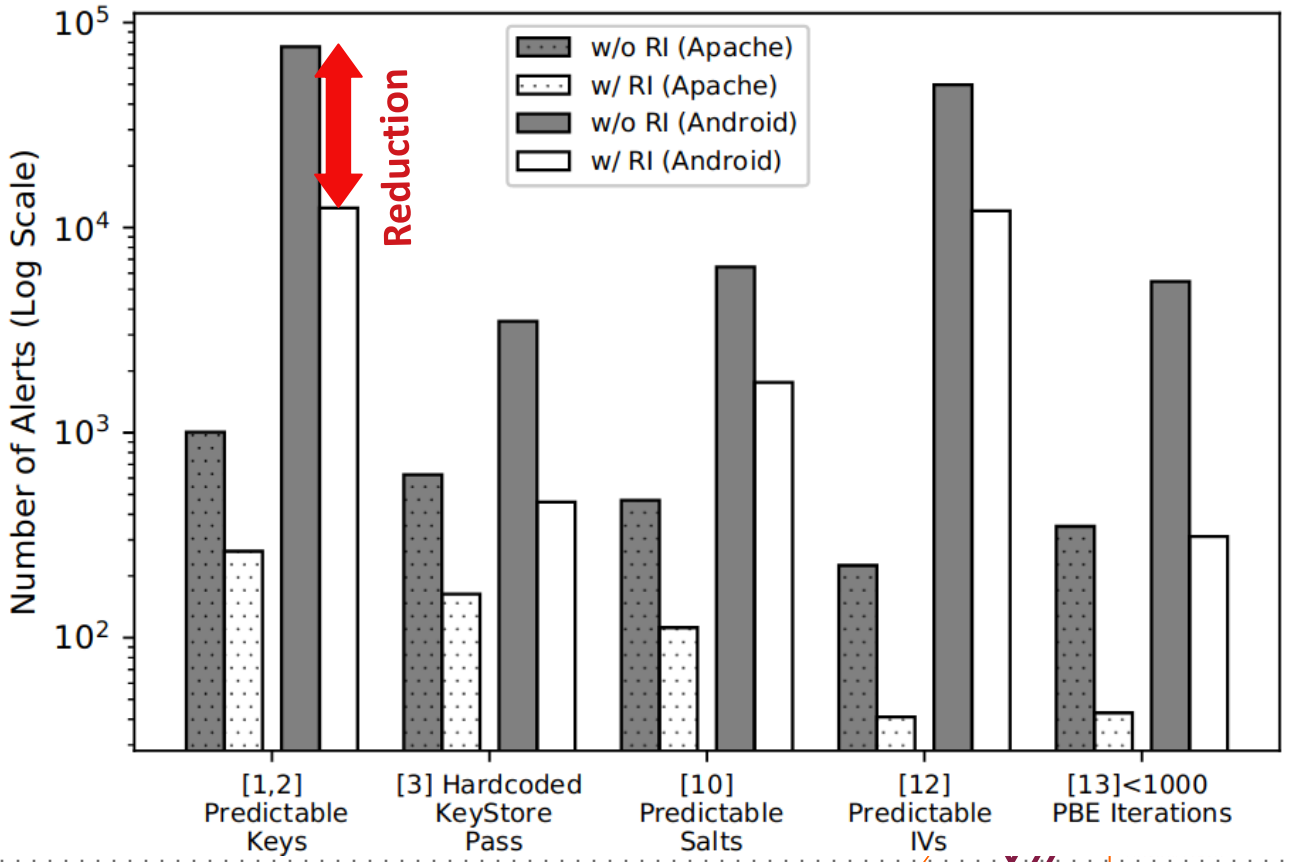
RI I: Removal of state indicators
RI II: Removal of resource identifiers
RI III: Removal of bookkeeping indices
RI IV: Removal of contextually incompatible constants
RI V: Removal of constants in infeasible paths

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 %**

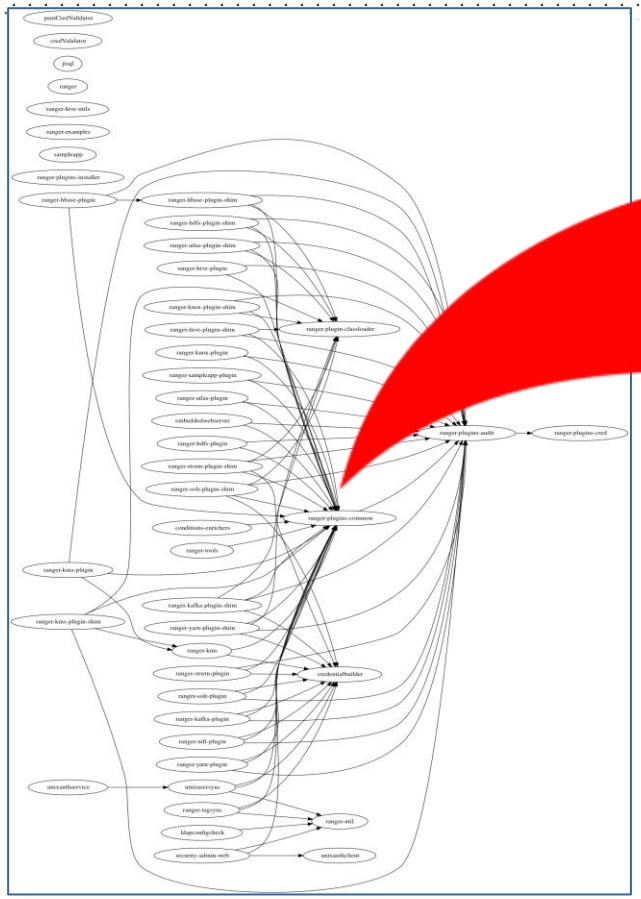
Manual analysis confirmed 18 false alerts ...

Only 1.39% false positives!

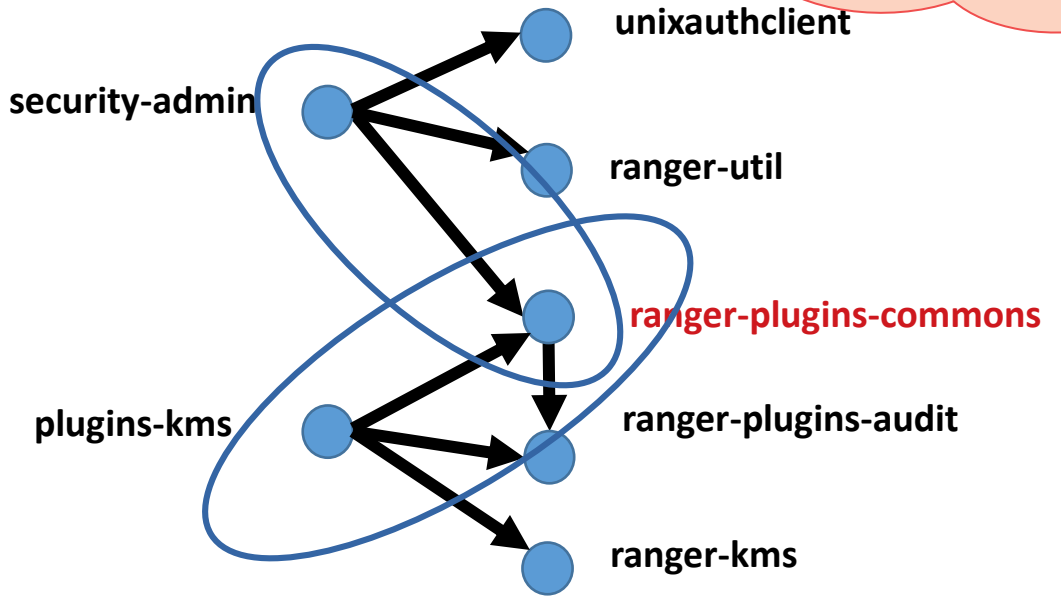
Challenge II: How to Achieve Scalability?

Maximum LoC: 2,571K (Hadoop); Average LoC: 402K

Insight: large code bases are modularized in sub-projects!



Subproject Dependency Graph (Apache Ranger)



Root-subprojects can be analyzed in parallel!

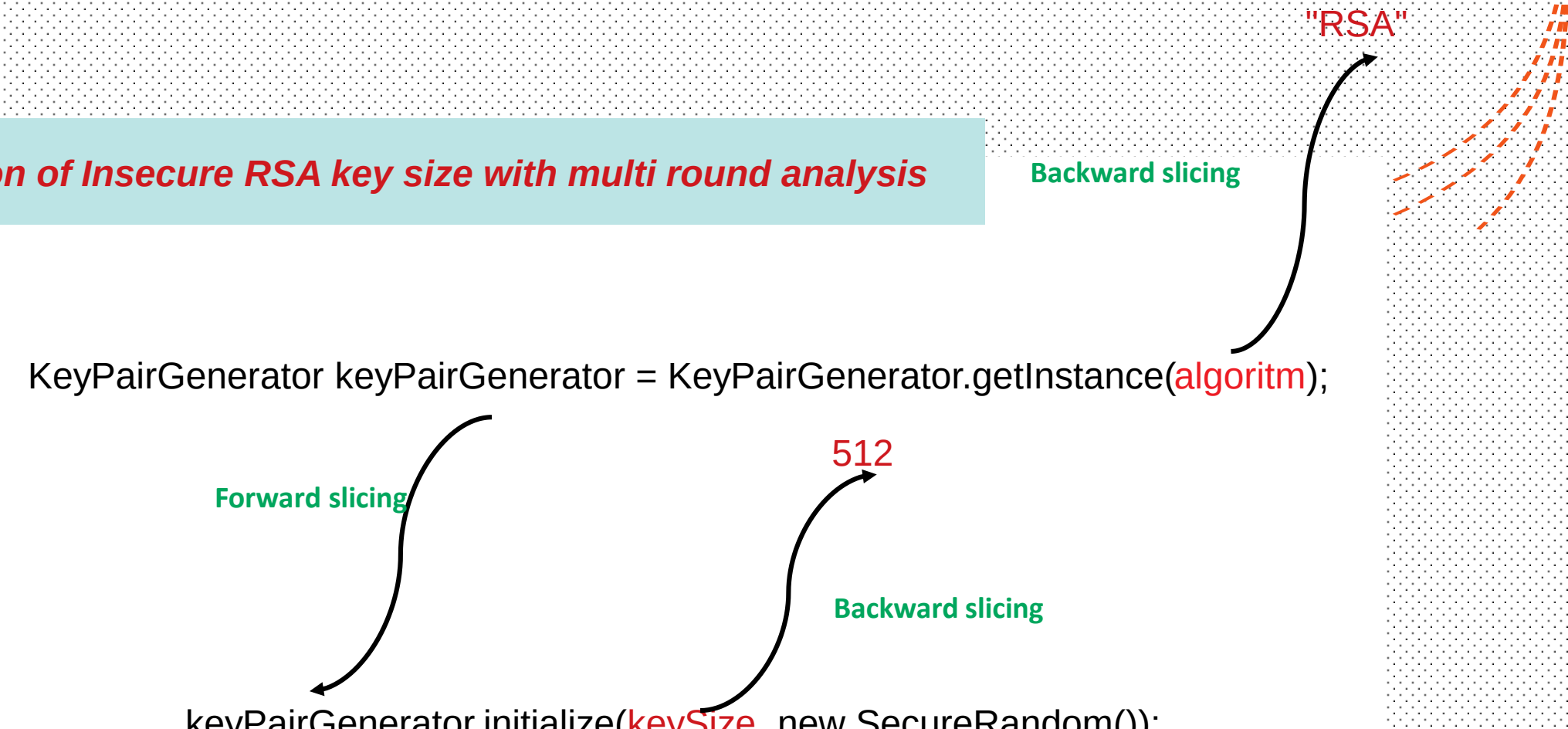
Other Features: CryptoGuard uses forward slicing for some rules (Insecure SSLSocket)

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



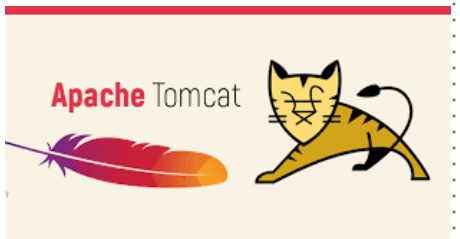
This is possible because of the lightweightness of the algorithms!

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

We discovered misuses in Apache top-tier projects!

Apache Ranger

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

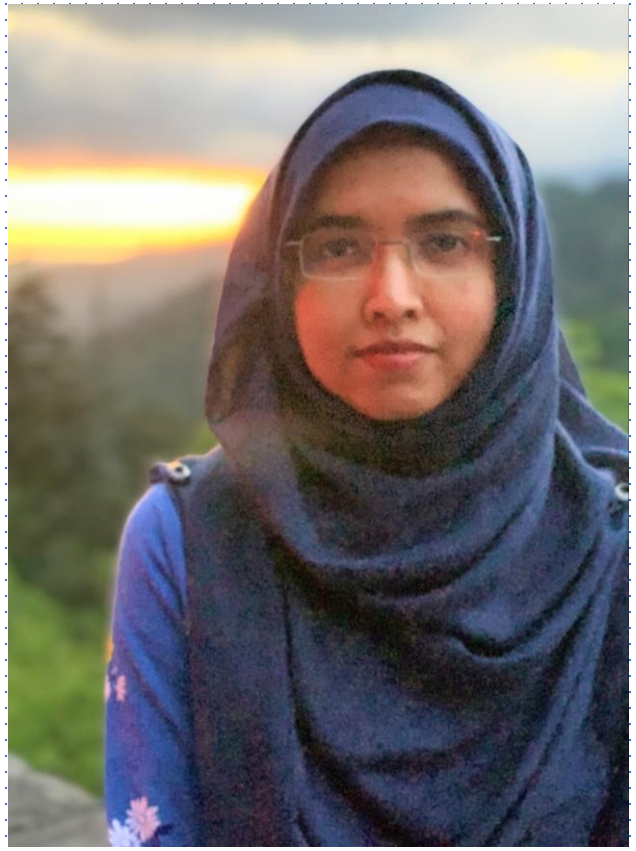
Package name	Violated Rules
com.google.api	3, **4, 5**, 7
com.umeng.anlytics	7, 9, 12, 16
com.facebook.ads	**5**, 9, 16
org.apache.commons	**5**, 9 , 16
com.tencent.open	2, 7, 9

	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

CryptoAPI-Bench Benchmark

Presenter:
Sharmin Afrose



CryptoAPI-Bench Benchmark

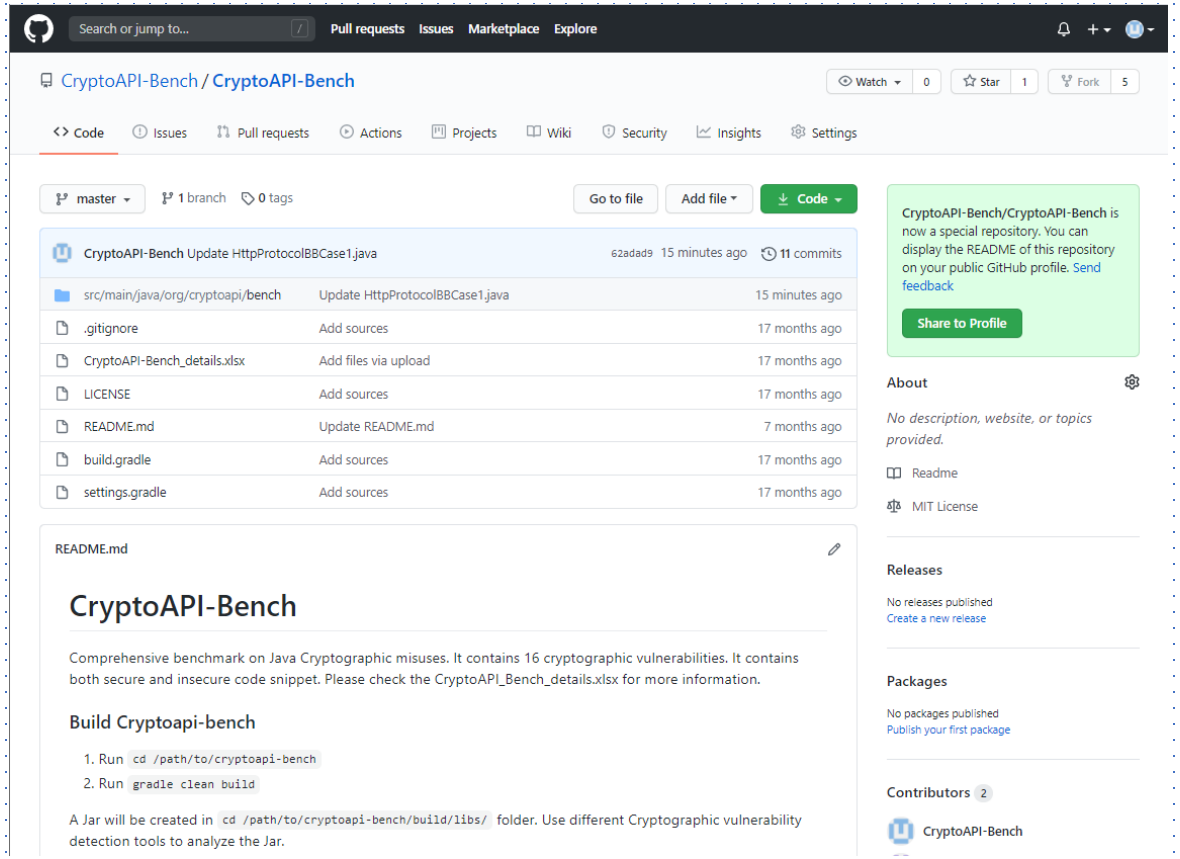
- Benchmark based on Java cryptographic API misuses
- Contains 171 unit test cases of 16 Rules

Improve tool's performance

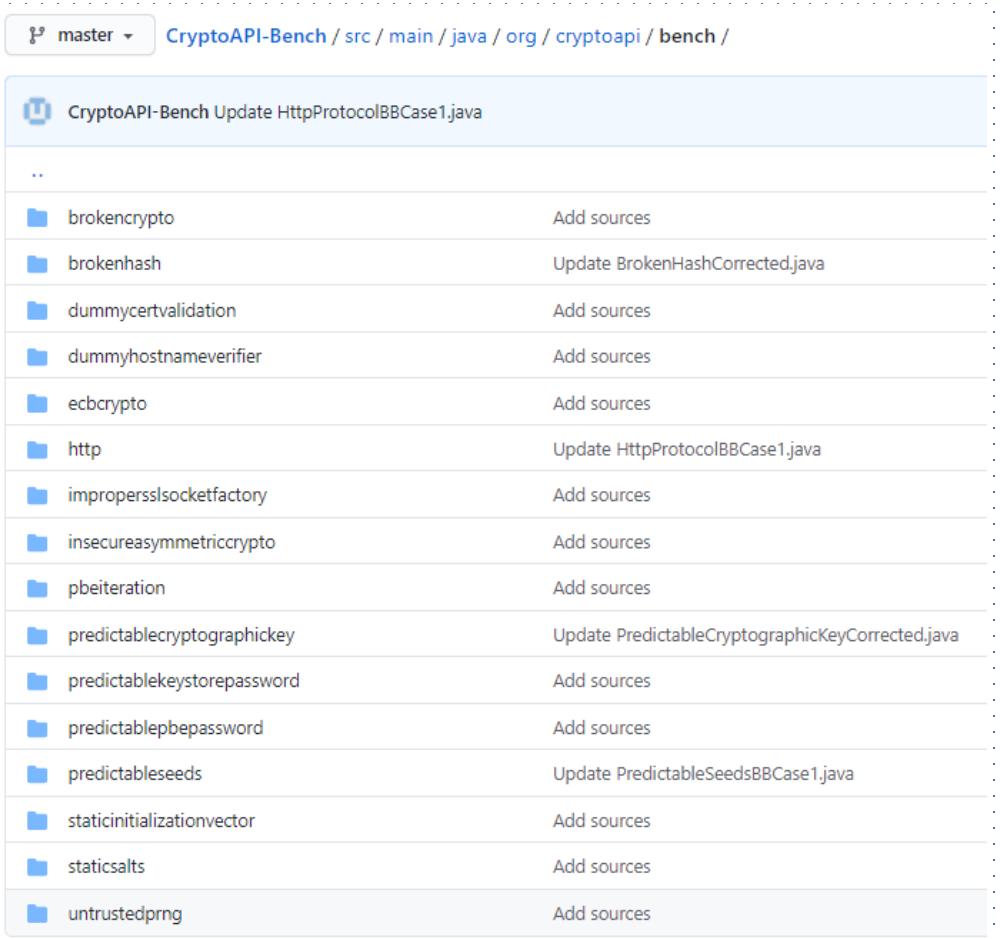
Compare different tools relative performance

Educate secure code VS insecure code

CryptoAPI-Bench: Open-Sourced



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



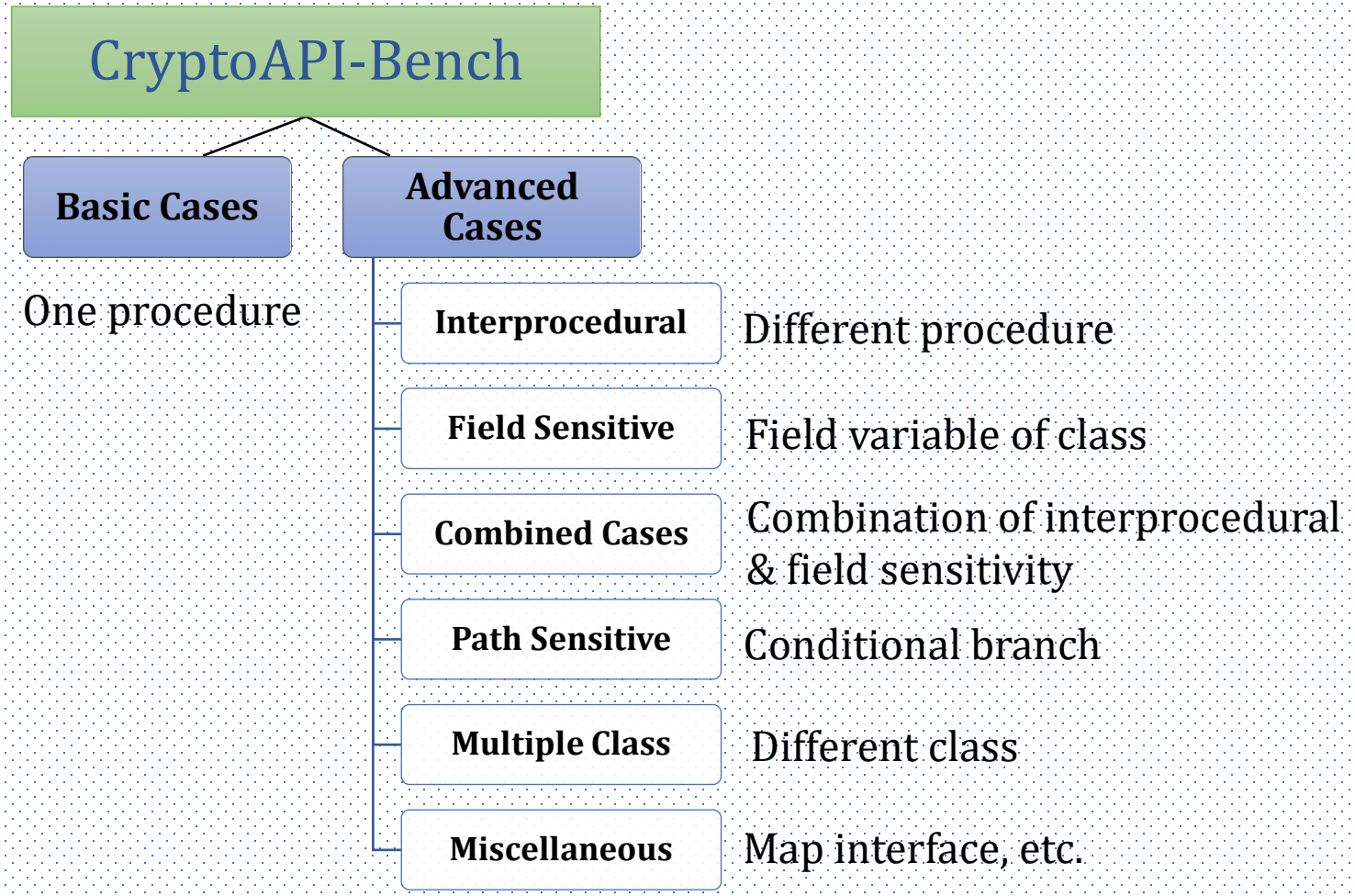
CryptoAPI-Bench: Navigation

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

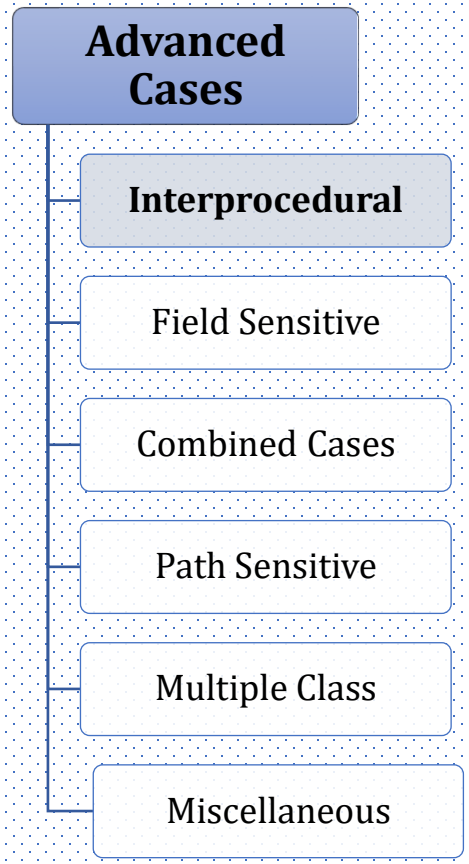
CryptoAPI-Bench: Structure



CryptoAPI-Bench: Interprocedural Example

```
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 }
```

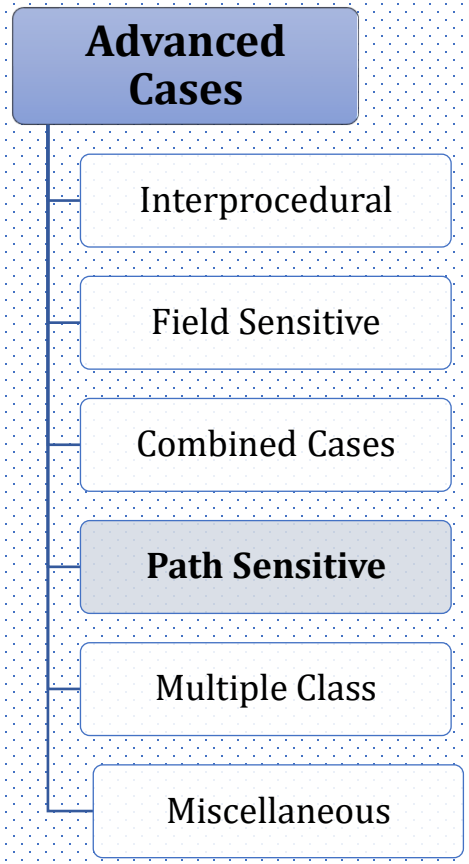
Iteration count value passed to another procedure



CryptoAPI-Bench: Path Sensitive Example

```
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



CryptoAPI-Bench: Tool Evaluation[1]

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**	**55.56**	**52.00**

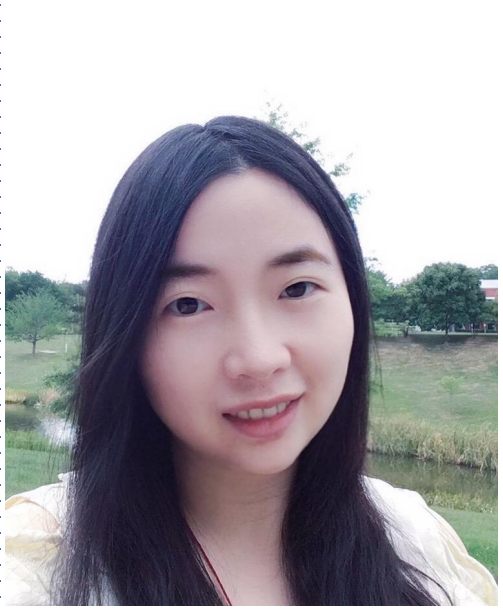
None designed to handle path sensitive cases

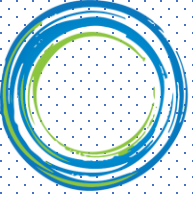
Sharmin Afrose, Sazzadur Rahaman, and Danfeng Yao. "CryptoAPI-Bench: A Comprehensive Benchmark on Java Cryptographic API Misuses." 2019 IEEE Cybersecurity Development (SecDev). IEEE, 2019

[1]Cryptoduard: Commit id c046892 ; CrySL: Commit id 5f531d1 ; SpotBugs: Version 3.1.0 ; Coverity: 29th March 2019

Parfait-CryptoScanner Design/Results

Presenter:
Ya Xiao





IEEE SecDev | 2020

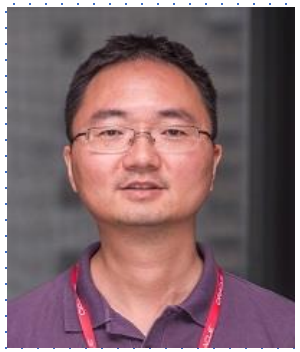
Oracle Labs Australia:

Parfait is a scalable bug used in Oracle.

Parfait-CryptoScanner:

the precise and scalable cryptographic vulnerability detection supported by Parfait.

Cristina Cifuentes



Yang Zhao

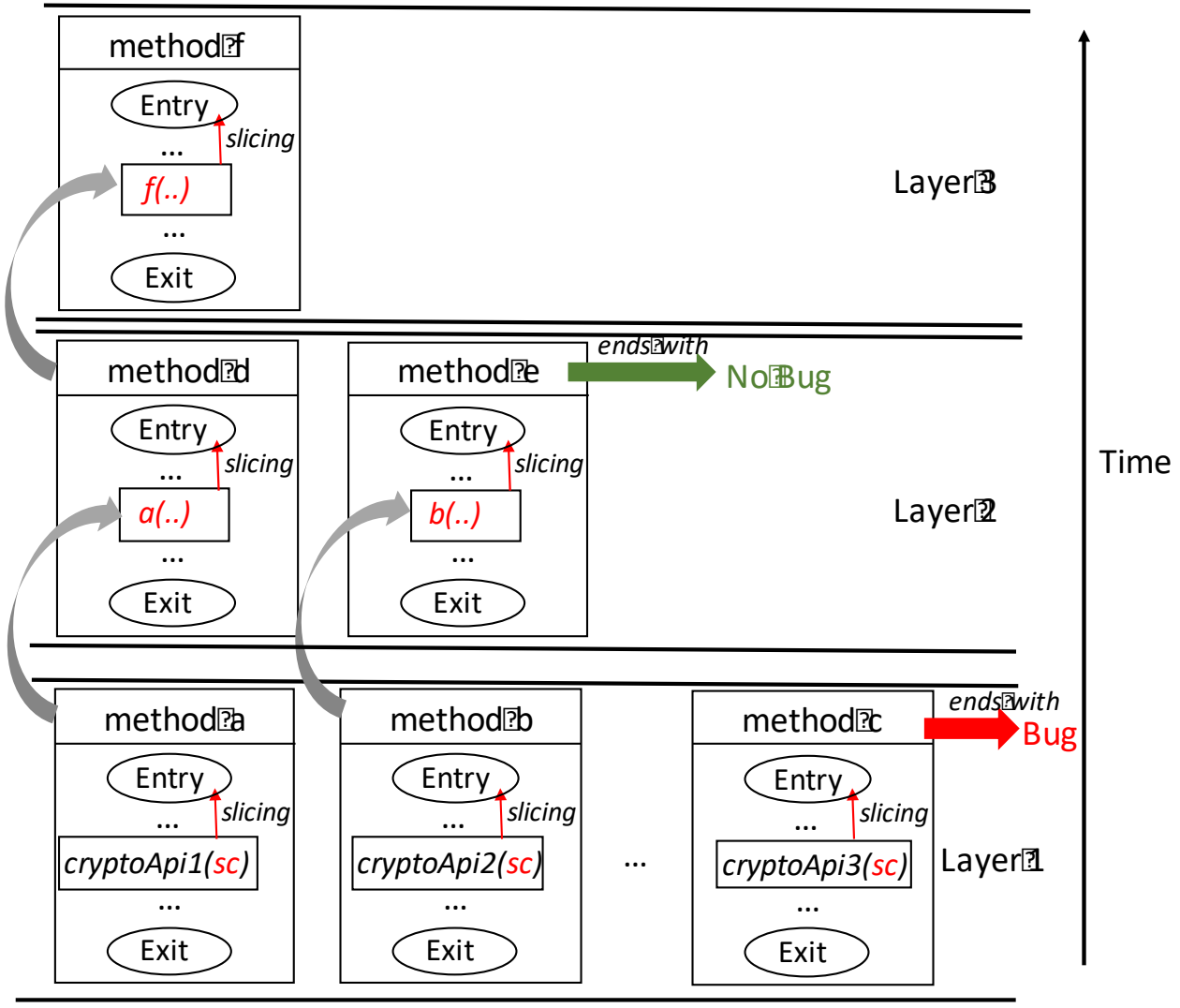
Nicholas Allen

Nathan Keynes

Xiao, Y., Zhao, Y., Allen, N., Keynes, N., & Cifuentes, C. (2020). Industrial Experience of Finding Cryptographic Vulnerabilities in Large-scale Codebases. *arXiv preprint arXiv:2007.06122*.

Scalable Layered Framework in Parfait

- The analyses ensemble is optimized.
- The analyses are scheduled from the quickest to the slowest.
- More vulnerabilities can be found with a lower time overhead.



Experimental Results: CryptoAPI-Bench

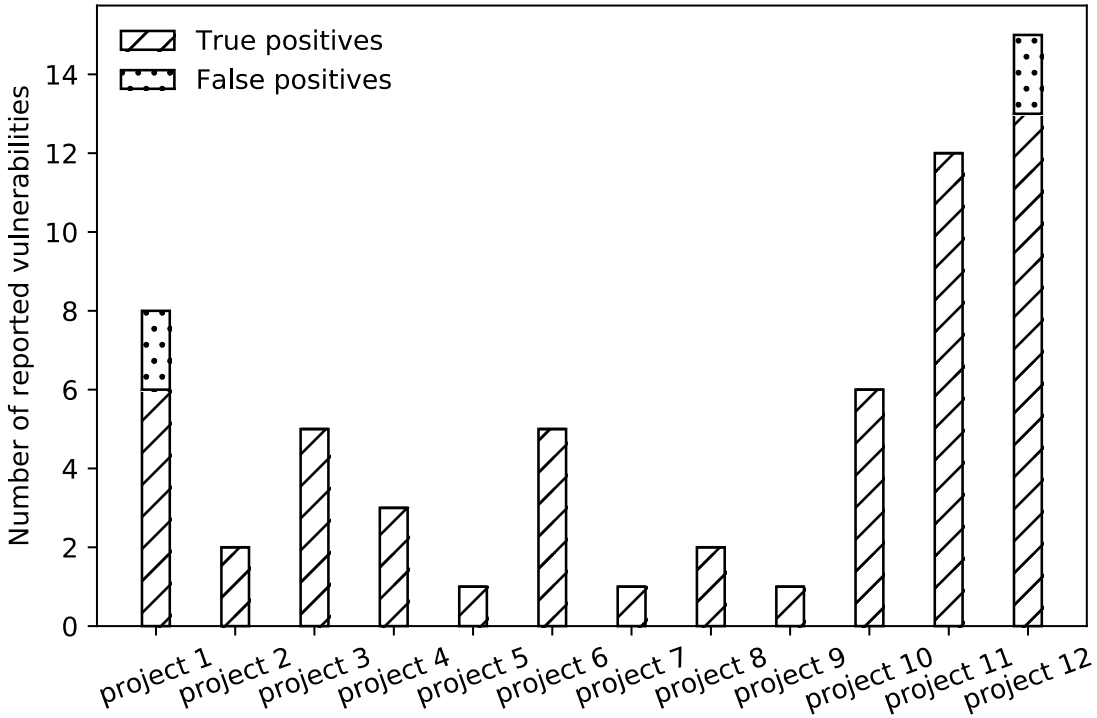
- Excellent Recall (98.4%)
- Perfect Precision (100%) excluding the Path sensitivity cases.

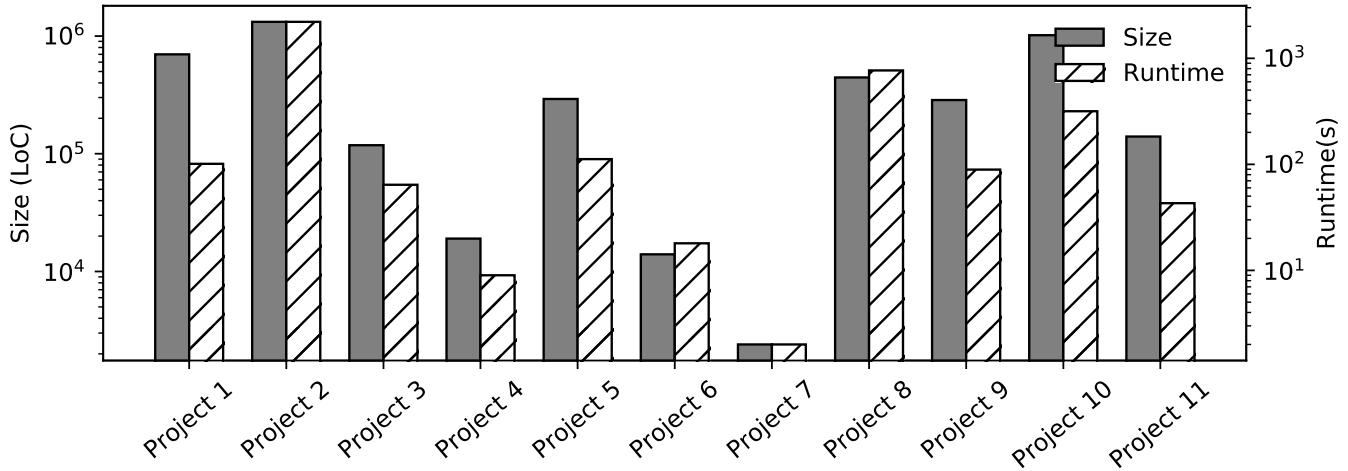
Type	Total Cases	Insecure Cases	Secure Cases	Reported Cases	False Positives	False Negatives	Precision	Recall
Basic Cases	27	24	3	24	0	0	100%	100%
Multiple methods	57	56	1	54	0	2	100%	96.43%
Multiple Classes	23	18	5	18	0	0	100%	100%
Field Sensitivity	19	18	1	18	0	0	100%	100%
Path Sensitivity	19	0	19	19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

Experimental Results: Real world applications

- Excellent Precision (93.44%)
- Good Runtime Performance

  <10 minutes for most of them, even including millions LoC (Project 10)

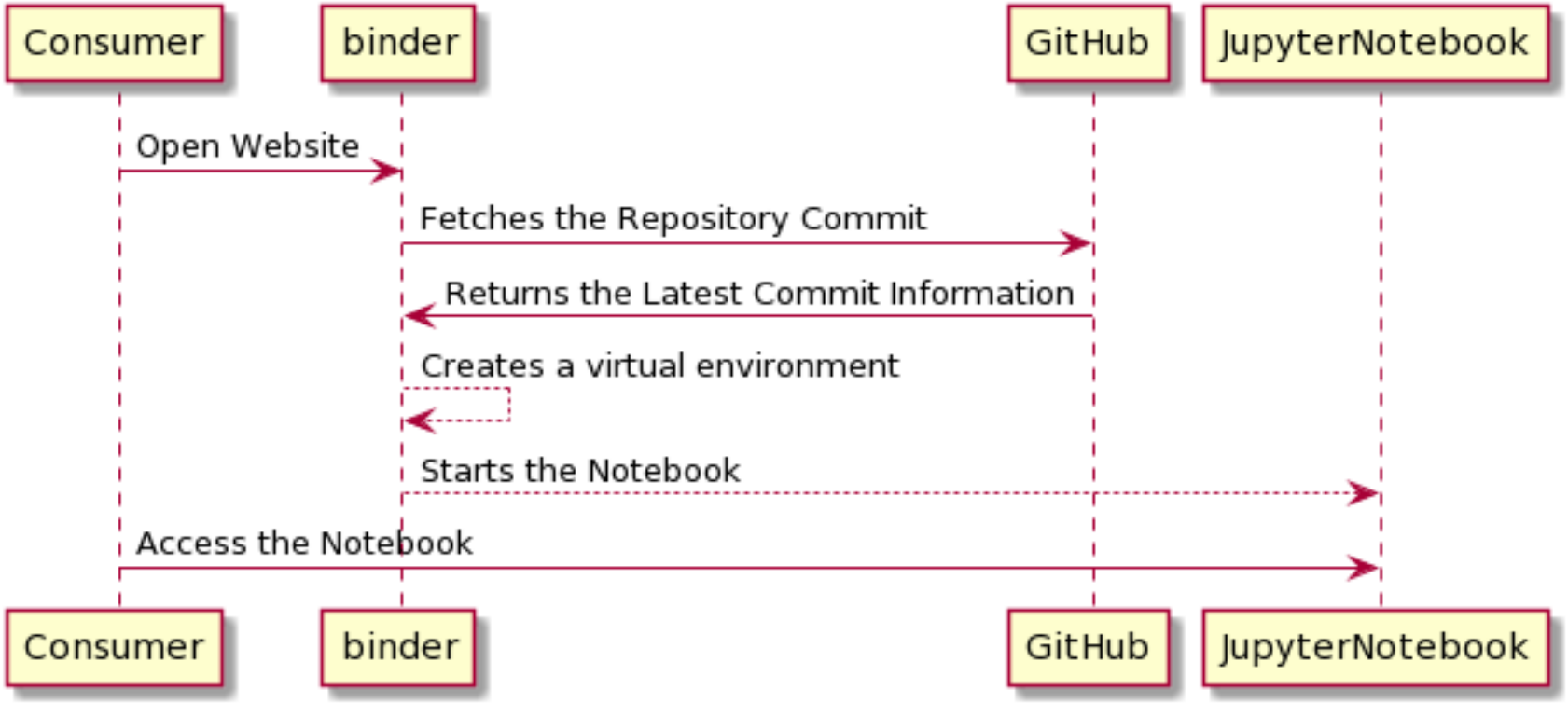




How to use CryptoGuard

Presenter:
Miles Frantz

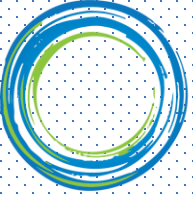
CryptoGuard Setup



https://mybinder.org

https://jupyter.org/

https://github.com/SpencerPark/IJava



IEEE SecDev|2020

CryptoGuard Usage

jupyter SecDev_Tutorial (unsaved changes)

Join this repo's Video Chat | Visit repo | Copy Binder link

File Edit View Insert Cell Kernel Widgets Help

Trusted | Java

Run | Code

SecDev Sampling Tutorial

Testing Imports

In [ ]:

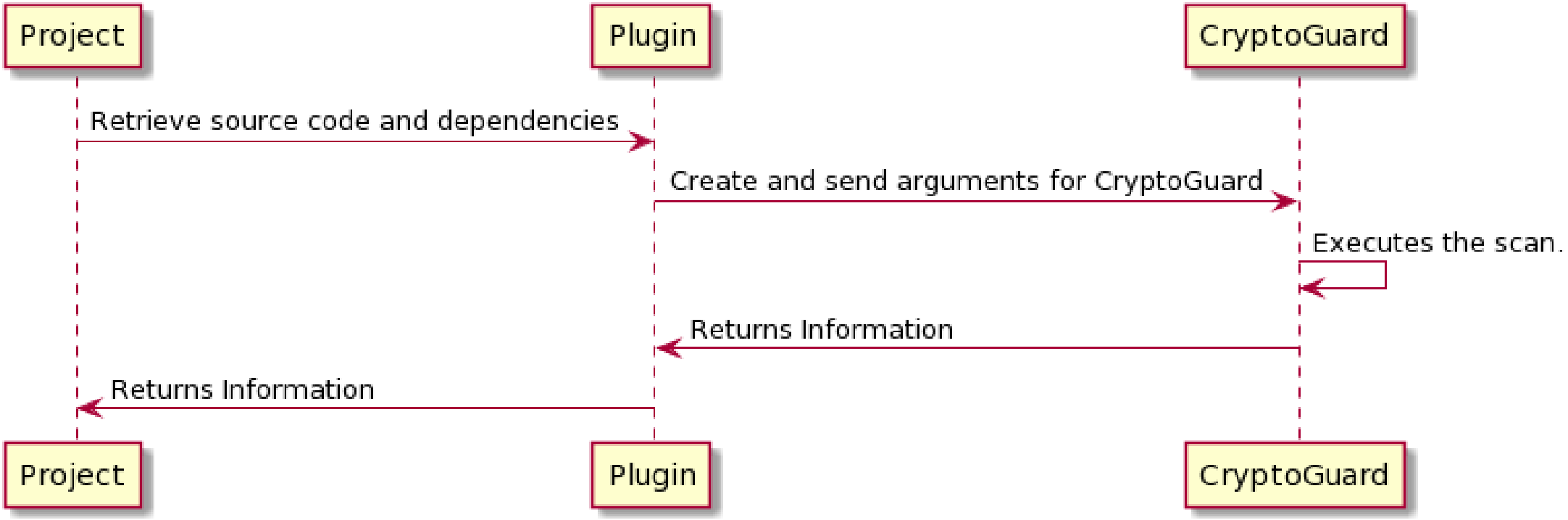
```
// Importing the local CryptoGuard Jar
List <String> addedJars = %jars *.jar
//*/

// Maven Imports
%maven junit:junit:4.12
%maven org.apache.commons:commons-io:1.3.2
%maven commons-io:commons-io:2.7
%maven com.binarytweed:quarantining-test-runner:0.0.3

// Gradle imports
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNull;
import static org.junit.Assert.assertNotNull;
import static util.Utils.makeArg;
import frontEnd.Interface.outputRouting.ExceptionHandler;
import frontEnd.Interface.ArgumentsCheck;
import frontEnd.MessagingSystem.routing.Listing;
```

https://mybinder.org/v2/gh/franceme/cryptoguard/2020_SecDev_Tutorial?filepath=SecDev_Tutorial.ipynb

Hooking CryptoGuard into Build Tools



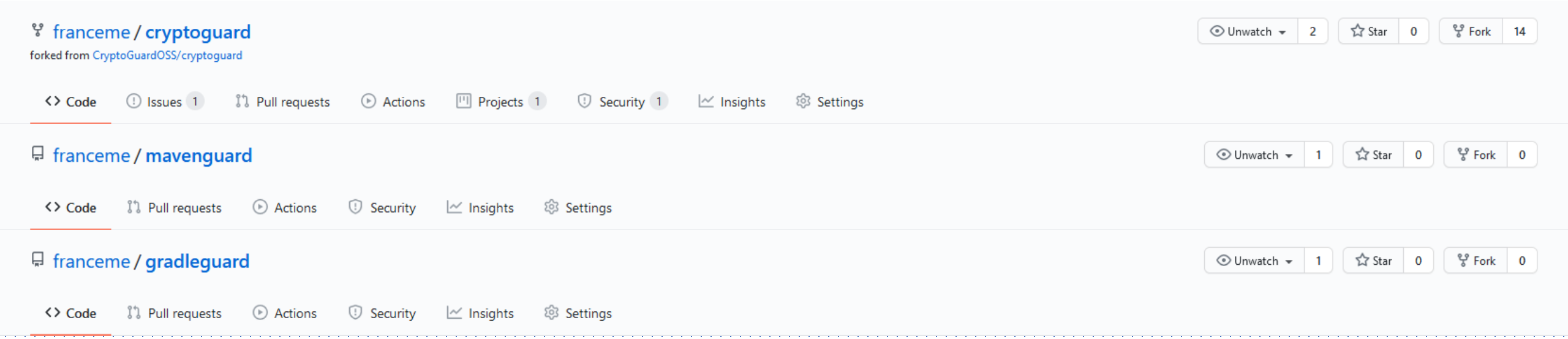
MavenGuard

GradleGuard

MavenGuard and GradleGuard Usage

1) Version: Prints the version of CryptoGuard and plugin
2) previewFiles: Displays the dynamically retrieved files
3) scanFiles: Scans the dynamically retrieved files

Contributing to CryptoGuard and/or Plugins



https://github.com/franceme/cryptoguard

https://github.com/franceme/mavenguard

https://github.com/franceme/gradleguard

Framework configurations are also heavily misuses

Coding Practices and Recommendations of Spring Security for Enterprise Applications

Mazharul Islam*, Sazzadur Rahaman*, Na Meng*, Behnaz Hassanshahi†, Padmanabhan Krishnan†,
Danfeng (Daphne) Yao*
Virginia Tech, Blacksburg, VA*, Oracle Labs, Australia†
{mazharul, sazzad14, nm8247, danfeng}@vt.edu, {behnaz.hassanshahi, paddy.krishnan}@oracle.com

***Abstract*—Spring security is tremendously popular among practitioners for its ease of use to secure enterprise applications. In this paper, we study the application framework misconfiguration vulnerabilities in the light of Spring security, which is relatively understudied in the existing literature. Towards that goal, we identify 6 types of security anti-patterns and 4 insecure vulnerable defaults by conducting a measurement-based approach on 28 Spring applications. Our analysis shows that security risks associated with the identified security anti-patterns and insecure defaults can leave the enterprise application vulnerable to a wide range of high-risk attacks. To prevent these**

designs [14], lack of proper guidelines [15], etc., behind this insecurity. Most of the existing methods to detect security API misuses rely on static code analysis [7], [8], [10], [12], [16]. Although, misconfiguring of security modules in application frameworks has great potential to cause insecurity, their nature, severity, prevalence, and detection feasibility are still mostly unknown.

In this paper, we present a thorough study of framework misconfiguration vulnerabilities in Spring Security. Our goal is to identify various classes of these vulnerabilities (referred

Mazharul Islam

Mazharul will talk about Spring security misconfiguration issues in the main Conference!

Spring security anti-patterns - Examples

```
app:
   auth:
      tokenExpirationMsec: 864000000
      // setting unnecessary long lifetime
```

How does lifelong valid access tokens affect security?

```
spring:
   datasource:
      username: root
      password: uOtmALFgsfxgYzEg1uLXl3O
```

```
@Value("${clientPassword}")
String client_password;
```

Storing password in application.yml

```
public class TokenProvider {
   public String createToken(Authentication auth) {
     Public String JWT_SIGN_KEY = "123456";
     token = Jwts.builder()
           ...
           .signWith(SignatureAlgorithm.HS512, JWT_SIGN_KEY)
           // signing with a hard coded fixed secret
           .compact();
           ...
     return token;
   }
}
```

Hardcoded JWT signing key

```
@Override
protected void configure(HttpSecurity hs) throws Exception {
                hs.csrf.disable()
  )
 }
```

Manually disabling csrf protection

How to send requests using AJAX?

```
.withHttpOnlyFalse())
```

Making the token available to Javascript

Detection with SpanL

We also modeled several new Spring framework anti-patterns in SpanL for automatic detection!

```
NAME: disabling_csrf
APIS: spring_sec_apis
  annotation.web.configuration.WebSecurityConfigurerAdapter:
  <void> configure(<http:annotation.web.builders.HttpSecurity>)

OPERATIONS:
   o1: intra-forward spring_sec_apis with http

EMITS:
  {http}: instructions matches "disable()"

CONSTRAINTS:
  c1: {http} not empty

EXEC:
  o1
  if c1:
     out "Disabled CSRF protection!"
```

Disabled csrf protection in Spring security

```
NAME: const_sym_key
API: jwt_signing_apis
  io.jsonwebtoken.Jwts: <ret:io.jsonwebtoken.Jwts>
    signWith(<algo:java.lang.String>, <key:java.lang.String>)

OPERATIONS:
   o1: inter-backward with jwt_signing_apis and key

EMITS:
  {key}: constants of-type java.lang.String

CONSTRAINTS:
  c1: {key} not empty

EXEC:
  o1
  if c1:
     out "Keys must not be derived from constants"
```

Hardcoded JWT token signing keys

Islam et al. [SecDev'20]

Need much more contributions from the scientific community!

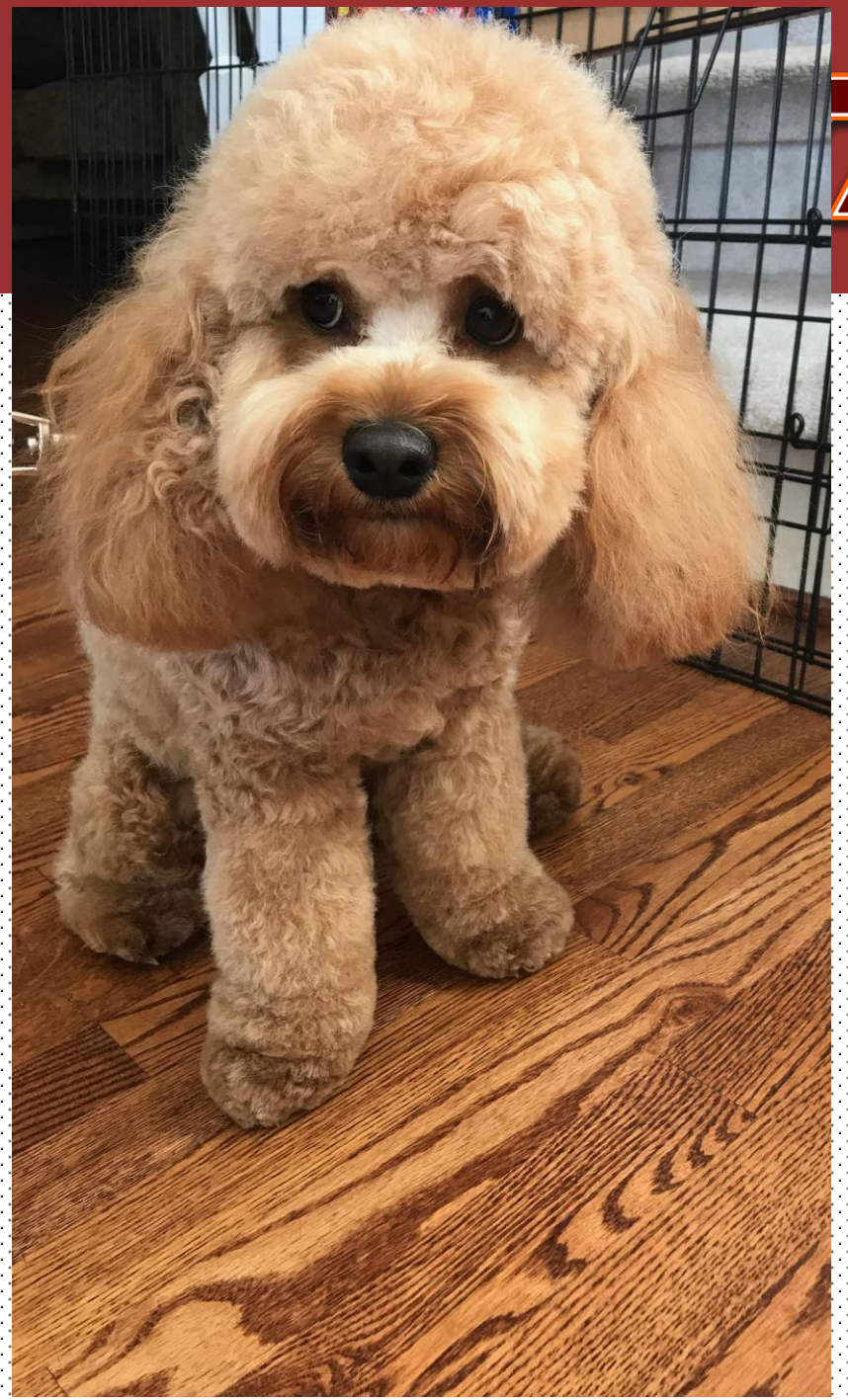
CryptoGuard Related References

Papers:
[1] 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.
[2] 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.
[3] 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).

Online Resources:
[1] CryptoGuard. https://github.com/CryptoGuardOSS/cryptoguard
[2] CryptoAPI-Bench. https://github.com/CryptoGuardOSS/cryptoapi-bench
[3] Secure TLS/SSL code examples. https://github.com/AthenaXiao/SecureTLSCodeExample
[4] https://mybinder.org/v2/gh/franceme/cryptoguard/2020_SecDev_Tutorial

Questions?

yax99@vt.edu
sharminafrose@vt.edu
frantzme@vt.edu
sazz@cs.arizona.edu
danfeng@vt.edu



Thanks!