1. Flexible Invocations – Taxonomy Code Snippets

```
public abstract void
 checkServerTrusted(X509Certificate[]
 x509CertificateArr, String str, Socket socket) throws
 CertificateException;
```

Listing 1: Abstract Method.

```
public void checkServerTrusted(X509Certificate[]
 x509CertificateArr, String str) {
        throw null;
}
--------
public void checkServerTrusted(X509Certificate[]
 x509CertificateArr, String str) {
        throw new AssertionError();
}
```

Listing 2: Throwing Null and Assertion Error.

```
public void checkServerTrusted(X509Certificate[]
 chain, String authType, Socket socket) throws
 CertificateException {
        validateChain(chain, false);
}
```

Listing 3: Method Call For Certificate Validation.

```
public void checkServerTrusted(X509Certificate[]
 x509CertificateArr, String str, Socket socket) {
        this.f39028a.m35993b(x509CertificateArr, str,
         socket);
}
```

Listing 4: Referencing Current Object.

```
public void checkServerTrusted(X509Certificate[]
 ax509certificate, String authType) throws
 CertificateException {
 if (ax509certificate != null) {
  for (X509Certificate x509Certificate :
   ax509certificate) {
   this.issuersList.add(x509Certificate);
 }
```

Listing 5: Method Call For Certificate Validation.

```

public void checkServerTrusted(X509Certificate[]
 x509CertificateArr, String str) throws
 CertificateException {
 if (x509CertificateArr.length != 1) {
  throw new CertificateException("Certificate could
   not be validated (not self-signed)");
        }
```

Listing 6: Cert Length Check and Cert Exception.

```
public void checkServerTrusted(X509Certificate[]
 x509CertificateArr, String str) {
 if (x509CertificateArr == null ||
  x509CertificateArr.length != 1) {
  return;
 }
   x509CertificateArr[0].checkValidity();
}
```

Listing 7: Cert Length, Null Check and Validity Check.

```
public void checkServerTrusted(X509Certificate[]
 x509CertificateArr, String str) {
Log.d("APIGatewayImpl", "checkServerTrusted:" +
 x509CertificateArr);
 for (X509Certificate x509Certificate :
  x509CertificateArr) {
  x509Certificate.checkValidity();
}
```

Listing 8: Logging and Validity Check.

```
public void checkServerTrusted(X509Certificate[]
 chain, String authType) throws CertificateException {
 for (X509Certificate x509Certificate : chain) {
  try {
   x509Certificate.verify(localCertificate.getPublicKey());
```

Listing 9: Verify Method and Throwing Certificate Exception.

```
public final void checkServerTrusted(X509Certificate[]
 chain, String str) {
if (x509CertificateArr != null &&
 x509CertificateArr.length != 0) {
 if (str != null && str.length() != 0) {
  try {
   x509CertificateArr[0].checkValidity();
        return;
```

Listing 10: Validity Check and AuthType Value Check.

```
public void checkServerTrusted(X509Certificate[]
 x509CertificateArr, String str) throws
 CertificateException {
 if (x509CertificateArr != null &&
  x509CertificateArr.length == 1) {
  x509CertificateArr[0].checkValidity();
 } else {
 this.standardTrustManager.checkServerTrusted
  (x509CertificateArr, str);
```

Listing 11: Validity Check and Referencing Current Object.

```
public void checkServerTrusted(X509Certificate[]
 x509CertificateArr, String str) throws
 CertificateException {
 if (this.cache.contains(x509CertificateArr[0])) {
 return;
}
 checkSystemTrust(x509CertificateArr, str);
 checkPinTrust(x509CertificateArr);
 this.cache.add(x509CertificateArr[0]);
}
```

Listing 12: Multiple Method Call, List Method, Referencing Current Object.

```
public void checkServerTrusted(X509Certificate[]
 x509CertificateArr, String str) throws
 CertificateException {
 if (this.inputStream != null) {
  Certificate generateCertificate =
   CertificateFactory.getInstance("X.509")
   .generateCertificate(this.inputStream);
  for (X509Certificate x509Certificate :
   x509CertificateArr) {
   x509Certificate.checkValidity();
    try {
         generateCertificate.verify(x509Certificate
          .getPublicKey());
    return;
 }
 throw new CertificateException("");
}
```

Listing 13: CertificateFactory, Validity Check and Verify

```
public void checkServerTrusted(X509Certificate[]
 x509CertificateArr, String str) throws
 CertificateException {
 for (X509Certificate x509Certificate :
  x509CertificateArr) {
  x509Certificate.checkValidity();
  try {
   x509Certificate.verify(this.f21137a.getPublicKey());
        }
```

Listing 14: Validity Check and Verify

```

public void checkServerTrusted(X509Certificate[]
 x509CertificateArr, String str) throws
 CertificateException {
if (x509CertificateArr == null) {
 throw new
  IllegalArgumentException("checkServerTrusted:
  X509Certificate array is null");
}
if (x509CertificateArr.length <= 0) {
 throw new
  IllegalArgumentException("checkServerTrusted:
  X509Certificate is empty");
}
if (str == null || !str.equalsIgnoreCase("RSA")) {
 throw new CertificateException("checkServerTrusted:
  AuthType is not RSA");
}
try {
 x509CertificateArr[0].checkValidity();
} catch (Exception unused) {
 throw new CertificateException("Server certificate
  not valid or trusted.");
}
}
```

Listing 15: AuthType, Cert Length, Null Check, String Operation and Validity Check.

```
public void checkServerTrusted(X509Certificate[]
 x509CertificateArr, String str) {
X509Certificate x509Certificate = this.expectedCert;
if (x509CertificateArr != null &&
 x509CertificateArr.length > 0) {
 X509Certificate x509Certificate2 =
  x509CertificateArr[0];
  this.lastCheckedCert = x509Certificate2;
   if (this.expectedCert != null) {
        byte[] encoded = x509Certificate2.getEncoded();
        byte[] encoded2 =
         this.expectedCert.getEncoded();
        Log.d(Util.f1167T, "Device presented cert" +
         x509Certificate2.getSubjectDN());
        if (!Arrays.equals(encoded, encoded2)) {
         throw new CertificateException("certificate
          does not match");
        }
        return;
        }
  return;
  }
this.lastCheckedCert = null;
throw new CertificateException("no server
 certificate");
}
```

Listing 16: Logging, getEncoded Method, Array Method, Non-null value check.

```
public void checkServerTrusted(X509Certificate[]
 x509CertificateArr, String str) throws
 CertificateException {
X509Certificate x509Certificate =
 x509CertificateArr[i];
 if (x509Certificate.getSubjectDN() != null &&
  x509Certificate.getSubjectDN().getName() != null) {
 String name =
  x509Certificate.getSubjectDN().getName();
 if (name.contains(".m2mservices.com") ||
  name.contains(".m2mbackup.com")) {
```

Listing 17: String Operation, Using Deprecated Methods

```
public void checkServerTrusted(X509Certificate[]
 certs, String authType) {
int i = 0;
for (X509Certificate x509Certificate : certs) {
 if (x509Certificate.getSubjectDN().toString()
  .contains("EMAILADDRESS=sales-usa@extron.com,
  CN=Quantum Ultra, OU=Engineering,
  O=ExtronElectronics, L=Anaheim, ST=CA, C=US")) {
  i++;
```

Listing 18: Using Deprecated Methods, String Operation

```
public void checkServerTrusted(X509Certificate[]
 x509CertificateArr, String str) {
if(x509CertificateArr != null) {
if(x509CertificateArr.length > 0) {
 if(str != null && str.contains("ECDSA")) {
  CertificateFactory certificateFactory =
   CertificateFactory.getInstance("X.509");
  InputStream openRawResource =
   MCSApplication.m5929a().getResources()
   .openRawResource(R.raw.ca_cert);
  try {
   for(X509Certificate x509Certificate :
    x509CertificateArr) {
    x509Certificate.checkValidity();
    if(x509Certificate.getSubjectDN().getName()
     .contains("ProRAE Guardian Root Certificate
     Authority")) {
     try{
      x509Certificate.verify(certificateFactory
       .generateCertificate(openRawResource)
       .getPublicKey());
```

Listing 19: CertificateFactory, String Operation, Using Deprecated Method and Verify

```
public void checkServerTrusted(X509Certificate[]
 x509CertificateArr, String str2) throws
 CertificateException {
try {
if(AdjustBridgeUtil.byte2HexFormatted(MessageDigest
 .getInstance("SHA1").digest(x509CertificateArr[0]
 .getEncoded()))
 .equalsIgnoreCase("7BCFF44099A35BC093BB48C5A6B9A5
16CDFDA0D1")) {
return;
}
```

Listing 20: SHA1 and getEncoded to check hard coded root cert.