# black hat

DECEMBER 7-8, 2022

BRIEFINGS

# Android parcels: the bad, the good and the better Introducing Android's Safer Parcel

Hao Ke Bernardo Rufino Yang Yang Maria Uretsky

#### About Us



Hao Ke (<u>@haoOnBeat</u>) Security Engineer Android Malware Research Google



Bernardo Rufino Software Engineer Android Platform Security Google Yang Yang Security Engineer Android VRP Google





#### Special thanks





Maria Uretsky Tech Lead Android VRP Google



Kevin Deus ISE Manager Android VRP Google

## Agenda

- Parcel Mismatch problems
- Bundle "FengShui" Self changing Bundle
- Bundle "FengShui" Making it safe(r)
- CVE-2021-0928 (Novel in Android 12-beta)
- CVE-2021-0928 Making it safe(r)
- Parcel Mismatch and Android VRP
- Questions

## Parcel Mismatch problems: Parcel and Parcelable

- Parcel: A container for sending serialized (aka. parceled) data across binder IPCs.
- Parcelable:
  - Sender Side: Objects serialized into the Parcel (writeToParcel)
  - Receiver Side:Reconstructed back into the original Object (createFromParcel)



## Parcel Mismatch problems: Parcelable Containers

- "A final class of methods are for writing and reading standard Java containers of arbitrary types."
- Array, List, ArrayList, Map, SparseArray..

```
public final ArrayList readArrayList(@Nullable
ClassLoader loader) {
    int N = readInt();
    ...
    ArrayList l = new ArrayList(N);
    readListInternal(l, N, loader);
    return l;
```

```
private void readListInternal(@NonNull List
outVal, int N, @Nullable ClassLoader loader) {
        while (N > 0) {
            Object value = readValue(loader);
            outVal.add(value);
            N - - ;
public final Object readValue(@Nullable
ClassLoader loader) {
       int type = readInt();
       switch (type) {
              case VAL_PARCELABLE:
                   return return readParcelable(loader);
              case VAL LIST:
                 return readArrayList(loader);
```

# Parcel Mismatch problems: Parcelable Containers: Cont'd

- Deserializes "everything" in the container
  - Other containers
  - Parcelables and Serializables
  - (Parcelables and Serializables in other containers)
- Of Arbitrary types
  - Deserializes Parcelables or Serializables of any type



## Parcel Mismatch problems

Parcelable Write:

}

```
public void writeToParcel(Parcel parcel, int
flags) {
    parcel.writeInt(f1);
    parcel.writeByteArray(f2);
```

#### Parcelable Read:

```
f1 = parcel.readInt();
if (f1 > 0) {
    parcel.readByteArray(f2);
}
```



## Parcel Mismatch problems: Cont'd

- A parcelable write/read mismatch makes the next entry read be misaligned
- Then, the receiver deserializes the data in an unexpected way





• b "!=" c => Self-changing bundle

- Vulnerable example
- Leveraged in following cross-process flow:
  - A: Sends Bundle x to B
  - A: <x is serialized>
  - B: <x is deserialized>
  - B: Inpects x (TOC) and sends to C
  - B: <x is serialized>
  - C: <x is deserialized>
  - C: Uses x (TOU)
- Challenge: Hide item ("intent" => 42) in Bundle from B
  - Item only appears to C
  - In Android 12







Legend Key Value Undefined



P <sub>A</sub>			P <sub>B</sub>				
A <sub>H</sub>	4B	int	$length(P_A)$	4B	int	length(P <sub>B</sub> )	Вн
	4B	int	magic	4B	int	magic	
	4B	int	3	4B	int	3	
	8B	string	"\u0005"	8B	string	"\u0005"	Bo
^	4B	type	VAL_PARCELABLE	4B	type	VAL_PARCELABLE	
~~ <sub>0</sub>	52B	string	"com.Vulnerable"	52B	string	"com.Vulnerable"	
	4B	int	0	4B	int	0	
	Ļ			4B	string length	Ø (spurious)	
	4B	string length	3	4B	'\0'	3	
	4B	string chars	\u000D\0	4B	type	VAL_BYTEARRAY	D
		string chars	\u0008\0	4B	byte[] length	8	D <sub>1</sub>
	4B	type	VAL_BYTEARRAY	4B	buto[]	[0d,00,00,00]	
A <sub>1</sub>	4B	byte[] length	28	4B	Dyte[]	[1c,00,00,00]	
	28B	byte[] <bytes></bytes>		20B	string	intent	
			<bytes></bytes>	4B	type	VAL_INT	B <sub>2</sub>
			4B	int	42		

	P <sub>A</sub>			P <sub>B</sub>			
A <sub>H</sub>	4B	int	length(P <sub>A</sub> )	4B	int	length(P <sub>B</sub> )	Вн
	4B	int	magic	4B	int	magic	
	4B	int	3	4B	int	3	
	8B	string	"\u0005"	8B	string	"\u0005"	Bo
^	4B	type	VAL_PARCELABLE	4B	type	VAL_PARCELABLE	
~ <sub>0</sub>	52B	string	"com.Vulnerable"	52B	string	"com.Vulnerable"	
	4B	int	0	4B	int	0	
	Ļ			4B	string length	Ø (spurious)	
	4B	string length	3	4B	'\0'	3	
	4B	string chars	\u000D\0	4B	type	VAL_BYTEARRAY	B
		string chars	\u0008\0	4B	byte[] length	8	D <sub>1</sub>
	4B	type	VAL_BYTEARRAY	4B	hute[]	[0d,00,00,00]	
A <sub>1</sub>	4B	byte[] length	28	4B	byte[]	[1c,00,00,00]	
	28B	byte[] <bytes></bytes>		20B	string	intent	
			<bytes></bytes>	4B	type	VAL_INT	B <sub>2</sub>
			4B	int	42		

P <sub>A</sub>			P <sub>B</sub>				
A <sub>H</sub>	4B	int	length(P <sub>A</sub> )	4B	int	length(P <sub>B</sub> )	Вн
	4B	int	magic	4B	int	magic	
	4B	int	3	4B	int	3	
	8B	string	"\u0005"	8B	string	"\u0005"	Bo
^	4B	type	VAL_PARCELABLE	4B	type	VAL_PARCELABLE	
~~0	52B	string	"com.Vulnerable"	52B	string	"com.Vulnerable"	
	4B	int	0	4B	int	0	
a	Ļ			4B	string length	Ø (spurious)	
	4B	string length	3	4B	'\0'	3	
	4B	string chars	\u000D\0	4B	type	VAL_BYTEARRAY	B
		string chars	\u0008\0	4B	byte[] length	8	D1
	4B	type	VAL_BYTEARRAY	4B	buto[]	[0d,00,00,00]	
A <sub>1</sub>	4B	byte[] length	28	4B	byte[]	[1c,00,00,00]	
	28B	byte[] <bytes></bytes>		20B	string	intent	
			<bytes></bytes>	4B	type	VAL_INT	B <sub>2</sub>
			4B	int	42		

Parcel p = Parcel.obtain();

// Write header

// Write A0
p.writeString("\u0005");
p.writeInt(VAL\_PARCELABLE);
p.writeString(Vulnerable.class.getName());
p.writeInt(0);

// Write A1
p.writeString("\r\0\u0008");
p.writeInt(VAL\_BYTEARRAY);
p.writeInt(28);
p.writeString("intent");
p.writeInt(VAL\_INTEGER);
p.writeInt(42);

// Write A2
p.writeString("\r\0\u0009");
p.writeInt(VAL\_NULL);

// Backpatch length and rewind

Bundle A = unparcel(p); System.out.println("A = " + inspect(A));

Bundle B = unparcel(parcel(A)); System.out.println("B = " + inspect(B));









- Abuses AccountManagerService
  - Where KEY\_INTENT check happens
  - TOCTOU mismatch:
    - Bundle object ("self-")changed from deserialization to reserialization
- Triggers arbitrary Activity launching, from Settings app
  - Settings app (uid=1000 SYSTEM\_UID) is privileged and can launch arbitrary activities
  - "LaunchAnyWhere"
- Knowingly used in Malware campaigns (not covered in this talk)
  - Silently install packages

#### • Fix the individual r/w mismatches

- Yes, but doesn't scale
- Fix AccountManagerService
  - Yes, but what about other code paths?
- Fix Bundle
  - Yes!
  - What's wrong with Bundle?
  - => Lazy Bundle

• Fix the individual r/w mismatches => Yes, but doesn't scale

CVE-2017-0806	GateKeeperResponse	CVE-2018-9474	MediaPlayer.TrackInfo
CVE-2017-0664	AccessibilityNodeInfo	CVE-2018-9431	OSUInfo
CVE-2017-13288	PeriodicAdvertisingReport	CVE-2018-9522	StatsLogEventWrapper
CVE-2017-13289	ParcelableRttResults	CVE-2018-9523	Parcel.writeMapInternal()
CVE-2017-13286	OutputConfiguration	CVE-2021-0748	ParsingPackageImpl
CVE-2017-13287	VerifyCredentialResponse	CVE-2021-0928	OutputConfiguration
CVE-2017-13315	DcParamObject	CVE-2021-0685	ParsedIntentInfo
CVE-2017-13310	ViewPager's SavedState	CVE-2021-0921	ParsingPackageImpl
CVE-2017-13312	ParcelableCasData	CVE-2021-0970	GpsNavigationMessage
CVE-2017-13311	ProcessStats	CVE-2021-39676	AndroidFuture
CVE-2018-9471	NanoAppFilter	CVE-2022-20135	GateKeeperResponse

- Fix the individual r/w mismatches
  - Yes, but doesn't scale
- Fix AccountManagerService
  - Yes, but what about other code paths?
- Fix Bundle
  - Yes!
  - What's wrong with Bundle?
  - o => Lazy Bundle

• => Fix AccountManagerService? Yes



- Fix the individual r/w mismatches
  - Yes, but doesn't scale
- Fix AccountManagerService
  - Yes, but what about other code paths?
- Fix Bundle
  - Yes!
  - What's wrong with Bundle?
  - o => Lazy Bundle

#### • Fix Bundle

- Yes!
- What's wrong with Bundle?
  - Structure implicitly defined by the items and their payloads
  - Eager deserialization upon first retrieval/query
- => Lazy Bundle

- What's wrong with Bundle?
  - Structure implicitly defined by the items and their payloads



• If there is a r/w mismatch, the next read is affected

- What's wrong with Bundle?
  - Structure implicitly defined by the items and their payloads



- What's wrong with Bundle?
  - Eager deserialization upon first retrieval/query
  - To read an item => read all previous items (in practice we read all the bundle)



- What's wrong with Bundle?
  - Eager description upon first retrieval/query
  - With length prefix, we can skip items => only read (custom) items when queried
  - => Lazy bundle



- What's wrong with Bundle?
  - Lazy bundle: More resilient against system crashes / DoS



- What's wrong with Bundle?
  - Lazy bundle: More resilient against system crashes / DoS



- What's wrong with Bundle?
  - Lazy bundle: More resilient against system crashes / DoS



- What's wrong with Bundle?
  - Lazy bundle: More resilient against system crashes / DoS


### Bundle "FengShui" - Making it safe(r)

- What's wrong with Bundle?
  - Lazy bundle: More resilient against system crashes / DoS



#### CVE-2021-0928 (Novel in Android 12-beta)

- Arbitrary code execution in any app's process (including system app process UID:1000)
  - Different exploit technique than Bundle FengShui
  - Fixed in Android 12's official release
  - Reported and PoC-ed by Michał Bednarski (@<u>BednarTildeOne</u>)

### CVE-2021-0928 - Background: Broadcast

What happens when an app calls **sendBroadcast(intent)**?



#### Background Cont'd: Broadcast - ActivityInfo

Arbitrary code execution via tampering the ActivityInfo value.

- ActivityThread (in app's process) calls <u>handleReceiver</u> and eventually uses the applicationInfo object (within ActivityInfo) to <u>create</u> a LoadedApk instance.
- The LoadedApk object assigns the applicationInfo object's sourceDir value to its <u>appdir</u>, which is eventually used to to <u>create</u> the application's classLoader.
- Hence controlling the sourceDir value, the attacker application can make the victim process load an attacker-controlled APK and execute arbitrary code from there.

#### CVE-2021-0928 - The Mismatch



- When <u>exceptions</u> occurs:
  - The read stops before fully consuming the data
  - Exceptions caught gracefully

#### CVE-2021-0928: OutputConfiguration Deserialization

```
private OutputConfiguration(@NonNull Parcel source)
   int rotation = source.readInt();
   int surfaceSetId = source.readInt();
   int surfaceType = source.readInt();
   int width = source.readInt();
   int height = source.readInt();
   boolean isMultiResolutionOutput =
source.readInt() == 1;
   ArrayList<Integer> sensorPixelModesUsed = new
ArrayList<Integer>();
   source.readList(sensorPixelModesUsed,
Integer.class.getClassLoader());
```

#### CVE-2021-0928: OutputConfiguration Deserialization

```
private OutputConfiguration(@NonNull Parcel source)
   int rotation = source.readInt();
   int surfaceSetId = source.readInt();
   int surfaceType = source.readInt();
   int width = source.readInt();
   int height = source.readInt();
   boolean isMultiResolutionOutput =
source.readInt() == 1;
   ArrayList<Integer> sensorPixelModesUsed = new
ArrayList<Integer>();
   source.readList(sensorPixelModesUsed,
Integer.class.getClassLoader());
```



CVE-2021-0928



#### CVE-2021-0928



#### CVE-2021-0928

- system\_server prepares the Parcel Object with Intent, ActivityInfo, other params
- Victim app reads the Parcel object:
  - Exception only triggers when victim app's deserializing the Intent
  - Read stops before data fully consumed
  - Exception handled, deserialization continues
  - Intent's deserialization finished before the full Intent object is read
  - Starts reading ActivityInfo, from the wrong offset, within the attacker-controlled Intent object

0 .....

- Victim app execute the broadcast, using attacker controlled ActivityInfo
  - Arbitrary code execution



- Parcelable R/W mismatch via triggering exceptions
  - Read "less" than write, causing the next parameter read at the wrong offset
  - Exception only triggers in application's process
- Build PoC in the Intent object
  - Embed (R/W Mismatched) Parcelable objects in Intent. (Only became available in Android 12(S)-beta)

- Parcelable R/W mismatch via triggering exceptions
  - Read "less" than write, causing the next parameter read at the wrong offset
  - Exception only triggers in application's process
    - ClassNotFoundException with system\_server specific class
       PackageManagerException
- Build PoC in the Intent object
  - Embed (R/W Mismatched) Parcelable objects in Intent. (Only became available in Android 12(S)-beta)

- Parcelable R/W mismatch via triggering exceptions
  - Read "less" than write, causing the next parameter read at the wrong offset
  - Exception only triggers in application's process
- Build PoC in the Intent object
  - Embed (R/W Mismatched) Parcelable objects in Intent. (Only became available in Android 12(S)-beta)

```
ClipData(Parcel in) {
   mClipDescription = new ClipDescription(in);
   if (in.readInt() != 0) {
        mIcon = Bitmap.CREATOR.createFromParcel(in);
   } else {
        mIcon = null;
   mItems = new ArrayList<>();
   final int N = in.readInt();
   for (int i=0; i<N; i++) {</pre>
       CharSequence text = TextUtils.CHAR SEQUENCE CREATOR.createFromParcel(in);
       String htmlText = in.readString8();
        Intent intent = in.readTypedObject(Intent.CREATOR);
       Uri uri = in.readTypedObject(Uri.CREATOR);
       ActivityInfo info = in.readTypedObject(ActivityInfo.CREATOR);
        TextLinks textLinks = in.readTypedObject(TextLinks.CREATOR);
        Item item = new Item(text, htmlText, intent, uri);
        item.setActivityInfo(info);
        item.setTextLinks(textLinks);
       mItems.add(item);
```

Build PoC in the Intent object

• Embed (R/W Mismatched) Parcelable objects in Intent. (Only became available in Android 12(S)-beta)



#### CVE-2021-0928 - Final PoC

Intent

- mClipData = ClipData
  - mltems.get(0).mActivityInfo = ActivityInfo
    - applicationInfo = ApplicationInfo
      - splitDependencies.get(0) = ZenPolicy // For Padding (Not covered in the talk)
        - mVisualEffects.get(0) = OutputConfiguration
          - mSensorPixelModesUsed.get(0) = WindowContainerTransaction
            - mHierarchyOps.get(0) = PackageManagerException
          - mSensorPixelModesUsed.get(1) = Bundle
            - Wrapup ClipData
            - Wrapup Intent
            - Next params for scheduleReceiver (ActivityInfo, compatinfo,...)

- Wrapup ClipData
- Wrapup Intent

### CVE-2021-0928: Victim process' read



<u>ttps://github.com/michalbednarski/ReparcelBug2</u>

- Apply targeted fixes
- Lazy bundle doesn't help, can we prefix every Parcelable (or IPC parameter)?
- Not enough validation
  - In domain-specific code
  - In infrastructure code
- Manual serialization/deserialization is prone to mismatches

#### • Apply targeted fixes

#### • Fix the mismatches: Exception swallowing

651		try {
652		OutputConfiguration outputConfiguration = new OutputConfiguration(source);
653		return outputConfiguration;
654		<pre>} catch (Exception e) {</pre>
655		Log.e(TAG, "Exception creating OutputConfiguration from parcel", e);
656		return null;
657		
	651	return new OutputConfiguration(source);

#### • Fix the mismatch: readList() for int array

738 739	738	<pre>ArrayList<integer> sensorPixelModesUsed = new ArrayList<integer>(); source.readList(sensorPixelModesUsed, Integer.class.getClassLoader()); int[] sensorPixelModesUsed = source.createIntArray();</integer></integer></pre>
	0	Remove ActivityInfo from ClipData
1211	1237	<pre>dest.writeTypedObject(item.mActivityInfo, flags);     dest.writeTypedObject(mParcelItemActivityInfos ? item.mActivityInfo : null, flags);</pre>

• Lazy bundle doesn't help



- Lazy bundle doesn't help
- Can we prefix every Parcelable or IPC parameter?
  - Not practical (backwards compatibility problems)
  - Performance :(

#### Java generated code



- Apply targeted fixes
- Lazy bundle doesn't help, can we prefix every Parcelable (or IPC parameter)?
- Not enough validation
  - In domain-specific code
  - In infrastructure code
- Manual serialization/deserialization is prone to mismatches

#### • Not enough validation:

• In infrastructure code

#### • Recap: Arbitrary deserialization Parcel container

Intent

ApplicationInfo.splitDependencies is SparseArray<int[]> but attacker inserts ZenPolicy item ZenPolicy.mVisualEffects is ArrayList<Integer> but attacker inserts OutputConfiguration item

- mClipData = ClipData
  - mItems.get(0).mActivityInfo = ActivityInfo
    - applicationInfo = ApplicationInfo
      - splitDependencies.get(0) = ZenPolicy
        - mVisualEffects.get(0) = OutputConfiguration
          - mSensorPixelModesUsed.get(0) = WindowContainerTransaction
            - mHierarchyOps.get(0) = PackageManagerException
          - mSensorPixelModesUsed.get(1) = Bundle
            - Wrapup ClipData
            - Wrapup Intent
            - Next params for scheduleReceiver (ActivityInfo, compatinfo,...)

- Wrapup ClipData
- Wrapup Intent

OutputConfiguration.mSensorPixelModesUsed is ArrayList<Integer> but attacker inserts WindowContainerTransaction item

. . .

#### • Not enough validation:

- In infrastructure code
- Recap: Arbitrary deserialization Parcel container



• Wrapup Intent

- Not enough validation:
  - In infrastructure code: Arbitrary deserialization Parcel container

In ApplicationInfo, we happily deserialize **ZenPolicy** as item

splitDependencies = source.readSparseArray(null);

But we know splitDependencies only holds int[] items a priori

public SparseArray<int[]> splitDependencies;

=> In the worst case, this enables attacks like this CVE
=> In the best case (non-malicious), this will cause exceptions later on

- Not enough validation:
  - In infrastructure code: Arbitrary deserialization Parcel container

In ApplicationInfo, we happily deserialize **ZenPolicy** as item

splitDependencies = source.readSparseArray(null);

But we know splitDependencies only holds int[] items a priori

public SparseArray<int[]> splitDependencies;

Fix: validate **splitDependencies** item types **<u>before</u>** deserialization

- Not enough validation:
  - In infrastructure code: Arbitrary deserialization Parcel container

Fix: validate **splitDependencies** item types <u>**before</u>** deserialization</u>

splitDependencies = source.readSparseArray(null);
splitDependencies = source.readSparseArray(null, int[].class);

=> New **Parcel**, **Bundle** & **Intent** replacement APIs that take an extra **Class<T>** parameter => Old APIs were **@Deprecated** 

- Not enough validation:
  - In infrastructure code: Arbitrary deserialization Parcel container
  - New Parcel, Bundle & Intent replacement APIs that take an extra Class<T>
    - How to migrate 600+ call sites?
      - Need to infer static type
      - Then, add inferred Class<T>

```
public ZenPolicy createFromParcel(Parcel source) {
    ZenPolicy policy = new ZenPolicy();
    policy.mPriorityCategories = source.readArrayList(Integer.class.getClassLoader());
    policy.mVisualEffects = source.readArrayList(Integer.class.getClassLoader());
    policy.mPriorityCategories = source.readArrayList(Integer.class.getClassLoader(), java.lang.Integer.class);
    policy.mVisualEffects = source.readArrayList(Integer.class.getClassLoader(), java.lang.Integer.class);
```

- Not enough validation:
  - In infrastructure code: Arbitrary deserialization Parcel container
  - New Parcel, Bundle & Intent replacement APIs that take an extra Class<T>
    - How to migrate 600+ call sites?
      - Need to infer static type
      - Then, add inferred Class<T>
      - => Use Android Lint!

```
public ZenPolicy createFromParcel(Parcel source) {
    ZenPolicy policy = new ZenPolicy();
    policy.mPriorityCategories = source.readArrayList(Integer.class.getClassLoader());
    policy.mVisualEffects = source.readArrayList(Integer.class.getClassLoader());
    policy.mPriorityCategories = source.readArrayList(Integer.class.getClassLoader(), java.lang.Integer.class);
    policy.mVisualEffects = source.readArrayList(Integer.class.getClassLoader(), java.lang.Integer.class);
```

- Not enough validation:
  - In infrastructure code: Parcel EOF check in IPC interfaces.
    - We can't enforce boundaries in IPC parameters :(
    - But we can verify that the parcel was fully consumed
    - Not strong mitigation, but
      - Uncovers mismatch/security issues
      - Uncovers correctness/non-security bugs

- Not enough validation:
  - In infrastructure code: Parcel EOF check in IPC interfaces.



- Not enough validation:
  - In infrastructure code: Parcel EOF check in IPC interfaces.



- Manual serialization/deserialization is prone to mismatches
  - Mismatch detection at runtime
  - => Use the length prefix in Lazy Bundle

```
private <T> T readValue(@Nullable ClassLoader loader, @Nullable Class<T> clazz,
        @Nullable Class<?>... itemTypes) {
   int type = readInt();
   final T object;
   if (isLengthPrefixed(type)) {
       int length = readInt();
       int start = dataPosition();
       object = readValue(type, loader, clazz, itemTypes);
       int actual = dataPosition() - start;
       if (actual != length) {
           Slog.wtfStack(TAG,
                    "Unparcelling of " + object + " of type " + Parcel.valueTypeToString(type)
                            + " consumed " + actual + " bytes, but " + length + " expected.");
     else
       object = readValue(type, loader, clazz, itemTypes);
   return object;
```

- Apply targeted fixes
- Not enough validation
  - In domain-specific code
  - In infrastructure code
    - New Parcel, Bundle & Intent APIs that take Class<?>
    - Parcel EOF check in IPC interfaces
- Manual serialization/deserialization is prone to mismatches
  - Mismatch detection at runtime

# Parcel Mismatch Ecosystem Perspective

1st report in <u>Sept, 2014</u> by Michal Bednarski (<u>@BednarTildeOne</u>)

28 **exploitable** Parcel Mismatch reports were submitted to the Android Vulnerability Rewards program (VRP).

Known to be abused by malware in the wild.



### **Android Vulnerability Rewards Program**

0

2016

# VRP ingestion consists of multiple sources:

Google Internal 8.2% Fuzzing Android Red Team 5.1% Partners 6.2% Researchers 62.1%

**2022 Incoming Reports Distribution** 

#### Severity Risk Assessment

2017

Based on Android security model that measures Android user risk.

Security Severity of Incoming Reports by Year Critical High Moderate Low NSBC 3000 2000 1000

2018

2019

2020

2021

2022

# Parcel Mismatch Security Severity- High

According to the Android Severity Guidelines, exploitable Parcel Mismatch is rated as a **High** Security vulnerability:

- "Local arbitrary code execution in a privileged context, the bootloader chain, THB, or the OS Kernel"
- Reason: parcel mismatch allows launching arbitrary activities as the Settings app with (SYSTEM\_UID). It can be used to achieve the following privileged actions:
  - Installing an arbitrary APK
  - Resetting the lock screen password
# **Exploit Reward Chain Program**

#### Code execution reward amounts

Description	Maximum Reward
Pixel Titan M with Persistence, Zero click	Up to \$1,000,000
Pixel Titan M without Persistence, Zero click	Up to \$500,000
Local App to Pixel Titan M without Persistence	Up to \$300,000
Secure Element	Up to \$250,000
Trusted Execution Environment	Up to \$250,000
Kernel	Up to \$250.000
Privileged Process	Up to \$100,000

On top of VRP, we also have an <u>exploit chain reward program</u>.

Michał Bednarski's (@BednarTildeOne) novel exploit CVE-2021-0928 qualified as a Code Execution exploit chain in a Privileged Process



## Life Cycle of a Vulnerability







- For potentially exploitable vulnerabilities, collaborate with Malware Team to add Play Store detection
- Collaborate with various security teams for in-depth remediation:
  - API/Platform hardening
  - Continuous Fuzzing
  - Red team engagement

### Once we know about it, we can fix it



#### bughunters.google.com



## Questions?



