The Lifetime of Android API vulnerabilities: case study on the JavaScript-to-Java interface

Daniel R. Thomas[1] Alastair R. Beresford[1] Thomas Coudray[2]
Tom Sutcliffe[2] Adrian Taylor[2]

[1]Computer Laboratory, University of Cambridge, United Kingdom
firstname.lastname@cl.cam.ac.uk

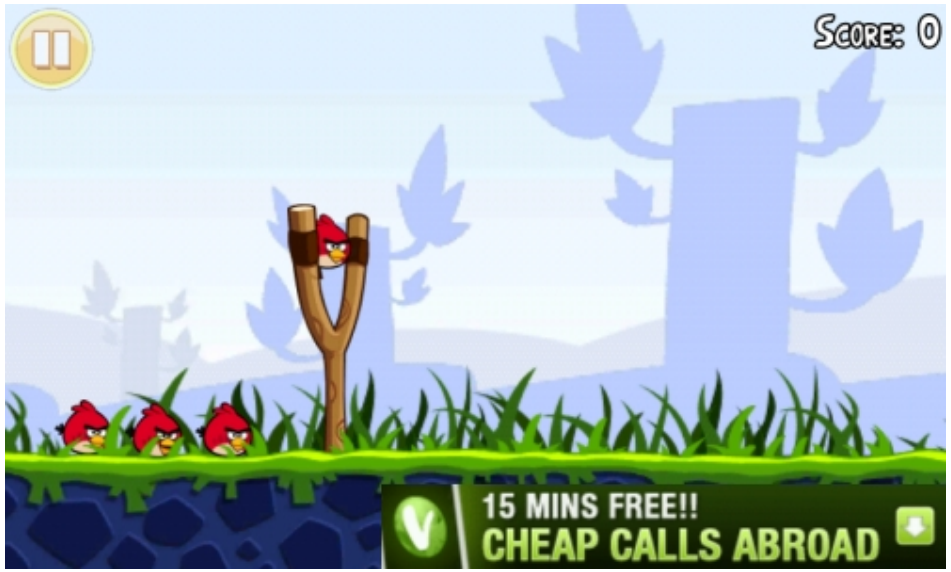
[2]Bromium, Cambridge, United Kingdom
thomas.coudray.fr@gmail.com, tom.sutcliffe@bromium.com,
adrian@bromium.com

23rd Security Protocols Workshop

UNIVERSITY OF CAMBRIDGE

Daniel: 5017 A1EC 0B29 08E3 CF64 7CCD 5514 35D5 D749 33D9
Alastair: 9217 482D D647 8641 44BA 10D8 83F4 9FBF 1144 D9B3

Android apps display ads in WebViews



WebViews display HTML/CSS and **JavaScript**.

JavaScript communicates with Java

- Collect information for ads
- Provide interactivity
- Sit down in a coffee shop and open angry birds, now your phone is compromised and infecting other phones.

```
/** Show a toast from the web page */
public void showToast(String toast) {
    Toast.makeText(context, toast, LENGTH).show();
}
```

The JavaScript-to-Java interface vulnerability

```
<script>
    android.getClass()
        .forName('java.lang.Runtime')
        .getMethod('getRuntime',null)
        .invoke(null,null).exec(['id']);
</script>
```

JavaScript attack, assuming *android* is the JavaScript alias for the exposed Java object.

- Apps use WebViews to display HTML fetched over HTTP.
- Bridge from JavaScript-to-Java exposes *all* public methods.
- Android worm.

Two approaches to fixes

1. Change the API and require apps to recompile (2012)

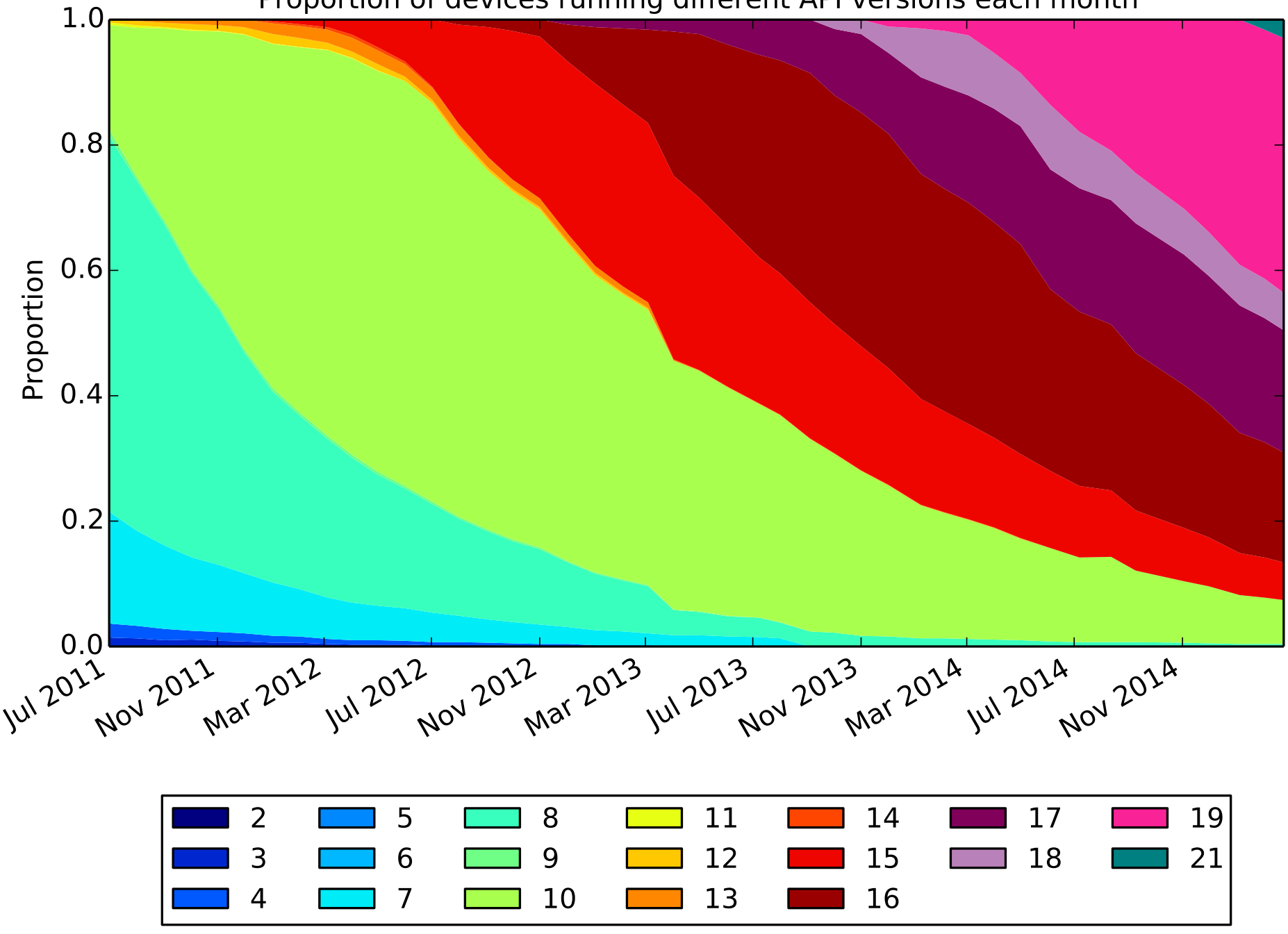
	target API < 17	target API ≥ 17
device API < 17	Vulnerable	Vulnerable
device API ≥ 17	Vulnerable	Safe

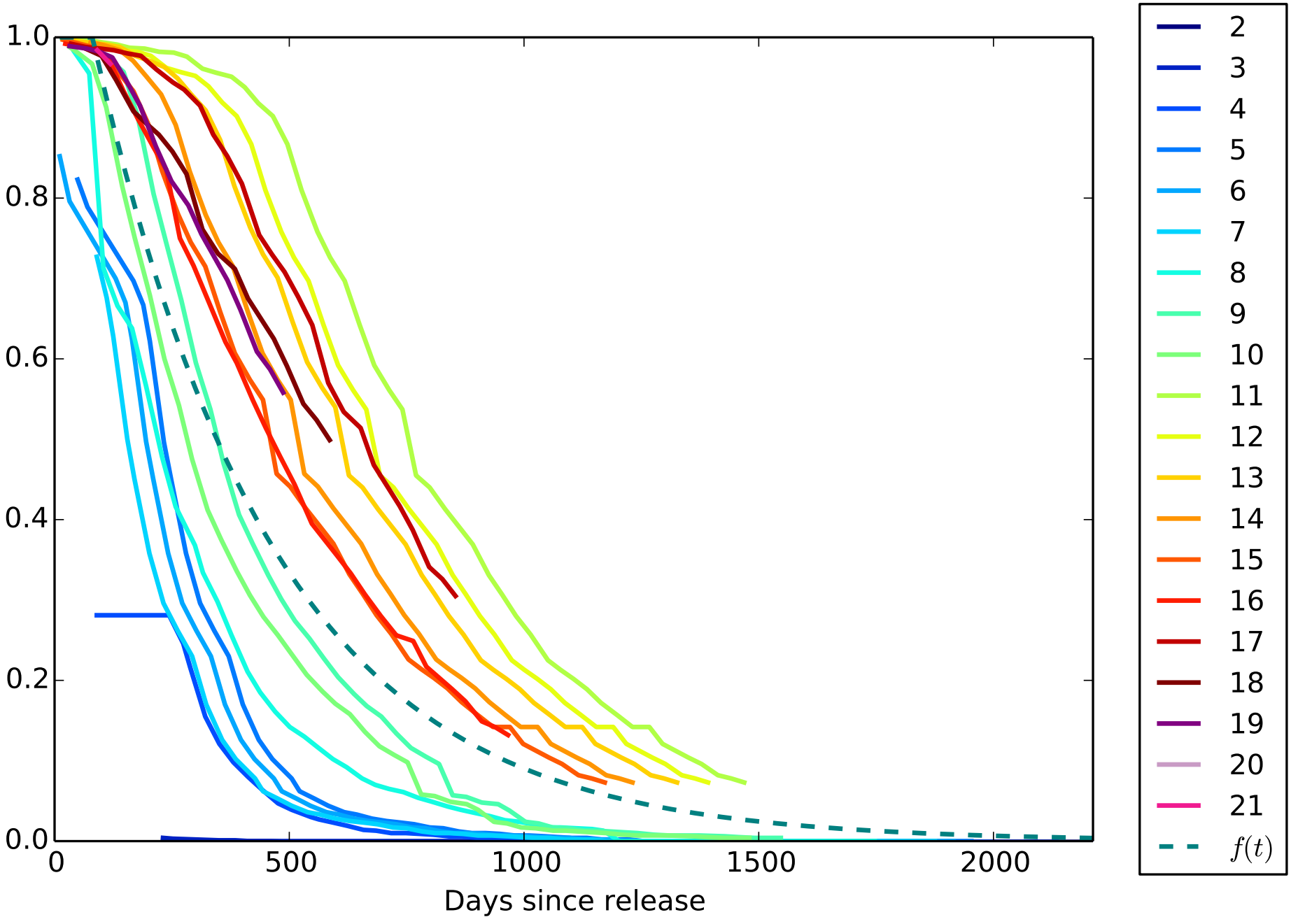
```
/** Show a toast from the web page */
@JavascriptInterface
public void showToast(String toast) {
    Toast.makeText(context, toast, LENGTH).show();
}
```

Two approaches to fixes

1. Change the API and require apps to recompile (2012)
2. Block calls to `.getClass` (2014)

	target API < 17	target API $\geq$ 17
API < 17	Vulnerable	Vulnerable
API $\geq$ 17 and OS < 4.4.3	Vulnerable	Safe
OS > 4.4.3	Safe(ish)	Safe

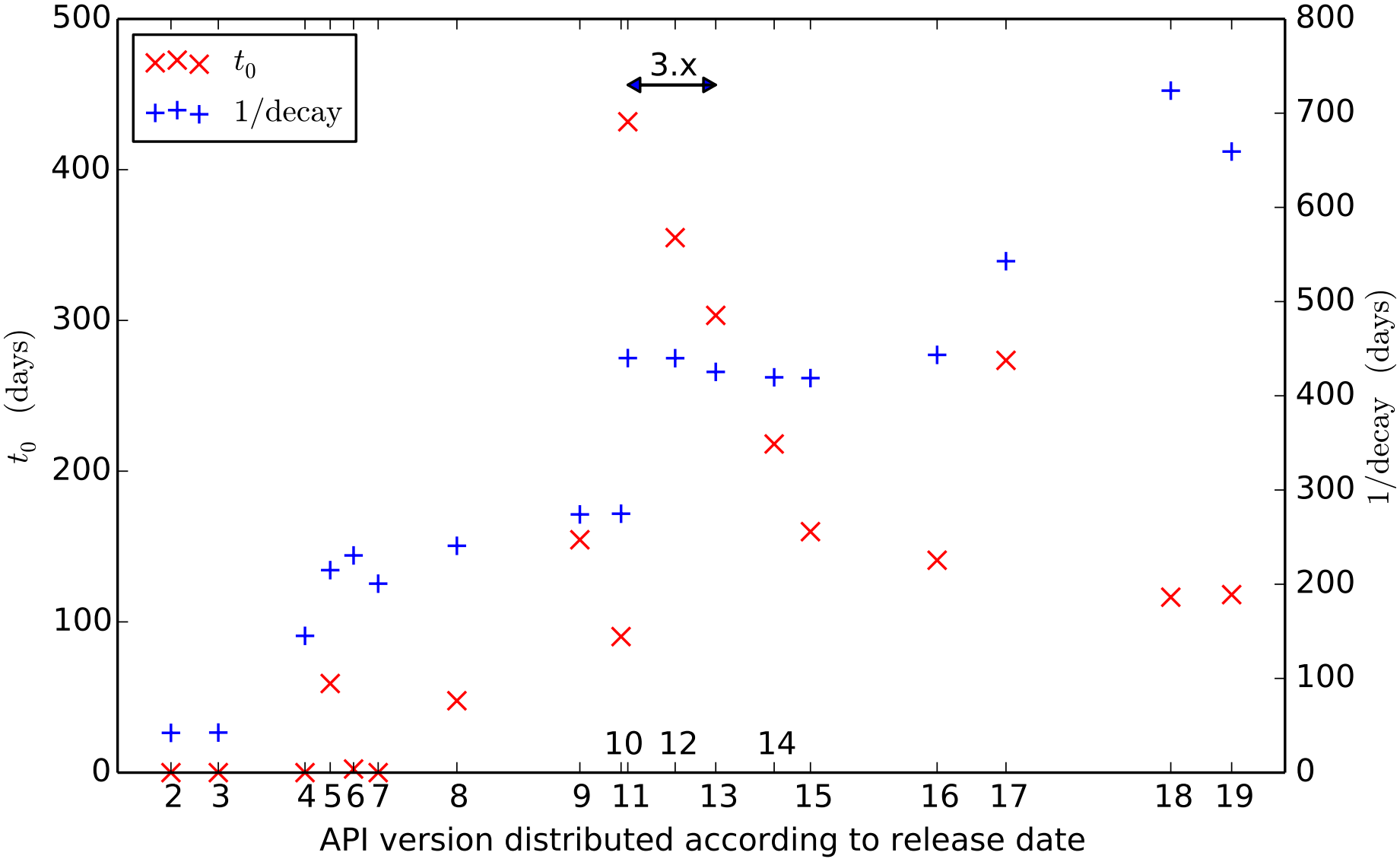

Proportion of devices running different API versions each month
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
Jul 2011
Nov 2011
Mar 2012
Jul 2012
Nov 2012
Mar 2013
Jul 2013
Nov 2013
Mar 2014
Jul 2014
Nov 2014
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
21

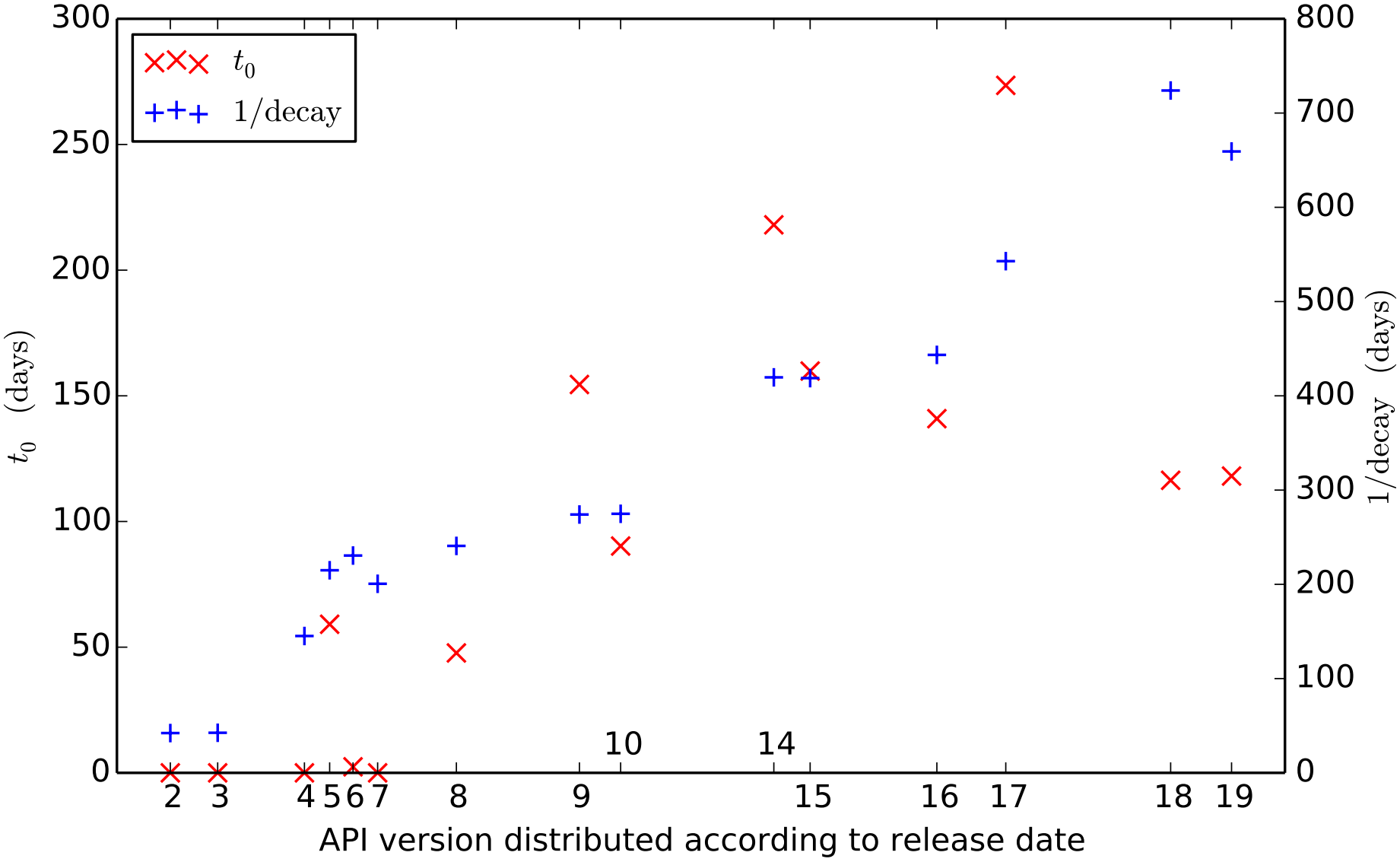

1.0
0.8
0.6
0.4
0.2
0.0
0
500
1000
1500
2000
Days since release
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
$f(t)$

Meaningful curve fit

$f(t)$: a combination of an exponential function together with a delay t_0 which offsets the start time:

$$f(t) = \begin{cases} 1.0 & \text{if } t < t_0 \\ e^{-\text{decay}(t-t_0)} & \text{otherwise} \end{cases}$$


t_0
1/decay
3.x
t_0 (days)
1/decay (days)
API version distributed according to release date
0
100
200
300
400
500
600
700
800
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19


t_0
1/decay
t_0 (days)
1/decay (days)
API version distributed according to release date
0
50
100
150
200
250
300
0
100
200
300
400
500
600
700
800
2
3
4
5
6
7
8
9
10
14
15
16
17
18
19

Apps are vulnerable (and have not upgraded)

- We scanned 102 174 apps.
- 59% of apps which could be vulnerable had not upgraded their target API version.
- On an outdated device vulnerable apps were started 1.38 ± 0.11 times a day.
- On an up to date device vulnerable apps were started 0.6 ± 0.0 times a day.

Conclusion

The Lifetime of Android API vulnerabilities: case study on the JavaScript-to-Java interface

Two sided fixes are hard for API vulnerabilities, even when there is one coordinating party (Google) who has a strong influence on both sides. Fixing it takes 5.2 ± 1.2 years.

- Daniel R. Thomas drt24@cam.ac.uk
  5017 A1EC 0B29 08E3 CF64 7CCD 5514 35D5 D749 33D9
- Alastair R. Beresford arb33@cam.ac.uk
  9217 482D D647 8641 44BA 10D8 83F4 9FBF 1144 D9B3

- Install Device Analyzer for Android
  https://deviceanalyzer.cl.cam.ac.uk/