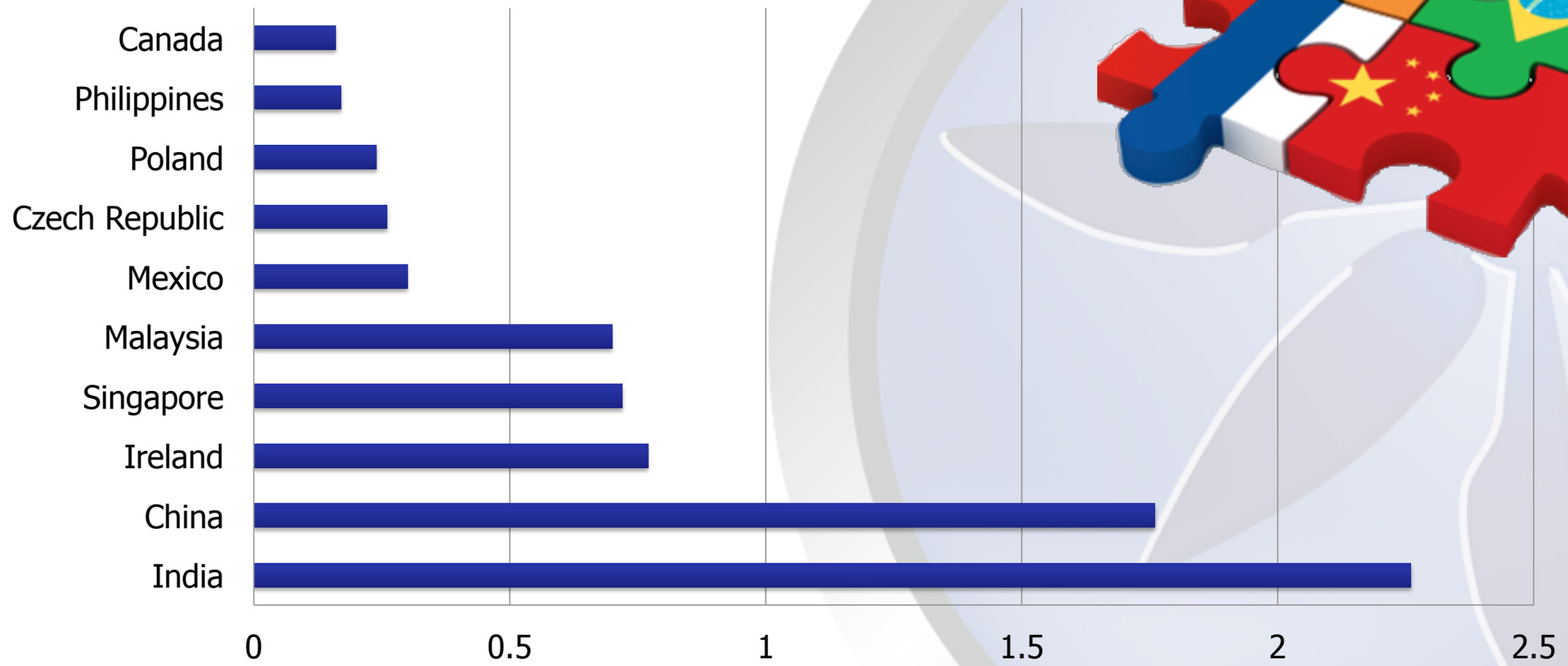
Malicious Code Detection

BRIC Breaking Through Static Analysis

Marina Khainson
Cigital
mkhainson@cigital.com

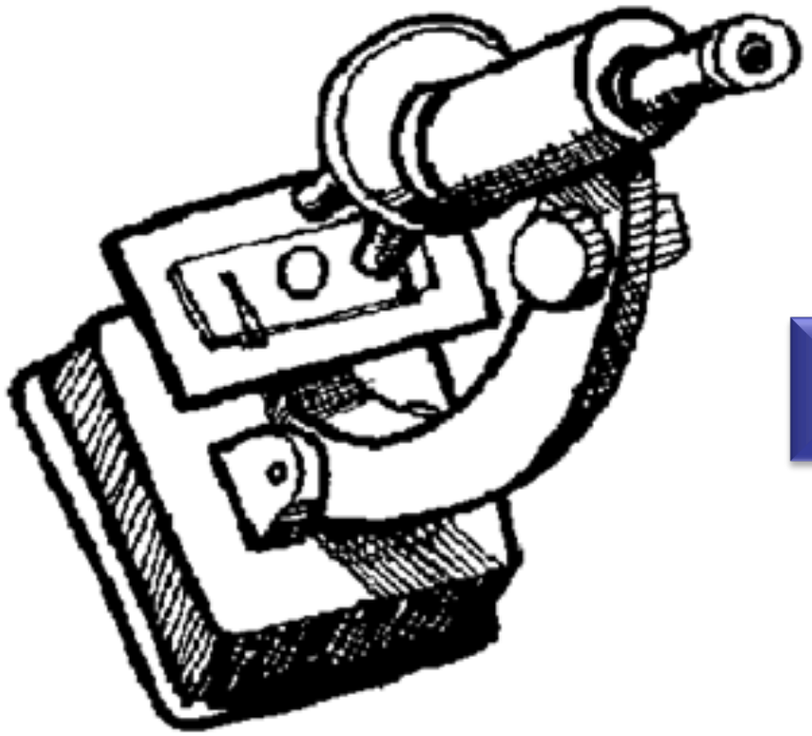
Why?

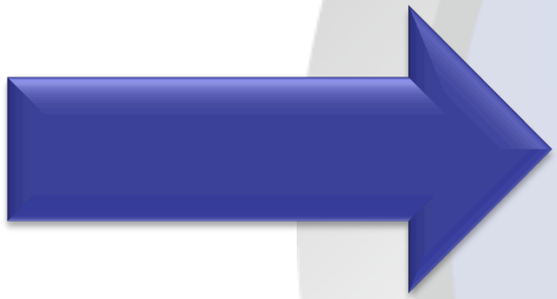


Guidance for Addressing Malicious Code Risk

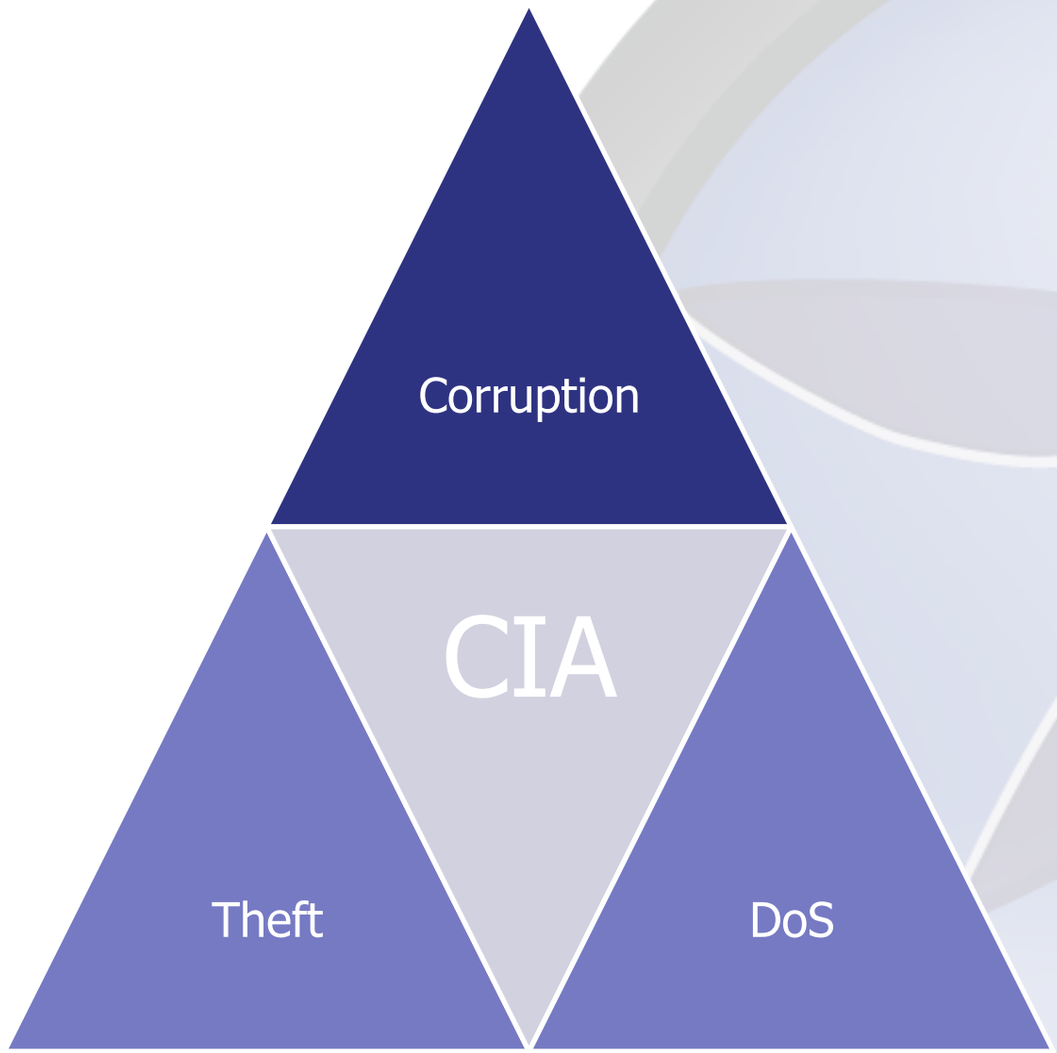
	Security-Enforcing Software		
	Custom-developed	**Pre-existing with Source**	**Pre-existing with Executable**
Acquisition	Trustworthy developers (contractor reputation)	NIAP or FIPS evaluation Acquisition Policy, including: • Background checks on developer/ supplier • Trusted distribution (checksums, signatures, authenticated out of band distribution channel) • Blind buys	NIAP or FIPS evaluation Acquisition Policy, including: • Background checks on developer/ supplier • Trusted distribution (checksums, signatures, authenticated out of band distribution channel) • Blind buys
Design	Separation of duties: Designers modify only parts for which they're responsible Reputable, well-understood high-level design languages and tools. Least Privilege/Fail Secure Design	n/a	n/a
Construction	Separation of duties: Coders modify only code for which they're responsible Software evaluation tools Code review	n/a	n/a
Testing	Software evaluation tools Security testing to ensure security requirements are fulfilled. Inter-team Testing	Software evaluation tools Security testing to ensure security requirements are fulfilled. Inter-team Testing	Software evaluation tools Security testing to ensure security requirements are fulfilled.

Malware Analysis vs. MCD

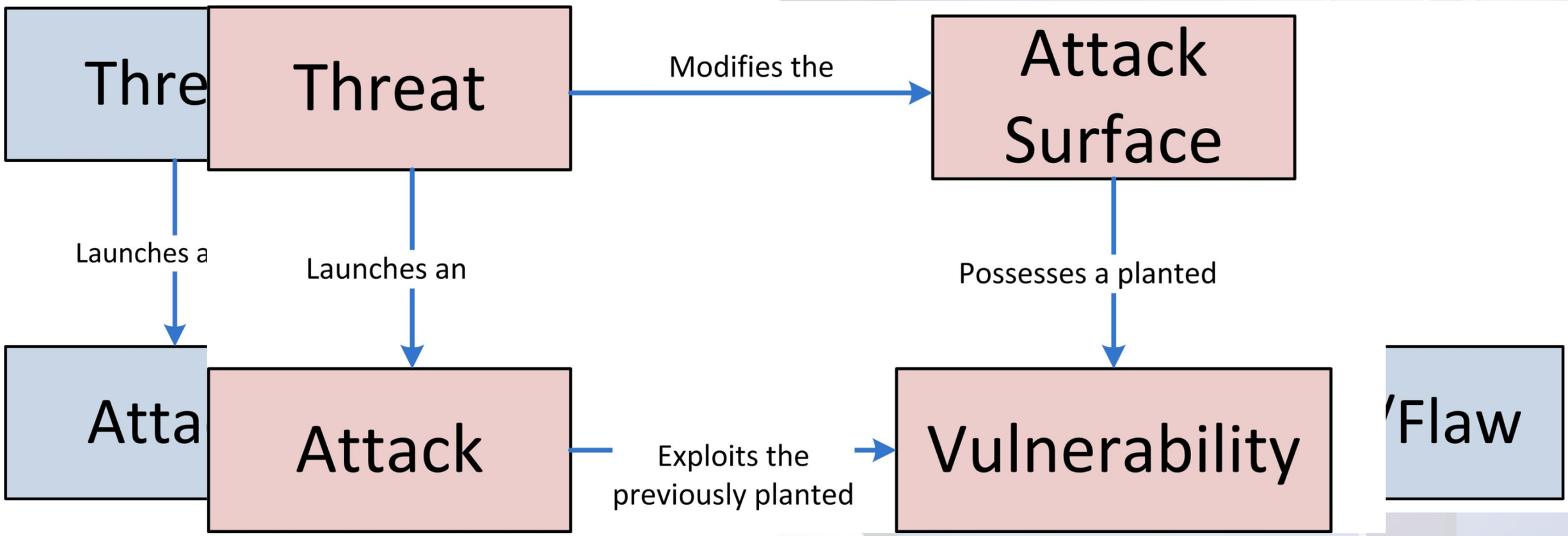




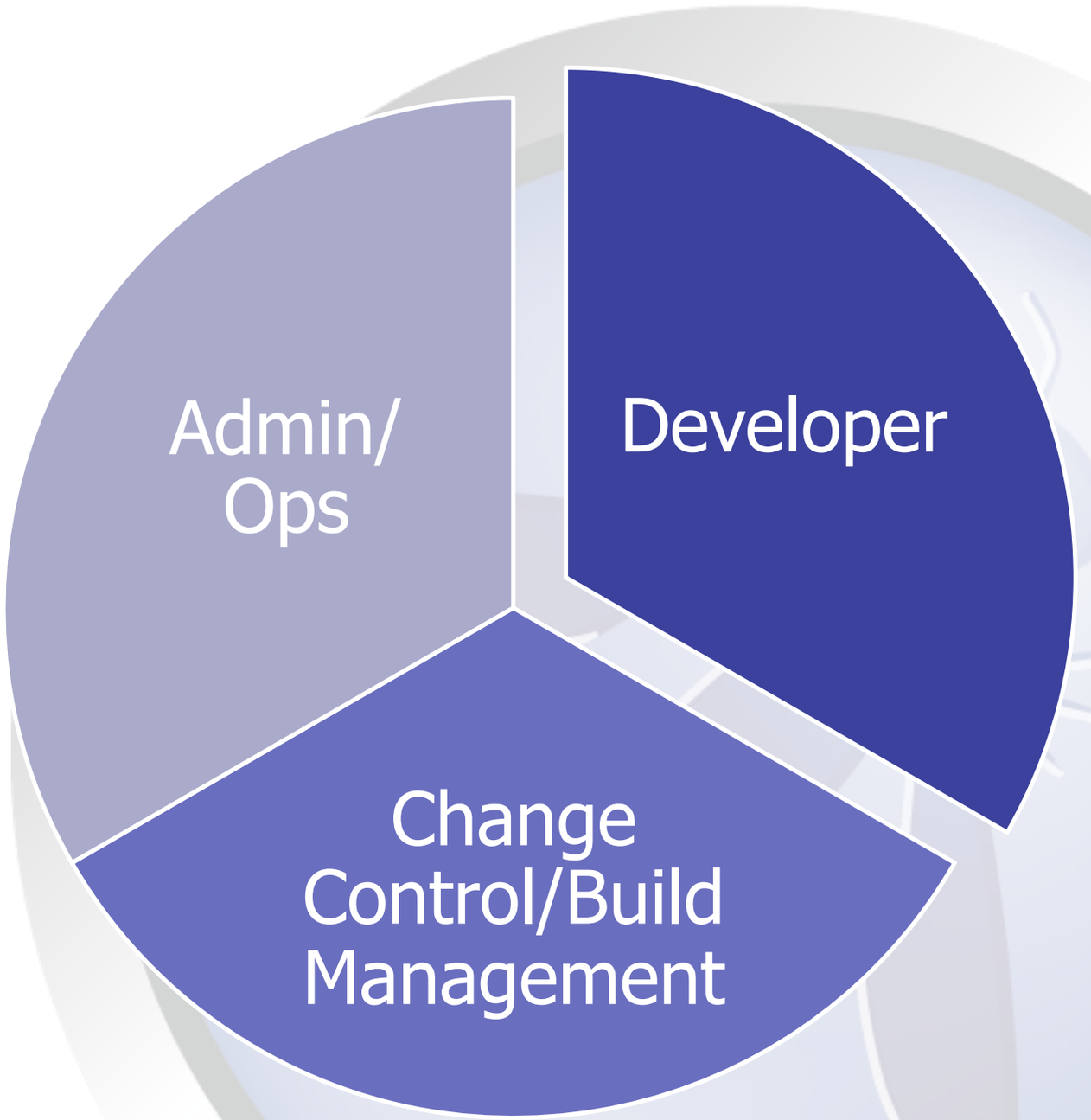
“Maliciousness”



Introducing MCD through SCR



The Threat



Development

- Source code
- 3rd Party library manipulation

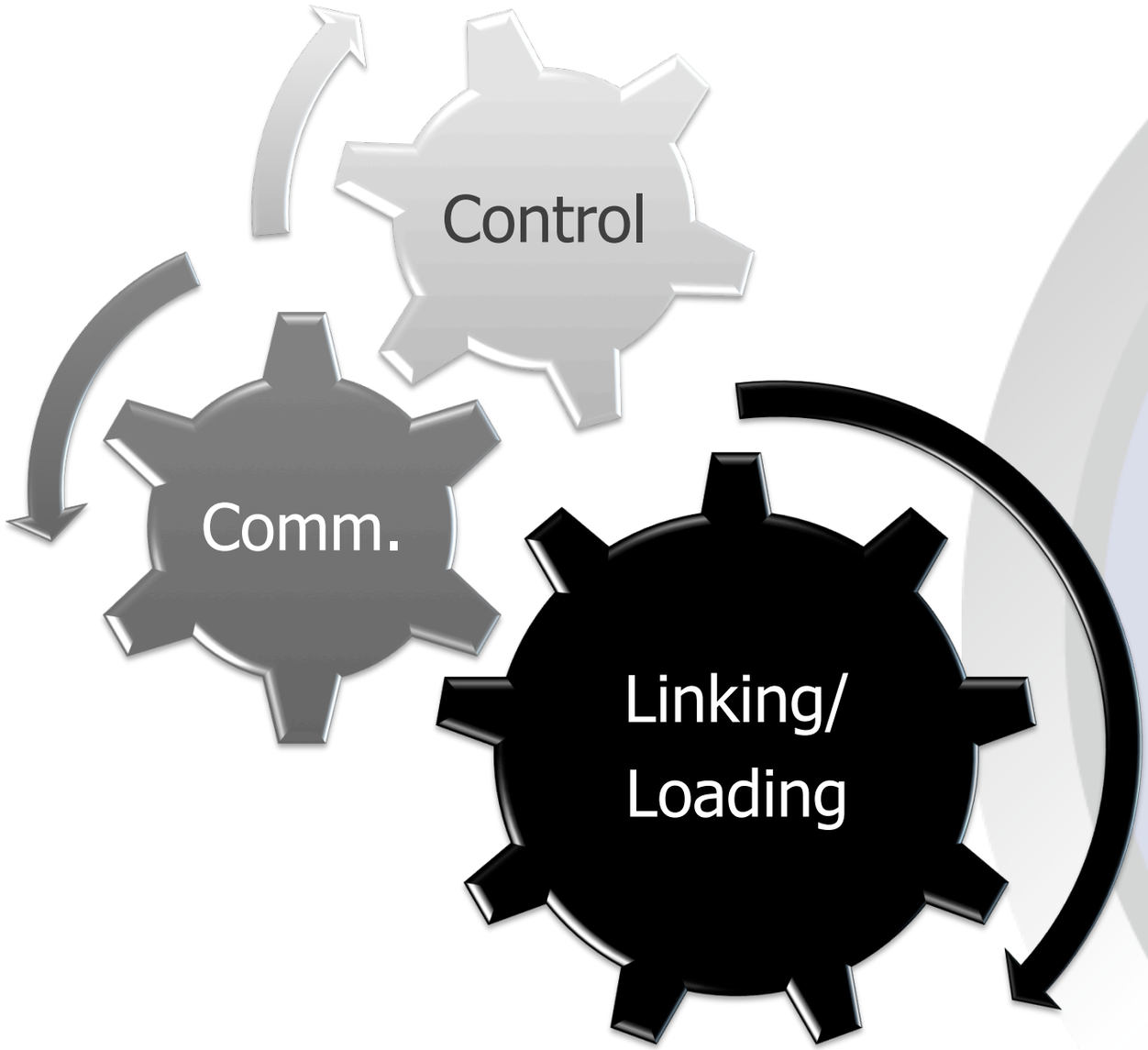
Promotion / CFG Mgmt

- Dependency manipulation / injection
- Unauthorized code promotion

Production

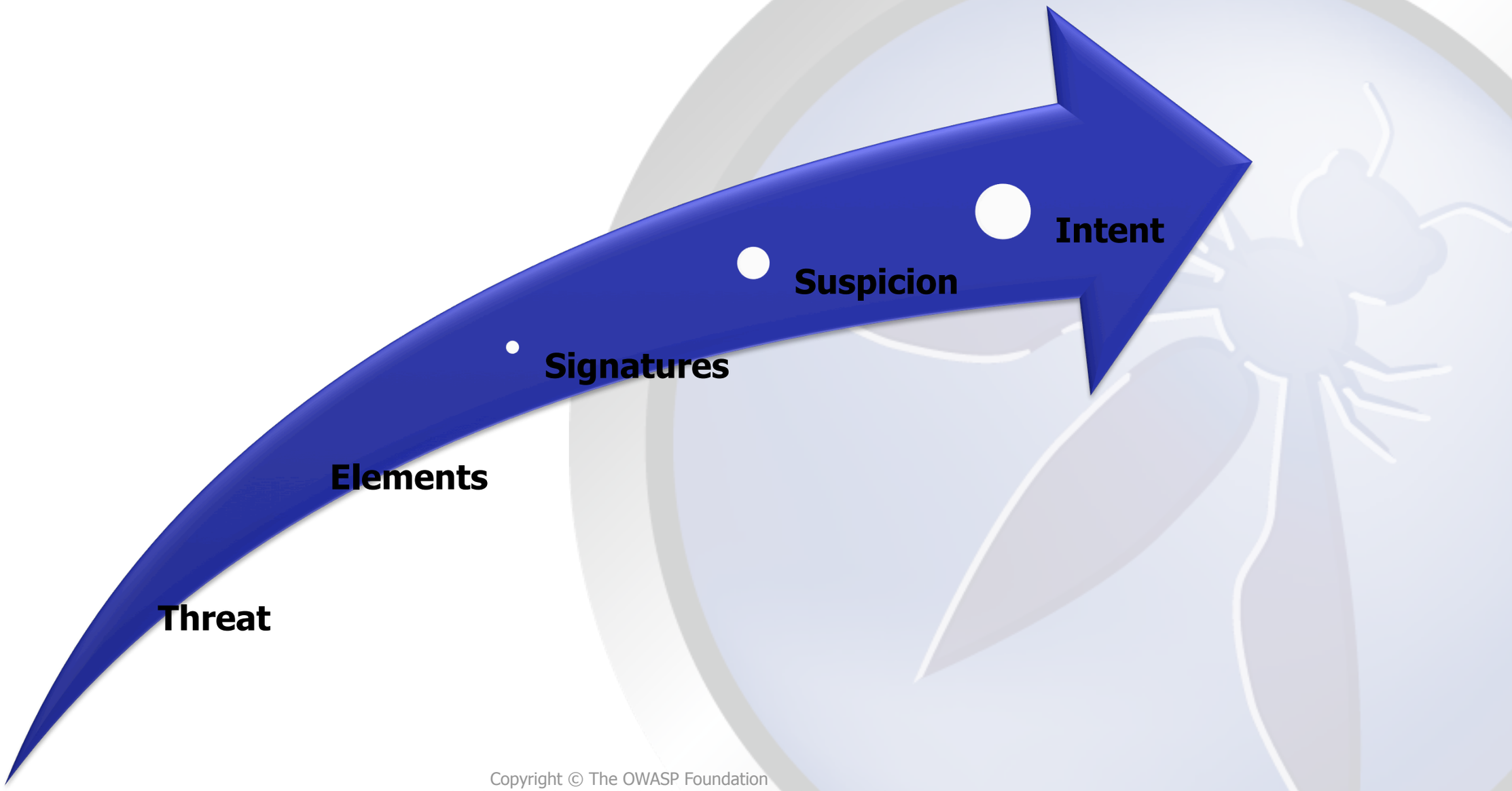
- Code injection vectors
- Dynamic update

Architecture



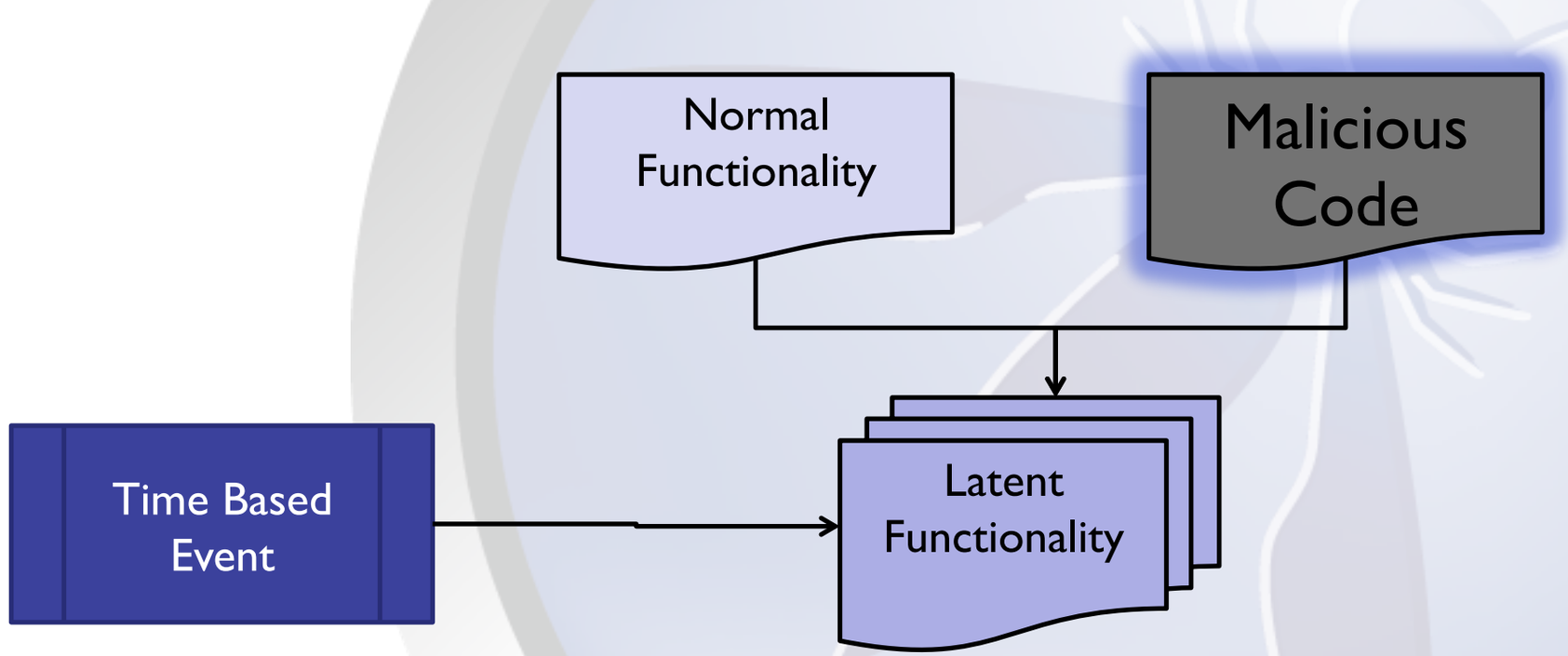
- Patterns
- Elements of malicious design

Evolution of a Methodology

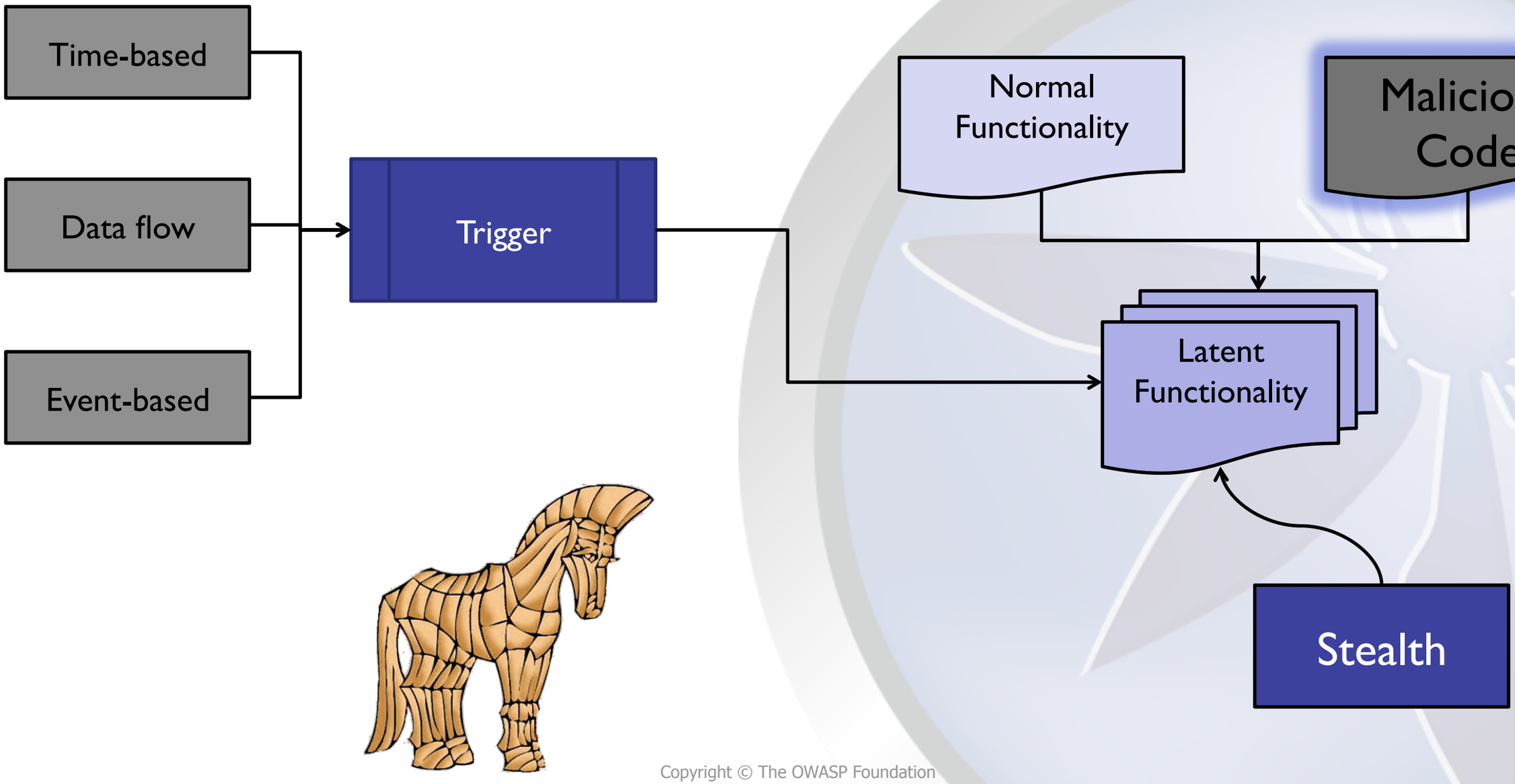


Time Bomb

- Trigger
- Functionality



Trojan



Resource Exhaustion

Intent?

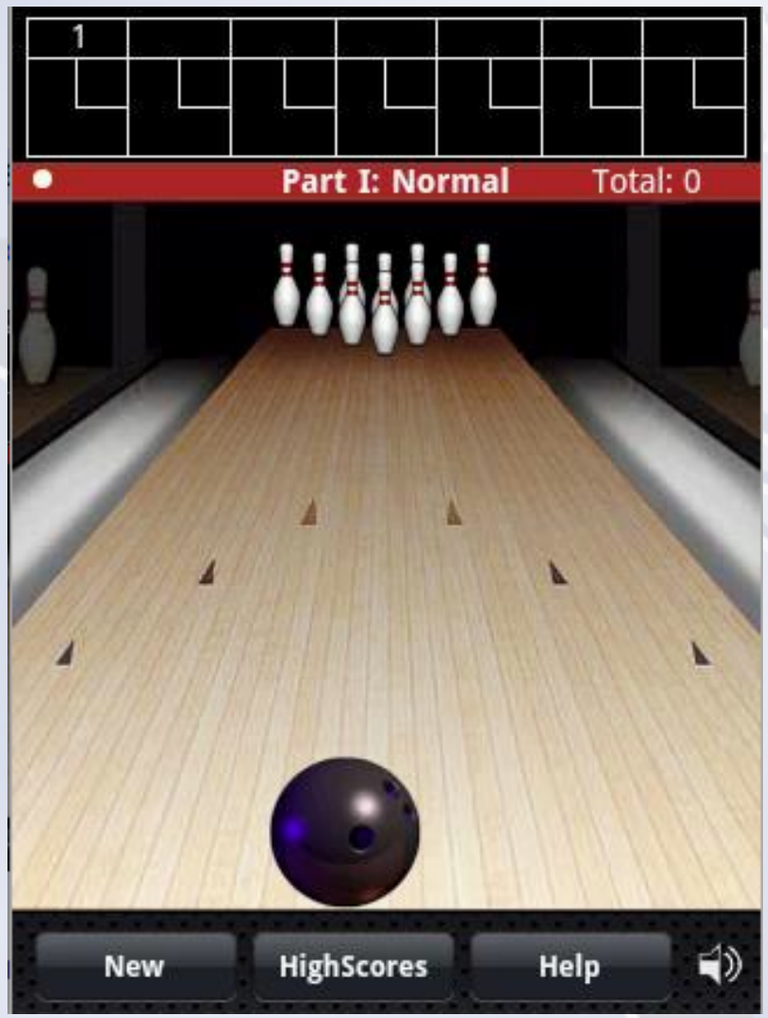
Case Study

Methodology:

1. Normal functionality
2. Automated analysis:
    a. *Semantic rules*
    b. *Static strings*
3. Manual analysis

Bowling Time

It is Bowling Time! Now on android with 3D graphics and realistic ball spin. Simply a good classic arcade style bowling game.



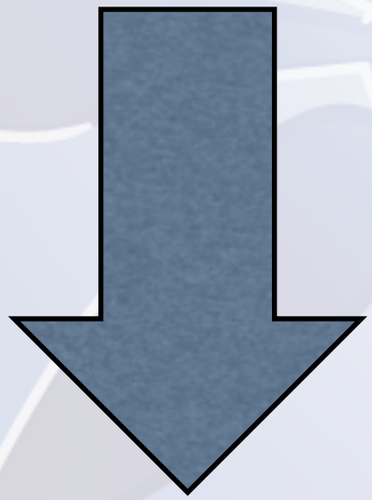
Semantic Rules

1. com/android/root/Setting.java:
   - *process = Runtime.getRuntime().exec(s);*
2. com/android/root/udevRoot.java:
   - *Process process = Runtime.getRuntime().exec("/system/bin/mount");*
   - *process = Runtime.getRuntime().exec("/system/bin/mount");*
   - *Process process1 = Runtime.getRuntime().exec(s10);*
   - *Process process2 = Runtime.getRuntime().exec(s12);*
3. jackpal/androidterm/Exec.java:
   - *System.loadLibrary("androidterm");*

Static Strings

a) "chmod 04755 /system/bin/profile"
b) "chmod 6755 /system/bin/profile\n"
c) "chmod 770 "
d) "chmod 777 "
e) "chown 0.0 /system/bin/profile\n"
f) "chown root.root /system/bin/profile\n"
g) "kill -9 "
h) "mount -o remount rw /system\n"
i) "mount -o remount rw system\nexit\n"
j) "mount -o remount,"
k) "mount -o remount,ro -t "
l) "mount -o remount,rw -t "
m) "6^)(9-p35a%3#4S!4S0)$Yt%^&5(j.g^&o(*0)$Yv!#O@6GpG@=+3j.&6^)(0-=1"

High Entropy



Manual Analysis I

- udevRoot.java:

```
private boolean runExploid()
{
    ...
    Process process = Runtime.getRuntime().exec("/system/bin/mount");
    ...
    ...
    if(s == null)
        s = "/dev/block/mtdblock5";
    if(s3 == null)
        s3 = "/dev/block/mtdblock3";
    String s5 = (new StringBuilder("mount -o remount,rw -t ")).append(s4).append(" ").append(s3).append(" /system").toString();
    ...
    File file1;
    File file = ctx.getFilesDir();
    file1 = new File(file, "exploid");
    if(!file1.exists())
        break MISSING_BLOCK_LABEL_434;
    StringBuilder stringbuilder = new StringBuilder("chmod 770 ");
    ...
}
```

Manual Analysis II

- Hard coded key:

```
KEYVALUE = "6^)(9-p35a%3#4S!4S0)$Yt%^&5(j.g^&o(*0)$Yv!#O@6GpG@=+3j.&6^)(0-=1".getBytes();
 …
 public static void crypt(byte abyte0[])
{
        int i = 0;
        int j = 0;
        do
        {
                …
                byte byte1 = KEYVALUE[i];
                byte byte2 = (byte)(byte0 ^ byte1);
                abyte0[j] = byte2;
                i++;
                i…
        } while(true);
}
```

Manual Analysis III

- "IMEI" & "IMSI":

"?><Request><Protocol>1.0</Protocol><Command>0</Command><ClientInfo><Partner>%s</Partner><ProductId>%s</ProductId><IMEI>%s</IMEI><IMSI>%s</IMSI><Modle>%s</Modle></ClientInfo></Request>"

Lessons Learned

- Broad searches
- Static strings yield good results
  - *Hex*
  - *IP/Host encodings*
- Imports are important

Future work

- Application archetypes
- Anomaly detection

Q&A