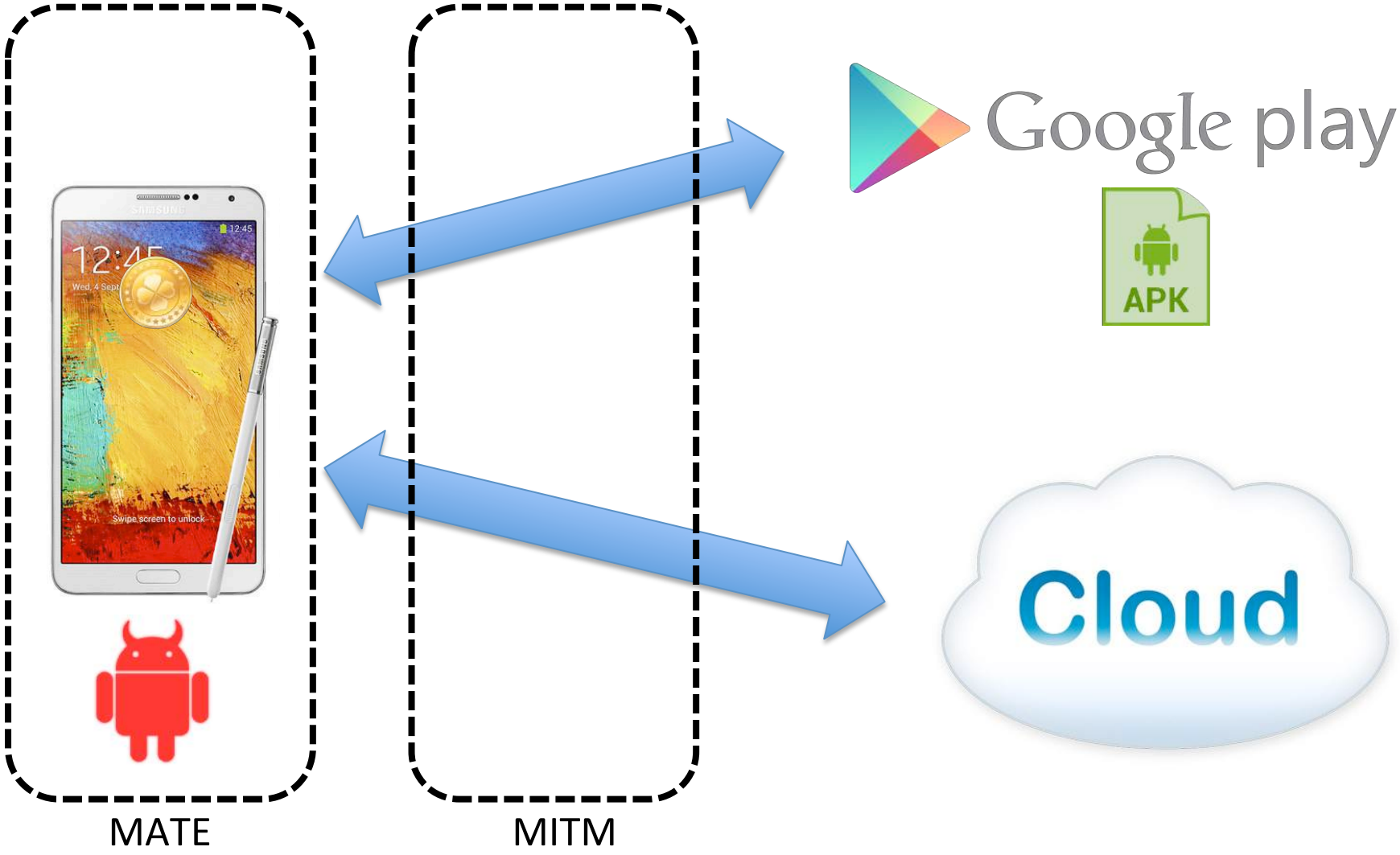
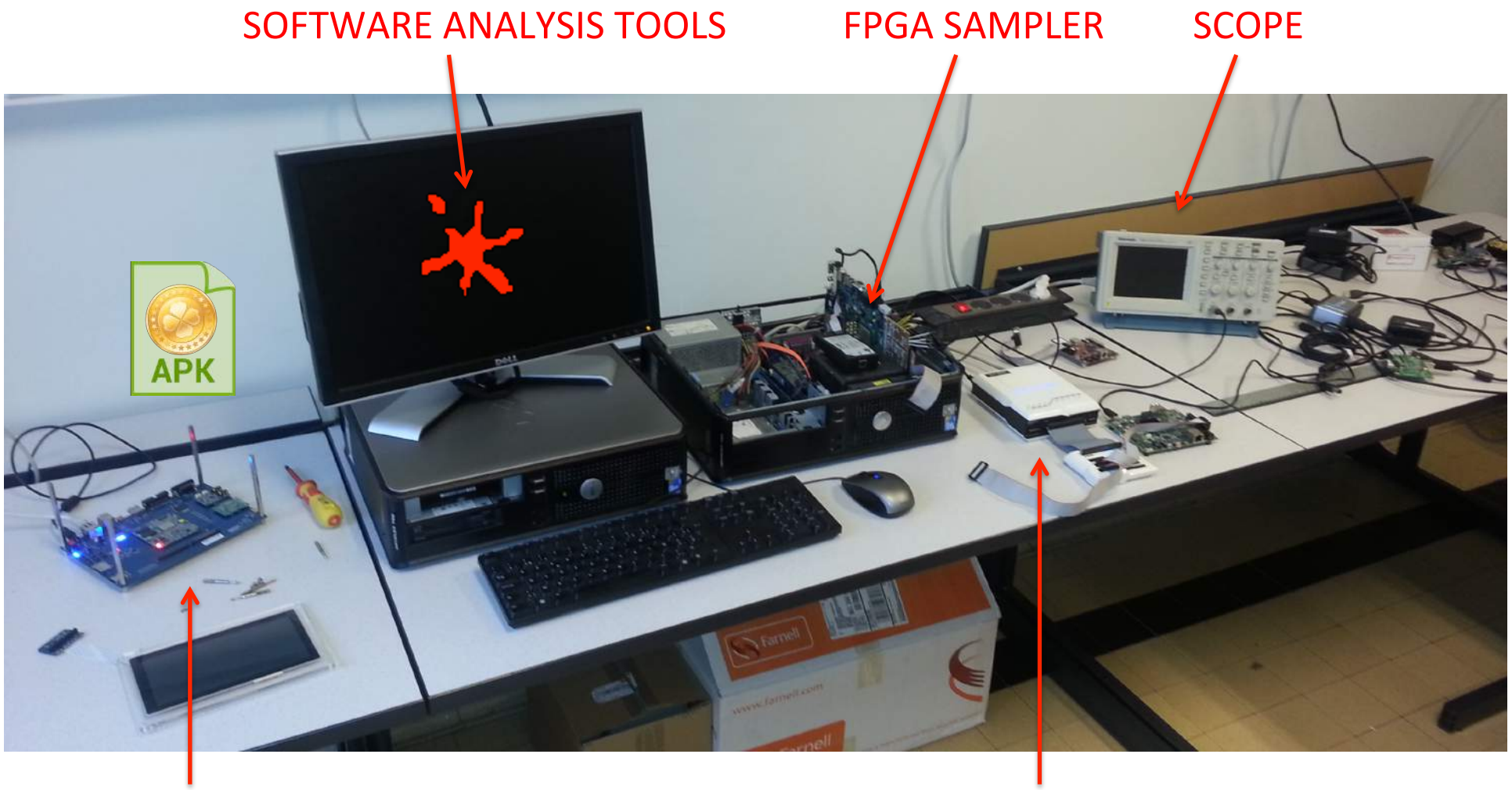
A Golden Standard for Evaluating Software Protection against Man-at-the-End Attacks

Bjorn De Sutter

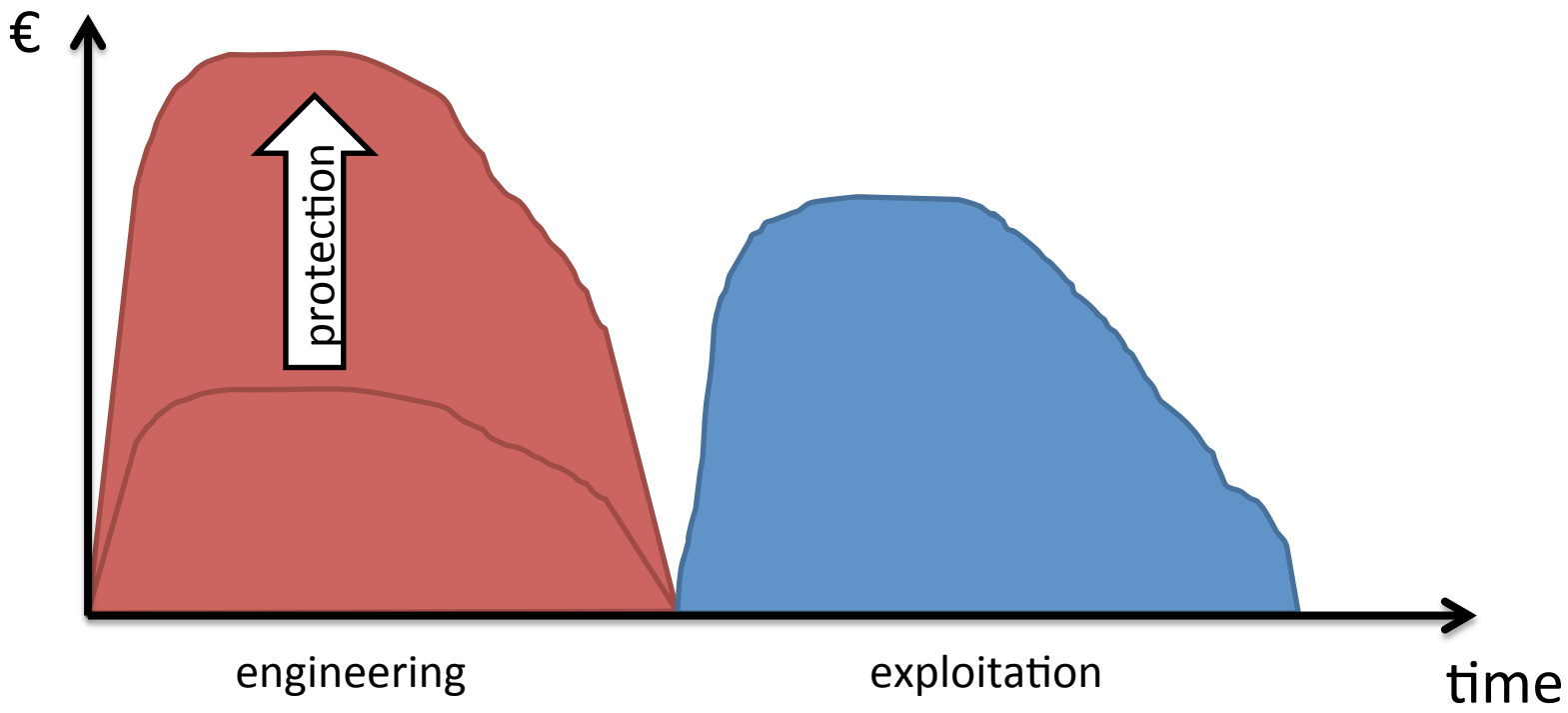
MATE attacks



MATE attacks

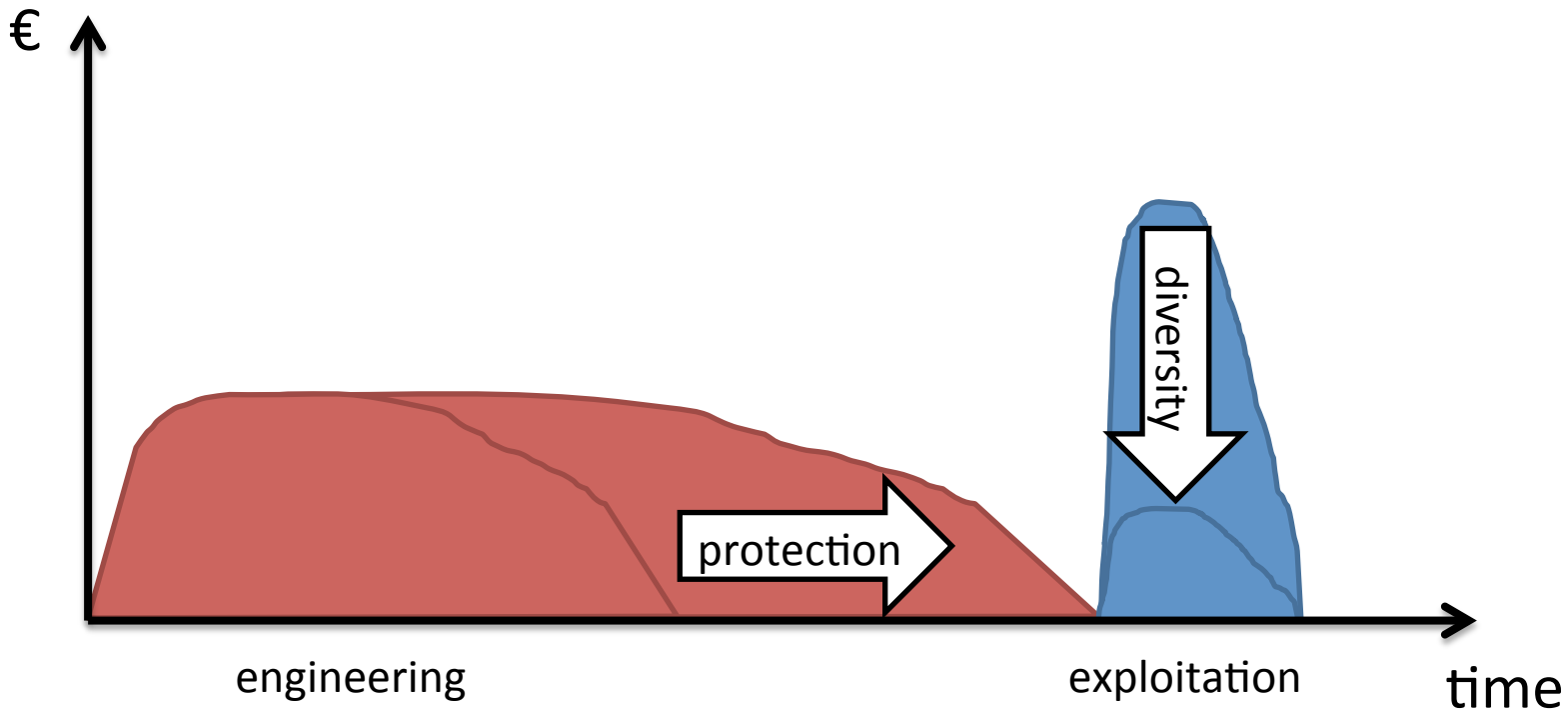


Economics of MATE attacks



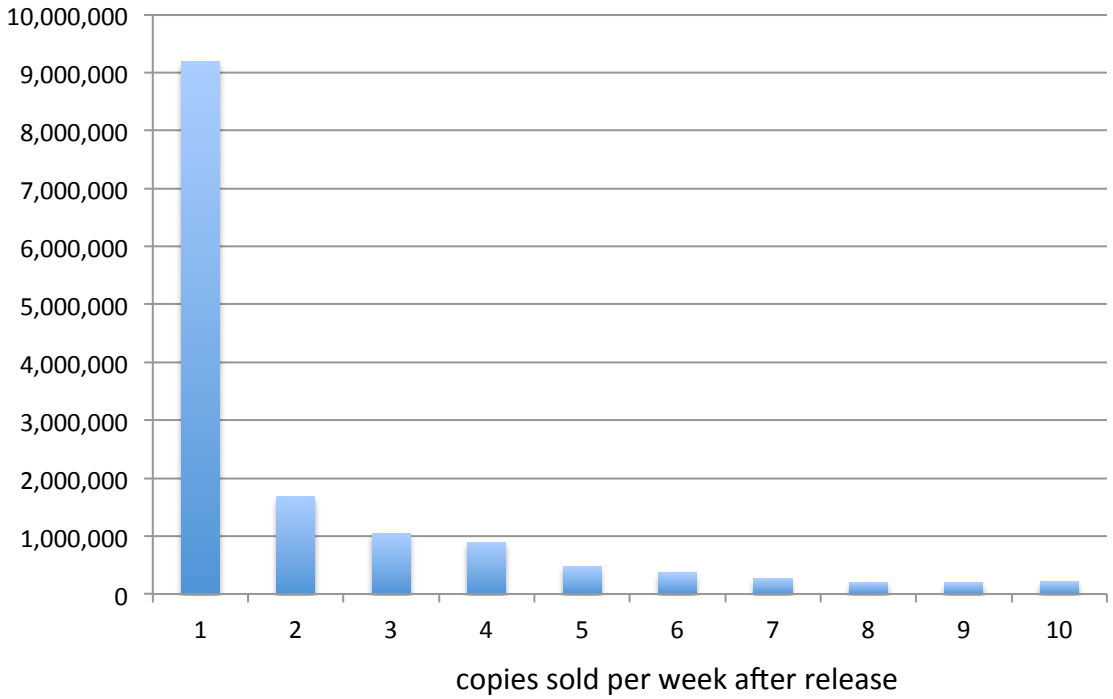
Economics of MATE attacks

What is the most appropriate protection?



Economics of MATE attacks

What is the value of protection?



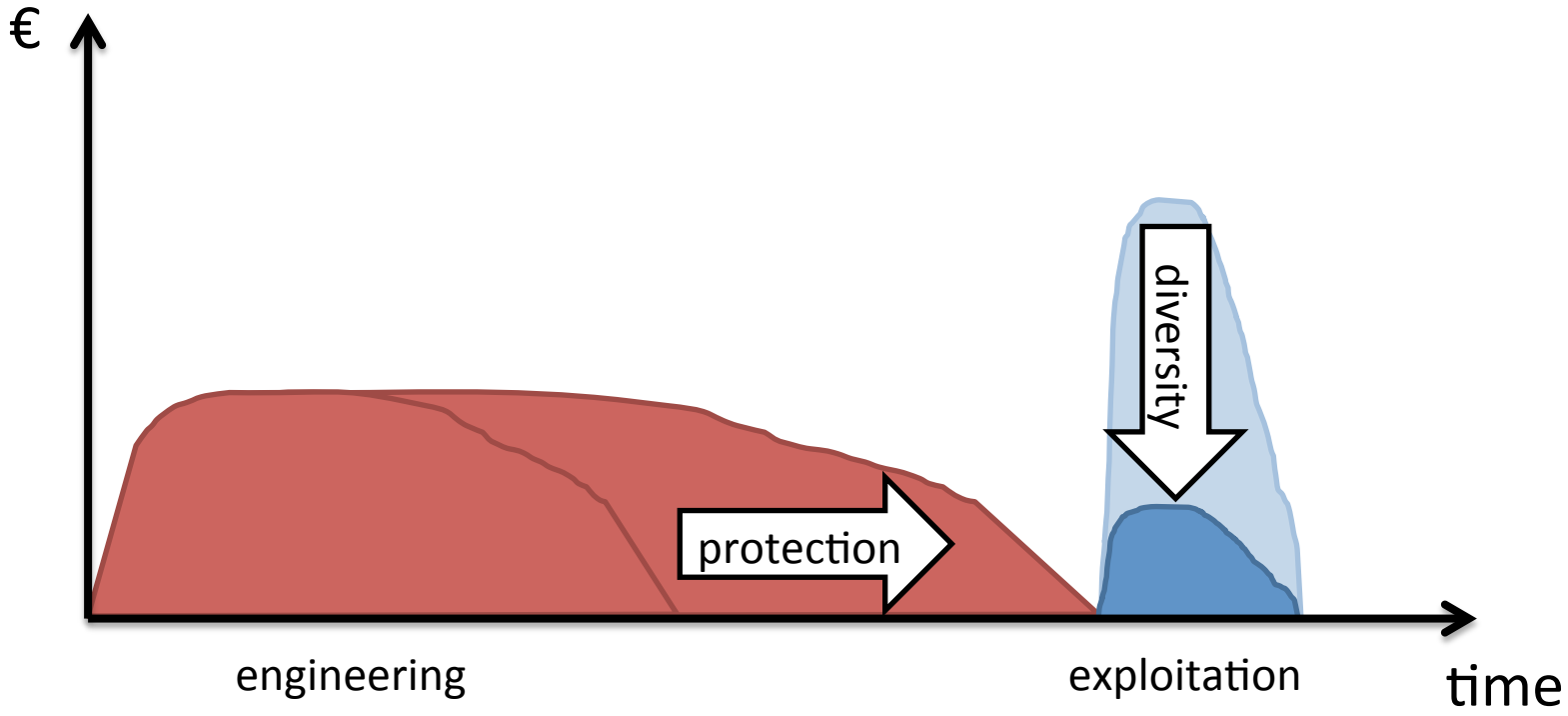
Economics of MATE attacks

What is the value of protection?

Economics of MATE attacks

~~What is the value of protection?~~

Which is the most appropriate protection and why?

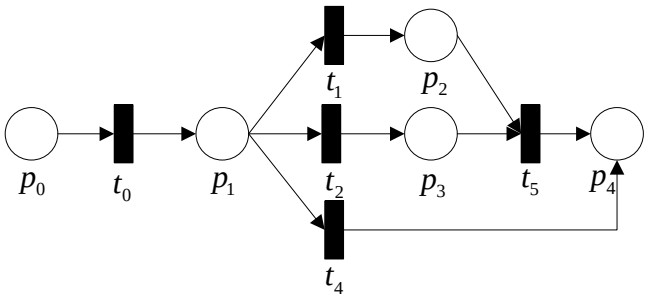


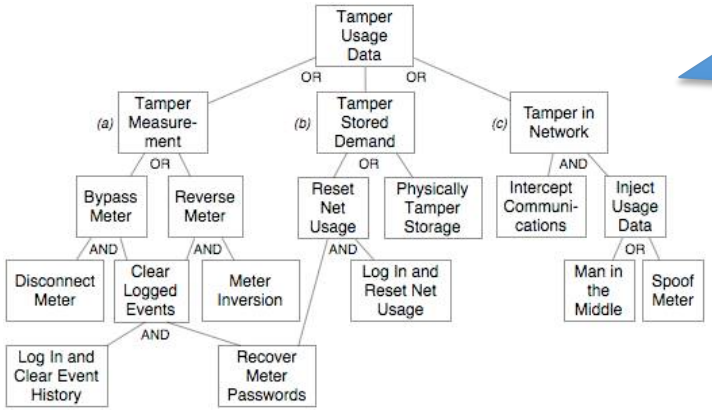
What is the most appropriate protection?

Aspire

1) model the attack paths

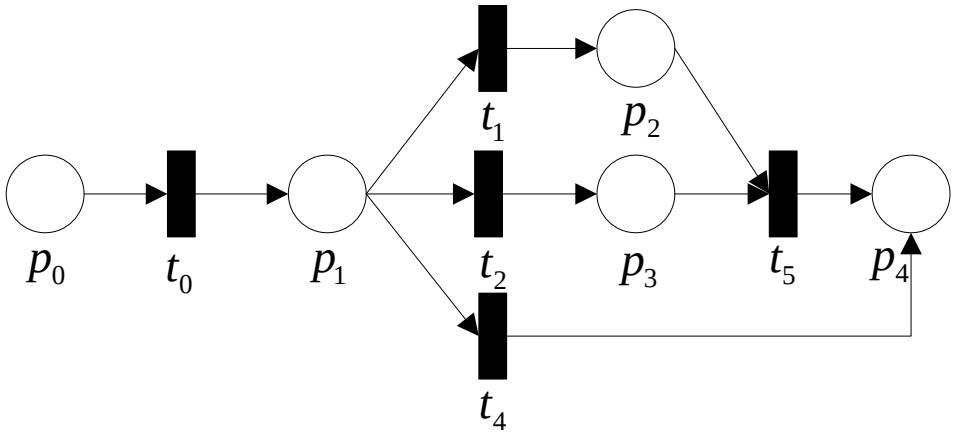
2) evaluate impact of protections





Evaluating the impact of protections

- What to evaluate?
  - potency
  - stealth
  - resilience



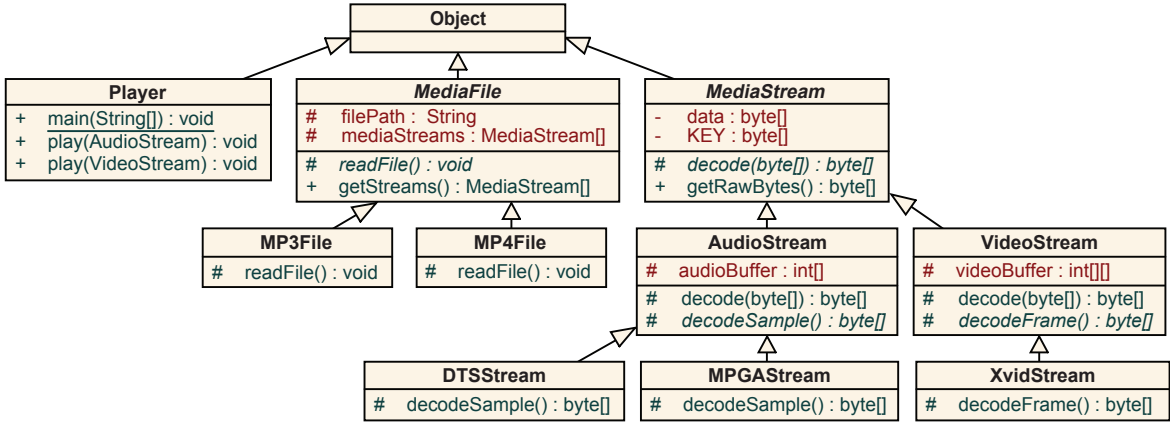
- How to measure impact?
  1. Traditional software engineering complexity metrics — LANGUAGES
  2. Software analysis complexity metrics — COMPILERS
  3. Graph and information theory — MODELS
  4. Tool-based metrics — ATTACK TOOLS
  5. Human experiments — HUMANS

1. Traditional software engineering complexity metrics

- QMOOD: Quality Model for Object-Oriented Design
  - abstraction
  - encapsulation
  - coupling
  - cohesion
  - polymorphism
  - complexity
  - design size

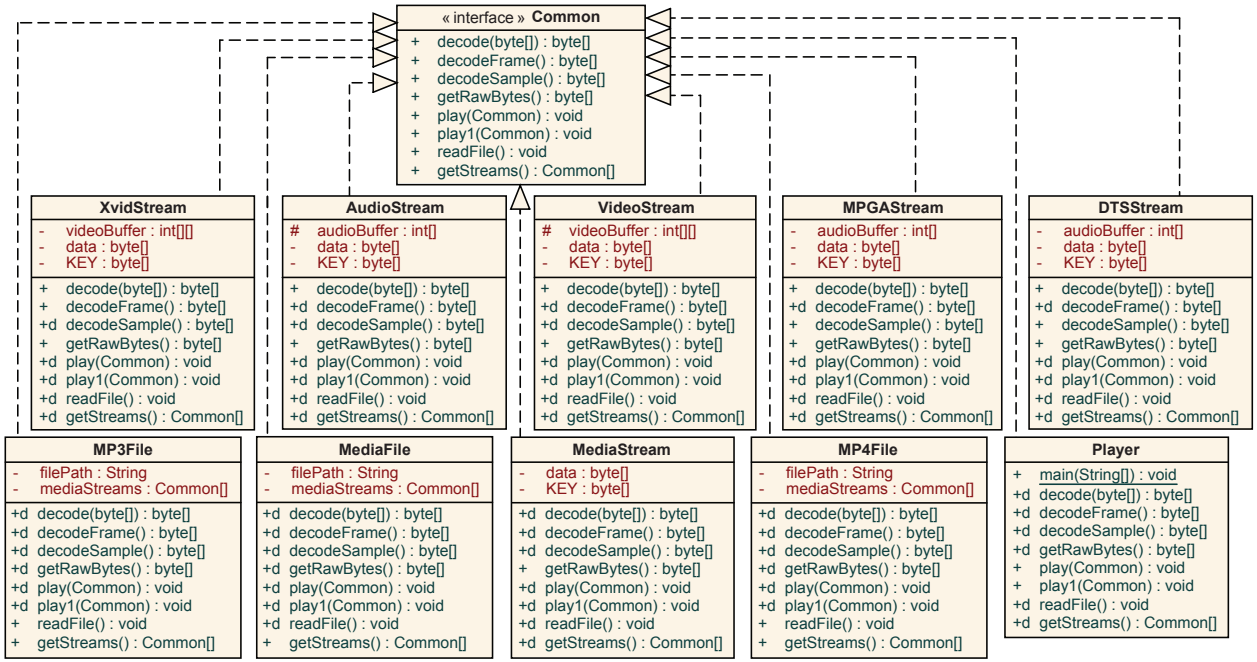
- "Understandability" is a weighted average

Example: class hierarchy flattening



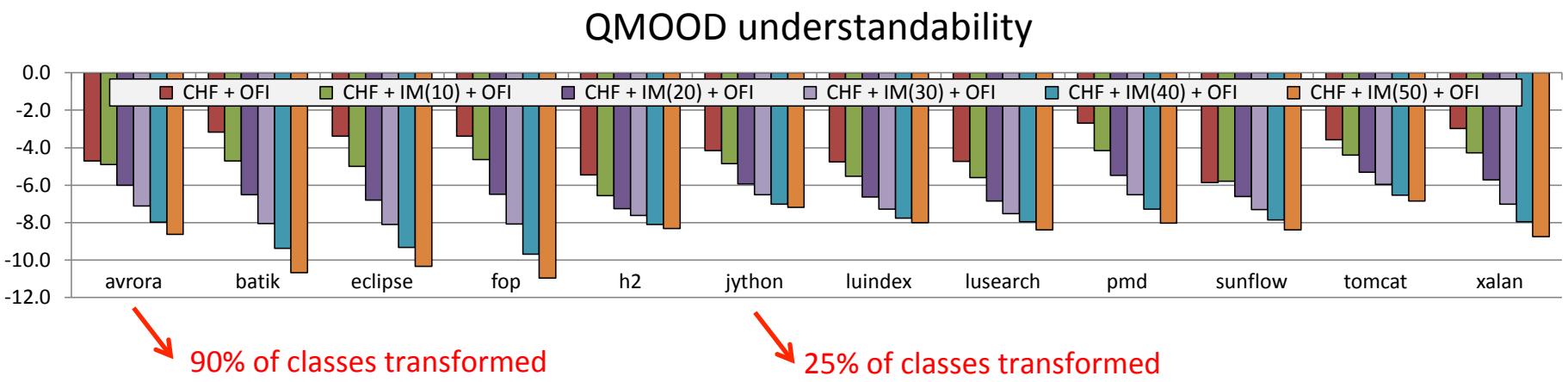

```
public class Player {
  public void play(AudioStream as) {
    /* send as.getRawBytes() to audio device */
  }
  public void play(VideoStream vs) {
    /* send vs.getRawBytes() to video device */
  }
  public static void main(String[] args) {
    Player player = new Player();
    MediaFile[] mediaFiles = ...;
    for (MediaFile mf : mediaFiles)
      for (MediaStream ms : mf.getStreams())
        if (ms instanceof AudioStream)
          player.play((AudioStream)ms);
        else if (ms instanceof VideoStream)
          player.play((VideoStream)ms);
  }
}
public class MP3File extends MediaFile {
  protected void readFile() {
    InputStream inputStream = ...;
    byte[] data = new byte[...];
    inputStream.read(data);
    AudioStream as = new MPGAStream(data);
    mediaStreams = new MediaStream[]{as};
    return;
  }
}
public abstract class MediaStream {
  public static final byte[] KEY = ...;
  public byte[] getRawBytes() {
    byte[] decrypted = new byte[data.length];
    for (int i = 0; i < data.length; i++)
      decrypted[i] = data[i] ^ KEY[i];
    return decode(decrypted);
  }
  protected abstract byte[] decode(byte[] data);
}
```

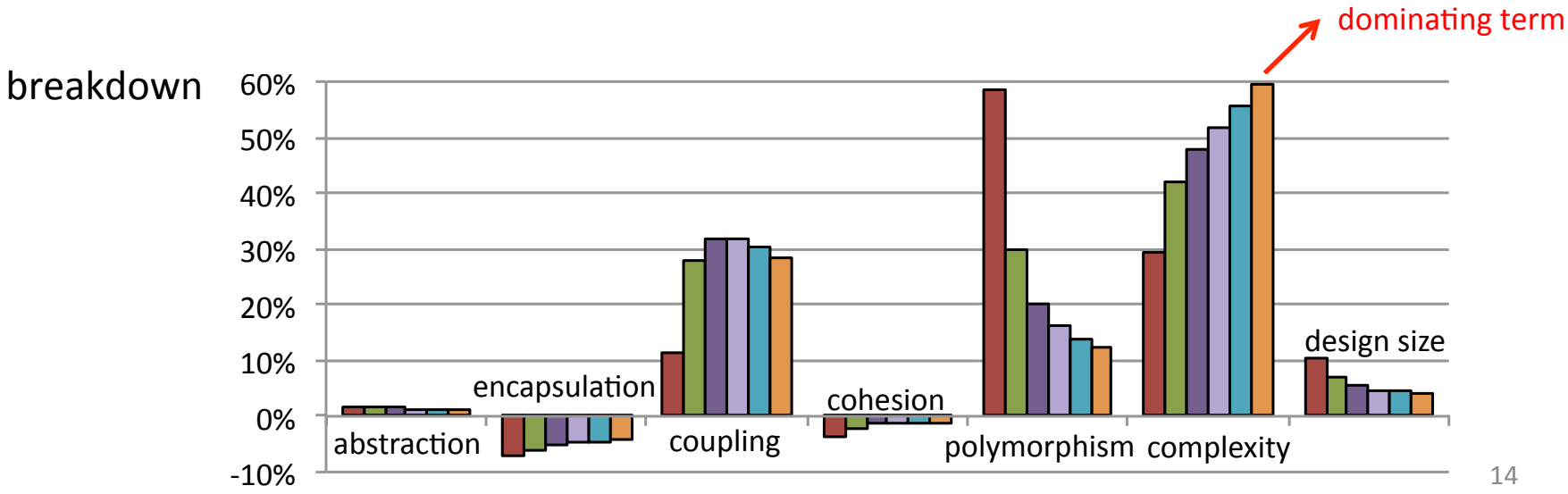
Example: class hierarchy flattening




```
public class Player implements Common {
  public byte[] merged1(Common as) {
    /* send as.getRawBytes() to audio device */
  }
  public Common[] merged2(Common vs) {
    /* send vs.getRawBytes() to video device */
  }
  public static void main(String[] args) {
    Common player = CommonFactory.create(…);
    Common[] mediaFiles = ...;
    for (Common mf : mediaFiles)
      for (Common ms : mf.getStreams())
        if (myCheck.isInst(0, ms.getClass()))
          player.merged1(ms);
        else if (myCheck.isInst(1, ms.getClass()))
          player.merged2(ms);
  }
}
public class MP3File implements Common {
  public byte[] merged1() {
    InputStream inputStream = ...;
    byte[] data = new byte[...];
    inputStream.read(data);
    Common as = CommonFactory.create(…);
    mediaStreams = new Common[]{as};
    return data;
  }
}
public class MediaStream implements Common {
  public static final byte[] KEY = ...;
  public byte[] getRawBytes() {
    byte[] decrypted = new byte[data.length];
    for (int i = 0; i < data.length; i++)
      decrypted[i] = data[i] ^ KEY[i];
    return decode(decrypted);
  }
  public byte[] decode(byte[] data){ … }
}
```

Result of class hierarchy flattening

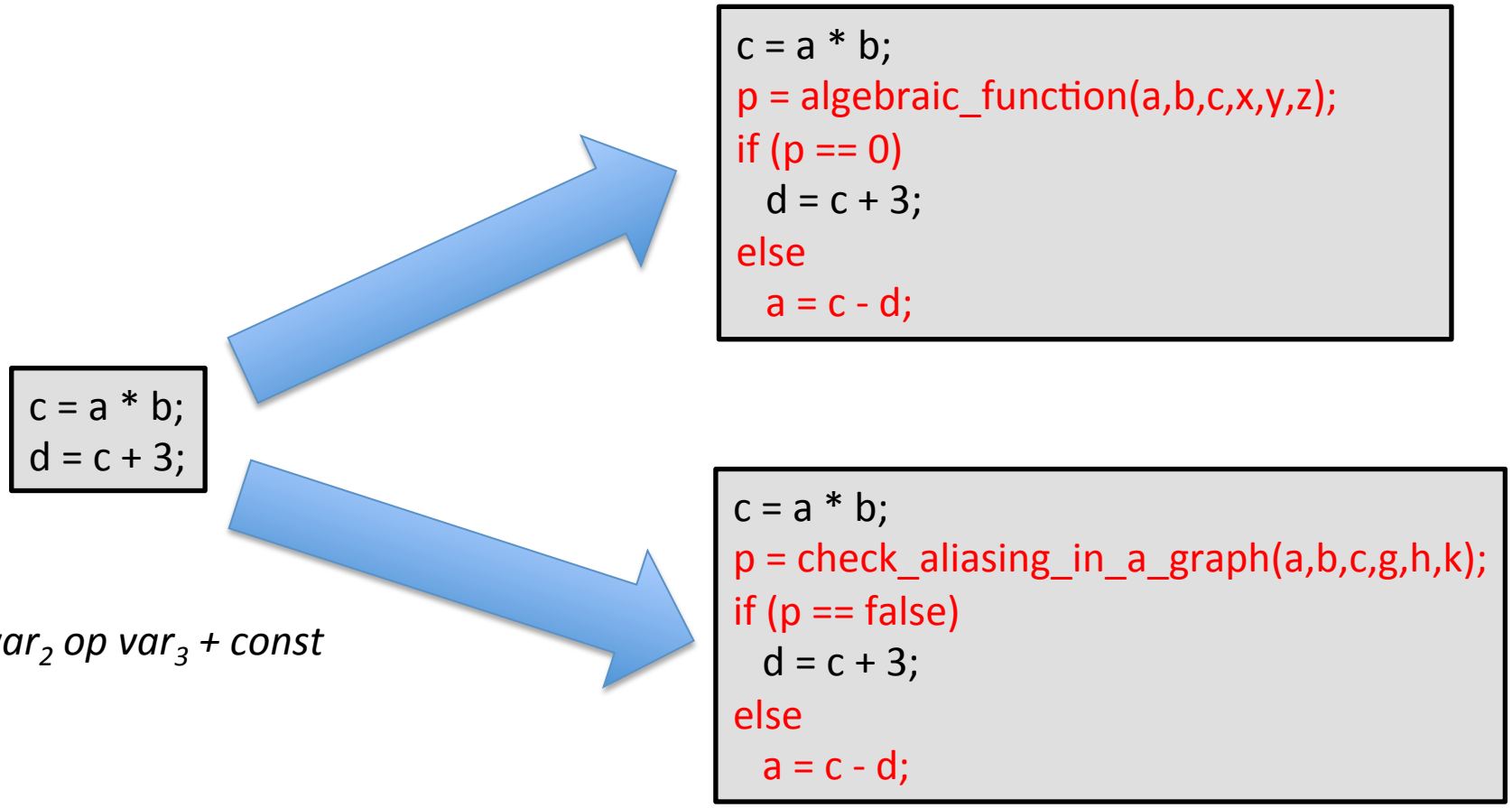




2. Software analysis complexity metrics

- Abstract interpretation
  - Abstract domains model program properties
  - Domains are partially ordered in terms of concreteness
  - Obfuscating transformation is less potent if it preserves more concrete properties

Example: opaque predicates

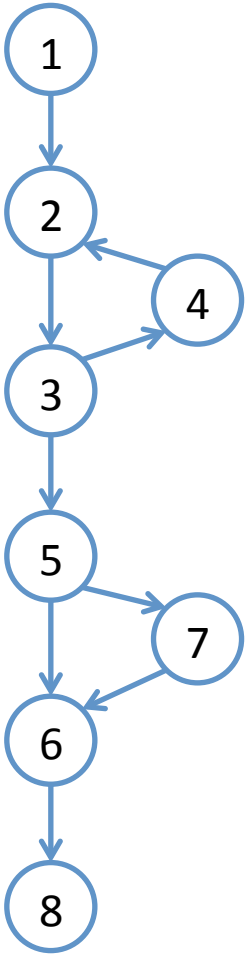


3. Graph and information theory

- Cyclomatic complexity:

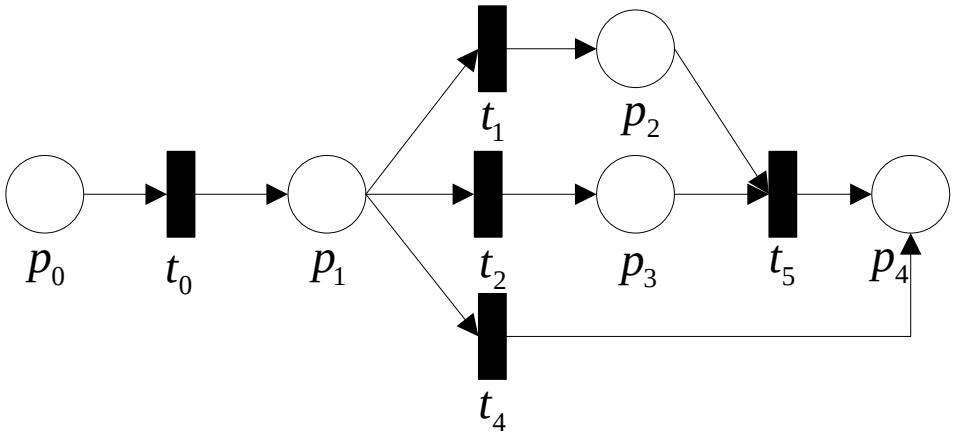
  #edges − #nodes + 2 #connected components

- Entropy
  - on static graphs
  - using a range of models
  - on dynamic graphs
  - on program traces ...$(234)^{10}$...$(234)^{10}$...$(234)^{10}$ ...
  - on variable names (?)

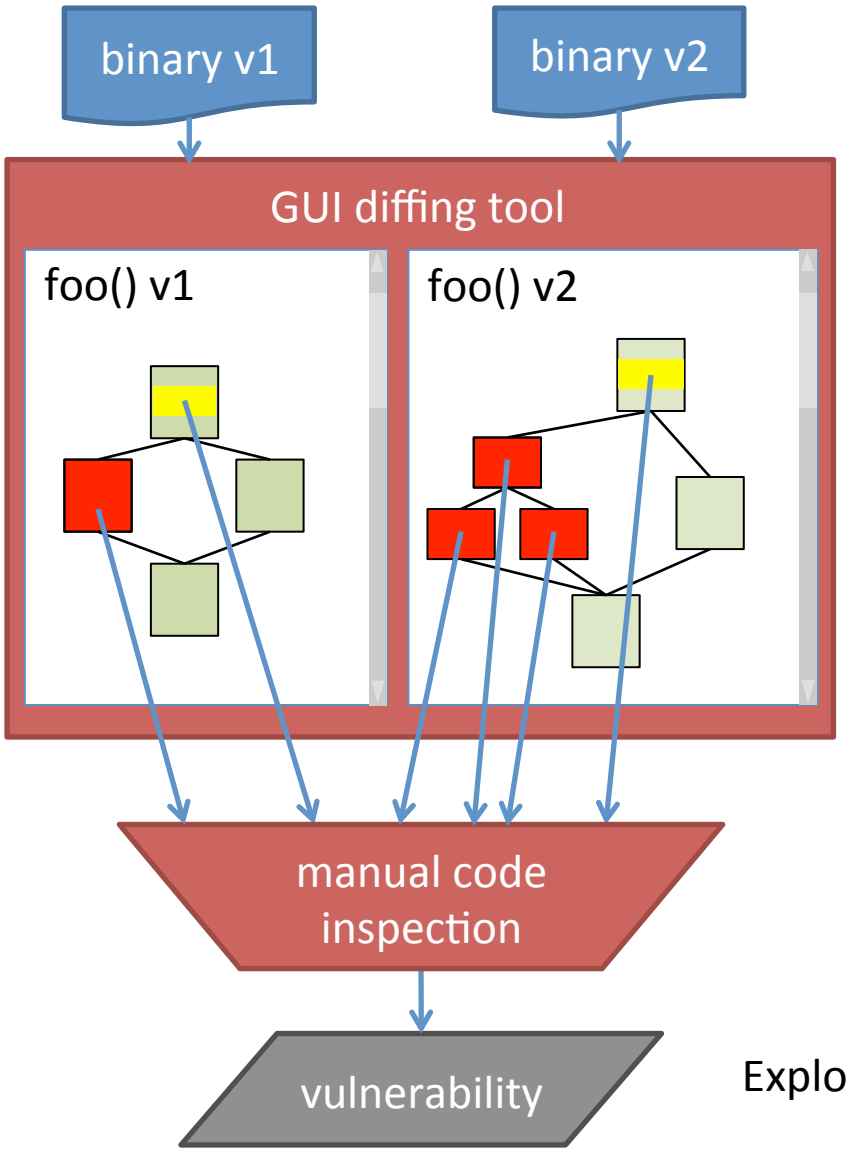


4. Tool-based metrics

- Attacker's tools, heuristics and mental models
  1. Model effort/time in terms of input size
  2. Compute output size
  3. Compute relevance of output
     - false positives/negatives
     - receiver operator curves (ROC)
     - recall and precision
     - pruning factors

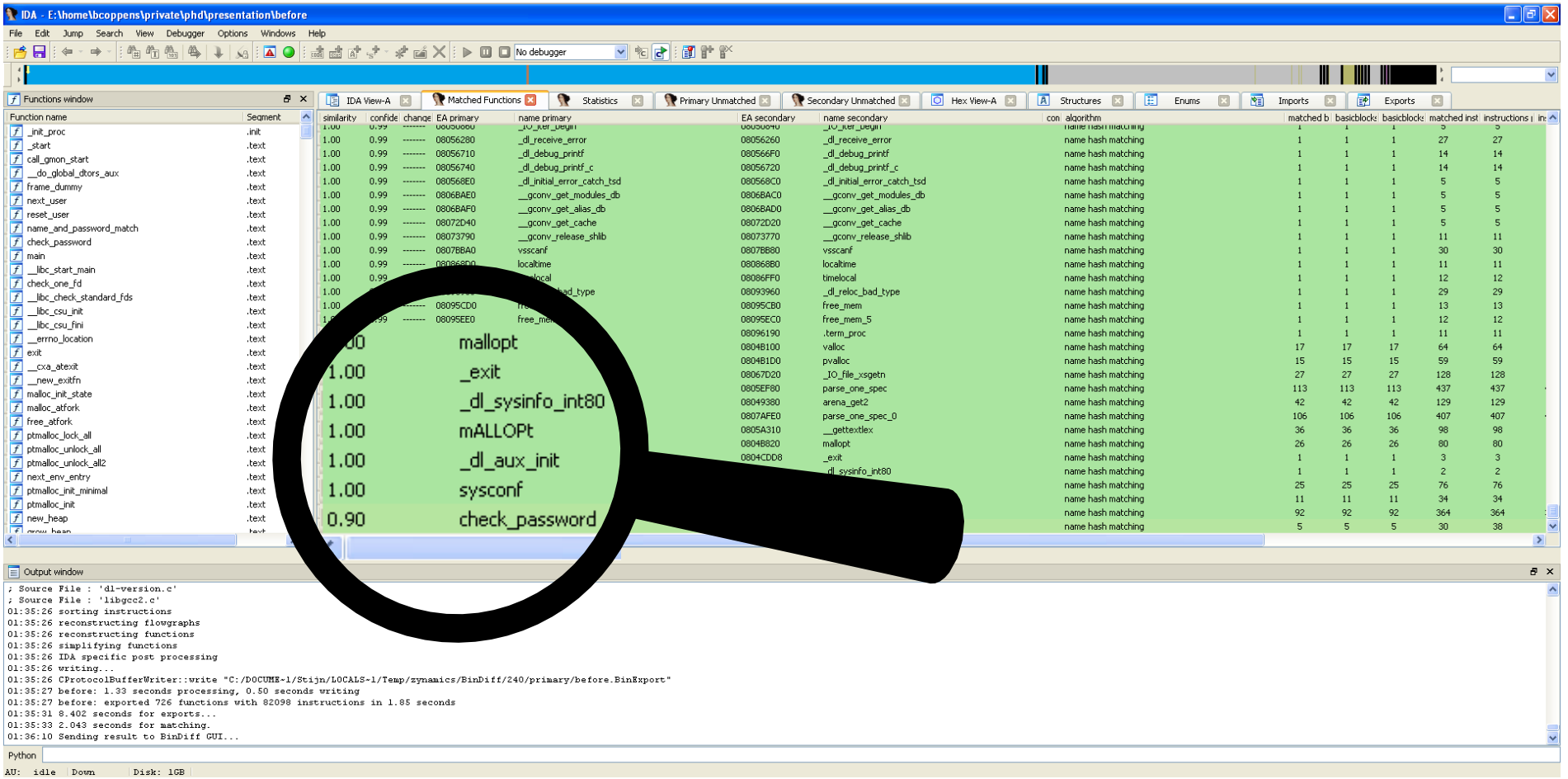


Example: Patch Tuesday

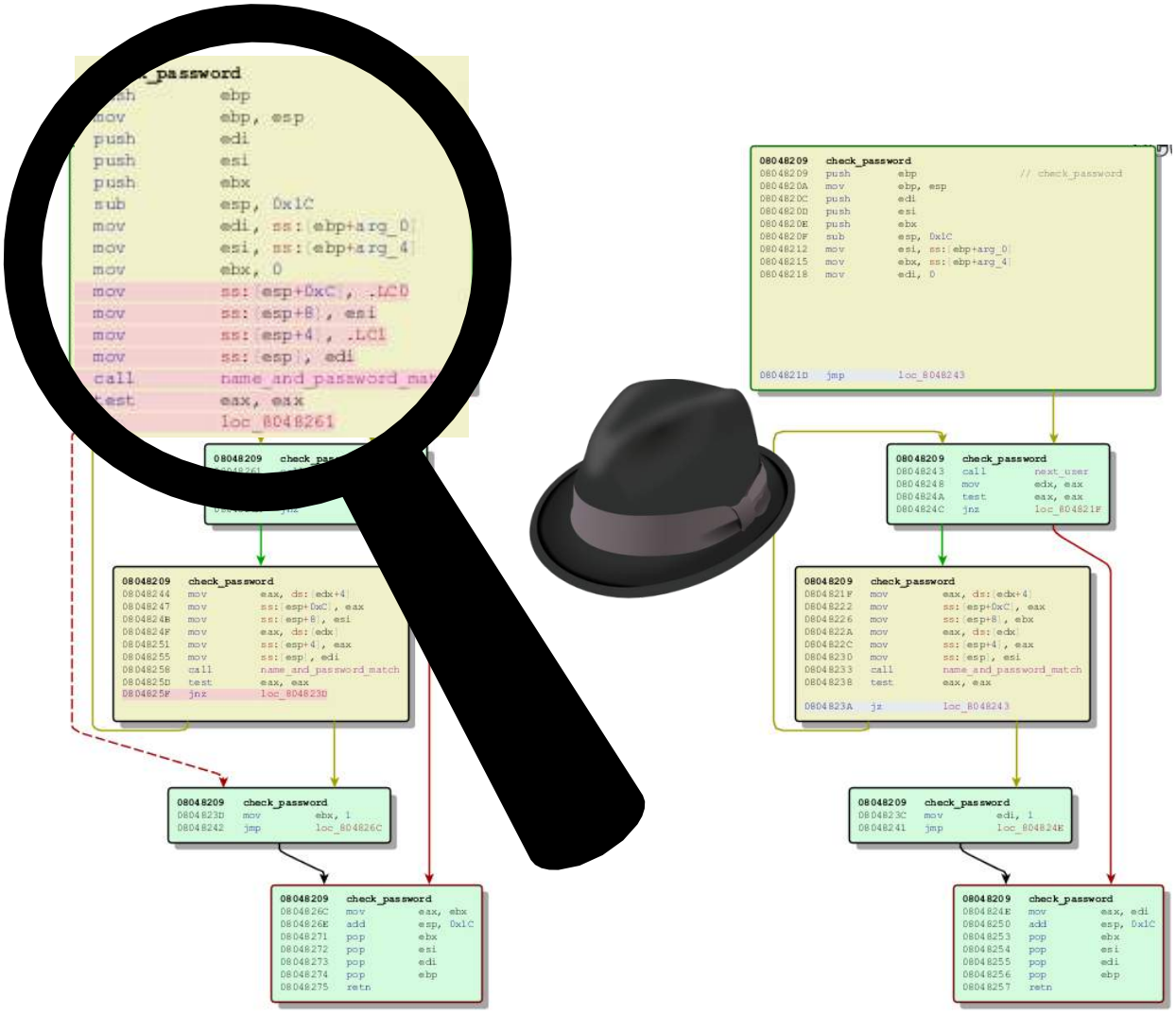


Exploit Wednesday

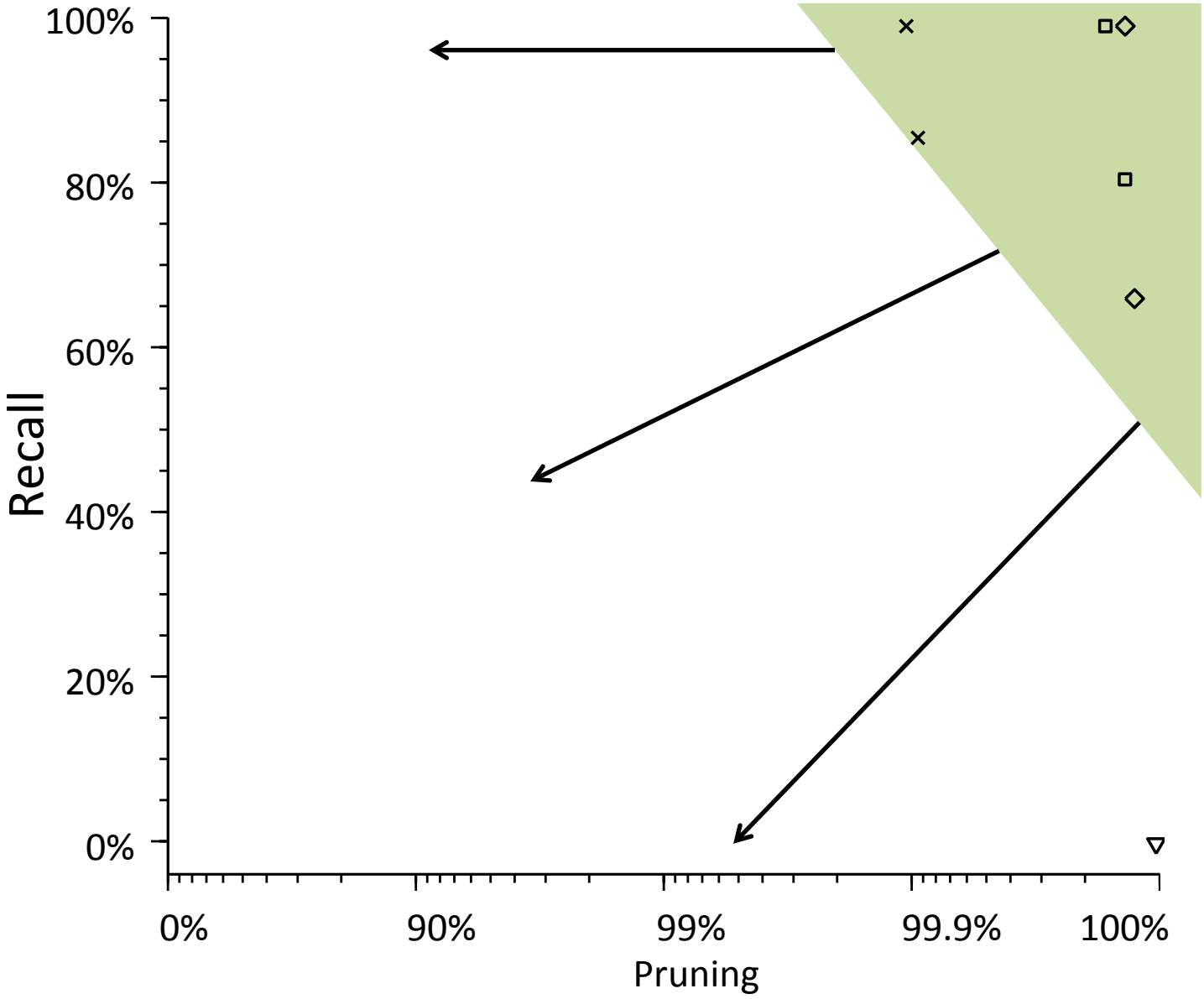
BinDiff on Patch Tuesday



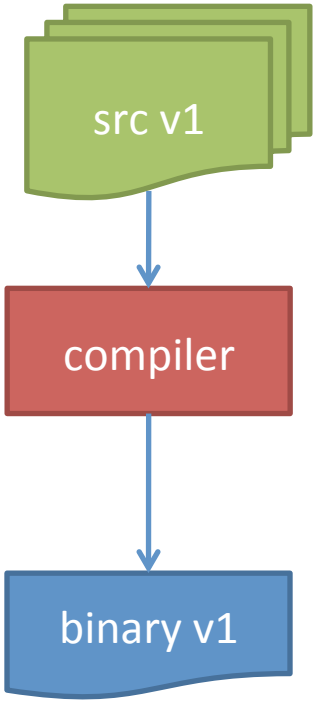
BinDiff on Patch Tuesday

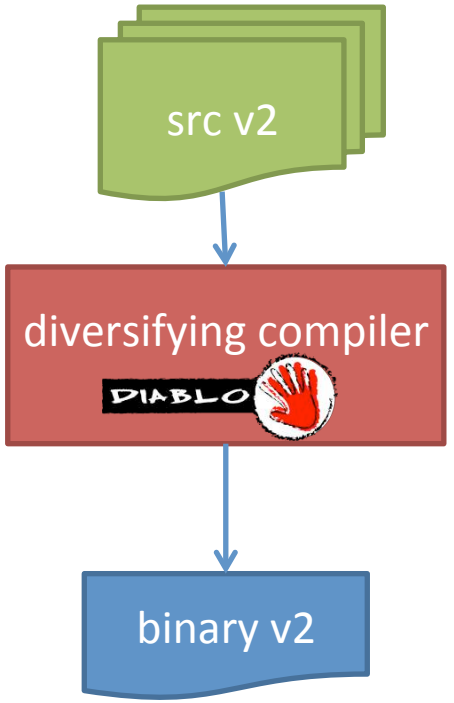


BinDiff on Patch Tuesday

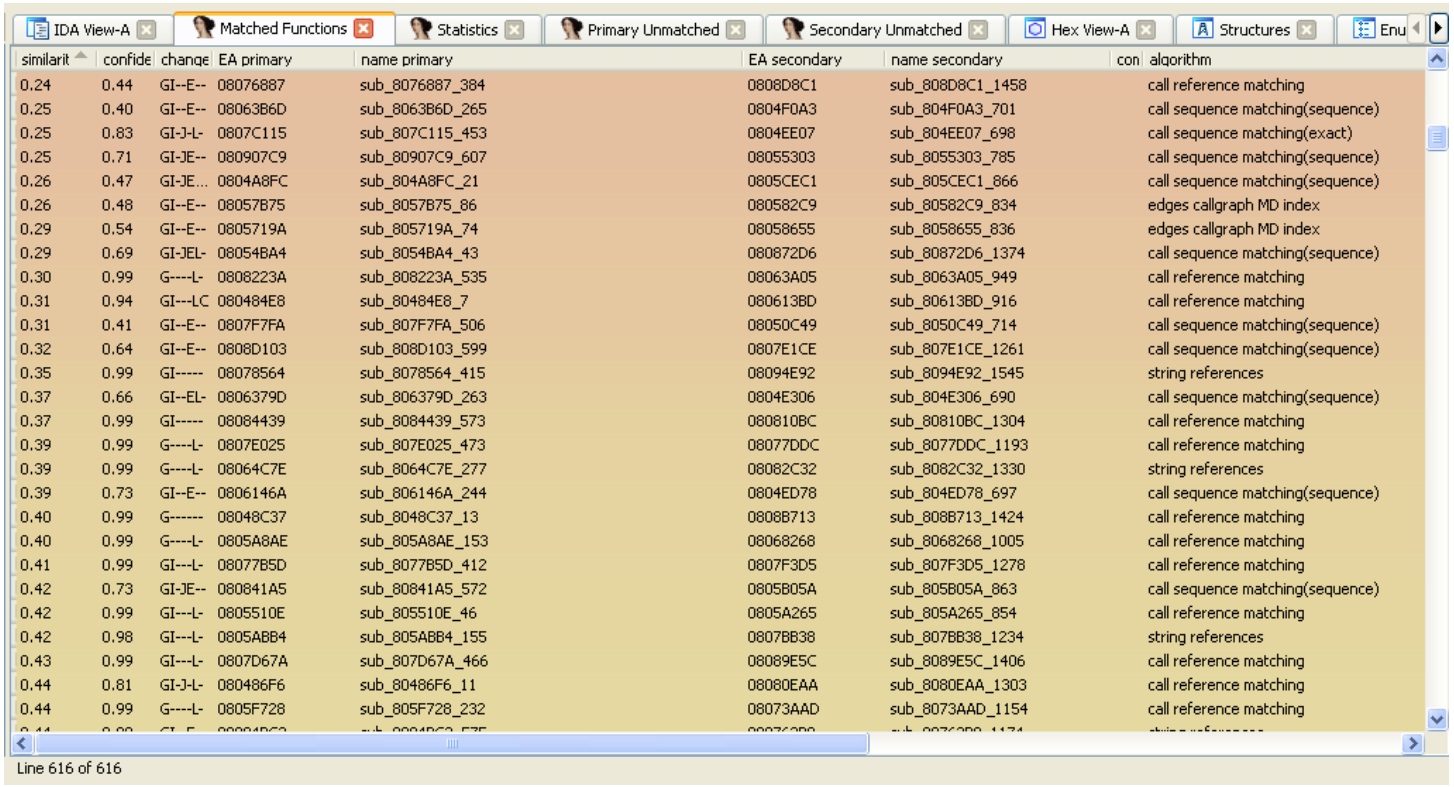


Software Diversification



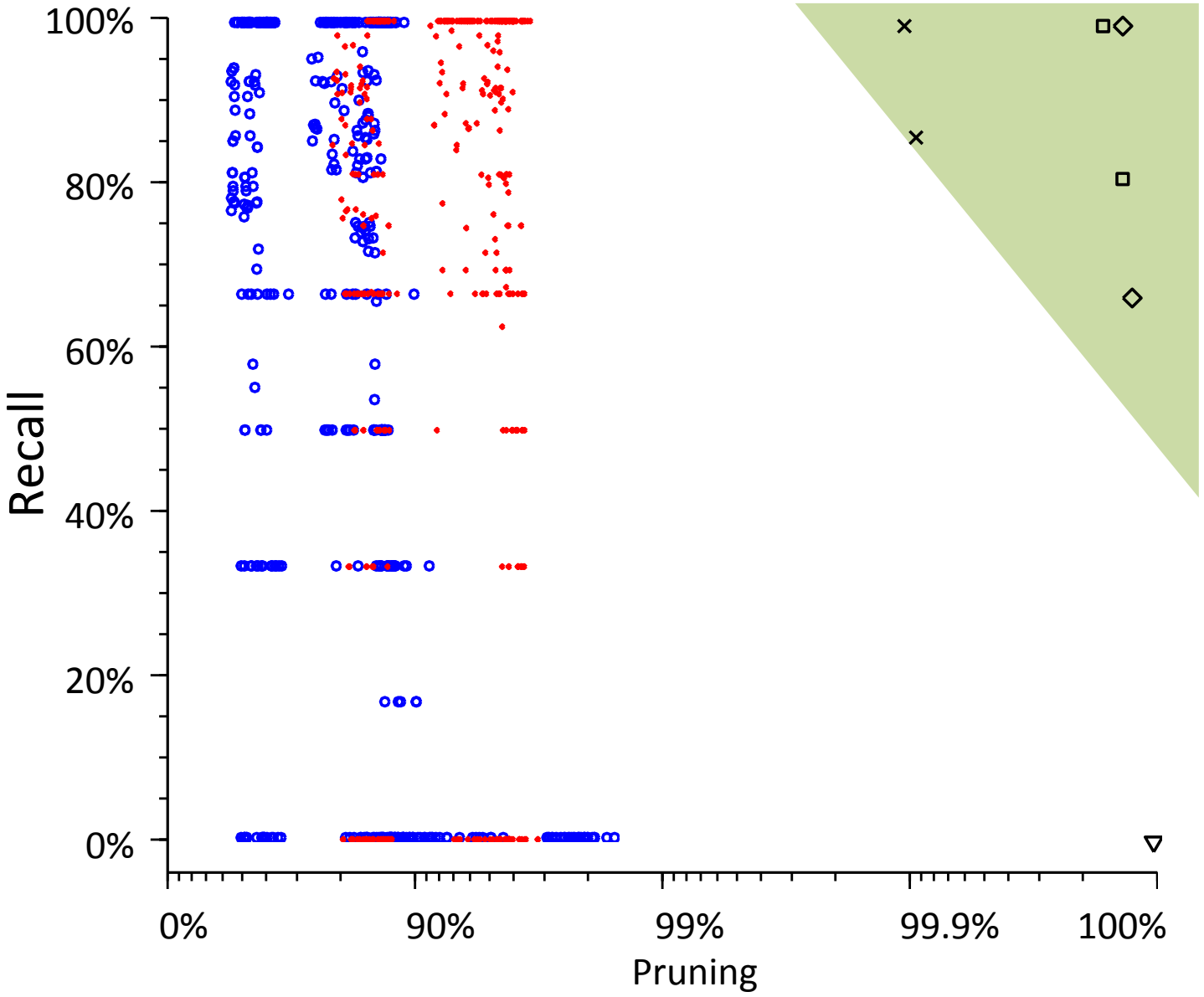


Bindiff on Patch Tuesday



similarit	confide	change	EA primary	name primary	EA secondary	name secondary	con	algorithm
0.24	0.44	GI--E--	08076887	sub_8076887_384	0808D8C1	sub_808D8C1_1458		call reference matching
0.25	0.40	GI--E--	08063B6D	sub_8063B6D_265	0804F0A3	sub_804F0A3_701		call sequence matching(sequence)
0.25	0.83	GI-J-L-	0807C115	sub_807C115_453	0804EE07	sub_804EE07_698		call sequence matching(exact)
0.25	0.71	GI-JE--	080907C9	sub_80907C9_607	08055303	sub_8055303_785		call sequence matching(sequence)
0.26	0.47	GI-JE...	0804A8FC	sub_804A8FC_21	0805CEC1	sub_805CEC1_866		call sequence matching(sequence)
0.26	0.48	GI--E--	08057B75	sub_8057B75_86	080582C9	sub_80582C9_834		edges callgraph MD index
0.29	0.54	GI--E--	0805719A	sub_805719A_74	08058655	sub_8058655_836		edges callgraph MD index
0.29	0.69	GI-JEL-	08054BA4	sub_8054BA4_43	080872D6	sub_80872D6_1374		call sequence matching(sequence)
0.30	0.99	G----L-	0808223A	sub_808223A_535	08063A05	sub_8063A05_949		call reference matching
0.31	0.94	GI---LC	080484E8	sub_80484E8_7	080613BD	sub_80613BD_916		call reference matching
0.31	0.41	GI--E--	0807F7FA	sub_807F7FA_506	08050C49	sub_8050C49_714		call sequence matching(sequence)
0.32	0.64	GI--E--	0808D103	sub_808D103_599	0807E1CE	sub_807E1CE_1261		call sequence matching(sequence)
0.35	0.99	GI-----	08078564	sub_8078564_415	08094E92	sub_8094E92_1545		string references
0.37	0.66	GI--EL-	0806379D	sub_806379D_263	0804E306	sub_804E306_690		call sequence matching(sequence)
0.37	0.99	GI-----	08084439	sub_8084439_573	080810BC	sub_80810BC_1304		call reference matching
0.39	0.99	G----L-	0807E025	sub_807E025_473	08077DDC	sub_8077DDC_1193		call reference matching
0.39	0.99	G----L-	08064C7E	sub_8064C7E_277	08082C32	sub_8082C32_1330		string references
0.39	0.73	GI--E--	0806146A	sub_806146A_244	0804ED78	sub_804ED78_697		call sequence matching(sequence)
0.40	0.99	G------	08048C37	sub_8048C37_13	0808B713	sub_808B713_1424		call reference matching
0.40	0.99	G----L-	0805A8AE	sub_805A8AE_153	08068268	sub_8068268_1005		call reference matching
0.41	0.99	GI---L-	08077B5D	sub_8077B5D_412	0807F3D5	sub_807F3D5_1278		call reference matching
0.42	0.73	GI-JE--	080841A5	sub_80841A5_572	0805B05A	sub_805B05A_863		call sequence matching(sequence)
0.42	0.99	GI---L-	0805510E	sub_805510E_46	0805A265	sub_805A265_854		call reference matching
0.42	0.98	GI---L-	0805ABB4	sub_805ABB4_155	0807BB38	sub_807BB38_1234		string references
0.43	0.99	GI---L-	0807D67A	sub_807D67A_466	08089E5C	sub_8089E5C_1406		call reference matching
0.44	0.81	GI-J-L-	080486F6	sub_80486F6_11	08080EAA	sub_8080EAA_1303		call reference matching
0.44	0.99	G----L-	0805F728	sub_805F728_232	08073AAD	sub_8073AAD_1154		call reference matching

BinDiff on Diversified Code



5. Human experiments

- Absolutely necessary
- Difficult to set up
  - expensive
  - or without experts

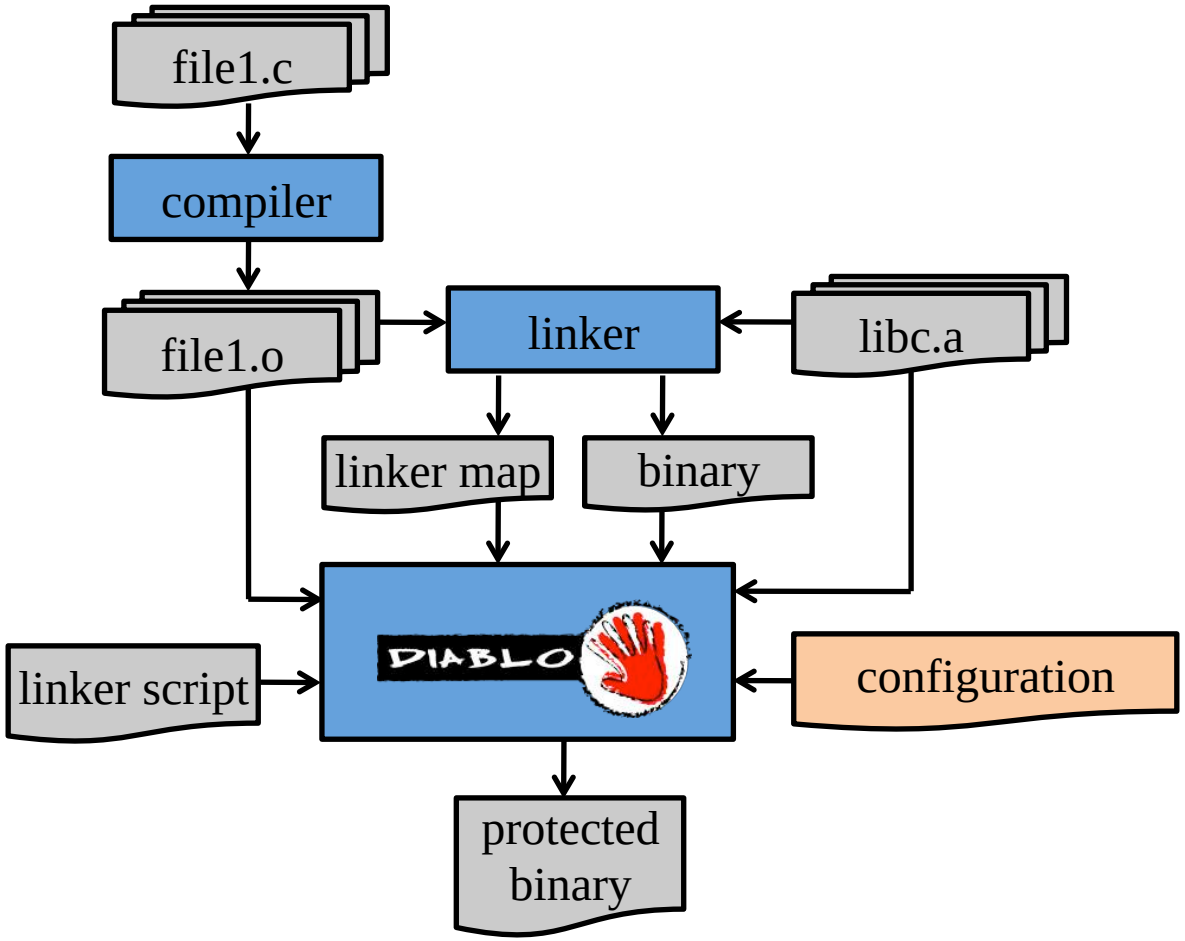
Open challenges

1. Comprehensive attack model
2. Automatic instantiation for application + assets
3. Model the relation between attack steps and protections
4. How to choose the best set of protections?
   - from possibly a very large set
   - applicable at many places in the code
   - tool-based metrics require running the tools
   - iterative methods or machine learning
5. How to specify the available protections?

Aspire

www.aspire-fp7.eu

Link-time smart card code hardening

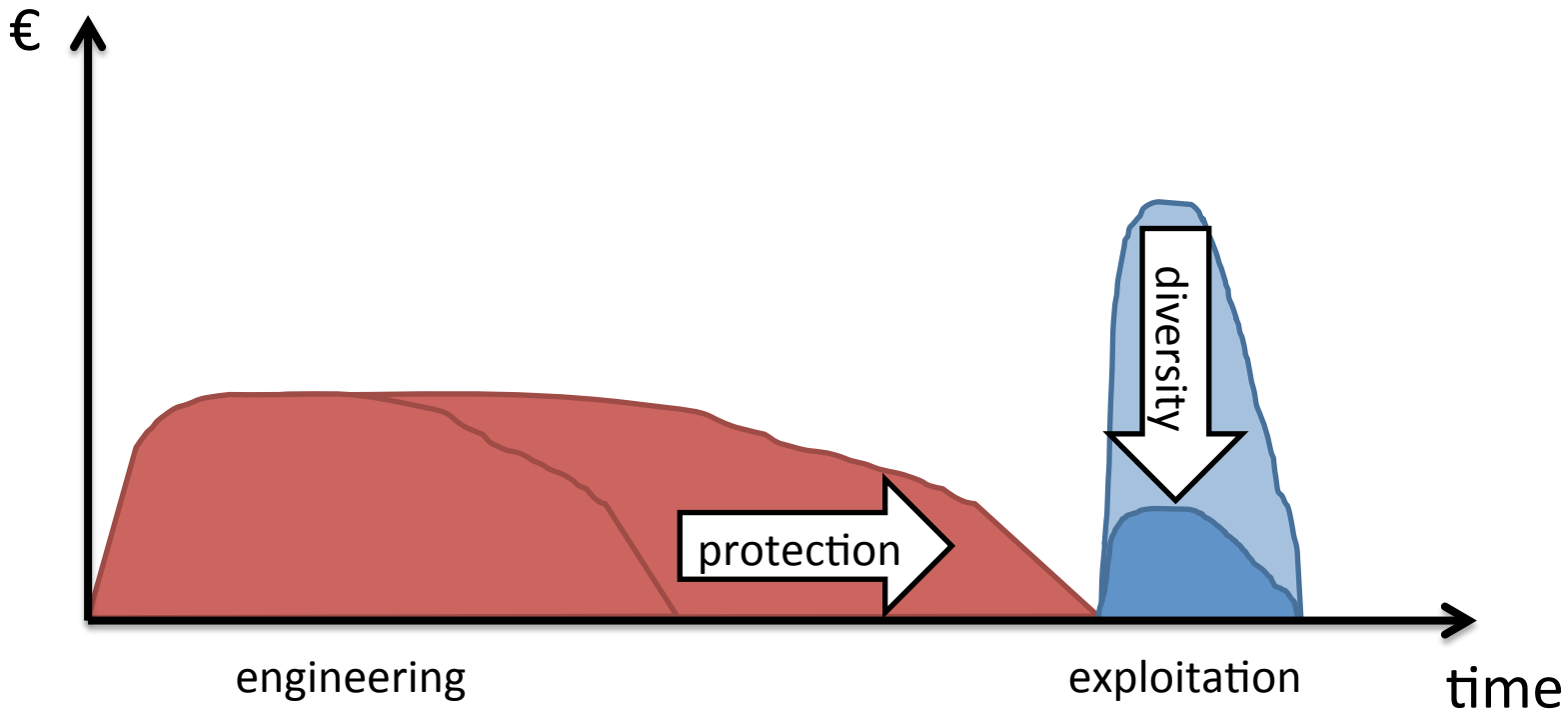


Open challenges

1. Comprehensive attack model
2. Automatic instantiation for application + assets
3. Model the relation between attack steps and protections
4. How to choose the best set of protections?
   - from possibly a very large set
   - applicable at many places in the code
   - tool-based metrics require running the tools
   - or machine learning

5. How to specify what protections are available?
6. Security through obscurity?

Economics of MATE attacks



White-box cryptography

embed the secret key in haystack of code and data

published (Cloakware)

obscure

Security through obscurity

- Definitely useful
- But what about
  - certification?
  - customer demands & expectations?
  - back doors?
  - leakage?
  - academic research?

Open challenges

1. Comprehensive attack model
2. Automatic instantiation for application + assets
3. Model the relation between attack steps and protections
4. How to choose the best set of protections?
5. How to specify what protections are available?
6. Security through obscurity?

Aspire Grant Agreement No 609734

The Aspire project has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement number 609734.

If you need further information, please contact the coordinator:

Bjorn De Sutter, Ghent University
Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium
Tel: +32 9 264 33 67 Fax: +32 9 264 35 94
Email: coordinator@aspire-fp7.eu
Website: http://www.aspire-fp7.eu

References

Jagdish Bansiya and Carl G. Davis. 2002. A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Trans. Softw. Eng. 28, 1 (January 2002), 4-17. DOI=10.1109/32.979986 http://dx.doi.org/10.1109/32.979986

Jonas Maebe, Ronald De Keulenaer, Bjorn De Sutter, Koen De Bosschere: Mitigating Smart Card Fault Injection with Link-Time Code Rewriting: A Feasibility Study. Financial Cryptography 2013: 221-229

Bart Coppens, Bjorn De Sutter, Koen De Bosschere: Protecting Your Software Updates. IEEE Security & Privacy 11(2): 47-54 (2013)

Bart Coppens, Bjorn De Sutter, Jonas Maebe: Feedback-driven binary code diversification. ACM TACO 9(4): 24 (2013)

Roberto Giacobazzi and Andrea Toppan: On Entropy Measures for Code Obfuscation. Proceedings of the ACM SIGPLAN Software Security and Protection Workshop 2012

McCabe, T.J. A Complexity Measure. IEEE Transactions on Software Engineering, vol.SE-2, no.4, pp.308-320, Dec. 1976 doi: 10.1109/TSE.1976.233837

M. Dalla Preda. Code Obfuscation and Malware Detection by Abstract Interpretation. Dipartimento di Informatica, Universita' di Verona, TD-02-07, 2007.

http://www.vgchartz.com/

Christophe Foket, Bjorn De Sutter, Koen De Bosschere. Pushing Java Type Obfuscation to the Limit. To appear in IEEE Trans. on Dependable en Secure Computing.