

How to Grow a TREE from CBASS

***Interactive Binary Analysis for
Security Professionals***

Lixin (Nathan) Li, Xing Li, Loc Nguyen,
James E. Just

Outline

- **Background**
- **Interactive Binary Analysis with TREE and CBASS**
- **Demonstrations**
- **Conclusions**

Interactive Binary Analysis

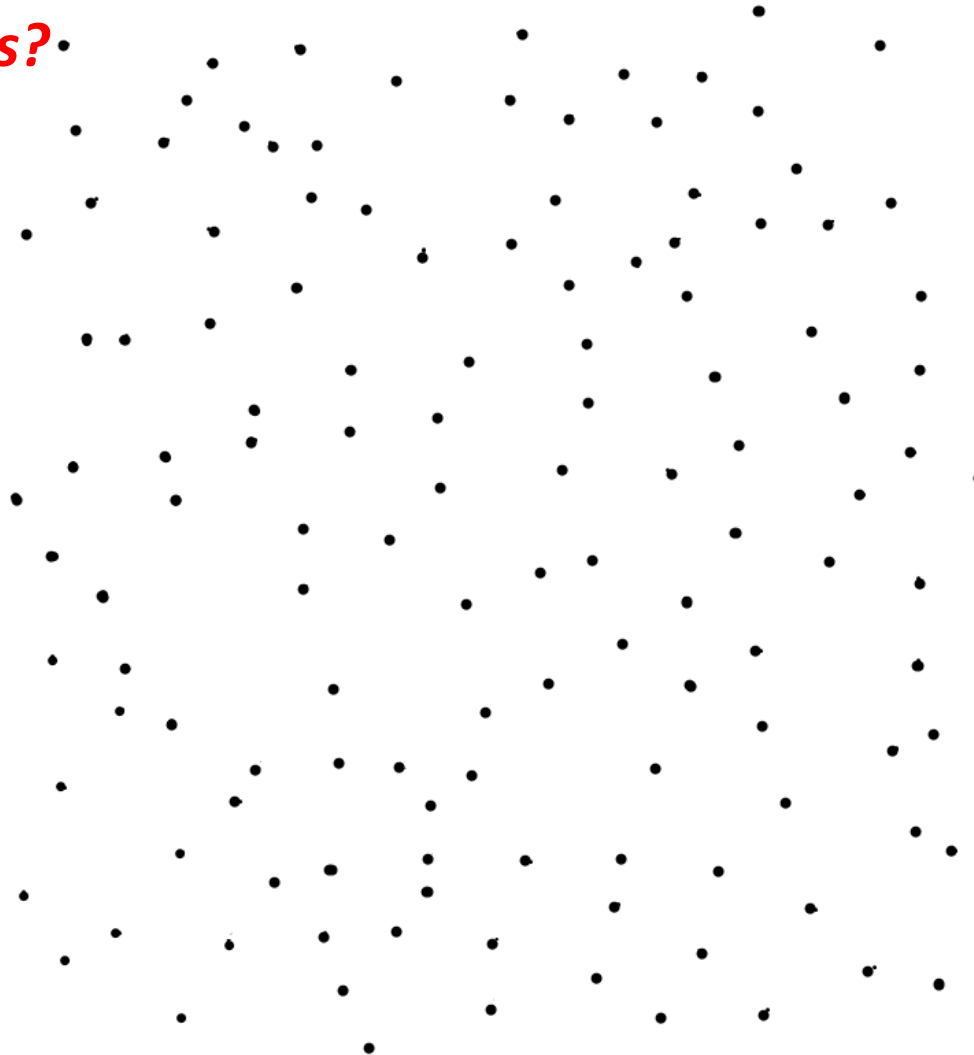
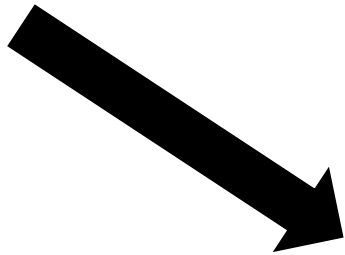
- Automated binary analyses useful for certain tasks (e.g., finding crashes)
- Many binary analyses can't be automated
- Expert experience and heuristics are still **key** to binary analyses

Benefits of Interactive Binary Analysis

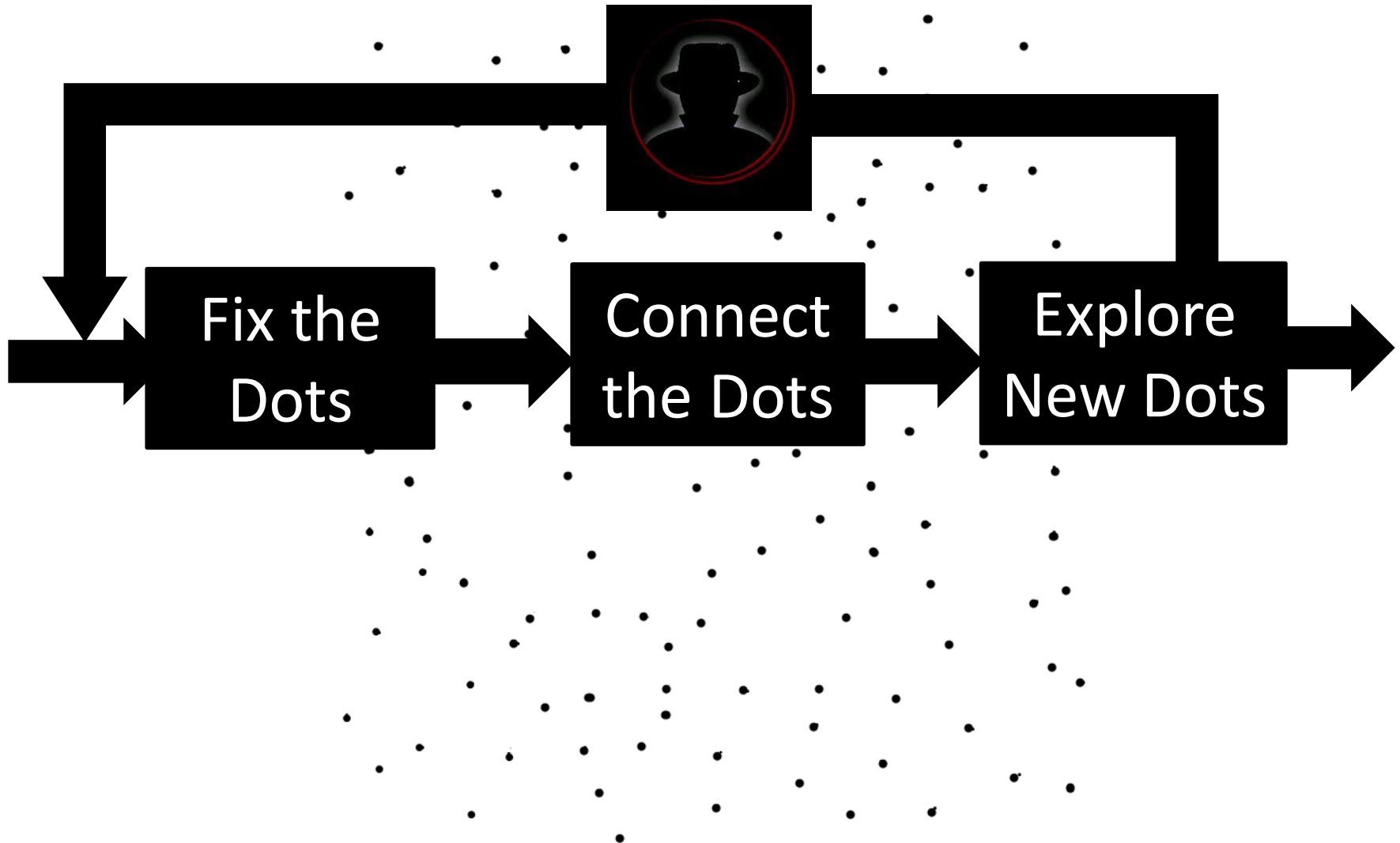
- Applicable to many security problems
- Our tools increase productivity in:
 - Finding vulnerabilities
 - Analyzing root causes
 - Exploitability and risk assessment

Interactive Analysis Like Connecting Dots

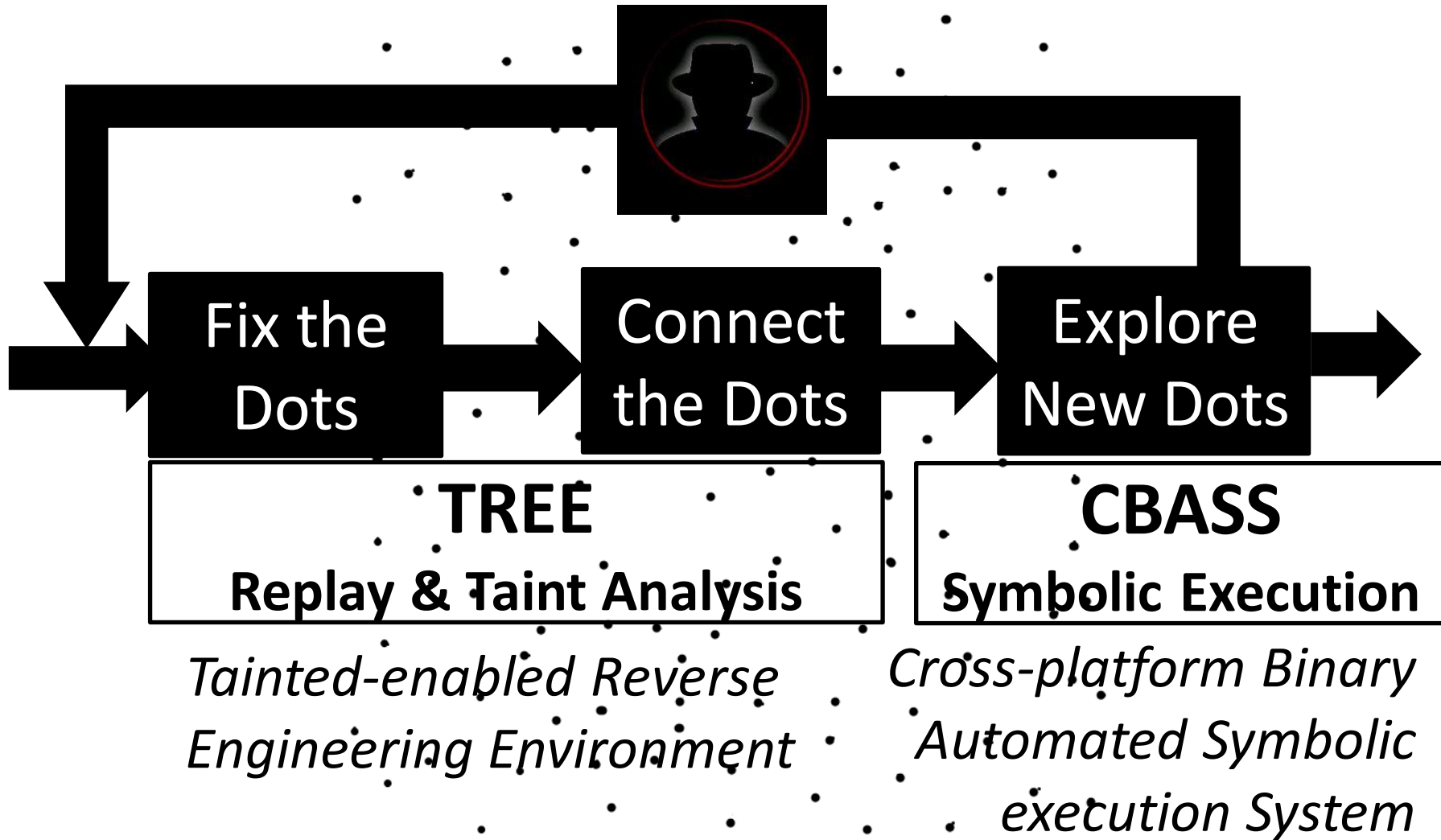
What's in the dots?



Our Tools are Designed to Help



What Do Our Tools Do?



Gaps between Research and Interactive Binary Analysis

- Existing research does not support interactive binary analysis
 - No practical tools
 - No uniform trace collection tools
 - No unified Instruction Set Architecture (ISA) -independent analysis tools

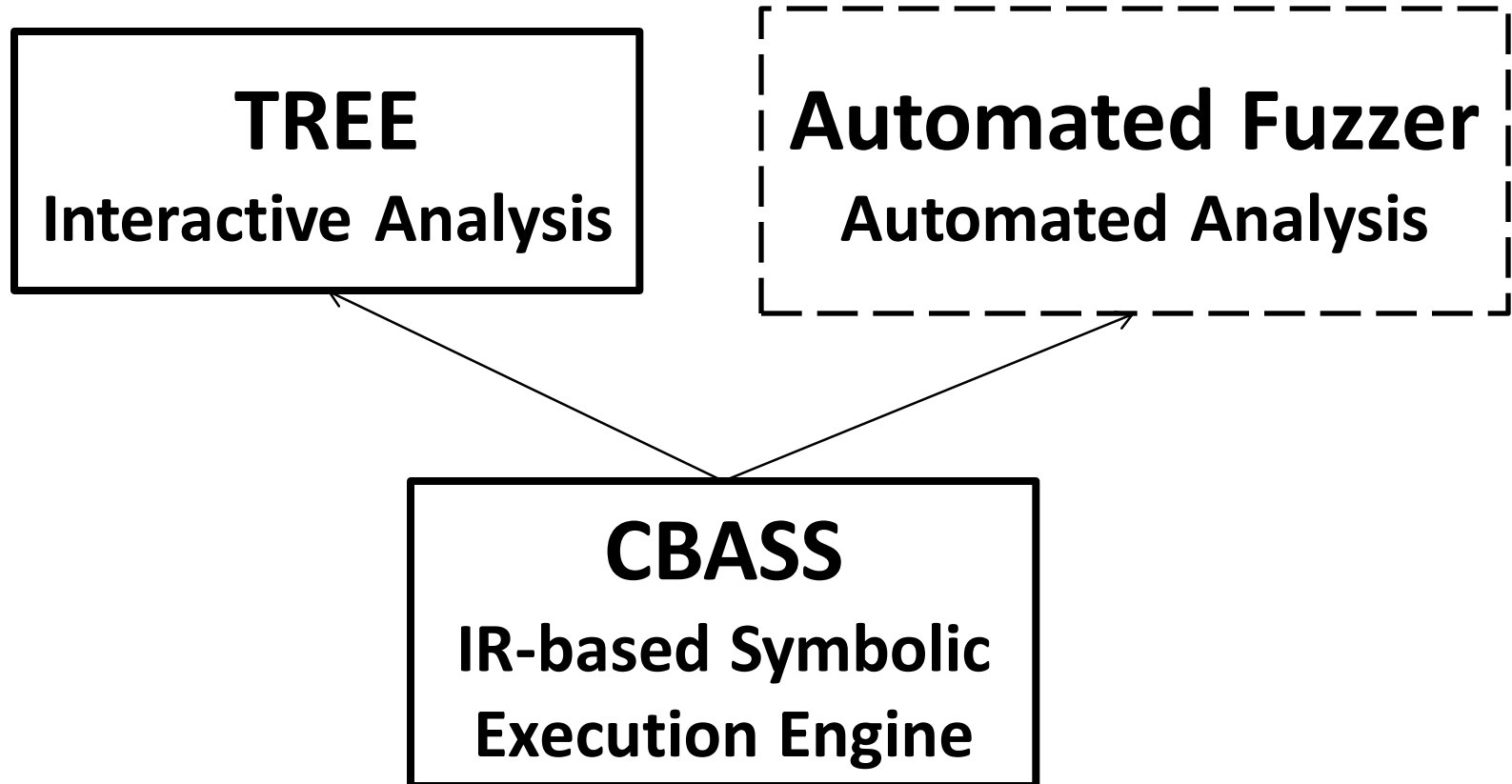
Bringing Proven Research Techniques to Interactive Binary Analysis

- Our tools use dynamic, trace-based, offline analysis approach
 - Interactive binary analysis [1]
 - Dynamic taint analysis ([2][3][4])
 - Symbolic execution/ SMT solver ([2][5])
 - Trace replay ([6])

Making It Practical

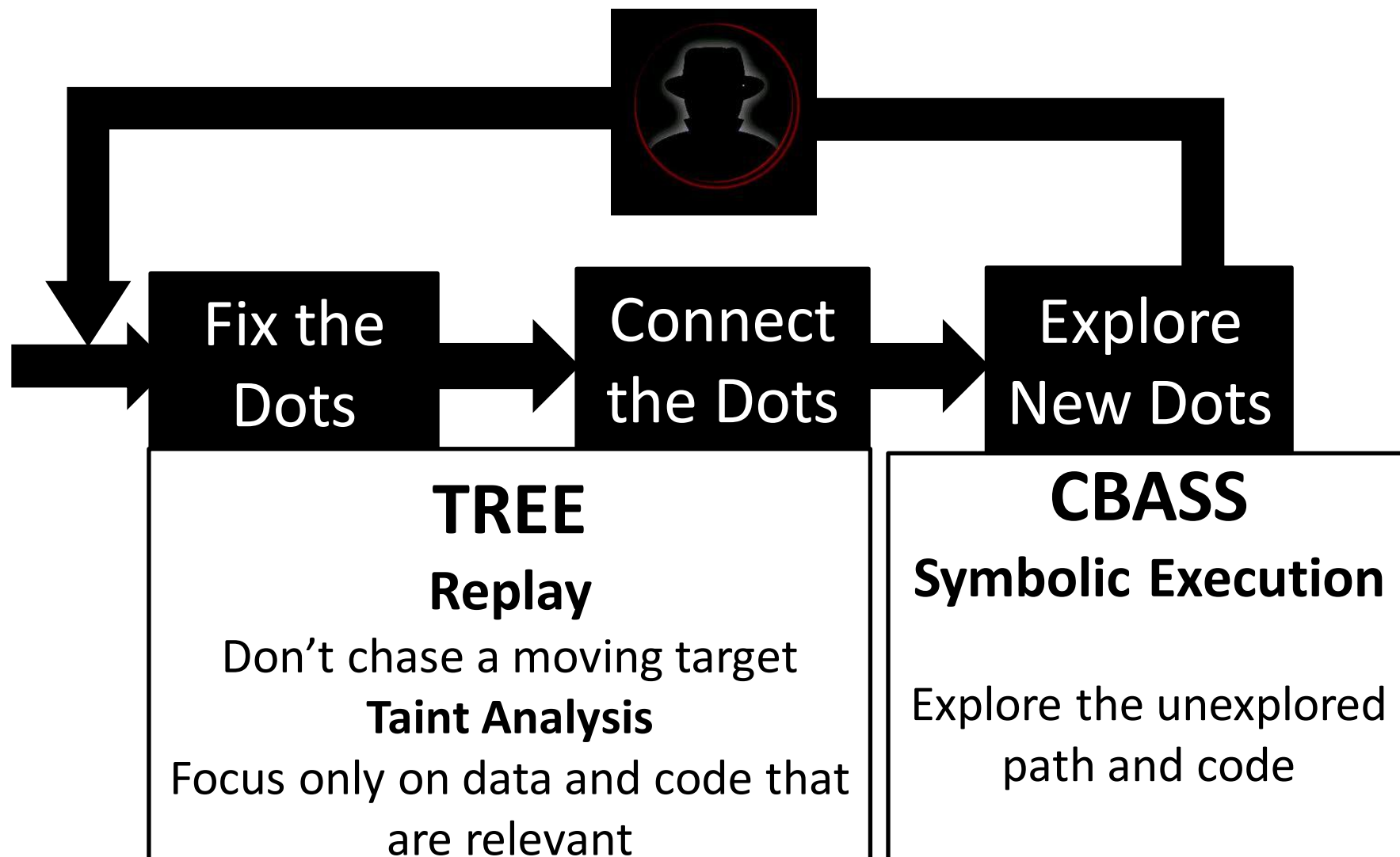
- TREE integrates with IDA Pro now and other mainstream binary analysis environments (later)
- TREE leverages debugging infrastructure to support tracing on multiple platforms
- CBASS uses Intermediate Representation (REIL [6][7])-based approach to support ISA-independent analysis

CBASS Supports Both Automated & Interactive Analysis



TREE fills gaps for interactive analysis

Tools Support Interactive Binary Analyses



Illustrative Dots in Vulnerability Analysis: A Running Example

//INPUT

ReadFile(hFile, sBigBuf, 16, &dwBytesRead, NULL);

//INPUT TRANSFORMATIONS

.....

//PATH CONDITIONS

if(sBigBuf[0]=='b') iCount++;

if(sBigBuf[1]=='a') iCount++;

if(sBigBuf[2]=='d') iCount++;

if(sBigBuf[3]=='!') iCount++;

if(iCount==4) // bad!

StackOVflow(sBigBuf,dwBytesRead)

else // Good

printf("Good!");

//Vulnerable Function

void StackOVflow(char *sBig,int num)

{

char sBuf[8]={0};

.....

for(int i=0;i<num;i++)

//Overflow when num>8

{

sBuf[i] = sBig[i];

}

.....

return;

}

Our Tools Support

Fixing the Dots (TREE)

Fix the Dots

- Reverse engineers don't like moving dots
- Why do the dots move?
 - Concurrency (multi-thread/multi-core) brings non-deterministic behavior
 - ASLR guarantees nothing will be the same

Fix the Dots

- How does TREE work?
 - Generates the trace at runtime
 - Replays it offline
- TREE trace
 - Captures program state = {Instruction, Thread, Register, Memory}
 - Fully automated generation
- TREE can collect traces from multiple platforms
 - Windows/Linux/Mac OS User/Kernel and real devices (Android/ARM, Cisco routers/MIPS, PowePC)

TREE Taint-based Replay vs. Debug-based Replay

- Debug-replay lets **you connect the dots**
 - Single step, stop at function boundary, Breakpoint
- TREE replay **connects dots for you**
 - Deterministic replay with taint-point break

Our Tools Support

Connecting the Dots (TREE)

Connecting Dots is Hard

- Basic elements complex in real programs
 - Code size can be thousands (++) of lines
 - Inputs can come from many places
 - Transformations can be lengthy
 - Paths grow exponentially
- Basic elements likely separated by millions of instructions, spatially and temporally
- Multiple protections built in

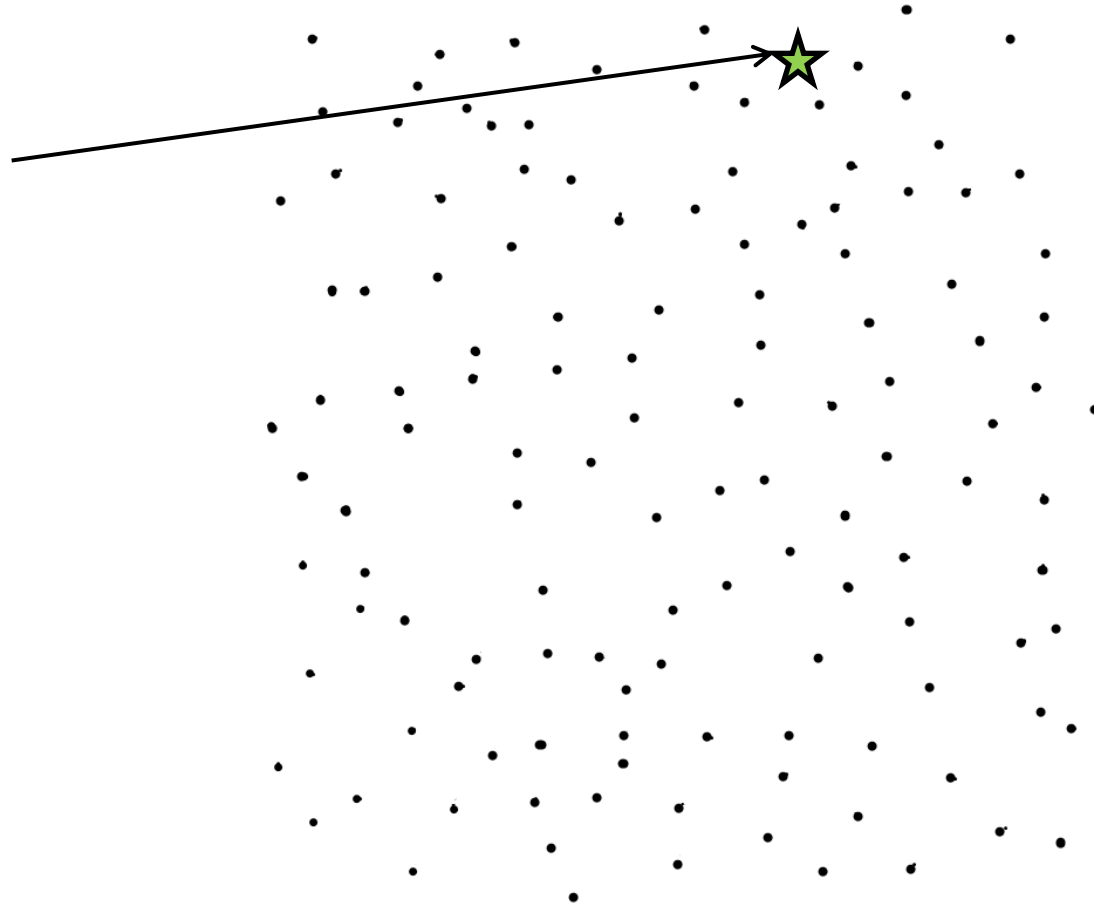
Techniques Help Connect the Dots

- Dynamic Taint Analysis
 - Basic Definitions
 - Taint source
 - Taint Sink:
 - Taint Policy:
- Taint-based Dynamic Slicing
 - Taint focused on data
 - Slicing focused on relevant instructions and sequences

Connect the Dots

- TREE connects dots -- using taint analysis

Taint Source:

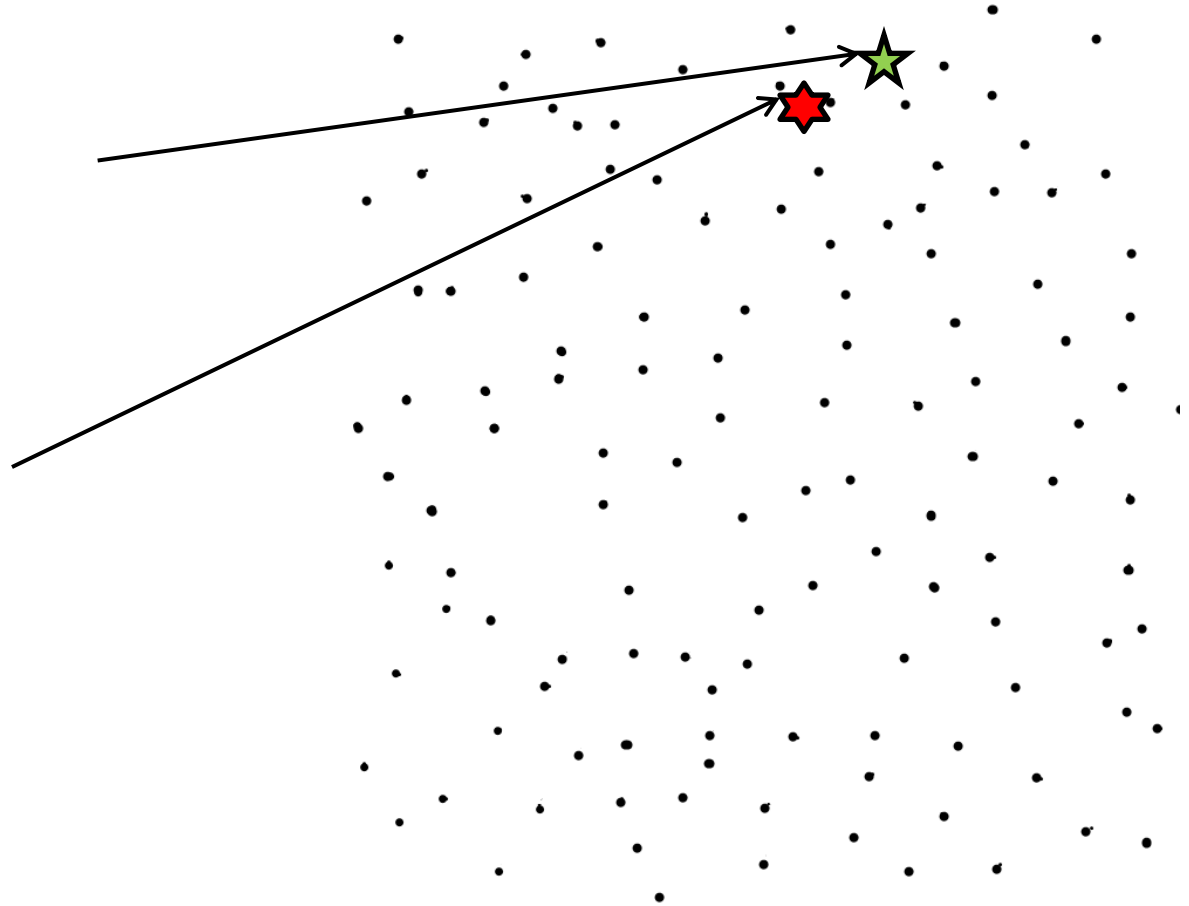


Connect the Dots

- TREE connects dots -- using taint analysis

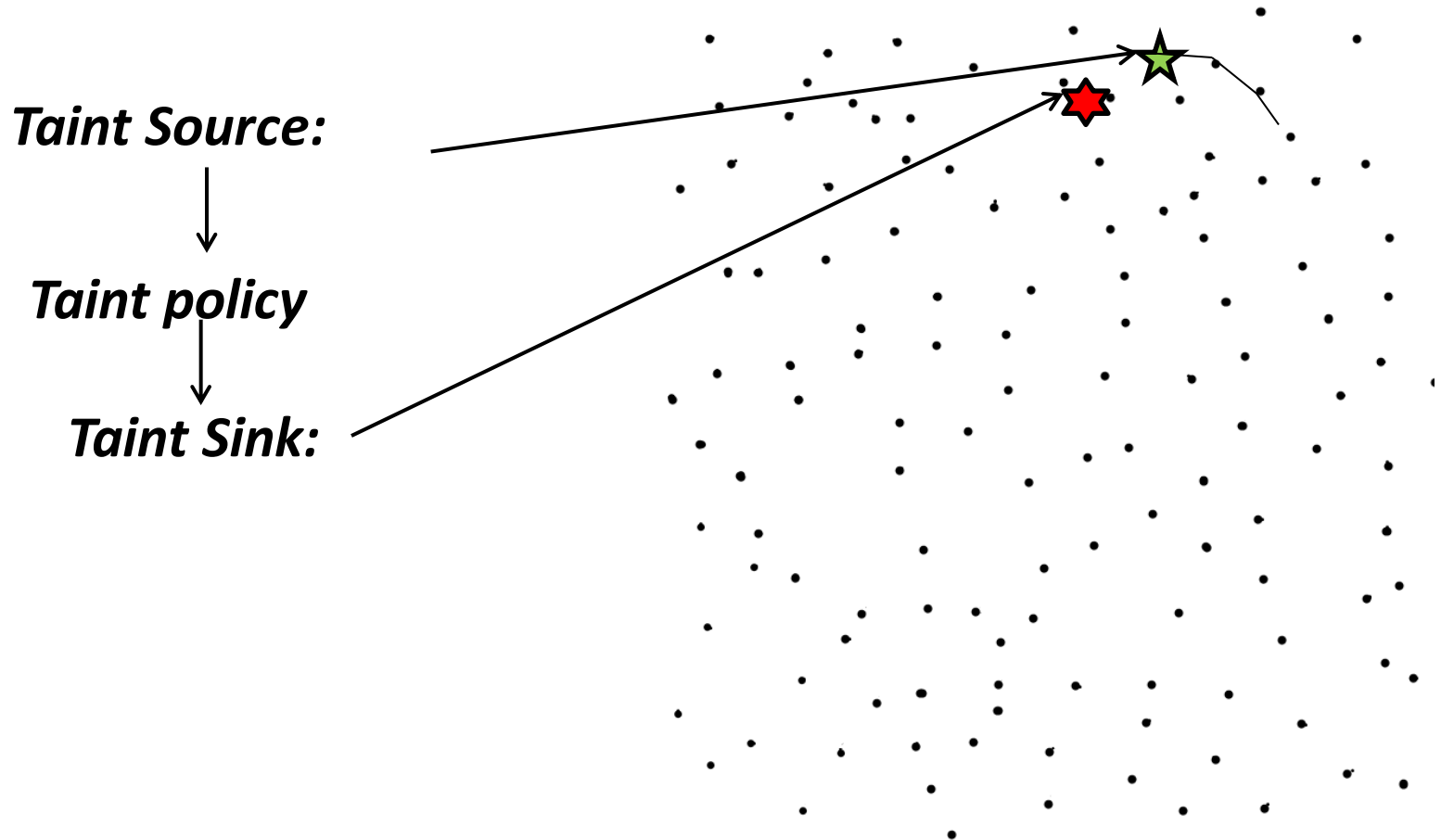
Taint Source:

Taint Sink:



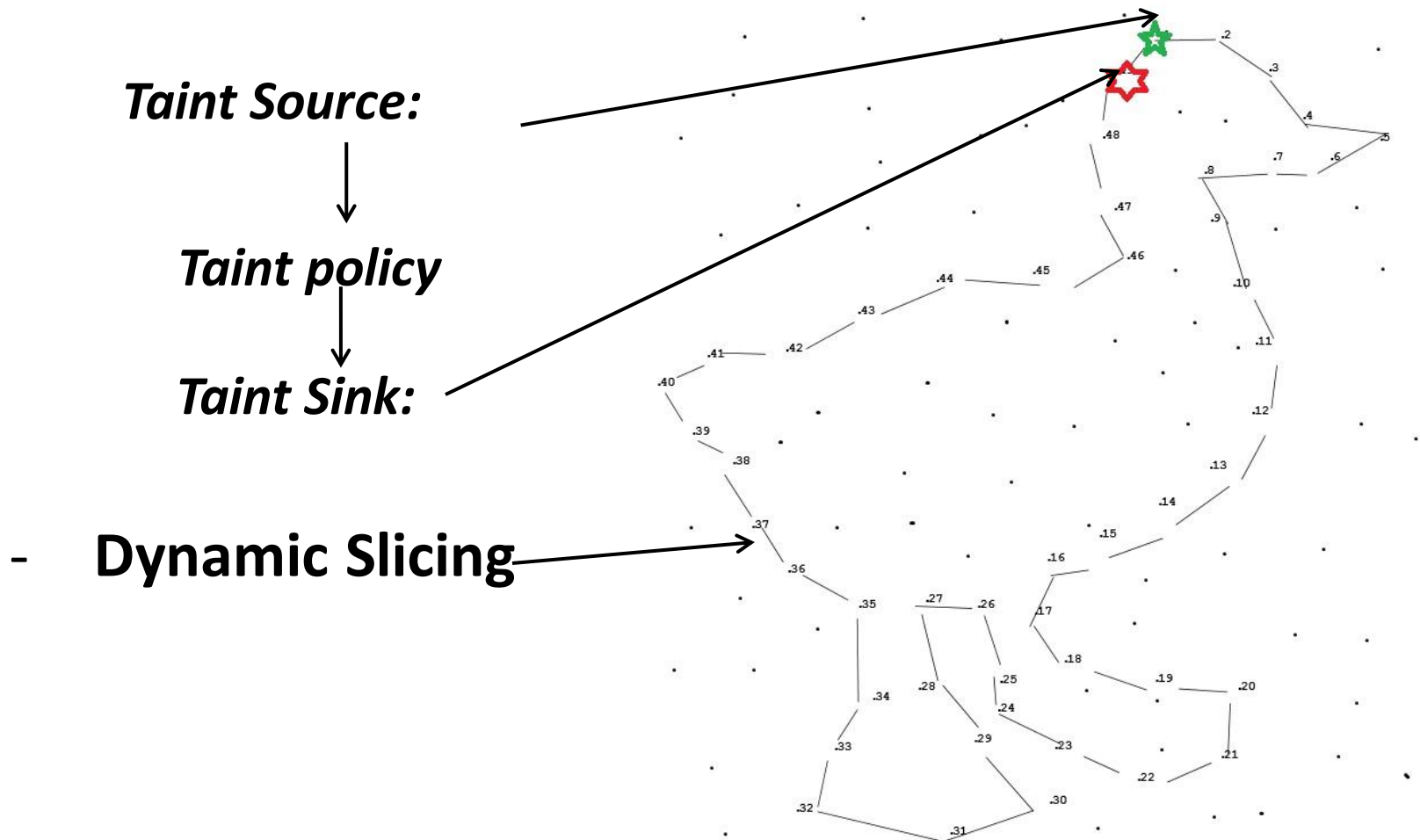
Connect the Dots

- TREE connects dots -- using taint analysis



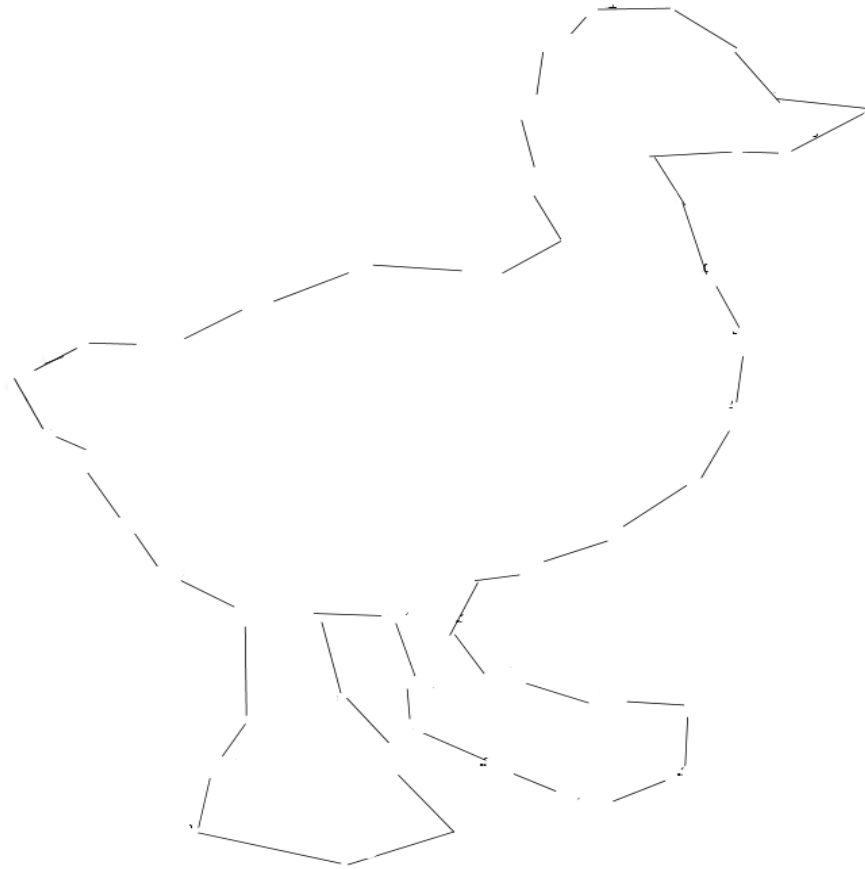
Connect the Dots

- TREE connects dots -- using taint analysis



Find the Dots and Slice that Matter

In practice, most dots don't matter –
eliminate them quickly to focus on what
matters



Connecting Dots in Running Example

Taint Source:
(Input)

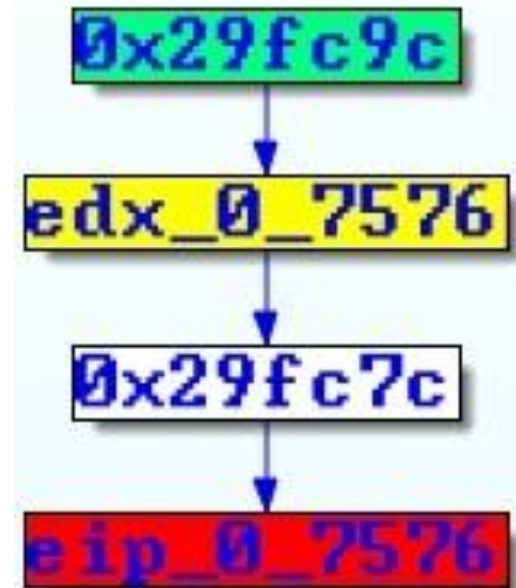
Taint policy
(Data)

Taint Sink: eip

The Slice

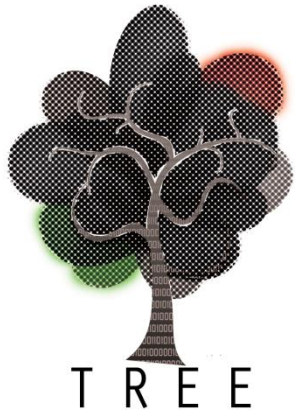
```
call ds:ReadFile
    ↓
movb (%eax), %dl
    ↓
movb %dl,
    -0x8(%ebp,%ecx,1)
    ↓
retl
```

The Taint Graph



What You Connect is What You Get

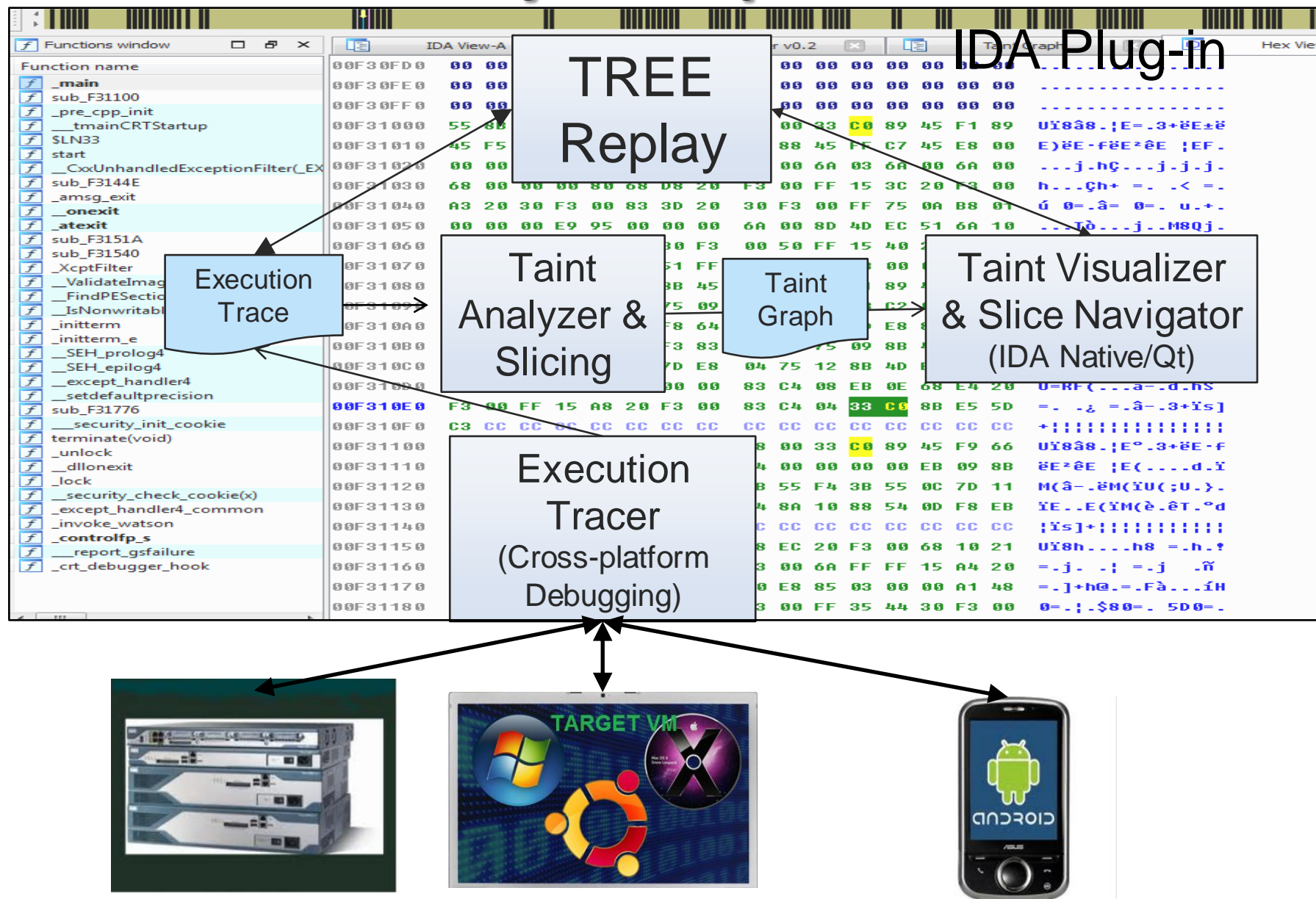
- Dots can be connected in different ways
 - Data dependency
 - Address dependency
 - Branch conditions
 - Loop counter
- Connect dots in different taint policies



TREE

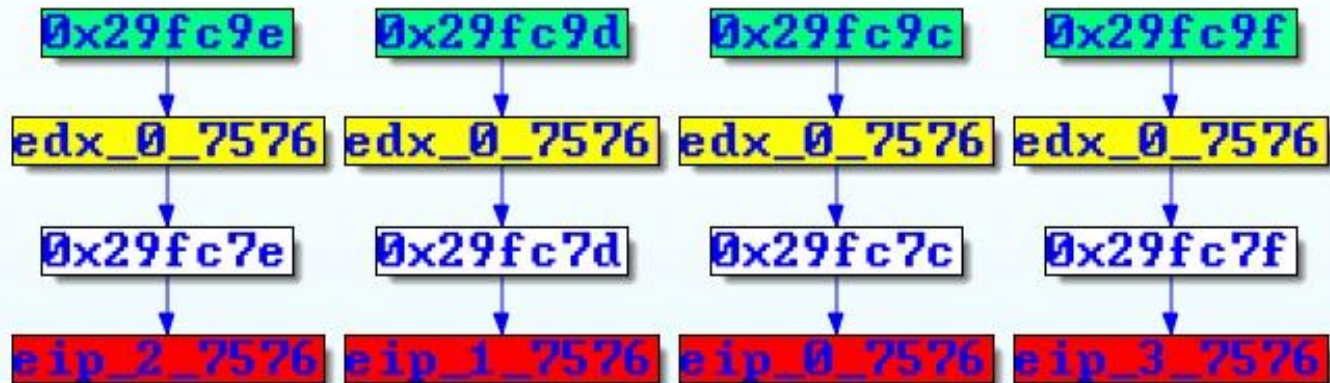
***TAINT-ENABLED
REVERSE ENGINEERING ENVIRONMENT***

TREE Key Components



TREE: The Front-end of Our Interactive Analysis System

Taint
Graph



TREE: The Front-end of Our Interactive Analysis System

Taint
Table

UUID	Type	Name	Start Sequence	End Sequence	formation Instru	Child C	Child D
60	register	eip_3_14876	0x11c		retl		52
59	register	eip_2_14876	0x11c		retl		50
58	register	eip_1_14876	0x11c		retl		48
57	register	eip_0_14876	0x11c		retl		46
52	memory	0x38f79b	0x112		movb %dl, -...		51
51	register	edx_0_14876	0x111	0x117	movb (%eax), %dl		16
50	memory	0x38f79a	0x106		movb %dl, -...		49
49	register	edx_0_14876	0x105	0x10b	movb (%eax)...		15
48	memory	0x38f799	0xfa		movb %dl, -...		47
47	register	edx_0_14876	0xf9	0xff	movb (%eax)...		14
46	memory	0x38f798	0xee		movb %dl, -0x8(%ebp,%...		45
45	register	edx_0_14876	0xed	0xf3	movb (%eax), %dl		13
16	input	0x38f7bb	0x0		0x12f106a		

TREE: The Front-end of Our Interactive Analysis System

Execution Trace Table

	Instruction Address	Disassembly	Registers	Memory Access
270	0x12f1130	mov eax, [ebp+arg_0]	eax=0x38f7ba ebp=0x38f794	R 4 0x38f79c
271	0x12f1133	add eax, [ebp+var_C]	eax=0x38f7ac ebp=0x38f794 eflags=0x287	R 4 0x38f788
272	0x12f1136	mov ecx, [ebp+var_C]	ebp=0x38f794 ecx=0xf	R 4 0x38f788
273	0x12f1139	mov dl, [eax]	eax=0x38f7bb dl=0xf	R 1 0x38f7bb
274	0x12f113b	mov [ebp+ecx+var_8], dl	dl=0x68 ebp=0x38f794 ecx=0xf	W 1 0x38f79b
275	0x12f113f	jmp short loc_12F111F	eip=0x12f113f	
276	0x12f111f	mov ecx, [ebp+var_C]	ebp=0x38f794 ecx=0xf	R 4 0x38f788
277	0x12f1122	add ecx, 1	eflags=0x216 ecx=0xf	
278	0x12f1125	mov [ebp+var_C], ecx	ebp=0x38f794 ecx=0x10	W 4 0x38f788

TREE: The Front-end of Our Interactive Analysis System

The screenshot displays the TREE software interface with three tabs: 'Stack View', 'Register View', and 'Memory View'. The 'Memory View' is active, showing a list of memory addresses and their corresponding hexadecimal values. A callout box points to the 'Register View' and 'Memory View' tabs, labeled 'Register/stack/memory Views'. The status bar at the bottom shows 'UNKNOWN | 0038F798: Stack[00003A1C]:0038F798'.

Address	Value
0038F758	0C 3F 7D 77 2C 85 4A 74 80 38 2F 0
0038F768	00 00 00 00 F0 53 7D 77 5C F7 38 0
0038F778	F0 F7 38 00 23 41 87 77 B4 4D 0F 0
0038F788	0C 3F 7D 77 70 10 2F 01 5C 00 00 0
0038F798	10 00 00 00 A8 F7 38 00 00 00 00 0
0038F7A8	10 00 00 00 62 61 64 21 62 65 74 7
0038F7B8	73 74 74 68 00 F8 38 00 E1 12 2F 0
0038F7C8	A0 1B 45 00 38 20 45 00 40 EA 7A 7
0038F7D8	00 00 00 00 00 E0 FD 7E 00 00 00 0
0038F7E8	D0 F7 38 00 B2 3E 7E 4A 3C F8 38 0

UNKNOWN | 0038F798: Stack[00003A1C]:0038F798

TREE: The Front-end of Our Interactive Analysis System

Replay is focal point of user interaction

Replayer

Taint Analysis

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

1001

1002

1003

1004

1005

1006

1007

1008

1009

1010

1011

1012

1013

1014

1015

1016

1017

1018

1019

1020

1021

1022

1023

1024

1025

1026

1027

1028

1029

1030

1031

1032

1033

1034

1035

1036

1037

1038

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

1062

1063

1064

1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

1075

1076

1077

1078

1079

1080

1081

1082

1083

1084

1085

1086

1087

1088

1089

1090

1091

1092

1093

1094

1095

1096

1097

1098

1099

1100

1101

1102

1103

1104

1105

1106

1107

1108

1109

1110

1111

1112

1113

1114

1115

1116

1117

1118

1119

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

1134

1135

1136

1137

1138

1139

1140

1141

1142

1143

1144

1145

1146

1147

1148

1149

1150

1151

1152

1153

1154

1155

1156

1157

1158

1159

1160

1161

1162

1163

1164

1165

1166

1167

1168

1169

1170

1171

1172

1173

1174

1175

1176

1177

1178

1179

1180

1181

1182

1183

1184

1185

1186

1187

1188

1189

1190

1191

1192

1193

1194

1195

1196

1197

1198

1199

1200

1201

1202

1203

1204

1205

1206

1207

1208

1209

1210

1211

1212

1213

1214

1215

1216

1217

1218

1219

1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242

1243

1244

1245

1246

1247

1248

1249

1250

1251

1252

1253

1254

1255

1256

1257

1258

1259

1260

1261

1262

1263

1264

1265

1266

1267

1268

1269

1270

1271

1272

1273

1274

1275

1276

1277

1278

1279

1280

1281

1282

1283

1284

1285

1286

1287

1288

1289

1290

1291

1292

1293

1294

1295

1296

1297

1298

1299

1300

1301

1302

1303

1304

1305

1306

1307

1308

1309

1310

1311

1312

1313

1314

1315

1316

1317

1318

1319

1320

1321

1322

1323

1324

1325

1326

1327

1328

1329

1330

1331

1332

1333

1334

1335

1336

1337

1338

1339

1340

1341

1342

1343

1344

1345

1346

1347

1348

1349

1350

1351

1352

1353

1354

1355

1356

1357

1358

1359

1360

1361

1362

1363

1364

1365

1366

1367

1368

1369

1370

1371

1372

1373

1374

1375

1376

1377

1378

1379

1380

1381

1382

1383

1384

1385

1386

1387

1388

1389

1390

1391

1392

1393

1394

1395

1396

1397

1398

1399

1400

1401

1402

1403

1404

1405

1406

1407

1408

1409

1410

1411

1412

1413

1414

1415

1416

1417

1418

1419

1420

1421

1422

1423

1424

1425

1426

1427

1428

1429

1430

1431

1432

1433

1434

1435

1436

1437

1438

1439

1440

1441

1442

1443

1444

1445

1446

1447

1448

1449

1450

1451

1452

1453

1454

1455

1456

1457

1458

1459

1460

1461

1462

1463

1464

1465

1466

1467

1468

1469

1470

1471

1472

1473

1474

1475

1476

1477

1478

1479

1480

1481

1482

1483

1484

1485

1486

1

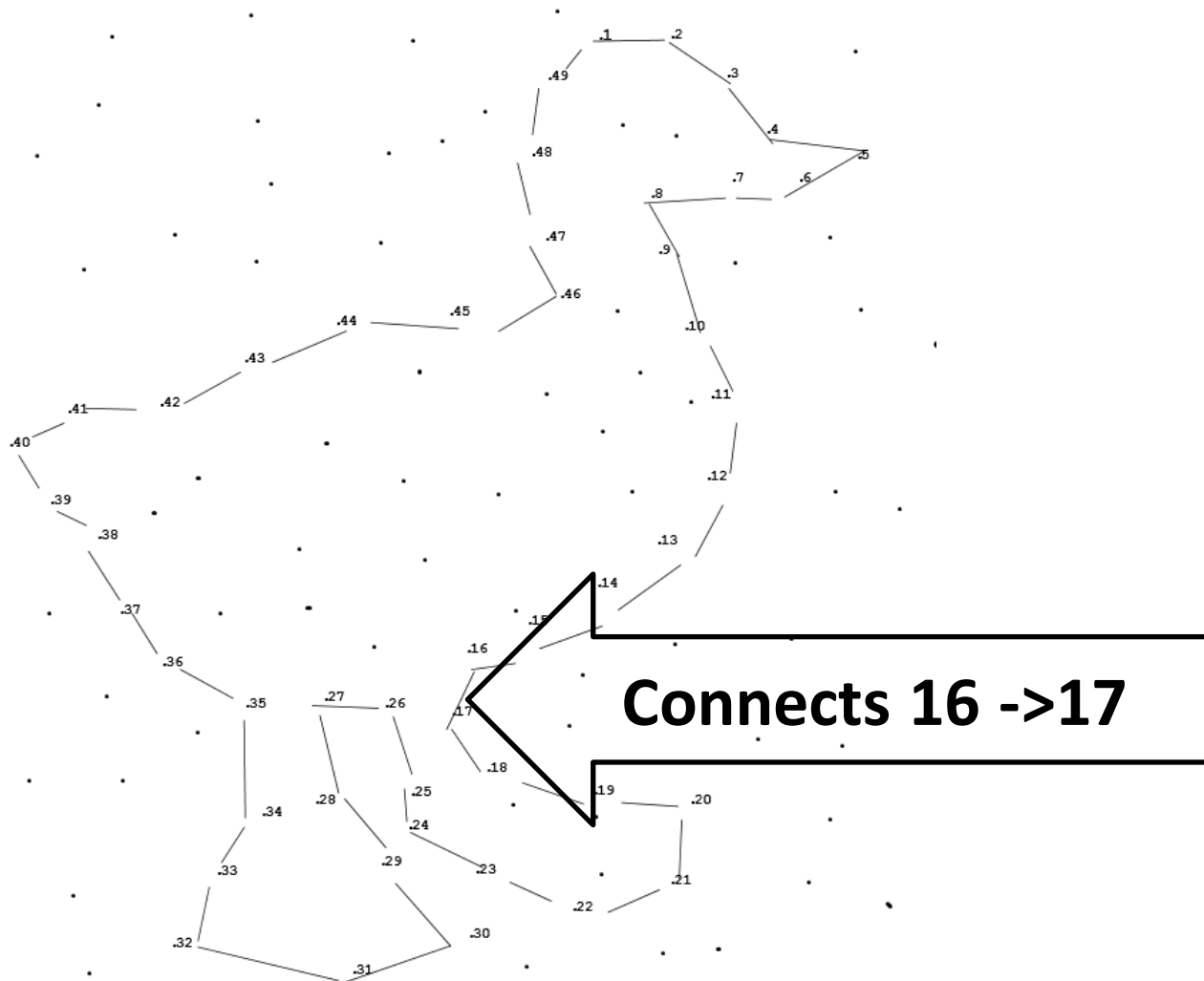
Tree Demo

Using TREE to Analyze a Crash

Our Tools Support

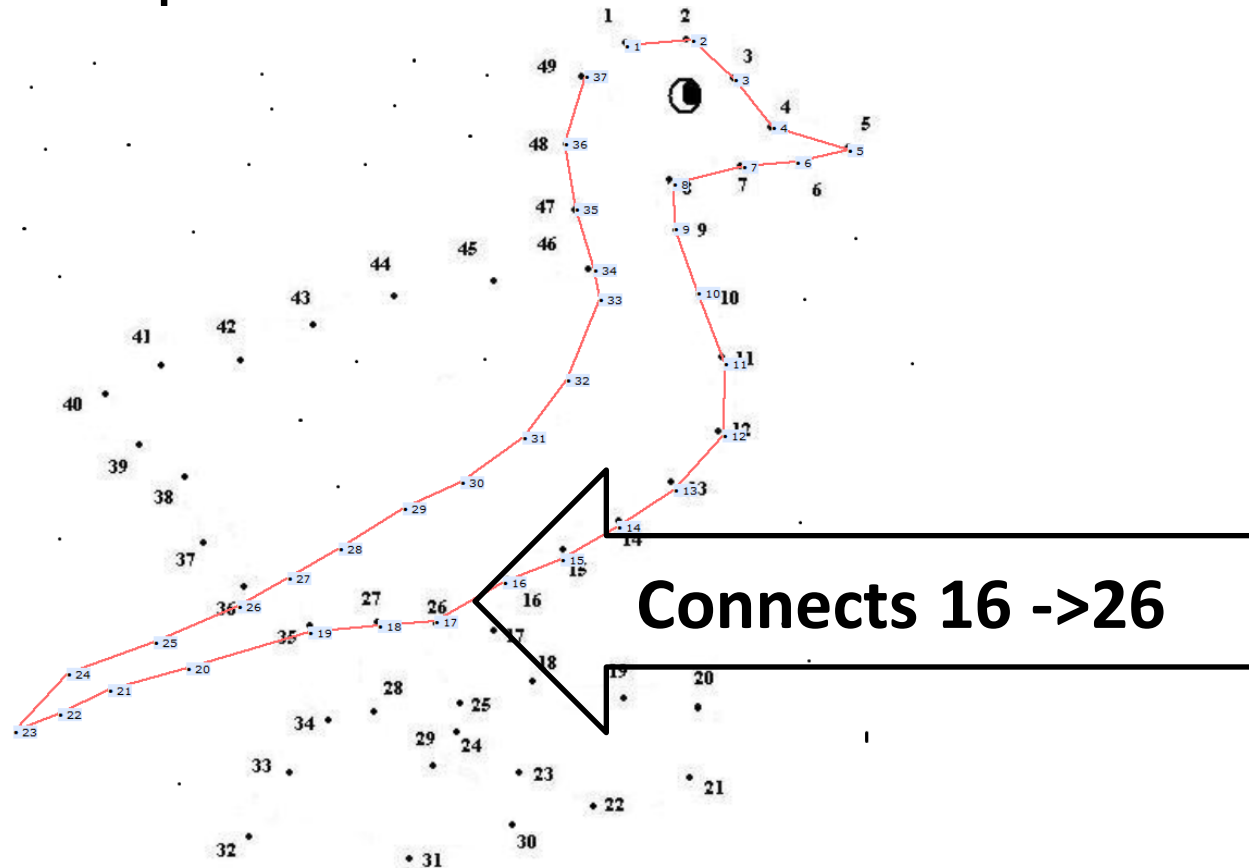
Exploring New Dots

A Key Branch Point for a Duck



The Path for a ...

- Reverse engineers don't just connect dots; they want to explore new dots:



Explore New Dots

- How do you force the program to take a different path to lead to “bad!”?

```
//INPUT
```

```
ReadFile(hFile, sBigBuf, 16, &dwBytesRead, NULL);
```

```
.....
```

```
//PATH CONDITION
```

```
if(sBigBuf[0]=='b') iCount++;
```

```
if(sBigBuf[1]=='a') iCount++;
```

```
if(sBigBuf[2]=='d') iCount++;
```

```
if(sBigBuf[3]=='!') iCount++;
```

```
if(iCount==4) // “bad!” path
```

```
StackOverflow(sBigBuf,dwBytesRead) ?
```

```
Else // “Good” path
```

```
printf(“Good!”);
```

Explore New Dots

- User wants execution to take different path at a branch point Y – what input will make that happen?

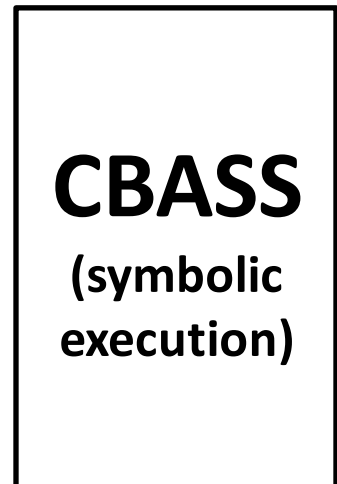
User:

*How to execute
different path
at branch Y?*

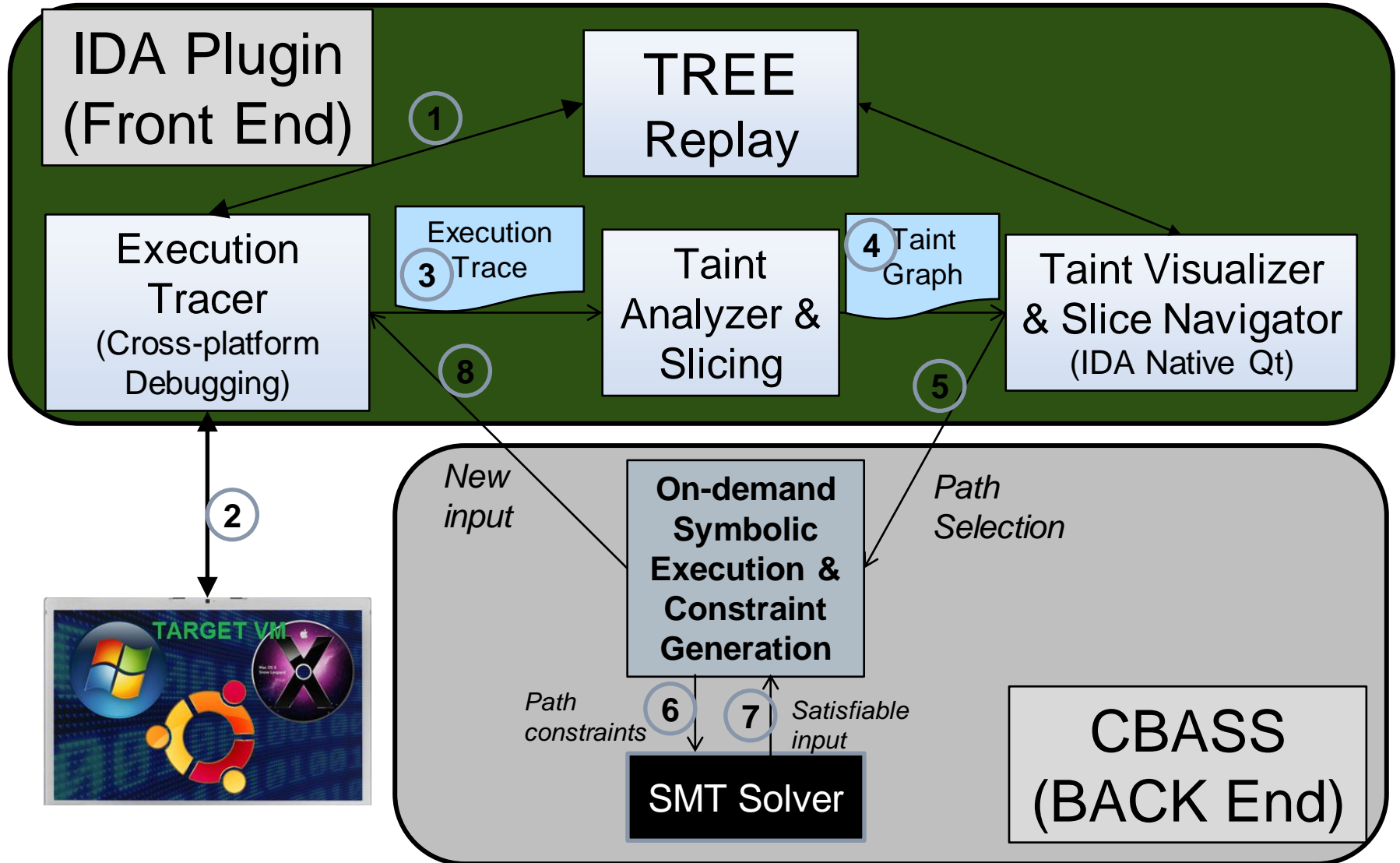
TREE: Input
[0] must be 'b'

*TREE: Can
we negate
path
condition
at Y?*

CBASS:
This byte
must be 'b'



Explore New Dots Demo



Task 1:

Force the Program to Take “bad!” Path

//INPUT

```
ReadFile(hFile, sBigBuf, 16,  
&dwBytesRead, NULL);
```

//INPUT TRANSFORMATION

.....

//PATH CONDITION

```
if(sBigBuf[0]=='b') iCount++;
```

```
if(sBigBuf[1]=='a') iCount++;
```

```
if(sBigBuf[2]=='d') iCount++;
```

```
if(sBigBuf[3]=='!') iCount++;
```

```
if(iCount==4) // “bad!” path
```

//Vulnerable Function

```
StackOVflow(sBigBuf,dwBytesRead)
```

```
else
```

```
printf(“Good!”);
```

Branch Conditions In Disassembly

```
movsx    edx, [ebp+Buffer]  
cmp      edx, 62h  
jnz      short loc_F3108F
```

```
mov      eax, [ebp+var_18]  
add      eax, 1  
mov      [ebp+var_18], eax
```

```
loc_F3108F:  
movsx    ecx, byte ptr [ebp+var_F]  
cmp      ecx, 61h  
jnz      short loc_F310A1
```

```
mov      edx, [ebp+var_18]  
add      edx, 1  
mov      [ebp+var_18], edx
```

```
loc_F310A1:  
movsx    eax, byte ptr [ebp+var_F+1]  
cmp      eax, 64h  
jnz      short loc_F310B3
```

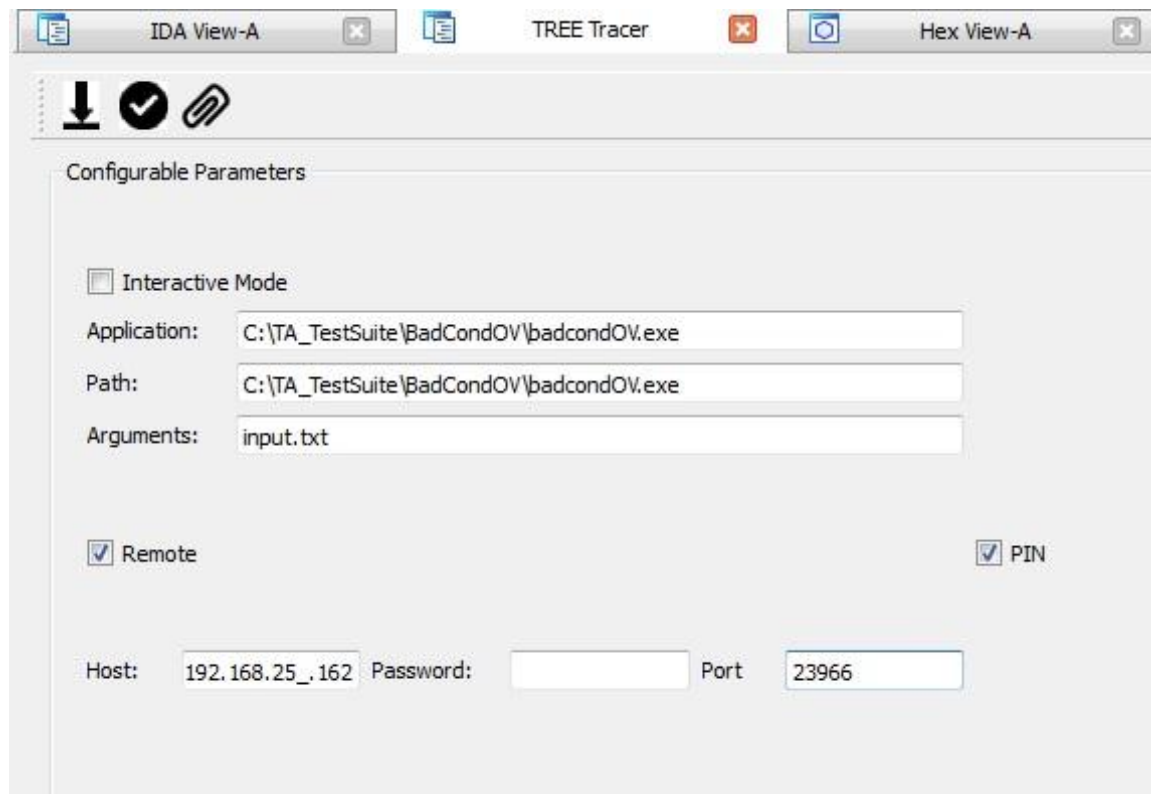
```
mov      ecx, [ebp+var_18]  
add      ecx, 1  
mov      [ebp+var_18], ecx
```

```
loc_F310B3:  
movsx    edx, byte ptr [ebp+var_F+2]  
cmp      edx, 21h  
jnz      short loc_F310C5
```

① **TREE Pin Trace**

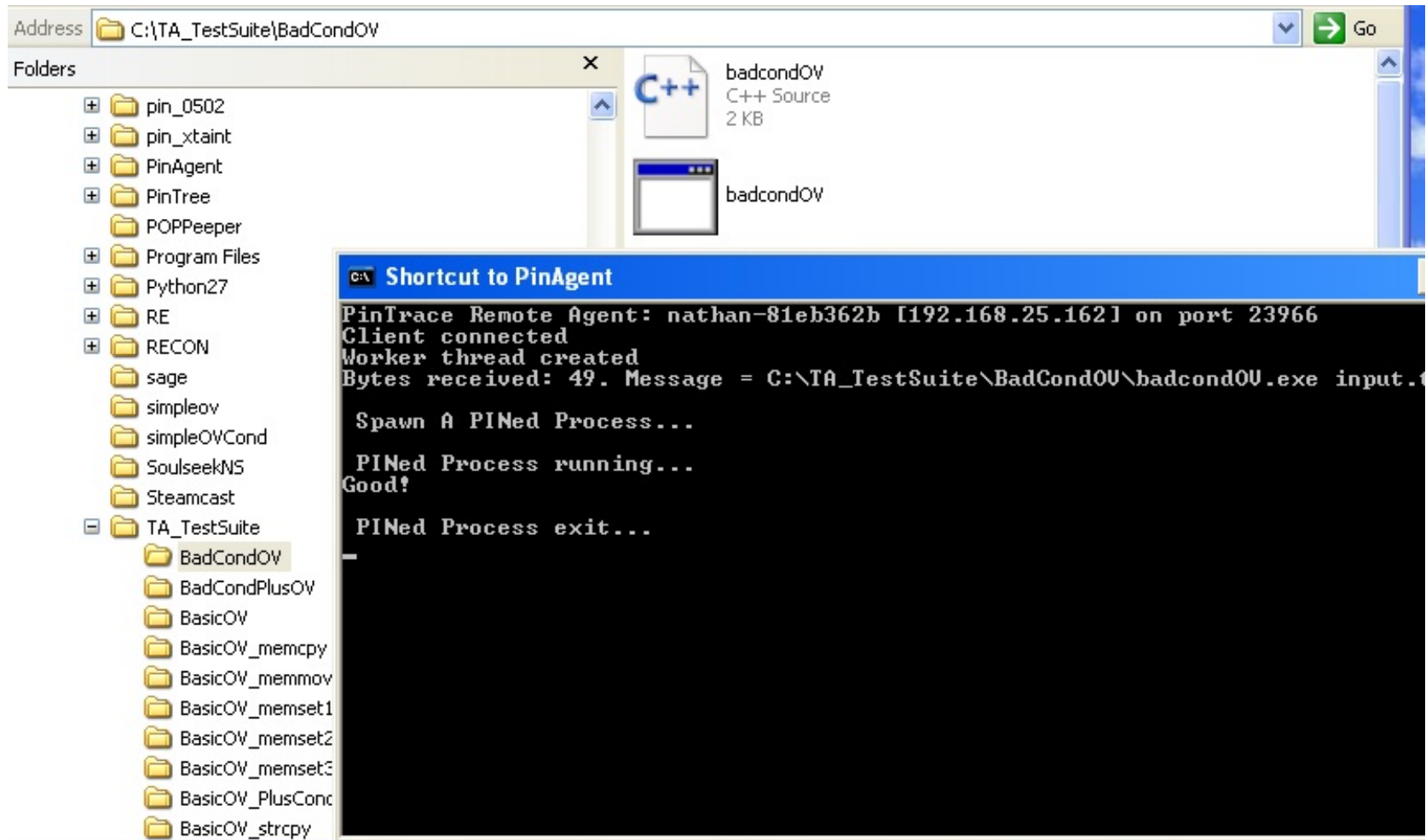
PIN: A popular Dynamic Binary Instrumentation (DBI) Framework

<http://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool>



② TREE Console: Trace Generation

PINAgent: Connects TREE with PIN tracer



③ TREE: Taint Analysis Configuration

☒ Replayer ☒ Taint Analysis

☒ ☒

Taint Propagation Policy

☐ Taint_DATA

☒ Taint_BRANCH

☐ Taint_COUNTER

☐ Taint_ADDRESS

Instruction Set Architecture

☒ x86

☐ x86_64

☐ ARM

☐ PPC

☐ MIPS

Misc

☒ PIN

☐ Verbose

Image Load Table

Name	Address	Size
['badcon...	0x12f0000	0x5000
ntdll.dll	0x77e00000	0x180000
kernel32.dll	0x777c0000	0x110000
kernel32.dll	0x777c0000	0x110000
KernelBas...	0x75ff0000	0x47000
user32.dll	0x75700000	0x100000

Taint Source Table

Input Address	Size	
0x38f7ac	16	626164210

Taint Graph Output

Path Conditions:

[19]bc_0x3a[0x3a:0x0] <-jnz 0x9

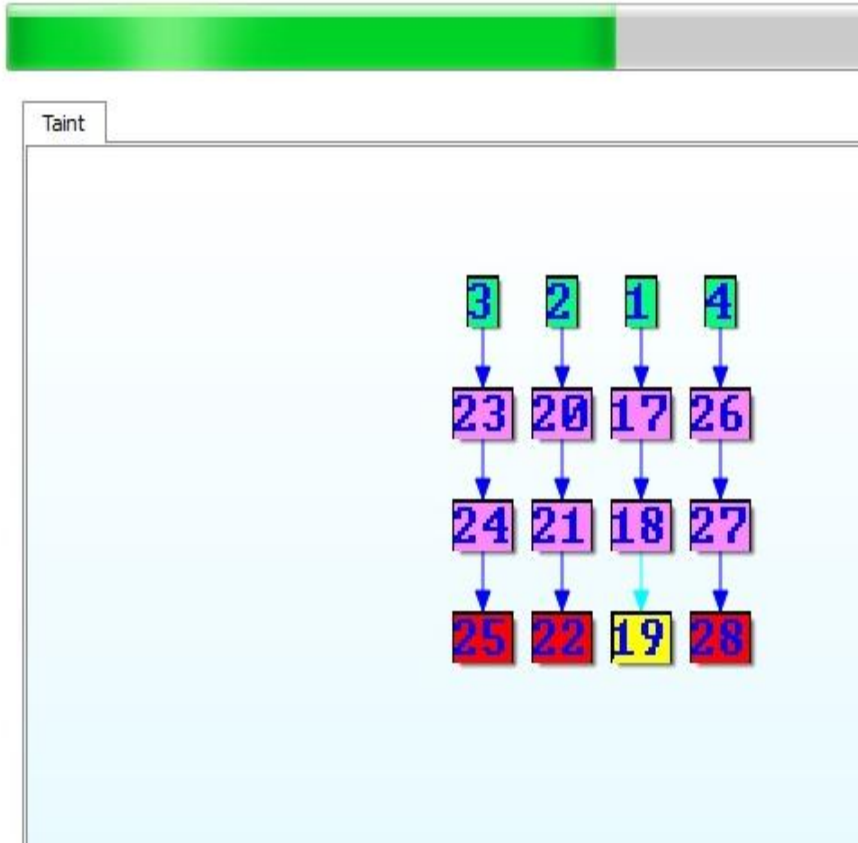
[18]reg_eflags_0[0x39:0x0]
[0x3c:0x0] <-cmp \$0x62, %edx

[17]reg_edx_0_0[0x38:0x0]
[0x41:0x0] <-movsxb -0x10(%ebp), %edx

[1]in_0x12ff6c[0x0:0x0]
[0xa5b:0x0] <-0xffff:ReadFile

[22]bc_0x3d[0x3d:0x0] <-jnz 0x9

4 TREE: Branch Taint Graph



UUID	Type	Name
1	input	0x12ff6c
17	register	edx_0_0
18	register	eflags_0
19	branch	0x3a
2	input	0x12ff6d
20	register	ecx_0_0
21	register	eflags_0
22	branch	0x3d
23	register	eax_0_0
24	register	eflags_0
25	branch	0x40
26	register	edx_0_0
27	register	eflags_0
28	branch	0x43
3	input	0x12ff6e
4	input	0x12ff6f

⑤ Negate Tainted Path Condition to Exercise a New (“Bad”) Path



“Bad!” Path Query

CBASS
(Cross-platform
Symbolic
Execution)

Result

Connecting to CBASS server at 127.0.0.1:8888

Query on branch condition of: [19]bc_0x3a

[jnz 0x9]

Result: Offset=0,Value=98

‘b’

Connecting to CBASS server at 127.0.0.1:8888

Query on branch condition of: [22]bc_0x3d

[jnz 0x9]

Result: Offset=1,Value=97

‘a’

Connecting to CBASS server at 127.0.0.1:8888

Query on branch condition of: [25]bc_0x40

[jnz 0x9]

Result: Offset=2,Value=100

‘d’

On-demand Symbolic Execution (What Happens Behind the Scene)

```
C:\windows\system32\cmd.exe
C:\CBass\src>CBass_basicov_deno_path.bat
```

6

```
(set-logic QF_AUFBV)
```

```
(declare-fun _IN_0x12ff6c_0x0_SEQ0 () (_ BitVec 8))
```

```
(declare-fun EXPR_0 () (_ BitVec 32))
```

```
(assert (= EXPR_0 (bvsub ((_ sign_extend 24) (bvxor _IN_0x12ff6c_0x0_SEQ0 (_ bv128 8))) (_ bv4294967168 32))))
```

```
(assert (= (ite (not (= (ite (not (= (bvand ((_ extract 63 0) (bvsub ((_ sign_extend 32) (bvand ((_ extract 31 0) EXPR_0) (_ bv4294967295 32))) (_ bv98 64))) (_ bv4294967295 64)) (_ bv0 64))) (_ bv1 32) (_ bv0 32)) (_ bv0 32))) (_ bv1 8) (_ bv0 8)) (_ bv0 8)))
```

```
(check-sat)
```

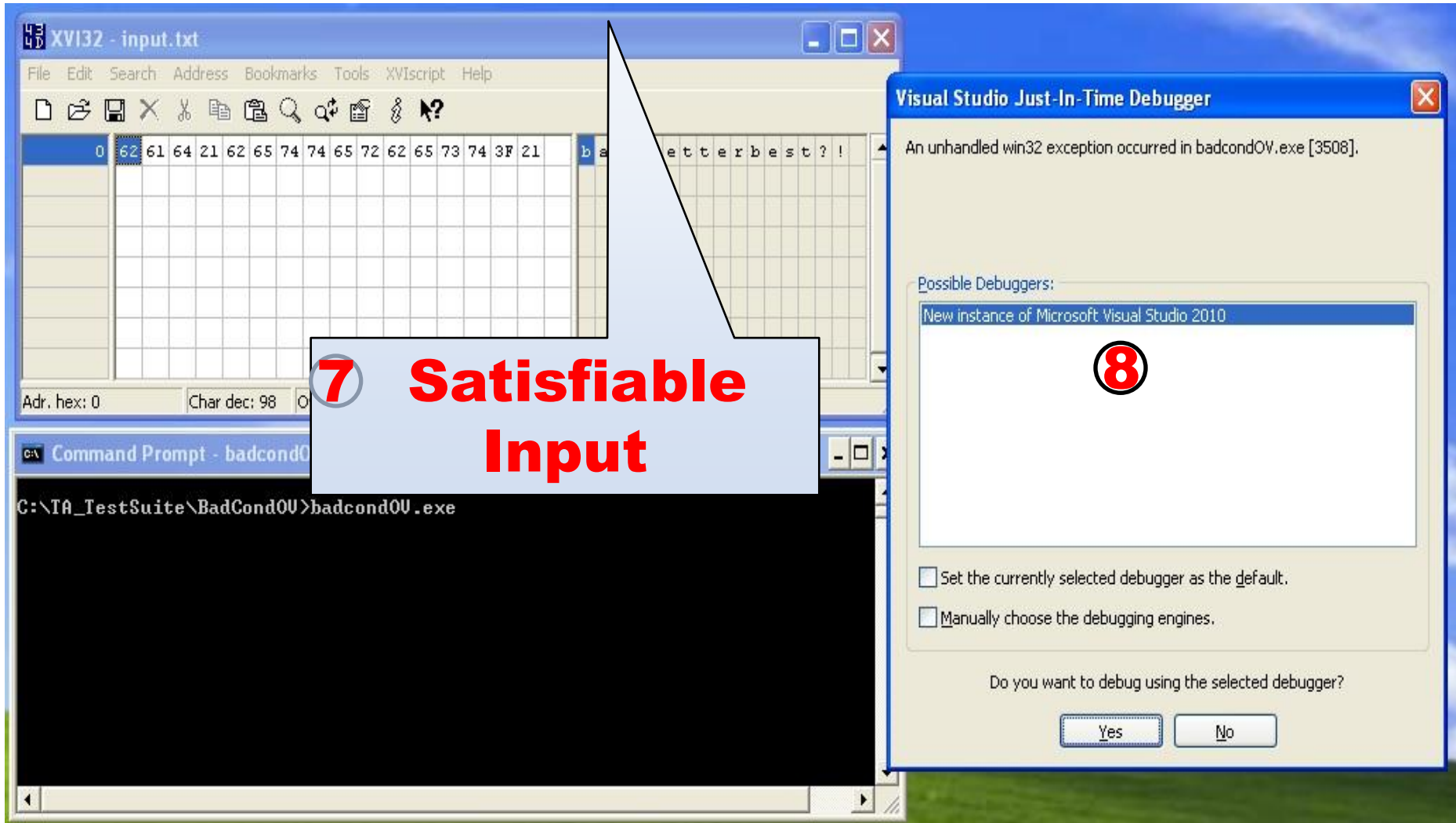
```
(get-value (_IN_0x12ff6c_0x0_SEQ0))
```

```
RD t5, EMPTY , DWORD edx]]
58:4B1877 cnp edx, 98
[4B187788: and [DWORD edx, DWORD 2147483648, DUORD t8], 4B187781: and [D
WORD 98, DWORD 2147483648, DUORD t1], 4B187782: sub [DUORD edx, DWORD 98, QWORD
t2], 4B187783: and [QWORD t2, QWORD 2147483648, DUORD t3], 4B187784: bsh [DWORD
t3, DWORD -31, BYTE SF], 4B187785: xor [DWORD t8, DWORD t1, DWORD t4], 4B187786:
xor [DUORD t8, DUORD t3, DUORD t5], 4B187787: and [DUORD t4, DUORD t5, DUORD t6
], 4B187788: bsh [DUORD t6, DUORD -31, BYTE OF], 4B187789: and [QWORD t2, QWORD
4294967296, QWORD t7], 4B18778A: bsh [QWORD t7, QWORD -32, BYTE CF], 4B18778B: a
nd [QWORD t2, QWORD 4294967295, DUORD t8], 4B18778C: biaz [DUORD t8, EMPTY , BYT
E ZF]]
59:4B187a jnz loc_4B188E
[4B187A88: biaz [BYTE ZF, EMPTY , BYT
Y , DWORD 4198542]]
Invoke Z3 solver z3.exe /smt2 /m path_conditions.smt2
sat
(<<_IN_0x12ff6c_0x0_SEQ0 #x62))
```

7

Satisfiable Input (0x62, 'b')

8 TREE: Re-execute with “Satisfiable” Input

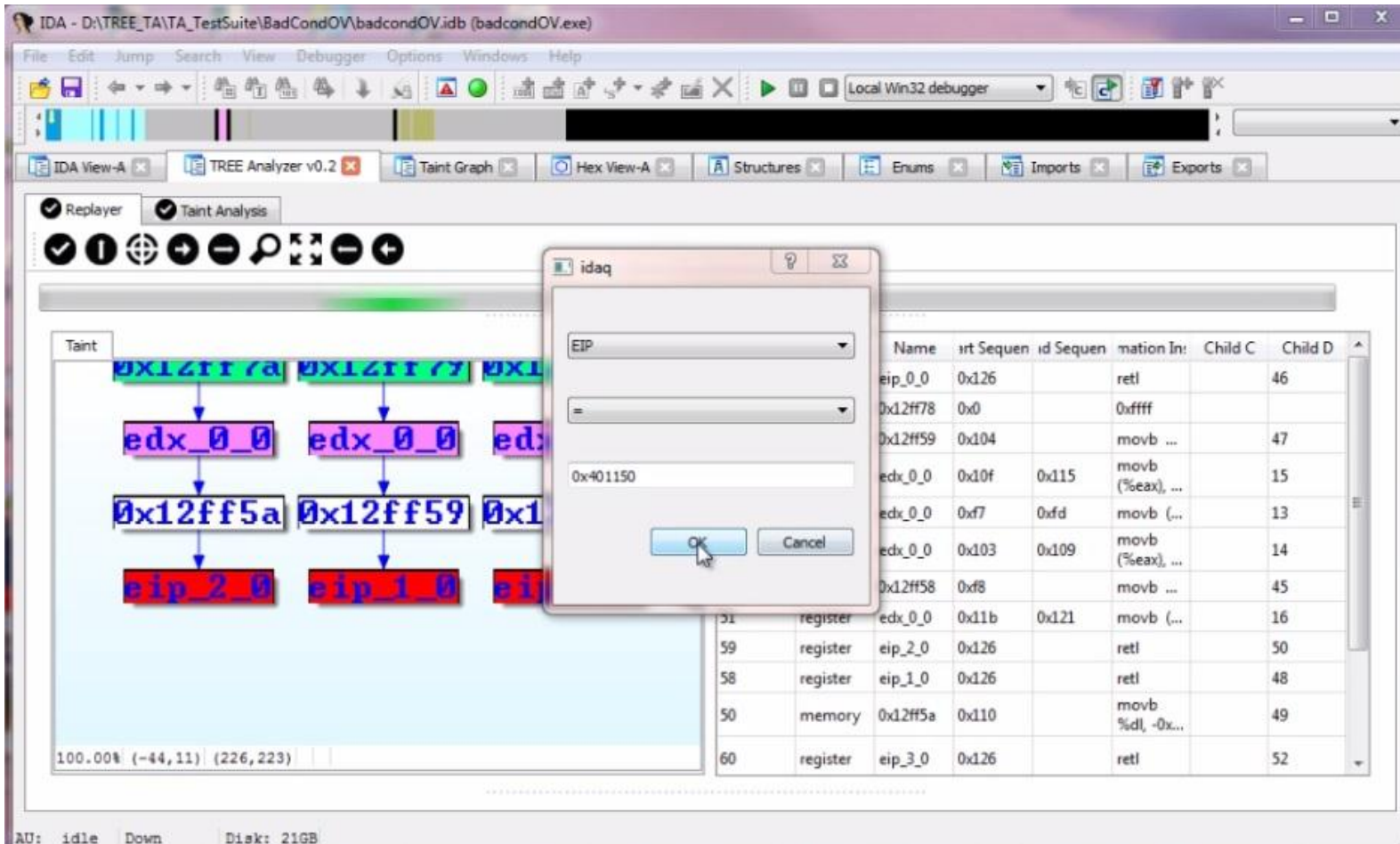


Task 2: Own the Execution

Assume Payload at 0x401150

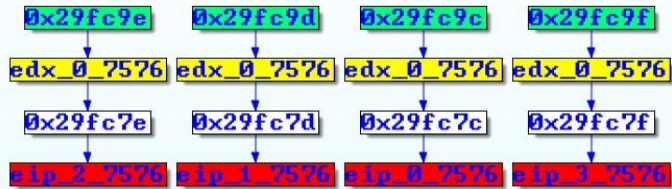
```
.text:00401144 , -----  
.text:00401145 align 10h  
.text:00401150 push ebp  
.text:00401151 mov ebp, esp  
.text:00401153 push 1010h  
.text:00401158 push offset aYouHaveBeenHac ; "*** Yo  
.text:0040115D push offset aCbassCrossPlat ; "CBASSC  
.text:00401162 push 0  
.text:00401164 call ds:MessageBoxA  
.text:0040116A push 0FFFFFFFFh  
.text:0040116C call ds:exit  
.text:00401172 : -----
```

TREE Constraint Dialogue



Task 2:

Own the Execution: From Crash to Exploit



Symbolize Input and perform
concrete-symbolic execution

Symbolic eip =

```
(= expr_0 (concat (bvand (bvor  
_IN_0x12ff6c_0xd_SEQ0 (_ bv0 8)) (_ bv255 8))  
(bvand (bvor _IN_0x12ff6c_0xc_SEQ0 (_ bv0 8))  
(_ bv255 8)))))
```

Query:

```
get-value (_IN_0x12ff6c_0xd_SEQ0  
_IN_0x12ff6c_0xc_SEQ0  
_IN_0x12ff6c_0xe_SEQ0  
IN_0x12ff6c_0xf_SEQ0)
```

Sat:

```
(_IN_0x12ff6c_0xd_SEQ0 #x11  
_IN_0x12ff6c_0xc_SEQ0 #x50  
_IN_0x12ff6c_0xe_SEQ0 #x40  
_IN_0x12ff6c_0xf_SEQ0 #x00)
```

SMT
Solver

TREE/CBASS Demo

**Using CBASS/TREE to Explore
Bad Paths and Refine Exploits**

Real World Case Studies

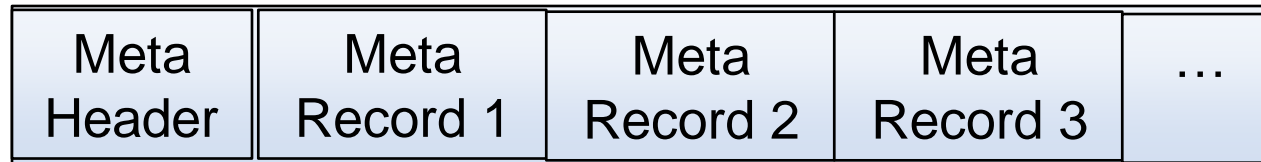
Target Vulnerability	Vulnerability Name	Target Application Mode	Target OS
CVE-2005-4560	Windows WMF	User Mode	Windows
CVE-2207-0038	ANI Vulnerability	User Mode	Windows
OSVDB-2939	AudioCoder Vulnerability	User Mode	Windows
CVE-2011-1985	Win32k Kernel Null Pointer De-reference	Kernel Mode	Windows
CVE-2004-0557	Sound eXchange (SoX) WAV Multiple Buffer Overflow	User Mode	Linux
Compression/Decompression	Zip on Android	User Mode	Real Device Trace Generation (In Progress)

Highlights from Real World Case Study: Windows WMF Vulnerability (CVE-2005-4560)

- WMF SETABORTPROC Escape Vulnerability
 - <http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4560>
 - The Windows Graphical Device Interface library (GDI32.DLL) in Microsoft Windows allows remote attackers to execute arbitrary code via a Windows Metafile (WMF) format image with a crafted SETABORTPROC GDI Escape function call, related to the Windows Picture and Fax Viewer (SHIMGVW.DLL).

WMF Format

- [MS-WMF]: Windows Metafile Format
 - <http://msdn.microsoft.com/en-us/library/cc250370.aspx>
- A Simplified One:
 - <http://wware.sourceforge.net/caolan/ora-wmf.html>
- Overall WMF File Structure:



- One type of record is “escape” record
- SETABORTPROC escape allow an application to register a hook function to handle spooler errors

WMF Crash

The WMF SETABORTPROC Vulnerability

```
rundll32.exe c:\windows\system32\shimgvw.dll,ImageView_Fullscreen  
C:\escape\escape.wmf
```

Dynamic Facts:
229,679
instructions
executed just to
cause the crash

Run a DLL as an App

Run a DLL as an App has encountered a problem and needs to close. We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

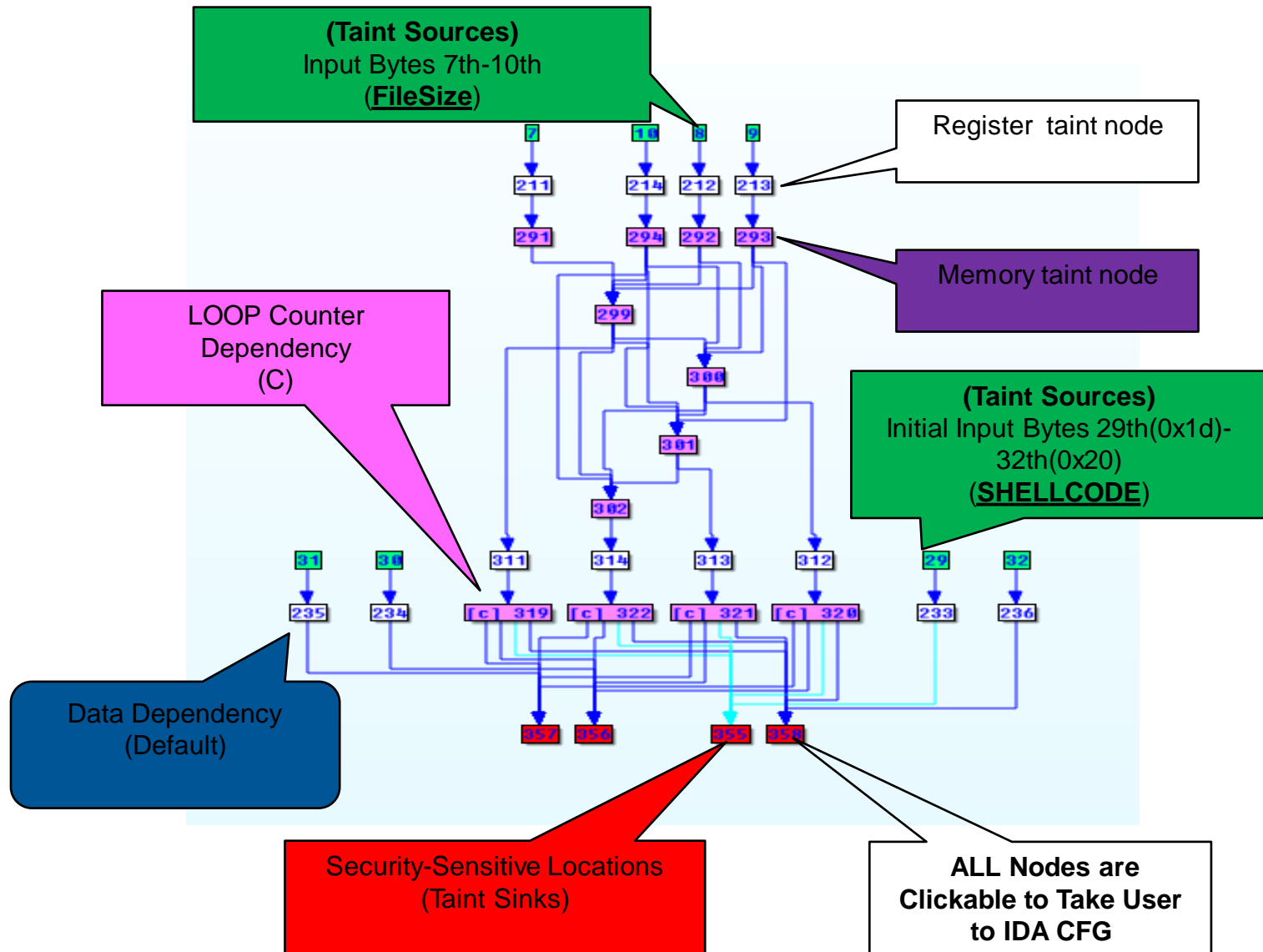
Please tell Microsoft about this problem.

We have created an error report that you can send to help us improve Run a DLL as an App. We will treat this report as confidential and anonymous.

To see what data this error report contains, [click here](#).

WMF Taint Graph

Partial TREE Taint Graph Visualization

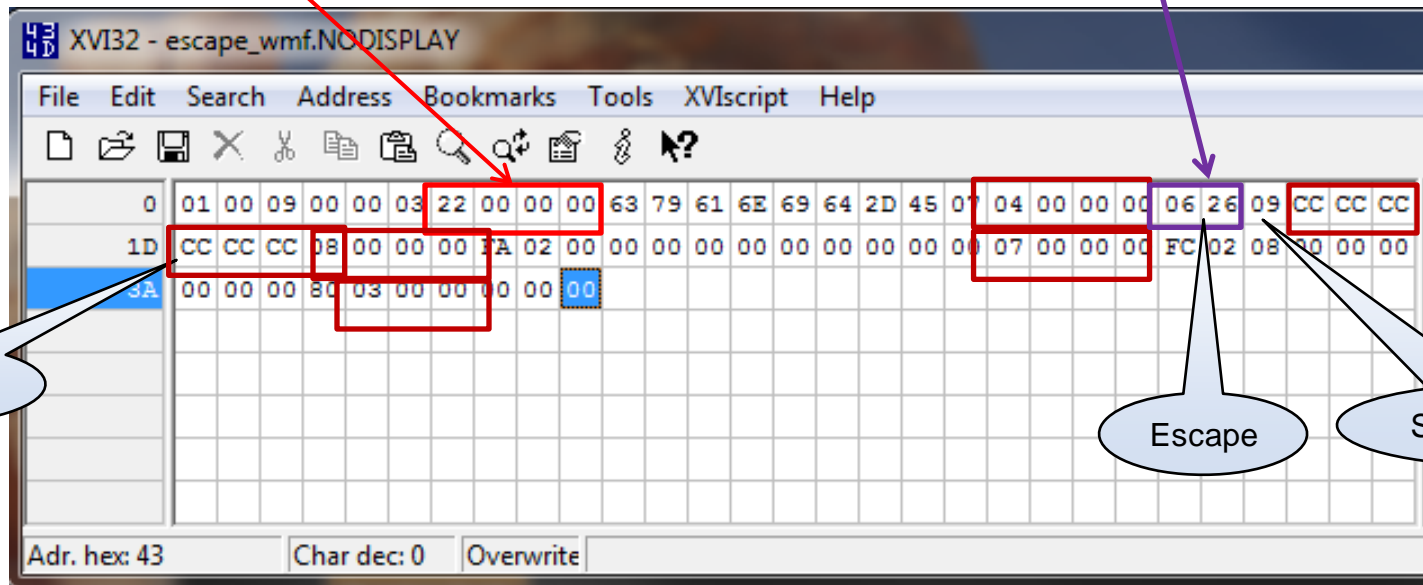


WMF File: The Fields & The Vulnerability

- Key Structures:

```
typedef struct _WindowsMetaHeader
{
    WORD FileType; /* Type of metafile (0=memory, 1=disk) */
    WORD HeaderSize; /* Size of header in WORDS (always 9) */
    WORD Version; /* Version of Microsoft Windows used */
    DWORD FileSize; /* Total size of the metafile in WORDs */
    WORD NumOfObjects; /* Number of objects in the file */
    DWORD MaxRecordSize; /* The size of largest record in WORDs */
    WORD NumOfParams; /* Not Used (always 0) */
} WMFHEAD;
```

```
typedef struct _StandardMetaRecord
{
    DWORD Size;
    /* Total size of the record in WORDs */
    WORD Function;
    /* Function number (defined in WINDOWS.H) */
    WORD Parameters[];
    /* Parameter values passed to function */
} WMFRECORD;
```



WMF Slicing (1)

An Instruction Slice Traced Back from Crash Site to Input

Each node uniquely trace back to one execution event through its sequence number

0x77f330a3 call eax 2 ffd0 0x0 0x3812f Reg(EAX=0xa8b94 ESP=0xb4fb88 EIP=0x77f330a3) W 4 b4fb88

0x77c472e3 rep movsd 2 f3a5 0x0 0xb142 Reg(EDI=0xa8804 eflags=0x10216 ESI=0xa9f8c ECX=0xa) R 4 a9f8c cc_cc_cc_cc W 4 a8804

0x77f2e997 mov ecx, [ebp+arg_8] 3 8b4d10 0x0 0xc5c3 Reg(EBP=0xb4fbf8 ECX=0x7c809a20) R 4 b4fc08 44_0_0_0

0x77f2e983 mov [ebp+arg_8], eax 3 894510 0x0 0xbd8c Reg(EAX=0x44 EBP=0xb4fbf8) W 4 b4fc08

0x77f2e97f add eax, eax 2 03c0 0x0 0xbd89 Reg(EAX=0x22 eflags=0x246)

0x77f2e949 mov eax, [edi+6] 3 8b4706 0x0 0xbd7d Reg(EAX=0xa8920 EDI=0xa87e8) R 4 a87ee 22_0_0_0

0x77c472e3 rep movsd 2 f3a5 0x0 0xb13c Reg(EDI=0xa87ec eflags=0x10216 ESI=0xa9f74 ECX=0x10) R 4 a9f74 0_3_22_0 W 4 a87ec

WMF Slicing (2)

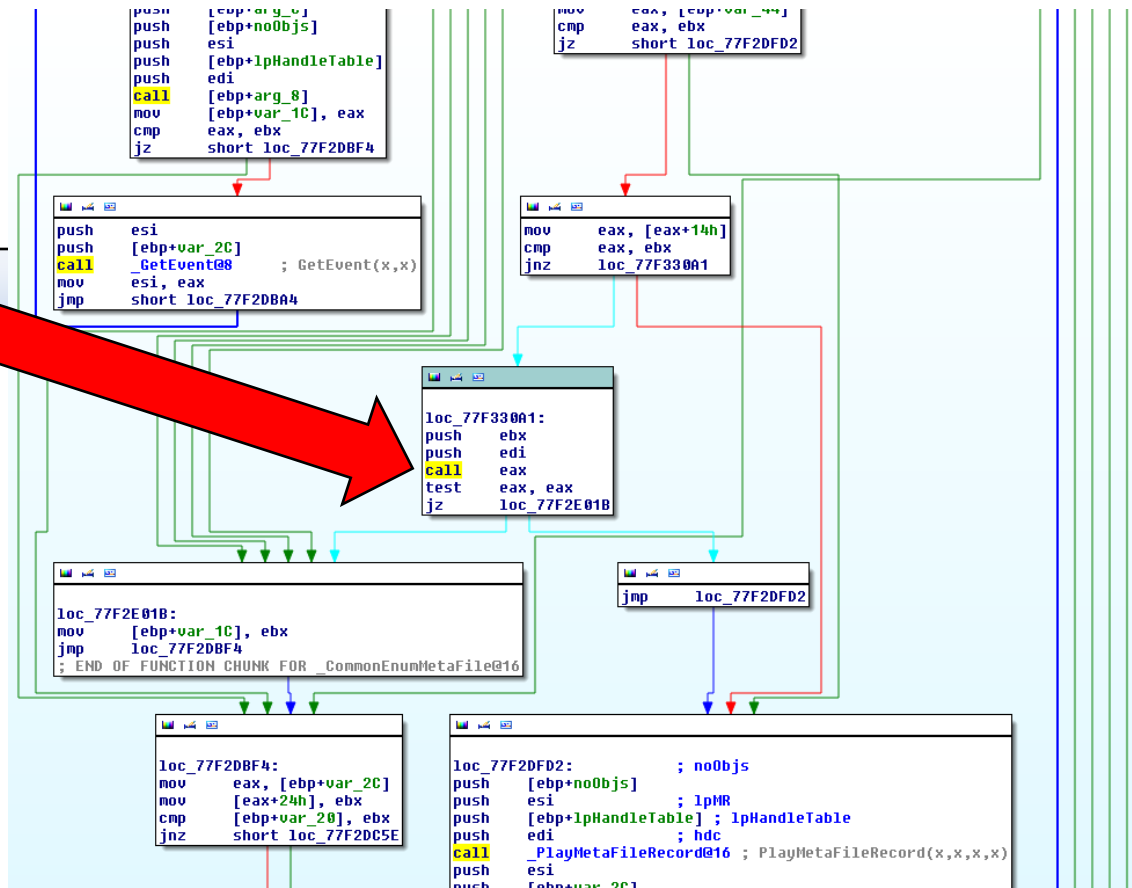
An Instruction Slice with Text Helps

Put Instruction In Its Context Helps
More

Module: **gdi32.dll**

Function: **CommonEnumMetaFile**

text:77F330A3 call eax

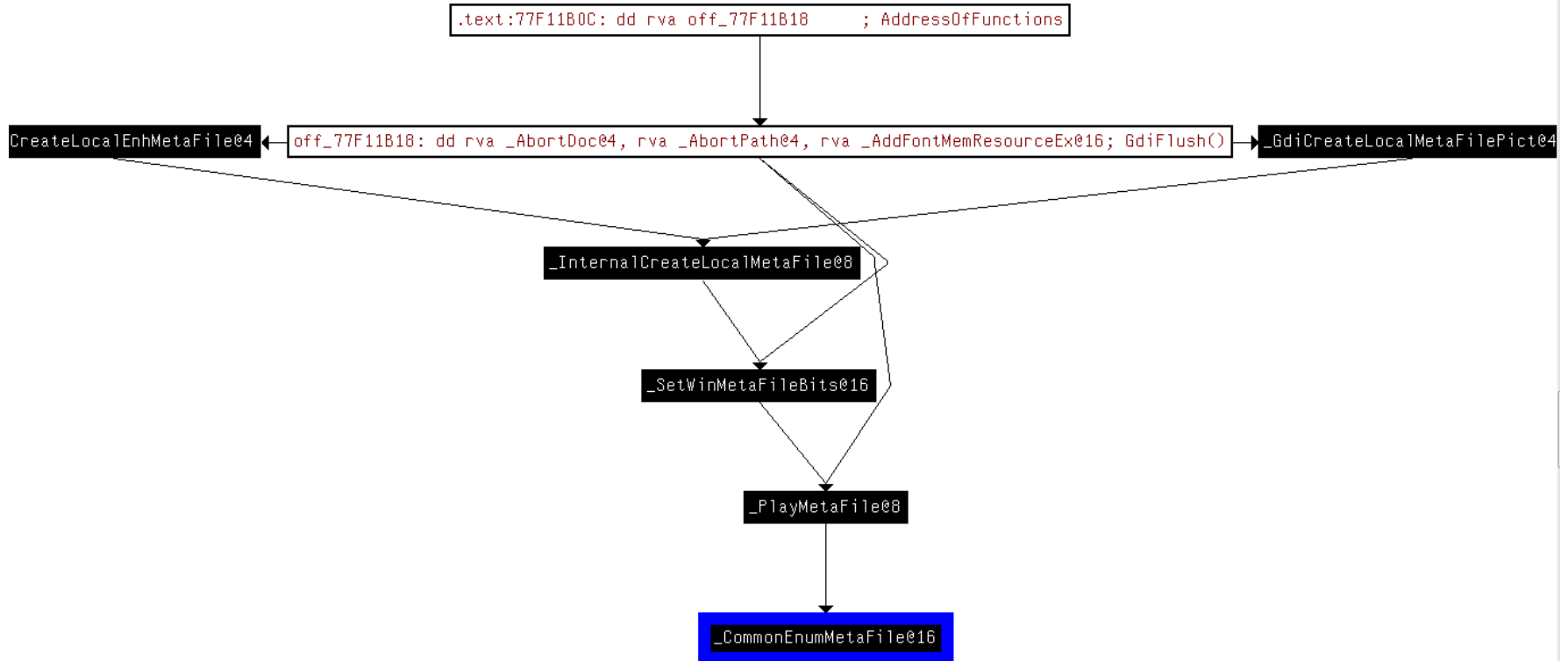


WMF Slicing (3)

An Instruction Slice with Text Helps a Little

text:77F330A3 call eax

More Context Helps More
Module: gdi32.dll
Function: CommonEnumMetaFile
Call Graph: caller PlayMetaFile



WMF -- The Relevant Parts

The WMF SETABORTPROC Vulnerability

```
rundll32.exe c:\windows\system32\shimgvw.dll,ImageView_Fullscreen  
C:\escape\escape.wmf
```

Dynamic Facts:

Out of 229,679
instructions
executed just to
cause the crash

**ONLY 12 Unique
Instructions Are
Relevant to the
CRASH**

Run a DLL as an App

Run a DLL as an App has encountered a problem and needs to close. We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

Please tell Microsoft about this problem.

We have created an error report that you can send to help us improve Run a DLL as an App. We will treat this report as confidential and anonymous.

To see what data this error report contains, [click here](#).

Conclusions

- Our tools support *interactive binary analysis*, with *Replay*, *Dynamic Taint Analysis*, and *Symbolic Execution*.
- **TREE** runs on top of IDA Pro and supports cross-platform trace collection, taint analysis and replay.
- **CBASS** (based on REIL) enables IR-based architecture-independent symbolic execution and can support both automated and interactive analysis.
- **YOU drive the tools!**

Where You Can Get TREE

- TREE is open source at:
<http://code.google.com/p/tree-cbass/>
 - First version of TREE (Taint Analysis) is released
 - Replay is in Progress
 - CBASS is Following
- Contacts:
 - Li.L.Lixin@gmail.com, Project Lead
 - xingzli@gmail.com, Developer
 - locvnguy@gmail.com, Developer
 - james.just@gmail.com, Program Manager

Acknowledgements

- Thanks to Ilfak Guilfanov and the IDA team for promptly fixing the bugs that we have reported to them and for their suggestions on the GUI integration.
- Thanks to Thomas Dullien and Tim Kornau of the Google Zynamics team for making their latest version of REIL available to us.
- Thanks to numerous reviewers at Battelle Memorial Institute for their feedback

References

- [1] L. Li and C. Wang. , Dynamic analysis and debugging of binary code for security applications, (to appear) *International Conference on Runtime Verification (RV'13)*. Rennes, France. 2013
- [2] Godefroid, P., Levin, M.Y., Molnar, D.A.: Automated whitebox fuzz testing. In: Network And Distributed System Security Symposium(2008)
- [3] Song, Dawn, et al. "BitBlaze: A new approach to computer security via binary analysis." *Information systems security*. Springer Berlin Heidelberg, 2008. 1-25.
- [4] Clause, James, Wanchun Li, and Alessandro Orso. "Dytan: a generic dynamic taint analysis framework." *Proceedings of the 2007 international symposium on Software testing and analysis*. ACM, 2007.
- [5] Schwartz, Edward J., Thanassis Avgerinos, and David Brumley. "All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)." In *Security and Privacy (SP), 2010 IEEE Symposium on*, pp. 317-331. IEEE, 2010.
- [6] Bhansali, Sanjay, et al. "Framework for instruction-level tracing and analysis of program executions." *Proceedings of the 2nd international conference on Virtual execution environments*. ACM, 2006.
- [7] Dullien, T., Porst, S.: REIL: A platform-independent intermediate representation of disassembled code for static code analysis. In:CanSecWest(2009)
- [8] REIL:URL:[http://www.zynamics.com/binnavi/manual/html/reil language.htm](http://www.zynamics.com/binnavi/manual/html/reil%20language.htm)