FuzzUEr: Enabling Fuzzing of UEFI Interfaces on EDK-2

Connor Glosner, Aravind Machiry

Purdue Systems and Software Security (PurS3) Lab

PURDUE UNIVERSITY | Elmore Family School of Electrical and Computer Engineering

Motivation: LogoFail Example

- LogoFail allows for an arbitrary boot logo to be loaded from the EFI partition that could lead to arbitrary code execution
- Resulting in 24 memory corruption bugs spanning 11 vendors

BIOS Image Parsing Function Vulnerabilities (LogoFAIL)

Lenovo Security Advisory: LEN-145284

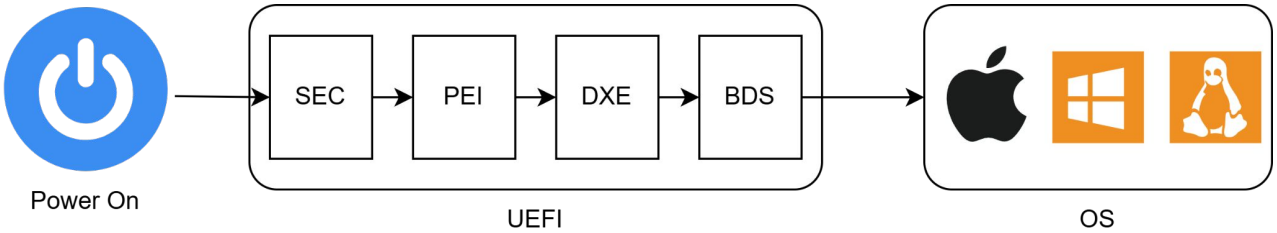
Potential Impact: Denial of Service, Privilege Escalation

Severity: High

Scope of Impact: Industry-wide

CVE Identifier: CVE-2023-5058, CVE-2023-39538, CVE-2023-39539, CVE-2023-40238

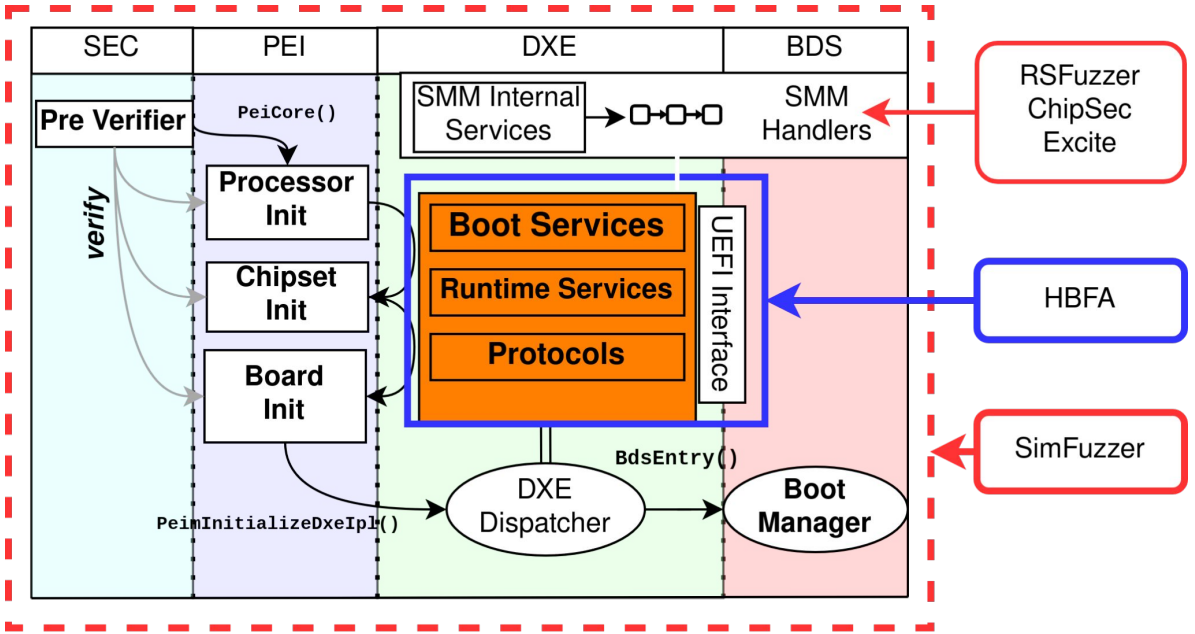
What is UEFI?



UEFI (Unified Extensible Firmware Interface) firmware is a modularized firmware on modern machines to aid the operating systems communication with the hardware. It provides the following improvements when compared to Legacy BIOS:

- Graphical User Interface
- Support for large drives (>2.2TB)
- Secure Boot
- Modular device driver support
- Processor Independence
- Networking and remote access (during boot)

Detailed UEFI View



Challenges: Type Identification

```
EFI_PXE_BASE_CODE_PROTOCOL *PxeBoot;
Status = gBS->LocateProtocol (&gEfiPxeBaseCodeProtocolGuid,
                              NULL,
                              (VOID **)&PxeBoot
                              );
EFI_MTFTP6_PROTOCOL *Mtftp6Prot;
EFI_PXE_BASE_CODE_PACKET Packet;
// Generate Packet Data (Generator Function)
Mtftp6Prot->GetInfo(..., (VOID **)&Packet);
// Set the packet (Call-Site)
PxeBoot->SetPackets(...,&Packet);
```

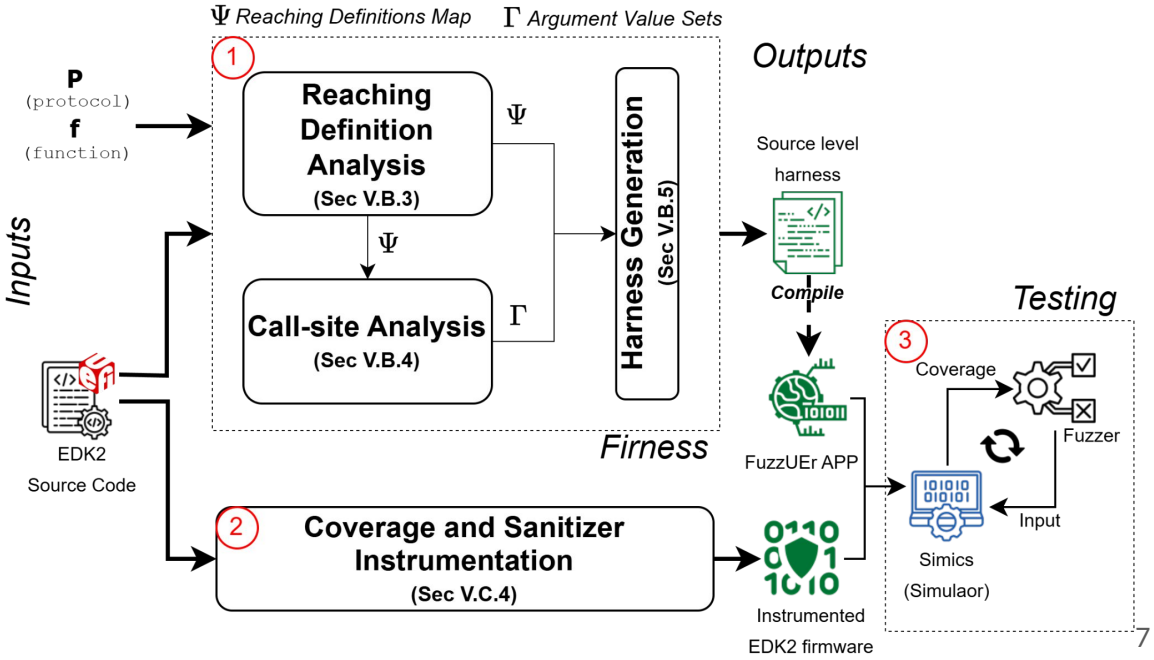
Takes a different type

Challenges: Generating State-Dependent Data

```
EFI_PXE_BASE_CODE_PROTOCOL *PxeBoot;
Status = gBS->LocateProtocol (&gEfiPxeBaseCodeProtocolGuid,
                              NULL,
                              (VOID **)&PxeBoot
                              );
EFI_MTFTP6_PROTOCOL *Mtftp6Prot;
EFI_PXE_BASE_CODE_PACKET Packet;
// Generate Packet Data (Generator Function)
Mtftp6Prot->GetInfo(..., (VOID **)&Packet);
// Set the packet (Call-Site)
PxeBoot->SetPackets(...,&Packet);
```

Our Solution: FuzzUEr

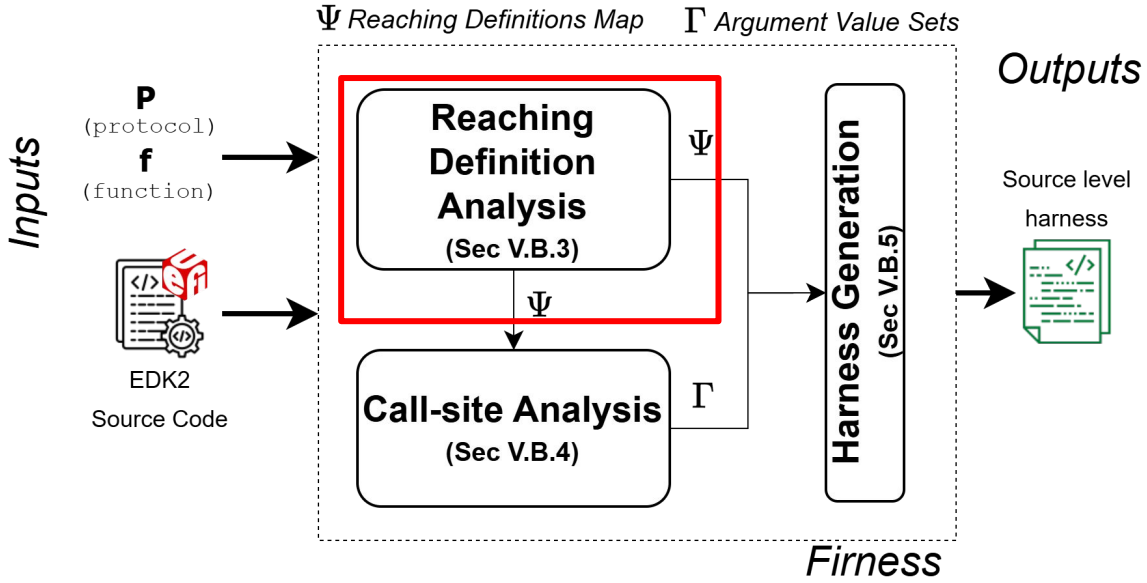
1) Firness: Static Analysis assisted harness generation
2) Sanitizer Instrumentation: ASan
3) Fuzz Testing: TSFFS



Firness: Static Analysis Assisted Harness Generation

1) **Reaching Definition Analysis**:

- Collect all definitions for L-value expressions:
  - Constants
  - Generator function
- Generator functions: A function that assigns a value for L-value expression of interest



Reaching Definition Analysis: Example

INPUT:

```
...
EFI_MTFTP6_PROTOCOL *Mtftp6Prot;
EFI_PXE_BASE_CODE_PACKET Packet;
// Generate Packet Data (Generator
Function)
Mtftp6Prot->GetInfo(..., (VOID
**)&Packet);
// Set the packet (Call-Site)
PxeBoot->SetPackets(...,&Packet);
```

OUTPUT:

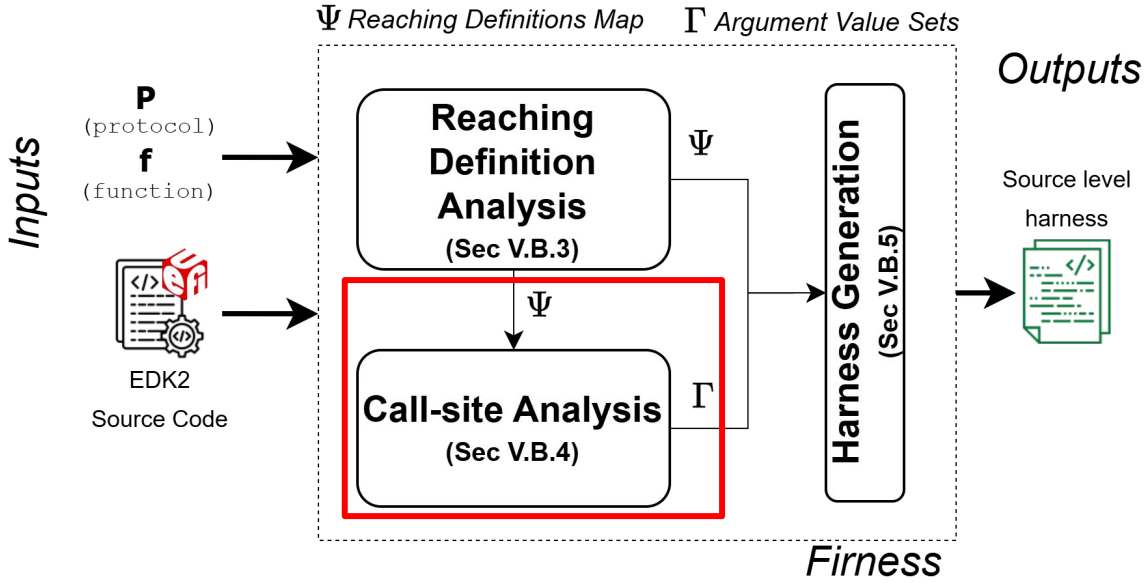
```
Packet: [
         {
          "assign": PxeBoot->SetPackets
          "direction": OUT
         },
         {
          "assign": Mtftp6Prot->GetInfo
          "direction": IN
         }
        ]
```

Reaching Definitions Map

Firness: Static Analysis Assisted Harness Generation

2) **Call-site Analysis**:

- Collect argument value sets for each call-site:
  - Function of interest
  - Generator function



Call-Site Analysis: Example

INPUT:

```
...
EFI_MTFTP6_PROTOCOL *Mtftp6Prot;
EFI_PXE_BASE_CODE_PACKET Packet;
// Generate Packet Data (Generator
Function)
Mtftp6Prot->GetInfo(..., (VOID
**)&Packet);
// Set the packet (Call-Site)
PxeBoot->SetPackets(...,&Packet);
```

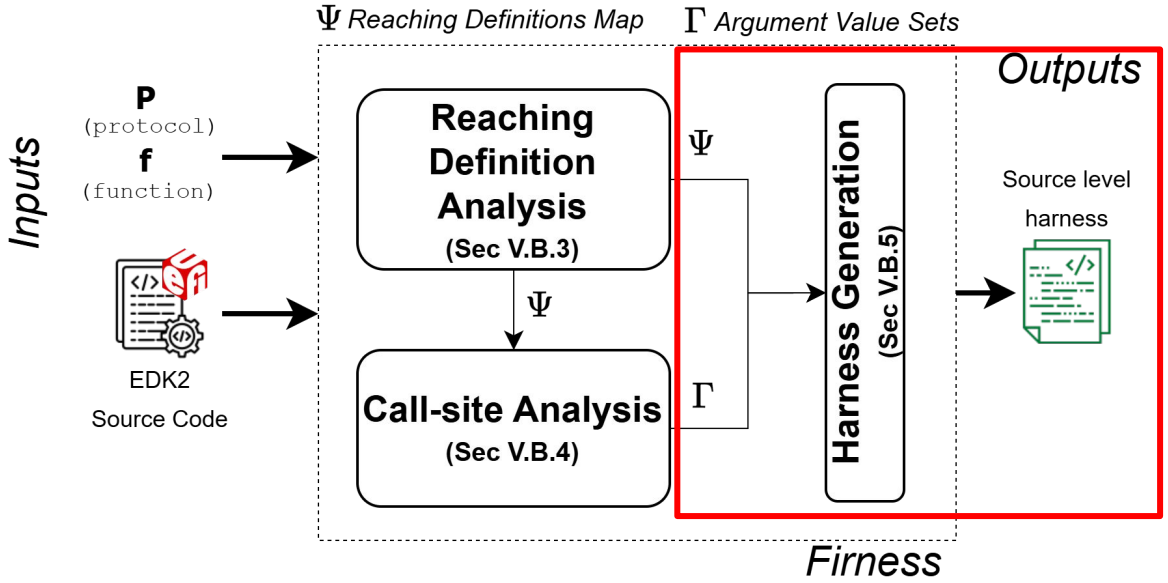
OUTPUT:


```
"arguments": {
  "Arg_0": [{
   "Arg Dir": "IN",
   "Arg Type": "EFI_PXE_BASE_CODE_PACKET",
   "Assignment": "Mtftp6Prot->GetInfo",
   "Data Type": "EFI_PXE_BASE_CODE_PACKET",
   "Usage": "&Packet",
   "Pointer Count": 1,
   "Potential Values": [],
   "Variable": "__PROTOCOL__"
}]},
"service": "protocol",
"function": "SetPackets",
"includes": [],
"return_type": "EFI_STATUS"
```

Firness: Static Analysis Assisted Harness Generation

3) **Harness Generation**:

- Argument Type Identification:
  - Determines the correct type for generic types (void*)
- Argument Value Identification:
  - Determines the set of all possible constants
- Generator Function:
  - Determines the set of all generator functions for each valid type

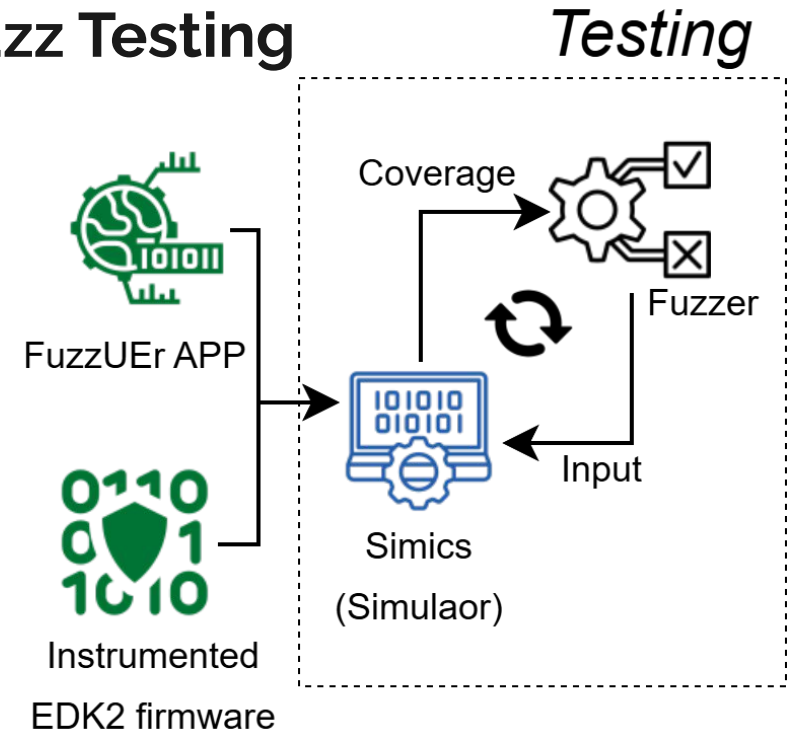


Harness Generation: Example

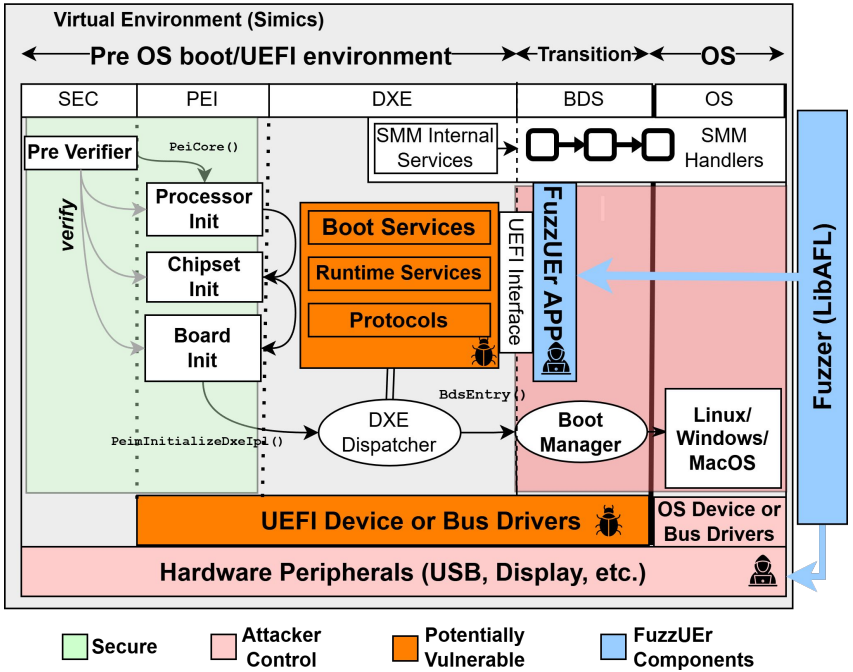
```
// Fuzzable Variable Initialization
ReadBytes(Input, sizeof(SetPackets_Arg_1), (VOID *)SetPackets_Arg_1);
// Generator Struct Variable Initialization
ReadBytes(Input, sizeof(EFI_MTFTP6_PROTOCOL_GetInfo_Arg_5->OptionStr),
(VOID *)(EFI_MTFTP6_PROTOCOL_GetInfo_Arg_5->OptionStr));
ReadBytes(Input, sizeof(EFI_MTFTP6_PROTOCOL_GetInfo_Arg_5->ValueStr),
(VOID *)(EFI_MTFTP6_PROTOCOL_GetInfo_Arg_5->ValueStr));
Status = ProtocolVariable2->GetInfo(EFI_MTFTP6_PROTOCOL_GetInfo_Arg_5,
                                    EFI_MTFTP6_PROTOCOL_GetInfo_Arg_6,
                                    (EFI_MTFTP6_PACKET **)&SetPackets_Arg_12);
Status = ProtocolVariable->SetPackets(ProtocolVariable,
                                      SetPackets_Arg_1,
                                      ...
                                      SetPackets_Arg_12);
```

Sanitizer Instrumentation + Fuzz Testing

- Added support for ASan within EDK-2 through additional integrated runtime libraries
- We utilize an existing open-source Simics fuzzer called Target Software Fuzzer for Simics (TSFFS)



System View: FuzzUEr



Research Questions

Q1: How effective is Firness at identifying necessary information to generated harnesses?

Q2: How effective is FuzzUEr in fuzzing EDK-2 specific UEFI protocols? Bug Finding Ability? Code Coverage?

Q3: How does FuzzUEr perform in a best-effort comparison to HBFA?

Q4: What is the contribution of each out techniques on the overall effectiveness of FuzzUEr?

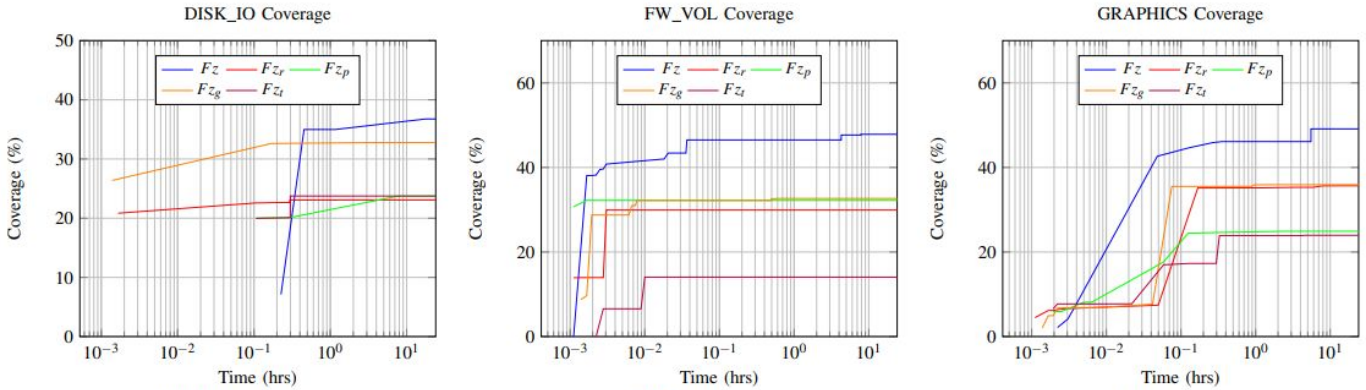
Evaluation: Bug Finding Ability (Q2)

	System Configuration				
	Fz_r (RSFuzzer)	Fz_g	Fz_t	Fz_p (FuzzGen)	Fz (FuzzUEr)
Previously Known Bugs	0%	0%	66%	66%	66%
New Bugs	55%	85%	90%	55%	100%

Without points-to information not all of the function pointers are able to be identified

Evaluation: Code Coverage (Q2)

Fuzzed for 24hrs



Discovering complex data types improves coverage

Evaluation: HBFA Comparison (Q3)

Generates larger harnessed and achieves greater coverage

Protocol	USB2_HC		DISK_IO		PCI_ROOT	
Tool	H	*Fz*	H	*Fz*	H	*Fz*
Harness LoC	63	**1,391**	**597**	319	312	**1,098**
Code Coverage (Number of Unique Edges)						
Total Coverage	319	**6,091 (↑19x)**	1,413	**8,797 (↑6x)**	762	**6,514 (↑8x)**
Driver Coverage	138	**2,041 (↑14x)**	595	**5,205 (↑8x)**	117	**3,690 (↑31x)**
Number of Unique Bugs Found						
Bugs Discovered	0	**2 (↑200%)**	0	**1 (↑100%)**	0	0

- HBFA Harnesses are simple
- They cover the same functions and are unable to find the bugs

Conclusion

- We proposed FuzzUEr a coverage-guided fuzzing framework for UEFI interface functions designed around EDK-2
- Firness is a combination of static analysis and templated harness generation to craft source-level harness
- We demonstrated FuzzUErs effectiveness in a best-effort comparative evaluation with HBFA
- We discovered 20 new vulnerabilities in EDK-2
- Available as open-source at: https://github.com/BreakingBoot/FuzzUEr

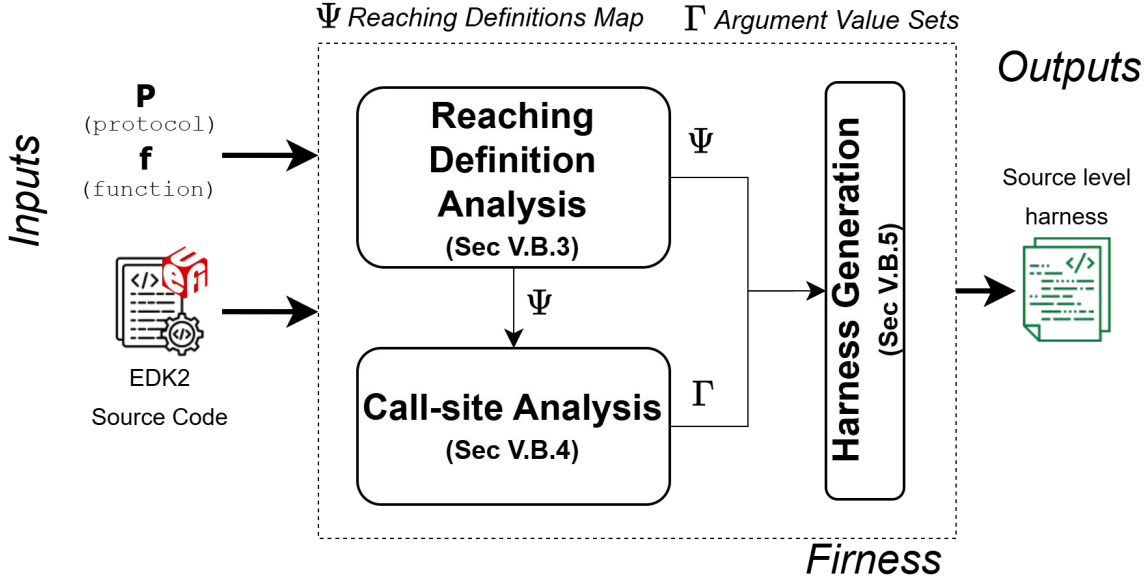
cglosne@purdue.edu

Backup

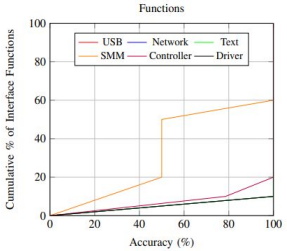
Firness: Static Analysis Assisted Harness Generation

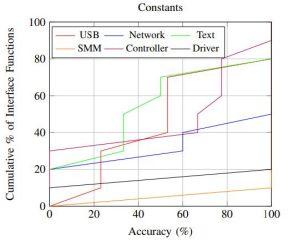
Pre-processing: Collect the following information:

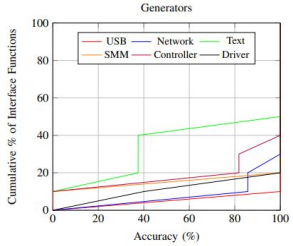
- Function definitions
- Function pointer aliases
- Parameter directions (IN, OUT)

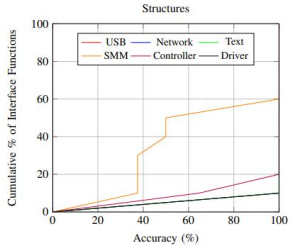


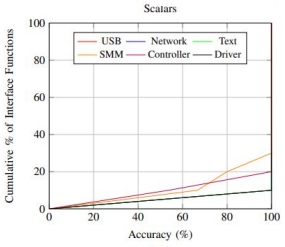
Evaluation: Type Identification (Q1)

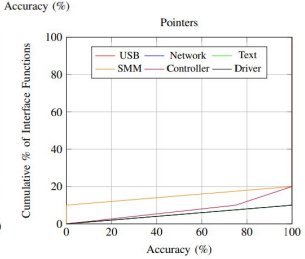


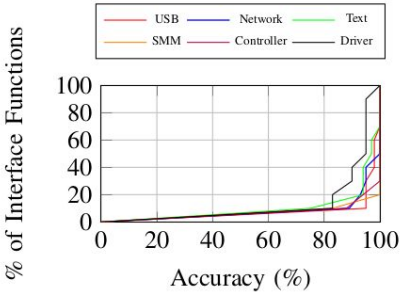












Preprocessing	Reaching Definition Analysis	Call-Site Analysis	Harness Generation

Existing UEFI Tools

Used in our evaluation

Tool	Target Component	Open Source?
RSFuzzer	SMI Callouts	No
Excite	SMI Callouts	No
HBFA	Interface Models	Yes
ChipSec	SMI Callouts	Yes
SimFuzzer	N/A	No

Motivation: UEFI CVE Analysis

DXE Interfaces		Vulnerabilities		
Type	**Percentage Contribution**	**Memory Corruption**	**Others**	**Total (% of Cummu)**
Services (Boot and Runtime)	30%	3 (7%)	39 (97%)	41 (29%)
Protocols	70%	48 (49%)	50 (51%)	98 (71%)
Cumulative (Cummu)				139