ISP RAS

Improving Hybrid Fuzzing Using Debug Information

Theodor Arsenij Larionov-Trichkine

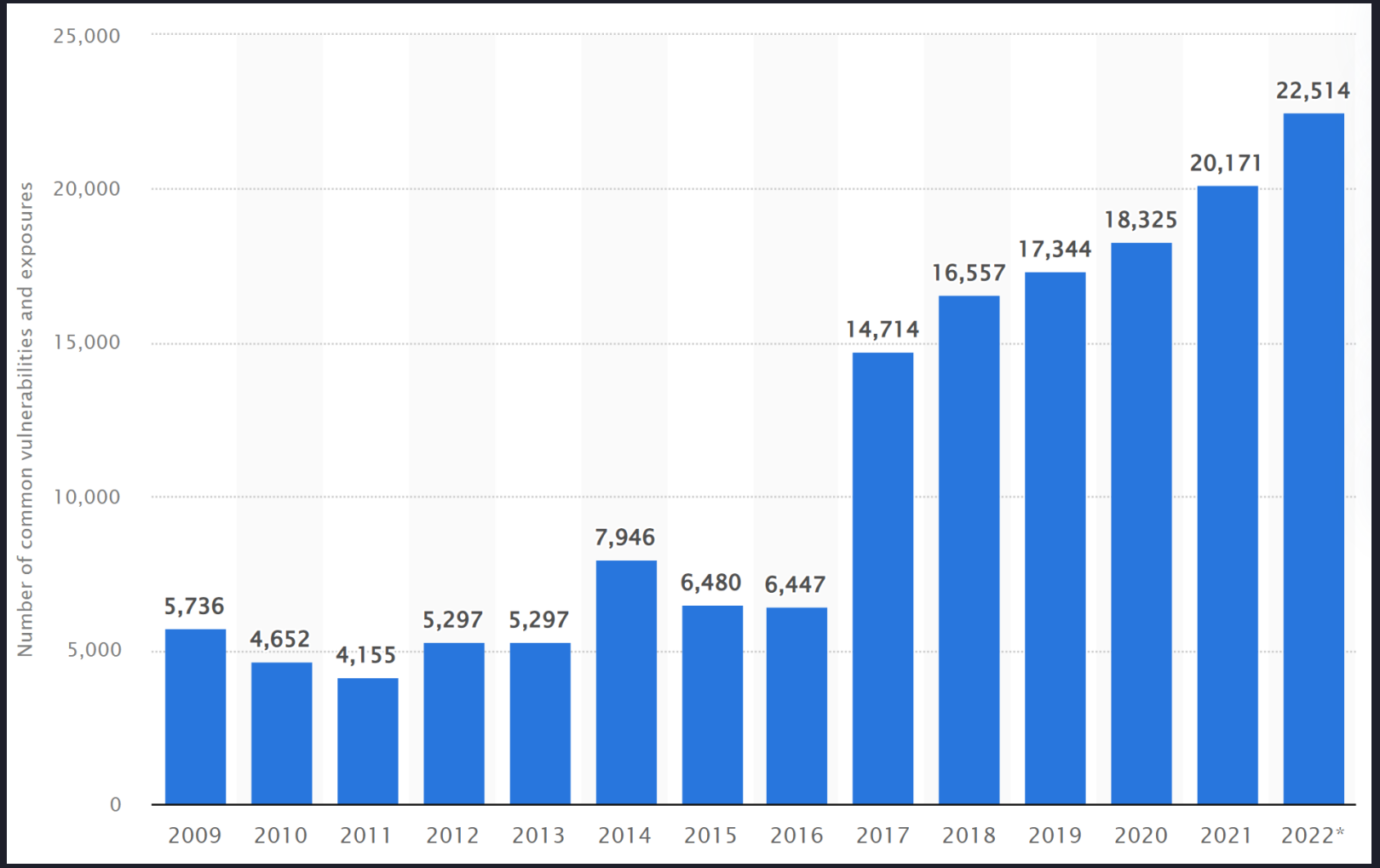
HSE University, MIEM, IS, 2023

Scientific Advisors: Kuts Daniil, Petrenko Alexander

Plan

1. Software Vulnerabilities
2. Dynamic Analysis – Hybrid Fuzzing
3. The Problem – Symbolic Execution Limitations
4. Proposed solution
5. Results
6. Conclusion

Software Vulnerabilities



Software Vulnerabilities

Hybrid Fuzzing

```
void vuln(int key) {
    if (key * 0x142a2d == 0xdeadbeef) {
        error();
    }
}
```

Hybrid Fuzzing

```
void vuln(int SYM_VAR) {
    if (SYM_VAR * 0x142a2d == 0xdeadbeef) {
        error();
    }
}
```

```
SYM_VAR = 0xdeadbeef / 0x142a2d
```

```
SYM_VAR = 0xb0b
```

Hybrid Fuzzing

```
vuln(0xb0b)
```

The Problem

```
1. uint16_t crc_ibm_table[256] = {
2.     0x0000, 0xc0c1, 0xc181, 0x0140, ...
3. };
4.
5. uint16_t crc_ibm_byte(uint16_t crc, const uint8_t c)
6. {
7.     uint8_t sym_idx = (crc ^ c) & 0xFF;
8.     return crc_ibm_table[sym_idx] ^ (crc >> 8);
9. }
10.
11. if (crc_ibm_byte(0, buf[0]) == 0x1337) {
12.     error();
13. }
```

The Problem

```
1. uint16_t crc_ibm_table[256] = {
2.     0x0000, 0xc0c1, 0xc181, 0x0140, ...
3. };
4.
5. uint16_t crc_ibm_byte(uint16_t crc, const uint8_t c)
6. {
7.     uint8_t sym_idx = (crc ^ c) & 0xFF;
8.     return crc_ibm_table[sym_idx] ^ (crc >> 8);
9. }
10.
11. if (crc_ibm_byte(0, buf[0]) == 0x1337) {
12.     error();
13. }
```

The Problem

```
1. uint16_t crc_ibm_table[256] = {
2.     0x0000, 0xc0c1, 0xc181, 0x0140, ...
3. };
4.
5. uint16_t crc_ibm_byte(uint16_t crc, const uint8_t c)
6. {
7.     uint8_t sym_idx = (crc ^ c) & 0xFF;
8.     return crc_ibm_table[sym_idx] ^ (crc >> 8);
9. }
10.
11. if (crc_ibm_byte(0, buf[0]) == 0x1337) {
12.     error();
13. }
```

The Problem. Existing Methods.

- A Survey of Symbolic Execution Techniques
- Sydr
- Mayhem
- …

Proposed Solution

Let’s use debug information!

Proposed Solution

```
1. uint16_t crc_ibm_table[256] = {
2.     0x0000, 0xc0c1, 0xc181,
3.     0x0140, ...
4. };
5.
6. crc_ibm_table[sym_idx];
```

Proposed Solution

```
1. uint16_t crc_ibm_table[256] = {
2.     0x0000, 0xc0c1, 0xc181,
3.     0x0140, ...
4. };
5.
6. crc_ibm_table[sym_idx];
```

Proposed Solution

```
crc_ibm_table[sym_idx];
```

```
Solve(sym_idx > len(crc_ibm_table)?)
```

Results

1. The method has already been implemented as a part of the Sydr tool, and a number of preliminary experiments are in progress
2. The proposed method makes it possible to find new classes of errors (local/global out-of-bounds accesses)

Thanks for listening

Questions?

References

1. “Chromium project memory safety report,” https://www.chromium.org/Home/chromium-security/memory-safety/, accessed: 2023-02-12.
2. A. V. Vishnyakov, A. Fedotov, D. O. Kuts, A. A. Novikov, D. Parygina, E. Kobrin, V. Logunova, P. Belecky, and S. F. Kurmangaleev, “Sydr: Cutting edge dynamic symbolic execution,” CoRR, vol. abs/2011.09269, 2020. [Online]. Available: https://arxiv.org/abs/2011.09269
3. D. O. Kuts, “Towards symbolic pointers reasoning in dynamic symbolic execution,” CoRR, vol. abs/2109.03698, 2021. [Online]. Available: https://arxiv.org/abs/2109.03698
4. R. Baldoni, E. Coppa, D. C. D’elia, C. Demetrescu, and I. Finocchi, “A survey of symbolic execution techniques,” ACM Comput. Surv., vol. 51, no. 3, may 2018. [Online]. Available: https://doi.org/10.1145/3182657
5. S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley, “Unleashing mayhem on binary code,” in IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA. IEEE Computer Society, 2012, pp. 380–394. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/SP.2012.31